1 Programming Java for OS X

What's so different about Java on a Mac? Pure Java applications run on any operating system that supports Java. Popular Java tools run on OS X. From the developer's point of view, Java is Java, no matter where it runs.

Users do not agree. To an OS X user, pure Java applications that ignore the feel and features of OS X are less desirable, meaning the customers will take their money elsewhere. Fewer sales translates into unhappy managers and all the awkwardness that follows.

In this book, I show how to build GUIs that feel and behave like OS X users expect them to behave. I explain development tools and libraries found on the Mac. I explore bundling of Java applications for deployment on OS X. I also discuss interfacing Java with other languages commonly used on the Mac.

This chapter is about the background and basics of Java development on OS X. I explain the history of Java development. I show you around Apple's developer Web site. Finally, I go over the IDEs commonly used for Java development on the Mac.

1 In This Chapter

- Exploring the history of Java on Apple computers
- Installing developer tools on OS X
- Looking at the Apple Developer Connection (ADC)
- Introducing Java IDEs available for OS X

Reviewing Apple Java History

Apple embraced Java technologies long before the first version of OS X graced a blue and white Mac tower. Refugees from the old tan Macs of the 1990s may vaguely remember using what was called the MRJ when their PC counterparts were busy using JVMs.

MRJ stands for Mac OS Runtime for Java. MRJ was Apple's version of the JVM.

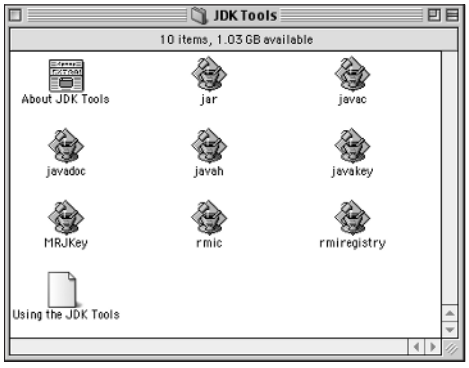
Classic Macs running OS 8 and earlier had a wonderful GUI. Macs were famous for their GUIs. What Macs were not famous for were their shells and command line interfaces. Old versions of the Mac OS were not Unix-based or Unix-friendly.

Those were the wild days before Java-friendly IDEs such as Xcode, Eclipse, and NetBeans ruled the world. Java used (and still uses) command-line tools, such as 'java,' 'javac,' and 'jar.' These tools did not have GUI equivalents. Apple filled the gap with GUI equivalents

of the most useful Java tools named after their command line counterparts. Figure 1.1 shows the MRJ folder with Apples GUI versions of the Java tools.

Figure 1.1

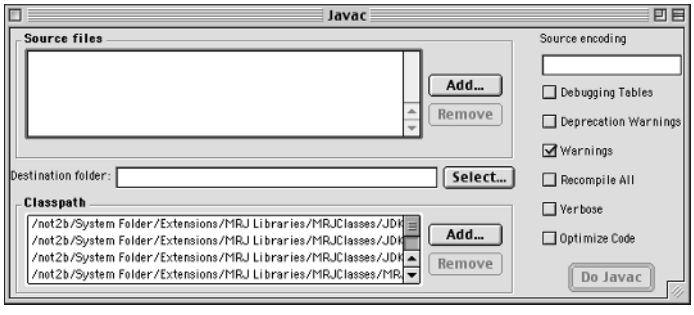
Classic Mac OS folder containing the MRJ GUI versions of the Java command-line tools



The javac command-line tool found on Windows or Unix had a GUI tool on the Mac, as shown in Figure 1.2. To compile a Java class, you double-clicked the javac application. You were presented with a form to fill out. After adding the source files, destination folder, and classpath desired, you clicked the Do Javac button.

Figure 1.2

Classic Mac OS MRJ javac tool



Interfacing Java and native C code was another hurdle. Under Mac OS 8, a technology called JDirect provided access to native C code on the Mac. JManager allowed C-based programs to

invoke Java. JNI was also available, but JDirect and JManager were meant to be easier to use for beginners.

With the new millennium came a new Mac OS: OS X is built on top of Darwin. Darwin provides a shell that Java's command-line tools run from. Apple added the Terminal application to OS X, giving access from its top-notch GUI to Darwin's shell. OS X came with new Java Cocoa APIs that provided easy access to OS X libraries from Java applications. Xcode arrived and turned out to be a Java-friendly IDE. Even OS X's new Interface Builder provided tools for easy creation of OS X-specific Java GUIs.

The classic OS and the MRJ began to disappear. Java programmers had a new arsenal of Java tools and libraries on OS X, and life was wonderful.

After years of real-world use, Apple discovered that Java programmers creating applications for Mac OS X used Swing and AWT for their Graphic User Interfaces instead of the Interface Builder and the Cocoa APIs for GUI development. Also, advanced Java programmers integrated with Apple's Cocoa libraries using JNI instead of Apple's custom bridges.

Because they were not needed, the Java Cocoa libraries were deprecated. Support for building Java UIs from inside of Interface Build was also removed. However, Xcode still supports Java development with several Java project templates built into Xcode. Also, diehard Mac OS X Java programmers can always use JNI to interface with Apple's Cocoa APIs.

CROSS-REF

Java Native Interface programming specific for Mac OS X is discussed in depth in Chapter 10.

NOTE

JNI is not the only technology for interfacing Java with Mac OS X-specific technologies. In this book, I explain Java integration with JNI, AppleScript, JavaScript, and the remaining non-deprecated Cocoa Java libraries. Integrating Java code with most OS X technologies is possible with a small nudge in the proper direction.

Apple continues to update Java for OS X. Many Java applications are distributed and tested specifically for OS X. Java is alive and strong on the Mac.

Installing the OS X Developer Tools

A suite of high-grade developer tools ship with every Mac. These tools include Xcode, GCC, Dashcode, and other useful GUI and command-line tools. Xcode is a top-notch IDE with built-in Java support. Ant and the command-line Java tools are also included. The full suite of development tools is free with every Mac.

NOTE

Ant is used by Xcode to build Java projects.

NOTE

Xcode has several predefined Java project templates. Predefined Xcode Java templates include a basic GUI application template, a JNI template, and a command-line tool template.

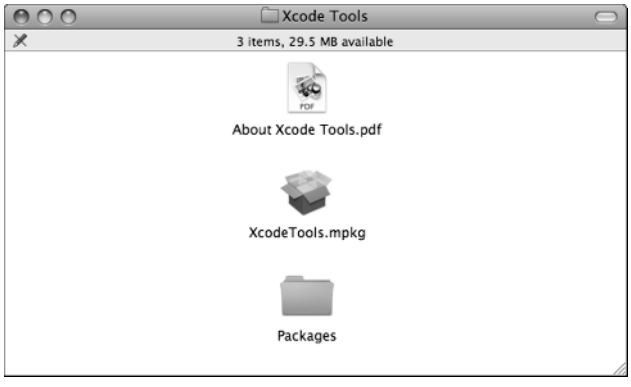
The OS X developer tools are not installed by default, because only developers find the tools useful. Install the developer tools using the following instructions:

1. **Insert your Mac OS installation DVD.**
2. **Navigate to your** XcodeTools **installer.**

   You should find a folder called `Optional Installs` at the top level. Inside that folder is the `Xcode Tools` folder. Finally, in the `Xcode Tools` folder, you find the `XcodeTools` installer, as shown in Figure 1.3.

Figure 1.3

Install DVD Xcode Tools folder



3. **Start up the** XcodeTools **installer.**
4. **Continue through the installer, and choose Custom Install.**

   You see five packages on the Custom Install screen: Developer Tools Essentials, System Tools, UNIX Development Support, Mac OS X 10.3.9 Support, and WebObjects.

5. **Select Developer Tools Essentials, System Tools, and UNIX Developer Support.**
6. **Click continue, and finish installing the developer tools.**

   You are now prepared to learn Java development on OS X.

During the OS X tools installation, developer tools Essentials is selected for you, as shown in Figure 1.4. Essentials contains Xcode, Interface Builder, Dashcode, and GCC.

TIP

The default installation location of the OS X developer tools is the `Developer` directory at the root of your primary drive. You have the option of changing the installation location of the developer tools, but I recommend sticking with the default `Developer` directory. Sticking with the `Developer` directory prevents confusion later when documentation says to look there for something that you don't realize you installed elsewhere.

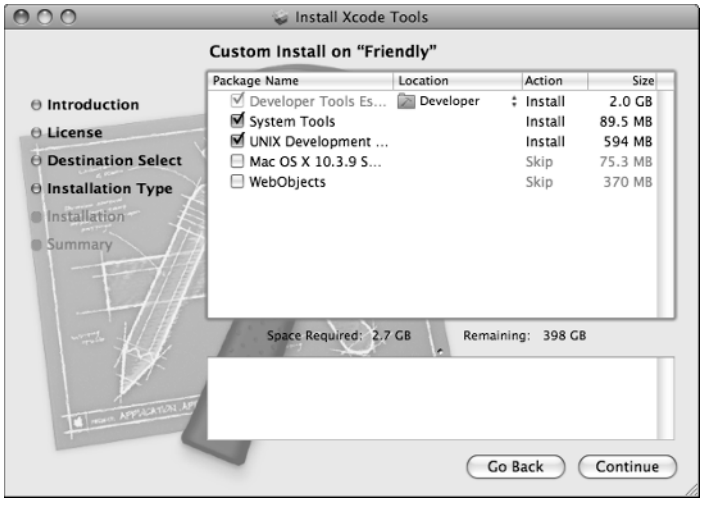
System Tools is a collection of applications that helps you debug and analyze your applications. Unix Development Support installs UNIX tools in the `/usr` directory. Make sure you select these two packages during installation of the developer tools. You need these tools later in the book.

Support for Mac OS X 10.3.9 may be useful to you. Some projects require support for older OS versions. You do not need the Mac OS X 10.3.9 Support package for this book.

WebObjects allow development for Apple's proprietary Java Web server. These applications are typically deployed on Mac OS X Server. WebObjects and Web applications are very large topics and are not covered in this book.

Figure 1.4

Install Xcode Tools screen



Exploring the Apple Developer Connection

Apple provides a wealth of Developer Articles, software seeds, developer news, and mailing lists for the community of developers making OS X their home. Apple consistently has high approval ratings from its customers. The company is often described as having a cult-like following of customers and developers.

After getting to know your way around Apple's Developer Connection, you may appreciate the support Apple gets from customers and developers a little better. The Developer Connection is typically easy to navigate and user-friendly for anyone trying to create Mac OS X applications. Technologies as diverse as Python, Perl, and even Java find a home on OS X. The Developer Connection does not ignore this diversity.

Exploring Reference Library topics

Apple offers an extensive library for developers on the Developer Connection site. One of the libraries targets Java, but several of the libraries are useful to Java developers. You do not need a membership in the Apple Developer Connection (ADC) to view the Reference Library.

The Reference Library is on Apple's Developer Connection site. The Developer Connection is at `http://developer.apple.com/`. Get to know your way around the Web site. Knowing the ins and outs of the Developer Connection will save you many random searches of the Internet. Follow these steps to find the library:

1. **Open** http://developer.apple.com **with Safari or the Web browser of your choice.**

   You are greeted by the Apple Developer Connection home page.

2. **Look at the top of the Web page for the Dev Centers drop-down menu.**

3. **Select Mac Dev Center from the drop-down menu. This brings you to the Mac Dev Center Web page.**

4. **Select the link to the Mac Reference Library on the Mac Dev Center Web page.**

Looking over the Mac OS X Reference Library Web page, you first notice the list of Reference libraries on the left. More than 50 reference libraries are listed by topic, and alphabetically. Other libraries worth noticing are Compiler Tools, User Experience, System Configuration, Screen Saver, Carbon, Cocoa, and Preference Panes. All these topics and more are covered in this book. These libraries are a great resource for learning additional details after reading this book.

NOTE

Finding a link to the Developer Connection on Apple's main Web site is difficult and maybe even impossible. You probably want to bookmark or memorize the address. The Developer Connection is found at:

`http://developer.apple.com/`

The Mac OS X Reference Library Web page contains a prominent tabbed pane containing Overview, Getting Started, Required Reading, and Featured tabs. Browsing these tabs gives you a quick overview of OS X development.

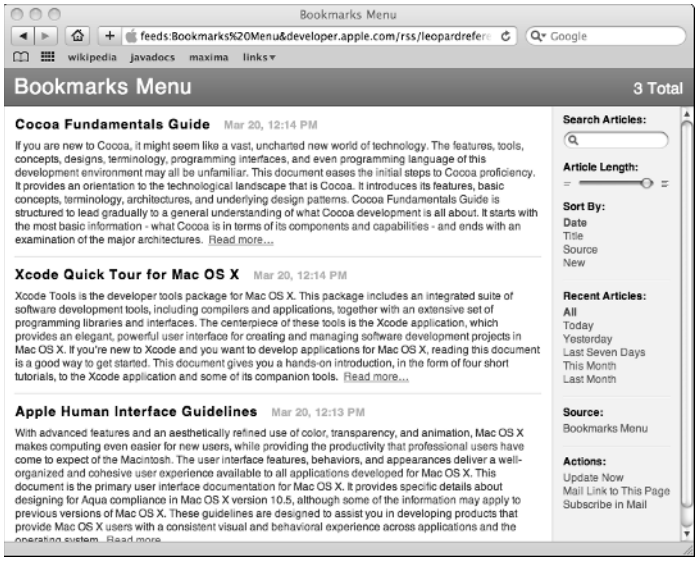
Under the tabbed view is a searchable list of documents available for OS X development. Enter **java**, and search. The search returns more than 60 Java-related documents on the developer site. The search returns a few JavaScript documents, but most are actual Java development articles and documents.

Hidden at the bottom of the page is a link to the RSS Feeds page. Traditionally, checking for the latest and greatest news and tips required navigating to the page with the news every day or multiple times each day. You can now avoid wasted random trips to the Reference Library just to see if new articles of interest have appeared.

RSS stands for Really Simple Syndication. RSS is simply a custom news feed, in this case, from the Reference Library. Apple's Safari Web browser contains a built-in RSS reader, as shown in Figure 1.5. Even better, bookmarking RSS feeds in Safari works the same as bookmarking Web pages.

Figure 1.5

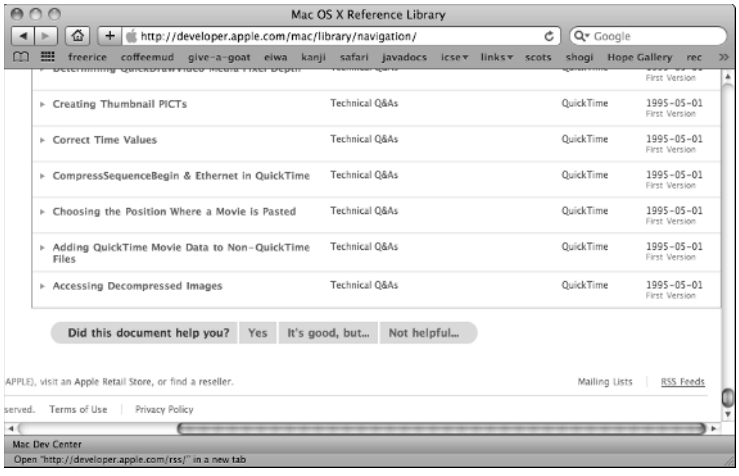
Built-in RSS reader of the Reference Library



If you browse to the Mac OS X Reference Library site with Safari, you see an RSS at the bottom right of the page, as shown in Figure 1.6. The RSS link is a little obscure, considering the utility it offers. Below are the steps for tracking the Reference Library RSS feed with Safari.

Figure 1.6

RSS link at the bottom right of the Mac OS X Reference Library Web page



1. **Click the RSS link to bring up a selection of RSS feeds offered by the Apple Developer Connection. The link is at the bottom-right corner of many of the Developer Connection pages.**
2. **Select an RSS feed from the list of Developer Connection feeds. ADC Headlines is a good choice for keeping up with the latest Developer Connection news.**

   The feed appears in Safari's built- in RSS reader.
3. **Bookmark the feed by selecting Add Bookmark... from the Bookmarks menu.**

Hidden at the bottom of the Mac OS X Reference Library page is the legacy documents link. Scroll down to the bottom of the left navigation links list and find the link entitled Legacy Mac OS X Reference Library. If you are tasked with fixing an older software project, this section is invaluable. The Legacy Documents page looks very similar to the regular reference library. However, the guides and code found there apply only to legacy development.

Finding developer articles

You are a Java programmer, and you want to see Java programming articles now. The Mac OS X Reference Library is the place to begin your search. The Developer Connection has more articles than you may ever read on Java and related topics. As I mention earlier, searching the Documents in the Mac OS X Reference Library brings up around 60 Java-related documents.

Many useful documents are not listed specifically as Java documents, so you need to browse around a bit. For instance, the Apple Human Interface Guidelines is invaluable for the creation

of Java applications that conform to expected OS X application behavior. The Apple Human Interface Guidelines is listed in the Guides documents. To quickly find the link, click Guides in the left navigation of the Mac OS X Reference Library Web page, and then type **human** in the Documents search field. The remaining listed link is for the Apple Human Interface Guidelines.

TIP

The Developer Connection has three Dev Centers. The Dev Centers are targeted for iPhone, Safari, and Macs. Most of the Java information you need is found in the Mac Dev Center.

Obtaining software seeds

To take full advantage of the Apple Developer Connection, you need an ADC Membership. Until you have an ADC membership, you cannot download software seeds or beta releases. Look around the Apple Developer Connection Mac Dev Center page to find a link to register for ADC membership.

Alternately, you may click "Log in" at the top of many of the Developer Connection pages. The login page has a link for "Join now." This allows you to register, too.

TIP

Paid memberships are encouraged, so some pages about membership do not have a big blinking button shouting, "Join here for free!" Just poke around a little, and you will spot the free Online membership link. On the `http://developer.apple.com/products/membership.html` Web page, the link is at the bottom-left corner under the title ADC Online Membership.

Currently you can choose from four levels of membership: Premier, Select, Student, and Online. The Online membership is free, but it has fewer benefits than the other paid memberships. Online (free) memberships have access to Introductory Videos for Coding and beta releases of new JDKs. Premier membership, costing around $3499, provides a World Wide Developer Conference (WWDC) ticket, access to Apple's compatibility labs, discounts on hardware purchases, and more advanced technical support.

To become a member, you must accept a membership agreement. The ADC membership agreement doesn't require you to spend time washing windows at the local Apple Store, but you're wise to read through the agreement anyway to see if you feel comfortable with it. Typically, the agreement prohibits you from talking about seeds and beta releases that you download, other than to technical support. See the agreement for full details.

If you choose the free ADC membership, log in and look around. Often, you have access to previews of future JDKs for OS X. If you are previewing these future Java releases and find bugs in them that affect your programs, be sure to create test cases and submit bug reports with the test cases to Apple. The developers at Apple will do their best to address them, and may even make your job easier by fixing bugs you report before they make it into a full Java release.

Apple distinguishes between the previews available with free ADC membership and the official Software Seeding Program. Much of the time, the online ADC membership is enough for Java

developers on OS X. However, if you want to test your software on seeds of the newest OS X beta or try out the newest pre-release of Xcode Tools, a paid membership with ADC monthly mailings is what you need.

Benefiting from membership

As mentioned earlier, you can choose from four types of memberships to the Apple Developer Connection: Online, Student, Select, and Premier. Online is free and gives you access to Apple Development Connection previews of some Java code in development. Having access to some previews is often seen as enough by Java developers on OS X. These previews are not as numerous as the actual Software Seeding Program.

If you download betas of JDKs or Xcode in development by Apple, remember that the non-disclosure agreement required by members usually prohibits you from speaking (or writing) about them to anyone except technical support.

TIP

You don't need an ADC membership to sign up for the Java developer technical discussions list. This is a Mac OS X specific list for Java development. Subscribe at:

```
http://lists.apple.com/mailman/listinfo/java-dev/
```

NOTE

At one time, discussion of beta releases of Java on the Mac was allowed on the Mac Java developer list. Currently, discussion of Java previews is not permitted on the list because of the ADC non-disclosure agreement.

Premier and Select members may get the Software Seeding Program in addition to their membership. Seeds often include pre-release versions of operating systems, betas of company products, development kits, and development tools. Currently, these may be delivered electronically or by snail mail on DVDs.

ADC on iTunes and Coding Headstarts are two more benefits of ADC membership. ADC on iTunes provides videos of training sessions for developers. These videos are (surprise, surprise) viewable on iTunes. Coding Headstarts are videos dedicated to teaching developers techniques for adding features to OS X software. The Online and Student Members have limited access to these benefits. Premier and Select members have full access to both ADC on iTunes and Coding Headstarts.

Student, Select, and Premier members have access to hardware discounts. A special version of the Apple Store is available for these purchases. Of course, the number of times the discounts can be used scales with the level of membership. Currently, Premier can purchase 10 systems per year at the discount, Select one per year, and Student one system (ever, not per year).

All ADC members can purchase extra technical support. Premier and Select members get a limited number of technical support incidents per year but can purchase additional support as needed. The available technical support is one-on-one time with Apple engineers selected to help you with your specific issues. If you are a diehard cutting-edge OS X developer, you may find this included personalized support useful. I have never needed it myself. (I like to think that is because Macs are such a good development platform.)

Avoiding Deprecated Java Cocoa Libraries

Apple provides several Java classes directed at developers seeking to create Java desktop applications that feel and behave like native applications. Apple's Java classes typically inhabit packages labeled `com.apple`. The number of `com.apple` packaged classes has dropped significantly over the last few years due to older classes being deprecated and removed. Also, Apple's native GUI designer, Interface Builder, has dropped direct support for Java programming.

It is tempting to believe the number of deprecated classes and lack of Interface Builder support on OS X for Java is an indication that Java applications with native behavior can no longer be created for OS X. The changes to Java development on OS X are rather the acknowledgement of real-world use by Apple of development tools. These changes have occurred because Apple understands how the majority of Java developers on their OS create native-feeling applications.

This section explains the streamlining process that has resulted in the current state of Java applications development on OS X.

Understanding the history of Java Cocoa libraries

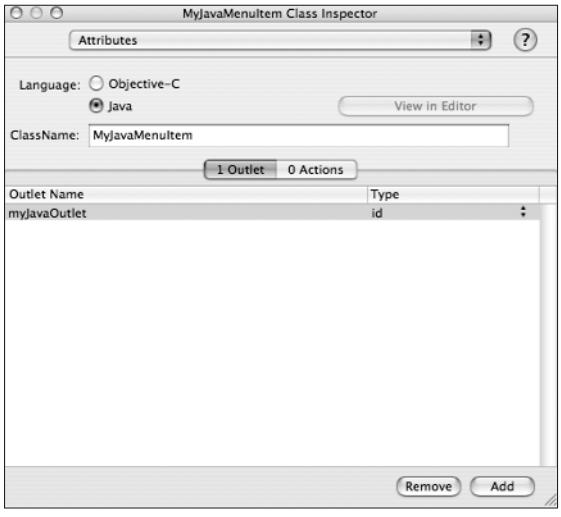
At the turn of the millennium when Apple introduced OS X, Java was still in its infancy. Java was an exciting buzzword in the development world. Apple already supported Java on OS 8 and OS 9. Apple was excited to continue supporting Java in the OS X environment.

Few of the Java GUI design tools commonly used now existed then. The GUI tools that developers used contained little ability to customize Java desktop applications to look and behave like native applications. This especially applied to OS X.

Mac application developers used a new GUI creation utility built for OS X called Interface Builder. Interface Builder allowed for drag-and-drop creation of application interfaces; mouse-driven creation of connections between GUI elements and class place holders; and then either Java or Objective-C class generation, as shown in Figure 1.7, depending on the type of project under development. Creating Java applications with a native look and feel was easy with this tool, because the GUI was actually native.

Figure 1.7

Legacy Interface Builder's Class Inspector with Java option selected



Apple's Java Bridge was still a commonly used tool for interfacing Java with Cocoa libraries. The Java Bridge was easy to use for simpler applications. JNI and the Java Bridge were both used depending on the complexity of the application developers built.

The OS X API, called Cocoa, came in two flavors: Objective-C and Java. The API was further divided into the Application Kit and the Foundation API. The Application Kit contained mostly classes that wrapped Apple's native GUI components. The Foundation API contained classes that supported the Application Kit and classes that contained functionality found in Cocoa, but not in standard Java.

With the release of OS X 10.4, Apple announced that the Java Bridge and the Java Cocoa Frameworks were deprecated. Apple dropped the Bridge in favor of JNI. The 200+ Java Cocoa Framework classes and interfaces were deprecated and replaced by seven Apple Java Extension classes.

Many Java programmers fond of OS X felt this was the end of native-feeling applications written in Java. However, Java is alive and well on OS X. As I show in this book, Java still interfaces with Apple native technologies through Apple Java Extensions, JNI, AppleScript, and JavaScript.

Reviewing deprecated libraries

The deprecated Java Cocoa Framework contained over 200 classes and interfaces. These are deprecated and should not be used. If you inherit a Java project that uses these classes and

interfaces, begin refactoring the code to use pure Java with Apple's Java Extensions and other Apple-supported Java Technologies. This section is intended to give you a quick overview of the legacy Java Cocoa Framework, so you have some idea of where to begin the refactoring process. The rest of this book is intended to give you knowledge necessary to write Java code that seamlessly integrates with current Apple-supported technologies.

Apple currently provides support for several Java packages. They include `javax.script`, `com.apple.eawt`, and `com.apple.eio`. If your project contains packages beginning with `com.apple.cocoa`, the packages need to be refactored out of your program. The `com.apple.cocoa` packages are always part of the legacy Java Cocoa Framework. Any classes contained in the `com.apple.cocoa` packages will cease to function on OS X at some point.

NOTE

javax.script is supported by Apple as the natural interface for AppleScript and Java. com.apple.eawt provides classes that make Java GUIs behave like native OS X applications. com.apple.eio contains classes that access OS X features that do not have parallel features in the standard Java APIs.

The Cocoa libraries are split between the Application Kit framework and the Foundation framework. The Foundation framework provides base classes and utility classes that form the "foundation" of Cocoa applications. The Application Kit framework provides GUI related classes.

The Java Foundation package is `com.apple.cocoa.foundation`. The Foundation classes are made up of useful utilities, data types, and classes that support Cocoa design patterns that did not exist in pure Java.

For example, several of the Java Foundation classes come in mutable and immutable varieties. Simply put, mutable mean changeable and immutable means final, in the Java sense of final. Having two like named classes, one optimized for changing and the other optimized for use without changing was uncommon in Java 1.1, back in 2000. Cocoa used this paradigm, so Java Foundation classes were created to match up with their Objective-C counterparts.

The Foundation framework's `NSObject` is worth special notice. `NSObject` is the root object of Cocoa Java classes. Think of it as the Cocoa counterpart to the Java `Object` class. `NSObject` has similar functionality as `Object`, such as cloning, equality comparisons, and hashing.

The Java Application Kit package is `com.apple.cocoa.application`. The Application Kit provides classes that represent Apple's OS X native GUI components and events. Some of the native widgets include `NSAlertPanel`, `NSComboBox`, and `NSMenu`.

NOTE

You may have noticed that the Cocoa Java classes by convention start with NS and not OSX as expected. NS stands for NextStep. NextStep was an OS that had its origins in the 1980s. NextStep evolved into OpenStep which in turn evolved into OS X. The NS naming convention is a reminder that OS X is not a descendent of the classic Mac OS.

The Applications Kit's `NSApplication` controls the Cocoa application event loop. Whether opening files, terminating the program, or showing help, `NSApplication` handles the events. `NSApplication` uses delegates to listen for applications events. Assigning methods in other classes to handle methods as delegates replaces the need to subclass `NSApplication` or provide an interface implementation as you see in pure Java applications when handling events. `NSApplication` is not subclassed normally.

NOTE

The current Javadocs for com.apple packages is located at:

```
http://developer.apple.com/documentation/Java/
Reference/1.5.0/appledoc/api/index.html
```

NOTE

The legacy Java Foundation classes and interfaces are documented at:

```
http://developer.apple.com/documentation/
LegacyTechnologies/Cocoa/Reference/Foundation/
Java/index.html
```

NOTE

The legacy Java Application Kit classes and interfaces are documented at:

```
http://developer.apple.com/documentation/
LegacyTechnologies/Cocoa/Reference/ApplicationKit/
Java/index.html#//apple_ref/doc/uid/20001094
```

The Cocoa Java paradigm felt very different from pure Java because it was written to match up with its Objective-C counterpart. With any luck, you will never need to refactor an old Java Cocoa application to be more of a pure Java application. If the application view is simple, you likely are better off creating a Swing view and rewriting the interface to your Java controller from scratch. It will save you lots of time and frustration.

Understanding why Java Cocoa libraries were redundant

The Java-based Cocoa Framework met a need that existed when OS X was first released. Along with the Java Bridge, the Java Cocoa Framework allowed for easy integration of Java code with native OS X frontends. Swing and AWT, at the time, were still a bit clunky and buggy. At the time, it was common for even diehard Java developers to refer to Java GUIs as "write once, debug everywhere." The OS X user interface was solid, groundbreaking, and beautiful. The Java Cocoa libraries allowed Java programmers to take full advantage of this elegant new operating system.

The Java Cocoa Framework, Interface Builder, and the Java Bridge were useful for simple communication between Objective-C and Java code. Soon it became clear that these two languages were too different for easy communication. The Java Bridge could not translate advanced behavior between code written in these two languages. Also, the libraries of GUI components provided with the Cocoa Framework could not be mixed and matched with Swing or AWT components. Unexpected crashes appeared if they were.

As OS X matured, so did Java. Swing and AWT views became easier to create and less buggy. Better tools for generating Java GUIs emerged. Java developers on OS X showed a definite preference toward using Swing, AWT, and third-party libraries over Interface Builder and the Java Cocoa libraries. With the proper subset of Java classes supported by Apple, JNI, and interfaces with JavaScript and AppleScript, no other technologies are needed to make a Java application feel like a fully native application.

Exploring Available IDEs

In the early days of Java development on OS X, the only choice for an OS X Java IDE was Xcode. Now several are available. The three most common are Netbeans, Eclipse, and Xcode. All three are excellent IDEs. All three are free (as in food). Eclipse and Netbeans have the advantage of being Java-centric environments. Xcode has the advantage of being OS X centric.

NOTE

Two common types of free software exist. Software that is referred to as "free as in food" is software that costs nothing. Software that is referred to as "free as in freedom" is software that makes its source code available. The second type of software sometimes costs money. The two types of "free" software are not mutually exclusive. Software may be "free as in food" and also "free as in freedom."

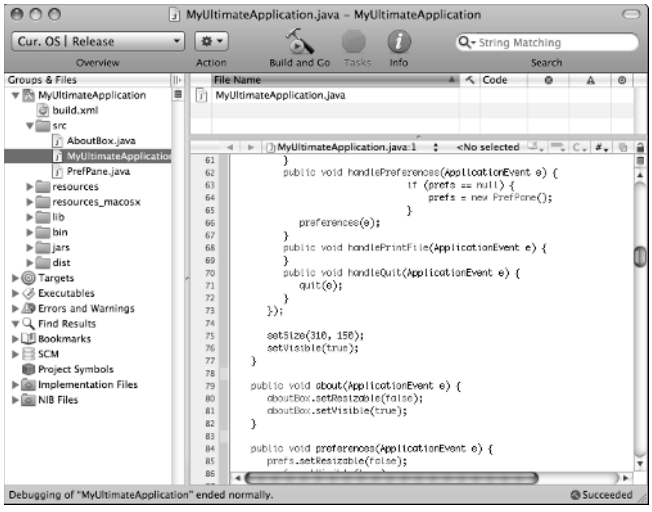
Many arguments have occurred over which of the three free IDEs should be used on OS X. All three are excellent IDEs and have common IDE features, including line numbering, project templates, debuggers, and version control integration. I tend to use a combination of Eclipse and Xcode (and TextEdit) when developing on OS X. NetBeans is also a valid option. In this section, I discuss the benefits of all three.

Developing with Xcode

If you intend to write an application that is heavy in Java code, but also integrates with native OS X libraries or applications, you should consider using Xcode as your development environment. Figure 1.8 shows Xcode. Xcode and related tools allow for easier integration of pure Java with OS X features. In this book, I use Xcode for most examples. Several of the chapters in this book explain the integration of Java with Xcode-specific technologies. Install Xcode to follow along more closely with the example code in this book.

Figure 1.8

Xcode IDE



NOTE

Install Xcode for free from your OS X installation disk.

Xcode ships with six Java-specific templates. These templates are for Java Applets, applications, JNI applications, signed Applets, command-line tools, and Web Start applications. Programmers interested in Java Enterprise Edition development of Web applications with JSP, Servlets, and Enterprise Java Beans often choose NetBeans or Eclipse as their preferred environment. Xcode is used frequently by developers of Java desktop applications and Java client applications.

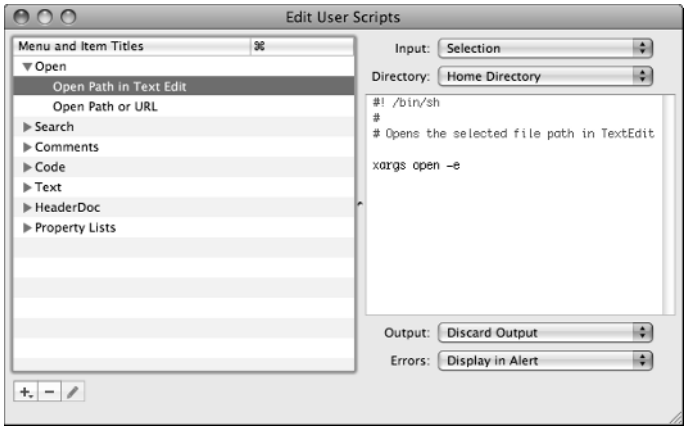
Xcode has built-in support for Source Code Management (SCM). CVS, Subversion, and Perforce are supported by the default install. Secure SSH connections to code repositories are supported also.

Xcode is highly customizable. As is expected in a modern IDE, many preference settings control editing, builds, and code versioning. As shown in Figure 1.9, the scripting menu supports the reorganizing, addition, and editing of custom and built-in scripts.

My favorite feature of Xcode is the ability to automatically package resources, icons, and libraries into OS X application bundles. Native OS X applications are actually folders with a structured set of files. The folders look and behave like double-clickable executable files to users on OS X. Application bundles are the preferred distribution method for applications on OS X. If you are distributing your applications as double-clickable JAR files, use of the application feels awkward to OS X users.

Figure 1.9

User Scripts dialog box in Xcode

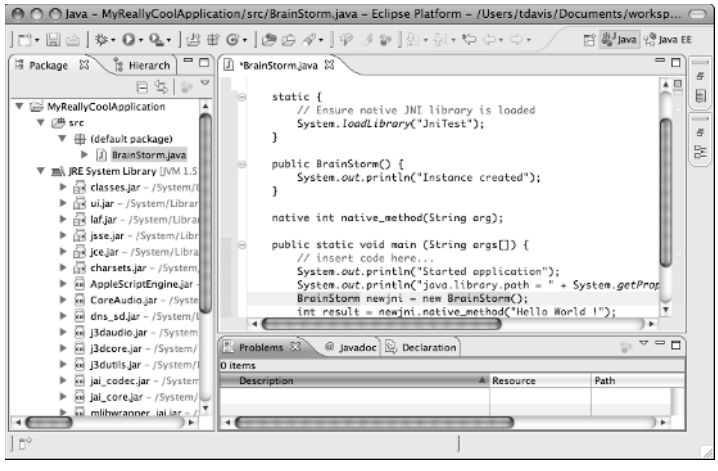


Developing with Eclipse

Eclipse, shown in Figure 1.10, is available on multiple operating systems. A big appeal in using Eclipse is that if you are required to use other operating systems such as Linux, you can still use the IDE you are comfortable with. Eclipse uses SWT instead of Swing for its interface. SWT is a library that is OS specific. Versions of SWT are available for most major operating systems.

Figure 1.10

Eclipse in Java perspective



NOTE

Installing the Enterprise Java bundle of Eclipse from `http://www.eclipse.org/` allows you to create client-server applications and traditional desktop applications.

NOTE

Many developers who use Eclipse swear by JFormDesigner for Java GUI development. JFormDesigner is made by FormDev Software found at:

```
http://www.formdev.com/
```

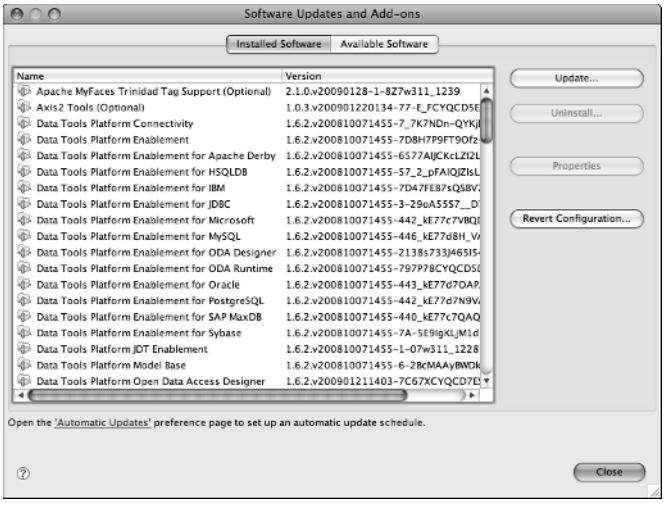
Eclipse has distributions and modules for many programming languages other than Java, but it shows its real strengths in creating and maintaining Java-based projects. Projects in Eclipse can be created based off Ant build scripts or simply using default project templates.

Eclipse makes heavy use of tabbed perspectives. Eclipse has perspectives for Java, Debugging, Java EE, SVN, Database Development, and many more. Swapping between multiple views during development of one project is common.

Updates and custom add-ons to Eclipse can often be accomplished from the Eclipse Software Updates and Add-ons dialog box, shown in Figure 1.11. Very rarely do modules used by Eclipse require anything more than a URL and a few clicks of the mouse before complete integration into Eclipse is accomplished.

Figure 1.11

Eclipse Software Updates and Add-ons dialog box

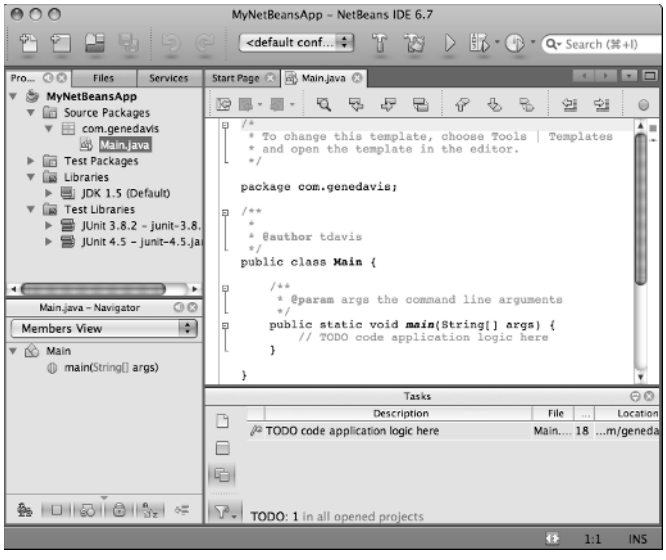


Developing with NetBeans

Years ago, NetBeans was considered the slowest of the three IDEs. These days, NetBeans responds just as you would expect any well-behaved application. NetBeans has a large user base and is sponsored by Sun. NetBeans, shown in Figure 1.12, is a strong contender for Java developers on OS X.

Figure 1.12

Netbeans on OS X



NOTE

NetBeans IDE is available in several flavors from:

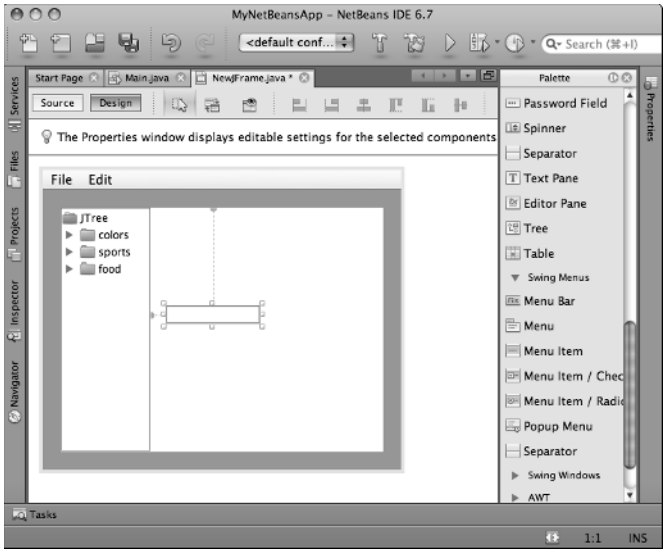
`http://www.netbeans.org/`

NetBeans supports GUI development with the Swing GUI Builder. Swing GUI Builder provides a drag-and-drop approach to GUI creation. Drag Swing components from the palette to the provided canvas. Swing GUI Builder, shown in Figure 1.13, comes free with NetBeans. Free is great when your budget is tight.

As with Xcode and Eclipse, NetBeans supports C/C++ and quite a few other languages including JavaScript, PHP, Ruby, and Python. NetBeans supports traditional application development and enterprise Web application development. Choose from seven different bundles on the download page at `http://www.netbeans.org/downloads/index.html`.

Figure 1.13

NetBean's Swing GUI Builder



Summary

In this chapter, you read about the long history of Java programming on the Mac. Apple provided Java for its computers before OS X became the OS of choice for Macs. Apple introduced Java Cocoa frameworks to ease the transition of Java programmers from the classic Mac OS to OS X. After Java programmers made the move to OS X, Apple deprecated the Java Cocoa frameworks.

You explored the Apple Developer Connection Web site. ADC membership comes at several levels and with various benefits, depending on your level. Even free members have access to articles, downloads, and tons of Java-related reference material on Apple's developer Web site.

Three free IDEs for developing Java are available on the Mac. They are Xcode, Eclipse, and NetBeans. All three are excellent IDEs. However, I use Xcode for the projects and examples in this book.