Porting Objective-C to Java

by Theresa Ray of Tensor Information Systems, Inc.

Sponsored by Apple Computer, Inc.
Apple Developer Connection

media

Porting Objective-C to Java

by Theresa Ray of Tensor Information Systems, Inc.

A quick reference guide to the major topics and tips involved converting an Objective-C application or Framework to Java.

Many who have applications and/or frameworks written in Objective-C are finding it necessary to convert their Objective-C applications and frameworks to Java. Java is a widely known and supported language. Many programmers are already trained in Java, and in many cases it is becoming more time-consuming and costly to retrain Java programmers to learn Objective-C programming syntax and techniques than it is to convert the Objective-C applications and frameworks to Java.

Java applications can have as fast a response time as Objective-C applications, and are just as scalable as Objective-C applications. Other than the investment of time and resources (which is recouped in the long run since more programmers have the knowledge required to work on these applications and frameworks), there is little rationale for not converting an application or framework from Objective-C to Java.

Contents and Prerequisites

This survival guide covers some of the topics relative to conversion of an Objective-C application or framework to Java. It assumes that the application or framework was either written in WebObjects 4.0 or later, or has already been successfully converted to WebObjects 4.0 or later. Refer to "Converting an Existing WebObjects Application" for more information on converting a WebObjects application to WebObjects 4.0.

This guide also assumes that the reader is very familiar with the topics presented in the online documentation "Using the Java Bridge" (found in NEXT_ROOT/Documentation/Developer/YellowBox/TasksAndConcepts/ProgrammingTopics/JavaBridge/JavaBridge.html). The Java Bridge is a technology from Apple that allows communication between Objective-C objects and Java objects. While the two languages are both object-oriented and fundamentally similar, there are enough differences between them that the use of a bridge to allow communication between the different types of objects is necessary.

Conversion Philosophies

There are two possible approaches to the conversion of an Objective-C application or framework to Java. Neither approach is particularly better than the other is – the approach you choose depends on the complexity of your application or framework and how quickly you need a converted solution available.

The first approach is to provide a Java wrapper for all of your Objective-C classes, and then port each class one by one, changing the Objective-C side to use *classWithName*: to gain access to those classes already ported to Java (allowing the Java Bridge to assist you in your conversion). Once a class has been ported, you would remove it from the wrapper. If all you need is for other Java classes to be able to access your Objective-C methods, this might be the right solution for you.

If you intend to fully convert your Objective-C code to Java code but your application or framework is large or complex, you may want to choose this approach anyway because of the "divide and conquer" capability it provides.

This approach allows Java classes to have immediate access to your Objective-C classes. Then you can take the time you need to convert the Objective-C classes to Java and to thoroughly test each converted class. You would start with the peripheral classes, convert, test, approve, and then move on to the core classes. This method is slower, but it allows you more control over the conversion process. It enables testing and debug throughout the conversion process.

Note that for very large classes, you can even subclass them through the Java Bridge and port the class from Objective-C to Java on a method-by-method basis in the subclass. At the end of the conversion process, your super class is empty and you can just remove it.

Read the installation documentation "Using the Java Bridge" (found in NEXT_ROOT/Documentation/Developer/YellowBox/TasksAndConcepts/ProgrammingTopics/JavaBridge/JavaBridge.html) for more information on creating Java wrappers for your Objective-C classes.

The second approach - conversion of the code directly from Objective-C to Java - may be more useful if you have the less demanding time restrictions and/or the application or framework that needs conversion from Objective-C to Java is not particularly large or complex. Using this technique, you would wait until the conversion and testing of the application or framework is complete before releasing it. This is more of an "all or nothing" approach, but it has the advantage of being much faster since you do not have to maintain three projects at once (an Objective-C project, a wrapped Objective-C project and a Java project).

Java Syntax

In either case, at some point you will begin to convert Objective-C code to Java code. The most obvious difference between the two languages is in the syntax. Java APIs are provided for all of Apple's frameworks including the Foundation Kit, AppKit, WebObjects, and Enterprise Objects. The Java method name in most cases is the same as the Objective-C selector name after removing everything beyond the first colon.

For example, the Objective-C method

```
[ myEditingContext saveChanges ] ;
```

would map to the Java call

```
myEditingContext.saveChanges() ;
```

If an argument is required, just place it within the parentheses. For example, the Objective-C method

```
[ myEditingContext insertObject: newCustomer ] ;
```

would be replaced with the Java call

```
myEditingContext.insertObject(newCustomer) ;
```

Multiple arguments in Java syntax are placed successively within the parenthesis and separated by commas. Remember that the Java method by default is the Objective-C selector name removing everything after the first colon. Therefore the Objective-C method

```
[ myApplication pageWithName: @”Foo” inContext:
   myContext ] ;
```

would map to the Java call

```
   myApplication.pageWithName("Foo", myContext) ;
```

Nested messaging is accomplished in Java by separating the calls with a "." – for example the Objective-C code

```
   [[mySession defaultEditingContext ] insertObject: myObject];
```

would be replaced by the Java calls

```
   mySession.defaultEditingContext().insertObject(myObject) ;
```

Memory Management and Initialization

A major difference between Objective-C code and Java code is that in Objective-C code, the programmer is responsible for retaining any persistent object references, and releasing all objects that are no longer necessary. In other words, the programmer is responsible for explicitly managing memory using the retain, release and autorelease selectors (see the WebObjects Survival Guide on Memory Management for more information on Objective-C memory management).

In Java, unused objects are automatically garbage collected and the programmer need not concern himself/herself with the details of memory management. One exception is if you create a new thread in Java, and that thread manipulates some objects with Objective-C counterparts. In that case, you must explicitly create an autorelease pool at the beginning of the thread, and remove it at the end of the thread.

Allocation and initialization of objects differs between Java and Objective-C. In Objective-C, you create objects with alloc and initialize them with class initialization methods such as init. In Java, you create objects with new and initialize them with the class's constructor.

For example, to create an instance of the Foo class in Objective-C you would write

```
   someObject = [ [ Foo alloc ] init ] ;
```

while the same code in Java would be

```
   someObject = new Foo();
```

In Objective-C, initialization methods are instance methods, applied to newly created objects. In Java, constructors are not used in the same way as instance methods. Therefore, the Objective-C initialization method would become the Java constructor method. If you have multiple initialization methods in Objective-C (i.e initWithName: and initWithDefaultAdministrator), you will need to convert these two methods into two constructors. For example:

```
-  (id) initWithName: (NSString *) aName
{
   self = [ super init ] ;
   name = [ aName retain ] ;
   return self ;
}
```

and

```
-  (id) initWithDefaultAdministrator
{
   self = [ super init ] ;
   name = [ [ NSString stringWithString: @”Tom Jones” ]
      retain ] ;
   return self ;
}
```

might become the Java methods:

```
public SomeClass ()
{
   super() ;
   name=”Tom Jones” ;
}
```

and

```
public SomeClass (String aName)
{
   super() ;
   name=aName;
}
```

If you had created two initializers that took the same argument types, you would not be able to accommodate both initializers with constructors. Instead, you might try creating a Java “cover” method that exposes the initialization methods as instance methods rather than constructors and adding a custom pure Java constructor such as:

```
public Foo (String myString, boolean isPathName)
{
   if (isPathName)
   {
        this.initWithPathName(myString);
   }
   else
   {
        this.initWithString(myString);
   }
}
```

Conversion Tips

When converting your Objective-C classes to Java, keep these tips in mind:

When converting custom classes from an Enterprise Object Model, you might want to begin by opening the model in EOModeler and choosing the “Generate Java Files…” option from the “Property” menu. This will create a skeleton class in Java containing the constructor method and accessor methods for all your public variables (those marked with class properties in EOModeler). You can then begin to add in any customized behavior you may have added to the class, converting the code from Objective-C to Java as you go.   Remember that this technique only creates the code for one class at a time, so this mechanism fits nicely if you choose the “divide and conquer” strategy for conversion.

In Objective-C, it's possible for an instance method to have the same name as a class method, while in Java, class (static) and instance methods share the same name space. Therefore, if an Objective-C class has methods -foo and +foo , you must change at least one of them to a different name in Java.

Java doesn't use pointers, while Objective-C (like C and C++) does. Therefore, you will need to change your code if the Objective-C method you are converting uses pointers. There are Java equivalents for all object pointers, but pointers such as int* or id* or void* will not work. The Objective-C method must be changed to remove these references.

Objective-C allows you to message null objects, but bridged Java does not. You may need to add extra checks to your code to ensure that an exception does not occur. If you add these checks as you port the code, you may find that it makes debugging easier later on.

Don't try to catch an exception if you don't intend to at least log the exception. You will find that it is very hard to track what is going wrong. You might also find that the Thread.dumpStackTrace() call is useful when debugging your application or framework.

Static initializers require you to catch exceptions in them. It is possible that an exception (RuntimeException) can be raised in a static initializer and these, too, are hard to track. When in doubt, catch (Exception e) in any static initializer that could raise even just a RuntimeException.

Starting Java with the verbose option for loading classes will help you track down any problems that you might have in your static initialization routines.

In Java, you cannot do anything before calling super() in a public initializer. You may have to change your initialization scheme if your Objective-C class required action before messaging the super class.

If you pass a struct as an argument to a Objective-C method, you will need to decide how to convert these under Java. You can either split out the fields or you can convert the struct to a class. If you convert the struct to a class, you may need to write some native methods that convert between the class on the Java side and a struct on the Objective-C side automatically.

All integer values in Java are signed. Try to avoid unsigned in Objective-C if possible, because you will have to promote them to the next larger size in Java (especially unsigned long which would have to be promoted to Java long which is 64 bits - inefficient on most processors).

The JavaBrowser Tool can be very useful when mapping Objective-C to Java methods during conversion, or to see how Apple has converted Objective-C methods to Java.

If any of your hybrid Java objects implements finalize() (the method called by the garbage collector before deallocating your object), then you also have to call super.finalize() to avoid potential memory leaks. Also, you should not do anything in a finalize method that could result in a method invocation across the bridge.

When instantiating a wrapped or hybrid object, simply checking for null upon return from the constructor methods is not enough. You should always use a try/catch statement to detect an instantiation error by throwing an exception.

Objective-C categories that were originally created on classes that you do not own (such as Foundation) can be ported to Java by creating a "helper" class that is a subclass of the original class. You would then call the helper methods from there. If you did own the original class, any categories can just be pulled back into the original class.

If you should happen to have a poser in an application or framework that you are converting from Objective-C to Java, there is no Java equivalent of a poser. However, WebObjects 4.5 has added API to let you insert your own subclasses for many crucial classes of WebObjects and EOF, hopefully removing the need of posing as.

To keep the performance of your converted application or framework high, avoid the + operator on String as this is the biggest performance hit. Avoid chaining calls as much as you would in Objective-C. Avoid going back and forth through the Java bridge as well. If it makes sense, stay in the Java side for whole methods at a time. Use NSArray instead of Vector and use NSDictionary instead of Hashtable as much as possible since they are faster and fit better in the bridged APIs.

Read the online documentation "Debugging a WebObjects Application" for tips on debugging Java and mixed-language applications.

Read the online documentation "Mixing Languages in an Application" for additional information about creating a project with both Java and Objective-C classes.

Read the online documentation "Deployment and Performance Issues" for information about tuning your application for the best possible performance.

Conclusion

With a well-organized plan of attack, conversion of an application or framework from Objective-C to Java will not be a trip to the torture chamber. The Java Bridge allows you to convert your application or framework at the speed that makes sense for your situation, testing and debugging every step of the way.

Resources…

http://developer.apple.com/techpubs/webobjects
WebObjects Developer's Guide
Enterprise Objects Framework Developer's Guide
Enterprise Objects Framework Tools and Techniques (online with developer release)
http://www.omnigroup.com/MailArchive/WebObjects
http://www.omnigroup.com/MailArchive/eof
http://www2.stepwise.com/cgi-bin/WebObjects/Stepwise/Sites
ftp://dev.apple.com/devworld/Interactive_Media_Resources
http://www.apple.com/developer
http://developer.apple.com/media
http://til.info.apple.com/

Acknowlegments

Special thanks to Francois Jouaux and Mai Nguyen of Apple Computer, Inc. for their technical expertise and assistance in preparing this guide.

About the Author…

Theresa Ray is a Senior Software Consultant for Tensor Information Systems in Fort Worth, TX (http://www.tensor.com). She has programmed in OPENSTEP (both WebObjects and AppKit interfaces) on projects for a wide variety of clients including the U.S. Navy, the United States Postal Service, America Online, and Lockheed-Martin. Her experience spans all versions of WebObjects from 1.0 to 4.0, EOF 1.1 to 3.0, NEXTSTEP 3.1 to OPENSTEP 4.2, Rhapsody for Power Macintosh, and yellow-box for NT. In addition, she is an Apple-certified instructor for WebObjects courses.

Tensor Information Systems is an Apple partner providing systems integration and enterprise solutions to its customers. Tensor's employees are experienced in all Apple technologies including OPENSTEP, NEXTSTEP, Rhapsody, EOF and WebObjects. Tensor also provides Apple-certified training in WebObjects, Oracle consulting and training, as well as systems integration consulting on HP-UX. And Oracle.

You may reach Theresa by e-mail: *theresa@tensor.com* or by phone at (817) 335-7770.

http://www.apple.com/developer Think different.

Interactive Media Resources Include:

Interactive Media Guidebooks

Market Research Reports

Survival Guides—Technical "How To" Guides

Comarketing Opportunities

Special Discounts

Apple Developer Connection

As Apple technologies such as QuickTime, ColorSync, and AppleScript continue to expand Macintosh as the tool of choice for content creators and interactive media authors, the Apple Developer Connection continues its commitment to provide creative professionals with the latest technical and marketing information and tools.

Interactive Media Resources

Whether looking for technical guides from industry experts or for market and industry research reports to help make critical business decisions, you'll find them on the Interactive Media Resources page.

Apple Developer Connection Programs and Products

ADC programs and products offer easy access to technical and business resources for anyone interested in developing for Apple platforms worldwide. Apple offers three levels of program participation serving developer needs .

Membership Programs

Online Program—Developers gain access to the latest technical documentation for Apple technologies as well as business resources and information through the Apple Developer Connection web site.

Select Program—Offers developers the convenience of technical and business information and resources on monthly CDs, provides access to prerelease software, and bundles two technical support incidents.

Premier Program—Meets the needs of developers who desire the most complete suite of products and services from Apple, including eight technical support incidents and discounts on Apple hardware.

Make the Connection.

Join ADC today!

http://developer.apple.com/programs

Standalone Products

Apple offers many standalone products that allow developers to choose their own level of support from Apple or enhance their Select or Premier Program membership. Choose from the following products and begin enjoying the benefits today.

Developer Connection Mailing—Subscribe to the Apple Developer Connection Mailing for the latest in development tools, system software, and more.

Technical Support—Purchase technical support and work directly with Apple's Worldwide Developer Technical Support engineers.

Apple Developer Connection News—Stay connected to Apple and developer-specific news by subscribing to our free weekly e-mail newsletter, *Apple Developer Connection News*. Each newsletter contains up-to-date information on topics such as Mac OS, Interactive Media, Hardware, Apple News and Comarketing Opportunities.

Macintosh Products Guide

The most complete guide for Macintosh products! Be sure to list your hardware and software products in our *free* online database!

http://developer.apple.com/media

The ultimate source for creative professionals.