

The Minimum You Need to Know

About Java and xBaseJ

Roland Hughes

Logikal Solutions

Copyright © 2010 by Roland Hughes
All rights reserved
Printed and bound in the United States of America

ISBN-13 978-0-9823580-3-0

This book was published by Logikal Solutions for the author. Neither Logikal Solutions nor the author shall be held responsible for any damage, claim, or expense incurred by a user of this book and the contents presented within or provided for download at <http://www.theminimumyouneedtoknow.com>.

These trademarks belong to the following companies:

Borland	Borland Software Corporation
Btrieve	Btrieve Technologies, Inc.
C-Index/II	Trio Systems LLC
Clipper	Computer Associates, Inc.
CodeBase Software	Sequiter Inc.
CodeBase++	Sequiter Inc.
CommLib	Greenleaf Software
Cygwin	Red Hat, Inc.
DataBoss	Kedwell Software
DataWindows	Greenleaf Software
dBASE	dataBased Intelligence, Inc.
DEC	Digital Equipment Corporation
DEC BASIC	Hewlett Packard Corporation
DEC COBOL	Hewlett Packard Corporation
Foxbase	Fox Software
FoxPro	Microsoft Corporation
FreeDOS	Jim Hall
GDB	Greenleaf Software
HP	Hewlett Packard Corporation
IBM	International Business Machines, Inc.
Java	Sun Microsystems, Inc.
Kubuntu	Canonical Ltd.
Linux	Linus Torvals
Lotus Symphony	International Business Machines, Inc.
MAC	Apple Inc.
MappppQuest	MapQuest, Inc.
MySQL	MySQL AB
Netware	Novell, Inc.
OpenVMS	Hewlett Packard Corporation
OpenOffice	Sun Microsystems, Inc.
openSuSE	Novell, Inc.
ORACLE	Oracle Corporation
OS/2	International Business Machines, Inc.
Paradox	Corel Corporation
Pro-C	Pro-C Corp.
Quicken	Intuit Inc.
RMS	Hewlett Packard Corporation
RDB	Oracle Corporation
SourceForge	SourceForge, Inc.
Ubuntu	Canonical Ltd.

Unix	Open Group
Visual Basic	Microsoft Corporation
Watcom	Sybase
Windows	Microsoft Corporation
Zinc Application Framework	Professional Software Associates, Inc.

All other trademarks inadvertently missing from this list are trademarks of their respective owners. A best effort was made to appropriately capitalize all trademarks which were known at the time of this writing. Neither the publisher nor the author can attest to the accuracy of any such information contained herein. Use of a term in this book should not be regarded as affecting the validity of any trademark or service mark.

Additional Books by Roland Hughes

You can always find the latest information about this book series by visiting <http://www.theminimumyouneedtoknow.com>. Information regarding upcoming and out-of-print books may be found by visiting <http://www.logikalsolutions.com> and clicking the “upcoming and out of print books” link. At the time of this writing, Logikal Solutions and Roland Hughes offer the following books either in print or as eBooks.

The Minimum You Need to Know About Logic to Work in IT

ISBN-13 978-0-9770866-2-7

Pages: 154

Covers logic, flowcharting, and pseudocode. If you only learned OOP, you really need to read this book first.

The Minimum You Need to Know To Be an OpenVMS Application Developer

ISBN-13 978-0-9770866-0-3

Pages: 795

Includes CD-ROM

Covers DCL, logicals, symbols, command procedures, BASIC, COBOL, FORTRAN, C/C++, Mysql, RDB, MMS, FMS, RMS indexed files, CMS, VMSPHone, VMSMAIL, LSE, TPU, EDT and many other topics. This book was handed out by HP at a technical boot camp because the OpenVMS engineering team thought so highly of it.

The Minimum You Need to Know About Java on OpenVMS, Volume 1

ISBN-13 978-0-9770866-1-0

Pages: 352

Includes CD-ROM

Covers using Java with FMS and RMS indexed files. There is a lot of JNI coding. We also cover calling OpenVMS library routines, building with MMS and storing source in CMS.

The Minimum You Need to Know About Service Oriented Architecture

ISBN-13 978-0-9770866-6-5

Pages: 370

The National Best Books 2008 Award Winner – Business: Technology/Computer

Covers accessing your MySQL, RDB, and RMS indexed file data silos via Java and port services from a Linux or other PC front end. Also covers design and development of ACMS back end systems for guaranteed execution applications.

Infinite Exposure

ISBN-13 978-0-9770866-9-6

Pages: 471

A novel about how the offshoring of IT jobs and data centers will lead to the largest terrorist attack the free world has ever seen and ultimately to nuclear war.

There are a number of reviews of this book available on-line. The first 18 chapters are also being given away for free at BookHabit, ShortCovers, Sony' s eBook store, and many other places. If you can' t decide you like it after the first 18 chapters, Roland really doesn' t want to do business with you.

Source Code License

This book is being offered to the public freely, as is the source code. Please leave comments about the source of origin in place when incorporating any portion of the code into your own projects or products.

Users of the source code contained within this book agree to hold harmless both the author and the publisher for any errors, omissions, losses, or other financial consequences which result from the use of said code. This software is provided “as is” with no warranty of any kind expressed or implied.

Visit <http://www.theminimumyouneedtoknow.com> to find a download link if you don't want to retype or cut and paste code from this book into your own text editor.

Table of Contents

Introduction.....	11
Why This Book?.....	11
Why xBaseJ?.....	13
A Brief History of xBASE.....	15
What is xBASE?.....	17
Limits, Restrictions, and Gotchas.....	24
Summary.....	28
Review Questions.....	28
Chapter 1	29
1.1 Our Environment.....	29
env1.....	29
1.2 Open or Create?.....	30
1.3 Example 1.....	32
example1.java.....	32
1.4 Exception Handling and Example 1.....	37
1.5 rollie1.java.....	39
rollie1.java.....	39
.....	48
testRollie1.java.....	49
1.6 Programming Assignment 1.....	49
1.7 Size Matters.....	49
example5.java.....	49
1.8 Programming Assignment 2.....	51
1.9 Examining a DBF.....	51
showMe.java.....	52
1.10 Programming Assignment 3.....	61
1.11 Descending Indexes and Index Lifespan.....	62
doeHistory.java.....	63
testDoeHistory.java.....	76
1.12 Programming Assignment 4.....	82
1.13 Deleting and Packing.....	83
testpackDoeHistory.java.....	84
1.14 Programming Assignment 5.....	88
1.15 Data Integrity.....	88
1.16 Programming Assignment 6.....	89
1.17 Summary.....	90

1.18 Review Questions.....	91
Chapter 2	93
2.1 Why This Example?.....	93
2.2 Supporting Classes.....	102
MegaDBF.java.....	102
StatElms.java.....	106
StatDBF.java.....	107
MegaXDueElms.java.....	112
.....	113
DueSortCompare.java.....	114
2.3 The Panels.....	115
MegaXbaseDuePanel.java.....	115
MegaXbaseBrowsePanel.java.....	124
MegaXbaseEntryPanel.java.....	128
2.4 The Import Dialog.....	153
MegaXImport.java.....	153
.....	157
.....	157
.....	157
MegaXbase.java.....	157
testMegaXbase.java.....	163
2.5 Programming Assignment 1.....	164
2.6 Programming Assignment 2.....	165
2.7 Programming Assignment 3.....	165
2.8 Programming Assignment 4.....	165
2.9 Summary.....	165
2.10 Review Questions.....	167
Chapter 3	169
3.1 Authoritative Resources.....	169
3.2 Timestamps on Reports.....	172
3.3 Doomed to Failure and Too Stupid to Know.....	176
Appendix A.....	181
Answers to Introduction Review Questions:.....	181
Answers to Chapter 1 Review Questions.....	182
Answers to Chapter 2 Review Questions.....	185

Introduction

Why This Book?

I asked myself that same question every day while I was writing it. Why am I going to write a book much like my other books and give it away for free? The simple answer is that I had to do a lot of the research anyway. If I have to do that much research, then I should put out a book. Given the narrowness of the market and the propensity for people in that market to believe all software developers work for free, the book would physically sell about two copies if I had it printed. (Less than 1/10th of 1% of all Linux users actually pay for any software or technology book they use.)

What started me down this path was a simple thing. In order to make a Web site really work, a family member needed to be able to calculate the 100 mile trucking rate for the commodity being sold. The commercial Web site had a really cool feature where it would automatically sort all bids within 300 miles based upon the per-bushel profit once the transportation costs were taken out. The person already had a printed list of the trucking rates, so how difficult could it be?

Some questions should never be asked in life. “What could go wrong?” and “How difficult could it be?” are two which fall into that category. When you ask questions like that, you tend to get answers you were unprepared to hear.

In my early DOS days, I would have sat down and begun coding up a C program using Greenleaf DataWindows and the Greenleaf Database library. Of course, back then, we didn't have the Internet, so I would have had to use the Greenleaf CommLib to dial out to some BBS to get the DOE (Department of Energy) national average fuel price.

During later DOS days, but before Microsoft wrote a task-switching GUI that sat on top of DOS and that was started by typing WIN at the C:> prompt which they had the audacity to call “The Windows Operating System,” I would have reached for a C/C++ code generator like Pro-C from Vestronix (later Pro-C Corp.) or DataBoss from Kedwell Software. Neither program did communications, but both could be used to quickly lay out xBASE databases, generating entry/maintenance screens, menus, and reports in a matter of minutes. You could create an entire application that used just a few files for distribution, all of which could be copied into a single directory, and the user would be happy.

Once Windows came out, things got ugly. I did a lot of OS/2 work, even though not many companies or people used it. The problem with OS/2 was that IBM had Microsoft develop it and Microsoft was intent on ensuring that OS/2 would never be a threat to Windows. (Windows wasn't even an actual operating system until many years after OS/2 came out.) Once IBM had the bulk of the Microsoft code removed from it, OS/2 became an incredibly stable platform which managed memory well. IBM didn't manage it well, saddling developers with expensive device driver development tools that would only work with an increasingly hard-to-find version of Microsoft C.

Most of us did cross-platform development in those days. I used Watcom C/C++ for DOS, Windows, and OS/2 development (now OpenWatcom as the project is OpenSource). It was easy when you used the Zinc Application Framework for your GUI. There were a ton of cross-platform indexed file libraries. Greenleaf supported a lot of compilers and platforms with its Database library for xBASE files. Of course, there were a lot of shareware and commercial Btree type indexed file systems out there. These had the advantage of locking the user into your services. These had the disadvantage of locking the user into your services. They weren't widely supported by 'common tools' like spreadsheets and word processors. The one I remember using the most was C-Index/II from Trio Systems LLC. As I recall it was written generically enough that it actually worked on DOS, MAC, Windows, and OS/2. Of course, that was during the brief period in life when the Metrowerks CodeWarrior toolset supported the MAC.

In short, from the 80s through the end of the 90s we always had some way of creating a stand-alone application with its own indexed local data storage that didn't require lots of other things to be installed. When Windows started going down the path of 'heeding lots of other stuff' was when you started seeing companies selling software to do nothing other than develop installation programs for Windows.

As an application developer who is quite long in the tooth, I don't want to link with DLLs, shared libraries, installed images, or any other thing which is expected to be installed on the target platform. I have heard every justification for it known to man. I was there and listened to people slam my C program because their Visual Basic (VB) application took only 8K and 'looked slicker' than my application which consumed an entire floppy. I was also there to watch production come to a screeching halt when a new version of the VB run-time got installed to support some other 'mission critical app' only to find all previous apps were now incompatible. (The machine running my C program which took a whole floppy continued to keep the business it supported running while much screaming and finger-pointing was going on all around it.)

Why this book? Because the person downloading your SourceForge project or other free piece of software doesn't consider recompiling a Linux Kernel a fun thing to do in his or her free time.

Why this book? Because Mom and Dad shouldn't have to take a course on MySQL administration just to enter their expenses and file their taxes.

Why this book? Because I had to do all of this research, which meant I had to take a lot of notes anyway.

Why this book? Because OpenSource libraries don't come with squat for documentation.

Why xBaseJ?

That's a good question. Part of the answer has to do with the history I provided in the previous section. The other part has to do with the language chosen.

I don't do much PC programming anymore. I needed this application to run on both Ubuntu Linux and Windows. There isn't a "good" OpenSource cross-platform GUI library out there. Most of the Linux GUI libraries require a lot of stuff to be installed on a Windows platform (usually the bulk of Cygwin) and that requires writing some kind of installation utility. Let's just say that the OpenSource installation generation tools for Windows haven't quite caught up to their expensive commercial counterparts. You don't really want to saddle a Windows machine which has the bare minimum Windows configuration with something like Cygwin on top of it.

When I did do PC programming, I never really did much with TCP/IP calls directly. If I magically found an OpenSource cross-platform GUI which did everything I needed on both Linux and Windows, I was still going to have to find a cross-platform TCP/IP library. Let us not forget that some 64-bit Linux distros won't run 32-bit software and some 32-bit software won't run on 64-bit versions of Windows Vista. Programming this in C/C++ was going to require a lot more effort than I wanted to put into something I would basically hand out free once it was working correctly. (You may also have noticed I didn't even mention finding a library which would work on Windows, Linux, *and* MAC.)

Java, while not my favorite language, tends to be installed on any machine which connects to the Internet. Most Windows users know where to go to download and install the JRE which isn't installed by default for some versions of Windows. From what I hear, the pissing contest is still going on between what is left of Bill Gates' s Evil Empire and what is left of Sun.

Java, while not my favorite language, provides a GUI for almost every platform it runs on.

Java, while not my favorite language, makes opening a URL and parsing through the text to find certain tokens pretty simple if you happen to know what class to use.

Java, while not my favorite language, will not care if the underlying operating system is 32- or 64-bit.

Most machines which use a browser and connect to the Web have some form of the Java Run-time Environment (JRE) installed on them. This is true of current Linux, MAC, and Windows machines.

Obviously I was going to have to develop this package with a language that wasn't my favorite.

The only question remaining was data storage. Would I force Mom, Dad, and Aunt Carol to enroll in a MySQL administration course even though they can barely answer email and find MapQuest on the Internet, or was I going to use something self-contained? Given my earlier tirade, you know I wanted to use something self-contained just to preserve my own sanity.

At the time of this writing, a search on SourceForge using "java index file" yields just shy of 29,000 projects and a search using "java xbase" yields just shy of 20,000 projects. Granted, after you get several pages into the search results, the percentage of relevancy drops exponentially, but there are still a lot of choices. Btree type indexed files which store the index in the file with the data tend to be far more reliable from a data integrity standpoint. All indexes are always kept in sync by the library/engine. xBASE type files store the indexes off in different files. You can add all of the records you want to an xBASE data file without ever updating an index.

I can hear the head-scratching now. 'But if that's true, why would you use 30-year-old xBASE technology instead of a Btree?' Because of the tools, child. OpenOffice can open a DBF file in a spreadsheet to let a user view the data. If any of these files become corrupted there are literally hundreds of xBASE repair tools out there. If a user decides he or she wants to load the data into some other database format for analysis, there are tools out there to export xBASE into CSV (Comma Separated Value) files which can easily be imported by most relational database engines. Some relational database engines can directly import xBASE files. Nearly every programming language out there has some form of xBASE library, or can call one written in some other language. Perl even has an xBASE library that I've used to extract data from an expense tracking system before. Under Linux, there is even a dbf_dump utility (dbfdump on OpenSuSE for some reason) which will let you dump a DBF file to a CSV in one command.

```
dbfdump /windows/D/net_f/xpns_$tax_year/payee.dbf > payee.csv
```

What happens if I use one of those really fast Btree or B+tree libraries and the user needs to get the data out? Such users cuss me pretty hard when none of the office suites on their computer can open the file to do an export. When they track me down via the Web and call my office, they get disappointed finding out I don't have time to drop everything and fly to their location to help them free of charge. Then they say my name, spit, and start bad-mouthing me all over the Internet. *That* really helps my consulting business.

Now that we have determined the data will be stored in an xBASE file format, we only have to choose an OpenSource xBASE library for Java. I selected xBaseJ because it used to be a commercial library known as XbaseJ and was sold by BMT Micro. The product has since become an OpenSource Project which gets periodic improvements. The developer actually monitors his SourceForge support forum and seems to be actively adding new things to the library. Some things don't work out so well, like the DTD to xBASE XML parser, but the attempt was made. Someone else in the community might finish it.

Please pay attention to the thought process of this section. A seasoned systems analyst and/or consultant goes through exactly this thought process when he or she tries to design a system. You look at what is available on the target platform, then walk backwards trying to reduce the amount of pain you feel when you can't change the target platform. I cannot change the computers people have, nor their personal skill levels. I have to design an application based upon the ability of the user, not my ability to be creative, or the tools I would prefer to use.

A Brief History of xBASE

There are many variations in the capitalization of xBASE, which is I guess fitting, since there are many slight variations for the actual file formats. The history of xBASE is a sordid tale, but all versions of xBASE in one way or another trace their roots back to the 1970s and the Jet Propulsion Laboratory. Here is the tale as best I can remember.

PCs were originally very expensive. In the late 1970s you could buy a "well equipped" Chevrolet Caprice Classic 4-door sedan for just over \$4,000. In the early 1980s you could buy a dual floppy monochrome PC for about the same amount. When clone vendors entered the market you started seeing dual floppy clone PCs for under \$2,000. The higher-end PCs started adding full height 10MEG hard drives to justify keeping their price so high. Eventually, you could get a clone PC with a whopping 20MEG hard drive for nearly \$2000.

Once that \$2000 price point for a PC with a hard drive was achieved, the PC started getting pushed into the world of business. The first thing the businesses wanted to do with it was keep things in sorted order. They heard from kids enrolled in computer programming courses that midrange and mainframe computers used a language called COBOL which supported indexed files that could be used to store invoices, payments, etc., all in sorted order, so information was quickly (for the day) retrievable. Well, the PC didn't have that, and business users needed it. There was a non-commercial product called Vulcan written by Wayne Ratliff which kind of answered some of those needs. Ashton-Tate eventually released a commercial product named dBASE II. (They used II instead of I to make the product seem more stable. I'm not making that up.)

Ashton-Tate had a lot of sales, a lot of money, a lot of attitude, and a lot of lawyers. This led to them believing they had the rights to all things dBASE. When the cash cow started giving lots of green milk the clone vendors piled into the fray. Ashton-Tate let loose with a blizzard of lawsuits trying to show it was the meanest dog in the junkyard. The clone vendors quickly got around the dBASE trademark infringement by calling their file formats xBASE. (Some called theirs X-Base, others XBase, etc.)

Times and public sentiment turned against Ashton-Tate. The people who spent many hundreds of dollars for these tools and even more money for some of the run-time licenses which had to be in place on the machines before applications written with the tool wanted a standard. When they were finally fed up with Ashton-Tate or one of the clones, they naively believed it would be like those old COBOL programs, recompile and run. Silly customers. This was the peak of proprietary software (the height of which turned out to be Microsoft Windows, which even today is considered one of the most proprietary operating systems running on a PC architecture), and there was no incentive for any of those receiving run-time license fees to agree to a standard. Well, no incentive until the business community as a whole deemed the fees they charged too high.

When the price of a run-time license reached hundreds, the business community cried foul. When the memory footprint of the run-time meant you couldn't load network drivers or other applications in that precious 640K window accessible by DOS, dirty laundry got aired rather publicly.

Vulture Capitalists, always sniffing the wind for dirty laundry and viewing it as opportunity, started hurling money at software developers who said they could write a C programming library which would let other programmers access these files without requiring a run-time image or license. The initial price tag for those libraries tended to be quite high. Since there were no royalty payments, the developers and the Vulture Capitalists thought the “best” price they could offer was something totalling about half of what the big corporations were currently paying for development + run-time license fees. For a brief period of time, they were correct. Then the number of these libraries increased and the price got down to under \$500 each. The companies vending products which required run-time license fees saw their revenue streams evaporate.

The evaporation was a good thing for the industry. It allowed Borland to purchase Ashton-Tate in 1991. Part of the purchase agreement appears to have been that Ashton-Tate drop all of its lawsuits. After that committee ANSI/X3J19 was formed and began working on xBASE standards. In 1994 Borland ended up selling the dBASE name and product line to dBASE Inc.

The standards committee accomplished little, despite all the major vendors participating. More of the data file formats, values, and structures were exposed by each vendor, but each of the vendors in the meetings wanted every other vendor to adopt *its* programming language and methods of doing things so it would be the first to market with the industry standard.

There are still commercial xBASE vendors out there. Microsoft owns what was Foxbase. dBASE is still selling products and migrating into Web application creation. Most of the really big-name products from the late 80s are still around; they just have different owners. Sadly, Lotus Approach was dropped by IBM and not resurrected when it came out with the Symphony Office Suite.

I will hazard a guess that some of the C/C++ xBASE programming libraries from my DOS days are still around and being sold by someone. That would make sense now that FreeDOS is starting to get a following. Not quite as much sense given all of the OpenSource C/C++ xBASE libraries out there, but the old commercial tools have a lot more time in the field, and should therefore be more stable. I know that Greenleaf is back in business and you can probably get a copy of GDB (Greenleaf Database) from them; I just don't know what platforms they still support.

There is a lot of history and folklore surrounding the history of xBASE. You could probably make a movie out of it like they made a movie out of the rise of Microsoft and Apple called “Pirates of Silicon Valley” in 1999. You can piece together part of the history, at least from a compatibility standpoint, by obtaining a copy of *The dBASE Language Handbook* written by David M. Kalman and published by Microtrend Books in 1989. Another work which might be worthy of your time is *Xbase Cross Reference Handbook* written by Sheldon M. Dunn and

published in 1993 by Sybex, Inc.

For our purposes, you only need to know that back in the 1970s the Jet Propulsion Laboratory needed an indexed file system to store data for various retrieval needs. What they designed eventually developed many flavors, but all of those flavors are now lumped under the xBASE heading by the general public.

What is xBASE?

Note: This information is correct. You will find other information on the Web which completely contradicts portions of this information, and it will also be correct. What you have to take into account is the *point in time* the information references. There are some new tools on the market which claim they are xBASE and have no maximum file size. As you will see later, this is not the original xBASE, which has a 1,000,000,000 byte file size limit, nor the later DOS/Windows xBASE products, which eventually expanded the maximum file size to 2GB. xBASE evolved with the DOS PC from the time when we had dual floppy systems which would have at least 64K of RAM, but could have all the way up to 640K. There was a time when the largest hard drive you could buy for a PC on the retail market was 20MB.

Note 2: A lot of well-meaning people have taken the time to scan in or re-key documentation from the 1980s which shipped with various products. I applaud their efforts. Hopefully we will find some method of parsing through all of this documentation and updating it for today's environment. The most confusing things you will read are where actual product literature says, "The maximum file size is 1,000,000,000 bytes unless large disk support is enabled, then you are limited only by the size of your disk." At the point in time when the author wrote that the xBASE format could store 2GB and the largest disk drive on the market was 1.08GB. The statement is blatantly wrong now, but the on-line documentation is still trapped at that point in time. I remember this point in time well. Shortly after that documentation came out, SCSI drives started increasing in size all of the way up to 8GB in less than a year. A lot of customers hit that 2GB wall pretty hard, then reached for lawyers claiming fraud. It wasn't deliberate fraud, it was simply outdated information.

Most on-line references will say that Xbase (xBASE) is a generic term for the dBASE family of database languages coined in response to threats of litigation over the copyrighted trademark "dBASE." That would be true for a point in time long ago. Today xBASE really refers to the data storage specification, not the language(s) involved in the application. People who are programmers know this; people who aren't programmers don't appear to have the ability to reason it out.

I have already talked about the various C/C++ xBASE libraries which are out there. If the definition found on-line were true, it would require those libraries to parse a dBASE script and execute it, rather than directly access the data and index files. The same would be required of the xBaseJ library we will be covering in this book. Most libraries don't provide any kind of script parsing capability. What they do provide are functions with names very close to some of the original dBASE syntax, along with a lot of other functions that access the data and index files.

Putting it in simple terms, xBASE is a system of flat files which can organize data in a useful manner when one or more specific sets of format rules are followed. Each file is in two parts: a file header and actual contents. Each header has two parts: a file descriptor and a content descriptor. A lot of definitions you find published and on-line won't use the word "content," they will use the word "record." Those definitions are only accurate for the data file. While it is true that each index value could be viewed as a record containing a key value, record number, sort order information and other internal data, we don't have any concept of the record organization unless we are writing an xBASE library of some kind.

The above does not describe a relational database by any stretch of the imagination. There have been various products on the market which put SQL type syntax around xBASE file structures, but the organization really is flat file. If you have gone to school for computer programming, you may have encountered the term "relative file." A relative file is accessed by record number, not a key value. It is one of the simplest file structures to create and is the foundation of several other file systems.

You may have also encountered the term "hashed file" or "hash file." This is an enhancement to the relative file. A particular field or set of fields is chosen from a record to be considered a "key value." Some form of algorithm (usually a math function) is fed the key value and out the other side of the function comes the record number where the record you want "should" be. If you have a really bad hash algorithm you end up with multiple keys hashing to the same record number, a condition known as "hash collision" or simply "collision." The program then has to go sequentially through from that record either forward or backward depending upon key sort order, until it finds the record you are looking for, or a key so different that it can tell your record isn't in the file. Almost every programmer has to write a program like this while earning his or her bachelors degree.

There was a lot of brain power involved with the creation of xBASE. You might remember that I told you it was a creation which fell out of the Jet Propulsion Laboratory and into the commercial world. When you write a data record to an xBASE file, it gets written contiguously in the next available slot. The actual record number is recorded with the key value(s) in the indexed files which are both open and associated with the data file. When you want to find a record, all of the dancing occurs in the file containing the index. As a general rule key values are smaller than record values so you can load/traverse many more of them in a shorter period of time. Once the engine locates the key, it has the record number for a direct read from the data file. The really good libraries and engines will also verify the key on the record read actually matches the key value from the index file. (More on that topic later.)

I don't know what magic actually happens when the key is being processed and I don't care. If you really want to find out, xBaseJ comes with source code, as do many other OpenSource projects which create and process xBASE files. Pull down the source and plow through it. From an application developer standpoint, all we need to know is that if the index file is open and associated with the data file, it will be updated. When a key is found we get the record and when it isn't we get an error value.

It is important to note that the original xBASE file systems stored only character data in the data files. Numbers and dates were all converted to their character representations. This severe restriction made the design highly portable. Binary data is far more efficient when it comes to storage, but tends to be architecture specific. (Refer to "The Minimum You Need to Know to Be an OpenVMS Application Developer" ISBN-13 978-0-9770866-0-3 page 10-3 for a discussion on Little Endian vs. Big Endian and data portability.)

Another benefit this severe restriction created was that it allowed non-programmers the ability to create databases. The average Joe has no idea what the difference between Single and Double precision floating point is or even what either of those phrases mean. The average MBA wouldn't know what G_FLOAT, F_FLOAT, and D_FLOAT were or that they exist even if the terms were carved on a 2x4 that smacked them between the eyes. The average user could understand "9 digits in size with 3 decimal digits," though. By that time in America, most everyone had filled out some government tax withholding or other form that provided neat little boxes for you to write digits in.

DOS, and by extension Windows, made significant use of three-character file extensions to determine file types. Linux doesn't support file extensions. It can be confusing for a PC user when they see MYFILE.DBF on a Linux machine and they hear the “.” is simply another character in a file name. It is even more confusing when you read documentation for applications written initially for Linux, like OpenOffice, and it talks about “files with an ODT” extension. I came from multiple operating systems which all used file extensions. I don't care that I'm writing this book using Lotus Symphony on KUbuntu, I'm going to call “.NNN” a file extension and the purists can just put their fingers in their ears and hum really loud.

The original file extension for the dBASE data file was .DBF. Some clone platforms changed this, and some did not. It really depended on how far along the legal process was before the suits were dropped. In truth, you could use nearly any file extension with the programming libraries because you passed the entire name as a string. Most of the C/C++, and Java libraries look at a special identifier value in the data file to determine if the file format is dBASE III, dBASE IV, dBASE III with Memo, dBASE IV with Memo, dBASE V without memo, FoxPro with Memo, dBASE IV with SQL table, Paradox, or one of the other flavors. Foxbase and FoxPro were actually two different products.

The Memo field was something akin to a train wreck. This added the DBT file extension to the mix (FPT for FoxPro.) A Memo field was much as it sounded, a large free-form text field. It came about long before the IT industry had an agreed upon “best practice” for handling variable length string fields in records. The free form text gets stored as an entity in the DBT file, and a reference to that entity was stored in a fixed length field with the data record.

You have to remember that disk space was still considered expensive and definitely not plentiful back in those days. Oh, we thought we would never fill up that 80MEG hard drive when it was first installed. It didn't take long before we were back to archiving things we didn't need right away on floppies.

The memo field gave xBASE developers a method of adding “comments sections” to records without having to allocate a great big field in every data record. Of course, the memo field had a lot of different flavors. In some dialects the memo field in the data record was 10 bytes plus however many bytes of the memo you wanted to store in the data record. The declaration M25 would take 35 bytes in the record. According to the CodeBase++ version 5.0 manual from Sequiter Software, Inc., the default size for evaluating a memo expression was 1024. The built-in memo editor/word processor for dBase III would not allow a user to edit more than 4000 bytes for a memo field. You had to load your own editor to get more than that into a field.

Memo files introduced the concept of “block size” to many computer users and developers. When a memo file was created it had a block size assigned to it. All memo fields written to that file would consume a multiple of that block size. Block sizes for dBASE III PLUS and Clipper memo files were fixed at 512 and there was a maximum storage size of 32256 bytes. Foxpro 2.0 allowed a memo block size to be any value between 33 and 16384. Every block had 8 bytes of overhead consumed for some kind of key/index value.

Are you having fun with memo fields yet? They constituted a good intention which got forced into all kinds of bastardizations due to legal and OS issues. Size limitations on disks tended to exceed the size limitations in memory. DOS was not a virtual memory OS, and people wanted ANSI graphics (color) applications, so, something had to give. A lot of applications started saying they were setting those maximum expression sizes to limit memo fields to 1024 bytes (1008 if they knew what they were doing $512 - 8 = 504 * 2 = 1008$.) Naturally the users popped right past the end of this as they were trying to write *War and Peace* in the notes for the order history. Sometimes they were simply trying to enter delivery instructions for rural areas when it happened. There were various “standard” sizes offered by all of the products during the days of lawsuits and nasty grams. 4096 was another popular size limit, as was 1.5MEG.

The larger memo size limits tended to come when we got protected mode run-times that took advantage of the 80286 and 32-bit DOS extenders which could take advantage of the 80386/80486 architectures. (The original 8086/8088 CPU architecture could only address 1 Meg of RAM while the 80286 could address 16 Meg in protected mode. The 80386DX could address 4GB directly and 64TB of virtual memory.) I just checked the documentation at <http://www.dbase.com> and they claim in the current product that a memo field has no limit. I also checked the CodeBase++ 5.0 manual, and Appendix D states memo entry size is limited to 64K. The 64K magic number came from the LIM (Lotus-Intel-Microsoft) EMS (Expanded Memory Standard). You can read a pretty good write-up in layman' terms by visiting http://www.atarimagazines.com/compute/issue136/68_The_incredible_expan.php

If you think memo fields were fun, you should consider the indexed files themselves. Indexes aren' stored with the data in xBASE formats. Originally each index was off in its own NDX file. You could open a data file without opening any associated index, write (or delete) records from it, then close, without ever getting any kind of error. As a general rule, most “production” applications which used xBASE files would open the data file, then rebuild the index they wanted, sometimes using a unique file name. This practice ended up leaving a lot of NDX files laying around on disk drives, but most developers engaging in this practice weren' trained professionals, they were simply getting paid to program; there *is* a difference.

It didn't take long before we had Multiple Index Files (MDX), Compound Index Files (CDX), Clipper Index Files (NTX), Database Container (DBC), and finally IDX files, which could be either compressed or un-compressed. There may even have been others I don't remember.

MDX was a creation which came with dBASE IV. This was a direct response to the problems encountered when NDX files weren't updated as new records were added. You could associate a "production" MDX file with a DBF file. It was promised that the "production" MDX file would be automatically opened when the database was opened...unless that process was deliberately overridden by a programmer. This let the run-time keep indexes up to date. Additional keys could be added to this MDX up to some maximum supported number. I should point out that a programmer could create non-production MDX files which weren't topped automatically with the DBF file. (xBaseJ is currently known to have compatibility issues with dBASE V formats and MDX files using numeric and/or date key datatypes.) MDX called the keys it stored "tags" and allowed up to 47 tags to be stored in a single MDX.

While there is some commonality of data types with xBASE file systems, each commercial version tried to differentiate itself from the pack by providing additional capabilities to fields. This resulted in a lot of compatibility issues.

<i>Type</i>	<i>Description</i>
+	Autoincrement – Same as long
@	Timestamp - 8 bytes - two longs, first for date, second for time. The date is the number of days since 01/01/4713 BC. Time is hours * 3600000L + minutes * 60000L + Seconds * 1000L
B	10 digits representing a .DBT block number. The number is stored as a string, right justified and padded with blanks. Added with dBase IV.
C	ASCII character text originally < 254 characters in length. Clipper and FoxPro are known to have allowed these fields to be 32K in size. Only fields <= 100 characters can be used in an index. Some formats choose to read the length as unsigned which allows them to store up to 64K in this field.
D	Date characters in the format YYYYMMDD
F	Floating point - supported by dBASE IV, FoxPro, and Clipper, which provides up to 20 significant digits of precision. Stored as right-justified string padded with blanks.
G	OLE – 10 digits (bytes) representing a .DBT block number, stored as string, right-justified and padded with blanks. Came about with dBASE V.

<i>Type</i>	<i>Description</i>
I	Long - 4 byte little endian integer (FoxPro)
L	<p>Logical - Boolean – 8 bit byte. Legal values ? = Not initialized Y,y Yes N,n No F,f False T,t True</p> <p>Values are always displayed as ‘T’, ‘F’, or ‘?’. Some odd dialects (or more accurately C/C++ libraries with bugs) would put a space in an un-initialized Boolean field. If you are exchanging data with other sources, expect to handle that situation.</p>
M	<p>10 digits (bytes) representing a DBT block number. Stored as right-justified string padded with spaces.</p> <p>Some xBASE dialects would also allow declaration as Mnn, storing the first nn bytes of the memo field in the actual data record. This format worked well for situations where a record would get a 10-15 character STATUS code along with a free-form description of why it had that status.</p> <p>Paradox defined this as a variable length alpha field up to 256MB in size.</p> <p>Under dBASE the actual memo entry (stored in a DBT file) could contain binary data.</p> <p><i>xbaseJ does not support the format Mnn and neither do most OpenSource tools.</i></p>
N	Numeric Field – 19 characters long. FoxPro and Clipper allow these fields to be 20 characters long. Minus sign, commas, and the decimal point are all counted as characters. Maximum precision is 15.9. The largest integer value storable is 999,999,999,999,999. The largest dollar value storable is 9,999,999,999,999.99
O	Double – no conversions, stored as double
P	Picture (FoxPro) Much like a memo field, but for images
S	Paradox 3.5 and later. Field type which could store 16-bit integers.

<i>Type</i>	<i>Description</i>
T	DateTime (FoxPro)
Y	Currency (FoxPro)

There was also a bizarre character name variable which could be up to 254 characters on some platforms, but 64K under Foxbase and Clipper. I don' have a code for it, and I don' care about it.

Limits, Restrictions, and Gotchas

Our library of choice supports only L, F, C, N, D, P, and M without any numbers following. Unless you force creation of a different file type, this library defaults to the dBASE III file format. You should never ever use a dBASE II file format or, more importantly, a dBASE II product/tool on a data file. There is a field on the file header which contains a date of last update/modification. dBASE III and later products have no problems, but dBASE II ceased working some time around Jan 1, 2001.

Most of today' libraries and tools support dBASE III files. This means they support these field and record limitations:

- dBASE II allowed up to 1000 bytes to be in each record. dBASE III allowed up to 4000 bytes in each record. Clipper 5.0 allowed for 8192 bytes per record. Later dBASE versions allowed up to 32767 bytes per record. Paradox allowed 10800 for indexed tables but 32750 for non-indexed tables.
- dBASE III allowed up to 1,000,000,000 bytes in a file without "large disk support" enabled. dBASE II allowed only 65,535 records. dBASE IV and later versions allowed files to be 2GB in size, but also had a 2 billion record cap. At one point FoxPro had a 1,000,000,000 record limit along with a 2GB file size limit. (Do the math and figure out just how big the records could be.)
- dBASE III allowed up to 128 fields per record. dBASE IV increased that to 255. dBASE II allowed only 32 fields per record. Clipper 5.0 allowed 1023 fields per record.
- dBASE IV had a maximum key size of 102 bytes. FoxPro allowed up to 240 bytes and Clipper 388 bytes.
- Field/column names contain a maximum of 10 characters.

I listed some of the non-dBASE III values to give you a sense of what you might be up against when a friend calls you up and says ‘I’ got some data on an old xBASE file, can you extract it for me?’ The flavors of xBASE which went well beyond even dBASE IV limitations have very limited support in the OpenSource community.

Let me say this plainly for those who haven’ figured it out: *xbase is like Linux. There are a zillion different flavors, no two of which are the same, yet, a few core things are common, so they are all lumped together under one heading.*

If you read through the comments in the source files, you’ lkee that xBaseJ claims to support only dBASE III and dBASE IV. If you are looking for transportability between many systems, this is the least common denominator (LCD) and should work in most cases. The comments may very well be out of date, though, because the createDBF() protected method of the DBF class supports a format value called FOXPRO_WITH_MEMO.

When I did a lot of C/C++ programming on the PC platform, I found GDB (Greenleaf Database Library) to be the most robust library available. I had used CodeBase from Sequiter Software and found it to be dramatically lacking. With the C version of their library, you could not develop an application which handled dBASE, FoxPro, and Clipper files simultaneously. Their entire object library was compiled for a single format at a time. GDB created separate classes and separate functions to handle opening/creating all of the database formats it supported. Each of those root classes/structures were tasked with keeping track of and enforcing the various limits each file type imposed. The library was also tested under Windows, Win-32, generic DOS, 16-bit DOS, 32-bit DOS, and OS/2. It was the cream of the crop and very well may still be today.

I’ nbringing up those commercial libraries to make a point here. After reading through the code, I have come to the conclusion that only the format was implemented by xBaseJ, not all of the rules. When you read the source for the DBF class, you will see that if we are using a dBASE III format, a field count of 128 is enforced, and everything else is limited to 255. The truth is that the original DOS-based Foxbase had a field limit of 128 as well, but that format isn’ directly supported.

There is also no check for maximum record length. The DBF class has a protected short variable named lrecl where it keeps track of the record length, but there are no tests that I could see implementing the various maximum record lengths. In truth, since it supports only a subset of the formats, a hard-coded test checking against 4000 would work well enough. Not a lot of DOS users out there with legitimate dBASE III Plus run-times to worry about.

Another gotcha to watch out for is maximum records. The DBF class contains this line of code:

```
file.writeInt(Util.x86(count));
```

All the Util.x86 call does is return a 4-byte buffer containing a binary representation of a long in the format used by an x86 CPU. (Java has its own internal representation for binary data which may or may not match the current CPU representation.) The variable “file” is simply an instance of the Java RandomAccessFile class, and writeInt() is a method of that class. There is no surrounding check to ensure we haven't exceeded a maximum record count for one of the architectures. Our variable count happens to be a Java int which is 32-bits. We know from our C programming days (or at least the C header file limits.h) the following things:

<i>Type</i>	<i>16-bit</i>	<i>32-bit</i>
unsigned	65,535	4,294,967,295
signed	32,767	2,147,483,647

While we will not have much trouble when handing data over to the other OpenSource tools which don't check maximums, we could have trouble if we added a lot of records to a file flagged as dBASE III then handed it off to an actual dBASE III run-time. Record maximums weren't as big a problem as file size. That funky 1 billion byte file size limit was a result of DOS and the drive technology of the day. We had a 1Gig wall for a while. Even after that barrier had been pushed back to 8Gig we still had that built-in 1Gig limit due in large part to 16-bit math and the FAT-16 disk structure used at the time. Most of you now use disk storage formats like FAT-32, NTFS, HPFS, EXT3, or EXT4. None of these newer formats have the 16-bit problems we had in days gone by. (For what it is worth, DOS floppy format still uses FAT-16.)

1 disk block = 512 bytes

1K = 1024 bytes or 2 blocks

1Meg = 1K squared or 1024² block units

1GB = 1K cubed or 1024 bytes * 1024 * 1024 = 1,073,741,824

1GB / 512 = 2,097,152 disk blocks

2GB = 2 * 1GB = 2,147,483,648 (notice 1 greater than max signed 32-bit value)

2GB / 512 = 4,194,304 disk blocks

4GB = 4 * 1GB = 4,294,967,296 (notice 1 greater than max unsigned 32-bit value)

4GB / 512 = 8,388,608 disk blocks

32767 * 512 = 16,776,704

16Meg = 16 * 1024 * 1024 = 16,777,216

Large disk support, sometimes referred to as “large file support” got its name from the DOS FDISK command. Whenever you tried to use the FDISK command after Windows 95 OSR2 came out on a disk larger than 512MB, it would ask you if you wanted to enable large disk support. What that really did was switch from FAT16 to FAT32. Under FAT32 you could have files which were up to 4GB in size and a partition 2TB in size. I provided the calculations above so you would have some idea as to where the various limits came from.

Today xBASE has a 2Gig file size limit. As long as xBASE remains 32-bit and doesn't calculate the size with an unsigned long, that limit will stand. I told you before that xBASE is a relative file format with records “contiguously” placed. When you want to load record 33, the library or xBASE engine takes the start of data offset value from the file header, then adds to it the record number minus one times the record size to obtain the offset where your record starts. Record numbers start at one, not zero. Some C/C++ libraries use the exact same method for writing changes to the data file as they do for writing new records. If the record number provided is zero, they write a new record; otherwise they replace an existing record.

In case the previous paragraph didn't make it obvious to you, data records are fixed length. Do not confuse entries in a memo file with data records. You can't treat an index on a memo file, or really do much more than read or write to it.

Various file and record locking schemas have been used throughout the years by the various xBASE flavors. During the dark days of DOS, a thing called SHARE.EXE came with the operating system. It never worked right.

SHARE could lock chunks of files. This led to products like MS Access claiming to be multi-user when they weren't. It also led to the infamous “Two User Boof” bug. Access (and several other database products at the time) decided to organize the internal database structure around arbitrary page sizes. A page was basically some number of 512 byte blocks. It was common to see page sizes of 8196 bytes, which was 16 blocks. SHARE would then be instructed to lock a page of the database file. A page actually contained many records. If two users attempted to modify different records on the same page, the second user's update would dutifully be blocked until the first user's update was written to disk. IO was performed a page at a time in order to increase overall efficiency. The update logic would dutifully check the contents of the modified record on disk to ensure nobody else had changed it before applying the updates. What the IO process didn't do was check every damned record in the page for changes. The last one in won. All changes made by the first user were lost. Some developers ended up making a record equal to a page as a cheap hack-type work around. A lot of disk was wasted when this was done.

Summary

Despite all of its limitations and faults, the xBASE data storage method was groundbreaking when it hit the market. Without some form of indexed file system, the PC would not have caught on.

It is important for both users and developers to understand the limitations of any chosen storage method before developing an application or systems around that method. While a relational database is much more robust from a data storage standpoint, it requires a lot more investment and overhead. Even a “free” relational database requires someone to install and configure it before an application can be written using it. A developer can use a C/C++/Java/etc. library and create a single executable file which requires no configuration, simply an empty directory to place it in. That program can create all of the files it needs then allow a user to store and access data in a meaningful fashion without them having any significant computer skills.

There will always be a role for stand-alone indexed file systems. Both commercial and OpenSource vendors need data storage methods which require no user computer skills. Just how many copies of Quicken do you think would have ever sold if a user had to download+install+configure a MySQL database before Quicken would install and let them track their expenses? No matter how old the technology is, the need for it still exists.

Review Questions

1. How many fields did dBASE III allow to be in a record?
2. What general computing term defines the type of file an xBASE DBF really is?
3. What does xBASE mean today?
4. What was the non-commercial predecessor to all xBASE products?
5. In terms of the PC and DOS, where did the 64K object/variable size limit really come from?
6. What company sold the first commercial xBASE product?
7. Is there an ANSI xBASE standard? Why?
8. What is the maximum file size for a DBF file? Why?
9. What was the maximum number of bytes dBASE III allowed in a record? dBASE II?
10. What form/type of data was stored in the original xBASE DBF file?
11. Can you store variable length records in a DBF file?
12. Does an xBASE library automatically update all NDX files?
13. What is the accepted maximum precision for a Numeric field?
14. What is the maximum length of a field or column name?

Page left blank intentionally.

Chapter 1

Fundamentals

1.1 Our Environment

I am writing the bulk of this code on a desktop PC running the 32-bit Karmic Koala pre-release of KUbuntu. I have Sun Java 6 installed on this machine, but several earlier releases of Java should work just fine with this library.

After unzipping the download file, I copied the JAR files into a working directory. Of course, the newer Java environments will only look for class files locally, not JAR files, so you need to create a CLASSPATH environment variable. I use the following command file since it loads just about everything I could want into CLASSPATH:

env1

```
1)  #!/bin/bash
2)  #set -v
3)  #sudo update-java-alternatives -s java-6-sun
4)
5)  export JAVA_HOME='/usr/lib/jvm/java-6-sun'
6)
7)  set_cp() {
8)  local curr_dir=$(echo *.jar | sed 's/ /:/g')': '
9)  local jvm_home_jars=$(echo $JAVA_HOME/*.jar | sed 's/ /:/g')': '
10) local shr_jars=$(echo /usr/share/java/*.jar | sed 's/ /:/g')': '
11) local loc_jars=$(echo /usr/local/share/java/*.jar | sed 's/ /:/g')': '
12) if [ "$curr_dir" == "*.jar" ]; then
13)   unset curr_dir
14) fi;
15) export CLASSPATH=$(echo .:$curr_dir$jvm_home_jars$shr_jars$loc_jars)
16) }
17)
18) ecp() {
19) echo $CLASSPATH | sed 's:/\n/g'
20) }
21)
22) # set class path by default
23) set_cp
24)
25) #set +v
```

```
roland@logikaldesktop:~$ cd fuelsurcharge2
roland@logikaldesktop:~/fuelsurcharge2$ echo $CLASSPATH
```

```
roland@logikaldesktop:~/fuelsurcharge2$ source ./env1
roland@logikaldesktop:~/fuelsurcharge2$ echo $CLASSPATH
.:commons-logging-1.1.1.jar:junit.jar:xBaseJ.jar:xercesImpl.jar:/usr/lib/jvm/
java-6-sun/*.jar:/usr/share/java/hsqldb-1.8.0.10.jar:/usr/share/java/hsqldb.jar:/
usr/share/java/hsqldbutil-1.8.0.10.jar:/usr/share/java/hsqldbutil.jar:/usr/share/
java/ItzamJava-2.1.1.jar:/usr/share/java/jsp-api-2.0.jar:/usr/share/java/jsp-
api.jar:/usr/share/java/LatestVersion.jar:/usr/share/java/libintl.jar:/usr/share/
java/mysql-5.1.6.jar:/usr/share/java/mysql-connector-java-5.1.6.jar:/usr/share/
java/mysql-connector-java.jar:/usr/share/java/mysql.jar:/usr/share/java/
QuickNotepad.jar:/usr/share/java/servlet-api-2.4.jar:/usr/share/java/servlet-
api.jar:/usr/local/share/java/*.jar:
```

As you can see, that script finds every JAR file and adds it to my environment variable. The occasional “*.jar” value in the symbol definition doesn't appear to impact the JVM when it goes searching for classes. If you don't have the JAR files specifically listed in your CLASSPATH variable, then you will see something like this the first time you try to compile:

```
roland@logikaldesktop:~/fuelsurcharge2$ javac example1.java
example1.java:3: package org.xBaseJ does not exist
import org.xBaseJ.*;
^
example1.java:4: package org.xBaseJ.fields does not exist
import org.xBaseJ.fields.*;
^
example1.java:5: package org.xBaseJ.Util does not exist
import org.xBaseJ.Util.*;
^
example1.java:18: cannot find symbol
symbol  : variable Util
location: class example1
    Util.setXBaseJProperty("fieldFilledWithSpaces", "true");
```

Windows users will need to view the information provided by Sun on how to set the CLASSPATH variable.

<http://java.sun.com/j2se/1.4.2/docs/tooldocs/windows/classpath.html>

You may also wish to look at this message thread:

<http://www.computing.net/answers/programming/how-to-set-java-classpath-in-vista/15739.html>

1.2 Open or Create?

You will find quite a few examples of various xBASE programming languages/tools on the Web. Examples are a little scarce for xBaseJ, as is documentation, hence, the creation of this book. Most of the examples piss me off. I understand that they are trying to show the simplest of things to a user who may have no other computer knowledge, but those well-meaning examples show a user how to do things badly, and that is exactly how they will continue to do them.

The main Web site for xBaseJ has two examples which, while well meaning, fall into this category: example1.java and example2.java. The first creates a database, the second opens it. While you can argue that the create always wanted to create, just having the open example crash out when the file is missing is probably not what you want when developing an application which will be sent out into the universe. Most of you don't even think about why some applications take so long to start up the very first time you run them. The startup code for those applications is very graciously running around checking for all of the necessary data files. When files are missing, it creates default ones. Just how many of you would use a word processor if it required you to run some special (probably undocumented) program before it would do anything other than crash out?

When I imbibe enough caffeine and think about it, the real problem is the constructor using a Boolean for the “destroy” parameter. A Boolean gives you only True and False. A production class system needs three options:

1. Use existing
2. Overwrite
3. Create if missing

If you have read some of my other books you will know that many languages name this type of parameter or attribute “disposition” or “file disposition.” The DBF constructor doesn't have a “file disposition” attribute, so we have some less-than-great examples floating around.

I'm not going to discuss Java much in this book. I will point out oddities as I see them, but if you are looking for a Java tutorial, there are many of those on the Web. I've even written a book on Java which some people like. (“The Minimum You Need to Know About Java on OpenVMS Volume 1” ISBN-13 978-0-9770866-1-0) I'm a veteran software developer, but not a tenured Java developer. A few discussions of oddities aside, we are really focusing on how to use xBaseJ with Java in this book.

There are very few classes of applications which always need to create an indexed file when they run. Most business systems use the disposition of “Create if missing.” Many will display some kind of message stating they are creating a missing indexed file, just in case it wasn't supposed to be missing, but in general, only extract-type applications always need to create when it comes to indexed files.

In case you do not understand the phrase “extract-type applications,” these are applications which are run against large data sets that pull out copies of records/rows which meet certain criteria and place these copies in a file. The file is known as an extract file and the application which creates it an extract application.

1.3 Example 1

example1.java is representative of the first example program floating around on the Web at the time of this writing. Note that some older examples don't show the proper import statements. You need to include the full path as I have done with listing lines 3 through 5.

example1.java

```

1)  import java.io.*;
2)  import java.util.*;
3)  import org.xBaseJ.*;
4)  import org.xBaseJ.fields.*;
5)  import org.xBaseJ.Util.*;
6)
7)  public class example1 {
8)
9)
10)     public static void main(String args[]){
11)
12)
13)         try{
14)             //
15)             // You must set this unless you want NULL bytes padding out
16)             // character fields.
17)             //
18)             Util.setxBaseJProperty("fieldFilledWithSpaces","true");
19)
20)             //Create a new dbf file
21)             DBF aDB=new DBF("class.dbf",true);
22)
23)             //Create the fields
24)             CharField classId = new CharField("classId",9);
25)             CharField className = new CharField("className",25);
26)             CharField teacherId = new CharField("teacherId",9);
27)             CharField daysMeet = new CharField("daysMeet",7);
28)             CharField timeMeet =new CharField("timeMeet",4);
29)             NumField credits = new NumField("credits",2, 0);
30)             LogicalField UnderGrad = new LogicalField("UnderGrad");
31)
32)
33)             //Add field definitions to database
34)             aDB.addField(classId);
35)             aDB.addField(className);
36)             aDB.addField(teacherId);
37)             aDB.addField(daysMeet);
38)             aDB.addField(timeMeet);
39)             aDB.addField(credits);
40)             aDB.addField(UnderGrad);
41)
42)             aDB.createIndex("classId.ndx","classId",true,true);      // true -
delete ndx, true - unique index,
43)             aDB.createIndex("TchrClass.ndx","teacherID+classId", true, false);
//true - delete NDX, false - unique index,
44)             System.out.println("index created");
45)
46)             classId.put("JAVA10100");
47)             className.put("Introduction to JAVA");
48)             teacherId.put("120120120");
49)             daysMeet.put("NYNYNYN");
50)             timeMeet.put("0800");
51)             credits.put(3);

```

```

52)         UnderGrad.put(true);
53)
54)         aDB.write();
55)
56)         classId.put("JAVA10200");
57)         className.put("Intermediate  JAVA");
58)         teacherId.put("300020000");
59)         daysMeet.put("NYNYNYN");
60)         timeMeet.put("0930");
61)         credits.put(3);
62)         UnderGrad.put(true);
63)
64)         aDB.write();
65)
66)         classId.put("JAVA501");
67)         className.put("JAVA And Abstract Algebra");
68)         teacherId.put("120120120");
69)         daysMeet.put("NNYNYNN");
70)         timeMeet.put("0930");
71)         credits.put(6);
72)         UnderGrad.put(false);
73)
74)         aDB.write();
75)
76)
77)     } catch(Exception e) {
78)         e.printStackTrace();
79)     }
80) }
81) }

roland@logikaldesktop:~/fuelsurcharge2$ javac example1.java
roland@logikaldesktop:~/fuelsurcharge2$ java example1
index created
roland@logikaldesktop:~/fuelsurcharge2$

```

Listing line 18 contains a very important property setting. By default, xBaseJ pads string fields with NULL bytes when writing to disk. While there was a time when this was done, most xBASE environments did away with that practice. As more and more tools became able to open raw data files, it became necessary to supply spaces. Please conduct the following test:

1. Compile and run this program as I have done.
2. Use OpenOffice to open class.dbf in a spreadsheet. Look closely at the data.
3. Comment out listing line 18; compile and re-run this program.
4. Use OpenOffice to open class.dbf in a spreadsheet. Look closely at the data.

What you will notice is that the first spreadsheet had some funky looking zero characters in the text columns. Those characters were the null bytes padding out the character fields. The second version of the file opened more as you expected. It should look much like the following:

	A	B	C	D	E	F	G	H
1	CLASSID,C,9	CLASSNAME,C,25	TEACHERID,C,9	DAYSMEET,C,7	TIME,C,4	CREDITS,C,2	UNDERGRAD,L	
2	JAVA10100	Introduction to JAVA	120120120	NYNYNYN	0800	3	TRUE	
3	JAVA10200	Intermediate JAVA	300020000	NYNYNYN	0930	3	TRUE	
4	JAVA501	JAVA And Abstract	120120120	NNYYNNN	0930	6	FALSE	
5								
6								
7								
8								

Please note column F on the spreadsheet. Even though the numeric database column was declared to have two digits, we don't get a leading zero. Column E (TIME) may seem a bit deceiving at first. This wasn't declared as a numeric database column; it was declared as a character so the leading zero could be forced. Listing line 29 is where CREDITS (column F) is declared, and listing line 28 declares TIMEMEET (column E). Please note that numeric field declarations have two numeric parameters. The first is the size of the field including the punctuation characters, and the second is the number of decimal places.

Listing line 21 is where the initial empty database file is created. The Boolean value "true" as the final parameter forces file creation.

Once you create a field, it has to be added to the database before it becomes a column in the database. We do this at listing lines 34 through 40.

An indexed data file isn't much use unless it has at least one index. Two index files are created at listing lines 42 and 43. The first Boolean value passed into these methods controls deletion of existing files. The second value controls whether the index is a unique key or not. A unique key will allow only one instance of a value to be stored as a key for only one record. A non-unique key will allow the same key value to point to multiple records. You cannot guarantee what order records will be retrieved for records having the same key value. If someone rebuilds the index file, or adds other records in that range, or packs the database, the retrieval order can change. Just because the record for 'FRED SMITH' came out first in this department report run doesn't mean it will come out first next time.

Note: xBASE files do not physically delete records. They flag records as being deleted. The only way to reclaim the wasted space is to create a new version of the database file with a function known as PACK. One of two things would happen depending upon the tool involved:

1. The data file would be walked through sequentially and records not flagged as deleted would be “shuffled up,” replacing deleted or newly emptied records.
2. A new version of the database would be created with a temporary name. This version would contain only the non-deleted records from the original database. Upon completion the original database would be deleted and the new one renamed.

The second approach was much more efficient, but required a lot of disk space. No matter which approach was taken, all index files for the database had to be rebuilt. Until we had MDX files, most libraries and dialects of xBASE had an option which would allow a developer to create an index anew each time they opened the file. xbaseJ has the same option:

```
public NDX(String name,  
            String NDXString,  
            DBF indatabase,  
            boolean destroy,  
            boolean unique) throws xBaseJException, IOException
```

When you pass in a destroy flag of true, xBaseJ rebuilds the index based upon the records currently in the database. Please note that if you do not PACK the database prior to creating a new index, the new index will contain entries for deleted records. When you open an MDX tag, the index is automatically rebuilt in place. We will discuss packing more later.

Please turn back to the screen shot showing this database in an OpenOffice spreadsheet and really look at row one. When this example program was written the programmer used mixed case column names because that looked very Java-like. (It actually looked very PASCALian, and PASCAL is a dead language, so you do the math on that one.) Notice what actually got written to the database, though: it is all upper case. I have found bugs over the years in various tools which end up letting lower case slip into the header record. It is a good programming practice to always use upper case inside of strings which will be used for column or index names. You will never be burned by an upcase() bug if you always pass in upper case.

Take a really good look at listing line 43. That key is composed of two database columns concatenated together. On page 17 of this book you were told that the original dBASE version supported only character data. All ‘h umeric’ values were stored in their character representations to increase portability. This feature also made index creation work. We aren’ adding two values together with that syntax, we are concatenating two strings into one sort/key value.

In truth, that “false” parameter at the end of listing line 43 is of little logical value. Yes, the database will set the key up to allow for duplicate values, but they cannot happen. Listing line 42 has declared a unique key based upon the column CLASSID. If one portion of a string key is required to be unique due to some other constraint, then all values in that key will be unique.

Listing lines 46 through 54 demonstrate how to assign values to the fields and finally write a shiny new record to the database. Because we have the index files open and associated with the data file, all of their keys will be updated. I must confess to being a bit disappointed at the choice of `put()` as the method name for assigning field values. I would have expected `assign()` or `set()`. Depending upon when this method was written, though, `put()` might have been in vogue. There was a dark period of time in the world of Java when an object itself was considered a data store, and when it was thought that one should always “put” into a data store. The Java programmers really just wanted to do something different than the C++ developers who were using `setMyField()`, `assignMyField()`, etc. Of course, GDB used to have a function `DBPutNamedStr()` which wrote a string value into the IO buffer for a named field, so maybe in hindsight I’m just picky.

That’s it. The first example to force creation of a data file along with two index files. Three records are written to the file, and we have verified this by opening the file with OpenOffice. One thing I don’t like about this example is that it didn’t bother to use the `close()` method of the DBF object. While it is true that `close()` is called by the `finalize()` method which is called upon destruction, it is always a good programming practice to close your files before exiting.

1.4 Exception Handling and Example 1

I will assume you are familiar enough with Java to know that statements which can throw an exception traditionally get encapsulated in a try/catch block. Nearly every exception class you will ever encounter is derived from the root Java Exception class. This allows every catch block series to have an ultimate catch-all like the one you see at listing line 77. As far as error handling goes, it doesn't do squat for the user. There is no recovery and they will have no idea what the stack trace means.

The code in the try block is really where the train went off the rails. Yes, I understand the intent was to show only the most straightforward method of creating a new xBASE file with an index. The actual flow will get lost if each statement has its own localized try/catch block, but you need to group things logically.

Those of you unfamiliar with object-oriented error handling won't be familiar with this particular rant. Others may be tired of hearing it, but the newbies need to be educated. The move to more modern languages meant a move away from line numbers and GOTO statements. While this wasn't a bad thing in general, it really waxed error handling. Most programmers didn't completely embrace the "localized error handler" methodology, and without line numbers and RESUME statements the quality of error handling tanked. We have a good example of the typical error handling quality I typically see with Java in this example. If any statement in the range of listing lines 14 through 76 throws an exception, we land in the catch block without any idea of which statement actually threw the exception. Even if we could identify exactly which line threw the exception, we would have no method of getting back there. Java doesn't have a RETRY or RESUME that would allow us to fix a problem then continue on.

Many people will try to characterize code like this as a programmer being lazy, and that would be unfair. The authors here were trying to show how to do something without the error handling getting in the way. The trouble is that most of these examples will be modified only slightly by programmers new to the field, then distributed to others. They don't know any better, and code like this will eventually creep into production systems.

If you want to be even more unfair you can also point out that catching the universal Exception class as is done at listing line 77 is now listed as a bad/undesirable practice by Sun. Lots of code currently in production does this. The problem with doing this is that you mask really hard run-time errors (like a disk failure or bad RAM) which really shouldn't be masked. Not only is there nothing you can do about them in your program, the system manager needs to know about them ASAP!

Part of the desire for a clean and simple source listing came from the early days of programming. Classically trained programmers learned structured analysis and design. More importantly, the first language they learned was BASIC. Later versions of BASIC removed nearly all line numbers from the language. This migration made the language nearly useless. The move to localized error handling with WHEN ERROR IN ... USE ... END WHEN constructs pretty much ruined the language for business use. It all came about because a lot of people trying to learn the language either refused to keep either a printout by their side or two edit windows open.

One of the very first executable lines you would find in most BASIC modules read as follows:

```
99  ON ERROR GOTO 32000          ! old style error handling
```

Other than checking a function return value, no other error handling existed in the source until you got to BASIC line 32000.

```
32000  !;;;;;;;;;;
      !  Old style error handling
      !;;;;;;;;;;

      SELECT ERL
        CASE = 910%
          L_ERR% = ERR
          PRINT "Unable to open input file"; drawing_data$
          PRINT "Error: ";L_ERR%;" ";ERT$( L_ERR%)
          RESUME 929

        CASE = 912%
          L_ERR% = ERR
          PRINT "Unable to open report file "; rpt_file$
          PRINT "Error: ";L_ERR%;" ";ERT$( L_ERR%)
          RESUME 929

        CASE = 930%
          PRINT "Invalid input"
          PRINT "Please re-enter"
          RESUME 930

        CASE = 940%
          L_ERR% = ERR
          PRINT "Unable to retrieve record GE |";BEG_DATE$;"|"
          PRINT "Error: ";L_ERR%;" ";ERT$( L_ERR%)
          RESUME 949

        CASE = 942%
          B_EOF% = 1%

      IF ERR <> 11%
      THEN
        L_ERR% = ERR
        PRINT "Unable to fetch next input record"
        PRINT "Error: ";L_ERR%;" ";ERT$( L_ERR%)
      END IF
```



```

        RESUME 942

        CASE ELSE
        ON ERROR GOTO 0
    END SELECT

32767 ! End of module
PROGRAM_EXIT:

```

I'll be the first to admit that this SELECT statement used to get out of hand. Some programmers refused to use a SELECT so you had an ugly series of nested IF-THEN-ELSE statements. It did, however, leave the logic flow clean and apparent (if you were a competent programmer) and it allowed you to handle just about every error you could potentially recover from. RESUME and RETRY allowed us to return program control to any line number or label in the program. Some abused it, for certain, but the power and grace of this technology is lacking from all OOP error handling today.

Everybody wants the clean look that BASIC with old style error handling had, so most Java programs have no usable error handling.

1.5 rolliel.java

Quite simply, we are going to take example1.java, fix a few things, then add some print functionality. I must stress that this isn't a great example, but it is a very common design. I have encountered this same design time and time again, no matter what language or xBASE library was being used.

rolliel.java

```

1)  import java.io.*;
2)  import java.util.*;
3)  import org.xBaseJ.*;
4)  import org.xBaseJ.fields.*;
5)  import org.xBaseJ.Util.*;
6)
7)  public class rolliel {
8)
9)      //  variables used by the class
10)     //
11)     private DBF aDB = null;
12)     private CharField classId = null;
13)     private CharField className = null;
14)     private CharField teacherId = null;
15)     private CharField daysMeet = null;
16)     private CharField timeMeet = null;
17)     private NumField credits = null;
18)     private LogicalField UnderGrad = null;
19)
20)     private boolean continue_flg = true;
21)
22)     //;;;;;;;;;
23)     //  Main module
24)     //;;;;;;;;;

```

```

25)     public void do_it(){
26)     try{
27)         //
28)         // You must set this unless you want NULL bytes padding out
29)         // character fields.
30)         //
31)         Util.setxBaseJProperty("fieldFilledWithSpaces","true");
32)
33)         open_database();           // use an existing database if possible
34)
35)         if (!continue_flg) {
36)             continue_flg = true;
37)             create_database(); // if none exists create
38)             if (continue_flg)
39)                 add_rows();
40)         } // end test for successful open of existing database
41)
42)         //;;;;
43)         // You cannot just blindly run the report.
44)         // We could have tried to create a database on a full disk
45)         // or encountered some other kind of error
46)         //;;;;
47)         if ( continue_flg) {
48)             dump_records();
49)             dump_records_by_primary();
50)             dump_records_by_secondary();
51)             aDB.close();
52)         } // end test for open database
53)
54)     }catch(IOException i){
55)         i.printStackTrace();
56)     } // end catch
57) } // end do_it
58)
59) //;;;;;;;;;;
60) // method to add some rows
61) //
62) // Notice that I added the rows in reverse order so we could
63) // tell if the unique index worked
64) //;;;;;;;;;;
65) private void add_rows() {
66)     try {
67)         classId.put("JAVA501");
68)         className.put("JAVA And Abstract Algebra");
69)         teacherId.put("120120120");
70)         daysMeet.put("NNYNNYNN");
71)         timeMeet.put("0930");
72)         credits.put(6);
73)         UnderGrad.put(false);
74)
75)         aDB.write();
76)
77)
78)         classId.put("JAVA10200");
79)         className.put("Intermediate JAVA");
80)         teacherId.put("300020000");
81)         daysMeet.put("NYNNYNN");
82)         timeMeet.put("0930");
83)         credits.put(3);
84)         UnderGrad.put(true);
85)
86)         aDB.write();
87)

```

```

88)         classId.put("JAVA10100");
89)         className.put("Introduction to JAVA");
90)         teacherId.put("120120120");
91)         daysMeet.put("NYNYNYN");
92)         timeMeet.put("0800");
93)         credits.put(3);
94)         UnderGrad.put(true);
95)
96)         aDB.write();
97)
98)         } catch( xBaseJException j){
99)             j.printStackTrace();
100)            continue_flg = false;
101)        } // end catch xBaseJException
102)        catch( IOException i){
103)            i.printStackTrace();
104)        } // end catch IOException
105)    } // end add_rows method
106)
107)    //;;;;;;;;;;
108)    //    Method to create a shiny new database
109)    //;;;;;;;;;;
110)    private void create_database() {
111)        try {
112)            //Create a new dbf file
113)            aDB=new DBF("class.dbf",true);
114)
115)            attach_fields(true);
116)
117)            aDB.createIndex("classId.ndx","classId",true,true);    // true -
delete ndx, true - unique index,
118)            aDB.createIndex("TchrClass.ndx","teacherID+classId", true, false);
//true - delete NDX, false - unique index,
119)            System.out.println("created database and index files");
120)
121)        } catch( xBaseJException j){
122)            j.printStackTrace();
123)            continue_flg = false;
124)        } // end catch
125)        catch( IOException i){
126)            i.printStackTrace();
127)        } // end catch IOException
128)    } // end create_database method
129)
130)    //;;;;;;;;;;
131)    //    Method to open an existing database and attach primary key
132)    //;;;;;;;;;;
133)    public void open_database() {
134)        try {
135)            //Create a new dbf file
136)            aDB=new DBF("class.dbf");
137)
138)            attach_fields( false);
139)
140)            aDB.useIndex("classId.ndx");
141)            System.out.println("opened database and primary index");
142)        } catch( xBaseJException j){
143)            continue_flg = false;
144)        } // end catch
145)        catch( IOException i){
146)            continue_flg = false;
147)        } // end catch IOException
148)    } // end open_database method

```

```

149)
150) //;;;;;;;;;;
151) //      Method to populate known class level field objects.
152) //      This was split out into its own method so it could be used
153) //      by either the open or the create.
154) //;;;;;;;;;;
155) private void attach_fields( boolean created_flg) {
156)     try {
157)         if ( created_flg) {
158)             //Create the fields
159)             classId      = new CharField("classId",9);
160)             className    = new CharField("className",25);
161)             teacherId    = new CharField("teacherId",9);
162)             daysMeet     = new CharField("daysMeet",7);
163)             timeMeet     = new CharField("timeMeet",4);
164)             credits      = new NumField("credits",2, 0);
165)             UnderGrad    = new LogicalField("UnderGrad");
166)
167)             //Add field definitions to database
168)             aDB.addField(classId);
169)             aDB.addField(className);
170)             aDB.addField(teacherId);
171)             aDB.addField(daysMeet);
172)             aDB.addField(timeMeet);
173)             aDB.addField(credits);
174)             aDB.addField(UnderGrad);
175)
176)         } else {
177)             classId      = (CharField) aDB.getField("classId");
178)             className    = (CharField) aDB.getField("className");
179)             teacherId    = (CharField) aDB.getField("teacherId");
180)             daysMeet     = (CharField) aDB.getField("daysMeet");
181)             timeMeet     = (CharField) aDB.getField("timeMeet");
182)             credits      = (NumField) aDB.getField("credits");
183)             UnderGrad    = (LogicalField) aDB.getField("UnderGrad");
184)         }
185)
186)     } catch ( xBaseJException j){
187)         j.printStackTrace();
188)     } // end catch
189)     catch( IOException i){
190)         i.printStackTrace();
191)     } // end catch IOException
192) } // end attach_fields method
193)
194)
195) //;;;;;;;;;;
196) //      Method to test private flag
197) //;;;;;;;;;;
198) public boolean ok_to_continue() {
199)     return continue_flg;
200) } // end ok_to_continue method
201)
202) //;;;;;;;;;;
203) //      Method to dump records by record number
204) //;;;;;;;;;;
205) public void dump_records() {
206)     System.out.println( "\n\nRecords in the order they were entered\n");
207)     System.out.println( "classId      className                " +
208)         "teacherId daysMeet time cr UnderGrad");
209)
210)     for (int x=1; x <= aDB.getRecordCount(); x++) {
211)         try {

```

```
212)         aDB.gotoRecord( x);
213)     }
214)     catch( xBaseJException j){
215)         j.printStackTrace();
216)     } // end catch IOException
217)     catch( IOException i){
218)         i.printStackTrace();
219)     } // end catch IOException
220)
221)     System.out.println( classId.get() + " " + className.get() +
222)         " " + teacherId.get() + " " + daysMeet.get() + " " +
223)         timeMeet.get() + " " + credits.get() + " " +
224)         UnderGrad.get());
225) } // end for x loop
226) } // end dump_records method
227)
228) //;;;;;;;;;;
229) // Method to dump records via primary key
230) //;;;;;;;;;;
231) public void dump_records_by_primary() {
232)     System.out.println( "\n\nRecords in primary key order\n");
233)     System.out.println( "classId      className      " +
234)         "teacherId daysMeet time cr UnderGrad");
235)
236)     try {
237)         aDB.useIndex("classId.ndx");
238)         continue_flg = true;
239)         aDB.startTop();
240)
241)         while( continue_flg) {
242)             aDB.findNext();
243)
244)             System.out.println( classId.get() + " " +
245)                 className.get() + " " +
246)                 teacherId.get() + " " +
247)                 daysMeet.get() + " " +
248)                 timeMeet.get() + " " +
249)                 credits.get() + " " +
250)                 UnderGrad.get());
251)
252)         } // end while loop
253)     }
254)     catch( xBaseJException j) {
255)         continue_flg = false;
256)     }
257)     catch( IOException i) {
258)         continue_flg = false;
259)     }
260)
261) } // end dump_records_by_primary method
262)
263) //;;;;;;;;;;
264) // Method to dump records off by secondary key
265) //;;;;;;;;;;
266) public void dump_records_by_secondary() {
267)     System.out.println( "\n\nRecords in secondary key order\n");
268)     System.out.println( "classId      className      " +
269)         "teacherId daysMeet time cr UnderGrad");
270)
271)     try {
272)         aDB.useIndex("TchrClass.ndx");
273)         continue_flg = true;
274)
```

```

275)         aDB.startTop();
276)
277)         while( continue_flg) {
278)             aDB.findNext();
279)
280)             System.out.println( classId.get() + " " +
281)                                 className.get() + " " +
282)                                 teacherId.get() + " " +
283)                                 daysMeet.get() + " " +
284)                                 timeMeet.get() + " " +
285)                                 credits.get() + " " +
286)                                 UnderGrad.get());
287)
288)         } // end while loop
289)     }
290)     catch( xBaseJException j) {
291)         continue_flg = false;
292)     }
293)     catch( IOException i) {
294)         continue_flg = false;
295)     }
296)
297) } // end dump_records_by_secondary method
298) } // end class rollie1

```

The first thing you will notice about this example is that I ripped out the main() method. Most people writing Java examples try to get by with a single source file example, even when they are using a complex library or database system. I'm nowhere near "One With the Object" level of OOP with this design, but it is typical of things you will encounter in the field.

This design works when you have created a single file database which is to be used by one and only one application. This design fails as soon as you need to use that same database in another application. When you enter a shop that had a programmer who liked this design, you will usually be entering *after* that programmer has left (or was asked to leave). When you need to add additional functionality you either have to cut and paste large chunks of code out of this class into a new one, or you watch this class grow to be hundreds of thousands of lines of source.

One of the many things I don't like about Java is its lack of header files. Most Java developers end up using some kind of IDE like Eclipse, not because it's a good editor, but because it has built-in Java-specific functionality which will create views of all the methods and members in a class if you load the correct plug-in. In C++ we had header files in which the class was prototyped and you could easily see all of its methods and members. This source file is just shy of 300 lines in length, and if I didn't prefix my methods with a comment containing ten ";" characters you would have trouble locating them. Imagine what it is like when the listing is 12,000 lines long.

All instances of the database and column names are moved out to the class level in this class. Doing so allows them to be shared by all methods in the class. I flagged them as private so others couldn't touch them from outside the class.

Listing line 25 is where the public method `do_it()` begins. This is really the whole application. The flow would be a little bit easier to read if we didn't have to keep checking the `continue_flg` variable, or if Java allowed statement modifiers like DEC BASIC did:

```
GOSUB C2000_PAGE_HEADING      IF LINE_CNT% >= page_size%
```

A lot of people complained about statement modifiers, but those people never wrote production systems. Eventually, BASIC became the only surviving commercial language to have this syntax. The flow of this particular method would clean up considerably if we could use such syntax.

Even with the cumbersome if statements, you should be able to ascertain the flow of the method. First we try to use an existing database. If that fails, we create the database. If database creation was successful, we add some data to the database. Once we have successfully established a database, we report off the data in three different sort orders, close the database, and exit.

Please take notice of listing lines 67, 78, and 88. These lines assign the primary key values to each row that we will be adding. What you need to notice is that I stored these records in descending order by primary key. Having data which was added in a known sort order is critical to understanding whether our reports worked correctly or not.

Both `create_database()` and `open_database()` call a method named `attach_fields()`. We have very little to discuss in the `create_database()` method since much of the code was stolen from `example1.java`. You will notice that in `open_database()` we don't provide the "true" parameter to the DBF constructor. It is this parameter which tells the DBF constructor whether to use an existing database or create a new one.

Notice at listing line 140 that we don't create an index, but rather use the existing index file. Using an existing index file can be an incredibly dangerous thing to do when working with xBASE files. Attempting to create a shiny new index file using the same hard-coded name as last time can also be a dangerous thing as another user may have the file opened, which means your process will fail. During the dark days of DOS it was almost impossible to generate a unique file name every time. The 8.3 naming schema was pretty restrictive. Not many of your disk partitions will be FAT16 these days, though. FAT32 came onto the scene in 1996 with Windows 95 OSR2. Floppy disk drives will still use FAT16, but most of the "super floppy" disks (120 and 240Meg) will use FAT32 or something else, which allows for very long file names.

In the DOS days, most xBASE libraries didn't have a `reindex()` function. They were all busy trying to be multi-user and there simply wasn't a good multi-user method of rebuilding an index while other users had the file open. (There really isn't even today.) We also didn't have a universal temporary directory. There were some environment variables you could hope were set (TMP, TEMP, etc.), but all in all, you were on your own.

Few things would cause more problems in xBASE software than one programmer forgetting to open the "production" index when they added records to the database. Any application which used the production index to access records would simply skip processing any records in the database which didn't have an index.

In a feat of purely defensive coding, most programmers would take a stab at generating a unique file name for the index, then create the needed index after they opened the database. When you had thousands of records on those old and slow 40Meg hard drives, it could take minutes for the first screen to load, but at least you knew you were processing all of the data...or did you? Nobody else knew about your shiny new indexed file. This means they weren't bothering to update any entries in it while they were adding records to the database. The lack of a common OS-enforced temporary directory led to a lot of policies and procedures concerning what files to delete when. More than one shop blew away their production index while trying to delete temporary index files to free up space.

Some shops learned to live with and work around the pitfalls. They put policies and procedures in place so users didn't have to wait entire minutes for the first application screen to display data. The world of xBASE eventually created the MDX file in an attempt to solve these issues. We will discuss the MDX file in a later example.

Listing lines 159 through 183 show a bit of difference between creating a new file and using an existing file. When the database is shiny and new, you must create the column objects, then add them to the database object. The act of adding them to the object actually creates the columns in the database. When you are using an existing database you must pull the field definitions out of the database object. If you create field definitions and attempt to add them, they will be new columns, unless they have a matching column name already in the database, then an exception will be thrown.

One thing I would have liked to seen in the library was a series of "get" methods, one for each supported data type. This would move any casting inside of a class method. Many of the C/C++ libraries I used over the years had this functionality to keep code as cast-free as possible. It would be nice to call a method named `aDB.getCharField("classId")` and have it either return a `CharField` object or throw an exception. Of course, it would also be nice if the exception could

have actual error codes which told you what the exception was, not just that it happened to have died.

The `dump_records()` method starting on listing line 205 doesn't have much complexity to it. I use a simple for loop to read from 1 to the maximum number of records in the database, printing each record out. The method `getRecordCount()` returns the current record count in the database.

The method `gotoRecord()` physically reads that record number from the database. You may recall that I told you xBASE is a relative file format. All relative file formats are actually accessed by record number. The index files are really storing a key value and corresponding record number in a Btree (binary tree) fashion. This method walks through the records *as they were written to the data file* without paying any attention to key values.

At listing line 239, I show you how to clear the “current record” value stored internally in the class. The method `startTop()` will set the current record value to zero and move the index pointer back to the root of the currently active index.

Most of you would have tried to use `read()` instead of `findNext()` at listing line 241. I will admit that once I read the comments in the source file, I gave it a whirl as well. It behaved the way I thought it would. Any of you who have read “The Minimum You Need to Know to Be an OpenVMS Application Developer” ISBN-13 978-0-9770866-0-3 would have expected it to not work as well. There is a problem with most “read” and “readNext” type functions in most languages. *You must first establish a key of reference* via some other IO operation before an ordinary read or readNext type function will work. Find and findNext type methods are almost always set up to find a key value “equal to or greater than” the value they currently have in some designated key buffer. If that buffer is null, they tend to find the first record in the file via the currently active index.

Please note: The technique I've shown you here will work with xBaseJ and its dBASE implementations. `findNext()` does not look at a key value, only the position of the index tree being traversed in memory. `find()` actually attempts to locate a value based on key. Some libraries have stored some numeric keys as binary integers. On most platforms an integer zero is a null value in binary integer form. This null value is greater than a negative value due to the way the sign bit is treated. You get lucky with many IEEE standards since there is usually at least one bit set to indicate the numeric base or some other aspect.

Our method `dump_records_by_primary()` has to specify the primary to control sort order. If you rely on some other logic path to set the key, then your sort order might appear random. Other than the heading and the changing of the index there really is no difference between `dump_records_by_secondary()` and `dump_records_by_primary()`.

Notice in each of the report methods that we have to call the `get()` method for each field in order to obtain its value. We do not have direct access to the data values in this library. Some others allow for direct retrieval and some don't. I don't really have a preference these days. During my DOS programming days I always wanted to use C libraries, which allowed direct access to the values. This wasn't because I was an Uber geek trying to be one with the CPU, but because of the wonderful 640K memory limitation of the day. If I allocated the storage for the returned values, I could put it in an EMS page which could be swapped out on demand. Most vendors of third-party libraries refused to provide any support if you were swapping their code in and out of the lower 640K via an overlay linker.

Compiling and running this thing isn't a big challenge, assuming you've already got your CLASSPATH environment variable set.

```
roland@logikaldesktop:~/fuelsurcharge2$ rm class.dbf
roland@logikaldesktop:~/fuelsurcharge2$ rm teacher.dbf
roland@logikaldesktop:~/fuelsurcharge2$ java testRolliel
created database and index files
```

Records in the order they were entered

classId	className	teacherId	daysMeet	time	cr	UnderGrad
JAVA501	JAVA And Abstract Algebra	120120120	NNYNNYN	0930	6	F
JAVA10200	Intermediate JAVA	300020000	NYNNYN	0930	3	T
JAVA10100	Introduction to JAVA	120120120	NYNNYN	0800	3	T

Records in primary key order

classId	className	teacherId	daysMeet	time	cr	UnderGrad
JAVA10100	Introduction to JAVA	120120120	NYNNYN	0800	3	T
JAVA10200	Intermediate JAVA	300020000	NYNNYN	0930	3	T
JAVA501	JAVA And Abstract Algebra	120120120	NNYNNYN	0930	6	F

Records in secondary key order

classId	className	teacherId	daysMeet	time	cr	UnderGrad
JAVA10100	Introduction to JAVA	120120120	NYNNYN	0800	3	T
JAVA501	JAVA And Abstract Algebra	120120120	NNYNNYN	0930	6	F
JAVA10200	Intermediate JAVA	300020000	NYNNYN	0930	3	T

I deleted the existing data file and index by hand so you could see the result of a first run situation. You will also want to do this if you have compiled and run the `example1.java` program. This particular set of test data is re-ordered. If you run it against the original data file, you won't see any differences between the first and the second report.

Just to be complete, let me show you the simple little test source.

testRollie1.java

```
1) public class testRollie1 {
2)
3)     public static void main(String args[]){
4)         rollie1 r = new rollie1();
5)
6)         r.do_it();
7)
8)     } // end main method
9)
10) } // end class testRollie1
```

1.6 Programming Assignment 1

Modify rollie1.java to remove the opening of the primary key file when opening the existing database. Replace `dump_records_by_primary()` and `dump_records_by_secondary()` with one method `dump_records_by_key()` which accepts a String parameter that is the index file name. Compile and run your program. Test it with both a valid file name and a nonexistent file name.

1.7 Size Matters

I know, that section heading makes it sound like I' mgoing to be selling gym equipment or male enhancement tablets, but it really is true with xBaseJ: size really does matter. It is your job to ensure your application doesn' toverrun a numeric field. Character fields will throw an exception, but numeric fields will not.

example5.java

```
1) import java.io.*;
2) import java.util.*;
3) import org.xBaseJ.*;
4) import org.xBaseJ.fields.*;
5) import org.xBaseJ.Util.*;
6)
7) public class example5 {
8)
9)
10)     public static void main(String args[]){
11)
12)
13)         try{
14)             //
15)             // You must set this unless you want NULL bytes padding out
16)             // character fields.
17)             //
18)             Util.setXBaseJProperty("fieldFilledWithSpaces","true");
19)
20)             //Create a new dbf file
21)             DBF aDB=new DBF("roi.dbf",true);
22)
23)             //Create the fields
24)             NumField pctrtn = new NumField("pctrtn",6, 3);
25)             CharField fundnm = new CharField("fundnm",20);
26)             NumField invstamt = new NumField("invstamt", 15,2);
```

```

27)
28)
29)         //Add field definitions to database
30)         aDB.addField(pctrtn);
31)         aDB.addField(fundnm);
32)         aDB.addField(invstamt);
33)
34)         aDB.createIndex("roik0.ndx","pctrtn",true,true);        // true -
delete ndx, true - unique index,
35)         System.out.println("\nindex created ... now adding records");
36)
37)         fundnm.put("LargeCap");
38)         pctrtn.put(-4.5);
39)         invstamt.put(550000);
40)         aDB.write();
41)
42)         fundnm.put("MidCap");
43)         pctrtn.put(2.3);
44)         invstamt.put(120000);
45)         aDB.write();
46)
47)         fundnm.put("Growth");
48)         pctrtn.put(3.4);
49)         invstamt.put(45000000);
50)         aDB.write();
51)
52)         fundnm.put("SmallCap");
53)         pctrtn.put(-6.2);
54)         invstamt.put(23000000000.0);
55)         aDB.write();
56)
57)         fundnm.put("Spyder");
58)         pctrtn.put(2);
59)         invstamt.put(78923425);
60)         aDB.write();
61)
62)         fundnm.put("PennyStk");
63)         pctrtn.put(26.5);
64)         invstamt.put(888000);
65)         aDB.write();
66)
67)         fundnm.put("BioTech");
68)         pctrtn.put(-34.6);
69)         invstamt.put(345567.89);
70)         aDB.write();
71)
72)         System.out.println("Records added\n");
73)         System.out.println("ROI      Fund      Amount");
74)         System.out.println("-----  -----  -----");
75)
76)         aDB.startTop();
77)         for( int i=0; i < aDB.getRecordCount(); i++)
78)         {
79)             aDB.findNext();
80)             System.out.println( pctrtn.get() + "      " + fundnm.get() +
81)                 invstamt.get());
82)         }
83)
84)     }catch(Exception e){
85)         e.printStackTrace();
86)     }
87) }
88) }

```

Notice at listing line 24 that I declare `pctrtn` to be six long with three decimal places. I then go ahead and make this an index for the data file. At listing line 68 I assign the value -34.6 to the field. It seems innocent enough, doesn't it? Let's take a look at what happens.

```
roland@logikaldesktop:~/fuelsurcharge2$ javac example5.java
roland@logikaldesktop:~/fuelsurcharge2$ java example5
```

```
index created ... now adding records
Records added
```

ROI	Fund	Amount
-6.200	SmallCap	23000000000.00
-4.600	BioTech	345567.89
-4.500	LargeCap	550000.00
2.000	Spyder	78923425.00
2.300	MidCap	120000.00
3.400	Growth	45000000.00
26.500	PennyStk	888000.00

Take a look at where our BioTech record ended up. That's not the value we assigned, is it? There was no exception thrown when we ran the program; we simply got the wrong value stored.

1.8 Programming Assignment 2

Modify `example5.java` by expanding the size of the ROI column and try to add a record with a fund name greater than 20 characters.

1.9 Examining a DBF

From the 1960s through much of the 1980s, software vendors tried to lock people into their product lines in a variety of ways. One of the most tried and true methods was to create your own proprietary data file format. Even if you used the indexed file system provided by the computer operating system, as long as you didn't tough up the record layouts, your customers couldn't access their data without using your software and/or buying additional services from you. Given that MBAs are creatures who go to school to have both their ethics and soul removed so they can run a business in the most profitable method possible, the fees kept going up and the lawyers kept getting richer over breach of contract lawsuits.

Ashton Tate certainly tried to go that route with dBASE, but there was a lot of existing technology out there for people to work with. The lawyers and the attitude continued to turn all potential new customers against Ashton Tate and the other xBASE platforms gained ground. Ultimately, there were quite a few things that did Ashton Tate in. First off, all of the xBASE file formats stored the file layout information in the header. All you had to do was figure out how to parse the header and you could get to most of the data. Second, Vulcan, the original version of

what became dBASE II, wasn't released as a commercial product. We didn't have the Internet back then, but we had BBS networks which participated in echo relays and gave access credit for file uploads. Once Vulcan made it to a couple of the larger boards, it was everywhere in under a month. This gave nearly every competing product the same starting point.

Given the memory restrictions of the day, Ashton Tate and others didn't have the option of hiding all of the information in some encrypted format and requiring an engine to be running like MySQL, Oracle, or any of the other database engines of today. The Jet Propulsion Laboratory wasn't in the business of putting out commercial software. They simply had a severe need to store data in some indexed format for reporting purposes. A need so severe that someone was allowed to take however much time it took them to solve the problem. They chose to solve the problem in the most easily supportable means available to them at the time.

If you aren't long in the tooth like myself, you probably don't understand just how easy it is to support the xBASE format. Our next example should give you some idea.

showMe.java

```

1)  import java.io.*;
2)  import java.util.*;
3)  import java.text.*;
4)  import org.xBaseJ.*;
5)  import org.xBaseJ.fields.*;
6)  import org.xBaseJ.Util.*;
7)
8)  public class showMe {
9)
10)     public static final int MAX_NAME_LEN = 11;
11)
12)     // variables used by the class
13)     //
14)     private DBF aDB = null;
15)
16)     private boolean continue_flg = true;
17)
18)     //;;;;;;;;;;
19)     // Main module
20)     //;;;;;;;;;;
21)     public void showDBF( String _dbfName){
22)         try{
23)             aDB = new DBF( _dbfName);
24)         } catch( xBaseJException j){
25)             System.out.println( "Unable to open " + _dbfName);
26)         } // end catch xBaseJException
27)         catch( IOException i){
28)             System.out.println( "Unable to open " + _dbfName);
29)         } // end catch IOException
30)
31)         System.out.println( "\n" + _dbfName + " has:");
32)         System.out.println( "    " + aDB.getRecordCount() + " records");
33)         System.out.println( "    " + aDB.getFieldCount() + " fields\n");
34)         System.out.println( "                                FIELDS");
35)         System.out.println( "Name                                " +
36)             "Type Length Decimals");
37)         System.out.println( "-----" +

```

```

38)         "-----");
39)
40)     StringBuilder sb = new StringBuilder();
41)     Formatter r = new Formatter( sb, Locale.US);
42)
43)     for( int i=1; i <= aDB.getFieldCount(); i++) {
44)         try {
45)             Field f = aDB.getField(i);
46)             r.format( " %-25s      %1c      %4d      %4d\n",
47)                     f.getName(),
48)                     f.getType(),
49)                     f.getLength(),
50)                     f.getDecimalPositionCount());
51)         } catch( xBaseJException x) {
52)             System.out.println( "Error obtaining field info");
53)         }
54)
55)     } // end for loop
56)
57)     System.out.println( r.toString());
58)
59)     try {
60)         aDB.close();
61)     } catch( IOException o) {}
62)
63) } // end showDBF method
64)
65) //;;;;;;;;;;
66) // Method to dump all records in database
67) //;;;;;;;;;;
68) public void dump_records( String _dbfName) {
69)     dump_records( _dbfName, -1);
70) } // end dump_records method
71)
72) //;;;;;;;;;;
73) // Method to dump first N records from database.
74) //;;;;;;;;;;
75) public void dump_records( String _dbfName, int _reccount) {
76)     int the_count = 0;
77)
78)     if (_reccount < 1) {
79)         try{
80)             aDB = new DBF( _dbfName);
81)             the_count = aDB.getRecordCount();
82)             aDB.close();
83)         } catch( xBaseJException j){
84)             System.out.println( "Unable to open " + _dbfName);
85)         } // end catch xBaseJException
86)         catch( IOException i){
87)             System.out.println( "Unable to open " + _dbfName);
88)         } // end catch IOException
89)     } else {
90)         the_count = _reccount;
91)     } // end test for negative _reccount parameter
92)
93)     dump_records( _dbfName, 1, the_count);
94) } // end dump_records method
95)
96) //;;;;;;;;;;
97) // Method to dump a range of records from start to end.
98) //;;;;;;;;;;
99) public void dump_records( String _dbfName, int _startRec, int _endRec) {
100)     int l_x=0;

```

```

101)         int curr_width=0;
102)         StringBuilder sb = new StringBuilder();
103)         Formatter r = new Formatter( sb, Locale.US);
104)
105)         try {
106)             aDB = new DBF( _dbfName);
107)         } catch( xBaseJException j){
108)             System.out.println( "Unable to open " + _dbfName);
109)         } // end catch xBaseJException
110)         catch( IOException i){
111)             System.out.println( "Unable to open " + _dbfName);
112)         } // end catch IOException
113)
114)         try {
115)             int field_count = aDB.getFieldCount();
116)             int heading_length = 0;
117)             String dash_line = "";
118)
119)             for (int i=1; i <= field_count; i++) {
120)                 int fld_width = MAX_NAME_LEN;
121)                 int x;
122)
123)                 Field f = aDB.getField(i);
124)                 String namStr = f.getName();
125)                 x = (fld_width > f.getLength()) ? fld_width : f.getLength();
126)                 String s8 = "%-" + x + "s ";
127)                 r.format( s8, namStr);
128)                 //
129)                 // I have never understood how Java could be declared
130)                 // so advanced by so many and the language not
131)                 // include something as fundamental as the STRING$( )
132)                 // function from BASIC to generate an N length string
133)                 // of some character.
134)                 //
135)                 char[] dl = new char[ x];
136)                 Arrays.fill( dl, '-');
137)                 dash_line += new String(dl) + " ";
138)
139)             } // end for loop to print headings
140)
141)             System.out.println( r.toString());
142)             System.out.println( dash_line);
143)
144)             for (l_x=_startRec; l_x <= _endRec; l_x++) {
145)                 if (sb.length() > 0)
146)                 {
147)                     sb.delete(0, sb.length()); // nuke output buffer
148)                 }
149)
150)                 aDB.gotoRecord( l_x);
151)
152)                 for (int j=1; j <= field_count; j++) {
153)                     Field f = aDB.getField(j);
154)                     switch (f.getType()) {
155)                         case 'C':
156)                             CharField c = (CharField) f;
157)                             curr_width = (MAX_NAME_LEN > c.getLength()) ?
158)                                 MAX_NAME_LEN : c.getLength();
159)                             String s = "%-" + curr_width + "s ";
160)                             r.format( s, c.get());
161)                             break;
162)
163)                         case 'D':

```



```

164)             DateField d = (DateField) f;
165)             r.format( "%8s ", d.get());
166)             break;
167)
168)         case 'F':
169)             FloatField o = (FloatField) f;
170)             curr_width = (MAX_NAME_LEN > o.getLength()) ?
171)                 MAX_NAME_LEN : o.getLength();
172)             String s6 = "%" + curr_width + "s ";
173)             r.format( s6, o.get());
174)             break;
175)         case 'L':
176)             LogicalField l = (LogicalField) f;
177)             curr_width = MAX_NAME_LEN;
178)             String s1 = "%" + curr_width + "s ";
179)             r.format( s1, l.get());
180)             break;
181)
182)         case 'M':             // we don't actually go get the memo
183)                               // just print the id for it.
184)             MemoField m = (MemoField) f;
185)             r.format( "%10s ", m.get());
186)             break;
187)
188)         case 'N':
189)             NumField n = (NumField) f;
190)             curr_width = (MAX_NAME_LEN > n.getLength()) ?
191)                 MAX_NAME_LEN : n.getLength();
192)             String s2 = "%" + curr_width + "s ";
193)             r.format( s2, n.get());
194)             break;
195)
196)         case 'P':
197)             PictureField p = (PictureField) f;
198)             curr_width = (MAX_NAME_LEN > p.getLength()) ?
199)                 MAX_NAME_LEN : p.getLength();
200)             String s3 = "%" + curr_width + "s ";
201)             r.format( s3, p.get());
202)             break;
203)
204)         default:
205)             r.format("?");
206)         } // end type switch
207)     } // end inner for loop to print each field
208)     System.out.println( r.toString());
209) } // end for loop to write detail
210)
211)     aDB.close();
212) } catch( xBaseJException j){
213)     System.out.println( "Error processing record ");
214) } // end catch xBaseJException
215) catch( IOException i){
216)     System.out.println( "Unable to open " + _dbfName);
217) } // end catch IOException
218)
219) } // end dump_records method
220) } // end showMe class

```

In case some of you don't really know anything about Java, let me point out that listing line 10 is how you declare a class constant. This constant can be referenced from any application even if users don't have an instance of this class, as long as they have imported the class file. To access it they would simply need to type `showMe.MAX_NAME_LEN` in any source file which imports the `showMe` class. The maximum name allowed by early xBASE implementations was ten characters; a value of eleven allows for a trailing space.

Listing lines 22 through 29 were an attempt to show you localized error handling. It does make the code ugly. If `xBaseJException` had a constructor which allowed for an integer parameter, or better yet, an enum type, we wouldn't have to perform localized handling just to trap for an open error. If all of the code is in one big try/catch block, we have no method of figuring out exactly where the exception occurred. Yes, you can print the stack trace, but if doing so only releases a JAR file containing your application, what good is that to the user?

One of the first things that should catch your attention is listing lines 32 and 33. The header record of a DBF actually keeps track of both the record count and the field count. We use the methods provided by the class to access these values.

I need to get on my high horse a minute about `StringBuilder` and `Formatter`. Up until `Formatter` was added to the Java language, it was completely unusable for business purposes. I have been lambasted by many a useless PhD for pointing out that Java was absolutely useless in the business world because it was physically incapable of producing a columnar report. Every language those never-worked-in-the-real-world academics put down as being inferior to Java because they preceded it could, and did, produce columnar reports. You see these reports every time you get your credit card statement, phone bill, etc. Business cannot function without the capability to produce columnar reports.

The C language gave us a format string very early on. It evolved over the years to become quite a dynamic thing. Listing line 46 shows you an example what we ended up with in Java. It resembles the C format string in many ways, but fails in one critical way. The C format string allowed a developer to use something like the following:

```
printf( "%-*s\n", curr_width, "some kind of day");
```

The hyphen, "-", forced the left justification as it does in Java. The asterisk, "*", told the formatting logic to take the next parameter from the last and use it as the WIDTH of the field. This allowed a programmer to do cool things like centering a heading on a page. It was very common to see things like the following:

```
printf( "%*s\n", 66-(strlen(heading_str)/2), " ", heading_str);
```

Most business reports printed on 132-column greenbar paper. If you subtracted half the length of the string from the midpoint of the line that told you roughly how many spaces to print in front of the string.

Java isn't quite so progressive. Listing lines 153 through 156 will show you an example of the hack I had to make to work around this failure. It isn't pretty, but I had to build a dynamic String which contained the actual width value and pass that in as the first parameter.

Notice all of the information the Field class contains. There are actually many more methods; I simply focused on the ones we would need. Prior versions of xBaseJ will not return the correct result from `f.getType()`. Field is a base class. Thankfully we can instantiate it. Because it is a base class, it has no knowledge of the classes which were derived from it. Java has RTTI (Run Time Type Identification) which keeps track of who is what to who, but the class Field doesn't know. If you try to call the `getType()` method of the Field class in an older version of xBaseJ, it will toss an exception. I turned in a modification which allows it to return either the correct result or ' ' to indicate an un-initialized value.

Older versions of xBaseJ have Field.java containing the following:

```
/**
 * @return char field type
 * @throws xBaseJException
 *          undefined field type
 */
public char getType() throws xBaseJException
{
    if (true)
        throw new xBaseJException("Undefined field");
    return ' ';
}
```

Every class derived from it ends up having code like this:

```
/**
 * return the character 'D' indicating a date field
 */
public char getType()
{
    return 'D';
}
```

Field.java now has the code below.

```
/**
 * @return char field type
 */
public abstract char getType();
```

This forces a Field instance pointer to use the `getType()` method provided by the derived class. It's a win all around.

You will notice that we have multiple `dump_records()` methods. All of them taking different parameters. This is an example of polymorphism. Actually I was forced into it because Java isn't polite enough to allow default argument values, as does C++. The first method will dump all of the records in the database, the second will dump only the first N records in the database, and the last will dump records X through Y from the database.

Of course, in order to dump the records, one has to know what each column is and how wide the column will be upon output. The for loop at listing lines 148 through 168 takes care of displaying the column headings. The calculation of `x` is an attempt to center the heading over the data column.

Take a look at listing lines 135 through 137. When you program in BASIC on real computers you make a string of a single character by calling `STRING$(len, ascii_value)`. If you want to create a string of 30 hyphens you would type the following:

```
STRING$( 30%, ASCII("-"))
```

The 30% could of course be an integer variable like `X%`. The String class for Java comes up short in this area. In the world of C programming we used to declare a generic buffer at the start of a module, then do the three-step shuffle:

```
char    work_str[1024];

memset( work_str, '\0', sizeof( work_str));
memset( work_str, '-', l_x);
strcat( output_str, work_str);
```

C used null-terminated strings. The first step nulled out the buffer. The second step put just the right number of characters into the buffer and the third step added the buffer to the destination string. It isn't that much different than what we were forced to do in Java. Here we had the restriction that Java doesn't use null-terminated char arrays for strings. The expedient method was to dynamically declare the char array each pass through the for loop. The `fill()` method of the Arrays class allows us to set every element to a specific value. Finally we can add the hyphen string along with some spaces to the string which will actually be printed.

Listing lines 152 through 206 get just a little ugly. The code isn't complex, it simply got ugly trying to fit. I had to deal with page margin issues when writing this, hence the use of the conditional operator `?:`. If you have read the other books in this series you will know that I am definitely NOT a fan of this operator. It makes code difficult to read, especially for a novice. It is really shorthand for "if () then otherwise." If the expression in the parenthesis evaluates to true, then you return the value between the `?` and the `:`, otherwise, you return the value following the `:`. It is true that I could have replaced each of those lines with the following:

```

if (MAX_NAME_LEN > p.getLength())
    curr_width = MAX_NAME_LEN
else
    curr_width = p.getLength();

```

None of you really wanted me to use “if” statements inside of switch cases, though if you work in the real world you will see them quite often. As a general rule, you are not in trouble until your nesting gets more than 3 levels deep.

The display logic for each column isn't complex, but might require a bit of explanation. MAX_NAME_LEN is defined to be eleven because ten used to be the maximum length for a column name under most xBASE flavors, and eleven ensures we will have at least one trailing space. When displaying a data column, I want this example to be at least wide enough to display the name. (One thing which really annoys me about most spreadsheet applications is they size columns to the data size, even when the column is a Boolean.) When we are dealing with a character field I use the “-” in the format string to left-justify the output. Most everything else I simply let slam against the right. I don't bother formatting date values in this example. You can write thousands of lines of code formatting dates in every format imaginable, and still somebody will want the date formatted differently.

testShowMe.java

```

1)  public class testShowMe {
2)
3)      public static void main(String args[]){
4)          showMe s = new showMe();
5)
6)          s.showDBF("class.dbf");
7)          System.out.println( "\n");
8)          s.showDBF( "teacher.dbf");
9)          System.out.println( "\n");
10)         System.out.println( "      Entire class.dbf");
11)         s.dump_records( "class.dbf");
12)         System.out.println( "\n");
13)         System.out.println( "      Records 2 and 3 from class.dbf");
14)         s.dump_records( "class.dbf", 2, 3);
15)         System.out.println( "\n");
16)         System.out.println( "      First record from teacher.dbf");
17)         s.dump_records( "teacher.dbf", 1);
18)
19)     } // end main method
20)
21) } // end class testShowMe

```

The test module simply displays various things from the two DBF files created by the sample programs which are posted on the xBaseJ Web site. I have shown you the program which creates the class.dbf file in this book. We don't really have any need to cover the teacher.dbf, but it is created by example3.java found in the xBaseJ distribution or on the SourceForge site.

```
roland@logikaldesktop:~/fuelsurcharge2$ source ./env1
roland@logikaldesktop:~/fuelsurcharge2$ javac showMe.java
jroland@logikaldesktop:~/fuelsurcharge2$ javac testShowMe.java
roland@logikaldesktop:~/fuelsurcharge2$ java testShowMe
```

```
class.dbf has:
  3 records
  7 fields
```

FIELDS				
Name	Type	Length	Decimals	
CLASSID	C	9	0	
CLASSNAME	C	25	0	
TEACHERID	C	9	0	
DAYSMEET	C	7	0	
TIMEMEET	C	4	0	
CREDITS	N	2	0	
UNDERGRAD	L	1	0	

```
teacher.dbf has:
  3 records
  4 fields
```

FIELDS				
Name	Type	Length	Decimals	
TEACHERID	C	9	0	
TEACHERNM	C	25	0	
DEPT	C	4	0	
TENURE	L	1	0	

Entire class.dbf						
CLASSID	CLASSNAME	TEACHERID	DAYSMEET	TIMEMEET	CREDITS	UNDERGRAD
JAVA501	JAVA And Abstract Algebra	120120120	NNYNNYN	0930	6	F
JAVA10200	Intermediate JAVA	300020000	NNYNNYN	0930	3	T
JAVA10100	Introduction to JAVA	120120120	NNYNNYN	0800	3	T

Records 2 and 3 from class.dbf						
CLASSID	CLASSNAME	TEACHERID	DAYSMEET	TIMEMEET	CREDITS	UNDERGRAD
JAVA10200	Intermediate JAVA	300020000	NNYNNYN	0930	3	T
JAVA10100	Introduction to JAVA	120120120	NNYNNYN	0800	3	T

First record from teacher.dbf				
TEACHERID	TEACHERNM	DEPT	TENURE	
120120120	Joanna Coffee	0800	T	

When I paste the output into this book layout, we end up with some wrapping problems due to the width restrictions of the page. I had to shrink the font so it would fit on a line for you. As you can see, the output is nicely formatted. Once I get past displaying all of the columns on each database, I display the entire contents of the class.dbf file. This first display allows us to verify that the second display, restricting output to the second and third record, actually worked. The last test is simply displaying only the first record from the teacher.dbf file.

You will squirrel away the showMe.java source file in your toolbox. You might even join the xBaseJ project team and fix a few things with the library, then clean this example up. If you were paying attention reading the chart starting on page 20 you also noted that the xBASE universe contains a lot of data types which simply aren't supported by this library. Autoincrement (+) would be a nice addition, but probably not the first you should tackle since doing so would most likely require that you understand more about the header record. The Datetime (T) column type would be a welcome addition. When you get into more transaction-oriented applications the field becomes extremely important. Double (O) and Integer (I) would be interesting for those who believe they are Uber geeks and capable of getting those datatypes to be correctly stored *no matter what platform xBaseJ is running on*.

I must warn you that I never tested the Picture (P) data type. I had no desire to create a database and store images in it. More importantly, I had no desire to figure out what image types (JPEG, BMP, PNG, etc.) were actually supported. I know *why* pictures were added, but I never played much in that world. Images were added so store catalog/inventory records could contain a memo field with the item description, and a picture field with an image of the item. If you have an eBay store with around 100 items, this is fine. If you are running a serious business or think it *might* turn into a serious business, you should really start with a relational database and bolt on what you need later. (Remember that 2GB data file limit? We always *say* the 4GB file limit was imposed by FAT32, but have you checked the header file and 10 digit tag algorithm of a memo file to ensure it isn't limited by an unsigned 32-bit integer as well?)

1.10 Programming Assignment 3

Modify the last two dump_records() calls in testShowMe.java to dump only the second record of class.dbf and only the third record respectively. This is a very simple assignment designed to build confidence in the boundary logic of the dump_records() method.

Create your own version of testShowMe.java which operates on the teacher.dbf file. I haven't provided you the source to create the teacher.dbf file, so you will need to pull it down from the xBaseJ project site.

1.11 Descending Indexes and Index Lifespan

You have already seen how indexes can be useful when it comes to keeping data in a sorted order. Even if the data isn't physically sorted, the index allows you to retrieve it in the order you want. We haven't done much with direct record access yet, but you probably understand it is another benefit of having an index.

One extremely useful type of index is the descending index. You will find this type of index used for many different things in a production world. Some inventory systems use a descending index for product keys in their inventory files. Let's say you run a gas station with a small convenience store in it. You add new products to your inventory file based on a 4-character product category and a 10-digit item number, then you assign it some kind of barcode scan value. You need to keep the categories split out for various tax reasons. As long as the item number is unique, you don't personally care what it is. You might have some data looking like this:

```
TOBA0009876543  CIG GENERIC MENTH BOX
TOBA0009876542  CIG GENERIC MENTH
TOBA0009876541  CIG GENERIC
DARY0000056432  QUART MILK 2%
DARY0000056431  QUART MILK WHOLE
DARY0000056430  QUART MILK CHOC 2%
```

Whenever you added a new product, you would only need to know what category to put it under and your system could automatically calculate the next item number by using the category against the descending key. Given the data above, the first hit for "TOBA" would return "TOBA00009876543" which would let our routine add one to the numeric portion for a new key. Likewise, "DARY" would return "DARY0000056432." (Yes, dary is really spelled dairy, but not in old-school inventoryspeak.)

xBaseJ didn't provide us with descending indexes. This was actually a flaw many early xBASE packages had. A lot of useless data ended up getting stored in production data files trying to work around this problem. It was not uncommon to find bogus numeric columns which contained the result of a field of all 9's with another column (usually a date) subtracted from it. It would be this bogus column, not the actual column, which would be made part of a key.

```
MYDT      MYDTDS
19990801  80009198
19550401  80449598
19200101  80799898
```

As you can see, the newest date has the smallest value, which makes it the first key in an ascending key list. Developers dealing with character fields wrote functions and subroutines which would subtract the string from a string of all Z's to achieve this same sort order.

If you are someone who has never had a machine slower than 2Ghz or less than 2GB of RAM, you probably have a lot of trouble understanding why descending indexes are so important. You will happily use the `startBottom()` and `readPrev()` methods provided by `xBaseJ` performing needless I/O on nearly one-third of the database looking for that one particular record. Those of us who grew up in the PC era understand the need completely. We used to have to wait multiple seconds for each record to be retrieved from that 10MB full-height hard drive. Even if they deny it, we all know that Seagate added that chirping cricket sound and flashing light simply to keep people entertained while they were desperately trying to find the disk block they were interested in.

While you can find the record you are looking for by brute force, index searching is much less resource intensive. I'm not going to print the source for `find_entry(NodeKey, Node, int)` of the `NDX.java` file in this book. It's some pretty intense code. It's not difficult to read, just one of those routines where you have to wrap your mind around it at one sitting, and not get up to go to the bathroom until you have found what you intended to find. All you need to know is that it relies on a bunch of other classes to walk the nodes in the Btree looking for your key value. Ultimately, it is this method which gets called from the `DBF` class whenever you call `find("abc")`. (Assuming, of course, that you are using `NDX` instead of `MDX` as your indexed file.)

doeHistory.java

```
1) import java.io.*;
2) import java.util.*;
3) import org.xBaseJ.*;
4) import org.xBaseJ.fields.*;
5) import org.xBaseJ.Util.*;
6) import org.xBaseJ.indexes.NDX;
7)
8) public class doeHistory {
9)
10)     // variables used by the class
11)     //
12)     private DBF aDB = null;
13)
14)     // fields
15)     public DateField effectiveDT = null;
16)     private NumField effectiveDTDesc = null;
17)     public NumField fuelPrice = null;
18)
19)     // file names
20)     public final String DEFAULT_DB_NAME = "doehst.dbf";
21)     public final String DEFAULT_K0_NAME = "doe_k0.ndx";
22)     public final String DEFAULT_K1_NAME = "doe_k1.ndx";
23)
24)     // work variables
25)     private boolean continue_flg = true;
26)     private boolean dbOpen      = false;
27)
28)     // result codes
29)     public static final int DOE_SUCCESS          = 1;
30)     public static final int DOE_DUPE_KEY        = 2;
```

```

31)     public static final int DOE_KEY_NOT_FOUND = 3;
32)     public static final int DOE_FILE_OPEN_ERR = 4;
33)     public static final int DOE_DEVICE_FULL = 5;
34)     public static final int DOE_NO_CURRENT_REC = 6;
35)     public static final int DOE_DELETE_FAIL = 7;
36)     public static final int DOE_GOTO_FAIL = 8;
37)     public static final int DOE_DB_CREATE_FAIL = 9;
38)     public static final int DOE_INVALID_DATA = 10;
39)     public static final int DOE_END_OF_FILE = 11;
40)
41)     //////////////////////////////////////
42)     //      Method to add a record
43)     //      This method assumes you have values already loaded
44)     //
45)     //      Many different flavors exist to accommodate what the
46)     //      user may have for input
47)     //////////////////////////////////////
48)     public int add_record() {
49)         int ret_val = DOE_SUCCESS;
50)         long x;
51)
52)         try {
53)             x = 99999999 - Long.parseLong(effectiveDT.get());
54)         } catch (NumberFormatException n) {
55)             x = 99999999;
56)         } // end catch NumberFormatException
57)
58)         //
59)         //      stop the user from doing something stupid
60)         //
61)         if (!dbOpen)
62)             return DOE_FILE_OPEN_ERR;
63)
64)         try {
65)             effectiveDTDesc.put( x);
66)             aDB.write();
67)         } catch ( xBaseJException j){
68)             ret_val = DOE_DUPE_KEY;
69)             System.out.println( j.getMessage());
70)         } // end catch
71)         catch( IOException i){
72)             ret_val = DOE_DEVICE_FULL;
73)         } // end catch IOException
74)
75)
76)         return ret_val;
77)     } // end add_record method
78)
79)     public int add_record( String d, String f) {
80)         int ret_val = DOE_SUCCESS;
81)
82)         try {
83)             effectiveDT.put( d);
84)             fuelPrice.put( f);
85)         } catch ( xBaseJException j){
86)             if (j.getMessage().indexOf( "Invalid length for date Field") > -1)
87)                 ret_val = DOE_INVALID_DATA;
88)             else
89)                 ret_val = DOE_DUPE_KEY;
90)         } // end catch
91)
92)
93)         if (ret_val == DOE_SUCCESS)

```

```

94)         return add_record();
95)     else
96)         return ret_val;
97) }
98)
99) public int add_record( Date d, float f) {
100)     int ret_val = DOE_SUCCESS;
101)
102)     try {
103)         effectiveDT.put( d);
104)         fuelPrice.put( f);
105)     } catch ( xBaseJException j){
106)         ret_val = DOE_DUPE_KEY;
107)     } // end catch
108)
109)     if (ret_val == DOE_SUCCESS)
110)         return add_record();
111)     else
112)         return ret_val;
113) }
114)
115) public int add_record( DateField d, NumField f) {
116)     effectiveDT = d;
117)     fuelPrice = f;
118)     return add_record();
119) }
120)
121) //;;;;;;;;;;;;;
122) //      Method to populate known class level field objects.
123) //      This was split out into its own method so it could be used
124) //      by either the open or the create.
125) //;;;;;;;;;;;;;
126) private void attach_fields( boolean created_flg) {
127)     try {
128)         if ( created_flg) {
129)             //Create the fields
130)             effectiveDT      = new DateField( "ef_dt");
131)             fuelPrice        = new NumField( "fuelprice", 6, 3);
132)             effectiveDTDesc  = new NumField( "ef_dtds", 8, 0);
133)
134)             //Add field definitions to database
135)             aDB.addField(effectiveDT);
136)             aDB.addField(effectiveDTDesc);
137)             aDB.addField(fuelPrice);
138)
139)         } else {
140)             effectiveDT      = (DateField) aDB.getField("ef_dt");
141)             fuelPrice        = (NumField) aDB.getField("fuelprice");
142)             effectiveDTDesc  = (NumField) aDB.getField("ef_dtds");
143)         }
144)
145)     } catch ( xBaseJException j){
146)         j.printStackTrace();
147)     } // end catch
148)     catch( IOException i){
149)         i.printStackTrace();
150)     } // end catch IOException
151) } // end attach_fields method
152)
153) //;;;;;;;;;;;;;
154) //      Method to close the database.
155) //      Don't print stack traces here.  If close fails it is
156) //      most likely because the database was never opened.

```

```

157) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
158) public void close_database() {
159)     if (!dbOpen)
160)         return;
161)     try {
162)         if (aDB != null) {
163)             aDB.close();
164)             dbOpen = false;
165)         }
166)     } catch (IOException i) {}
167) } // end close_database method
168)
169)
170) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
171) // Method to create a shiny new database
172) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
173) public void create_database() {
174)     try {
175)         //Create a new dbf file
176)         aDB=new DBF(DEFAULT_DB_NAME,true);
177)
178)         attach_fields(true);
179)
180)         aDB.createIndex(DEFAULT_K1_NAME,"ef_dtds", true, true);
181)         aDB.createIndex(DEFAULT_K0_NAME,"ef_dt",true,true);
182)         dbOpen = true;
183)     } catch( xBaseJException j){
184)         System.out.println( "xBaseJ Error creating database");
185)         j.printStackTrace();
186)         continue_flg = false;
187)     } // end catch
188)     catch( IOException i){
189)         System.out.println( "IO Error creating database");
190)         i.printStackTrace();
191)         continue_flg = false;
192)     } // end catch IOException
193) } // end create_database method
194)
195) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
196) // Method to delete a record from the database
197) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
198) public int delete_record() {
199)     int ret_val = DOE_SUCCESS;
200)
201)     if (!dbOpen)
202)         return DOE_FILE_OPEN_ERR;
203)
204)     if (aDB.getCurrentRecordNumber() < 1) {
205)         System.out.println( "current record number " +
206)             aDB.getCurrentRecordNumber());
207)         ret_val = DOE_NO_CURRENT_REC;
208)     }
209)     else {
210)         try {
211)             aDB.delete();
212)         } catch( xBaseJException j){
213)             ret_val = DOE_DELETE_FAIL;
214)         } // end catch
215)         catch( IOException i){
216)             ret_val = DOE_DELETE_FAIL;
217)         } // end catch IOException
218)     } // end test for current record
219)

```

```

220)     return ret_val;
221) } // end delete_record method
222)
223) public int delete_record( String d) {
224)     int ret_val = DOE_SUCCESS;
225)
226)     ret_val = find_EQ_record( d);
227)     if ( ret_val == DOE_SUCCESS)
228)         ret_val = delete_record();
229)     return ret_val;
230)
231) } // end delete_record method
232)
233) public int delete_record( int record_num) {
234)     int ret_val = DOE_SUCCESS;
235)
236)     if (!dbOpen)
237)         return DOE_FILE_OPEN_ERR;
238)
239)     try {
240)         aDB.gotoRecord( record_num);
241)     } catch( xBaseJException j){
242)         j.printStackTrace();
243)         ret_val = DOE_NO_CURRENT_REC;
244)     } // end catch
245)     catch( IOException i){
246)         i.printStackTrace();
247)         ret_val = DOE_NO_CURRENT_REC;
248)     } // end catch IOException
249)
250)     if (ret_val == DOE_SUCCESS)
251)         ret_val = delete_record();
252)
253)     return ret_val;
254) } // end delete_record method
255)
256) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
257) // Method to dump first 10 records
258) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
259) public void dump_first_10() {
260)     if (!dbOpen) {
261)         System.out.println( "Must open database first");
262)         return;
263)     } // end test for open database
264)
265)     try {
266)         System.out.println( "\nDate      Price");
267)         System.out.println( "-----  -----");
268)         for (int x=1; x < 11; x++) {
269)             aDB.gotoRecord(x);
270)             System.out.println( effectiveDT.get() + "    " + fuelPrice.get());
271)         } // end for x loop
272)     } catch( xBaseJException j){
273)         j.printStackTrace();
274)     } // end catch
275)     catch( IOException i){
276)         i.printStackTrace();
277)     } // end catch IOException
278)
279) } // end dump_first_10 method
280)
281) public void dump_first_10_k0() {
282)     if (!dbOpen) {

```

```

283)         System.out.println( "Must open database first");
284)         return;
285)     } // end test for open database
286)
287)     try {
288)         aDB.useIndex( DEFAULT_K0_NAME);
289)         aDB.startTop();
290)         System.out.println( "\nDate      Price");
291)         System.out.println( "-----  -----");
292)         for (int x=1; x < 11; x++) {
293)             aDB.findNext();
294)             System.out.println( effectiveDT.get() + "    " + fuelPrice.get());
295)         } // end for x loop
296)     } catch( xBaseJException j){
297)         j.printStackTrace();
298)     } // end catch
299)     catch( IOException i){
300)         i.printStackTrace();
301)     } // end catch IOException
302)
303) } // end dump_first_10_k0 method
304)
305) public void dump_first_10_k1() {
306)     if (!dbOpen) {
307)         System.out.println( "Must open database first");
308)         return;
309)     } // end test for open database
310)
311)     try {
312)         aDB.useIndex( DEFAULT_K1_NAME);
313)         aDB.startTop();
314)         System.out.println( "\nDate      Price");
315)         System.out.println( "-----  -----");
316)         for (int x=1; x < 11; x++) {
317)             aDB.findNext();
318)             System.out.println( effectiveDT.get() + "    " + fuelPrice.get());
319)         } // end for x loop
320)     } catch( xBaseJException j){
321)         j.printStackTrace();
322)     } // end catch
323)     catch( IOException i){
324)         i.printStackTrace();
325)     } // end catch IOException
326)
327) } // end dump_first_10_k1 method
328)
329) //;;;;;;;;;;;;;
330) // Method to find a record
331) //;;;;;;;;;;;;;
332) public int find_EQ_record( String d) {
333)     int ret_val = DOE_SUCCESS;
334)     boolean perfect_hit;
335)
336)     if (!dbOpen)
337)         return DOE_FILE_OPEN_ERR;
338)
339)     try {
340)         aDB.useIndex( DEFAULT_K0_NAME);
341)         perfect_hit = aDB.findExact( d);
342)         if ( !perfect_hit) {
343)             System.out.println( "missed");
344)             System.out.println( "Current Record " +
aDB.getCurrentRecordNumber());

```

```

345)         ret_val = DOE_KEY_NOT_FOUND;
346)     }
347) } catch( xBaseJException j){
348)     System.out.println( j.getMessage());
349)     ret_val = DOE_KEY_NOT_FOUND;
350) } // end catch
351) catch( IOException i){
352)     ret_val = DOE_KEY_NOT_FOUND;
353) } // end catch IOException
354)
355)     return ret_val;
356) } // end find_EQ_record method
357)
358) public int find_GE_record( String d) {
359)     int ret_val = DOE_SUCCESS;
360)
361)     if (!dbOpen)
362)         return DOE_FILE_OPEN_ERR;
363)
364)     try {
365)         aDB.useIndex( DEFAULT_K0_NAME);
366)         aDB.find( d);
367)     } catch( xBaseJException j){
368)         ret_val = DOE_KEY_NOT_FOUND;
369)     } // end catch
370)     catch( IOException i){
371)         ret_val = DOE_KEY_NOT_FOUND;
372)     } // end catch IOException
373)
374)     return ret_val;
375) } // end find_GE_record method
376)
377) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
378) //    Method to retrieve the newest record by date
379) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
380) public int get_newest() {
381)     int ret_val = DOE_SUCCESS;
382)
383)     if (!dbOpen)
384)         return DOE_FILE_OPEN_ERR;
385)
386)     try {
387)         aDB.useIndex( DEFAULT_K1_NAME);
388)         aDB.startTop();
389)         aDB.findNext();
390)     } catch( xBaseJException j){
391)         ret_val = DOE_KEY_NOT_FOUND;
392)     } // end catch
393)     catch( IOException i){
394)         ret_val = DOE_KEY_NOT_FOUND;
395)     } // end catch IOException
396)
397)     return ret_val;
398) } // end get_newest method
399)
400) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
401) //    method to get next record no matter
402) //    what index is in use.
403) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
404) public int get_next() {
405)     int ret_val = DOE_SUCCESS;
406)     try {
407)         aDB.findNext();

```

```

408)         } catch( xBaseJException j){
409)             ret_val = DOE_KEY_NOT_FOUND;
410)         } // end catch
411)         catch( IOException i){
412)             ret_val = DOE_KEY_NOT_FOUND;
413)         } // end catch IOException
414)
415)         return ret_val;
416)
417)
418)     } // end get_next method
419)     //////////////////////////////////////////////////
420)     //      method to retrieve the oldest record
421)     //////////////////////////////////////////////////
422)     public int get_oldest() {
423)         int ret_val = DOE_SUCCESS;
424)         if (!dbOpen)
425)             return DOE_FILE_OPEN_ERR;
426)
427)         try {
428)             aDB.useIndex( DEFAULT_K0_NAME);
429)             aDB.startTop();
430)             aDB.findNext();
431)         } catch( xBaseJException j){
432)             ret_val = DOE_KEY_NOT_FOUND;
433)         } // end catch
434)         catch( IOException i){
435)             ret_val = DOE_KEY_NOT_FOUND;
436)         } // end catch IOException
437)
438)         return ret_val;
439)     } // end get_oldest method
440)
441)     //////////////////////////////////////////////////
442)     //      Method to test private flag and see
443)     //      if database has been successfully opened.
444)     //////////////////////////////////////////////////
445)     public boolean isOpen() {
446)         return dbOpen;
447)     } // end ok_to_continue method
448)
449)     //////////////////////////////////////////////////
450)     //      Method to open an existing database and attach primary key
451)     //////////////////////////////////////////////////
452)     public int open_database() {
453)         int ret_val = DOE_SUCCESS;
454)
455)         try {
456)
457)             //Create a new dbf file
458)             aDB=new DBF(DEFAULT_DB_NAME);
459)
460)             attach_fields( false);
461)
462)             aDB.useIndex( DEFAULT_K0_NAME);
463)             //      aDB.useIndex( DEFAULT_K1_NAME);
464)             dbOpen = true;
465)         } catch( xBaseJException j){
466)             continue_flg = false;
467)         } // end catch
468)         catch( IOException i){
469)             continue_flg = false;
470)         } // end catch IOException

```



```

471)
472)         if (!continue_flg) {
473)             continue_flg = true;
474)             System.out.println( "Open failed, attempting create");
475)             create_database();
476)         } // end test for open failure
477)
478)         if (isOpen())
479)             return DOE_SUCCESS;
480)         else
481)             return DOE_FILE_OPEN_ERR;
482)     } // end open_database method
483)
484)     //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
485)     //      Method to re-index all of the associated index files.
486)     //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
487)     public void reIndex() {
488)         if (aDB != null) {
489)             if (isOpen()) {
490)                 try {
491)                     NDX n = null;
492)                     for (int i=1; i <= aDB.getIndexCount(); i++) {
493)                         n = (NDX) aDB.getIndex( i);
494)                         n.reIndex();
495)                     }
496)                 } catch( xBaseJException j){
497)                     j.printStackTrace();
498)                 } // end catch
499)                 catch( IOException i){
500)                     i.printStackTrace();
501)                 } // end catch IOException
502)             } // end test for open database
503)         } // end test for initialized database object
504)     } // end reIndex method
505)
506)     //
507)     // We don't need to create a finalize method since
508)     // each object we create has one. You only create a finalize
509)     // method if you allocate some resource which cannot be
510)     // directly freed by the Java VM.
511)     //
512) } // end class doeHistory

```

I took this example a little farther than most of you probably would in real life. Those of you who have read other books in this series know that at one point in my life I worked at a DEC VAR which sold a customized ERP package. The package was written in DEC BASIC. All of our main files had I/O routines written for them. There were one or two include files you added to the top of your program, and presto, you had access to all of the I/O routines for that file. The file name and channel number was pre-assigned, methods were there for reading/writing/deleting/finding records, and all exceptions were caught in the routine itself. The functions returned values which had standardized names; in fact, all of the function/method names were the same for all of the files.

Writing that first I/O module was a massive pain. Every other file we added after that one was almost free. Depending upon how fast a typist you were, it took 1-4 hours to create all of the record layouts and clone a new I/O source file from the original. The programs themselves got a lot simpler.

Most of you will note that I didn't take this I/O routine quite that far. I didn't want to print a 5000-line source file in this book. Even a 500-line source file is pushing it if I actually want you to read it.

I told you the above story so you would have some frame of reference for why this source file was structured the way it was. Java doesn't have record layouts and maps, so I made the two primary fields public. I deliberately made `effectiveDTDesc` private because that is really for class use only. I kept the `DBF` private as well. As we proceed with our design I may regret that decision, but I wanted to fend off 'bit twiddler' impulse." Odds are greater that I will add the function to the class if I have to open the class source file to get access to the database object...at least that is my working theory.

This class also has a bunch of public constants. All of the file names are string constants and the result codes of integer constants. If you spend any time at all working in IT and working with indexed files, you will see `K0` used to refer to a primary key, `K1` to the first alternate key, etc. I have used this notation throughout my career and this book series. When you are dealing with segmented keys you will see each field of the key have a ".n" after the key number, such as `K1.1`, `K1.2`, `K1.3` etc. There is never a zero segment. It's not because we don't like zero, we simply don't want to confuse management telling them `K0` on this file is a single field primary key, but `K0.0` on this other file indicates the first field of a segmented primary key. As a general rule, MBAs don't do zero well.

Since I did not want some user directly manipulating the Boolean work variables the class needed I made them private. One of them is for internal use only and the other has a method which will be presented later to allow read access.

I did quite a bit of polymorphism with this class. The `add_record()` methods should serve as a good example. The one which does actual IO is also the default one. Notice how I subtract the date value from 99999999 to get a value for `effectiveDTDesc`. While it is true that 99999999 isn't a valid date, it is also true that a string of all 9s will do what we need. Since our indexes only ascend, we need a value that gets smaller as the date gets bigger. The overridden methods of `add_record()` are simply there to provide a convenient way of adding a record in a single step.

The `attach_fields()` method shouldn't require much explanation as you have already seen me create a method just like this. It is easier to handle this all in one method than replicate it in other methods.

We could have quite a bit of discussion over the `close_database()` method. Not so much concerning the method itself, but the fact I didn't include a `finalize()` method calling it. There are multiple schools of thought on this topic, and now is a good time to discuss them.

My class didn't allocate any system resources on its own. It didn't open any files, allocate any devices, or physically allocate regions of memory on its own. Every time it did such a thing it used something else to do it. Our class variables were allocated by the Java Virtual Machine (JVM). They were either native datatypes or instances of classes provided by others. When you develop a class, you are responsible for freeing any resources which cannot be freed by the JVM in your `finalize()` method.

When an object goes out of scope it is flagged for garbage collection inside of the JVM. That collection may occur instantaneously or it may take days for the JVM to garbage-collect it. The `finalize()` method must physically free any resources (other than dynamically allocated RAM) which might be needed somewhere else. (We are not going into a discussion over the differences between dynamically allocated and physically mapped memory as it is a topic for low-level programming, not business application programming.)

The class we have presented here uses other classes to open the files. When you look at the source for `DBF.java`, you will see the following:

```
public void finalize() throws Throwable {
    try {
        close();
    } catch (Exception e) {
        ;
    }
}
```

When you poke around in that same source file and find the `close()` method, you will see the following:

```

public void close() throws IOException {
    short i;

    if (dbtobj != null)
        dbtobj.close();

    Index NDxes;
    NDX n;

    if (jNDxes != null) {
        for (i = 1; i <= jNDxes.size(); i++) {
            NDxes = (Index) jNDxes.elementAt(i - 1);
            if (NDxes instanceof NDX) {
                n = (NDX) NDxes;
                n.close();
            }
        }
    }
    // end test for null jNDxes 20091010_rth

    if (MDXfile != null)
        MDXfile.close();

    dbtobj = null;
    jNDxes = null;
    MDXfile = null;
    unlock();

    file.close();
}

```

As you should be able to tell, there is a cascading effect when garbage collection starts to clean up our class. When it attempts to reclaim the class variable `aDB` it will be forced to call the `finalize()` method for the `DBF` class. That will free up the data file and the index objects. The index objects will then be garbage-collected, which will free up the files they used, etc.

Since we are talking about deleting things, we should move on to the `delete_record()` method and its overrides. I seriously debated making the default method, which deletes the current record, private. It's a dangerous thing, especially if someone doesn't bother to check the results of their `find` call before performing the delete. The overridden versions of the method actually ensure they locate the correct record prior to deletion.

I probably should not have stuck the dump methods in this class, but it made them quick to write. Besides, doing so ensures I have a programming assignment for you after this. Notice that each dump routine makes certain the database is open before proceeding. A lot of Java developers don't do stuff like this; they simply let things crash, throwing exceptions. One thing you may not readily notice is that each dump routine which needs an index makes certain the index it needs is currently on top by calling `useIndex()`. Readers of other books in this series will recognize that as "establishing the key of reference" from other languages.

Once we have our key of reference established, the DBF class provides two handy index positioning methods: `startTop()` and `startBottom()`. Once you have established a position and a key of reference you can use either `findNext()` or `findPrev()` to navigate forward or backward from your current position. The DBF class also provides `read()` and `readPrev()`. There is a big difference between the read and the find methods, though. The read methods require that you have a current record to establish a position in the file. The find methods only require some position to have been established in the currently active index. If you open a database, call `startTop()`, then attempt to call `read()` the call will fail.

I added two find methods to this class. Both of them only work with the primary key. The first will succeed only if a perfectly matching key is found; the second will succeed if a key which is greater than or equal to the search value is found. It should be noted that these are highly restrictive methods which should have names which better indicate their restrictiveness.

The only method I provided to operate on the second index is `get_newest()`. The application I need to write in real life needs to quickly identify the newest record on file. That record will provide the “current” value until a newer record is added. This method looks much like our other find methods. We establish a key of reference, call `startTop()` to establish a position, then call `findNext()` to pull in the record.

You might think at first glance that `get_next()` is coded incorrectly. It makes no attempt to establish any key of reference or position via that key. It couldn't care less where it is or where it is going. If you happen to hit one end or the other on the file it will let you know by returning `DOE_KEY_NOT_FOUND`, otherwise it returns `DOE_SUCCESS` and you can be fairly certain you got the record you wanted.

If you were confused by `get_next()` or `get_newest()`, then you should love `get_oldest()`. It's not a complex routine; it simply needs you to really understand both how the indexes work and what the key actually contains. Remember, indexes are stored only in ascending order with this library. The smallest date value on file will be the first record in the primary index. We find the oldest record by getting the record at the top of the primary index (K0) and the newest record by getting the record at the top of the secondary (K1) index. It is true that you could also get the newest record on file by calling `startBottom()` on the primary key and work your way back to the oldest record by using `findPrev()`, but when you get your programming assignments you will thank me.

Finally we get to listing line 445. I had to keep track of the database open state and provide a method of determining its current state to the outside world. I must apologize to the Java hackers of the world who think it is just dandy to never provide this capability and to just let things throw exceptions in production. I am not from that school. I'm from the school of those who used to work in operations and had to wake people up at 2AM. I'm also from the school of those who used to work production support along with their development duties and had to get those calls at 2AM. The reason we used to make programmers start out in operations, then work production support, is to teach them what happens when busted things get turned in to production. Call `isOpen()` before you do something and avoid crashing from an unhandled exception.

We will talk about `reIndex()` after we discuss the test application.

testDoeHistory.java

```

1)  import java.text.*;
2)  import java.util.*;
3)  import java.io.*;
4)
5)  import org.xBaseJ.*;
6)
7)  public class testDoeHistory {
8)
9)      public static void main(String args[]){
10)         doeHistory d = new doeHistory();
11)
12)         //
13)         //  You must set this unless you want NULL bytes padding out
14)         //  character fields.
15)         //
16)         try {
17)             Util.setXBaseJProperty("fieldFilledWithSpaces","true");
18)         } catch (IOException e) {
19)             System.out.println( "An IO Exception occurred");
20)             System.out.println( e.toString());
21)             e.printStackTrace();
22)         }
23)         d.create_database();
24)
25)         if (d.isOpen()) {
26)             String line_in_str = null;
27)             long l_record_count = 0;
28)             boolean eof_flg = false;
29)             FileReader in_file = null;
30)             BufferedReader input_file = null;
31)
32)             try {
33)                 in_file = new FileReader( "fuel_prices.csv");
34)             } catch (FileNotFoundException f) {
35)                 System.out.println( "File Not Found  fuel_prices.csv");
36)                 eof_flg = true;
37)             } // end catch for file not found
38)
39)             if (eof_flg == false) {
40)                 input_file = new BufferedReader( in_file,4096);
41)                 System.out.println("\nPopulating database");
42)             }

```

```
43)
44)         while (eof_flg == false) {
45)             try {
46)                 line_in_str = input_file.readLine();
47)             }
48)             catch (EOFException e) {
49)                 System.out.println( "End of file exception");
50)                 eof_flg = true;
51)             }
52)             catch (IOException e) {
53)                 System.out.println( "An IO Exception occurred");
54)                 System.out.println( e.toString());
55)                 e.printStackTrace();
56)                 eof_flg = true;
57)             }
58)             if (eof_flg == true)         continue;
59)             if (line_in_str == null) {
60)                 System.out.println( "End of input file reached");
61)                 eof_flg = true;
62)                 continue;
63)             }
64)
65)             l_record_count++;
66)             String input_flds[] = line_in_str.split( ",");
67)
68)             try {
69)                 d.effectiveDT.put( input_flds[0]);
70)                 d.fuelPrice.put( input_flds[1]);
71)                 d.add_record();
72)             } catch ( xBaseJException j){
73)                 j.printStackTrace();
74)             } // end catch
75)         } // end while loop to load records
76)
77)         System.out.println( "Finished adding " + l_record_count +
78)                             " records\n");
79)         //
80)         // Now that we have some data, let's use some
81)         // of the other methods
82)         //
83)         // First make sure the open works
84)         d.close_database();
85)
86)         doeHistory h = new doeHistory();
87)         h.open_database();
88)         if (!h.isOpen()) {
89)             System.out.println("Unable to open the database");
90)         } else {
91)
92)             // add a record with a future date
93)             //
94)             int x;
95)             x = h.add_record( "20121003", "13.41");
96)             System.out.println( "Result of add " + x);
97)             try {
98)                 h.effectiveDT.put( "20110830");
99)                 h.fuelPrice.put( "29.95");
100)            } catch( xBaseJException j) { j.printStackTrace();}
101)
102)            x = h.add_record();
103)            System.out.println( "result of second add " + x);
104)
105)            System.out.println( "First 10 in order added");
```

```

106)         h.dump_first_10();
107)         System.out.println( "First 10 in descending date order");
108)         h.dump_first_10_k1();
109)         System.out.println( "First 10 in ascending date order");
110)         h.dump_first_10_k0();
111)
112)         // Now let us see what keys have actual
113)         // data
114)         //
115)         System.out.println( "\nBefore reIndex\n");
116)         System.out.println( "finding 20071010");
117)         x = h.find_EQ_record( "20071010");
118)         System.out.println( "\nResult of EQ find " + x + "\n");
119)         System.out.println( "Date: " + h.effectiveDT.get()
120)             + " Price: " + h.fuelPrice.get());
121)
122)         x = h.get_newest();
123)         System.out.println( "Result of get_newest " + x);
124)         System.out.println( "Date was: " + h.effectiveDT.get());
125)
126)         // Not all keys are updated when using NDX
127)         //
128)         h.reIndex();
129)
130)         System.out.println( "\nAfter reIndex\n");
131)         System.out.println( "First 10 in descending date order\n");
132)         h.dump_first_10_k1();
133)         System.out.println( "\nfinding 20071010");
134)         x = h.find_GE_record( "20071010");
135)         System.out.println( "\nResult of EQ find " + x + "\n");
136)         System.out.println( "Date: " + h.effectiveDT.get()
137)             + " Price: " + h.fuelPrice.get());
138)         if ( x == h.DOE_SUCCESS) {
139)             x = h.delete_record();
140)             System.out.println( "Result of delete " + x + "\n");
141)         }
142)
143)         x = h.get_newest();
144)         System.out.println( "Result of get_newest " + x);
145)         System.out.println( "Date was: " + h.effectiveDT.get());
146)
147)         h.close_database();
148)     } // end test for successful open
149) } // end test for open dbf
150)
151) } // end main method
152)
153) } // end class testShowMe

```

This is one of the longer test programs I have provided you. A big part of that is due to the fact I created a CSV (Comma Separated Value) file called `fuel_prices.csv` which has lines in it looking like this:


```
20070905,289.3
20070912,292.4
20070919,296.4
20070926,303.2
20071003,304.8
20071010,303.5
20071017,303.9
20071024,309.4
20071031,315.7
20071107,330.3
20071114,342.5
20071121,341.0
20071128,344.4
```

Actually it has over 100 lines in it, but I'm certainly not going to print it here. If you want, you can visit the Department of Energy Web site and pull down the spreadsheet which has historic diesel fuel prices, and create your own file.

In theory I could have done the Util call found at listing line 17 inside of the doeHistory class, but I didn't have a warm and fuzzy feeling about the actual run-time scope of Util in all situations. Feel free to experiment on your own with placing this call at various places in the class hierarchy.

Listing lines 26 through 78 serve no other purpose than to read a line from this CSV and load it as a record in the database. Since I tried to implement localized error handling and provide meaningful error messages, this code is a lot larger than you will see in most examples which would simply trap all exceptions at one place and print a stack trace.

We should discuss this code briefly for those who have never tried to read lines in from a text file before. First you have to create a FileReader object as I did at listing line 33. Once you have done that you can create a BufferedReader object to read from and buffer the FileReader object you just created as I did at listing line 40. The second parameter (4096) is an optional buffer size in bytes. If you do not pass a buffer size, there is some value which gets used by default.

One has to use a BufferedReader object if one wishes to read a line of input at a time as we do at listing line 46. The readLine() method of a BufferedReader object ensures that we either get all characters as a String up to the newLine character or the end of the stream. You will not receive the newLine or end of stream termination character(s) in the String.

After we get done dealing with the potential end of file situation we increment the record counter then use the really cool split() method provided by the String class. Since we know the number and order of data in the input file, we can directly put the values into the database fields and add the record to the database. Roughly 50 lines of code just to get our test data, but now we have it.

Listing lines 84 through 147 contain the meat of this test. We need to see the output before we talk about them, though.

```
roland@logikaldesktop:~/fuelsurcharge2$ java testDoeHistory
```

```
Populating database
End of input file reached
Finished adding 107 records
```

```
Result of add 1
result of second add 1
First 10 in order added
```

Date	Price
20070905	89.300
20070912	92.400
20070919	96.400
20070926	03.200
20071003	04.800
20071010	03.500
20071017	03.900
20071024	09.400
20071031	15.700
20071107	30.300

First 10 in descending date order

Date	Price
20090916	63.400
20090909	64.700
20090902	67.400
20090826	66.800
20090819	65.200
20090812	62.500
20090805	55.000
20090729	52.800
20090722	49.600
20090715	54.200

First 10 in ascending date order

Date	Price
20070905	89.300
20070912	92.400
20070919	96.400
20070926	03.200
20071003	04.800
20071010	03.500
20071017	03.900
20071024	09.400
20071031	15.700
20071107	30.300

Before reIndex

finding 20071010

Result of EQ find 1

Date: 20071010 Price: 03.500

```
Result of get_newest 1
Date was: 20090916
```

```
After reIndex
```

```
First 10 in descending date order
```

Date	Price
20121003	13.410
20110830	29.950
20090916	63.400
20090909	64.700
20090902	67.400
20090826	66.800
20090819	65.200
20090812	62.500
20090805	55.000
20090729	52.800

```
finding 20071010
```

```
Result of EQ find 1
```

```
Date: 20071010 Price: 03.500
Result of delete 1
```

```
Result of get_newest 1
Date was: 20121003
```

You should note that the result of both the first add (20121003, 13.41) and the second add (20110830, 29.95) returned a 1, meaning they were successfully added to the database, yet they didn't show up on our initial dump reports. The records don't show up until I call reIndex().

Here is another lovely little tidbit for you. NDX objects don't monitor changes. If, instead of obtaining the the NDX currently attached to the DBF object, I simply create two new objects and re-index, those changes will be reflected in the file, but not in our application.

```
NDX a = new NDX( DEFAULT_K0_NAME, aDB, false);
a.reIndex()
NDX b = new NDX( DEFAULT_K1_NAME, aDB, false);
b.reIndex()
```

The code above will not place entries in our index even though the values will be correct on file. Why? Because the Btree gets loaded into RAM. You have to manipulate the exact same Btree the database object is using. Make no mistake, a call to reIndex() changes the contents of the file, but the other loaded view of it does not. *You should never, under any circumstances, attempt to let multiple users have write access to the same DBF for this, and many other, reasons.* There is no triggering method in place to keep Btrees in synch because there is no database engine in place.

Take another look at listing line 463 in `doeHistory.java`. I have the `useIndex()` for the second key commented out. On page twelve in this book I told you that records could be added to a DBF file without ever creating an entry in an index file. This test has been a shining example. When we call `open_database()` we only open one index. Indeed, the database object doesn't care if we choose to not open any. A good many xBASE libraries out there support only reading and writing of records in xBASE format. They provide no index support whatsoever.

1.12 Programming Assignment 4

This is a multi-part assignment. Your first assignment is to un-comment listing line 463, recompile and re-run this application. You will note that the first set of dump reports now has the added records. Can you explain why?

Part 2: Move all of the `dump_` methods out of `doeHistory.java` and make them methods in `testDoeHistory.java`. Get them to actually work. **DO NOT MAKE THE DBF OBJECT PUBLIC OR USE A METHOD WHICH PROVIDES A COPY OF IT TO A CALLER.** Don't forget to check for the database being open prior to running. Do not add any new methods to `doeHistory.java`.

Part 3: After completing part two, change `getNext()` to use `read()` instead of `findNext()`, compile and document what, if any, difference there is in the output.

Part 4: After completing part three, add one method to `doeHistory.java` which uses the `readPrev()` method of the DBF class to read the previous record. Clone the `dump_first_10_k1()` method you just moved to `testDoeHistory.java` to a method named `dump_last_10_k0()`. Without using the alternate index, get the same 10 records to display.

1.13 Deleting and Packing

I mentioned much of this information earlier but we are going to go over it again in detail because it tends to catch most newbies off-guard even after they have been told a hundred times. Deleting a record in an xBASE file does not physically delete the record (in most versions), nor does it update any NDX index file information. (A production MDX is a slightly different situation.)

Basically, each record in a DBF file has an extra byte at the front of it. When that byte is filled in, usually with an asterisk, the record is considered to be “deleted.” Your applications will continue to read this record and process it unless they call the `deleted()` method immediately after reading a record. The `deleted()` method of the DBF class returns true if the record is flagged for deletion.

One of the dBASE features general users found most endearing was the ability to “undelete” a record they had accidentally deleted. This feature was possible simply because the record had not been physically deleted. The DBF class provides an `undelete()` method for you as well. If you find a record which has been marked as deleted that you wish to restore, you simply read it and call `undelete()`.

It is not unusual to find xBASE files which have never had deleted records removed. As long as a user never hits the 2GB file size limit for a DBF, there is nothing which forces them to get rid of deleted records. Until you hit a size limit (either maximum file size or run out of disk space), you can just go happily on your way.

What if you want to get that space back? What if you need to get it back? Well, then you need to know about the `pack()` method of the DBF class. Many books will tell you that `pack()` removes the deleted records from your database. There may actually be an xBASE toolset out there somewhere which actually implements `pack()` that way. Almost every library I have used throughout my career does what xBaseJ does. They create a shiny new database with a temporary file name, copy all of the records which are not flagged for deletion to the temporary database, close and nuke the original database, then rename/copy the temporary back to where the original database was. If you are looking to `pack()` because you are out of disk space, it is probably already too late for you unless your `/tmp` or `tmp` environment variable is pointing to a different physical disk.

Careful readers will note that I didn't say anything about your index files. `pack()` couldn't care less about them. If you do not re-index your index files or create new index files after calling `pack()`, then you are asking for disaster.

testpackDoeHistory.java

```

1)  ...
2)      System.out.println( "Finished adding " + l_record_count +
3)          " records\n");
4)      //
5)      // Now that we have some data, let's use some
6)      // of the other methods
7)      //
8)      // We need to delete a few records now
9)      for ( int i=1; i < 20; i +=3)
10)         d.delete_record( i);
11)
12)         // First make sure the open works
13)         d.close_database();
14)
15)         // Cheat because I didn't supply the pack method
16)         //
17)         try {
18)             DBF aDB = new DBF(d.DEFAULT_DB_NAME);
19)             System.out.println( "\npacking the database");
20)             aDB.startTop();
21)             aDB.pack();
22)             System.out.print( "\nDatabase has been packed ");
23)             System.out.println( "record count " + aDB.getRecordCount());
24)             aDB.close();
25)         } catch( xBaseJException j) {
26)             j.printStackTrace();
27)         } catch( IOException e) {
28)             e.printStackTrace();
29)         } catch ( CloneNotSupportedException c) {
30)             c.printStackTrace();
31)         }
32)
33)
34)         doeHistory h = new doeHistory();
35)         h.open_database();
36)
37)         if (!h.isOpen()) {
38)             System.out.println("Unable to open the database");
39)         } else {
40)
41)             // add a record with a future date
42)             //
43)             System.out.println( "\nadding records with future dates");
44)             int x;
45)             x = h.add_record( "20121003", "13.41");
46)             try {
47)                 h.effectiveDT.put( "20110830");
48)                 h.fuelPrice.put( "29.95");
49)             } catch( xBaseJException j) { j.printStackTrace();}
50)
51)             x = h.add_record();
52)
53)             x = h.add_record( "20201003", "19.58");
54)             x = h.add_record( "20190903", "21.58");
55)             x = h.add_record( "20180803", "19.58");
56)             x = h.add_record( "20170703", "21.58");
57)             x = h.add_record( "20160603", "19.58");
58)
59)             System.out.println( "First 10 in order added");
60)             h.dump_first_10();
61)             System.out.println( "First 10 in descending date order");

```

```

62)         h.dump_first_10_k1();
63)         System.out.println( "First 10 in ascending date order");
64)         h.dump_first_10_k0();
65)
66)         // Now let us see what keys have actual
67)         // data
68)         //
69)         System.out.println( "\n\nBefore reIndex\n");
70)         System.out.println( "finding 20071010");
71)         x = h.find_EQ_record( "20071010");
72)         System.out.println( "\nResult of EQ find " + x + "\n");
73)         System.out.println( "Date: " + h.effectiveDT.get()
74)         + " Price: " + h.fuelPrice.get());
75)
76)         x = h.get_newest();
77)         System.out.println( "Result of get_newest " + x);
78)         System.out.println( "Date was: " + h.effectiveDT.get());
79)
80)         // Not all keys are updated when using NDX
81)         //
82)         h.reIndex();
83)
84)         System.out.println( "\nAfter reIndex\n");
85)         System.out.println( "First 10 in descending date order\n");
86)         h.dump_first_10_k1();
87)         System.out.println( "\nfinding 20071010");
88)         x = h.find_GE_record( "20071010");
89)         System.out.println( "\nResult of EQ find " + x + "\n");
90)         System.out.println( "Date: " + h.effectiveDT.get()
91)         + " Price: " + h.fuelPrice.get());
92)         if ( x == h.DOE_SUCCESS) {
93)             x = h.delete_record();
94)             System.out.println( "Result of delete " + x + "\n");
95)         }
96)
97)         x = h.get_newest();
98)         System.out.println( "Result of get_newest " + x);
99)         System.out.println( "Date was: " + h.effectiveDT.get());
100)
101)         h.close_database();
102)     } // end test for successful open
103) } // end test for open dbf
104)
105) } // end main method

```

I did not provide the beginning of this source file because I didn't feel like re-printing the code to load the records again. If you want to get this program running you can simply steal the CSV import code from the previously presented program.

At listing line 9 I created a for loop which will delete records from the database. We only need a few near the beginning to disappear.

Listing lines 18 through 24 contain the code where I open the database and call pack(). I cheated here and directly created a database object because I had not added a pack_database() method to the doeHistory class.

You will notice at listing lines 53 through 57 that I chose to add some more records. I just wanted to make things painfully obvious during the rest of the test. There is nothing really magic about the values in those records, other than the fact they are easy to spot.

Pay special attention to listing line 82. Do you remember what I said earlier? I deliberately left this line where it was to prove that statement. Now, let's take a look at the output.

```
roland@logikaldesktop:~/fuelsurcharge2$ javac doeHistory.java
roland@logikaldesktop:~/fuelsurcharge2$ javac testpackDoeHistory.java
roland@logikaldesktop:~/fuelsurcharge2$ java testpackDoeHistory
```

```
Populating database
End of input file reached
Finished adding 107 records
```

```
packing the database
```

```
Database has been packed record count 100
```

```
adding records with future dates
First 10 in order added
```

Date	Price
-----	-----
20070912	92.400
20070919	96.400
20071003	04.800
20071010	03.500
20071024	09.400
20071031	15.700
20071114	42.500
20071121	41.000
20071205	41.600
20071212	32.500
First 10 in descending date order	

Date	Price
-----	-----
20160603	19.580
20170703	21.580
20180803	19.580
20190903	21.580
20201003	19.580
20110830	29.950
20121003	13.410
20090916	63.400
20090909	64.700
20090902	67.400
First 10 in ascending date order	

Date	Price
-----	-----
20070912	92.400
20070919	96.400
20071003	04.800
20071010	03.500
20071024	09.400
20071031	15.700
20071114	42.500


```
20071121    41.000
20071205    41.600
20071212    32.500
```

Before reIndex

```
finding 20071010
```

```
Result of EQ find 1
```

```
Date: 20071031 Price: 15.700
```

```
Result of get_newest 1
```

```
Date was: 20160603
```

After reIndex

First 10 in descending date order

Date	Price
-----	-----
20201003	19.580
20190903	21.580
20180803	19.580
20170703	21.580
20160603	19.580
20121003	13.410
20110830	29.950
20090916	63.400
20090909	64.700
20090902	67.400

```
finding 20071010
```

```
Result of EQ find 1
```

```
Date: 20071010 Price: 03.500
```

```
Result of delete 1
```

```
Result of get_newest 1
```

```
Date was: 20201003
```

Please look at the highlighted lines in the output. When we are done loading the CSV file into the database there are 107 records. After deleting and packing we have 100 records. Take a look at the descending date order report. The output is quite obviously trashed. The indexes haven't been updated. They still have values which point to record numbers. The only problem is that those records no longer contain information which corresponds to the index value.

I need to point out that it is quite possible to crash when using a stale index file against a recently packed database. You could attempt to read a record number which doesn't exist in the database. It all really depends on what key you are using and how many records were deleted.

Scan down to the report generated after we called reIndex(). Notice that everything is back to the way you expect it to be.

If you use xBASE products long enough, you will eventually find yourself in a situation in which you need to pack a database. Packing a database is always a gamble. If you are in a production environment you will simply not know every index file every application created to view your data the a manner it chose. You also won' have any way to tell those applications they need to rebuild the index. It is the responsibility of the packer to contact all of the other users, not just let them crash.

1.14 Programming Assignment 5

Add a `pack_database()` method to `doeHistory()`. Don' t just call `pack()`, re-index BOTH indexes. You won' just be able to call `reIndex()`. If you read that code carefully you will see that it relies on all index files having been opened and attached already.

1.15 Data Integrity

We touched on this a bit in the last section, but I need to drive the point home now. I don' t care what xBASE library you use or what language you work in, without a database engine between every application and the actual data, you cannot and will not have data integrity.

Data integrity is much more than simply keeping indexes in synch with actual data records. Data integrity involves data rules and you cannot implement rules without an engine, VM, or some other single access point between the application and the actual data.

Don' worry, I' mnot going to bore you with a highly technical rant that sounds like a lecture on venereal disease. Data integrity is quite easy to explain in layman' s terms.

Let us say that you run a landfill. Every company which has signed a contract with you has provided a list of truck numbers along with other truck-identifying information. These are the only trucks allowed to dump at your landfill because the companies will pay for their own trucks only. You dutifully put all of this information into a DBF file using a unique segmented key of `companyId` and `truckNum`.

With that task out of the way you set about writing your scale ticket application. The inbound scale operator will pick a truck from the list you provide and key in the truck full (gross) weight. The system will fill in the date and time before writing the record to the transaction file. After the truck is dumped, it will pull onto the outbound scale where that operator will pull pick the ticket record from the list of available tickets based upon the truck number and company. Once the operator keys in the empty (tare) weight the system will print the ticket and the scale operator will hand a copy to the driver.

Have you noticed any problems yet?

You, the developer, had to write an application which enforced data integrity on its own. You didn't let the scale operator key in truck information, he or she had to pick from a list, presumably a list you populated from your truck and company database. The outbound scale operator had to choose one of the currently open tickets to complete.

There is nothing in the environment which would stop a ticket record from getting written to the ticket database with a truck and company that isn't on the truck and company database. Your application is providing that data rule, but the next programmer may not be aware of the rule.

Without a database engine providing a single point of access for all users and applications, there is no method of enforcing integrity rules. There is no requirement that this engine be relational, it simply needs to provide control and restrict access.

It might seem difficult to understand, but there are people out there in today's world paying hundreds and sometimes thousands of dollars for commercial xBASE products which provide this very thing. A run-time or virtual machine is installed under a different user ID which owns and controls all data and index files. The run-time coordinates all access to the data and in many cases will enforce data integrity rules. Some will even force the rebuilding of index files whenever a database file is packed.

We aren't dealing with an engine or a single point of access. If your application is going to have any form of data integrity then you are going to have to code it in.

1.16 Programming Assignment 6

This is more of a “learn it on your own” assignment. Pick any one of the databases we have covered which has a unique primary key defined. Write or modify a program which adds five records to that file. After adding five records, make sure the records appear when reporting via the unique key. Once they are there, delete three of those records, then attempt to add three records which have the same primary key value.

What happens?

If you manage to get the records added, what happens when you attempt to reIndex()?

How about when you try to undelete?

1.17 Summary

It should now be obvious that xBaseJ provides a lot of functionality for applications of limited size and scope. When all you need is an indexed file of some kind for a stand-alone application this can be a great tool. As you should have learned from the index problems we covered, it is not for multi-user use. You will find that most xBASE libraries out there only talk about being multi-user for the data file, not the indexes. In order to gain speed, most of these libraries load the index file contents into memory.

You can get a lot of speed out of an xBASE file format on today's computer hardware. Most of the concepts and original libraries were written to run on dual floppy computers running 4.77Mhz. Later re-implementations of these libraries used less efficient higher level languages, but most tried to honor the original specifications. Java is not a language known for performance, but on a machine with a 1Ghz or faster clock speed, it really doesn't have to be that efficient.

With all that it has going for it, a developer has to remember that xBASE was originally created to solve a data storage and ordered input problem, not provide the types of data services we associate with today's relational databases. The original creator could not have realistically envisioned all of the bastardized uses for this solution people would come up with in the following decades. Oddly enough, it is the horror stories from implementations that should have never been signed off on which gave xBASE its somewhat maligned reputation.

You might find it difficult to believe, but next to a CSV (Comma Separated Value) file, a DBF file is one of the most common methods of data exchange. While some core DBF formats are widely supported, NDX and MDX files are not widely supported. xBaseJ focuses on supporting the minimal core of DBF data types along with the memo fields. I don't really know why the Picture datatype was supported, unless that was simply fallout from adding Memo field support.

It would be nice if one or more of you reading this would take it upon yourselves to add support for Datetime (T) and Autoincrement (+) since those datatypes became rather widely used in later years. I haven't researched either subject, but I imagine that neither type will be easy to add until support has been added for native Integer (I).

I did not cover MDX support at this time because it doesn't work. Keys are added, but sort order is not maintained.

1.18 Review Questions

1. What two situations force a user or application to physically remove deleted records?
2. By default, what are string and character fields padded with when using xBaseJ?
3. If you have a DBF open with NDX files attached to it then call a subroutine which creates new NDX objects for those same files and calls reIndex() on them, will the changes to the index files be reflected in the NDX objects your DBF holds? Why or why not?
4. What two Java classes do you need to use to build create a report line making the data line up in columns?
5. How does one tell xBaseJ to pad string and character fields with spaces?
6. What DBF class method physically removes records from the database?
7. What is the maximum size of a DBF file?
8. What DBF class method is used to retrieve a value from a database Field regardless of field type?
9. After creating a shiny new DBF object and corresponding data file, what method do you use to actually create columns in the database?
10. What DBF class method is used to assign a value to a database Field?
11. What DBF class method do you call to change the NDX key of reference?
12. What DBF class method ignores all indexes and physically reads a specific record?
13. When you delete a database record, is it actually deleted?
14. What DBF class method sets the current record to zero and resets the current index pointer to the root of the current index?
15. What is the main difference between readNext() and findNext()?
16. What function or method returns the number of records on file?
17. What happens when you attempt to store a numeric value too large for the column?
18. What happens when you attempt to store a character value too large for the column?
19. When accessing via an index, how do you obtain the record occurring before the current record?
20. What DBF method returns the number of fields currently in the table?
21. When retrieving data from a database column, what datatype is returned?
22. What is the maximum length of a column name for most early xBASE formats?
23. What does the instanceof operator really tell you?
24. Are descending keys directly supported by xBaseJ?
25. What NDX method can you call to refresh index values stored in the NDX file?
26. What Java String method allows you to split a String into an array of Strings based upon a delimiting String?
27. Do NDX objects monitor database changes made by other programs or users?

28. Can you "undelete" a record in a DBF file? If so, why and for how long?
29. When a Numeric field is declared with a width of 6 and 3 decimal places, how many digits can exist to the left of the decimal when the field contains a negative value?
30. When do you need to create a finalize() method for your class?
31. What Java class provides the readLine() method to obtain a line of input from a text file or stream?
32. Do xBASE data files provide any built-in method of data integrity?
33. What must exist, no matter how the data is stored, to provide data integrity?

Chapter 2

Mega-Zillionaire Application

2.1 Why This Example?

Those of you who have read the other books in this series won't be the ones asking this question. I'm answering this question for the newcomers. Whenever I cover a new language or tool, I use this application to put the fundamental features through their paces. I try to implement the application as close as possible to the original implementation covered in ISBN-13 978-0-9770866-0-3. Besides being a good test application, having the same application developed over and over again using different tools and languages allows people who own legitimate copies of this book series to quickly transition between languages, tools, and platforms.

The application is not all that involved. It is a simple lottery tracking system which contains a primary data file ordered by drawing date and two statistics files ordered by element number. Admittedly, I used a very broad definition of the term *statistics* since we are simply keeping a few counts and a couple of percentages. The application does encompass the minimum of what you need to know about any toolset before you can realistically begin using it. You need to be able to create a menu, an import utility, an entry screen, a data browsing screen, and some reports. If you can do all of that, you can pretty much figure everything else out as you go.

Here are the three files:

Drawing_Data

Draw_dt	Date	k0
No_1	Numeric 2	
No_2	Numeric 2	
No_3	Numeric 2	
No_4	Numeric 2	
No_5	Numeric 2	
Mega_no	Numeric 2	

Drawing Stats

Elm_no	Numeric 2	k0
Hit_count	Numeric 6	
Last_draw_no	Numeric 6	
Since_last	Numeric 6	
Curr_seq	Numeric 4	
Longest_seq	Numeric 4	
Pct_hits	Numeric 8,4	
Max_btwn	Numeric 6	
Ave_btwn	Numeric 8,4	

Mega Stats

Elm_no	Numeric 2	k0
Hit_count	Numeric 6	
Last_draw_no	Numeric 6	
Since_last	Numeric 6	
Curr_seq	Numeric 4	
Longest_seq	Numeric 4	
Pct_hits	Numeric 8,4	
Max_btwn	Numeric 6	
Ave_btwn	Numeric 8,4	

Careful readers will notice that the layout for both stats files is the same. When I originally designed this application I didn't want to complicate an application I was going to use for several introductory texts. Yes, I have enough technical skills to merge both of our stats files into one with the addition of a "type" or "flag" byte to segregate the records. Exactly how many people who are brand new to software development or brand new to the concept of indexed files reading this (or any) book will have all of the fundamental skills needed to jump right into an application with a segmented primary key that spans two data types? Some storage systems won't even allow such a beast to exist.

Hopefully you spent some time working through programming assignment six from Chapter 1. If you did, then regardless of your prior skill level, you have a grasp on how convoluted this demonstration application would get when trying to create records in the stats files for the *n*th time. While it may not be sexy, a design which allows each statistics file to be deleted and created from scratch each time a user chooses to generate stats is both cleaner and more understandable. Let's face it: the first program you are assigned in college prints "Hello" to the printer, it doesn't solve the Tower of Hanoi problem. Eventually you get to the Tower of Hanoi problem, but you don't start there.

With the exception of the Logic book, the books in this series have all been aimed at professional programmers. I didn't give two full chapters of explanation prior to dropping this example on them. This book is primarily aimed at people who either have had one programming course covering Java or have read a "Teach Yourself How to Be Totally Useless in 21 Days or Less" type book and are looking to obtain actual skills. Because of this application the book will also be useful to anyone who owns the rest of the book series and needs to quickly get up to speed using xBaseJ, or even Java under Linux.

Unlike prior books in this series, this one is going to describe what the application looks like first, then we will discuss it. The main menu/form for this application looks much like many other applications implementing the CUA (Common User Access) interface. It has a main menu across the top, and those drop down when you click on the entries.

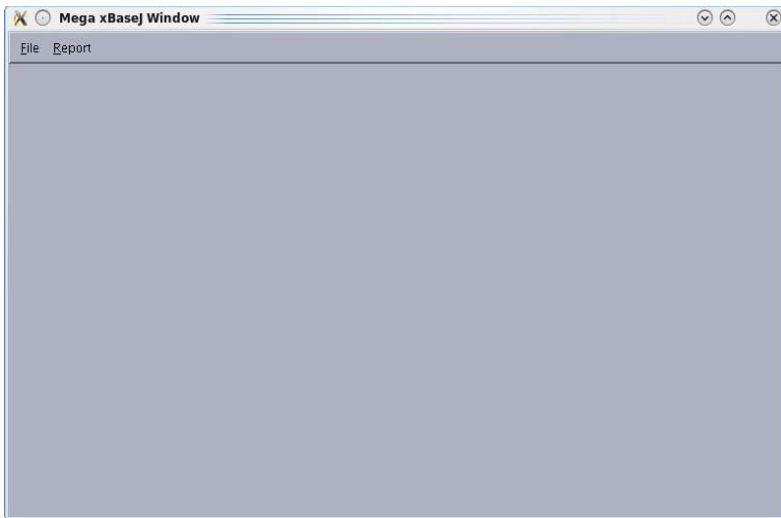


Figure 1 Main menu at startup

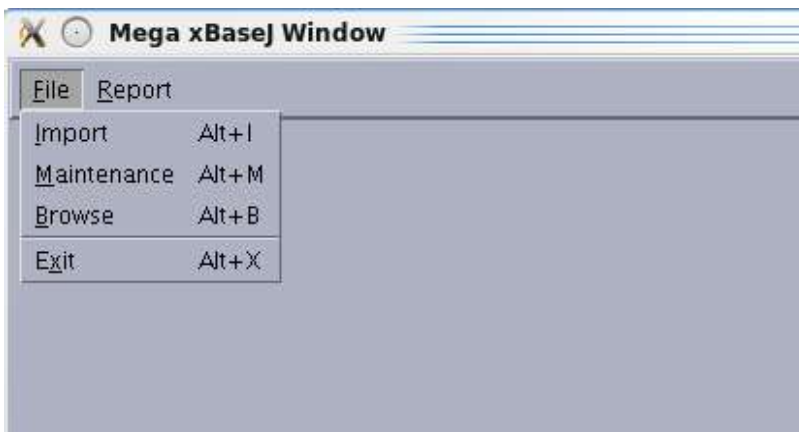


Figure 2 File menu contents

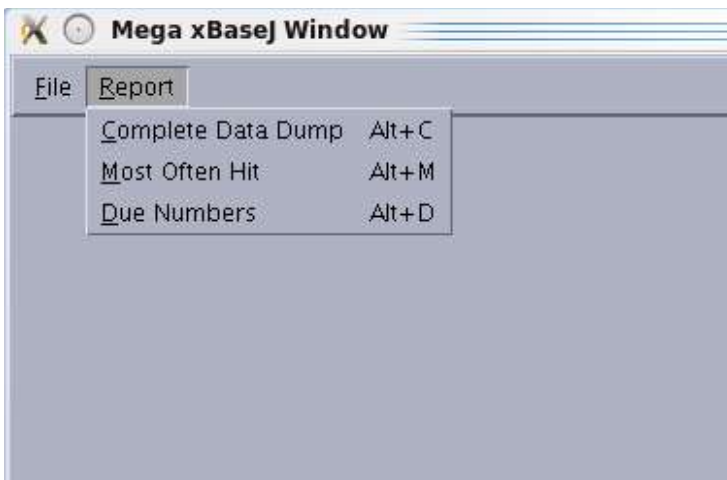


Figure 3 Report menu contents

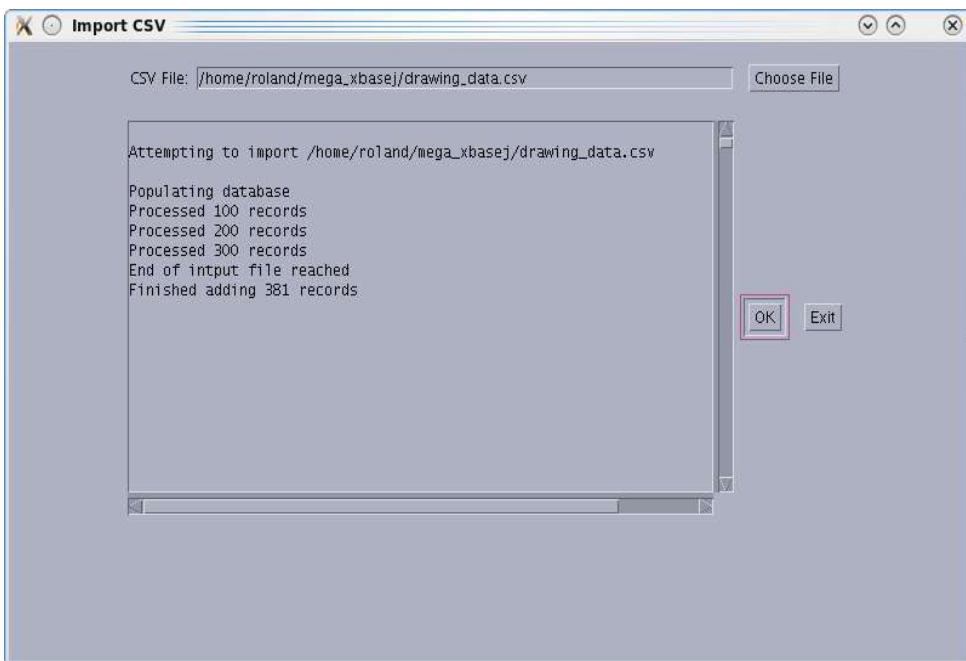


Figure 4 Import after file import

When you click on the “Choose File” button in the Import form, some flavor of the File Chooser window will pop-up. I say “some flavor” because it will depend upon what Look and Feel libraries you have installed. As we will see later, the startup code searches for a couple of widely known Look and Feel implementations before defaulting to the incredibly ugly Metal Look and Feel which is the default Java Look and Feel. In case you are wondering, this is the File Chooser displayed when a Motif-based Look and Feel has been selected. Many people (developers included) will visit <http://www.javootoo.com> to pull down various free Look and Feel packages. You will see how to change the Look and Feel when we cover the test module which runs this application.

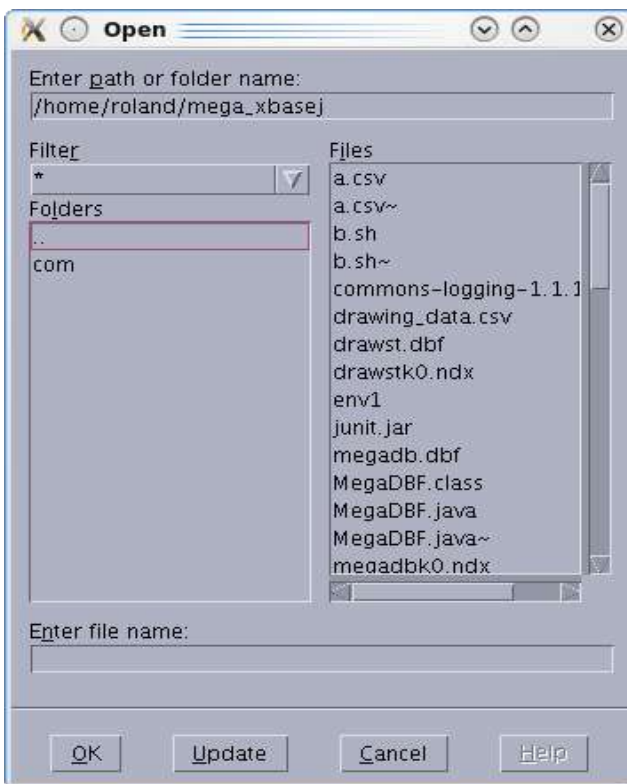


Figure 5 File Chooser used with import

Changing just a couple lines of code in our application (and making sure another JAR file is included) creates an application which looks like this:

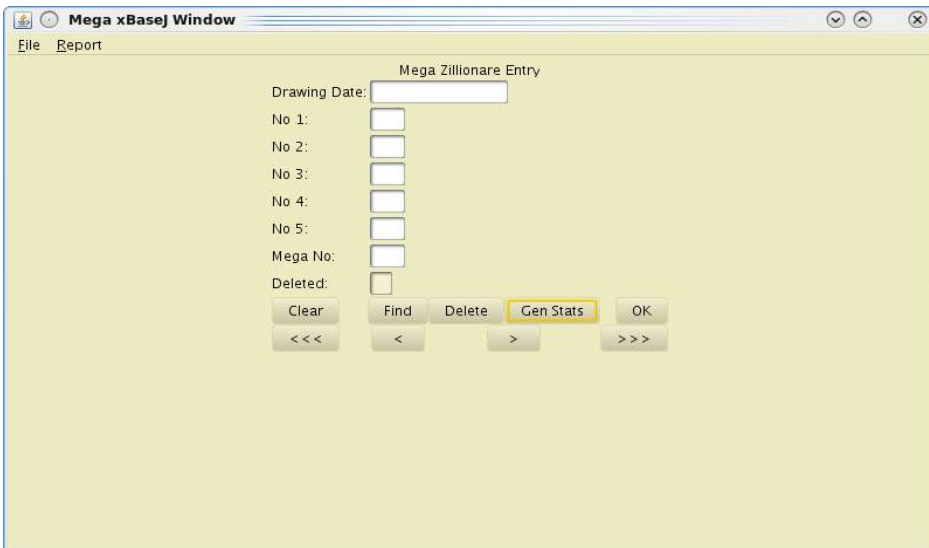


Figure 6 Entry form with Nimrod look and feel

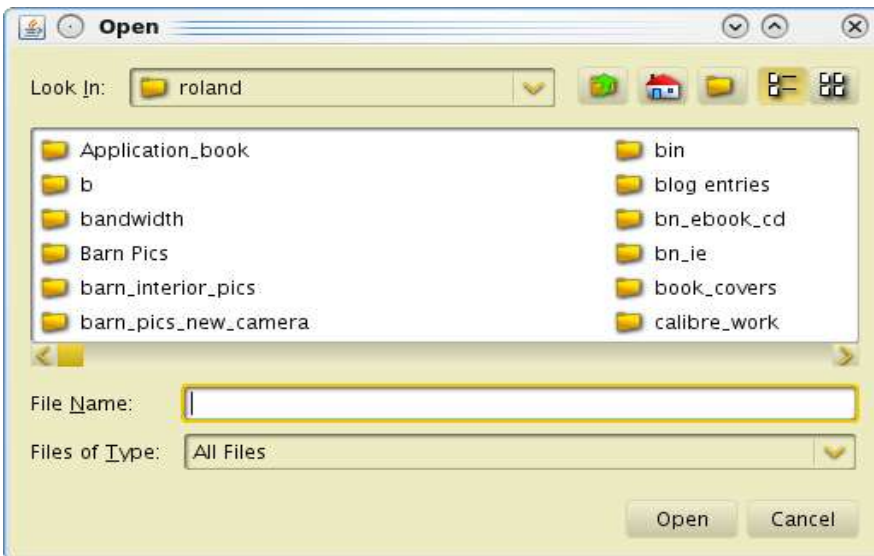
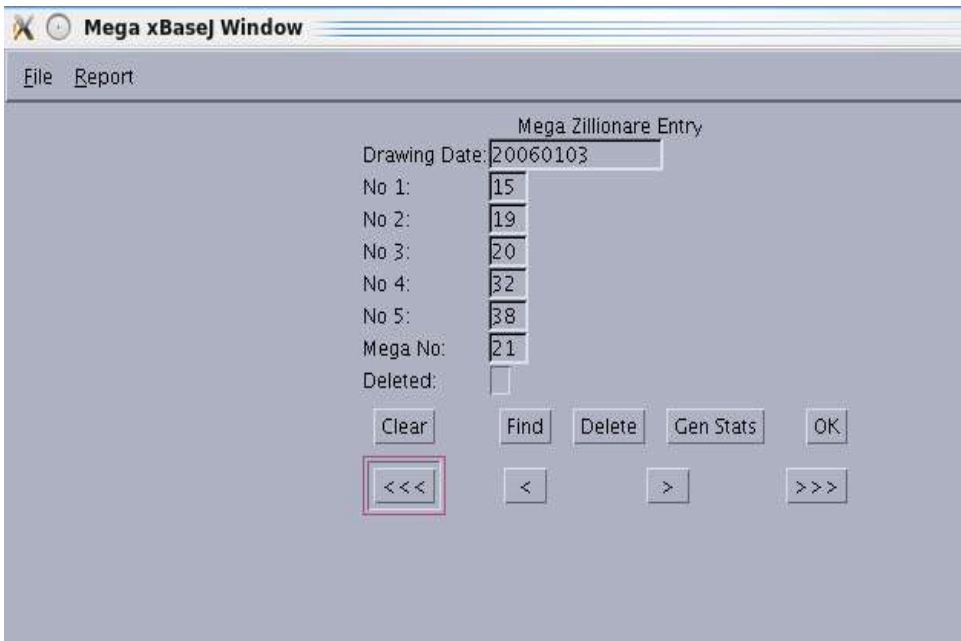


Figure 7 FileChooser with Nimrod look and feel

You might notice that it isn't only the coloring which changes, but the layout and style of things. The common Java tools like FileChooser also are dramatically different. Not all Look and Feels are available for all versions of Java.



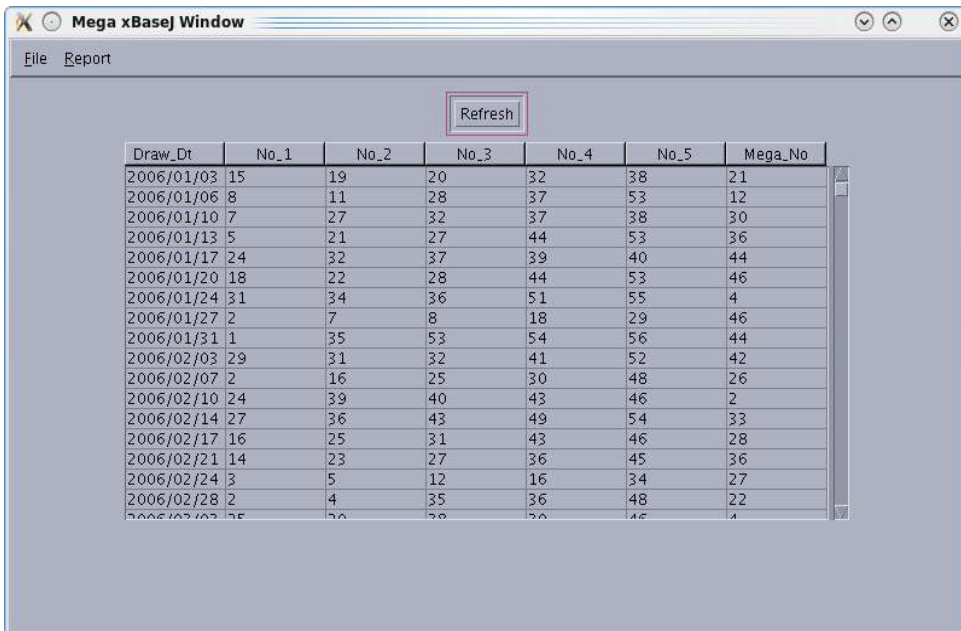
The screenshot shows a window titled "Mega xBaseJ Window" with a menu bar containing "File" and "Report". The main area is titled "Mega Zillionaire Entry" and contains the following fields and controls:

- Drawing Date: 20060103
- No 1: 15
- No 2: 19
- No 3: 20
- No 4: 32
- No 5: 38
- Mega No: 21
- Deleted: ☐
- Buttons: Clear, Find, Delete, Gen Stats, OK
- Navigation buttons: <<<, <, >, >>> (The <<< button is highlighted with a red box).

Figure 8 Entry form

One thing which might not be obvious is the “Deleted:” prompt. You cannot enter a value here, but when a record which has been flagged for deletion is displayed “*” will display in this box. Unless you use a lot of VCR-type software the row of buttons containing less-than and greater-than signs might not be intuitive. A single less-than sign moves towards the beginning, multiples move to the very beginning. A single greater-than moves towards the end and multiples move to the very end.

It probably wasn't the best choice of labels, but “OK” performs either an Add or an Update depending upon which mode you are currently in. There is no button to set the mode per se. If you find a record via the “Find” button or one of the navigation buttons, you will be in find mode. By default the screen starts out in add mode. If you need to get back to add mode you must “Clear,” which will both clear all entries on the screen and reset the screen back to add mode.



The screenshot shows a window titled "Mega xBaseJ Window" with a menu bar containing "File" and "Report". Below the menu bar is a "Refresh" button. The main area contains a spreadsheet with the following data:

Draw Dt	No_1	No_2	No_3	No_4	No_5	Mega_No
2006/01/03	15	19	20	32	38	21
2006/01/06	8	11	28	37	53	12
2006/01/10	7	27	32	37	38	30
2006/01/13	5	21	27	44	53	36
2006/01/17	24	32	37	39	40	44
2006/01/20	18	22	28	44	53	46
2006/01/24	31	34	36	51	55	4
2006/01/27	2	7	8	18	29	46
2006/01/31	1	35	53	54	56	44
2006/02/03	29	31	32	41	52	42
2006/02/07	2	16	25	30	48	26
2006/02/10	24	39	40	43	46	2
2006/02/14	27	36	43	49	54	33
2006/02/17	16	25	31	43	46	28
2006/02/21	14	23	27	36	45	36
2006/02/24	3	5	12	16	34	27
2006/02/28	2	4	35	36	48	22
2006/03/03	25	30	38	50	46	4

Figure 9 Browse form

Although it is against my religion to design applications which load every record from a database into a spreadsheet, that is what end users have come to expect thanks to the world's lowest quality software vendor, Microsoft. Nobody with even the tiniest shred of computer science education would ever consider getting users *used* to seeing data displayed this way. It works only under the condition which lets it work here: a very limited set of data stored locally and access read only. I did it because most of you were going to whine and snivel about wanting to do it.

Most of you reading this book will not have had professional software development training. I cover this topic quite a bit in the OpenVMS Application Developer book (ISBN-13 978-0-9770866-0-3) and the SOA book (ISBN-13 978-0-9770866-6-5). The spreadsheet design is horribly inefficient. I'm not talking about the code to create the spreadsheet itself, I'm talking about the concepts behind the design. It is a resource-intensive pig that imposes severe data access restrictions by requiring either exclusive access to the entire data set, or a live data monitor communicating with the database to monitor for any and all record changes.

The xBASE architecture doesn't lend itself to multi-user access without an intervening database engine locking all files and providing all access. We don't have that, so we are already living in multi-user Hell, and choose to handle the multi-user problem procedurally by not creating the data files on a server and telling users not to run multiple instances of our application.

It used to be easy to restrict applications to non-network file storage. You had to either be working at a large corporation or be an Uber Geek yourself to install and configure a Netware file server in your own small business or home. Then Microsoft came out with their own pathetic excuse for a Netware replacement, lowering the skill level and exponentially lowering the bar on quality. Today, even a blind squirrel can find the acorn. For well under \$1000 the average user can buy a network storage device, plug it in, follow a short list of configuration commands, and have their own file server. Security on these devices tends to be almost non-existent, given that they are created from a "share everything" viewpoint for non-technical users. Many of these devices cost under \$500 and provide nearly 1TB of storage. Unlike Netware, these file servers don't provide an indexed file system. Btrieve Technologies, Inc. really needs to get into this personal file server market. There are probably still a lot of tools out there which support Btrieve and let end users create things by picking and pointing.

Memory and bandwidth issues simply cannot be overlooked when designing an application. I provided only a few hundred records for our test database and I'm creating the files locally. What happens when you modify this application to open a DBF and NDX which are on a Web site or remote file server? Unless you are on dial-up, you probably have enough bandwidth to transfer fewer than 400 records. How about when the file is approaching 2GB and the end user is on a satellite connection with a 120MB per day bandwidth restriction? One must always take such things into consideration when designing an application or applet which could have its data hosted remotely.

Ordinarily, a screen like the browse screen would be designed to display five to ten records, and it would have a search prompt. Perhaps the search prompt would also have a combo box allowing a user to select a search field, otherwise the search would be on the primary key. When a user clicked on the Search or Find button the application would perform indexed look up logic against the database and display up to 5 records. While that design may not seem as slick as being able to drag a scroll bar through a spreadsheet, it works at all resource levels. The poor Schmoe who was given a corporate desktop running Windows XP with only 256Meg of RAM can use it just as easily as the power user and their 2+Ghz multi-core CPU with 4GB or RAM.

2.2 Supporting Classes

MegaDBF.java

```

1)  package com.logikal.megazillxBASEJ;
2)
3)  import java.io.*;
4)  import java.util.*;
5)  import org.xBaseJ.*;
6)  import org.xBaseJ.fields.*;
7)  import org.xBaseJ.Util.*;
8)  import org.xBaseJ.indexes.NDX;
9)
10) public class MegaDBF {
11)
12)     // variables used by the class
13)     //
14)     private DBF aDB = null;
15)
16)     // fields
17)     public DateField Draw_Dt = null;
18)     public NumField No_1 = null;
19)     public NumField No_2 = null;
20)     public NumField No_3 = null;
21)     public NumField No_4 = null;
22)     public NumField No_5 = null;
23)     public NumField Mega_No = null;
24)
25)     // file names
26)     public final String DEFAULT_DB_NAME = "megadb.dbf";
27)     public final String DEFAULT_K0_NAME = "megadbk0.ndx";
28)
29)     // work variables
30)     private boolean continue_flg = true;
31)     private boolean dbOpen      = false;
32)
33)     // result codes
34)     public static final int MEGA_SUCCESS      = 1;
35)     public static final int MEGA_DUPE_KEY     = 2;
36)     public static final int MEGA_KEY_NOT_FOUND = 3;
37)     public static final int MEGA_FILE_OPEN_ERR = 4;
38)     public static final int MEGA_DEVICE_FULL  = 5;
39)     public static final int MEGA_NO_CURRENT_REC = 6;
40)     public static final int MEGA_DELETE_FAIL  = 7;
41)     public static final int MEGA_GOTO_FAIL    = 8;
42)     public static final int MEGA_DB_CREATE_FAIL = 9;
43)     public static final int MEGA_INVALID_DATA = 10;
44)     public static final int MEGA_END_OF_FILE  = 11;
45)
46)
47)     ///////////////////////////////////////////////////
48)     // Method to populate known class level field objects.
49)     // This was split out into its own method so it could be used
50)     // by either the open or the create.
51)     ///////////////////////////////////////////////////
52)     private void attach_fields( boolean created_flg) {
53)         try {
54)             if ( created_flg) {
55)                 //Create the fields
56)                 Draw_Dt      = new DateField( "DrawDt");
57)                 No_1          = new NumField( "No1", 2, 0);
58)                 No_2          = new NumField( "No2", 2, 0);
59)                 No_3          = new NumField( "No3", 2, 0);

```



```

60)                No_4            = new NumField( "No4", 2, 0);
61)                No_5            = new NumField( "No5", 2, 0);
62)                Mega_No         = new NumField( "MegaNo", 2, 0);
63)
64)                //Add field definitions to database
65)                aDB.addField(Draw_Dt);
66)                aDB.addField(No_1);
67)                aDB.addField(No_2);
68)                aDB.addField(No_3);
69)                aDB.addField(No_4);
70)                aDB.addField(No_5);
71)                aDB.addField(Mega_No);
72)
73)            } else {
74)                Draw_Dt          = (DateField) aDB.getField("Drawdt");
75)                No_1             = (NumField) aDB.getField("No1");
76)                No_2             = (NumField) aDB.getField("No2");
77)                No_3             = (NumField) aDB.getField("No3");
78)                No_4             = (NumField) aDB.getField("No4");
79)                No_5             = (NumField) aDB.getField("No5");
80)                Mega_No          = (NumField) aDB.getField("MegaNo");
81)            }
82)
83)            } catch ( xBaseJException j){
84)                j.printStackTrace();
85)            } // end catch
86)            catch( IOException i){
87)                i.printStackTrace();
88)            } // end catch IOException
89)        } // end attach_fields method
90)
91)        //;;;;;;;;;;;;;
92)        // Method to close the database.
93)        // Don't print stack traces here. If close fails it is
94)        // most likely because the database was never opened.
95)        //;;;;;;;;;;;;;
96)        public void close_database() {
97)            if (!dbOpen)
98)                return;
99)            try {
100)                if (aDB != null) {
101)                    aDB.close();
102)                    dbOpen = false;
103)                }
104)            } catch (IOException i) {}
105)
106)        } // end close_database method
107)
108)        //;;;;;;;;;;;;;
109)        // Method to create a shiny new database
110)        //;;;;;;;;;;;;;
111)        public void create_database() {
112)            try {
113)                //Create a new dbf file
114)                aDB=new DBF(DEFAULT_DB_NAME,true);
115)
116)                attach_fields(true);
117)
118)                aDB.createIndex(DEFAULT_K0_NAME,"DrawDt",true,true);
119)                dbOpen = true;
120)            } catch ( xBaseJException j){
121)                System.out.println( "xBaseJ Error creating database");
122)                j.printStackTrace();

```

```

123)         continue_flg = false;
124)     } // end catch
125)     catch( IOException i){
126)         System.out.println( "IO Error creating database");
127)         i.printStackTrace();
128)         continue_flg = false;
129)     } // end catch IOException
130) } // end create_database method
131)
132)
133) //;;;;;;;;;;;;;
134) // method to retrieve a copy of the DBF object
135) //;;;;;;;;;;;;;
136) public DBF getDBF() {
137)     return aDB;
138) } // end getDBF method
139)
140) //;;;;;;;;;;;;;
141) // Method to test private flag and see
142) // if database has been successfully opened.
143) //;;;;;;;;;;;;;
144) public boolean isOpen() {
145)     return dbOpen;
146) } // end ok_to_continue method
147)
148) //;;;;;;;;;;;;;
149) // Method to open an existing database and attach primary key
150) //;;;;;;;;;;;;;
151) public int open_database() {
152)     int ret_val = MEGA_SUCCESS;
153)
154)     try {
155)
156)         //Create a new dbf file
157)         aDB=new DBF(DEFAULT_DB_NAME);
158)
159)         attach_fields( false);
160)
161)         aDB.useIndex( DEFAULT_K0_NAME);
162)         dbOpen = true;
163)         reIndex(); // gets around problem with stale index info
164)
165)     } catch( xBaseJException j){
166)         continue_flg = false;
167)     } // end catch
168)     catch( IOException i){
169)         continue_flg = false;
170)     } // end catch IOException
171)
172)     if (!continue_flg) {
173)         continue_flg = true;
174)         System.out.println( "Open failed, attempting create");
175)         create_database();
176)     } // end test for open failure
177)
178)     if (isOpen())
179)         return MEGA_SUCCESS;
180)     else
181)         return MEGA_FILE_OPEN_ERR;
182) } // end open_database method
183)
184) //;;;;;;;;;;;;;
185) // Method to re-index all of the associated index files.

```

```

186) //;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
187) public void reIndex() {
188)     if (aDB != null) {
189)         if (isOpen()) {
190)             try {
191)                 NDX n = null;
192)                 for (int i=1; i <= aDB.getIndexCount(); i++) {
193)                     n = (NDX) aDB.getIndex( i);
194)                     n.reIndex();
195)                 }
196)             } catch( xBaseJException j){
197)                 j.printStackTrace();
198)             } // end catch
199)             catch( IOException i){
200)                 i.printStackTrace();
201)             } // end catch IOException
202)         } // end test for open database
203)     } // end test for initialized database object
204) } // end reIndex method
205)
206) public int find_EQ_record( String d) {
207)     int ret_val = MEGA_SUCCESS;
208)     boolean perfect_hit;
209)
210)     if (!dbOpen)
211)         return MEGA_FILE_OPEN_ERR;
212)
213)     try {
214)         perfect_hit = aDB.findExact( d);
215)         if ( !perfect_hit) {
216)             System.out.println( "missed");
217)             System.out.println( "Current Record " +
aDB.getCurrentRecordNumber());
218)             ret_val = MEGA_KEY_NOT_FOUND;
219)         }
220)     } catch( xBaseJException j){
221)         System.out.println( j.getMessage());
222)         ret_val = MEGA_KEY_NOT_FOUND;
223)     } // end catch
224)     catch( IOException i){
225)         ret_val = MEGA_KEY_NOT_FOUND;
226)     } // end catch IOException
227)
228)     return ret_val;
229) } // end find_EQ_record method
230)
231) public int find_GE_record( String d) {
232)     int ret_val = MEGA_SUCCESS;
233)
234)     if (!dbOpen)
235)         return MEGA_FILE_OPEN_ERR;
236)
237)     try {
238)         aDB.find( d);
239)     } catch( xBaseJException j){
240)         ret_val = MEGA_KEY_NOT_FOUND;
241)     } // end catch
242)     catch( IOException i){
243)         ret_val = MEGA_KEY_NOT_FOUND;
244)     } // end catch IOException
245)
246)     return ret_val;
247) } // end find_GE_record method

```

```

248)
249) } // end class MegaDBF

```

This is a good example of the farthest most of you will go when writing your own classes for application re-use, be it re-use within the application or other applications. Once again you will see the open, close, and create methods have been provided. A hidden attach_fields() method ensures we always have the same field names. At the end of the source listing I added methods to find a matching key and find a key which is greater than or equal to a provided key value. I did not provide methods for deletion, but I did provide the reIndex() method. The nice thing about the reIndex() method is that you will probably cut and paste it into every DBF class you create. As long as you make the DBF variable name aDB this method will always work for you.

Listing lines 56 through 62 might just provide some interesting confusion. I chose the externally visible column names deliberately. They are consistent with the names used in the other books of this series. Given the naming restrictions xBASE enforces on column names, you will note that the actual physical column names don't match the externally visible. I did not want to experiment with trying to make the “_” character work as a column name. Some character sets for some countries use the “_” as the “delete” character. I ran into this years ago. When I'm working with something like a relational engine I will use the “_” in a column name. The engine protects me from the possibility of the character set changing.

We should discuss listing line 163 before moving on. I would like to say I originally designed this application in such a way as to insulate it from changes made by others. That was my original intention. I, however, did not originally have the reIndex() being called on each open. I had to do this because of a bug in the xBaseJ library which apparently I introduced while fixing another bug. Thankfully the original developer ran the debugger on an example I provided and found where things went bad. When you initially opened an NDX file, something was not loaded correctly with the index. You can get around this problem by moving a value, any value, to the Field object named in the key, or you can call reIndex(). Calling reIndex() at the time of open won't be a big burden on most applications having fewer than 5000 records. Today's computer speeds are fast enough that you probably won't even notice. Unless you are the only person using your application on your computer, you should always call reIndex() after opening an existing NDX file anyway.

StatElms.java

```

1) package com.logikal.megazillxBaseJ;
2)
3) public class StatElms extends Object {
4)     public int elmNo, hitCount, lastDrawNo, sinceLast, currSeq,
5)         longestSeq, maxBtwn;
6)     public double pctHits, aveBtwn;
7) } // end StatElms class

```

Other than changing the package name, this source file is unchanged from how it appeared in other books in the series. If you are unfamiliar with the shortcomings of Java, this class file will help point them out. When generating statistics, we need an array to hold a bunch of values. Some of these values are updated each time a number occurs in a drawing, others are updated only once we have processed all drawing records. Every 3GL the book series covers allows us to declare a record/structure containing only these fields, then create an array of that structure. Java doesn't understand the concept of records or data structures, as it is completely Object Oriented. OOP is good for some things, but for most standard data processing tasks, it fails. When you need a limited number of records containing a handful of fields OOP fails rather spectacularly. Ultimately, the contents of this array get written to one of the two statistics databases.

StatDBF.java

```

1)  package com.logikal.megazillxBASEJ;
2)
3)  import java.io.*;
4)  import java.util.*;
5)  import org.xBaseJ.*;
6)  import org.xBaseJ.fields.*;
7)  import org.xBaseJ.Util.*;
8)  import org.xBaseJ.indexes.NDX;
9)
10) public class StatDBF {
11)
12)     // variables used by the class
13)     //
14)     private DBF aDB = null;
15)
16)     // fields
17)     public NumField Elm_No = null;
18)     public NumField Hit_Count = null;
19)     public NumField Last_Draw_No = null;
20)     public NumField Since_Last = null;
21)     public NumField Curr_Seq = null;
22)     public NumField Longest_Seq = null;
23)     public NumField Pct_Hits = null;
24)     public NumField Max_Btwn = null;
25)     public NumField Ave_Btwn = null;
26)
27)     // file names
28)     public String DEFAULT_DB_NAME = null;
29)     public String DEFAULT_K0_NAME = null;
30)
31)     // work variables
32)     private boolean continue_flg = true;
33)     private boolean dbOpen      = false;
34)
35)     // result codes
36)     public static final int MEGA_SUCCESS          = 1;
37)     public static final int MEGA_DUPE_KEY        = 2;
38)     public static final int MEGA_KEY_NOT_FOUND    = 3;
39)     public static final int MEGA_FILE_OPEN_ERR    = 4;
40)     public static final int MEGA_DEVICE_FULL      = 5;
41)     public static final int MEGA_NO_CURRENT_REC   = 6;
42)     public static final int MEGA_DELETE_FAIL      = 7;
43)     public static final int MEGA_GOTO_FAIL        = 8;

```

```

44)     public static final int MEGA_DB_CREATE_FAIL = 9;
45)     public static final int MEGA_INVALID_DATA   = 10;
46)     public static final int MEGA_END_OF_FILE    = 11;
47)
48)
49)     //;;;;;;;;;;;;;
50)     //      Method to populate known class level field objects.
51)     //      This was split out into its own method so it could be used
52)     //      by either the open or the create.
53)     //;;;;;;;;;;;;;
54)     private void attach_fields( boolean created_flg) {
55)         try {
56)             if ( created_flg) {
57)                 //Create the fields
58)                 Elm_No      = new NumField( "ElmNo",      2, 0);
59)                 Hit_Count   = new NumField( "HitCount",   6, 0);
60)                 Last_Draw_No = new NumField( "LstDrwNo",   6, 0);
61)                 Since_Last  = new NumField( "SinceLst",    6, 0);
62)                 Curr_Seq    = new NumField( "CurrSeq",     4, 0);
63)                 Longest_Seq = new NumField( "LngstSeq",    4, 0);
64)                 Pct_Hits    = new NumField( "PctHits",     8, 4);
65)                 Max_Btwn    = new NumField( "MaxBtwn",     6, 0);
66)                 Ave_Btwn    = new NumField( "AveBtwn",     8, 4);
67)
68)                 //Add field definitions to database
69)                 aDB.addField(Elm_No);
70)                 aDB.addField(Hit_Count);
71)                 aDB.addField(Last_Draw_No);
72)                 aDB.addField(Since_Last);
73)                 aDB.addField(Curr_Seq);
74)                 aDB.addField(Longest_Seq);
75)                 aDB.addField(Pct_Hits);
76)                 aDB.addField(Max_Btwn);
77)                 aDB.addField(Ave_Btwn);
78)
79)             } else {
80)                 Elm_No      = (NumField) aDB.getField("ElmNo");
81)                 Hit_Count   = (NumField) aDB.getField("HitCount");
82)                 Last_Draw_No = (NumField) aDB.getField("LstDrwNo");
83)                 Since_Last  = (NumField) aDB.getField("SinceLst");
84)                 Curr_Seq    = (NumField) aDB.getField("CurrSeq");
85)                 Longest_Seq = (NumField) aDB.getField("LngstSeq");
86)                 Pct_Hits    = (NumField) aDB.getField("PctHits");
87)                 Max_Btwn    = (NumField) aDB.getField("MaxBtwn");
88)                 Ave_Btwn    = (NumField) aDB.getField("AveBtwn");
89)             }
90)
91)         } catch ( xBaseJException j){
92)             j.printStackTrace();
93)         } // end catch
94)         catch( IOException i){
95)             i.printStackTrace();
96)         } // end catch IOException
97)     } // end attach_fields method
98)
99)     //;;;;;;;;;;;;;
100)    //      Method to close the database.
101)    //      Don't print stack traces here.  If close fails it is
102)    //      most likely because the database was never opened.
103)    //;;;;;;;;;;;;;
104)    public void close_database() {
105)        if (!dbOpen)
106)            return;

```

```

107)         try {
108)             if (aDB != null) {
109)                 aDB.close();
110)                 dbOpen = false;
111)             }
112)         } catch (IOException i) {}
113)
114)     } // end close_database method
115)
116)     ///////////////////////////////////////////////////
117)     //      Method to create a shiny new database
118)     ///////////////////////////////////////////////////
119)     public void create_database( String dbf_name) {
120)         try {
121)             // Define default names
122)             DEFAULT_DB_NAME = dbf_name + ".dbf";
123)             DEFAULT_K0_NAME = dbf_name + "k0.ndx";
124)
125)             //Create a new dbf file
126)             aDB=new DBF(DEFAULT_DB_NAME,true);
127)
128)             attach_fields(true);
129)
130)             aDB.createIndex(DEFAULT_K0_NAME,"ElmNo",true,true);
131)             dbOpen = true;
132)         } catch( xBaseJException j){
133)             System.out.println( "xBaseJ Error creating database");
134)             j.printStackTrace();
135)             continue_flg = false;
136)         } // end catch
137)         catch( IOException i){
138)             System.out.println( "IO Error creating database");
139)             i.printStackTrace();
140)             continue_flg = false;
141)         } // end catch IOException
142)     } // end create_database method
143)
144)
145)     ///////////////////////////////////////////////////
146)     //      method to retrieve a copy of the DBF object
147)     ///////////////////////////////////////////////////
148)     public DBF getDBF() {
149)         return aDB;
150)     } // end getDBF method
151)
152)     ///////////////////////////////////////////////////
153)     //      Method to test private flag and see
154)     //      if database has been successfully opened.
155)     ///////////////////////////////////////////////////
156)     public boolean isOpen() {
157)         return dbOpen;
158)     } // end ok_to_continue method
159)
160)     ///////////////////////////////////////////////////
161)     //      Method to open an existing database and attach primary key
162)     ///////////////////////////////////////////////////
163)     public int open_database( String dbf_name) {
164)         int ret_val = MEGA_SUCCESS;
165)
166)         // Define default names
167)         DEFAULT_DB_NAME = dbf_name + ".dbf";
168)         DEFAULT_K0_NAME = dbf_name + "k0.ndx";
169)

```

```

170)         try {
171)
172)             //Create a new dbf file
173)             aDB=new DBF(DEFAULT_DB_NAME);
174)
175)             attach_fields( false);
176)
177)             aDB.useIndex( DEFAULT_K0_NAME);
178)             reIndex();
179)             dbOpen = true;
180)             reIndex();
181)
182)             } catch( xBaseJException j){
183)                 continue_flg = false;
184)             } // end catch
185)             catch( IOException i){
186)                 continue_flg = false;
187)             } // end catch IOException
188)
189)             if (!continue_flg) {
190)                 continue_flg = true;
191)                 System.out.println( "Open failed, attempting create");
192)                 create_database( dbf_name);
193)             } // end test for open failure
194)
195)             if (isOpen())
196)                 return MEGA_SUCCESS;
197)             else
198)                 return MEGA_FILE_OPEN_ERR;
199)         } // end open_database method
200)
201)         //;;;;;;;;;;;;;
202)         //      Method to re-index all of the associated index files.
203)         //;;;;;;;;;;;;;
204)         public void reIndex() {
205)             if (aDB != null) {
206)                 if (isOpen()) {
207)                     try {
208)                         NDX n = null;
209)                         for (int i=1; i <= aDB.getIndexCount(); i++) {
210)                             n = (NDX) aDB.getIndex( i);
211)                             n.reIndex();
212)                         }
213)                     } catch( xBaseJException j){
214)                         j.printStackTrace();
215)                     } // end catch
216)                     catch( IOException i){
217)                         i.printStackTrace();
218)                     } // end catch IOException
219)                 } // end test for open database
220)             } // end test for initialized database object
221)         } // end reIndex method
222)
223)
224)     } // end class StatDBF

```

You will notice that I didn't provide methods to find individual records via a key. I did provide the getDBF() method so a user of the class could do most of what he or she wanted. There was a reIndex() method provided to facilitate working around the previously mentioned bug, but otherwise, this class is pretty basic.

Please allow me to direct your attention to listing lines 119 through 123 and 163 through 168. You may recall that both statistics files have exactly the same layout. (If you don't recall this please flip back to the first page of the chapter and look at the proposed record layouts again.) The only good way of re-using a database class under those conditions is to make the open and create methods set the file names for the class.

As you progress in your professional career you will encounter many OOP programmers who consider a class operating in this manner complete sacrilege. They would be wrong. They would also not be production programmers. Please do not confuse someone who is always working on new development with someone who is a production programmer. New-development-only people create the mess production people have to clean up.

Classes which require a file name at the time of object creation and/or open a database/file resource at that time are two of the main hallmarks of bad application design. They seem oh-so-politically-correct when you are sitting there in school doing your homework, but they are a train wreck waiting to happen in real life. Please take a good look at the top of this source file where the data which will be global for the class instance is declared. Do you see a constructor anywhere after that global data which forces a file name to be supplied? For that matter, do you see a constructor declared?

Some of you are going to take the opportunity to open the source for the DBF class, or flip to the back of this document and point to the constructor documentation and cry foul. You aren't pointing out my error when you do that, you are pointing out the fact that you do not understand the difference between application design and library design. In particular, low-level library design. Why do you think the four example programs provided with the library itself are so hard coded and have to be run in a specific order? This is a low-level library, at least as far as xBASE access is concerned. While it hides some of the ugly details from you, it certainly doesn't provide much in the way of high-level interfaces.

I have seen C/C++ code generators which take a list of fields along with their data types and a list of index components, then generate a complete class for you. That class doesn't allow any low-level access to any portion of the data storage. Each field has its own uniquely named get() and set() methods and each key has its own find EQ, GT, LT methods. In many cases, the developer using the class has no idea where the data is stored or how it is stored. In effect, it is like those I/O modules I told you about earlier. The theory behind them is that the data storage could be changed without anyone having to change the application in any way. It never works in practice, but it is a nice theory.

You are the one responsible for creating the level of abstraction necessary for your project. It is up to you to decide the definition of “code re-use” in your environment.

Some of you will be hard-core coders who think that cutting and pasting from inside the editor is what the industry means when they say “code re-use.” Your view is that if only you know all of the modules a particular piece of code has been pasted into, then you can never be fired. You would be wrong, but that would be your view.

Others reading this book will take it upon themselves to write a database class generator like the one I described. You will also include `extract_to()` and `copy()` methods to allow developers to generate CSV files and safety copies of data. The stars may align and you may choose to add your creation to the project, thus improving it.

Most of you will fall someplace in the middle of those two extremes. If you hadn't found a copy of this book you would have probably tried to stumble through via the cut and paste method, but now you have some file-based DBF classes as a starting point. They will start out as direct copies or just subsets of the classes I have provided, but as your coding and needs increase, so will the “default” methods you provide. You will eventually come to realize that the real value of this book isn't just the library explanation, but the instructions concerning how to design applications.

MegaXDueElms.java

```

1) package com.logikal.megazillxBaseJ;
2)
3) import java.util.*;
4)
5)
6) import com.logikal.megazillxBaseJ.StatElms;
7)
8) public class MegaXDueElms extends StatElms
9)     implements Comparable {
10)
11)     //
12)     // method to cause sort in Descending order
13)     // based upon how many drawings it has been since
14)     // it hit. Number is only "due" if it is past its average.
15)     //
16)     public int compareTo( Object o2) {
17)         MegaXDueElms s2 = (MegaXDueElms) o2;
18)         double o1_value, o2_value;
19)         int ret_val=0;
20)
21)         o1_value = this.aveBtwn - this.sinceLast;
22)         o2_value = s2.aveBtwn - s2.sinceLast;
23)
24)         if ( o2_value > o1_value)
25)             ret_val = 1;
26)         else if ( o2_value < o1_value)
27)             ret_val = -1;
28)
29)         return ret_val;
30)     } // end compare method
31)

```

```
32)     public boolean equals( MegaXDueElms m) {
33)         boolean ret_val = false;
34)
35)         if (this.elmNo == m.elmNo)
36)             if (this.hitCount == m.hitCount)
37)                 if (this.lastDrawNo == m.lastDrawNo)
38)                     if (this.sinceLast == m.sinceLast)
39)                         if (this.currSeq == m.currSeq)
40)                             if (this.longestSeq == m.longestSeq)
41)                                 if (this.maxBtwn == m.maxBtwn)
42)                                     ret_val = true;
43)
44)         return ret_val;
45)     } // end equals method
46)
47) } // end MegaXDueElms class
```

You didn't really think I was going to let you off with only one rant about the shortcomings of OOP, did you? I must admit that in Java 1.6 things got a bit better, but you can't simply code in Java 1.6 since Java 1.4 is still the most widely used in the field. Because of that, I had to extend the StatElms class just to create a class which could implement the Comparable interface. Technically I didn't have to code the equals() method since I didn't call it, but compareTo() is required.

The major downside to this design is that I have to create a separate class for every sort order needed. I'm not talking about an individual sort compare function for each order, but an actual class that I create an array of and load data into.

Some documentation claims that the Comparator interface started with 1.4.2. It may have, but on my machine I couldn't get it to compile when using the 1.4 switch. A Comparator object would have allowed me to have a separate object for each sorting of the StatElms array, but would not require different arrays and copies of data. In that case, the class would have encompassed about as much code, but you would have had less work to do with your assignments later in this chapter. Had we been using version 5 or higher we could have done this.

DueSortCompare.java

```

1) package com.logikal.megazillxBASEJ;
2)
3) import java.util.*;
4)
5)
6) import com.logikal.megazillxBASEJ.StatElms;
7)
8) //;;;;;;;;;;;;;
9) // EXAMPLE ONLY
10) // Code not actually used in application
11) //;;;;;;;;;;;;;
12) public class DueSortCompare
13)     implements Comparator<StatElms> {
14)
15)     //
16)     // method to cause sort in Descending order
17)     // based upon how many drawings it has been since
18)     // it hit. Number is only "due" if it is past its average.
19)     //
20)     public int compare( StatElms s1, StatElms s2) {
21)         double o1_value, o2_value;
22)         int ret_val=0;
23)
24)         o1_value = s1.aveBtwn - s1.sinceLast;
25)         o2_value = s2.aveBtwn - s2.sinceLast;
26)
27)         if ( o2_value > o1_value)
28)             ret_val = 1;
29)         else if ( o2_value < o1_value)
30)             ret_val = -1;
31)
32)         return ret_val;
33)     } // end compare method
34)
35) } // end DueSortCompare class

```

The call to `Arrays.sort()` would have had the `Comparator` object provided as an additional parameter rather than expecting the object array to implement the `Comparable` interface. It is hard to show just how resource-intensive this shortcoming is without having an application that already squeezes system resources and needing the data sorted multiple ways. What if the data file which fed this array was over 1GB in size and we needed to sort it three different ways for a report? Even if we directly copied from one array element to another, we would consume at least 2GB of RAM making the copy, then we would have to *hope* garbage collection quickly got around to reclaiming the original array so we could make our next copy. This, of course, assumes that we had at least 2GB of RAM available to the Java VM.

I didn't use the `equals()` method in this application, but we should talk about it a bit. Some of you may be horrified to see if statements nested that deep, but, in truth, it is the simplest way to implement such a method. If any one of those fields didn't match, we could stop looking. You may have noticed that I omitted checking either of the double fields for being equal, but did you hazard a guess as to why?

Java isn't a perfect language. The VM brings with it much of the baggage we have seen throughout the years with floating point numbers. True, it reduced the problem to exactly two floating point implementations, float and double. It is also true that those two data types use two different IEEE standards to help reduce problems with porting the VM to various platforms. Even given all of that, we still have floating point baggage to deal with. (A language called DIBOL used Binary Coded Decimal or BCD to do floating point math while early xBASE formats stored everything in character format to completely sidestep the issue.)

The main baggage problem we have to deal with is the fact that 1.234 will not equal 1.234 most of the time. The IEEE floating point standards are approximations. They introduce precision errors and rely on rounding rules of the output utilities to correct these errors. Depending upon how you arrived at 1.234 it may be 1.2344123 or 1.2344389. To us humans it will be displayed as 1.234, but, to a binary equality test like `==` the two values are not equal.

You don't know enough about the application yet, but `pctHits` and `AveBtwn` are calculated based upon the total number of drawings and the integer values we have already tested. The total number of drawings will be an integer which is the same for all elements and is not stored on the file. The beauty of this reality is that we only have to compare the integers to see if the elements match.

Our `MegaXDueElms` array is used to calculate the values of our Due report. I have not provided a screen shot of this report, but we will discuss its logic anyway.

2.3 The Panels

MegaXbaseDuePanel.java

```
1) package com.logikal.megazillxBaseJ;
2)
3)
4) import java.awt.*;
5) import java.awt.event.*;
6) import javax.swing.*;
7) import java.util.*;
8) import java.text.*;
9) import java.lang.Integer;
10)
11) import org.xBaseJ.*;
12) import org.xBaseJ.fields.*;
13) import org.xBaseJ.Util.*;
14)
15) import com.logikal.megazillxBaseJ.MegaXDueElms;
16) import com.logikal.megazillxBaseJ.StatDBF;
17)
18) // You need to import the java.sql package to use JDBC
19) //
20) import java.sql.*;
21) import java.io.*;
22)
```

```

23)
24)
25) public class MegaXbaseDuePanel extends JPanel
26)     implements ActionListener {
27)
28)     public final int ELM_COUNT = 57;    // highest number is 56 but I don't
29)                                         // feel like messing with zero
30)
31)     private JPanel      mainPanel;
32)     private JScrollPane sp;
33)     private JButton     refreshButton;
34)     private JTextArea   dueRptArea;
35)
36)
37)     //;;;;;;;;;
38)     // Constructor
39)     //;;;;;;;;;
40)     public MegaXbaseDuePanel( ) {
41)         mainPanel = new JPanel( new GridBagLayout());
42)         GridBagConstraints gbc = new GridBagConstraints();
43)
44)         // Add our refresh button first
45)         // This way we have an object to find the root panel of
46)         //
47)         JPanel buttonPanel = new JPanel();
48)         refreshButton = new JButton("Refresh");
49)         refreshButton.addActionListener( this);
50)         buttonPanel.add( refreshButton, BorderLayout.NORTH);
51)         gbc.anchor      = GridBagConstraints.NORTH;
52)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
53)         mainPanel.add( buttonPanel, gbc);
54)
55)         dueRptArea = new JTextArea();
56)         // Gives you a fixed width font.
57)         dueRptArea.setFont(new Font("Courier", Font.PLAIN, 12));
58)         dueRptArea.setEditable( false);
59)         dueRptArea.setTabSize(4);
60)         dueRptArea.setColumns( 80);
61)         dueRptArea.setRows(120);
62)         dueRptArea.setDoubleBuffered( true);
63)         sp = new JScrollPane( dueRptArea);
64)         sp.setPreferredSize( new Dimension( 500, 300));
65)
66)
67)         mainPanel.add( sp);
68)         add(mainPanel);
69)         setVisible( true);
70)     } // end constructor
71)
72)
73)     public void actionPerformed(ActionEvent event) {
74)         System.out.println( "Entered action event");
75)
76)         generateReport();
77)
78)     } // end actionPerformed method
79)
80)
81)     //;;;;;
82)     // Method which does all of the interesting work.
83)     //;;;;;
84)     private void generateReport() {
85)         StatDBF dDB = new StatDBF();

```

```

86)         StatDBF mDB = new StatDBF();
87)         MegaXDueElms d_elms[] = new MegaXDueElms[ELM_COUNT];
88)         MegaXDueElms m_elms[] = new MegaXDueElms[ELM_COUNT];
89)         //
90)         // These are needed to format the detail lines on the report
91)         //
92)
93)         dueRptArea.setText(""); // clear out in case this isn't first run
94)
95)         try {
96)             // load the array
97)             dDB.open_database("drawst");
98)             for (int i=1; i < ELM_COUNT; i++) {
99)                 dDB.getDBF().gotoRecord( i);
100)
101)                 d_elms[i] = new MegaXDueElms();
102)
103)                 d_elms[i].elmNo = Integer.parseInt(dDB.Elm_No.get().trim());
104)                 d_elms[i].hitCount = Integer.parseInt(dDB.Hit_Count.get
105)                 ().trim());
106)                 d_elms[i].lastDrawNo = Integer.parseInt(dDB.Last_Draw_No.get
107)                 ().trim());
108)                 d_elms[i].sinceLast = Integer.parseInt(dDB.Since_Last.get
109)                 ().trim());
110)                 d_elms[i].currSeq = Integer.parseInt(dDB.Curr_Seq.get().trim
111)                 ());
112)                 d_elms[i].longestSeq = Integer.parseInt(dDB.Longest_Seq.get
113)                 ().trim());
114)                 d_elms[i].maxBtwn = Integer.parseInt(dDB.Max_Btwn.get().trim
115)                 ());
116)                 d_elms[i].pctHits = Double.parseDouble(dDB.Pct_Hits.get
117)                 ().trim());
118)                 d_elms[i].aveBtwn = Double.parseDouble(dDB.Ave_Btwn.get());
119)             }
120)
121)             System.out.println("Finished loading array");
122)             // finished with this database
123)             dDB.close_database();
124)
125)             // Sort the array.
126)             Arrays.sort( d_elms, 1, ELM_COUNT-1);
127)
128)             // generate report
129)             DateFormat heading_format = DateFormat.getDateInstance
130)             ( DateFormat.SHORT);
131)             NumberFormat nf_elm      = NumberFormat.getInstance();
132)             NumberFormat nf_hits     = NumberFormat.getInstance();
133)             NumberFormat nf_since    = NumberFormat.getInstance();
134)             NumberFormat nf_pct      = NumberFormat.getInstance();
135)             NumberFormat nf_ave      = NumberFormat.getInstance();
136)
137)             nf_elm.setMinimumIntegerDigits(2);
138)             nf_pct.setMinimumFractionDigits(3);
139)             nf_ave.setMinimumFractionDigits(3);
140)
141)             Calendar c                = Calendar.getInstance();
142)             // obtain current date
143)             //
144)             c.setTime( new java.util.Date());
145)
146)             dueRptArea.append( "Date: " + heading_format.format( c.getTime()
147)
148)
149)
150)
151)
152)
153)
154)
155)
156)
157)
158)
159)
160)
161)
162)
163)
164)
165)
166)
167)
168)
169)
170)
171)
172)
173)
174)
175)
176)
177)
178)
179)
180)
181)
182)
183)
184)
185)
186)
187)
188)
189)
190)
191)
192)
193)
194)
195)
196)
197)
198)
199)
200)
201)
202)
203)
204)
205)
206)
207)
208)
209)
210)
211)
212)
213)
214)
215)
216)
217)
218)
219)
220)
221)
222)
223)
224)
225)
226)
227)
228)
229)
230)
231)
232)
233)
234)
235)
236)
237)
238)
239)
240)
241)
242)
243)
244)
245)
246)
247)
248)
249)
250)
251)
252)
253)
254)
255)
256)
257)
258)
259)
260)
261)
262)
263)
264)
265)
266)
267)
268)
269)
270)
271)
272)
273)
274)
275)
276)
277)
278)
279)
280)
281)
282)
283)
284)
285)
286)
287)
288)
289)
290)
291)
292)
293)
294)
295)
296)
297)
298)
299)
300)
301)
302)
303)
304)
305)
306)
307)
308)
309)
310)
311)
312)
313)
314)
315)
316)
317)
318)
319)
320)
321)
322)
323)
324)
325)
326)
327)
328)
329)
330)
331)
332)
333)
334)
335)
336)
337)
338)
339)
340)
341)
342)
343)
344)
345)
346)
347)
348)
349)
350)
351)
352)
353)
354)
355)
356)
357)
358)
359)
360)
361)
362)
363)
364)
365)
366)
367)
368)
369)
370)
371)
372)
373)
374)
375)
376)
377)
378)
379)
380)
381)
382)
383)
384)
385)
386)
387)
388)
389)
390)
391)
392)
393)
394)
395)
396)
397)
398)
399)
400)
401)
402)
403)
404)
405)
406)
407)
408)
409)
410)
411)
412)
413)
414)
415)
416)
417)
418)
419)
420)
421)
422)
423)
424)
425)
426)
427)
428)
429)
430)
431)
432)
433)
434)
435)
436)
437)
438)
439)
440)
441)
442)
443)
444)
445)
446)
447)
448)
449)
450)
451)
452)
453)
454)
455)
456)
457)
458)
459)
460)
461)
462)
463)
464)
465)
466)
467)
468)
469)
470)
471)
472)
473)
474)
475)
476)
477)
478)
479)
480)
481)
482)
483)
484)
485)
486)
487)
488)
489)
490)
491)
492)
493)
494)
495)
496)
497)
498)
499)
500)

```

```

140)                                + "Due NumbersReport\n");
141)
142)                                dueRptArea.append("\n                                Regular
Drawing Numbers\n\n");
143)                                dueRptArea.append(" NO      Hits   Since      Pct_Hits      Ave_Btwn
\n");
144)                                dueRptArea.append(" --      ----   -----      -
\n");
145)
146)                                String detail_line=null;
147)                                int l_x = 1;
148)                                while ( l_x < ELM_COUNT ) {
149)                                    if ((double)d_elms[l_x].sinceLast > d_elms[l_x].aveBtwn) {
150)                                        detail_line = " " + nf_elm.format( d_elms[ l_x].elmNo)
151)                                            + " " + nf_hits.format( d_elms[ l_x].hitCount)
152)                                            + " " + nf_since.format( d_elms[l_x].sinceLast)
153)                                            + " " + nf_pct.format( d_elms[ l_x].pctHits)
154)                                            + " " + nf_ave.format( d_elms[ l_x].aveBtwn)
155)                                            + "\n";
156)                                        dueRptArea.append( detail_line);
157)                                    }
158)                                    l_x++;
159)                                } // end while loop
160)
161)                                dueRptArea.append( "\n\n");
162)                                dueRptArea.append( "\n\n=====
\n");
163)                                updateText();
164)
165)                                //-----
166)                                // Now process the Mega numbers
167)                                //-----
168)                                // load the array
169)                                mDB.open_database( "megast");
170)                                for (int i=1; i < ELM_COUNT; i++) {
171)                                    mDB.getDBF().gotoRecord( i);
172)                                    m_elms[i] = new MegaXDueElms();
173)                                    m_elms[i].elmNo = Integer.parseInt(mDB.Elm_No.get().trim());
174)                                    m_elms[i].hitCount = Integer.parseInt(mDB.Hit_Count.get
().trim());
175)                                    m_elms[i].lastDrawNo = Integer.parseInt(mDB.Last_Draw_No.get
().trim());
176)                                    m_elms[i].sinceLast = Integer.parseInt(mDB.Since_Last.get
().trim());
177)                                    m_elms[i].currSeq = Integer.parseInt(mDB.Curr_Seq.get().trim
());
178)                                    m_elms[i].longestSeq = Integer.parseInt(mDB.Longest_Seq.get
().trim());
179)                                    m_elms[i].maxBtwn = Integer.parseInt(mDB.Max_Btwn.get().trim
());
180)                                    m_elms[i].pctHits = Double.parseDouble(mDB.Pct_Hits.get
().trim());
181)                                    m_elms[i].aveBtwn = Double.parseDouble(mDB.Ave_Btwn.get
().trim());
182)                                }
183)
184)                                // finished with this database
185)                                mDB.close_database();
186)
187)                                // Sort the array.
188)                                Arrays.sort( m_elms, 1, ELM_COUNT-1);
189)
190)                                // generate report

```



```

191)
192)         dueRptArea.append( "Date: " + heading_format.format( c.getTime()
193)         )
194)         + "\n"
195)         + "Due NumbersReport\n");
196)         dueRptArea.append("\n"
197) Numbers\n\n");
198)         dueRptArea.append(" NO      Hits   Since      Pct_Hits      Ave_Btwn
199) \n");
200)         dueRptArea.append(" --      ----  -----  -----  -----
201) \n");
202)         l_x = 1;
203)         while ( l_x < ELM_COUNT ) {
204)             if ((double)m_elms[l_x].sinceLast > m_elms[l_x].aveBtwn) {
205)                 detail_line = " " + nf_elm.format( m_elms[ l_x].elmNo)
206)                 + " " + nf_hits.format( m_elms[ l_x].hitCount)
207)                 + " " + nf_since.format( m_elms[l_x].sinceLast)
208)                 + " " + nf_pct.format( m_elms[ l_x].pctHits)
209)                 + " " + nf_ave.format( m_elms[ l_x].aveBtwn)
210)                 + "\n";
211)                 dueRptArea.append( detail_line);
212)             }
213)             l_x++;
214)         } // end while loop
215)         dueRptArea.append( "\n\n");
216)         dueRptArea.setCaretPosition(0); // scroll back to top
217)         } catch (xBaseJException x) {
218)         } catch (NumberFormatException n) {
219)             n.printStackTrace();
220)         } catch (IOException e) {
221)         }
222)     } // end generateReport method
223)
224)     public void updateText() {
225)         mainPanel.invalidate();
226)         mainPanel.validate();
227)         mainPanel.paintImmediately( mainPanel.getVisibleRect());
228)         mainPanel.repaint();
229)     } // end updateText method
230)
231)
232) } // end MegaXbaseDuePanel class

```

You will be getting two assignments based upon this module, so I'm going to take the discussion a bit slower than I have been for the other modules. With the exception of the import function, all of the other menu options are implemented as panels which display on the main menu panel using CardLayout. The CardLayout allows you to stack 1-N panels on top of each other like a deck of cards which are face up. You then shuffle the card you want to the top of the face up deck so it is displayed. All of the other cards are left in their current state when you change the displayed card. When you choose to re-display one of the cards you do not create a new instance of it or re-initialize its contents in any way; it is simply shown again as it was last seen.

Each panel which gets placed into the CardLayout can, and does, have its own layout. It could actually have many different layouts on it, but we have only one layout in use because we don't have much to display.

This particular panel has only a refresh button to display at the top of the screen and a text area which gets displayed inside of a scroll pane. In truth, it looks much like the browse screen snapshot I've provided you, only it has regular text instead of a spreadsheet inside of the scroll pane.

Our panel uses a GridBagLayout. Don't ask me to tell you where the name came from or all of the details. GridBagLayout is one of the few functional layouts existing in Java, at least in the days of version 1.4. The topic of layouts has become so heated and debated that the C++ cross platform GUI library named Qt has released a Java wrapper for its library and many Java developers have begun migrating away from Swing. When it comes to the subject of layout managers and Java, it looks like the subject was "tabled until later" and later still hasn't come.

You position objects in a GridBagLayout via a GridBagConstraints object. Ordinarily you will fill in only the anchor and gridwidth values, leaving the rest of the GridBagConstraints fields at their default values. Normally gridwidth is a numeric constant such as 1 or 2, but it can be a couple of "special" values. One such value is REMAINDER as shown on listing line 52. This tells the layout manager the object is the last one on the current line of objects it is building.

The anchor portion of GridBagConstraints has a lot of direction-sounding constants, which can be a little confusing. Most people assume that all of the NORTH-based constraints have something to do with the top of the display and the SOUTH-based constraints have something to do with the bottom of the display. In general, this is true, but not as true as it sounds. To start with, the enclosing object, in this case our JPanel, can change the component orientation and instead of left-to-right-top-to-bottom the screen might be displayed right-to-left-bottom-to-top. In any case, NORTH is associated with the starting point and SOUTH with the finishing point. We will discuss this topic in greater detail when we cover the Entry screen. I just wanted to get you thinking about it now. Layouts in Java are not an easy subject to discuss in a book. Some readers will have ten years of coding under their belts and a few hundred source templates saved on disk; others will have only written HelloWorld.java.

We create our text area at listing lines 55 through 62, then place it in a scroll pane at listing line 63. Most of what I did was pretty self-explanatory. We do, however, need to talk about the `setFont()` call. I chose to use the font name "Courier." Most systems which have some kind of Web browser will have some kind of "Courier" font installed on them. When you are creating columnar reports you need to have a fixed-width font so those columns have a chance at lining up.

Most Java purists reading this will scream that I should have used the logical font name “Monospaced” which is required to be implemented by all Java VMs. The simple truth is that “Monospaced” is not required to be “implemented,” it is required to have a physical font mapped to it. That font may have absolutely nothing to do with fixed width or monospace. Even Courier is not a fixed-width font when dealing with TrueType fonts. At certain point sizes things will line up, but it won’ be perfect. Ultimately, the font you choose to use is up to you. I chose a font which works well on most platforms. If it doesn’ work well for you, change the source and recompile it.

The workhorse of this class is the `generateReport()` method. Here we read each record from each stat file, save the values into our arrays, sort our arrays, then print our report into the text area. You will note that I call `updateText()` from time to time. Whenever you call `append()` to add text to a `JTextArea`, the text is added to the in memory object, but an event is only queued to update any associated on-screen display. The display manager in Java waits until it “thinks” the system is idle to update the display, or finds itself forced to update the display. The lines of code in `updateText()` force the display manager to consolidate all of the updates and display them. This step does slow down processing, so you should do it only at points in time when you feel the user must see the progress which has been made.

I need to point out one tiny little thing at listing line 103. You may not grasp why I called `trim()` after calling `get()`. The `parseInt()` static method throws exceptions if the numeric string you hand it contains spaces. I don’ know why it doesn’ call `trim()` on its own, but it doesn’ .t As you can see by listing line 111, `parseDouble()` managed to handle things just fine.

Listing lines 123 through 131 contain something I truly hate about Java 1.4 and earlier. The `NumberFormat` object is very primitive. It does provide methods to set the minimum number of fractional digits, and minimum number of integer digits, but it has no concept of justification, fill character, or display width. If you try to set both the integer and fraction digits for a column, it will zero fill on the front and force the display to look something like the following.

NO	Hits	Since	Pct_Hits	Ave_Btwn
30	0,035	00,010	000.092	009.886
16	0,036	00,010	000.094	009.583
52	0,041	00,009	000.108	008.293
23	0,027	00,014	000.071	013.111
25	0,040	00,010	000.105	008.525
48	0,042	00,010	000.110	008.071
43	0,038	00,011	000.100	009.026
45	0,031	00,014	000.081	011.290
03	0,030	00,015	000.079	011.700
46	0,044	00,011	000.116	007.659
04	0,031	00,016	000.081	011.290

Not exactly what I would call human-friendly output. We discussed the `Formatter` class on page 54. This class added something which was required to bring Java into the business world, the ability to create a columnar report. Not one of you would pay your credit card bill if the text swam all over the page, but that is just what the Java developers at Sun thought we should put up with until Java 1.5 came out. Our output will look as follows, and we will live with it for now.

Mega Numbers				
NO	Hits	Since	Pct_Hits	Ave_Btwn
---	----	-----	-----	-----
02	13	11	0.000	10.222
27	7	15	0.000	11.333
22	13	16	0.000	9.946
32	7	18	0.000	11.333
31	7	19	0.000	10.686
08	6	22	0.000	12.931
19	5	23	0.000	11.750
05	6	21	0.000	9.146
37	8	27	0.000	14.346
15	10	25	0.000	11.242
14	7	24	0.000	9.590

Please note that we add a `newLine` character at listing line 155. We are not ‘printing’ to the text area, we are appending. It is our responsibility to insert the appropriate number of `newLine` characters at the appropriate places.

Let's now discuss the call to `sort()` at listing lines 119 and 188. I needed to pass in the second and third parameter because I chose to use elements 1-56 instead of 0-55. The zero element was never filled in and I didn't want to have stale garbage influencing the outcome of the sort. I have already discussed the fact that I implemented `Comparable` with our object because the compiler wouldn't let me implement a `Comparator` object when compiling for version 1.4 targets. In theory, I guess the makers of Java 1.4 did you a favor. Some of your assignments will be much more cut and paste than I like.

Listing line 215 is a trick you really need to know. The problem with text areas is that when you get done writing to them, the display is at the bottom of them, not the top. You need to get back to the top of the display so the user isn't completely lost. This little statement handles that. It resets the cursor position back to offset zero. This forces the eventual screen update showing the top of the text area.

While it is probably more code than you wanted to look at, the Due report isn't all that complex. I've already shown you how to create similar reports to the screen using `xBaseJ` and `DBF` files. All you had to learn here was how to create a panel with a text area. Once you knew that, you could simply walk down the databases, sort the data, and print the report.

MegaXbaseBrowsePanel.java

```

1) package com.logikal.megazillxBaseJ;
2)
3) import java.io.*;
4) import java.awt.*;
5) import java.awt.event.*;
6) import javax.swing.*;
7) import java.util.*;
8) import java.text.*;
9)
10) import org.xBaseJ.*;
11) import org.xBaseJ.fields.*;
12) import org.xBaseJ.Util.*;
13) import org.xBaseJ.indexes.NDX;
14)
15) import com.logikal.megazillxBaseJ.MegaDBF;
16)
17)
18) public class MegaXbaseBrowsePanel extends JPanel
19)     implements ActionListener {
20)
21)     private JPanel        mainPanel;
22)     private JScrollPane  sp;
23)     private JButton       refreshButton;
24)     private JTable        drawTable;
25)     private DateFormat    file_date_format = new SimpleDateFormat( "yyyyMMdd");
26)     private DateFormat    out_date_format = new SimpleDateFormat( "yyyy/MM/dd");
27)
28)
29)     final static    String columnNames [] = {"Draw_Dt", "No_1", "No_2",
30)                                             , "No_3", "No_4", "No_5", "Mega_No"};
31)
32)     //;;;;;;;;;
33)     // Constructor
34)     //;;;;;;;;;
35)     public MegaXbaseBrowsePanel( ) {
36)         mainPanel = new JPanel( new GridBagLayout());
37)         GridBagConstraints gbc = new GridBagConstraints();
38)
39)         // Add our refresh button first
40)         // This way we have an object to find the root panel of
41)         //
42)         JPanel buttonPanel = new JPanel();
43)         refreshButton = new JButton("Refresh");
44)         refreshButton.addActionListener( this);
45)         buttonPanel.add( refreshButton, BorderLayout.NORTH);
46)         gbc.anchor = GridBagConstraints.NORTH;
47)         gbc.gridwidth = GridBagConstraints.REMAINDER;
48)         mainPanel.add( buttonPanel, gbc);
49)
50)         Object tData [][] = new Object [1][7];    // dummy table.
51)
52)         drawTable = new JTable( tData, columnNames);
53)         drawTable.setAutoResizeMode( JTable.AUTO_RESIZE_ALL_COLUMNS);
54)         sp = new JScrollPane(drawTable);
55)         sp.setPreferredSize( new Dimension( 500, 300));
56)
57)         mainPanel.add( sp);
58)         add(mainPanel);
59)         setVisible( true);
60)     } // end constructor
61)

```

```

62)
63) private Object[][] fetchTableData( ) {
64)     int     l_record_count=0, l_x, l_y, l_try_count;
65)     String   serverResponse=null;
66)
67)     MegaDBF aDB = new MegaDBF();
68)     aDB.open_database();
69)
70)     DBF d = aDB.getDBF();
71)
72)     l_record_count = d.getRecordCount();
73)     System.out.println( "Record count " + l_record_count);
74)
75)     // declare an array to fill based upon rows in table
76)     //
77)     Object tableData [][] = new Object[ l_record_count] [7];
78)
79)
80)     // Fill our new array"
81)     //
82)     try {
83)         l_x = 0;
84)         d.startTop();
85)         while ( l_x < l_record_count) {
86)             try {
87)                 d.findNext();
88)
89)                 tableData[ l_x] [0] = out_date_format.format(
90)                                     file_date_format.parse
( aDB.Draw_Dt.get() ));
91)                 tableData[ l_x] [1] = new Integer( aDB.No_1.get().trim());
92)                 tableData[ l_x] [2] = new Integer( aDB.No_2.get().trim());
93)                 tableData[ l_x] [3] = new Integer( aDB.No_3.get().trim());
94)                 tableData[ l_x] [4] = new Integer( aDB.No_4.get().trim());
95)                 tableData[ l_x] [5] = new Integer( aDB.No_5.get().trim());
96)                 tableData[ l_x] [6] = new Integer( aDB.Mega_No.get().trim());
97)
98)             } catch(ParseException p) {
99)                 l_x = l_record_count + 1;
100)                 System.out.println( p.toString());
101)             }
102)
103)             l_x++;
104)
105)         } // end while loop
106)         System.out.println( "processed " + l_x + " rows");
107)     }
108)     catch( IOException s) {
109)         l_x = l_record_count + 1;
110)         JRootPane m = (JRootPane)
111)             SwingUtilities.getAncestorOfClass( JRootPane.class,
refreshButton);
112)         if ( m != null)
113)         {
114)             JOptionPane.showMessageDialog(m, s.toString(), "Browse",
JOptionPane.ERROR_MESSAGE);
115)         }
116)         else
117)             System.out.println( "m was null");
118)     }
119)
120)     catch( xBaseJException j) {
121)         l_x = l_record_count + 1;
122)         JRootPane m = (JRootPane)

```

```

123)         SwingUtilities.getAncestorOfClass( JRootPane.class,
refreshButton);
124)         if ( m != null)
125)         {
126)             JOptionPane.showMessageDialog(m, j.toString(), "Browse",
127)                                           JOptionPane.ERROR_MESSAGE);
128)         }
129)         else
130)             System.out.println( "m was null");
131)     }
132)
133)     aDB.close_database();
134)
135)     return tableData;
136) } // end fetchTableData method
137)
138)
139)
140) public void actionPerformed(ActionEvent event) {
141)     System.out.println( "Entered action event");
142)     mainPanel.setVisible( false);
143)     mainPanel.remove( sp);
144)
145)     //
146)     // Build a new table and scroll panel
147)     //
148)     Object tData [] [] = fetchTableData();
149)     drawTable = new JTable( tData, columnNames);
150)     sp = new JScrollPane(drawTable);
151)     sp.setPreferredSize( new Dimension( 600, 300));
152)     mainPanel.add( sp);
153)     mainPanel.setVisible(true);
154) } // end actionPerformed method
155)
156)
157) public void updateText() {
158)     mainPanel.invalidate();
159)     mainPanel.validate();
160)     mainPanel.paintImmediately( mainPanel.getVisibleRect());
161)     mainPanel.repaint();
162) }
163) } // end MegaXbaseBrowsePanel class definition

```

Other than creating and manipulating a `JTable` object, the code for this panel doesn't work much differently from the Due report panel. I'm not fond of listing lines 50 through 55, but I had to have them. Remember my earlier rant about requiring values to instantiate? This is a great example of how that gets you into trouble. I had to create a useless table so the screen would be somewhat self-explanatory when a user first sees it.

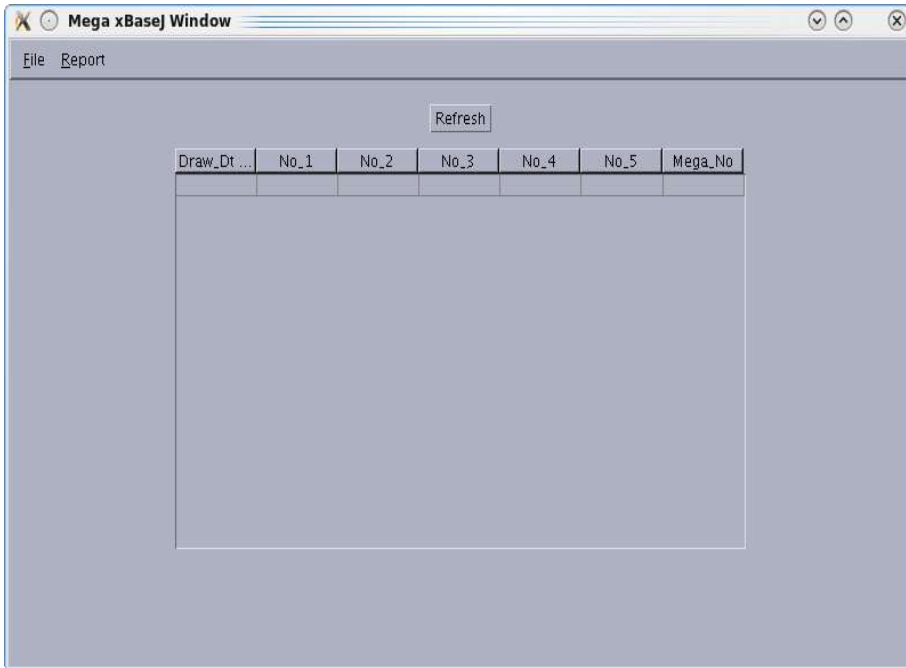


Figure 10 Empty Browse window

If I didn' put an empty table on the screen a user' first thought would be 'Refresh what?' when they saw the screen. This wouldn' be so bad if you could cleanly add data to it, but there wasn' t a clean way to achieve that along with the refresh concept.

Why do I need the refresh concept? This panel attaches to the database, loads all of the rows, closes the database, then displays the spreadsheet. It does all of that when the user clicks the refresh button. It has to wait for the user to click that button because the very first time the application is run there won' be a database. Even if there was a database, I would still need the refresh button so the user can verify that records he or she adds via the Entry panel are actually in the database along with all of the other data.

Listing line 63 might look a bit odd if you haven' done much with arrays in Java. This private method returns a newly allocated two dimensional array of objects. While I could have hard-coded the second dimension at the method level, I opted to leave it at the point of allocation. We have seven fields, so we need seven columns, but until we obtain the record count from the database, we have no idea how many rows are needed.

Notice how I loaded the array. At listing line 70 I obtain a reference to the internal DBF object. At listing line 84 I use that DBF object to force index positioning to the beginning. Inside of the while loop at listing line 87 I use the DBF object again to call `findNext()`.

Why did I go to all of this trouble? Because I didn't add a `startTop()` or `findNext()` wrapper method to the `MegaDBF.java` source file, and you cannot pass a null string or a string of all spaces to a DBF find method when a date or numeric key is involved. I wanted the data to appear in sorted order. While it is possible to sort a table after it is loaded, that is not the kind of code I want to write when compiling against a 1.4 target. Java 6 added `RowSorter` and `TableRowSorter` to the language to make sorting data in a table much easier. You should spend some time reading up on those capabilities.

Please look at listing lines 122 through 130. It's not a lot of code and most example programs you find won't show you how to do it. The result is that most example programs show a nice GUI which writes all of its errors to a terminal window a user is supposedly monitoring. This little code snippet pops up a message dialog when an error happens. Depending on the class of error indicated (error, warning, informational, etc.) a different dialog displays. Under normal circumstances you will also get whatever has been configured as the associated system sound for that type of message.

Listing lines 148 through 153 contain the code which actually performs the refresh function. We call `fetchTableData()` to create a new dynamic array of Objects. Once we have that we create a new `JTable` object then wrap it in a shiny new `JScrollPane` and display it. I tweaked the preferred size so the date column would display completely without requiring a user to manually resize it.

We actually never call the `updateText()` method. That is just a method I carry around from panel class to panel class.

MegaXbaseEntryPanel.java

```
1) package com.logikal.megazillxBaseJ;
2)
3)
4) import java.awt.*;
5) import java.awt.event.*;
6) import javax.swing.*;
7) import java.util.*;
8) import java.text.*;
9)
10) import org.xBaseJ.*;
11) import org.xBaseJ.fields.*;
12) import org.xBaseJ.Util.*;
13) import org.xBaseJ.indexes.NDX;
14)
15) import com.logikal.megazillxBaseJ.MegaDBF;
16) import com.logikal.megazillxBaseJ.StatElms;
```

```

17) import com.logikal.megazillxBASEJ.StatDBF;
18)
19) // You need to import the java.sql package to use JDBC
20) //
21) import java.sql.*;
22) import java.io.*;
23)
24)
25)
26) public class MegaXbaseEntryPanel extends JPanel
27)     implements ActionListener {
28)
29)     public final int ELM_COUNT = 57;    // highest number is 56 but I don't
30)                                         // feel like messing with zero
31)
32)     public final int FIND_MODE = 1;
33)     public final int ENTRY_MODE = 0;
34)
35)     private JFormattedTextField drawingDate;
36)     private JFormattedTextField no1;
37)     private JFormattedTextField no2;
38)     private JFormattedTextField no3;
39)     private JFormattedTextField no4;
40)     private JFormattedTextField no5;
41)     private JFormattedTextField megaNo;
42)     private JTextField deletedFlg;
43)
44)     private JButton    okButton;
45)     private JButton    findButton;
46)     private JButton    genStatsButton;
47)     private JButton    deleteButton;
48)     private JButton    clearButton;
49)     private JButton    nextButton;
50)     private JButton    prevButton;
51)     private JButton    firstButton;
52)     private JButton    lastButton;
53)     private Connection conn;
54)
55)     private int currMode;
56)
57)     private StatElms drawNoStat[], megaNoStat[];
58)     private int l_draw_no;
59)     private Integer zeroInt;
60)     private String errorMsg;
61)
62)     private DateFormat out_format = new SimpleDateFormat( "yyyyMMdd");
63)
64)     private MegaDBF    megaDBF=null;
65)
66)     //
67)     // Fields to hold previous record values from Find
68)     //
69)     private int lNo1, lNo2, lNo3, lNo4, lNo5, lMegaNo;
70)     private java.util.Date dDrawingDate;
71)     private String draw_dt_str;
72)
73)     public MegaXbaseEntryPanel( ) {
74)
75)         //
76)         // This internal class handles verification of numeric
77)         // input for JTextField
78)         //
79)         InputVerifier verifier = new InputVerifier() {

```

```

80)         public boolean verify(JComponent comp) {
81)             int currVal=0;
82)             boolean returnValue;
83)             JFormattedTextField textField = (JFormattedTextField)comp;
84)             try {
85)                 currVal = Integer.parseInt(textField.getText());
86)                 returnValue = true;
87)             } catch (NumberFormatException e) {
88)                 Toolkit.getDefaultToolkit().beep();
89)                 returnValue = false;
90)             }
91)             if (returnValue == true) {
92)                 if ( currVal < 1 || currVal >= ELM_COUNT)
93)                     returnValue = false;
94)             }
95)             return returnValue;
96)         }
97)
98)         public boolean shouldYieldFocus(JComponent input) {
99)             verify(input);
100)            return true;
101)        }
102)    };
103)
104)    zeroInt = new Integer( 0);
105)
106)    JPanel controlPanel=new JPanel();
107)    controlPanel.setLayout( new GridBagLayout());
108)    GridBagConstraints gbc = new GridBagConstraints();
109)
110)    gbc.anchor        = GridBagConstraints.NORTH;
111)    gbc.gridwidth      = GridBagConstraints.REMAINDER;
112)
113)    JLabel panelTitle=new JLabel("Mega Zillionare Entry");
114)    controlPanel.add( panelTitle, gbc);
115)
116)    //
117)    //  Our Date prompt
118)    //
119)    gbc.anchor        = GridBagConstraints.WEST;
120)    gbc.gridwidth      = 1;
121)    JLabel dateLabel   = new JLabel( "Drawing Date:");
122)    controlPanel.add( dateLabel);
123)    drawingDate        = new JFormattedTextField( new DecimalFormat
124)    ("#####"));
125)    drawingDate.setColumns( 10);
126)    gbc.gridwidth      = GridBagConstraints.REMAINDER;
127)    controlPanel.add( drawingDate, gbc);
128)
129)    //
130)    //  Prompts for the drawing numbers
131)    //
132)    gbc.anchor        = GridBagConstraints.WEST;
133)    gbc.gridwidth      = 1;
134)    JLabel nolLabel = new JLabel( "No 1:");
135)    controlPanel.add( nolLabel, gbc);
136)    gbc.gridwidth      = GridBagConstraints.REMAINDER;
137)    nol = new JFormattedTextField(new DecimalFormat("##"));
138)    nol.setInputVerifier( verifier);
139)    nol.setColumns(2);
140)    controlPanel.add( nol, gbc);
141)

```

```
142)         gbc.anchor      = GridBagConstraints.WEST;
143)         gbc.gridwidth   = 1;
144)         JLabel no2Label = new JLabel( "No 2:");
145)         controlPanel.add( no2Label, gbc);
146)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
147)         no2 = new JFormattedTextField( new DecimalFormat("##"));
148)         no2.setInputVerifier( verifier);
149)         no2.setColumns(2);
150)         controlPanel.add( no2, gbc);
151)
152)
153)         gbc.anchor      = GridBagConstraints.WEST;
154)         gbc.gridwidth   = 1;
155)         JLabel no3Label = new JLabel( "No 3:");
156)         controlPanel.add( no3Label, gbc);
157)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
158)         no3 = new JFormattedTextField( new DecimalFormat("##"));
159)         no3.setInputVerifier( verifier);
160)         no3.setColumns(2);
161)         controlPanel.add( no3, gbc);
162)
163)
164)         gbc.anchor      = GridBagConstraints.WEST;
165)         gbc.gridwidth   = 1;
166)         JLabel no4Label = new JLabel( "No 4:");
167)         controlPanel.add( no4Label, gbc);
168)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
169)         no4 = new JFormattedTextField(new DecimalFormat("##"));
170)         no4.setInputVerifier( verifier);
171)         no4.setColumns(2);
172)         controlPanel.add( no4, gbc);
173)
174)
175)         gbc.anchor      = GridBagConstraints.WEST;
176)         gbc.gridwidth   = 1;
177)         JLabel no5Label = new JLabel( "No 5:");
178)         controlPanel.add( no5Label, gbc);
179)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
180)         no5 = new JFormattedTextField(new DecimalFormat("##"));
181)         no5.setInputVerifier( verifier);
182)         no5.setColumns(2);
183)         controlPanel.add( no5, gbc);
184)
185)
186)         gbc.anchor      = GridBagConstraints.WEST;
187)         gbc.gridwidth   = 1;
188)         JLabel megaNoLabel = new JLabel( "Mega No:");
189)         controlPanel.add( megaNoLabel, gbc);
190)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
191)         megaNo = new JFormattedTextField(new DecimalFormat("##"));
192)         megaNo.setInputVerifier( verifier);
193)         megaNo.setColumns(2);
194)         controlPanel.add( megaNo, gbc);
195)
196)         gbc.anchor      = GridBagConstraints.WEST;
197)         gbc.gridwidth   = 1;
198)         JLabel deletedFlgLabel = new JLabel( "Deleted:");
199)         controlPanel.add( deletedFlgLabel, gbc);
200)         gbc.gridwidth   = GridBagConstraints.REMAINDER;
201)         deletedFlg = new JtextField(1);
202)         deletedFlg.setEditable(false);
203)         controlPanel.add( deletedFlg, gbc);
204)
```

```
205)        //
206)        //   The Clear Button
207)        //
208)        gbc.anchor        = GridBagConstraints.SOUTHWEST;
209)        gbc.gridwidth     = 1;
210)        clearButton = new JButton("Clear");
211)        clearButton.addActionListener( this);
212)        controlPanel.add( clearButton, gbc);
213)
214)        //
215)        //   The Find Button
216)        //
217)        gbc.anchor        = GridBagConstraints.SOUTHWEST;
218)        gbc.gridwidth     = 1;
219)        findButton = new JButton("Find");
220)        findButton.addActionListener( this);
221)        controlPanel.add( findButton, gbc);
222)
223)        //
224)        //   The Delete Button
225)        //
226)        gbc.anchor        = GridBagConstraints.SOUTHWEST;
227)        gbc.gridwidth     = 1;
228)        deleteButton = new JButton("Delete");
229)        deleteButton.addActionListener( this);
230)        controlPanel.add( deleteButton, gbc);
231)
232)        //
233)        //   The Gen Stats Button
234)        //
235)        gbc.anchor        = GridBagConstraints.SOUTH;
236)        gbc.gridwidth     = 1;
237)        genStatsButton = new JButton("Gen Stats");
238)        genStatsButton.addActionListener( this);
239)        controlPanel.add( genStatsButton, gbc);
240)
241)        //
242)        //   The OK Button
243)        //
244)        gbc.anchor        = GridBagConstraints.SOUTHEAST;
245)        gbc.gridwidth     = GridBagConstraints.REMAINDER;
246)        okButton = new JButton("OK");
247)        okButton.addActionListener( this);
248)        controlPanel.add( okButton, gbc);
249)
250)
251)        //
252)        //   The First Button
253)        //
254)        gbc.anchor        = GridBagConstraints.LINE_START;
255)        gbc.gridwidth     = 1;
256)        gbc.weightx       = 1.0;
257)        firstButton = new JButton("<<<");
258)        firstButton.addActionListener( this);
259)        controlPanel.add( firstButton, gbc);
260)
261)        //
262)        //   The Prev Button
263)        //
264)        gbc.anchor        = GridBagConstraints.SOUTH;
265)        gbc.gridwidth     = 1;
266)        gbc.weightx       = 0.0;
267)        prevButton = new JButton("< ");
```



```
331)        }
332)        return;
333)    } // end test for deleteButton
334)
335)    if ( e.getSource() == findButton) {
336)        currMode = FIND_MODE;
337)        if ( findRecord() == false)
338)            display_error_msg( "Drawing Not Found");
339)        return;
340)    } // end test for findButton
341)
342)    if ( e.getSource() == genStatsButton) {
343)        createStatsTable();
344)        return;
345)    } // end test for clear button
346)
347)    if ( e.getSource() == clearButton) {
348)        currMode = ENTRY_MODE;
349)        draw_dt_str = " ";
350)        lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
351)        no1.setValue( null);
352)        no2.setValue( null);
353)        no3.setValue( null);
354)        no4.setValue( null);
355)        no5.setValue( null);
356)        megaNo.setValue( null);
357)        deletedFlg.setText( "");
358)        drawingDate.setValue( null);
359)    } // end test for clear button
360)
361)    if ( e.getSource() == firstButton) {
362)        currMode = FIND_MODE;
363)        if ( firstRecord() == false)
364)            display_error_msg( "Drawing Not Found");
365)        return;
366)    } // end test for firstButton
367)
368)    if ( e.getSource() == prevButton) {
369)        if ( prevRecord() == false)
370)            display_error_msg( "Drawing Not Found");
371)        return;
372)    } // end test for prevButton
373)
374)    if ( e.getSource() == nextButton) {
375)        if ( nextRecord() == false)
376)            display_error_msg( "Drawing Not Found");
377)        return;
378)    } // end test for nextButton
379)
380)    if ( e.getSource() == lastButton) {
381)        currMode = FIND_MODE;
382)        if ( lastRecord() == false)
383)            display_error_msg( "Drawing Not Found");
384)        return;
385)    } // end test for lastButton
386)
387) } // end actionPerformed method
388)
389)
390) /*;;;;;
391) * method to add a record to the database
392) *;;;;;
393) */
```

```

394)     private boolean addRecord() {
395)         boolean retVal = false;
396)         int         l_x;
397)         String      ins_str;
398)         String      localDateStr=null;
399)         // Obtain values from the panel
400)         //
401)         draw_dt_str = drawingDate.getText();
402)         pad_draw_dt_str();
403)
404)         if (!is_record_valid())
405)             return false;
406)
407)
408)         try {
409)             localDateStr = out_format.format( out_format.parse(draw_dt_str));
410)         }
411)         catch( ParseException p) {
412)             display_error_msg( "Error parsing date" + p.toString());
413)         }
414)
415)         if ( localDateStr == null)
416)             return false;
417)
418)         // Attempt to add the record
419)         //
420)         try {
421)             megaDBF.open_database();
422)             megaDBF.No_1.put( no1.getText().trim());
423)             megaDBF.No_2.put( no2.getText().trim());
424)             megaDBF.No_3.put( no3.getText().trim());
425)             megaDBF.No_4.put( no4.getText().trim());
426)             megaDBF.No_5.put( no5.getText().trim());
427)             megaDBF.Mega_No.put( megaNo.getText().trim());
428)             megaDBF.Draw_Dt.put( localDateStr);
429)
430)             megaDBF.getDBF().write();
431)             megaDBF.close_database();
432)
433)             retVal = true;
434)             currMode = ENTRY_MODE;
435)             draw_dt_str = " ";
436)             lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
437)             no1.setValue( null);
438)             no2.setValue( null);
439)             no3.setValue( null);
440)             no4.setValue( null);
441)             no5.setValue( null);
442)             megaNo.setValue( null);
443)             deletedFlg.setText( " ");
444)             drawingDate.setValue( null);
445)         } catch( xBaseJException s) {
446)             display_error_msg( "Error adding record " + s.toString());
447)         } catch( IOException i) {
448)             display_error_msg( "Error adding record " + i.toString());
449)         }
450)
451)
452)         return retVal;
453)     } // end addRecord method
454)
455)
456)     /*;;;;;

```



```
457)      * method to update a record in the database
458)      *;;;;
459)      */
460)  private boolean updateRecord() {
461)      boolean retVal = false;
462)      int      l_x;
463)      String   originalDrawDtStr;
464)      String   localDateStr=null;
465)      String   upd_str;
466)
467)
468)      if (!is_record_valid())
469)          return false;
470)
471)      // Obtain values from the panel
472)      //
473)      originalDrawDtStr = draw_dt_str; // save original value
474)      draw_dt_str = drawingDate.getText();
475)      pad_draw_dt_str();
476)      try {
477)          localDateStr = out_format.format( out_format.parse(draw_dt_str));
478)      } catch( ParseException p) {
479)          display_error_msg( "Error parsing date " + p.toString());
480)      }
481)
482)      if (localDateStr == null)
483)          return false;
484)
485)      if (localDateStr.compareTo( originalDrawDtStr) != 0) {
486)          display_error_msg( "Not allowed to change drawing date");
487)          return false;
488)      }
489)
490)      // Attempt to add the record
491)      //
492)      try {
493)          megaDBF.open_database();
494)          megaDBF.find_EQ_record( localDateStr);
495)          megaDBF.No_1.put( no1.getText().trim());
496)          megaDBF.No_2.put( no2.getText().trim());
497)          megaDBF.No_3.put( no3.getText().trim());
498)          megaDBF.No_4.put( no4.getText().trim());
499)          megaDBF.No_5.put( no5.getText().trim());
500)          megaDBF.Mega_No.put( megaNo.getText().trim());
501)          megaDBF.Draw_Dt.put( localDateStr);
502)
503)          megaDBF.getDBF().update();
504)          megaDBF.close_database();
505)
506)          retVal = true;
507)          currMode = ENTRY_MODE;
508)          draw_dt_str = "";
509)          lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
510)          no1.setValue( null);
511)          no2.setValue( null);
512)          no3.setValue( null);
513)          no4.setValue( null);
514)          no5.setValue( null);
515)          megaNo.setValue( null);
516)          deletedFlg.setText( " ");
517)          drawingDate.setValue( null);
518)      } catch( xBaseJException s) {
519)          display_error_msg( "Error updating record " + s.toString());
```

```

520)         } catch( IOException i) {
521)             display_error_msg( "Error adding record " + i.toString());
522)         }
523)
524)
525)         return retVal;
526)     } // end updateRecord method
527)
528)     /*;;;;;
529)     * method to delete a record which has been found
530)     *;;;;;
531)     */
532) private boolean deleteRecord() {
533)     boolean retVal = false;
534)     int     l_x;
535)     String  del_str;
536)     String  localDateStr=null;
537)
538)     draw_dt_str = drawingDate.getText();
539)     //pad_draw_dt_str();
540)
541)     try {
542)         localDateStr = out_format.format( out_format.parse(draw_dt_str));
543)     }
544)     catch( ParseException p) {
545)         display_error_msg( "Error parsing date " + p.toString());
546)     }
547)
548)     if (localDateStr == null)
549)         return false;
550)
551)     try {
552)         megaDBF.open_database();
553)         megaDBF.find_EQ_record( localDateStr);
554)         megaDBF.getDBF().delete();
555)         megaDBF.close_database();
556)
557)         retVal = true;
558)         currMode = ENTRY_MODE;
559)         draw_dt_str = " ";
560)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
561)         no1.setValue( null);
562)         no2.setValue( null);
563)         no3.setValue( null);
564)         no4.setValue( null);
565)         no5.setValue( null);
566)         megaNo.setValue( null);
567)         drawingDate.setValue( null);
568)         deletedFlg.setText( " ");
569)     } catch( xBaseJException s) {
570)         display_error_msg( "Error deleting record " + s.toString());
571)     } catch( IOException i) {
572)         display_error_msg( "Error adding record " + i.toString());
573)     }
574)
575)     return retVal;
576) } // end deleteRecord method
577)
578)     /*;;;;;
579)     * method to find a record based upon drawing date
580)     *;;;;;
581)     */
582) private boolean findRecord() {

```

```

583)         int      l_x=0;
584)         boolean retVal=false;
585)         String find_str;
586)         String localDateStr=null;
587)
588)         draw_dt_str = drawingDate.getText();
589)         pad_draw_dt_str();
590)
591)         try {
592)             localDateStr = out_format.format( out_format.parse(draw_dt_str));
593)         }
594)         catch( ParseException p) {
595)             display_error_msg( "Error parsing date " + p.toString());
596)             localDateStr = null;
597)         }
598)
599)         if (localDateStr == null)
600)             return false;
601)
602)         try {
603)             megaDBF.open_database();
604)             l_x = megaDBF.find_GE_record( localDateStr);
605)
606)             dDrawingDate      = out_format.parse( megaDBF.Draw_Dt.get());
607)             lNo1               = Integer.parseInt( megaDBF.No_1.get().trim());
608)             lNo2               = Integer.parseInt( megaDBF.No_2.get().trim());
609)             lNo3               = Integer.parseInt( megaDBF.No_3.get().trim());
610)             lNo4               = Integer.parseInt( megaDBF.No_4.get().trim());
611)             lNo5               = Integer.parseInt( megaDBF.No_5.get().trim());
612)             lMegaNo            = Integer.parseInt( megaDBF.Mega_No.get().trim
613)         ());
614)
615)         if (megaDBF.getDBF().deleted())
616)             deletedFlg.setText(" *");
617)         else
618)             deletedFlg.setText(" ");
619)
620)         megaDBF.close_database();
621)
622)         // Update the screen
623)         drawingDate.setValue( new Integer(out_format.format
624)         (dDrawingDate)));
625)         draw_dt_str = out_format.format( dDrawingDate);
626)         no1.setValue( new Integer(lNo1));
627)         no2.setValue( new Integer(lNo2));
628)         no3.setValue( new Integer(lNo3));
629)         no4.setValue( new Integer(lNo4));
630)         no5.setValue( new Integer(lNo5));
631)         megaNo.setValue( new Integer(lMegaNo));
632)         retVal = true;
633)     } catch( ParseException p) {
634)         display_error_msg( "Error parsing date " + draw_dt_str +
635)         "Error was " + p.toString());
636)         draw_dt_str = " ";
637)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
638)         no1.setValue( null);
639)         no2.setValue( null);
640)         no3.setValue( null);
641)         no4.setValue( null);
642)         no5.setValue( null);
643)         megaNo.setValue( null);
644)         deletedFlg.setText(" ");

```

```

644)         drawingDate.setValue( null);
645)     } catch( NumberFormatException n) {
646)         display_error_msg( "Error parsing date " + draw_dt_str +
647)             "Error was " + n.toString());
648)         draw_dt_str = " ";
649)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
650)         no1.setValue( null);
651)         no2.setValue( null);
652)         no3.setValue( null);
653)         no4.setValue( null);
654)         no5.setValue( null);
655)         megaNo.setValue( null);
656)         deletedFlg.setText( " ");
657)         drawingDate.setValue( null);
658)     }
659)     return retVal;
660) } // end findRecord method
661)
662)
663) //;;;;;
664) // Internal method to pad the display of numbers to 2 digits
665) //;;;;;
666) private void pad_draw_dt_str() {
667)     draw_dt_str = draw_dt_str.trim();
668)     switch( draw_dt_str.length()) {
669)         case 4: // only year provided
670)             draw_dt_str += "0101";
671)             break;
672)         case 6: // year and month
673)             draw_dt_str += "01";
674)             break;
675)         case 8: // full date, no padding
676)             break;
677)         default: // no idea, pick a default
678)             draw_dt_str = "19900101";
679)             break;
680)     } // end switch of length
681)
682) } // end pad_draw_dt_str method
683)
684)
685) /*;;;;;
686) * Begin logic to rebuild the Stats tables
687) *;;;;;
688) */
689) public void createStatsTable()
690) {
691)     int     l_x;
692)     int     l_record_count;
693)
694)     l_draw_no = 0;
695)     //
696)     // It will seem odd to you, but you have to do 2
697)     // allocations when working with a non native array.
698)     //
699)     //
700)     drawNoStat = new StatElms[ ELM_COUNT];
701)     for (l_x=1; l_x < ELM_COUNT; l_x++) {
702)         drawNoStat[ l_x] = new StatElms();
703)         drawNoStat[ l_x].elmNo = l_x;
704)     } // end for loop to init stats
705)
706)

```

```

707)         megaNoStat = new StatElms[ ELM_COUNT];
708)         System.out.println( "Initializing megaNoStat");
709)         for (l_x=1; l_x < ELM_COUNT; l_x++ ) {
710)             megaNoStat[ l_x]           = new StatElms();
711)             megaNoStat[ l_x].elmNo     = l_x;
712)         } // end for loop to init stats
713)
714)
715)         megaDBF.open_database();
716)
717)         // Create a statement
718)         try {
719)             l_record_count = megaDBF.getDBF().getRecordCount();
720)
721)             for (l_x=1; l_x <= l_record_count; l_x++) {
722)                 megaDBF.getDBF().gotoRecord( l_x);
723)                 l_draw_no++;
724)                 updateDrawStats( Integer.parseInt( megaDBF.No_1.get().trim
725)             ));
726)                 updateDrawStats( Integer.parseInt( megaDBF.No_2.get().trim
727)             ));
728)                 updateDrawStats( Integer.parseInt( megaDBF.No_3.get().trim
729)             ));
730)                 updateDrawStats( Integer.parseInt( megaDBF.No_4.get().trim
731)             ));
732)                 updateDrawStats( Integer.parseInt( megaDBF.No_5.get().trim
733)             ));
734)                 updateMegaStats( Integer.parseInt( megaDBF.Mega_No.get
735)             ));
736)
737)                 if (l_draw_no % 100 == 0)
738)                     System.out.println( "Processed " + l_x + " Records");
739)                 } // end for l_y loop
740)
741)                 megaDBF.close_database();
742)             } catch( xBaseJException s) {
743)                 display_error_msg( "Error reading DBF " + s.toString());
744)             } catch( IOException i) {
745)                 display_error_msg( "Error reading DBF " + i.toString());
746)             } catch( NumberFormatException n) {
747)                 display_error_msg( "Error parsing integer " + n.toString());
748)             }
749)
750)             System.out.println( "Processed " + l_draw_no + " Records");
751)
752)             writeDrawStats( );
753)         } // end createStatsTable method
754)
755)         //;;;;;;;;;;
756)         // Method to actually write all of our Stats records to the database.
757)         //;;;;;;;;;;
758)         private void writeDrawStats( ) {
759)
760)             int l_x, l_missed, l_y;
761)             StatDBF drawStatDBF=null;
762)             StatDBF megaStatDBF=null;
763)
764)             drawStatDBF = new StatDBF();
765)             megaStatDBF = new StatDBF();
766)
767)             drawStatDBF.create_database( "drawst");
768)             megaStatDBF.create_database( "megast");

```

```

764)
765)
766)      System.out.println( "Writing mega stats records");
767)      System.out.println( "Value of l_draw_no " + l_draw_no);
768)
769)      for (l_x=1; l_x < ELM_COUNT; l_x++ ) {
770)          drawNoStat[ l_x].pctHits = (double)( megaNoStat[ l_x].hitCount) /
771)                                     (double)( l_draw_no);
772)          l_missed      = l_draw_no - megaNoStat[ l_x].hitCount;
773)          megaNoStat[ l_x].aveBtwn = (double)( l_missed) /
774)                                     (double)( drawNoStat[ l_x].hitCount);
775)          megaNoStat[ l_x].sinceLast = l_draw_no - megaNoStat
[l_x].lastDrawNo;
776)
777)          try {
778)              megaStatDBF.Elm_No.put( megaNoStat[ l_x].elmNo);
779)              megaStatDBF.Hit_Count.put( megaNoStat[ l_x].hitCount);
780)              megaStatDBF.Last_Draw_No.put( megaNoStat[ l_x].lastDrawNo);
781)              megaStatDBF.Since_Last.put( megaNoStat[ l_x].sinceLast);
782)              megaStatDBF.Curr_Seq.put( megaNoStat[ l_x].currSeq);
783)              megaStatDBF.Longest_Seq.put( megaNoStat[ l_x].longestSeq);
784)              megaStatDBF.Pct_Hits.put( megaNoStat[ l_x].pctHits);
785)              megaStatDBF.Max_Btwn.put( megaNoStat[ l_x].maxBtwn);
786)              megaStatDBF.Ave_Btwn.put( megaNoStat[ l_x].aveBtwn);
787)
788)              megaStatDBF.getDBF().write();
789)
790)          } catch( xBaseJException s) {
791)              display_error_msg( "Error adding Mega stat record " +
792)                               s.toString());
793)          } catch( IOException i) {
794)              display_error_msg( "Error reading DBF " + i.toString());
795)          }
796)      } // end for l_x loop
797)
798)      System.out.println( "Writing Drawing stats records");
799)
800)      for (l_x=1; l_x < ELM_COUNT; l_x++ ) {
801)          drawNoStat[ l_x].pctHits = (double)( drawNoStat[ l_x].hitCount) /
802)                                     (double)( l_draw_no);
803)          l_missed      = l_draw_no - drawNoStat[ l_x].hitCount;
804)          drawNoStat[ l_x].aveBtwn = (double)( l_missed) /
805)                                     (double)( drawNoStat[ l_x].hitCount);
806)          drawNoStat[ l_x].sinceLast = l_draw_no - drawNoStat
[l_x].lastDrawNo;
807)
808)          try {
809)              drawStatDBF.Elm_No.put( drawNoStat[ l_x].elmNo);
810)              drawStatDBF.Hit_Count.put( drawNoStat[ l_x].hitCount);
811)              drawStatDBF.Last_Draw_No.put( drawNoStat[ l_x].lastDrawNo);
812)              drawStatDBF.Since_Last.put( drawNoStat[ l_x].sinceLast);
813)              drawStatDBF.Curr_Seq.put( drawNoStat[ l_x].currSeq);
814)              drawStatDBF.Longest_Seq.put( drawNoStat[ l_x].longestSeq);
815)              drawStatDBF.Pct_Hits.put( drawNoStat[ l_x].pctHits);
816)              drawStatDBF.Max_Btwn.put( drawNoStat[ l_x].maxBtwn);
817)              drawStatDBF.Ave_Btwn.put( drawNoStat[ l_x].aveBtwn);
818)
819)              drawStatDBF.getDBF().write();
820)          } catch( xBaseJException s) {
821)              display_error_msg( "Error adding Drawing Stat record " +
822)                               s.toString());
823)          } catch( IOException i) {
824)              display_error_msg( "Error reading DBF " + i.toString());

```

```
825)        }
826)    } // end for l_x loop
827)
828)        System.out.println( "All Stats records successfully written");
829)        JRootPane j = (JRootPane) SwingUtilities.getAncestorOfClass
( JRootPane.class, nol);
830)        if ( j != null) {
831)            JOptionPane.showMessageDialog(j,"All Stats records successfully
written",
832)                                           "Stats Generated",
JOptionPane.PLAIN_MESSAGE);
833)        }
834)        else
835)            System.out.println( "j was null");
836)
837)    } // end writeDrawStats method
838)
839)
840)    private void updateDrawStats( int num_sub) {
841)        int        l_x;
842)
843)        l_x = l_draw_no - drawNoStat[ num_sub].lastDrawNo;
844)
845)        if (l_x == 1)
846)        {
847)            drawNoStat[ num_sub].currSeq++;
848)            if (drawNoStat[ num_sub].currSeq > drawNoStat
[ num_sub].longestSeq)
849)            {
850)                drawNoStat[ num_sub].longestSeq = drawNoStat
[ num_sub].currSeq;
851)            }
852)        }
853)        else {
854)            drawNoStat[ num_sub].currSeq = 0;
855)            if (l_x > drawNoStat[num_sub].maxBtwn)
856)            {
857)                drawNoStat[ num_sub].maxBtwn = l_x;
858)            }
859)        } // end test for sequence
860)
861)        drawNoStat[ num_sub].hitCount++;
862)        drawNoStat[ num_sub].lastDrawNo = l_draw_no;
863)        drawNoStat[ num_sub].sinceLast = l_x;
864)    } // end updatetrawStats method
865)
866)    private void updateMegaStats( int num_sub) {
867)        int        l_x;
868)
869)        l_x = l_draw_no - megaNoStat[ num_sub].lastDrawNo;
870)
871)        if (l_x == 1)
872)        {
873)            megaNoStat[ num_sub].currSeq++;
874)            if (megaNoStat[ num_sub].currSeq > megaNoStat
[ num_sub].longestSeq)
875)            {
876)                megaNoStat[ num_sub].longestSeq = megaNoStat
[ num_sub].currSeq;
877)            }
878)        }
879)        else {
880)            megaNoStat[ num_sub].currSeq = 0;
```

```

881)         if (l_x > megaNoStat[num_sub].maxBtwn)
882)         {
883)             megaNoStat[ num_sub].maxBtwn = l_x;
884)         }
885)     } // end test for sequence
886)
887)     megaNoStat[ num_sub].hitCount++;
888)     megaNoStat[ num_sub].lastDrawNo = l_draw_no;
889)     megaNoStat[ num_sub].sinceLast = l_x;
890) } // end updateMegaStats method
891)
892)
893) private boolean is_record_valid() {
894)     if (no1.getValue() == null) {
895)         errorMsg = "No 1 Invalid";
896)         no1.requestFocus();
897)         return false;
898)     }
899)
900)     if (no2.getValue() == null) {
901)         errorMsg = "No 2 Invalid";
902)         no2.requestFocus();
903)         return false;
904)     }
905)
906)     if (no3.getValue() == null) {
907)         errorMsg = "No 3 Invalid";
908)         no3.requestFocus();
909)         return false;
910)     }
911)
912)     if (no4.getValue() == null) {
913)         errorMsg = "No 4 Invalid";
914)         no4.requestFocus();
915)         return false;
916)     }
917)
918)     if (no5.getValue() == null) {
919)         errorMsg = "No 5 Invalid";
920)         no5.requestFocus();
921)         return false;
922)     }
923)
924)     if ( megaNo.getValue() == null) {
925)         errorMsg = "Mega No Invalid";
926)         megaNo.requestFocus();
927)         return false;
928)     }
929)
930)     return true;
931) } // end is_record_valid method
932)
933) void display_error_msg( String msg) {
934)     JRootPane m = (JRootPane)
935)     SwingUtilities.getAncestorOfClass( JRootPane.class, drawingDate);
936)     if ( m != null)
937)     {
938)         JOptionPane.showMessageDialog(m, msg, "Entry",
939)                                     JOptionPane.ERROR_MESSAGE);
940)     }
941)     else
942)         System.out.println( "m was null msg was |" + msg + "|");
943) } // end display_error_msg

```



```

944)
945)
946)    /*;;;;;
947)    * method to find first record
948)    *;;;;;
949)    */
950) private boolean firstRecord() {
951)     int    l_x=0;
952)     boolean retVal=false;
953)     String find_str;
954)     String localDateStr=null;
955)
956)     try {
957)         megaDBF.open_database();
958)         megaDBF.getDBF().startTop();
959)         megaDBF.getDBF().findNext();
960)
961)         dDrawingDate    = out_format.parse( megaDBF.Draw_Dt.get());
962)         lNo1             = Integer.parseInt( megaDBF.No_1.get().trim());
963)         lNo2             = Integer.parseInt( megaDBF.No_2.get().trim());
964)         lNo3             = Integer.parseInt( megaDBF.No_3.get().trim());
965)         lNo4             = Integer.parseInt( megaDBF.No_4.get().trim());
966)         lNo5             = Integer.parseInt( megaDBF.No_5.get().trim());
967)         lMegaNo          = Integer.parseInt( megaDBF.Mega_No.get().trim
    ());
968)
969)         if (megaDBF.getDBF().deleted())
970)             deletedFlg.setText(" *");
971)         else
972)             deletedFlg.setText(" ");
973)
974)         megaDBF.close_database();
975)
976)         // Update the screen
977)         //
978)         drawingDate.setValue( new Integer(out_format.format
    (dDrawingDate)));
979)         draw_dt_str = out_format.format( dDrawingDate);
980)         no1.setValue( new Integer(lNo1));
981)         no2.setValue( new Integer(lNo2));
982)         no3.setValue( new Integer(lNo3));
983)         no4.setValue( new Integer(lNo4));
984)         no5.setValue( new Integer(lNo5));
985)         megaNo.setValue( new Integer(lMegaNo));
986)         retVal = true;
987)     } catch( xBaseJException s) {
988)         display_error_msg( "Unable to find " + localDateStr +
989)             "Error was " + s.toString());
990)         draw_dt_str = " ";
991)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
992)         no1.setValue( null);
993)         no2.setValue( null);
994)         no3.setValue( null);
995)         no4.setValue( null);
996)         no5.setValue( null);
997)         megaNo.setValue( null);
998)         deletedFlg.setText(" ");
999)         drawingDate.setValue( null);
1000)    } catch( IOException i) {
1001)        display_error_msg( "Unable to find " + localDateStr +
1002)            "Error was " + i.toString());
1003)        draw_dt_str = " ";
1004)        lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;

```

```

1005)         no1.setValue( null);
1006)         no2.setValue( null);
1007)         no3.setValue( null);
1008)         no4.setValue( null);
1009)         no5.setValue( null);
1010)         megaNo.setValue( null);
1011)         deletedFlg.setText( " ");
1012)         drawingDate.setValue( null);
1013)     } catch( ParseException p) {
1014)         display_error_msg( "Error parsing date " + draw_dt_str +
1015)             "Error was " + p.toString());
1016)         draw_dt_str = " ";
1017)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1018)         no1.setValue( null);
1019)         no2.setValue( null);
1020)         no3.setValue( null);
1021)         no4.setValue( null);
1022)         no5.setValue( null);
1023)         megaNo.setValue( null);
1024)         deletedFlg.setText( " ");
1025)         drawingDate.setValue( null);
1026)     } catch( NumberFormatException n) {
1027)         display_error_msg( "Error parsing date " + draw_dt_str +
1028)             "Error was " + n.toString());
1029)         draw_dt_str = " ";
1030)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1031)         no1.setValue( null);
1032)         no2.setValue( null);
1033)         no3.setValue( null);
1034)         no4.setValue( null);
1035)         no5.setValue( null);
1036)         megaNo.setValue( null);
1037)         deletedFlg.setText( " ");
1038)         drawingDate.setValue( null);
1039)     }
1040)
1041)     return retVal;
1042) } // end firstRecord method
1043)
1044) /*****
1045)  * method to find previous record based upon drawing date
1046)  *****/
1047) */
1048) private boolean prevRecord() {
1049)     int     l_x=0;
1050)     boolean retVal=false;
1051)     String  find_str;
1052)     String  localDateStr=null;
1053)
1054)     if (currMode != FIND_MODE) {
1055)         display_error_msg( "Must have previously found to move back one
record");
1056)         return false;
1057)     }
1058)
1059)     draw_dt_str = drawingDate.getText();
1060)     pad_draw_dt_str();
1061)
1062)     try {
1063)         localDateStr = out_format.format( out_format.parse
(draw_dt_str));
1064)     }
1065)     catch( ParseException p) {

```

```

1066)         display_error_msg( "Error parsing date " + p.toString());
1067)         localDateStr = null;
1068)     }
1069)
1070)     if (localDateStr == null)
1071)         return false;
1072)
1073)     try {
1074)         megaDBF.open_database();
1075)         megaDBF.find_EQ_record( draw_dt_str);
1076)         megaDBF.getDBF().findPrev();
1077)
1078)         dDrawingDate      = out_format.parse( megaDBF.Draw_Dt.get());
1079)         lNo1              = Integer.parseInt( megaDBF.No_1.get().trim());
1080)         lNo2              = Integer.parseInt( megaDBF.No_2.get().trim());
1081)         lNo3              = Integer.parseInt( megaDBF.No_3.get().trim());
1082)         lNo4              = Integer.parseInt( megaDBF.No_4.get().trim());
1083)         lNo5              = Integer.parseInt( megaDBF.No_5.get().trim());
1084)         lMegaNo           = Integer.parseInt( megaDBF.Mega_No.get().trim
1085)     ());
1086)
1087)         if (megaDBF.getDBF().deleted())
1088)             deletedFlg.setText( "*");
1089)         else
1090)             deletedFlg.setText( " ");
1091)
1092)         megaDBF.close_database();
1093)
1094)         // Update the screen
1095)         //
1096)         drawingDate.setValue( new Integer(out_format.format
1097)     (dDrawingDate)));
1098)         draw_dt_str = out_format.format( dDrawingDate);
1099)         no1.setValue( new Integer(lNo1));
1100)         no2.setValue( new Integer(lNo2));
1101)         no3.setValue( new Integer(lNo3));
1102)         no4.setValue( new Integer(lNo4));
1103)         no5.setValue( new Integer(lNo5));
1104)         megaNo.setValue( new Integer(lMegaNo));
1105)         retVal = true;
1106)     } catch( xBaseJException s) {
1107)         display_error_msg( "Unable to find " + localDateStr +
1108)             "Error was " + s.toString());
1109)         draw_dt_str = " ";
1110)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1111)         no1.setValue( null);
1112)         no2.setValue( null);
1113)         no3.setValue( null);
1114)         no4.setValue( null);
1115)         no5.setValue( null);
1116)         megaNo.setValue( null);
1117)         drawingDate.setValue( null);
1118)         deletedFlg.setText( " ");
1119)     } catch( IOException i) {
1120)         display_error_msg( "Unable to find " + localDateStr +
1121)             "Error was " + i.toString());
1122)         draw_dt_str = " ";
1123)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1124)         no1.setValue( null);
1125)         no2.setValue( null);
1126)         no3.setValue( null);
1127)         no4.setValue( null);
1128)         no5.setValue( null);

```

```

1127)         megaNo.setValue( null);
1128)         drawingDate.setValue( null);
1129)         deletedFlg.setText( " ");
1130)     } catch( ParseException p) {
1131)         display_error_msg( "Error parsing date " + draw_dt_str +
1132)             "Error was " + p.toString());
1133)         draw_dt_str = " ";
1134)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1135)         no1.setValue( null);
1136)         no2.setValue( null);
1137)         no3.setValue( null);
1138)         no4.setValue( null);
1139)         no5.setValue( null);
1140)         megaNo.setValue( null);
1141)         drawingDate.setValue( null);
1142)         deletedFlg.setText( " ");
1143)     } catch( NumberFormatException n) {
1144)         display_error_msg( "Error parsing date " + draw_dt_str +
1145)             "Error was " + n.toString());
1146)         draw_dt_str = " ";
1147)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1148)         no1.setValue( null);
1149)         no2.setValue( null);
1150)         no3.setValue( null);
1151)         no4.setValue( null);
1152)         no5.setValue( null);
1153)         megaNo.setValue( null);
1154)         drawingDate.setValue( null);
1155)         deletedFlg.setText( " ");
1156)     }
1157)
1158)     return retVal;
1159) } // end prevRecord method
1160)
1161) /*****
1162)  * method to find a next record based upon drawing date
1163)  *****/
1164) /*
1165) private boolean nextRecord() {
1166)     int     l_x=0;
1167)     boolean retVal=false;
1168)     String find_str;
1169)     String localDateStr=null;
1170)
1171)     draw_dt_str = drawingDate.getText();
1172)     pad_draw_dt_str();
1173)
1174)     try {
1175)         localDateStr = out_format.format( out_format.parse
(draw_dt_str));
1176)     }
1177)     catch( ParseException p) {
1178)         display_error_msg( "Error parsing date " + p.toString());
1179)         localDateStr = null;
1180)     }
1181)
1182)     if (localDateStr == null)
1183)         return false;
1184)
1185)     try {
1186)         megaDBF.open_database();
1187)         megaDBF.find_GE_record( localDateStr.trim());
1188)         megaDBF.getDBF().findNext();

```

```

1189)
1190)         dDrawingDate      = out_format.parse( megaDBF.Draw_Dt.get() );
1191)         lNo1               = Integer.parseInt( megaDBF.No_1.get().trim() );
1192)         lNo2               = Integer.parseInt( megaDBF.No_2.get().trim() );
1193)         lNo3               = Integer.parseInt( megaDBF.No_3.get().trim() );
1194)         lNo4               = Integer.parseInt( megaDBF.No_4.get().trim() );
1195)         lNo5               = Integer.parseInt( megaDBF.No_5.get().trim() );
1196)         lMegaNo            = Integer.parseInt( megaDBF.Mega_No.get().trim
    ( ) );
1197)         if (megaDBF.getDBF().deleted())
1198)             deletedFlg.setText( "*" );
1199)         else
1200)             deletedFlg.setText( " " );
1201)
1202)         megaDBF.close_database();
1203)
1204)         // Update the screen
1205)         //
1206)         drawingDate.setValue( new Integer(out_format.format
    (dDrawingDate)) );
1207)         draw_dt_str = out_format.format( dDrawingDate );
1208)         no1.setValue( new Integer(lNo1) );
1209)         no2.setValue( new Integer(lNo2) );
1210)         no3.setValue( new Integer(lNo3) );
1211)         no4.setValue( new Integer(lNo4) );
1212)         no5.setValue( new Integer(lNo5) );
1213)         megaNo.setValue( new Integer(lMegaNo) );
1214)         retVal = true;
1215)     } catch( xBaseJException s ) {
1216)         display_error_msg( "Unable to find " + localDateStr +
1217)             "Error was " + s.toString() );
1218)         draw_dt_str = " ";
1219)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1220)         no1.setValue( null );
1221)         no2.setValue( null );
1222)         no3.setValue( null );
1223)         no4.setValue( null );
1224)         no5.setValue( null );
1225)         megaNo.setValue( null );
1226)         drawingDate.setValue( null );
1227)     } catch( IOException i ) {
1228)         display_error_msg( "Unable to find " + localDateStr +
1229)             "Error was " + i.toString() );
1230)         draw_dt_str = " ";
1231)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1232)         no1.setValue( null );
1233)         no2.setValue( null );
1234)         no3.setValue( null );
1235)         no4.setValue( null );
1236)         no5.setValue( null );
1237)         megaNo.setValue( null );
1238)         drawingDate.setValue( null );
1239)     } catch( ParseException p ) {
1240)         display_error_msg( "Error parsing date " + draw_dt_str +
1241)             "Error was " + p.toString() );
1242)         draw_dt_str = " ";
1243)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1244)         no1.setValue( null );
1245)         no2.setValue( null );
1246)         no3.setValue( null );
1247)         no4.setValue( null );
1248)         no5.setValue( null );
1249)         megaNo.setValue( null );

```

```

1250)         drawingDate.setValue( null);
1251)     }
1252)
1253)     return retVal;
1254) } // end nextRecord method
1255)
1256) /*;;;;;
1257)  * method to find last record based upon drawing date
1258)  *;;;;;
1259)  */
1260) private boolean lastRecord() {
1261)     int l_x=0;
1262)     boolean retVal=false;
1263)     String find_str;
1264)     String localDateStr=null;
1265)
1266)     try {
1267)         megaDBF.open_database();
1268)         megaDBF.getDBF().startBottom();
1269)         megaDBF.getDBF().findPrev();
1270)
1271)         dDrawingDate = out_format.parse( megaDBF.Draw_Dt.get());
1272)         lNo1 = Integer.parseInt( megaDBF.No_1.get().trim());
1273)         lNo2 = Integer.parseInt( megaDBF.No_2.get().trim());
1274)         lNo3 = Integer.parseInt( megaDBF.No_3.get().trim());
1275)         lNo4 = Integer.parseInt( megaDBF.No_4.get().trim());
1276)         lNo5 = Integer.parseInt( megaDBF.No_5.get().trim());
1277)         lMegaNo = Integer.parseInt( megaDBF.Mega_No.get().trim
1278)     ());
1279)
1280)         if (megaDBF.getDBF().deleted())
1281)             deletedFlg.setText("");
1282)         else
1283)             deletedFlg.setText(" ");
1284)
1285)         megaDBF.close_database();
1286)
1287)         // Update the screen
1288)         drawingDate.setValue( new Integer(out_format.format
1289)     (dDrawingDate)));
1290)         draw_dt_str = out_format.format( dDrawingDate);
1291)         no1.setValue( new Integer(lNo1));
1292)         no2.setValue( new Integer(lNo2));
1293)         no3.setValue( new Integer(lNo3));
1294)         no4.setValue( new Integer(lNo4));
1295)         no5.setValue( new Integer(lNo5));
1296)         megaNo.setValue( new Integer(lMegaNo));
1297)         retVal = true;
1298)     } catch( xBaseJException s) {
1299)         display_error_msg( "Unable to find " + localDateStr +
1300)             "Error was " + s.toString());
1301)         draw_dt_str = " ";
1302)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1303)         no1.setValue( null);
1304)         no2.setValue( null);
1305)         no3.setValue( null);
1306)         no4.setValue( null);
1307)         no5.setValue( null);
1308)         megaNo.setValue( null);
1309)         drawingDate.setValue( null);
1310)     } catch( IOException i) {
1311)         display_error_msg( "Unable to find " + localDateStr +

```

```

1311)         "Error was " + i.toString());
1312)         draw_dt_str = " ";
1313)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1314)         no1.setValue( null);
1315)         no2.setValue( null);
1316)         no3.setValue( null);
1317)         no4.setValue( null);
1318)         no5.setValue( null);
1319)         megaNo.setValue( null);
1320)         drawingDate.setValue( null);
1321)     } catch( ParseException p) {
1322)         display_error_msg( "Error parsing date " + draw_dt_str +
1323)         "Error was " + p.toString());
1324)         draw_dt_str = " ";
1325)         lNo1 = lNo2 = lNo3 = lNo4 = lNo5 = lMegaNo = 0;
1326)         no1.setValue( null);
1327)         no2.setValue( null);
1328)         no3.setValue( null);
1329)         no4.setValue( null);
1330)         no5.setValue( null);
1331)         megaNo.setValue( null);
1332)         drawingDate.setValue( null);
1333)     }
1334)
1335)     return retVal;
1336) } // end findRecord method
1337)
1338)
1339) } // end MegaXbaseEntryPanel class definition

```

I'm sorry. The listing for MegaXbaseEntryPanel is just long. It would have been even longer if I had continued adding some other neat features.

We need to start our discussion at listing lines 79 through 102. I have no way of knowing what level your Java skills are, so let me point out the internal class contraption we use to verify drawing number input is within a valid range. Everything between the first { and the ending } becomes the body of the abstract class. We must provide a Boolean verify() method and a Boolean shouldYieldFocus() method. The actual editing/validating happens in the verify() method. Here we convert the JComponent object to a type we know it to be. (Yes, I could have made this even more robust by checking with instanceof.) We pass back either true or false to indicate the quality of the data.

You will find documentation online which claims the shouldYieldFocus() method is optional. I wouldn't buy that statement even if they paid me to take it, mainly because years ago I tried to leave it out. We aren't far enough into the source code yet, but it shouldn't take much for you to believe that I like to toss up error message dialogs when something failed validation. After the first error tries to throw a dialog up you find yourself in a deadly embrace with multiple things absolutely demanding they be allowed to have focus at the same time. It's not a pretty picture. Depending on how you launched your application you could even be forced to reboot. You don't directly call shouldYieldFocus(), the underlying Swing API does.

Please look at listing lines 254 through 259. This is where we create the “top of data” or “first record” button (<<<<). The LINE_START value is basically a synonym for WEST. The really important line is where we set gbc.weightx to 1.0. This creates the gap between the (<<<<) button and the (<) button. I did not spend much time closing the gap between the (<) and (>) buttons. This gap exists for two reasons:

1. buttons are sized based upon the text they contain
2. buttons are positioned based upon the order in which they are declared and gridwidth

You can change the gridwidth for the (>) button to 1 from RELATIVE and you can add both a leading and trailing space in the text of both buttons. This will cause the buttons to mush together under Find and Delete, but they won't really be visually centered. GridBagConstraints does have a public integer array named columnWidths and it provides a method getLayoutDimensions() column widths and row heights for the current layout. If you put more columnWidth values into the array than currently exist, columns will be added to the layout. I leave the centering of those buttons as an exercise for the reader.

Listing line 294 is something I haven't talked much about; I've just been using it. When you get done creating a panel or dialog you can tell Swing what component you want to have focus by calling this method. Be sure to remember that calling it in the constructor means the focus is only set the first time the object is displayed.

Our actionPerformed() method at listing line 311 would really benefit if Java could add switch constructs which used non-constant values in the case statements. The logic would at least look a lot cleaner.

Normally I wouldn't put a bunch of assignment statements in an event switch as I did at listing lines 348 through 358, but I got cut-and-paste happy, and I wanted to give you an easy assignment. Eventually you will be creating a private method to replace all of these code blocks.

Notice that I call the display_error_msg() method from many places in the code. If you take a look at listing line 933 you will see that tiny snippet of code I talked about in the last module was placed into its own method where it could serve many purposes.

Listing line 402 contains a call to pad_draw_dt_str(). I had to create that method to allow for partial date entry. I probably shouldn't have called it in the addRecord() method because it will allow for a user to enter a partial date when adding a record. The method will use the date string “19990101” when it is looking to fill in missing date components.

The section of code at listing lines 493 through 504 contains a serious multi-user problem. I dutifully found the record to update, moved in the values, and wrote it back to the file. What I didn't do was check to see whether any other user or process had changed the values of that record between the time the entry screen loaded it and the user chose to save it.

We don't have anything interesting to talk about until we get down to listing lines 762 and 763. This is the first, and only, time I use the StatDBF class we created. Notice how I passed a short name without extension to the create_database() method. I do not know if the xBaseJ library has a file name length limitation, but it was 8.3 at one point during the days of DOS. Since I will be adding 'k0' to the end of the name to create the NDX files, I opted to pass only 6 characters in. I also allow Java garbage collection to close off these files sometime after the objects go out of scope.

Listing line 832 contains another version of that message dialog. This time we pass PLAIN_MESSAGE as the option so Swing displays a run-of-the-mill status message dialog instead of an error message.

Listing lines 969 through 972 contain an if statement which is replicated in a few places. Here we call the deleted() method provided by the DBF class to determine if we display a space or an '*' in the Deleted field on the screen.

Notice that each 'find' method in this class stores the values found in class global fields. You need to pay attention to that design feature if you intend to complete one of the assignments coming up at the end of the chapter.

Listing lines 978 and 979 may require a tiny bit of explanation. The object which allows for date entry on our panel is a simple JformattedTextField. I chose not to play games trying to make this display with a pretty format. It's not that I don't like pretty date formats, it's just that I didn't want to install and include the apache libraries to get their JdateField object and I didn't want to complicate the entry sequence by creating a JDateChooser object. The end result is that we have to convert the date from the database from string to a date object Java likes, then format it to a new string to convert to an Integer object. The conversion from string into a date data type helps validate the column on the database. We could choose to trust it, but why bother when it doesn't cost that many cycles to be sure?

There you have it: The biggest source file I've shown you so far in this book. Perhaps you noticed we talked more about the Java aspects of this module than the xBaseJ aspects. You already have most of the fundamentals down. The purpose of this chapter is to give you ideas on how to bolt them together correctly, at least from a design standpoint.

2.4 The Import Dialog

The import dialog is going to seem really lame after the main portion of this application. I chose to make the import module a dialog before I started writing the rest of the application. There was actually a method to the madness. If the Import function was simply another panel, it would be possible for a user to choose a file name, then leave the screen by selecting another menu option. He or she would not have actually performed the import, but might believe it was complete. Making this a dialog stopped that from happening.

MegaXImport.java

```

1)  package com.logikal.megazillxBASEJ;
2)
3)  import java.awt.*;
4)  import java.awt.event.*;
5)  import javax.swing.*;
6)  import javax.swing.filechooser.*;
7)  import java.text.*;
8)  import java.util.*;
9)  import java.io.*;
10)
11) import org.xBaseJ.*;
12) import org.xBaseJ.fields.*;
13) import org.xBaseJ.Util.*;
14) import org.xBaseJ.indexes.NDX;
15)
16) import com.logikal.megazillxBASEJ.MegaDBF;
17)
18) public class MegaXImport extends JDialog implements ActionListener{
19)     // Variables
20)     public JButton      importData=null;
21)     public JButton      exitButton=null;
22)     public JButton      chooseFile=null;
23)     private JTextField  csvNameField=null;
24)     private JPanel      line1Panel=null;
25)     private JPanel      line2Panel=null;
26)     private MegaDBF      MegaDB = null;
27)     private String      CsvName=null;
28)     private JTextArea    importRptArea;
29)     private JScrollPane sp;
30)     private JFileChooser fc;
31)
32)
33)     // constructor
34)     public MegaXImport(Frame owner) {
35)
36)         super( owner, "Import CSV", true); // constructor for parent class
37)         setSize( 800, 500);
38)         setLayout( new FlowLayout());
39)
40)         line1Panel = new JPanel( new FlowLayout( FlowLayout.LEFT));
41)         JLabel csvNameLabel = new JLabel("CSV File:");
42)         line1Panel.add(csvNameLabel);
43)
44)         csvNameField = new JTextField(40);
45)         csvNameField.setEditable( false);
46)         line1Panel.add(csvNameField);
47)

```

```

48)         line2Panel = new JPanel( new FlowLayout( FlowLayout.LEFT));
49)
50)         importRptArea = new JTextArea();
51)         // Gives you a fixed width font.
52)         importRptArea.setFont(new Font("Courier", Font.PLAIN, 12));
53)         importRptArea.setEditable( false);
54)         importRptArea.setTabSize(4);
55)         importRptArea.setColumns( 80);
56)         importRptArea.setRows(1000);
57)         importRptArea.setDoubleBuffered( true);
58)         sp = new JScrollPane( importRptArea);
59)         sp.setPreferredSize( new Dimension( 500, 300));
60)
61)         line2Panel.add(sp);
62)
63)
64)         chooseFile = new JButton();
65)         chooseFile.setText("Choose File");
66)         chooseFile.addActionListener( this);
67)         line1Panel.add( chooseFile);
68)
69)         importData = new JButton();
70)         importData.setText("OK");
71)         importData.addActionListener( this);
72)         line2Panel.add(importData);
73)
74)         exitButton = new JButton();
75)         exitButton.setText( "Exit");
76)         exitButton.addActionListener( this);
77)         line2Panel.add(exitButton);
78)
79)         add( line1Panel);
80)         add( line2Panel);
81)         chooseFile.requestFocus();
82)         getRootPane().setDefaultButton( importData);
83)
84)         setLocationRelativeTo(owner);
85)
86)         MegaDB = new MegaDBF();
87)
88)     } // end constructor for RdbLogin
89)
90)
91)     //;;;;;
92)     // Obtain file name and kick off import process
93)     //;;;;;
94)     public void actionPerformed(ActionEvent event) {
95)
96)         String actionStr = event.getActionCommand();
97)         if (actionStr.indexOf( "Exit") > -1) {
98)             this.setVisible( false);
99)             return;
100)        }
101)
102)        if (actionStr.indexOf( "Choose") > -1) {
103)            fc = new JFileChooser();
104)            int ret_val = fc.showOpenDialog(this);
105)            if (ret_val == JFileChooser.APPROVE_OPTION) {
106)                File f = fc.getSelectedFile();
107)                csvNameField.setText( f.getAbsolutePath());
108)            }
109)        } // end test for choose actionStr
110)

```

```

111)
112)         if (actionStr.indexOf( "OK") > -1) {
113)             if (!importCSV(csvNameField.getText() )) {
114)                 importRptArea.append( "Import not successfull\n");
115)                 importRptArea.append( "please try again\n");
116)             }
117)         }
118)     } // end actionPerformed
119)
120)     //;;;;;
121)     // Actual import logic happens here
122)     //;;;;;
123)     private boolean importCSV( String the_file) {
124)         String line_in_str = null;
125)         long l_record_count = 0;
126)         boolean eof_flg = false, ret_val = false;
127)         FileReader in_file = null;
128)         BufferedReader input_file = null;
129)
130)         importRptArea.append( "\nAttempting to import " + the_file + "\n");
131)         updateText();
132)
133)         try {
134)             in_file = new FileReader( the_file);
135)         } catch (FileNotFoundException f) {
136)             importRptArea.append("File Not Found " + the_file);
137)             eof_flg = true;
138)         } // end catch for file not found
139)
140)         if (eof_flg == true)
141)             return ret_val;
142)
143)         MegaDB.create_database();
144)
145)         input_file = new BufferedReader( in_file,4096);
146)         importRptArea.append("\nPopulating database\n");
147)         updateText();
148)
149)         while (eof_flg == false) {
150)             try {
151)                 line_in_str = input_file.readLine();
152)             }
153)             catch (EOFException e) {
154)                 importRptArea.append( "End of file exception\n");
155)                 eof_flg = true;
156)             }
157)             catch (IOException e) {
158)                 importRptArea.append( "An IO Exception occurred\n");
159)                 importRptArea.append( e.toString());
160)                 //e.printStackTrace();
161)                 eof_flg = true;
162)             }
163)             if (eof_flg == true) continue;
164)             if (line_in_str == null) {
165)                 importRptArea.append( "End of input file reached\n");
166)                 eof_flg = true;
167)                 continue;
168)             }
169)
170)             l_record_count++;
171)             String input_flds[] = line_in_str.split( ",");
172)
173)             try {

```

```

174)                MegaDB.No_1.put( input_flds[1]);
175)                MegaDB.No_2.put( input_flds[2]);
176)                MegaDB.No_3.put( input_flds[3]);
177)                MegaDB.No_4.put( input_flds[4]);
178)                MegaDB.No_5.put( input_flds[5]);
179)                MegaDB.Mega_No.put( input_flds[6]);
180)
181)                String date_parts[] = input_flds[0].split("-");
182)                String dt_str = date_parts[0] + date_parts[1] + date_parts
[2];
183)                MegaDB.Draw_Dt.put( dt_str);
184)
185)                MegaDB.getDBF().write();
186)
187)                } catch ( xBaseJException j){
188)                    j.printStackTrace();
189)                } catch( IOException i) {
190)                    importRptArea.append( i.getMessage());
191)                }
192)
193)                if ( (l_record_count % 100) == 0) {
194)                    importRptArea.append( "Processed " + l_record_count +
195)                        " records\n");
196)                    updateText();
197)                } // end of test for status message
198)
199)            } // end while loop to load records
200)
201)            importRptArea.append( "Finished adding " + l_record_count +
202)                " records\n");
203)
204)            return true;
205)
206)        } // end importCSV method
207)
208)        public void updateText() {
209)            importRptArea.invalidate();
210)            importRptArea.validate();
211)            importRptArea.paintImmediately( importRptArea.getVisibleRect());
212)        }
213)    } // end MegaXImport class

```

Listing lines 24 and 25 are worthy of note. Some of you may have the impression that a panel is a screen. As our constructor shows, this is simply not the case. We allocate one panel to contain the the CSV file name prompt, text field, and the Choose button. A second panel is created to contain the text area and scroll pane along with the Ok and Exit buttons. When you are using the FlowLayout instead of the GridLayout it is quite common to have multiple panels in a containing object. It provides a method of “controlling the flow” by grouping objects together.

Notice listing lines 81 through 84. After we have added the panels to the dialog, we have the button to choose a file request focus but we set the default button to be the import button. If you have tried running the application you will already have learned “the last one in won.” The text entry field is the field which actually has focus, but the Ok button is highlighted to indicate that hitting return will activate it.

Listing line 104 is what ensures the user must complete the file chooser dialog one way or another before this dialog continues. I hope you don't find it strange that a dialog can throw up a dialog which can throw up another dialog that can throw up another dialog. I haven't conducted a test to see just how deep you can go, but I assume it has something to do with a 2GB memory limit imposed on many JVM installs.

The dialog returns an integer value to inform us of its completion status. If the user chose and approved a file name, we call `getSelectedFile()` to obtain the File object. We then have to call `getAbsolutePath()` to obtain the full path name. Under most circumstances, you cannot open the file unless you provide the full path name. I didn't provide a starting location for the file chooser so it will start in the user's home directory instead of the current working directory. If you want it to start there simply change listing line 103 to read as follows:

```
fc = new JFileChooser(System.getProperty("user.dir"));
```

There isn't much left to discuss in the `importCSV()` method. You know that I call the `updateText()` method to force screen updates so my status messages get displayed while they are relevant instead of after the task completes. I have already provided you several examples which read a line of input from a text file and use the `String split()` method to break it into separate data items. We have used the `Field put()` method and the `DBF write()` method many times over in previous source listings.

MegaXbase.java

```

1)  import java.awt.*;
2)  import java.awt.event.*;
3)  import javax.swing.*;
4)  import javax.swing.plaf.*;
5)
6)  import com.sun.java.swing.plaf.windows.WindowsLookAndFeel;
7)  import com.sun.java.swing.plaf.gtk.GTKLookAndFeel;
8)  import com.sun.java.swing.plaf.motif.MotifLookAndFeel;
9)
10) import com.logikal.megazillxBaseJ.MegaXImport;
11) import com.logikal.megazillxBaseJ.MegaXbaseBrowsePanel;
12) import com.logikal.megazillxBaseJ.MegaXbaseEntryPanel;
13) import com.logikal.megazillxBaseJ.MegaXbaseDuePanel;
14)
15) public class MegaXbase implements ActionListener, ItemListener {
16)
17)     private JFrame          mainFrame=null;
18)     private JPanel          mainPanel=null;
19)     private JMenuBar        mb = null;
20)     private JPanel          blankPanel=null;
21)     private MegaXbaseBrowsePanel mxb=null;
22)     private MegaXbaseEntryPanel mxe=null;
23)     private MegaXbaseDuePanel md=null;
24)
25)     final static String MOSTPANEL = "Most Report";
26)     final static String DUEPANEL = "Due Report";
27)     final static String DUMPPANEL = "Dump Report";
28)     final static String ENTRYPANEL = "Entry";

```

```
29)     final static String BLANKPANEL = "Blank";
30)     final static String BROWSEPANEL = "Browse";
31)
32)     public MegaXbase() {
33)         //-----
34)         // All of this code just to set the look and feel
35)         //
36)         int nimbus_sub = -1;
37)         int motif_sub = -1;
38)         int chosen_sub;
39)
40)         try {
41)             // Set System Look and Feel
42)             UIManager.LookAndFeelInfo lf[] = UIManager.getInstalledLookAndFeels();
43)             for( int y=0; y < lf.length; y++) {
44)                 String s = lf[ y].getName();
45)                 System.out.println( s);
46)                 if ( s.indexOf( "Nimbus") > -1) {
47)                     nimbus_sub = y;
48)                     System.out.println( "\tNimbus found");
49)                 }
50)                 if ( s.indexOf( "Motif") > -1) {
51)                     motif_sub = y;
52)                     System.out.println( "\tMotif found");
53)                 }
54)             }
55)
56)             if ( nimbus_sub > -1)
57)                 chosen_sub = nimbus_sub;
58)             else if ( motif_sub > -1)
59)                 chosen_sub = motif_sub;
60)             else chosen_sub = 0;
61)
62)             UIManager.setLookAndFeel( lf[chosen_sub].getClassName());
63)             // UIManager.getSystemLookAndFeelClassName());
64)         }
65)         catch (UnsupportedLookAndFeelException e) {
66)             System.out.println( "Unsupported look and feel");
67)         }
68)         catch (ClassNotFoundException e) {
69)             System.out.println( "classnotfound");
70)         }
71)         catch (InstantiationException e) {
72)             System.out.println( "Instantiation exeception");
73)         }
74)         catch (IllegalAccessException e) {
75)             System.out.println( "Illegal Access");
76)         }
77)
78)         mainFrame = new JFrame("Mega xBaseJ Window");
79)         mainFrame.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE);
80)
81)         mainFrame.setJMenuBar( createMenu());
82)
83)         mainPanel = new JPanel(new CardLayout());
84)         mainPanel.setOpaque(true);
85)
86)         blankPanel = new JPanel();
87)         mxb = new MegaXbaseBrowsePanel();
88)         mxe = new MegaXbaseEntryPanel();
89)         md = new MegaXbaseDuePanel();
90)
91)         mainPanel.add( blankPanel, BLANKPANEL);
```

```
92)      mainPanel.add( mxb, BROWSEPANEL);
93)      mainPanel.add( mxe, ENTRYPANEL);
94)      mainPanel.add( md, DUEPANEL);
95)
96)      mainFrame.setContentPane( mainPanel);
97)
98)      mainFrame.setSize(800, 500);
99)      mainFrame.setVisible( true);
100)
101)  } // end default constructor
102)
103)  private JMenuBar createMenu() {
104)      JMenu fileMenu, reportMenu;
105)      JMenuItem menuItem;
106)
107)      mb = new JMenuBar();
108)
109)      //;;;;
110)      // Build the File menu
111)      //;;;;
112)      fileMenu = new JMenu("File");
113)      fileMenu.setMnemonic( KeyEvent.VK_F);
114)      fileMenu.getAccessibleContext().setAccessibleDescription(
115)          "File operation menu");
116)
117)      // Import menu option
118)      menuItem = new JMenuItem("Import", KeyEvent.VK_I);
119)      menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_I,
120)          ActionEvent.ALT_MASK));
121)      menuItem.getAccessibleContext().setAccessibleDescription(
122)          "Imports data from CSV creating new DBF");
123)      menuItem.setActionCommand("Import");
124)      menuItem.addActionListener( this);
125)      fileMenu.add( menuItem );
126)
127)      // Maintenance Menu Option
128)      menuItem = new JMenuItem("Maintenance", KeyEvent.VK_M);
129)      menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_M,
130)          ActionEvent.ALT_MASK));
131)      menuItem.getAccessibleContext().setAccessibleDescription(
132)          "Manual Entry/Editing/Deletion");
133)      menuItem.setActionCommand("Entry");
134)      menuItem.addActionListener( this);
135)      fileMenu.add( menuItem);
136)
137)      // Browse option
138)      menuItem = new JMenuItem("Browse", KeyEvent.VK_B);
139)      menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_B,
140)          ActionEvent.ALT_MASK));
141)      menuItem.getAccessibleContext().setAccessibleDescription(
142)          "View Data");
143)      menuItem.setActionCommand("Browse");
144)      menuItem.addActionListener( this);
145)      fileMenu.add( menuItem);
146)
147)
148)      fileMenu.addSeparator();
149)
150)      menuItem = new JMenuItem("Exit", KeyEvent.VK_X);
151)      menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_X,
152)          ActionEvent.ALT_MASK));
153)      menuItem.getAccessibleContext().setAccessibleDescription(
154)          "Exit program");
```



```

155)     menuItem.setActionCommand("Exit");
156)     menuItem.addActionListener( new ActionListener() {
157)         public void actionPerformed(ActionEvent evt){
158)             System.exit(0);}});
159)     fileMenu.add( menuItem);
160)
161)     //;;;;;
162)     // Build the File menu
163)     //;;;;;
164)     reportMenu = new JMenu("Report");
165)     reportMenu.setMnemonic(KeyEvent.VK_R);
166)     reportMenu.setAccessibleContext().setAccessibleDescription(
167)         "Report creation menu");
168)
169)     // Import menu option
170)     menuItem = new JMenuItem("Complete Data Dump", KeyEvent.VK_C);
171)     menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_C,
172)         ActionEvent.ALT_MASK));
173)     menuItem.getAccessibleContext().setAccessibleDescription(
174)         "Reports all data on file");
175)     menuItem.setActionCommand("Dump");
176)     menuItem.addActionListener( this);
177)     reportMenu.add( menuItem );
178)
179)     menuItem = new JMenuItem("Most Often Hit", KeyEvent.VK_M);
180)     menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_M,
181)         ActionEvent.ALT_MASK));
182)     menuItem.getAccessibleContext().setAccessibleDescription(
183)         "Most frequently drawn numbers");
184)     menuItem.setActionCommand("Most");
185)     menuItem.addActionListener( this);
186)     reportMenu.add( menuItem );
187)
188)     menuItem = new JMenuItem("Due Numbers", KeyEvent.VK_D);
189)     menuItem.setAccelerator( KeyStroke.getKeyStroke( KeyEvent.VK_D,
190)         ActionEvent.ALT_MASK));
191)     menuItem.getAccessibleContext().setAccessibleDescription(
192)         "Due numbers");
193)     menuItem.setActionCommand("Due");
194)     menuItem.addActionListener( this);
195)     reportMenu.add( menuItem );
196)
197)     //;;;;;
198)     // Add the new menus to the bar
199)     //;;;;;
200)     mb.add( fileMenu);
201)     mb.add( reportMenu);
202)
203)     return mb;
204) } // end createMenu method
205)
206)
207)
208) //;;;;;
209) // When a user choses a menu item we process it here
210) // If Java would allow a switch to use strings, or would give
211) // the JButton class a number field which got passed to a field in
212) // the ActionEvent class, this code would be a lot cleaner.
213) //;;;;;
214) public void actionPerformed( ActionEvent e) {
215)     String actionStr = e.getActionCommand();
216)     System.out.println( "\nSelected action " + actionStr);
217)

```

```

218)     Frame f = (Frame) SwingUtilities.getAncestorOfClass( Frame.class, mb);
219)
220)     if (actionStr.indexOf( "Import") > -1) {
221)         MegaXImport importDialog = new MegaXImport( f);
222)         importDialog.setVisible(true);
223)         importDialog.dispose();
224)     }
225)     else if (actionStr.indexOf("Browse") > -1) {
226)         CardLayout cl = (CardLayout)(mainPanel.getLayout());
227)         cl.show(mainPanel, BROWSEPANEL );
228)     }
229)     else if (actionStr.indexOf("Entry") > -1) {
230)         CardLayout cl = (CardLayout)(mainPanel.getLayout());
231)         cl.show(mainPanel, ENTRYPANEL );
232)     }
233)     else if (actionStr.indexOf("Due") > -1) {
234)         CardLayout cl = (CardLayout)(mainPanel.getLayout());
235)         cl.show(mainPanel, DUEPANEL );
236)     }
237)     } else {
238)         System.out.println( "unhandled action");
239)     }
240) } // end actionPerformed method
241)
242)
243) public void itemStateChanged( ItemEvent e) {
244)     System.out.println( "state change");
245)     System.out.println( e.toString());
246) }
247)
248)
249)
250) } // end MegaXbase class definition

```

The code for the main menu is just a tiny bit convoluted. Listing lines 41 through 76 exist because I believe Metal is probably the ugliest Look and Feel anyone could have invented. I needed to change that look and feel without having this thing crash the first time you tried to compile it. The safe thing for me to do was scan through the list of Look and Feels which Java “thought” were installed. Until the advent of Java 1.6 and the creation of a file called `swing.properties`, Java had no real way of finding out about any look and feel Sun didn't provide.

Traditionally, applications will include an extra JAR file containing a Look and Feel and make certain that JAR file gets added to the `CLASSPATH` environment variable. This allows the code to change a Look and Feel to be much shorter. Simply add an import line at the top and then replace the try block with a cleaner piece of code.

```

import com.nilo.plaf.nimrod.*;

try {
    UIManager.setLookAndFeel( new com.nilo.plaf.nimrod.NimRODLookAndFeel());
}
catch (UnsupportedLookAndFeelException e) {
    System.out.println( "Unsupported look and feel");
}

```

It shouldn't surprise you to learn that this is exactly what I did to generate the screen shots shown on page 96. Some Look and Feels have very subtle changes, and some work only on specific platforms. If you are going to pick one or create one, please make certain it works on *all* desktop operating systems before you release it. I cannot tell you how many Office XP-type look and feel packages are out there, and every one of them works only on the Windows platform. Gee, thanks, guys.

If you have not done much Java programming, please let me take the time to point out listing line 78. We have not declared an instance of this, as our classes have all been support or panel classes up to this point. An application requires you to construct a frame. The frame is a container with an optional title that holds all other components which make up the application.

Listing line 79 is one line you won't notice you forgot until you try to close the application. If you started it from the command line, your prompt won't return. The window will be gone, but the app will still be running. At that point you either get very good with system utilities to find it, or reboot and hope your operating system doesn't try to 'help you out' by restarting all applications which were running at time of shutdown.

After calling a method to create our menubar at listing line 81, I create an instance of each panel and add each to the mainPanel along with a String name so I can find it again. Once I have all of the panels added, I set the content pane of the frame to be the mainPanel. Trust me, it sounds far more complex than it is.

Why do you think I added a blank panel?

Come on, think about it. Why would I add a blank panel to the application and give it a name so I could find it again? Perhaps to clear the screen? That would be the correct answer. I run into a lot of GUI-based menu applications written with a lot of different tools and a lot of them have the same bug. Once you change that initial panel under the menu, they don't provide any method of clearing it other than exiting the program and re-entering.

The createMenu() method shows the funky process Java makes a developer endure just to build a standard menu. To a human, the whole thing, File+Report+drop downs, is the menu. In Swing, File is its own menu as well as Reports. Each menu is hung on the menuBar and the name of the menu is displayed at that location on the menuBar.

Please pay attention to the nested if-else structure starting at listing line 220. Your assignments will require you to create new conditions in this structure. Once we identify which menu option was chosen based upon the text of its action we need to either launch the associated dialog or shuffle the correct panel to the top. We need the name each panel was added with in order to find it with the show() method.

testMegaXbase.java

```

1)  import java.awt.*;
2)  import java.awt.event.*;
3)  import javax.swing.*;
4)  import javax.swing.plaf.*;
5)
6)  import com.logikal.megazillxBaseJ.*;
7)
8)  public class testMegaXbase {
9)
10)      public static void main(String args[]){
11)          MegaXbase mx = new MegaXbase();
12)
13)      }
14)
15) } // end testMegaXbase class

```

We don't have anything to discuss with this module. I simply needed to include it for completeness. My build command file is equally non-complex.

b.sh

```

1)  #! /bin/bash
2)  #
3)  #
4)  # rm *.dbf
5)  # rm *.ndx
6)  #
7)  javac -source 1.4 -target 1.4 -d . MegaDBF.java
8)  javac -source 1.4 -target 1.4 -d . StatDBF.java
9)  javac -source 1.4 -target 1.4 -d . StatElms.java
10) #
11) jar -cfv megaX.jar com/logikal/megazillxBaseJ/MegaDBF.class
12) jar -ufv megaX.jar com/logikal/megazillxBaseJ/StatDBF.class
13) jar -ufv megaX.jar com/logikal/megazillxBaseJ/StatElms.class
14) #
15) javac -source 1.4 -target 1.4 -d . MegaXImport.java
16) javac -source 1.4 -target 1.4 -d . MegaXbaseBrowsePanel.java
17) javac -source 1.4 -target 1.4 -d . MegaXbaseEntryPanel.java
18) javac -source 1.4 -target 1.4 -d . MegaXDueElms.java
19) #
20) jar -ufv megaX.jar com/logikal/megazillxBaseJ/MegaXbaseBrowsePanel.class
21) jar -ufv megaX.jar com/logikal/megazillxBaseJ/MegaXImport.class
22) jar -ufv megaX.jar com/logikal/megazillxBaseJ/MegaXbaseEntryPanel.class
23) jar -ufv megaX.jar com/logikal/megazillxBaseJ/MegaXDueElms.class
24) #
25) javac -source 1.4 -target 1.4 -d . MegaXbaseDuePanel.java
26) #
27) jar -ufv megaX.jar com/logikal/megazillxBaseJ/MegaXbaseDuePanel.class
28) #
29) javac -source 1.4 -target 1.4 MegaXbase.java
30) #

```

```

31) javac -source 1.4 -target 1.4 testMegaXbase.java
32)
33) javac -source 1.4 -target 1.4 testNDXBug.java

```

The `-source` qualifier tells the Java compiler to restrict the input source to 1.4 syntax. We control the bytecode output by the `-target` switch, which tells the Java compiler to generate bytecode sequences which are compatible with version 1.4 JVMs.

I put almost all of this code into a JAR file. Whenever you wish to create a package you need to indicate to the Java compiler where to put the class files. This is done by the “package” statement in the source file and the “`-d .`” switch I put on the command line. This switch tells the compiler to use the current directory as the root of the package.

```

roland@logikaldesktop:~/mega_xbasej$ ./b.sh
added manifest
adding: com/logikal/megazillxBaseJ/MegaDBF.class(in = 4429) (out= 2404) (deflated 45%)
adding: com/logikal/megazillxBaseJ/StatDBF.class(in = 4152) (out= 2227) (deflated 46%)
adding: com/logikal/megazillxBaseJ/StatElms.class(in = 394) (out= 282) (deflated 28%)
adding: com/logikal/megazillxBaseJ/MegaXbaseBrowsePanel.class(in = 5063) (out= 2713) (deflated 46%)
adding: com/logikal/megazillxBaseJ/MegaXImport.class(in = 5728) (out= 3134) (deflated 45%)
adding: com/logikal/megazillxBaseJ/MegaXbaseEntryPanel.class(in = 20237) (out= 8860) (deflated 56%)
adding: com/logikal/megazillxBaseJ/MegaXDueElms.class(in = 823) (out= 546) (deflated 33%)
adding: com/logikal/megazillxBaseJ/MegaXbaseDuePanel.class(in = 6054) (out= 3189) (deflated 47%)
roland@logikaldesktop:~/mega_xbasej$ dir com
logikal
roland@logikaldesktop:~/mega_xbasej$ dir com/logikal
megazillxBaseJ
roland@logikaldesktop:~/mega_xbasej$ dir com/logikal/megazillxBaseJ
MegaDBF.class      MegaXbaseDuePanel.class      MegaXbaseEntryPanel.class  MegaXImport.class
StatElms.class
MegaXbaseBrowsePanel.class  MegaXbaseEntryPanel$1.class  MegaXDueElms.class        StatDBF.class

```

One thing which may be confusing to many of you is that Linux uses “/” as a directory separator, Java uses “.”, and DOS (Windows) uses “\”. If you type “`dir com.logikal`” and nothing appears it’s simply a user error.

2.5 Programming Assignment 1

Modify the `updateRecord()` method of the Entry module to make certain no values have changed on the file record between the time of reading and the time of attempted update. If they have, issue an appropriate error message and stop the update. You can test this by changing a record, updating it, then find the same record again and perform an Import between the time you find it and the time you write your new changes to the database.

2.6 Programming Assignment 2

Modify the Entry module by creating a `clearScreen()` method which consists of the code found at listing lines 348 through 358. Replace all such code occurrences by a call to your new method. Test the application to ensure it is working. How many lines of code did you save?

2.7 Programming Assignment 3

Modify the `open_database()` method of `StatDBF.java` to check whether the database is already open. If it is open, close the currently open database before proceeding with the open.

2.8 Programming Assignment 4

There are currently two reports listed on the menu which do not currently have any implementation provided. The “Most Often Hit” report can easily be implemented by providing a new class, `MegaXMostElms`, which compares only the hit counts. You can then clone the `Due` report, changing report headings, while loops, and array data type names. *Be certain your new report runs from the menu!*

You have to create the dump report from scratch. There is nothing complex about it. The report will be very much like the browse window except that records will be written to a text area instead of a spreadsheet.

2.9 Summary

This chapter has been meant to provide a real-world example to help users new to `xBaseJ`, and possibly even new to Java, get up to speed. Most of the examples you find on-line are very “one-shot” in nature. An attempt was made to provide you with a nearly complete business-type application. You should be able to pick and choose pieces of this design for your own use. *Don' t just steal the code, consider the design!*

Professional developers can design their way around most of the critical weak points found in any toolset they are forced to use. People who cut and paste code without considering the design constraints in effect at the time tend to blindly stumble into every critical flaw known to man. Don' t stumble around; read the explanation which has been provided first.

You now know how to add records to a data file, create an `NDX` file, and read records both in index order and directly. More importantly, you have been informed of some of the bugs and been shown code which works around them. There is no reason you should not be able to develop usable applications after having read this entire book. You might “think” you can develop applications after merely skimming this chapter and stealing the code, but you would be mistaken. The beginning of this book provides you with questions you need to ask before designing any application. The most important question of all is “Should you be using an `xBASE` file to store this data?”

Too many application developers simply reach for what they used last time without considering the lifespan of what they will produce. Sometimes this lapse leads to overkill, such as a complete MySQL database to store 50 records, and other times it is under-kill, such as trying to manage what is now a 100,000-item product catalog with xBASE files. You must look at the current need and the potential future need when designing an application.

Whenever you have a system which will be used by only one person, will store fewer than a couple thousand records, and needs to be stand-alone, xBASE files are a good option. Just be certain you aren't limiting someone's future when you make the decision. I see a lot of comments and messages on-line to the various commercial xBASE engine providers from people and companies who have been developing a system with the tool in question since the 1980s. Business kept growing and they kept customizing, and now they issue pleas to the vendors to do something about the 2GB limit, as they have had to hack their systems to support multiple primary DBF files.

Laugh all you want -- I've actually read more than one message like that recently. I'm not trying to discourage you from using this tool, I'm trying to educate you as to its proper use. In each case, those poor bastards started out with a few hundred items, but business grew into a few hundred thousand items and their custom system now cannot handle it. They are now looking at a complete system redevelopment, and as the emails suggest, are willing to try anything to avoid it.

Certain applications will always lend themselves to a small self-contained indexed file system. Our lottery tracker is a good example. Personal income tax filing systems are another. Small retail systems can do very well, but you have to develop complete IO classes to completely shield the application from data storage. I do mean completely shield. Your main application can never use an object from the library or trap an exception from it. Instead of having your Field objects public as I did, you have to make them private and write a unique get() and set() method for each column. Most of you won't do this. Instead you will develop much as I have shown you. It seems pretty clean at first glance, until you try to replace the underlying database with PostgreSQL or MySQL. Then it becomes a re-write. If you are designing for the here and now knowing there could be a migration, you have to design in the migration path now, not later.

As a community project, xBaseJ is almost perfect. I'm not saying that it is bug-free, I'm saying it is the perfect class of project for a community (OpenSource) effort. There are literally hundreds of places on-line containing documentation about the various xBASE formats. There are many Python, Perl, and C/C++ OpenSource xBASE libraries one can obtain the source code for as well. Despite their current Java skill level, developers participating in the project can obtain all the information they need without having to have a large support forum. You can even test your changes for interoperability with the other OpenSource xBASE projects so you don't have to wonder if you did them correctly. If it works cleanly with two other OpenSource products, you did it correctly enough for the OpenSource community. Remember, there is no ANSI standard for xBASE. What really matters is that all of the stuff in the OpenSource world works together. Don't forget that OpenOffice and KSpread both provide the ability to open a DBF file directly. Be sure to test your results with these applications as well. Some day IBM may even add direct support for DBF files to Lotus Symphony.

2.10 Review Questions

1. What does CUA stand for?
2. What is the default look and feel for Java applications?
3. What DBF method tells you if a record has been deleted?
4. Under what conditions is it okay to load DBF contents into a spreadsheet?
5. After opening an existing DBF and attaching one or more existing NDX files, what should you do?
6. Are the various Field variable names required to match their corresponding xBASE column names?
7. What interface must a class implement in order for it to be used with an Arrays.sort() call?
8. Does the statement:

```
MegaXDueElms d_elms[] = new MegaXDueElms[ELM_COUNT];
```

completely allocate an array of MegaXDueElms or do you still have to create each individual element? If create, what is the syntax to create an element?
9. What String method must be used when attempting to convert the result of a get() for a NumField to an Integer()? Why?
10. What javac command line option restricts the content of your source code?

Chapter 3

Ruminations

Some of you may not be familiar with this book series, so let me explain. I try to end each one with a chapter named Ruminations. These are essays about IT and life in general meant to make you think. Some may very well piss you off or morally offend you. So be it. Some may cause you to shout out support on a crowded train or plane when most everybody is asleep. So be it.

In short, this chapter is my reward for writing the book and may very well be your reward for reading it. On the whole, you should take away something from each essay which makes you a more worldly IT developer, if not a better person.

Enjoy!

3.1 Authoritative Resources

The events I'm about to relate actually occurred while I was writing this book. The primary developer/creator of the xBaseJ OpenSource project was shocked, to put it mildly, that I didn't use Eclipse for Java development. He mentioned some "Authoritative Resources" claiming some extremely high percentage of developers used Eclipse to write Java. I've been in IT long enough to know the truth about those "authoritative resources," so please allow me to shed some light on the subject.

Most of these "Authoritative Resources" come from a publisher or a supposedly independent analyst. The vast majority also point to a survey done by some standards body to help re-enforce their own manufactured data. Such surveys draw in the gullible (read that MBAs) who haven't got even the slightest technical clue. In short, people running around quoting these "authoritative resources" without any concept of how the data was obtained are naively helping to perpetrate a fraud.

It is not often in this book series that you will find it spouting any percentage pulled from any place other than my direct experiences in life. I hold this series to a much higher standard than the mass market publishers hawking everything from romance novels, to cook books, to supposedly scholarly tomes on IT. The content in this book series comes from a trench warrior, not a wishful thinker or a marketing department.

Given everything I have just told you, it should come as no surprise I wrote an essay on this when the developer quoted a mass market publisher and a standards body as the source of his information. The version presented here is a slightly more sanitized version of what I told him. You see, he forgot that he was talking to not only an author, but a book publisher. I know the publishing game. At least I know the kind of game played by the mass market houses.

When mass market Publisher X puts out a bunch of “free” information stating that N% of all developers are currently using technology Z you can pretty much bet that this information was written by the marketing department and has absolutely no basis in fact. You can pretty much prove this by looking at their title selection and counting the number of titles they have “recently” released covering competing technologies. You will find fifteen to twenty titles covering the N% technology and its complimentary topics (an example would be Eclipse + Java + OOP) and at best two “recent” titles on competing technologies (Kate, SciTE, jEdit, TEA, C/C++, Python, etc.)

The “recent” qualifier is most important. You will find dozens of titles on C/C++ from the mass market houses, but very few published in the past two years. The mass market houses (and Microsoft for that matter) make money only if technology is continually being replaced. You will find both mass market publishers and retail software vendors trying to create trends where none exist (or are needed) simply to generate revenue. The vast majority of people working for either vendor will completely ignore the fact that once a business application is written and in place, it tends to stay around for 30 years.

Oh, please, don' take my word for how long business applications stay in place once written. Simply search through the news archives for all of those Y2K stories talking about systems which have been making money for companies for roughly 30 years. When you are done with that, search for all of the current stories about “Heritage Data Silos” and “Legacy Systems.” Any application more than eight years old tends to fall into the “heritage” category. When Microsoft found itself being viewed as a “legacy” system with XP, they quickly tried to re-invent themselves as an Indian software company via the release of Windows Vista. It was such a rousing success, with high-ranking business officials from around the globe giving interviews stating their companies would never “upgrade” that Microsoft had to pull a lot of the development back onshore and put out a billable bug fix labeled “Windows 7.” They have removed the word “Vista” from most everything they control. Once again, go search for the articles, don' just quote me.

So, when mass market Publisher X tells you that 80% of all developers are using tool Z, you look and see that they said the same thing eight years ago about tool C and haven't published a book on tool C in the past 6 years. Oh my! They currently have eleven books in print on tool Z and a search of the Books In Print database shows they have four more being published this year. What happened to all of those applications written using tool C? Hmmm. Nary a word is spoken about that. Businesses must re-write 100% of their applications each and every time mass market Publisher X wants to sell more books. No wonder businesses are having trouble making money!

Of course Publisher X will try to quote some survey from a recognized standards or intellectual body like IEEE or ACM. IEEE is a standards body. I've never belonged to IEEE, but I have belonged to ACM and DECus. I currently belong to Encompass. Let me tell you how numbers get generated by such organizations. They send out a survey to their members. Those who respond right away are those working in academia and are desperate to have a quote printed somewhere because they are in a publish-or-perish world. Everyone else, those working for a living, file the email away saying they will get to it when time allows. Less than 1% of those people actually get around to responding, which is roughly the same number of members who bother to vote for the organization's leaders. So, when an organization like this puts out a number saying 80% of its surveyed members use tool Z, it's really 80% of 1%. The number is valid as far as it goes, but it doesn't go very far.

If you have never been a voting member of any organization, you probably don't have a good frame of reference for this. You see, most of these organizations provide some hook to keep members, then pretty much disregard the wishes of their membership. In one case, you have to be a member to obtain a free Hobbyist operating system license. The organization then turns around and completely disregards any and all who have an interest in that operating system because the token few on the board are busy trying to kiss up to a vendor championing a far less capable operating system. No matter how many people cast ballots, only the names in the leadership change, not the actual leadership. Most get tired of fighting the battle. They take the free license, do what they need to do with it, and let the scam continue.

The scam gets even worse when marketing firms try to relabel a portion of themselves as "industry analysts." I cannot tell you how many MBAs get handed four-color marketing glossies and honestly buy the story that this was independent industry research.

Simply put, almost nobody goes to prison for wire or mail fraud these days, certainly not the marketing departments for publicly traded companies. This means that there are no "Authoritative Resources" for current trends in software, simply marketing fraud, which most are conditioned to follow.

3.2 Timestamps on Reports

This won't be a long lecture, but it is one I need to cover in this edition. Hopefully, you have been a loyal reader from the logic book through the first edition of this application development book, Java, SOA, and now this very book. This discussion will make more sense to my loyal readers than my casual readers.

During the 70s and early 80s when jobs were run only daily or weekly we would put the date on the flag page and sometimes on the first heading line. Some report writing standards had each job creating its own "heading page" or "cover page" as the first page in the report file. This page would be the only place non-detail-level date information was placed.

Before the young whippersnappers reading this get all up in arms about our standards, let me paint a picture of the times. The vast majority of reports came from batch jobs run by computer operators. These jobs were either part of a regularly scheduled production cycle, or they were requested by various managers "on demand" and the lead operator would work them into the schedule. (Not all data was kept on-line back then, so jobs had to be scheduled based upon availability of tape and removable disk drives.) There was no such thing as a "personal printer" and very few "work group" printers scattered around the campus from which users could print their own reports. (In truth, probably the biggest driving force behind floppy-based personal computers getting into companies probably wasn't the software, but the fact you could get \$300 dot matrix printers to go with them. An office worker who generated a lot of paper had a better chance of surviving a layoff than an office worker who simply fetched coffee and attended meetings.)

The most important thing for you to remember is that printers were expensive, noisy, and used tractor-fed continuous form. Unlike today's laser printers which will take one or more bundles of copier paper and quietly hum out hundreds of duplex-printed pages sitting on a table in the center of a workgroup, early printers had to be kept in the machine room because of the sound dampening and security provided by that room. Most reports of the day were financial in nature. You certainly didn't want just anybody knowing you were 120 days past due for all of your vendors.

Many jobs would come off the system printer before an operator got around to separating them. Normally the batch jobs creating them would print the report with a statement like this:

```
$ PRINT/FLAG/BURST/NOTE="Deliver to Roland   created '$time()' "   some.rpt
```

The /BURST qualifier would cause two flag pages with a ‘burst bar’ printed between them. A burst bar was simply a bunch of characters printed near the common edge of the flag pages several lines deep. This made it easy for operations staff to find the reports when flipping through the pile of paper in front of them. The /NOTE contents would be printed in a specific location on each flag page. Full time operations staff knew to look for this. As turnover increased, and good letter-quality printers came into being, it was common for operations staff to have a print job which printed up nice cover sheets for all regularly scheduled jobs. The operator would then staple the cover sheet (which said who received this particular copy) to the front of each report copy. The ‘one off’ jobs still required more senior operations staff to ensure proper delivery.

We weren't obsessed with the time portion displayed on the flag page of a report, unless we had some service level agreement for some muckety-muck that absolutely had to have Report A in his hands by 9:05 A.M. Yes, I've worked for people like that before. I even stuck around one day after the report was delivered late just to see what he did with it. He handed it to his secretary who put it in a binder and filed it in a cabinet. We still got beat up for delivering late, but word got out that he didn't actually use that report.

During the mid 1980s we started to see cheap serial printers scattered about company locations and print queues created specifically for end users. Some of our batch jobs even started to print reports directly to those printers so operations didn't have to deliver them. Once users started being able to run reports from their menus we had to start having reporting standards which placed the time on the very first line of every page heading. Most shops reported the time on the left margin of the first line as I've shown you in this book. The time remained the same for an entire report. We didn't go out and snag a new timestamp each time we printed report headings. Most users were okay with the time being consistent on all page headings.

We started getting into trouble with long running jobs that used shared indexed files. We got into even more trouble when relational databases were introduced into the mix along with lower disk prices. Reports which took hours to run could have some transactions come into the files which would skew results. To help you understand the pain, try thinking about what happens when a billing job for a credit card company is running while transactions are posting.

Bill and Fred both walk into the same store with their credit card issued by the same company and have the same 27-day billing cycle ending on the very same day. They check out at the exact same time on two different registers. Bill's account begins with 04 and Fred's account begins with 98. Bill's statement has already been created at the time the authorization went through so his charge won't appear until next month. It just plain sucks to be Fred at this point.

This is the era of the 24-hour lifestyle and instant gratification. Large batch jobs like credit card invoicing still happen. Companies will create dozens, if not thousands, of print files each containing hundreds, if not thousands, of individual credit card statements. In some obscure location, and possibly encrypted, each statement will have the timestamp printed on it. It may even be printed using a very small font in what looks like a running page footer. Employees of the credit card company will know how to read it. They will divulge this knowledge during a customer service call when explaining to a husband and wife who made charges at the exact same time in the exact same store on their personal credit cards why only one had the charge show up on the statement.

While many reports on the Internet will be generated by some Java or Javascript calls to back-end report services, the services themselves will most likely be on a trusty OpenVMS cluster or an IBM mainframe. The Web pages which will receive the information will be in the business of aggregating this information from the different reporting services they are calling. It is really funny when they don't bother coordinating "as of" timestamps from the various back-end services prior to display.

Once again we can use credit cards as an example. Many people have a credit card which will give them some kind of award, such as airline miles or hotel bucks or some such thing. When a user opens up an online account to monitor this information he or she usually gets presented some kind of "summary" screen. The summary screen will coordinate responses received from all of the back-end services into one cohesive display showing current charges, payments, mileage, hotel bucks, etc. Most of the responses it chews on to create the display will be stored in some temporary storage on the Web browsing computer, usually in the form of a cookie, but could be anything. Things get hilarious when one or more of the back-end services are down but you haven't deleted all your cookies nor emptied your cache since your last visit to the site. You look at a summary report showing your last time's awarded mileage (or charges) with this time's charges (or mileage). These things get to be a problem when you are looking to use up your miles for your vacation.

Many PC-based operating systems will only give you a timestamp down to the number of seconds since midnight January 1, 1970. That level of resolution is not enough for today's world. It might get you by on a report heading, but not at the transaction level. Given that most of your time arguments are going to be about transaction level data, you need to provide some method of defending yourself when the argument becomes "why did this transaction show up on this invoice?" If your report page heading contains only the start time and there is no report footer containing the completion time, how do you defend yourself?

Let me leave you pondering reporting timestamps with this little ditty.

At any given stock exchange around the world, there will be orders coming in at a furious pace along with consolidated last sales. Orders which aren't "market" orders (meaning to buy or sell at the "market" price) get placed in the book to become part of the current market. The actual current market price is determined by the consolidated last sale, which is generated every second or so (longer time spans for lightly traded stocks) by sales which are "printed to tape." The consolidated last sale has the effect of causing orders in the specialist's book to suddenly be due a fill, and in some cases these automatically execute. Any executed order has its sale information sent to the primary exchange for that issue to become part of the next consolidated last sale. It is called "consolidated" because it is a calculated value based upon all sales "printed to tape" during the calculation time span. (Trade information used to print on paper tapes called "ticker tapes." Now that trade information is sent to the primary exchange in a particular data format, but the lingo "printed to tape" still exists.)

Things get dicey with market orders. Many exchanges can allow a market order to be briefly held for manual price improvement. There is no fear of the market moving because the market order is required to be tagged with the consolidated last sale value that was in effect at the time the market order came in or "current last sale price."

This last paragraph sounds kind of simple, doesn't it? It is...until you get around to defining "current." High-volume stocks with millions or billions of transactions per day can require "current" to be down to the nanosecond. Now we get into the issue of "in effect." Technically that consolidated last sale is "in effect" the nanosecond the primary market sends it out. The reality is that it takes more than a nanosecond for that last sale to transmit from one coast to the other and be recorded. Once it is recorded, all of the "book checking" then happens to see if any fixed price orders are now due a fill.

When I was just starting in Junior College, this time thing was something only academics thought about. We had disk seek times measured in seconds, and those seeks only happened once an operator got around to mounting the removable disk...if we were lucky enough to be using a disk. Now that I'm approaching the autumn of my career it is becoming a real issue. To those of you just starting out, what can I say other than, "it sucks to be you, Fred."

3.3 Doomed to Failure and Too Stupid to Know

I get a lot of phone calls and emails for contracts. There are few phone calls more entertaining than ‘large project to get off of OpenVMS.’ Inevitably, these projects are run by the Big N consulting firms with all of the actual work whored out to “preferred vendors.” Recently, I got another one of these calls. Here is pretty much how it went. (Caller is in blue.)

“How many years have you worked on OpenVMS?”

“20 plus.”

“How many years have you worked with a thing called PowerHouse?”

“It’s not a thing, it is a 4GL and it was a godsend when it came out. I’ve been working with it since the VAX 11/750 was the pride of most shops.”

“How long is that?”

“Well, DEC was still in business...1985? Shortly after that DEC came out with the Alpha.”

“Oh wow. We have a project to migrate from a VAX. I didn’t realize the hardware was that old.”

“Most likely you have a project to migrate from either an Alpha or an Itanium. Since OpenVMS migrated from VAX to Alpha to Itanium, most people still call the hardware which runs it a VAX. This is wrong, and until we end teacher tenure in this country, the problem isn’t likely to improve.”

“Oh. Well I’ve been more involved in the requirements gathering phase of the project for the past six months. We are only now starting to look at the existing hardware. They have something called a multi-node cluster running RDB, have you ever worked on that hardware?”

“Once again, it was OpenVMS. Multiple machines could be clustered together to really increase computing power and fault tolerance. I’ve worked on clusters that ranged in size from two machines in a room to lots of nodes scattered around the globe. From a development standpoint it really doesn’t make much difference how many or where they are.”

“Well, I’m sure that will come up. We are looking at replacing it with an OpenSource platform.”

“Well, if they were using RDB on a multi-node cluster, odds are they needed the fault tolerance the combination provided. You cannot create a fault-tolerant solution via Unix/Linux because the OS kernel doesn't have the concept of a record. Without the concept of a record, you cannot have a kernel level lock manager. Without a kernel level lock manager, you cannot develop a local transaction manager. Without a local transaction manager you cannot build a distributed transaction manager. Without being able to distribute transactions across a cluster with two-phase commit, you cannot create fault tolerance.”

“I'm sure we will discuss fault tolerance at some point.”

“You mean to tell me you've burned through 6 months of billable hours gathering requirements and never discussed fault tolerance? You never asked if their current system provided guaranteed delivery + guaranteed execution?”

“We will be using a message queue that provides guaranteed delivery.”

“Without the guaranteed execution part, delivery doesn't matter. A distributed transaction manager like DECdtm allows you to have a single transaction which encompasses a message from an MQ Series message queue, multiple RMS Journaled file, RDB, ACMS, and every other product which was developed to participate in a DECdtm transaction. If that transaction is spread across nine nodes in a cluster and one of those nodes goes down mid-process, DECdtm will rollback the transaction. If the transaction was also part of an ACMS queued task, ACMS will re-dispatch the transaction up to N times until it either successfully executes or exceeds the retry count and has to be routed off to an error handling queue.”

“Oh. Well, those discussions will happen in a couple of months. I can tell you have a lot of experience on this platform, so I'm going to present you to the primary and one of them will get back to you for a technical interview.”

“You mean to tell me they actually have OpenVMS professionals?”

“Well, it will probably be someone from HR explaining the non-compete and other policies.”

I originally wrote the above essay when I was working on the second edition of “The Minimum You Need to Know to Be an OpenVMS Application Developer.” That edition won't be out for a couple of years and most of the people who buy/read that book live out the above scene far too often. It is more appropriate for me to place such a story here, where many of the problems will come from.

Oh, don't go getting all defensive now. If that statement offended you it is most likely because you don't know enough about the industry to understand the truth of the matter. We did not cover writing fault-tolerant applications in this book because doing so is nearly impossible using an xBASE file format and having no controlling engine providing a barrier between all applications and the data. Even if you used this library to create such an engine, you would have to ensure no file used an easily recognizable extension and that the engine had its own user ID which owned all data files and did not allow shared access. Even if you achieved all of these things, you would not have provided fault tolerance. These things fix only the data sharing and index problems which are notorious with xBASE data files.

Fault tolerance is as described above, the transactions continue no matter what fails.

Think about that line. It's not just an academic mantra. It is a business philosophy where a BRP (Business Recovery Plan) document isn't treated because the systems are designed to never allow the business to fail. You will find the vast majority of publicly traded companies either have a completely untested BRP in place, or have simply never gotten around to writing one. A token few companies design applications and infrastructure to survive anything.

When the twin towers fell on 9/11 there were companies in those buildings using every operating system known to man. The companies which were using distributed OpenVMS clusters hesitated for up to 15 minutes while the cluster made certain all hardware at the location had ceased to respond, then continued processing each and every transaction. No transaction was lost. Despite the loss of life and location, the company neither ceased operation nor went out of business. Every company basing its business on other platforms stopped doing business that day. Many never returned to business.

I'm telling this story in a book covering Java and xBASE development to provide you with some concept of the limitations your tools impose. Developers tend to become enamored with their choice of development tools. They run around trying to use them in situations in which they are completely inappropriate. I have sat in conference rooms where Clipper hackers proposed to use a system written in Clipper to meet all the accounting needs of a Fortune 500 company. They wanted all data stored on a single instance of a file server which had some minimal RAID capabilities and honestly believed that was "good enough." Don't you make this same mistake.

Early on in this book I walked you through the thought process which had me selecting xBaseJ as a development tool for a project. Granted, the project I ended up writing wasn't provided in this book, but it is out there. You can find it on SourceForge: <http://sourceforge.net/projects/fuelsurcharge/> I used the lottery application because I always use that application in order to learn or teach new tools. The thought process used to select the tools is the important part, not the application I ended up writing.

Re-read the conversation I had with the person who called about the contract. They had burned six months and not covered the most important question. They had not asked the first question which must be asked at the start of each requirements gathering session. "What is the expected availability of this system?"

Would it surprise you to learn that the project was for a healthcare company and that the system being "replaced" was handling patient records along with prescriptions and pharmacy dispensing? Would you be shocked to learn that many of the patients being handled by this system were suffering from HIV and could enter an emergency room on any given minute of any given 24-hour period and that it might not be a hospital where their doctor works or even be in their home city or country?

If you think you can use Java on a platform which doesn't provide a distributed lock manager integrated into the operating system kernel to deliver a system for that environment, you aren't any better than the person who called me about the project. In most cases, the combination of high availability and fault tolerance preclude the use of any kind of VM. In general, a VM designed to run on multiple platforms cannot make use of a distributed lock manager which was integrated into the OS kernel of one platform because the lesser platforms the VM runs on don't have a prayer of ever having such a tool. If you store 100% of all data in a relational database *which is native to the platform providing the distributed lock manager and integrated with said manager*, and you have a message queueing system which is integrated with the distributed lock manager, and a message dispatching system which is not only integrated with the distributed lock manager, but will rollback and re-dispatch the message when the process handling it hangs or dies, then and only then, can you think about using a VM-based language for development. Yes, there were a lot of ands in that sentence, and for good reason.

Before you can go out working in the real world, you need to know two things:

1. The limits of your tools.
2. The first question you ask is "What is this system's expected availability?"

Availability wasn't an issue for the lottery tracking application. It was meant to be for a single person and the database could be completely recreated from a CSV file in just a few steps. All of the data which went into the CSV file would be gleaned from some state lottery system Web page, so even the CSV could be recreated from scratch if needed. This type of lottery has drawings which happen, at most, a few times per week, so a recovery process which takes a day is fine.

Let's contrast those requirements with an application which must be able to provide medical records to any hospital in the world for any patient contained in the system. Do you think a 1-2 day recovery period is acceptable? Just how long do you think the emergency room has when a patient comes in already seizing and the staff needs to identify which medication the patient has been on before giving him something to help with the seizing? How do you accomplish system and data backup yet provide the necessary availability? How do you do OS and hardware upgrades without impacting availability?

As the availability requirement goes up, your tool choices go down and the number of mandatory questions rises. With a 1-2 day recovery period, we didn't ask about hardware and operating system upgrades because they didn't matter. When a system is needed 24x7x365 these questions become important. Never make the mistake of assuming you can do anything you want with "language A" or "tool Z." The availability requirements dictate the tools. If management cannot be made to understand this, you have to either educate them or leave without warning.

Appendix A

Answers to Introduction Review Questions:

How many fields did dBASE III allow to be in a record?

128

What general computing term defines the type of file an xBASE DBF really is?

Relative file

What does xBASE mean today?

It refers to the data storage format used by various applications.

What was the non-commercial predecessor to all xBASE products?

Vulcan

In terms of the PC and DOS, where did the 64K object/variable size limit really come from?

The LIM (Lotus Intel Microsoft) EMS (Expanded Memory Standard)

What company sold the first commercial xBASE product?

Ashton-Tate

Is there an ANSI xBASE standard? Why?

No

Each of the vendors wanted its own product to be the standard put forth by ANSI and they refused to reach any compromise.

What is the maximum file size for a DBF file? Why?

2GB. That is the maximum value for a 32-bit signed integer.

What was the maximum number of bytes dBASE III allowed in a record? dBASE II?

4000 bytes; 1000bytes.

What form/type of data was stored in the original xBASE DBF file?

Character. Numeric fields were converted to character representation before storing.

Can you store variable length records in a DBF file?

No.

Does an xBASE library automatically update all NDX files?

No. It is only required to update those which are both opened and attached.

What is the accepted maximum precision for a Numeric field?

15.9: Total width of 15 with 9 digits of precision.

What is the maximum length of a field or column name?

10 characters

Answers to Chapter 1 Review Questions

What two situations force a user or application to physically remove deleted records?

- 1) The DBF reaches the maximum file size.
- 2) The disk holding the data file runs out of free space.

By default, what are string and character fields padded with when using xBaseJ?

Null bytes.

If you have a DBF open with NDX files attached to it then call a subroutine which creates new NDX objects for those same files and calls reIndex() on them, will the changes to the index files be reflected in the NDX objects your DBF holds? Why or why not?

No.

NDX objects load the entire Btree into RAM and do not monitor data file changes.

What two Java classes do you need to use to build create a report line making the data line up in columns?

StringBuilder and Formatter.

How does one tell xBaseJ to pad string and character fields with spaces?

Util.setxBaseJProperty("fieldFilledWithSpaces","true");

What DBF class method physically removes records from the database?

pack()

What is the maximum size of a DBF file?

2GB

What DBF class method is used to retrieve a value from a database Field regardless of field type?

get()

After creating a shiny new DBF object and corresponding data file, what method do you use to actually create columns in the database?

addField()

What DBF class method is used to assign a value to a database Field?

put()

What DBF class method do you call to change the NDX key of reference?

useIndex()

What DBF class method ignores all indexes and physically reads a specific record?

gotoRecord()

When you delete a database record, is it actually deleted?

No, it is flagged as deleted.

What DBF class method sets the current record to zero and resets the current index pointer to the root of the current index?

startTop()

What is the main difference between readNext() and findNext()?

readNext() requires a key of reference to have been established via some other I/O operation and findNext() does not.

What function or method returns the number of records on file?

getRecordCount()

What happens when you attempt to store a numeric value too large for the column?

A truncated version of the value is stored

What happens when you attempt to store a character value too large for the column?

An exception is thrown

When accessing via an index, how do you obtain the record occurring before the current record?

findPrev() or readPrev()

What DBF method returns the number of fields currently in the table?

getFieldCount()

When retrieving data from a database column, what datatype is returned?

String

What is the maximum length of a column name for most early xBASE formats?

10

What does the instanceof operator really tell you?

Whether an object can be safely cast from one type to another.

Are descending keys directly supported by xBaseJ?

No.

What NDX method can you call to refresh index values stored in the NDX file?

reIndex()

What Java String method allows you to split a String into an array of Strings based upon a delimiting String?

split()

Do NDX objects monitor database changes made by other programs or users?

No

Can you "undelete" a record in a DBF file? If so, why and for how long?

Yes.

Records are only flagged as deleted; they are not physically removed until a pack() is performed on the database.

When a Numeric field is declared with a width of 6 and 3 decimal places, how many digits can exist to the left of the decimal when the field contains a negative value?

1: you must allow one space for the decimal and one space for the negative sign.

When do you need to create a finalize() method for your class?

Whenever you allocate system resources like files or physical memory directly instead of by the JVM or using a class which already provides a finalize() method.

What Java class provides the readLine() method to obtain a line of input from a text file or stream?

BufferedReader

Do xBASE data files provide any built-in method of data integrity?

No.

What must exist, no matter how the data is stored, to provide data integrity?

An engine or other service through which all data access is routed without exception. That engine or service is the only method of enforcing data rules.

Answers to Chapter 2 Review Questions

What does CUA stand for?

Common User Access

What is the default look and feel for Java applications?

Metal

What DBF method tells you if a record has been deleted?

deleted()

Under what conditions is it okay to load DBF contents into a spreadsheet?

When the data is local and being loaded in read-only mode.

After opening an existing DBF and attaching one or more existing NDX files, what should you do?

reIndex()

Are the various Field variable names required to match their corresponding xBASE column names?

No

What interface must a class implement in order for it to be used with an Arrays.sort() call?

Comparator

Does the statement:

```
MegaXDueElms d_elms[] = new MegaXDueElms[ELM_COUNT];
```

completely allocate an array of MegaXDueElms or do you still have to create each individual element?

Create

If create, what is the syntax to create an element?

```
d_elms[i] = new MegaXDueElms();
```

What String method must be used when attempting to convert the result of a get() for a NumField to an Integer()? why?

trim()

Because parseInt() cannot handle non-numeric characters like spaces.

What javac command line option restricts the content of your source code?

-source