

Understanding the differences and similarities between PowerBuilder and Java

Converting PowerBuilder to Java

WRITTEN BY ROBERT BREIDECKER

As more PowerBuilder applications are being converted to Java, I wrote this article to provide insight to those who might be involved in such a project. My experience comes from one particular project where the goal was to convert an order entry system written in PowerBuilder 6.5 to a Java 2 (1.2) application using Swing components.

This article is for an experienced PowerBuilder developer with an intermediate knowledge of the Java language and object-oriented programming concepts. It provides insights only into the issues that may be involved in converting a PowerBuilder application to Java. Java is a different technology and therefore has many different issues and concerns that need to be dealt with. This article shouldn't be used as a replace-

ment for proper object-oriented design and system analysis. Also, any Java project should include one or more experienced Java developers who have object-oriented design and implementation experience.

Parts of the conversion were straightforward since PowerBuilder supports many object-oriented capabilities such as inheritance, polymorphism, and encapsulation. These features translate nicely into Java. Other parts weren't so easy, as PowerBuilder is a fourth-generation language with many high-level functions built into it. These features aren't so easy to implement in Java, which is considered by most to be a third-generation language. When I use the term *Java*, I'm referring to the core Java language along with the standard Java application programming interfaces (APIs). Java, along with the standard APIs, is also referred to as the standard Java edition. Other editions of Java include other APIs, but they're not discussed here.

When converting PowerBuilder to Java, it's beneficial to have a good understanding of the differences and similarities between the two technologies. In the next section I'll briefly describe the project I was

involved in. In the following two sections we'll take a look at the mappings of datatypes and controls from PowerBuilder to Java. We'll then look at conversion topics and some examples of how PowerBuilder translates into Java. At the end we'll cover ideas on how a PowerBuilder developer should prepare for a Java project.

Project Description

The order entry system is a tab-based, single-document interface that utilizes non-SQL DataWindows, command buttons, and response windows. The DataWindows contain single-line edits, DropDownDataWindows, lists, check boxes, radio buttons, and group boxes. The application retrieves and saves its data through a C++ Windows DLL library of functions that communicates with a UNIX server running an Oracle database.

My client had two main goals in converting the PowerBuilder application to Java. The first was to allow for easy access to the application over the Internet. To achieve this, we implemented the application as an applet that required users to install the Java plug-in. The second was to utilize standard Web technologies. My client perceived Java as a standard Web technology they could load onto their users' systems. The new Java application retrieves and saves its data through IBM's WebSphere application server. This application server won't be discussed here since any other compatible server could have been used instead.

The Java integrated development environment (IDE) used for this project was IBM's VisualAge for Java 1.2 version 3.0. Since Java is an open language supported by a large number of software vendors, the IDE isn't critical. I don't focus on the IDE since any one could be used for a conversion project. If you're interested in IDEs, there are a number to choose from. Here's a list of a few of the most popular:

- Borland JBuilder
- IBM VisualAge
- Oracle JDeveloper
- Sun Forté
- Sybase PowerJ
- WebGain VisualCafé (formerly Symantec VisualCafé)

Datatype Mappings

The conversion of the basic datatypes from PowerBuilder to Java is fairly easy (see Table 1). *Note:* The integer datatype in PowerBuilder converts to the short datatype in Java, and the long datatype in PowerBuilder converts to the integer datatype in Java. To keep things simple in our project we converted both the PowerBuilder integer and long datatypes to the integer datatype in Java. The conversion of real and decimal datatypes can be tricky if precision is an issue. Research and test the datatypes in both PowerBuilder and Java to be sure you're getting the results you want.

Control Mappings

The conversion of graphical controls from PowerBuilder to Java is not quite as easy. Picking a control in Java to replace the PowerBuilder control can usually be done without too much effort. However, for many controls you may have to be creative. Sometimes there can be more than one choice. The tricky part usually has to do with the conversion of the events associated with the control. In our project we soon learned not to get hung up on making controls work exactly the same between PowerBuilder and Java, but instead making the Java controls achieve the same business purpose as their PowerBuilder counterparts.

Table 2 provides suggested mappings of PowerBuilder controls to Java; don't be afraid to stray from this list. Many possible mappings are available. You may even disagree with some of them. Also, I selected Swing components over AWT components whenever possible.

Conversion Topics

This section covers various topics and issues you may encounter when converting PowerBuilder to Java. *Note:* These examples reflect the

DATATYPE	POWERBUILDER	JAVA
Boolean	boolean	boolean
Character	char or character	char
Date	date	Java.util.Date
Date Time	datetime	Java.util.Date
Decimal	dec or decimal	float/double (if precision is a concern, be sure to thoroughly research and test these data types)
Integer	int or integer	short
Long	long	int
Real	real	float/double (if precision is a concern, be sure to thoroughly research and test these datatypes)
String	string	java.lang.String or java.lang.StringBuffer
Time	time	java.util.Date

TABLE 1 Mappings of basic datatypes from PowerBuilder to Java

way we did things on our project. As with any programming assignment, there are always many ways to accomplish a task.

JAVA VERSIONS

One of the first things you need to decide for your project is what version of Java to use as there are many different versions available. The most current version of Java is Java 2 version 1.3. The term *Java 2* may be confusing to you. Between the 1.1 and 1.2 versions of Java, Sun changed the name of Java to Java 2.

You may need to use other Java software products, such as the Java plug-in, and make sure any other Java software you use is compatible with your version.

LOOK AND FEEL

PowerBuilder developers usually write applications that run on the Windows operating system. However, Java applications are often meant to run on many different systems including Windows, UNIX, Linux, and Macintosh. Because of this, the creators of Java created a concept called *Pluggable Look and Feel*. Programmers have the ability to change the way their applications look and feel (behave) at runtime. The default look and feel for Java is the Java (Metal), while others exist that look like Windows, Motif, and Macintosh. The Java or Metal look and feel is similar but distinctly different from Windows'.

PowerBuilder developers tend to want to make their Java applications look like Windows applications. Two ways to achieve this are (1) use the Windows look and feel and (2) modify the User Interface Manager (javax.swing.UIManager) object. The problem with the first solution is the Windows look and feel is only available for Windows platforms because of trademark reasons. The problem with the second solution is the developer must make manual changes to the look and feel and will be left with a mixed-up look and feel somewhere in between Java and Windows. Also, the Java look and feel is recommended by Sun for cross-platform development. Sun claims the Java look and feel will be the most stable across the various operating systems that Java runs on.

For our project we decided to go with the Java look and feel. We planned for our application to run on more than one operating system and didn't want to be bothered with the details of manually programming the look and feel for each individual component. If you decide to use the Java one for your application, you may want to let your users know of your plans up front so they don't expect the Windows look and feel when you deliver the finished product.

CONTROL TYPE	POWERBUILDER	JAVA
Check Box	CheckBox control or DataWindow check box object	javax.swing.JCheckBox
Command Button	CommandButton control or DataWindow command button object	javax.swing.JButton
DropDown List	DropDownListBox control, DataWindow drop down data DataWindow object or DataWindow drop down list box object	java.swing.JComboBox
Edit Mask	EditMask control or DataWindow edit mask object	No native support
Form Style Data	DataWindow control	Various Swing components
Grid Style Data	DataWindow control	javax.swing.JTable
Group Box	GroupBox control or DataWindow group box control	javax.swing.JPanel used with javax.swing.Border
Horizontal Scroll Bar	HScrollBar control	java.awt.Scrollbar
Line	Line control or DataWindow line object	java.awt.Graphics (using the drawLine method)
List Box	ListBox control	javax.swing.JList
List Style Data	DataWindow control	javax.swing.JTable
Multiple Line Edit	MultiLineEdit control or DataWindow column object	javax.swing.JTextArea
Oval	Oval control or DataWindow oval object	java.awt.Graphics (using the drawOval method)
Picture	Picture control or DataWindow picture control	java.awt.Image or javax.swing.ImageIcon
Picture Button	PictureButton control	javax.swing.JButton
Picture List Box	PictureListBox control	javax.swing.JTable
Radio Button	RadioButton control or DataWindow radio button object	javax.swing.JRadioButton
Rectangle	Rectangle control or DataWindow rectangle object	java.awt.Graphics (using the drawRect method)
Round Rectangle	RoundRectangle control or DataWindow round rectangle object	java.awt.Graphics (using the drawRoundRect method)
Single Line Edit	SingleLineEdit control or DataWindow column edit object	javax.swing.JTextField
Static Text	StaticText control or DataWindow text object	javax.swing.JLabel
Tab	Tab control	javax.swing.JTabbedPane
TreeView	TreeView control	javax.swing.JTree
Vertical Scroll Bar	VScrollBar control	java.awt.Scrollbar

TABLE 2 Possible mappings of graphical controls from PowerBuilder to Java

LEARN ABOUT SWING COMPONENTS

Swing components are significantly different from PowerBuilder components. One of the most significant differences is that most of them are built using the model-view-controller architecture (MVC), which allows the data and presentation layers to be separated during development and for more flexible design. Learning how Swing and MVC work together will greatly enhance your Java programming abilities.

Other differences between Swing and PowerBuilder components are subtle and may lead to confusion. One example is the `setEditable` method used by `JTextField` in Java. PowerBuilder doesn't use that method; it uses `setEnabled` for disabling a component. `JTextField` also

uses a `setEnabled` method in addition to `setEditable`. Calling `setEnabled(false)` on a `JTextField` will prevent the user from entering new text in the component. However, `setEditable(false)` will need to be called in order to make the background of the `JTextField` gray.

LAYOUT MANAGERS

Layout managers are an important part of the Java graphical user interface (GUI) development environment. They're complex and all Java GUI developers should be familiar with them. I won't go into the details of layout managers. The main purpose of addressing this topic here is to encourage you to use a layout manager.

If the layout manager of a Java container is set to null, then the container will honor all *x* and *y* coordinates and widths and heights for various components set by the programmer. If you don't use a layout manager, the application may not display properly on a different operating system or even run properly. Even though layout managers take a while to learn, they're well worth the effort. The only time a null layout manager is acceptable is when you're creating a prototype and have very limited development time.

Not only do layout managers help applications adapt to different operating systems, they also relieve the programmer from having to perform the tedious task of setting *x* and *y* coordinates along with the widths and heights for graphical components.

For most good IDEs, the grid bag layout is the best choice for layout manager. It's one of the most complicated of the layout managers, but also one of the most flexible and powerful. Most good IDEs generate the code needed for using grid bag layout.

POWERBUILDER UNITS TO PIXELS

If you're converting visual components from PowerBuilder to Java, you may want to keep some of the same look and feel. This may include making your new Java components a size similar to what they were in PowerBuilder.

One way of doing this is by converting the PowerBuilder units (PowerBuilder component measurements) to pixels (Java measurements). PowerBuilder provides two nice functions for doing this:

- `XUnitsToPixels`
- `YUnitsToPixels`

We actually used a small PowerBuilder calculator application that utilizes these two functions to convert PowerBuilder units to pixels.

REFERENCE PARAMETERS

Java doesn't support reference parameters. Parameters are passed by value in Java. To simulate passing a parameter by reference, you must pass an object as an argument and then update data inside the object.

Listing 1 provides a class definition in Java for the `TeamInfo` object.

The following code calls the `lookupTeamInfo` method, gets data out of the `teamInfo` variable, then prints it.

```
String memberName = "John Smith";
TeamInfo teamInfo = new TeamInfo();
LookupTeamInfo (memberName, teamInfo);
System.out.println(teamInfo.getTeamName());
System.out.println(teamInfo.getTeamLeaderName());
System.out.println(teamInfo.getNumberOfMembers());
```

Warning: Strings and the standard wrapper classes don't allow their values to be updated (this is called *immutable*). Therefore they can't be used for passing data in this way. Instead, make your own wrapper classes or use arrays of objects. For strings you can use the `StringBuffer` instead.

The wrapper classes wrap the primitive datatypes `int`, `long`, `char`, and more. For example, the `Integer` class wraps the `int` datatype. Once the value for this class is set, it can't be modified. Instead, create a custom class such as `ReferenceInteger` that you can use for passing integer values by reference.

FROM STRUCTURES TO OBJECTS

PowerBuilder has a structure datatype, Java doesn't. A common way of creating structures in Java is to create a class and add instance variables for each field in the structure. These fields should be declared private. To access them, getter and setter methods should be created. The TeamInfo class in the Reference Parameters example is a good example of a Java structure that has getter and setter methods.

PASSING DATA TO AND FROM WINDOWS

In PowerBuilder, you passed parameters to dialog (response) windows in the Open function, and retrieved data from dialog windows using the Message object. In Java, passing data to and from dialog windows is a little different; however, I think you'll like the change.

To create and open a dialog window in Java, first create the dialog and then make it visible. When a dialog becomes visible, it pops up on the screen for the user to see and interact with. You must first set the dialog to modal to force the user to interact only with the dialog and not the rest of the application.

To pass data to the dialog prior to making the dialog visible, call setter methods on the dialog object. The setter methods are added to the dialog class in the same manner as in the Reference Parameters section. When the dialog becomes visible, the windowOpened method is triggered (see java.awt.WindowListener for more information on window events). In the windowOpened method you can access the variables that were previously set in your code prior to making the dialog visible.

To pass data back from the dialog to the calling script, the dialog sets values to its variables and then makes itself invisible. When a dialog is no longer visible, the modal effect ends and processes transfers back to the calling script. The calling script can then call getter methods on the dialog to get the data out of it.

To discard the dialog, set its reference to null and garbage collection will clear the memory allocated for it. If you don't set the reference to null, when the script finishes processing, garbage collection will clear the memory for the dialog as long as there are no other references to it. Actually, you can never be sure of when garbage collection will clean up unused objects. Garbage collection is another subject entirely and is explained in most introductory Java books.

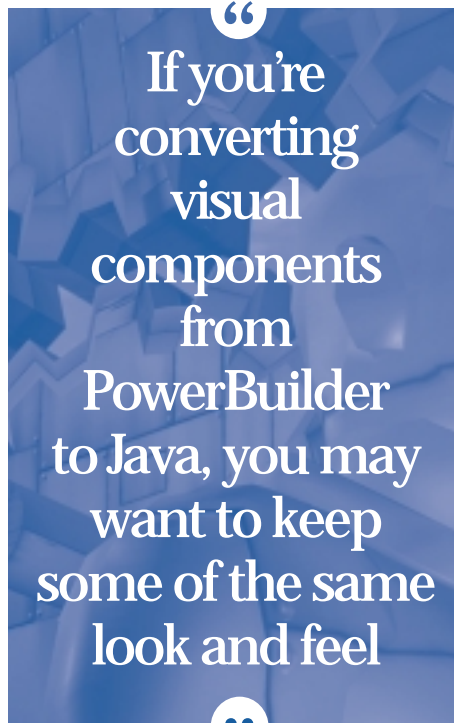
Here's an example of passing data to and from a dialog.

```
// Create the dialog and make it modal.
JDialog myDialog = new JDialog();
myDialog.setModal(true);
```

```
// Pass data to the dialog.
myDialog.setFirstName("James");
myDialog.setLastName("Gosling");
```

```
// Display the dialog to the user.
myDialog.setVisible(true);
```

```
// Get the return value from the dialog.
```



```
rtncode = myDialog.getReturnCode();
```

```
// Get rid of the dialog.
myDialog = null;
```

CONVERTING DATAWINDOWS

Of all the PowerBuilder objects, DataWindows can be the most complicated to convert to Java. They're the most used and functional of all the PowerBuilder components. However, they're probably the most unorthodox. It's well worth your time to analyze how your DataWindows have been used and what functionality they provide, then determine a plan for converting their functionality to Java. There are many different uses of DataWindows and too many to list here; I'll address only two of them.

- **Form DataWindows:** The most straightforward way is to create a JPanel and add Java controls to it that are similar to the PowerBuilder controls in the DataWindow. (See Table 2 for a list of control mappings between PowerBuilder and Java.) Remember, a number of DataWindow attributes and expressions can be set on controls inside DataWindows.

- **List DataWindows:** The JTable is probably the best choice for most list DataWindows. However, you may choose to use other controls such as JList to implement a single-column list DataWindow. You may even decide that a JTree can be used to replace some of them. The choice is up to you.

CONVERTING A DROPDOWNLISTBOX

The DataWindow's DropDownListBox in PowerBuilder supports both display and data values. The latter is used for associating the former to a unique code used for storage (usually in a relational database). For DropDownList Java

has the JComboBox. JComboBox doesn't support data values, only the display value. To overcome this deficiency, you'll need to extend JComboBox and add your own support for data values.

ADDING A DROPDOWNLISTBOX TO A JTABLE

You may be tempted to convert many of your DataWindows to JTables. JTables are easy to set up for basic functionality. However, be careful to set aside enough time to work on advanced JTables. Adding advanced features to JTables usually involves some extra research and experience. Here's an example of how to add a JComboBox to a JTable. You'll need this when you want to display a list of data and a DropDownList in one of the columns.

```
public class SampleTable extends
    javax.swing.JTable {
    // Constructor
    public SampleTable() {
        Class column = this.getColumnClass(2);
        // Get third column in table.
        JComboBox comboBox = new JComboBox();
        DefaultCellEditor cellEditor = new DefaultCellEditor(comboBox);
        this.setDefaultEditor(column, cellEditor);
    }
    .
    .
    .
}
```

CONVERTING A GROUP BOX

One possible solution for replacing the group box control in PowerBuilder is to use a JPanel and add a border to it. The controls that would normally be inside the group box control would then be added to the JPanel.

Here's an example Java statement that adds a border to a JPanel:

```
myPanel.setBorder(new
    javax.swing.border.TitledBorder(new
    javax.swing.border.EtchedBorder(), "My
    Panel");
```

CONVERTING EVENTS

Events are one of the most complicated issues when converting PowerBuilder to Java. There are many ways to map events from PowerBuilder to Java. There are also a large number of events to learn to understand what events are even possible in Java. Here are three examples of events we converted from PowerBuilder to Java.

- **ButtonClicked Event:** For this event we used the ActionListener.actionPerformed event.
- **Window Opened Event:** For this event we used the WindowListener.windowOpened event.
- **DataWindow ItemChanged Event:** For this event we used the FocusListener.focusLost event. We converted most of the form style DataWindows to JPanels with JTextFields on them. JTextField is a descendant of java.awt.Component, which is capable of trapping the FocusGained and FocusLost events. The

former is when the tab focus goes to the component; the latter, when focus is moved from the current component to another one.

COMPARING STRING VALUES

A common pitfall for an inexperienced Java programmer is to incorrectly compare String variables using the “==” comparison operator instead of the “equals” method. Strings are objects and when “==” is used to compare two objects, the expression checks if the objects are the same. The expression doesn’t check that the contents of both objects are the same. All classes in Java inherit from the Object class. The Object class has a method, “equals,” which can be used for evaluating the contents of the object. The String object overrides this method for that exact purpose.

```
// Probably incorrect – may work by luck because Java Strings are shared in a resource pool.
```

```
if (employeeName == managerName) {  
    .  
    .  
}
```

```
// Correct
```

```
if (employeeName.equals(managerName) {  
    .  
    .  
}
```

Note: You can use the “==” comparison operator under special circumstances to compare strings for equality. However, this is an advanced technique that should be used only by experienced Java programmers.

DATE AND TIME FUNCTIONS

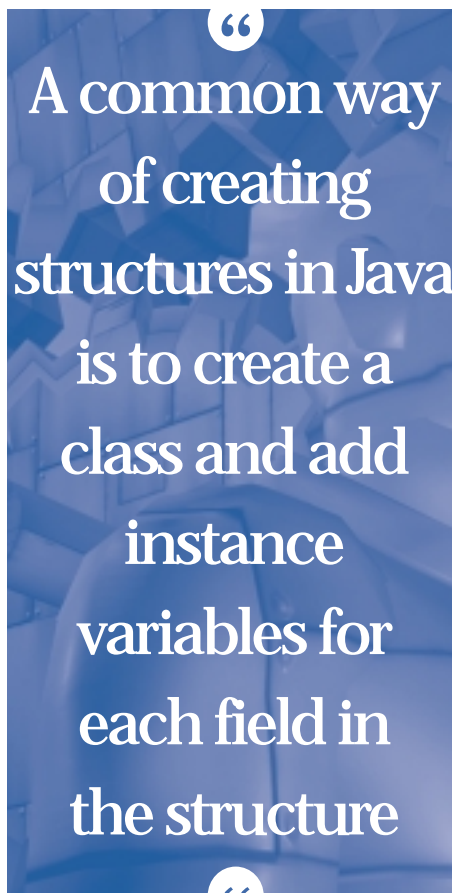
PowerBuilder offers some nice date and time functions as part of the core PowerScript language and the PowerBuilder Foundation Classes (PFC). In addition, PowerBuilder has both the date and time datatypes.

Instead, Java has the Calendar and Date classes in the java.util.package. These classes are full featured and can deliver the same functionality achieved in PowerBuilder. You’ll want to study both of these classes to determine how they function. For our project we built functions that imitated the PowerBuilder functions on top of these classes.

DO NOT IMITATE TOO MUCH

In the previous section about Date and Time functions, I mentioned that we imitated some of the PowerBuilder date and time function in Java. While this can be beneficial for some of the more complex functions you’re used to using, it can be a bad thing to do when trying to learn Java. We made a conscious effort not to imitate too many of the PowerBuilder functions because we wanted our PowerBuilder developers to learn the Java way.

An example of a PowerBuilder function you



may be tempted to imitate in Java is MessageBox. We decided not to imitate it because we wanted everyone to learn how to use JOptionPane in Java.

CONVERTING THE MESSAGEBOX

Since we’re on the topic of MessageBoxes, the conversion of a MessageBox from PowerBuilder to Java is simple. Here’s an example.

```
PowerBuilder:  
MessageBox("Title", "Message Text", INFORMATION)
```

```
Java:  
OptionPane.showMessageDialog(null, "Message Text", "Title", JOptionPane.INFORMATION_MESSAGE)
```

USING INTERFACES FOR JAVA CONSTANTS

There are two ways to implement constants in Java. The first is the most used and involves creating final static member variables in a class. Here’s a sample class with two final static member variables.

```
public class Constants {  
    public final static int FAILURE = -1;  
    public final static int SUCCESS = 1;  
}
```

Here’s an example Java statement referencing one of the constants.

```
public class MyClass {
```

```
.  
. .  
if (myMethod() = Constants.FAILURE) {  
    return;  
}  
. . .
```

The second is not as well known but has a nice feature. Instead of using a class with final static member variables, use an interface. By definition, Java makes all interface variables final and static. If your class extends a class or an interface, when you reference a member variable you don’t have to specify the name of the class or interface before the name of the variable. Also, if there are ambiguities between variable names, the Java compiler will force you to eliminate them. You don’t want to extend another class (as in the first example) simply for this feature. However, classes can implement as many interfaces as you want. The one drawback of using an interface is that the variables are always final. Here’s the same example as above using an interface instead.

```
public interface Constants {  
    public final static int FAILURE = -1;  
    public final static int SUCCESS = 1;  
}
```

```
public class MyClass implements Constants {
```

```
.  
. .  
if (myMethod() = FAILURE) {  
    return;  
}  
. . .
```

IMPORTANT CUSTOM COMPONENTS

Some components that you’ll want to use in your Swing applications will take considerable time to develop. You can try to develop these components yourself or save time by purchasing them from third-party software vendors. The prices are usually affordable, and many of the vendors also sell the source code. Two of the components we acquired from third-party vendors include both a calendar and an edit mask object. Many components are also available for free.

Some Web sites to help locate third-party components are:

- www.componentsource.com
- www.flashline.com
- www.freewarejava.com
- www.gamelan.com
- www.jars.com

DOCUMENTING YOUR CODE

JavaDoc is Java’s self-documentation feature built into Java. First, program JavaDoc-compatible comments right into your code. Then,

when you're ready, generate HTML documentation from these comments using the Javadoc utility that comes with the Java SDK. Therefore, when you finish coding your application, much of your documentation will already be completed.

PRINTING

This topic probably caused us the most problems on our project. PowerBuilder does a great job at printing. In particular, the DataWindow is excellent for printing and creating reports. However, printing is probably one of the weakest areas in Java. Java also wasn't designed to be a reporting tool. Therefore, converting PowerBuilder reports to Java is a very difficult task.

To assist you in this area you'll want to find printing examples on Sun's Java Web site and other sites. You'll also want to consider purchasing third-party Java printing components and maybe even a Java reporting tool.

Because printing in Java is such an issue, look into your printing requirements early so you can plan what approach to take.

Preparing Yourself

This section covers various topics that can help you prepare for a PowerBuilder-to-Java project.

JAVA.SUN.COM

You should become very familiar with Sun's Java Web site. They have one of the best Web sites designed for developers that I've found. You'll find programming tips, tutorials, resources, bug lists, and news about Java and its upcoming releases. Their site also has links to other sites dedicated to Java.

TRAINING

If you have little or no Java experience, you should probably look into taking a class or two. A class will introduce you to the basics of Java, and the instructor will be able to help you with concepts you're having difficulty with.

JAVADOC

The standard Java HTML documentation was itself generated using Javadoc. All team members should be familiar with Javadoc and its contents. It's an excellent resource and can increase your productivity. Javadoc for each version of Java is available on Sun's Java Web site along with the Java Development Kit (JDK).

RESOURCES

To keep up with the new Internet technologies that are continually changing, you need to become an avid reader. There are many excellent books that are fit for beginning developers all the way up to Java experts.

Here are a few to get you started:

- Eckel, B. *Thinking in Java*
- Horstmann, C., and Cornell, G. *Core Java 2, Volume 1*
- Horstmann, C., and Cornell, G. *Core Java 2, Volume 2: Advanced Features*
- Flanagan, D. *Java in a Nutshell: A Desktop Quick Reference*
- Eckstein, R., Loy, M., and Wood, D. *Java Swing*
- Chan, P., and Lee, R. *The Java Developer's Almanac 2000*

SUN'S JAVA CERTIFICATION

Becoming Java certified not only helps build your résumé, but will also most likely improve your Java skills. Unless you have a tremendous amount of Java knowledge already, studying for the certification exam will strengthen your existing skills and teach you a few things you didn't already know.

Certification exams can be intimidating to some. Even if you don't plan on taking the test, reading a certification study guide can be a worthwhile exercise. The guides cover the basic Java language features you'll use on all Java projects.

Here are a few of the popular Java certification study guides:

- Roberts, R., Heller, P., and Ernest, M. *The Complete Java 2 Certification Study Guide*
- Mughal, K.A., and Rasmussen, R.W. *A Programmer's Guide to Java Certification*
- Brogden, B. *Java 2 Exam Prep*
- Brogden, B. *Java 2 Exam Cram*

Information about taking the official Sun Java certification test can be found on Sun's Java Web site.

Conclusion

Every project needs a good plan. Hopefully, we've covered some ideas that can help you create that plan. I believe the most important ideas covered in this paper are getting your PowerBuilder developers trained in Java, adding experienced Java developers to your project, and understanding the differences and similarities between PowerBuilder and Java. ▼

AUTHOR BIO

Robert Breidecker, a senior consultant for Ferguson Consulting in St. Louis, Missouri, has over 10 years of experience working in the IT industry. A Powersoft certified PowerBuilder associate, he's also a Sun certified Java programmer for the Java 2 Platform.

rbreidec@rollingsoft.com

Listing 1

```
public class TeamInfo {
    private String teamName;
    private String teamLeaderName;
    private int numberOfMembers;

    public String getTeamName() {
        return teamName;
    }

    public String getTeamLeaderName() {
        return teamLeaderName;
    }

    public int getNumberOfMembers() {
        return numberOfMembers;
    }

    public void setTeamName(String newTeamName) {
        teamName = newTeamName;
    }

    public void setTeamLeaderName(String newTeamLeaderName) {
        teamLeaderName = newTeamLeaderName;
    }

    public void setNumberOfMembers (String newNumberOfMembers) {
        numberOfMembers = newNumberOfMembers;
    }
}

public void lookupTeamInfo(String memberName, TeamInfo teamInfo) {
    if (memberName.equals("Mary Smith")) {
        teamInfo.setTeamName("Finance");
        teamInfo.setTeamLeaderName("Mary Jones");
        teamInfo.setNumberOfMembers(12);
    }
    else {
        teamInfo.setTeamName("Unknown");
        teamInfo.setTeamLeaderName("Unknown");
        teamInfo.setNumberOfMembers(0);
    }
}
```

