Richard G Baldwin (512) 223-4758, baldwin@austin.cc.tx.us,
http://www2.austin.cc.tx.us/baldwin/

JavaBeans, A Skeleton Bean in the BeanBox

Java Programming, Lecture Notes # 504, Revised 02/18/98.

Preface

The information in this lesson is very important. However, it is information that does not lend itself well to teaching in a classroom environment, and it would be very difficult to test the student's knowledge of this material. Therefore, students in Prof. Baldwin's **Advanced Java Programming** classes at ACC **not** are responsible for knowing and understanding the material in this lesson.

JDK 1.1 was released on February 18, 1997 and JDK 1.1.1 was released on March 27, 1997. This lesson was originally written on April 7, 1997 using the software and documentation in the JDK 1.1.1 download package along with the February 97 release of the BDK 1.0 download package.

Introduction

Previous lessons provided an overview of Java Beans and developed a skeleton Bean program. A previous lesson also provided a test program to test the skeleton Bean. As it turned out, the test program was longer and more complex than the Bean itself.

This lesson will take the next step by showing you how to place that skeleton Bean in the JavaSoft BeanBox and how to use the BeanBox to test it. In this lesson, you will see that once you get over a few initial hurdles, testing in the BeanBox can be simpler than developing test programs.

However, you will also learn that those initial hurdles can be serious. The BeanBox program is very package and path sensitive, so you must be very careful how you build your JAR files for inclusion in the ToolBox for the BeanBox.

The BeanBox

The Beans Development Kit (BDK) is a Java application that requires JDK 1.1 or later to be installed. It provides support for the JavaBeans APIs along with a test container referred to as the BeanBox. This test container can be used to test Bean behavior.

The BDK also provides the JavaBeans Specification and a Tutorial along with a number of sample Beans.

The BeanBox is a sample container designed for testing Beans. As of April 1997, the BDK 1.0 BeanBox would handle visible Beans, but purely computational beans could not be tested in the BeanBox.

When you start the BeanBox, a *ToolBox* of sample Beans is displayed. Source code for these Beans is provided in the *demo* subdirectory of the Win95 BDK download package.

You can start the BeanBox application on a Win95 system by locating the beanbox directory and making it the current directory. There you will find two files named **run.bat** and **run.sh** respectively. You can invoke the **run.bat** file to start the BeanBox application on a Win95 system. Although I don't have a Solaris system available, it is my understanding that you can use the **run.sh** file to start the BeanBox application on a Solaris system.

You should probably start the BeanBox application and familiarize yourself with it before attempting to use it to test your Beans.

Once the BeanBox application is running, three forms will appear on your screen with the titles of *ToolBox*, *BeanBox*, and *PropertySheet*. If you are familiar with development tools like *Visual Basic* and *Delphi*, you may recall the existence of three similar forms with similar names in those development environments.

When the BeanBox application starts, you will see about seventeen sample Beans already in the ToolBox. Our objective in this lesson will be to place our skeleton Bean in the ToolBox so that we can test it, but we haven't gotten there yet.

To place a ToolBox-Bean in the BeanBox, click on the Bean in the ToolBox and then click on the area in the BeanBox where you want the Bean to appear.

As a quick and interesting example of how to work with the BeanBox, we will place an animated Bean in the BeanBox and cause the animation to start and stop on command.

First, get a *Juggler* Bean from the ToolBox and place it in the BeanBox. Then get two *OurButton* Beans and place them in the BeanBox also. As you do this, you will notice that the Bean most recently clicked is highlighted with a striped outline. The Bean highlighted with the striped outline is the Bean to which the menus and the PropertySheet apply.

Highlight first one and then the other of the buttons. When each button is highlighted, look at the PropertySheet and notice the text entry box on the PropertySheet that contains the word "press" and is labeled "label". This is the property editor for the button-property named **label**.

Modify the **label** property for each of the buttons, causing one of them to display *start* and the other to display *stop*.

Then highlight the button with the *stop* label and pull down the **Edit** menu. Select *Events/action/actionPerformed*. The next time you move the mouse, you will notice that a red line extends from the mouse pointer to the *stop* button. Place the mouse pointer on the animated Juggler Bean and click the mouse. This will open a dialog box that lists all of the methods of the Juggler Bean. Select the stopJuggling method.

You have now hooked an Action event on the *stop* button to the *stopJuggling* method of the animated Bean. If you click the *stop* button, the animation should stop.

Use the same procedure to hook the *start* button to the *startJuggling* method. Now you should be able to make Duke start and stop juggling at will by selecting first one and then the other of the two buttons.

Spend a little time familiarizing yourself with the BeanBox before moving on.

Installing a Bean in the ToolBox

Once you have written and compiled your Bean, there are basically two steps involved in installing it in the ToolBox for the BeanBox:

- Encapsulate it in a *jar* file.
- Store the *JAR* file in the *jars* directory of the BDK.

This sounds easy enough, but it can actually be very difficult to get it right. The entire process is extremely sensitive to the specification of the Java package and the relative locations of various components when the *jar* file is created.

Rather than attempt to explain all of the issues, I am simply going to provide you with an example that works on my system. Hopefully you can use or modify that example to make it work on your system.

If you read the tutorial from the BDK 1.0, you will find that the author of that tutorial places Beans in the ToolBox using *make* files and the *nmake* program from Microsoft.

I continue to be amazed that a tutorial from JavaSoft would require access to a Microsoft program that, quite frankly, is not easy to come by. Apparently people who do other software development work using Microsoft development packages such as Visual C++ have copies.

At the time, I was not able to locate a downloadable copy of *nmake* anywhere on the web. I finally located a copy on a CD ROM in the back of one of my books but there were no usage instructions. It took me several more days of searching to learn that a -f switch is required to correctly process a new make-file. All in all, not a fun experience.

(I later found something that indicates that the *nmake* program can be downloaded from <ftp://ftp.microsoft.com/Softlib/MSLFILES/nmake15.exe> but I haven't tried it yet.)

All that aside, if you are familiar with the use of *make* files and have a copy of *nmake* available, go ahead and use the procedure in the Beans tutorial. Be aware, however, that there are a large number of typographical errors in the PDF version that comes with the Win95 version of the BDK. (Hopefully these typographical error will be corrected in a future release of the BDK.) If you are going to use a *make* file, start with one of the existing *make* files from one of the sample Beans. Don't try using the material printed in the tutorial.

The approach that I am going to show you in this lesson doesn't use a *make* file. Rather, it uses a batch file to accomplish enough of the same operations to be usable.

Jar Files in Brief

What is a JAR file? Briefly, a JAR file is a file created by the *Java Archive Tool* which is a part of JDK 1.1. A JAR file contains one or more other files. It can contain all the files that make up a Bean, and can contain more than one Bean.

JAR files use the same format as ZIP files and have a *manifest file* that describes the contents of the JAR file. For some uses, the manifest file is optional, but it is not optional when a JAR file is used to contain Beans that are to be installed in the ToolBox..

JAR files are produced using the *jar tool*. To invoke the jar tool, enter **jar** at the command line followed by one or more parameters. (This assumes that your JDK 1.1 software is on the path.)

In this lesson, we will only concern ourselves with the parameters required to achieve our objective of adding a new Bean to the ToolBox. Hopefully, a future lesson will discuss JAR files in more detail.

Our skeleton Bean application is named **Beans01.java**. To install the new Bean in the ToolBox, we will need three files:

- The compiled Bean file named Beans01.class
- The batch file named Beans01.bat
- The manifest file named Beans01.mft

Obviously the contents of these files is important. Equally important is where they reside within the directory structure on your computer. First lets take care of the contents.

The file named Beans01.bat contains the following line:

```
jar cfm ..\jars\Beans01.jar Beans01.mft sunw\demo\beans01\*.class
```

Likewise, the file named Beans01.mft contains the following two lines:

```
Name: sunw/demo/beans01/Beans01.class
Java-Bean: True
```

The batch file executes the **jar** program with the *cfm* switches set.

Briefly, the c-switch says to create a new JAR file in a file specified by the f-switch. The f-switch says to create the new file as specified by the parameter immediately following the switches. In this case, create the following file:

- `..\jars\Beans01.jar`

If you are not familiar with path specifications of this sort, you would do well to find someone to explain it to you.

The m-switch says to use the file named Beans01.mft in the same directory as the batch file as the manifest file.

Finally, the last parameter

- `sunw\demo\beans01\*.class`

specifies the files to be included in the new JAR file in addition to the manifest file. In this case, we are including only class files which are contained in the specified path and directory. In fact, there is only one class file. However, there could be many class files and possibly other types of files as well for complex Beans.

Path and Package Specifications

Path and package specifications are critical here. If you get them wrong, you may not be able to install your Bean in the ToolBox. Sometimes it is possible to get it wrong and still make the installation, but not be able to hook events to methods.

The source and class files for the skeleton Bean were located in the directory named **beans01** (the full path to that directory will be shown later).

The following package specification was required in the skeleton Bean source code in order to install the compiled Bean in the JavaSoft **BeanBox**.

- `package sunw.demo.beans01;`

Here is the path from the root directory to the directory containing the class file for the skeleton Bean.

- `c:\java_jdk\Bdk\demo\sunw\demo\beans01\Beans01.class`

You should compare the paths information shown earlier with this path to see how it all fits together.

The entire directory structure at and below the BDK directory was established by the BDK installation process, except that I added a directory named **beans01** to contain the class file for the skeleton Bean.

As shown, the class file for the skeleton Bean was in the **beans01** directory.

The batch file and the manifest file shown earlier were installed in the **demo** directory closest to the root. If you look, you will find that is also where all the make files for the sample Beans are located.

This process creates a JAR file containing the class file for the skeleton Bean along with the manifest file. That JAR file is deposited in the **jars** directory which, like the top-level **demo** directory is a child of the **BDK** directory.

Testing the Bean in the BeanBox

The first sign of success is that the new skeleton Bean named **Beans01** will join the sample Beans in the ToolBox.

Testing the skeleton Bean in the ToolBox is fairly straightforward.

Testing the Methods

A good way to test the two methods is to do the same thing that we did to make the juggler stop and start juggling. Place a **Beans01** Bean in the BeanBox along with two buttons. Hook the **Action** events from the two buttons up to the **makeBlue()** and **makeRed()** methods of the Bean. You should then be able to use these two buttons to switch the color of the Bean back and forth between blue and red.

Testing the Events

Here you need to determine if the Bean exposes the ability to *add* and *remove* **Action** events. In this case, click on the Bean to highlight it. Then pull down the Edit menu and select *Events/action/actionPerformed*. (Don't actually select it, just make sure that it is there for you to select if you choose to.)

This Bean is created by extending **panel** which doesn't normally generate **Action** events. Therefore, the presence of the above selection on the Edit menu indicates that you were successful in adding this event type to the Bean and were successful in exposing the *add* and *remove* methods.

Testing the Properties

Here you are going to confirm that the two properties appear on the PropertySheet and also confirm that they can be edited.

Highlight your Bean and confirm that there is a *color* property and a *myBooleanProperty* on the PropertySheet. If so, click on them. You should be able to switch the boolean property back and forth between true and false using a pulldown list that is provided. You should be able to select among many different colors for the *color* property using a color dialog that is provided.

Conclusion

These simple tests indicate that we were successful in exposing our *properties*, *events*, and *methods*. They confirm that the two methods behave as planned and that the Property editor can both *set* and *get* properties.

Since we didn't really make our *add* and *remove* **Action** events operational, there is no further testing that we can do on them, but at least we have confirmed that they are properly exposed.

When we wrote the bean, we were careful to **synchronize** exposed methods and were careful to make the entire class **serializable**. Therefore, at this point, we can conclude with reasonable confidence that we have created a Bean, albeit simple, that adheres to the interface requirements and (except for the fact that the code for our events is incomplete) should be suitable for installation in and manipulation by a Visual Builder Tool.

-end-