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

JavaBeans, Properties of Beans, Simple and Indexed

Java Programming, Lecture Notes # 508, Revised 02/19/98.

Preface

Students in Prof. Baldwin's **Advanced Java Programming** classes at ACC are responsible for knowing and understanding all of the material in this lesson.

The material in this lesson is extremely important. However, there is simply too much material to be covered in detail during lecture periods. Therefore, students in Prof. Baldwin's **Advanced Java Programming** classes at ACC will be responsible for studying this material on their own, and bringing any questions regarding the material to class for discussion.

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 13, 1997 using the software and documentation in the JDK 1.1.1 download package along with the April 97 release of the BDK 1.0 download package.

You may be wondering why I continue to provide this information. The reason is simple. The Java JDK and BDK continue to evolve at a very rapid rate with bugs being fixed with each new release. For example, you may read a lesson and test the programs in that lesson using a later version of the JDK and BDK than was used when the lesson was originally written. In some cases, you may get different results. In that case, you need to be aware that things may have changed with the development software.

For example, earlier lessons which were based on the Beta 3 version of JDK 1.1 discussed a coordinate offset problem when extracting mouse coordinates from a mouse event. As of JDK

1.1.1, that problem no longer exists. I don't know whether it was fixed in JDK 1.1 Final or JDK 1.1.1, but it has been fixed.

Also, an earlier lesson based on the February 1997 release of BDK 1.0 had an undocumented and unexpected requirement for special code to properly size a bean when it is placed in the BeanBox. That requirement no longer exists in the April 1997 version and in that respect, the BeanBox seems to behave as would be expected. Apparently that was a bug that has also been fixed.

Introduction

This lesson summarizes information presented earlier on *Simple* properties and provides information not previously discussed on *Indexed* properties. An example **bean** class is developed that includes *Indexed* properties. This **bean** class is examined with the **Introspector** and is tested with a test program designed specifically to exercise the **get** and **set** methods for the *Indexed* property.

According to JavaSoft,

> "A Java Bean is a reusable software component that can be manipulated visually in a builder tool."

The purpose of a Bean is to be installed in the toolbox of a visual builder tool (**VBT)** so that it can be incorporated into new applications and applets with no requirement to recompile the code for the Bean. Furthermore, two or more Beans should be able to be installed in such an application and caused to communicate with one another without the requirement to recompile either of them.

The following five attributes are common to Beans:

- Introspection
- Properties
- Events
- Customization
- Persistence

Previous lessons have provided a general introduction to beans, have provided instructions on how to use the BeanBox, and have discussed Introspection in detail. This lesson will concentrate on certain aspects of Properties. Future lessons will continue the discussion of Properties and will discuss the other attributes as well.

Overview of Properties

In the simplest case, a property of a Bean is represented by an instance variable whose value can be manipulated using a pair of *set* and *get* methods. The **Introspector** uses *design patterns* to identify propeties.

If the *set* method exists without a corresponding *get* method, then the property is a *write-only* property. Similarly, if the *get* method exists whthout a corresponding *set* method, then the property is a *read-only* property.

It is also possible to forego *design patterns* and provide explicit information regarding various aspects of the interface including Properties.

The **Introspector** will accept an *is* method as an alternative to or in addition to a *get* method for the special case of **boolean** properties.

There are four kinds of properties:

- Simple
- Indexed
- Bound
- Constrained

This lesson will concentrate on *Simple* and *Indexed* properties. *Bound* and *Constrained* properties will be discussed in a future lesson. Explicit specification of properties will also be covered in a future lesson.

Design Patterns for Simple Properties

The **Introspector** uses *design patterns* to locate properties by looking for methods having signatures of the form

```
public void set<PropertyName>(<PropertyType> a)

public <PropertyType> get<PropertyName>()
```

The existence of a matching pair of such methods is regarded as defining a *read-write* property whose name will be <propertyName>. (Note the change in case of the first letter in the property name. The rules for upper and lower case were explained in the lesson on the **Introspector**.)

The two methods are used to *get* and *set* the property values as the names of the methods imply.

Both methods in the pair may be located either in the same class, or one may be in a subclass and the other may be in a superclass.

If only one of the methods from the pair exists, then it is regarded either as a *read-only* or a *write-only* property.

The default assumption is that the property is neither *bound* nor *constrained*. As mentioned earlier, this will be discussed in more detail in a subsequent lesson.

Reflecting the above general description in more concrete terms might result in the following pair of methods for a property named **myProperty** of type **int**.

```
public void setMyProperty(int a){//...};

public int getMyProperty(){//...};
```

Design Patterns for Boolean Properties

As a special case for **boolean** properties, the **Introspector** will recognize the following form either in place of or in addition to the *get* method.

```
public boolean is<PropertyName>()
```

In either case, if the " is<PropertyName>" method is present for a boolean property then it will be used to read the property value.

An example for a boolean property named **ready** might be:

```
public boolean isReady(){//...}

public void setReady(boolean m){//...}
```

It is important to remember that the instance variable used to maintain the value of the property is not required to have the same name as the property, but it may have the same name.

Design Patterns for Indexed Properties

An indexed property is a property having multiple values stored in an array. The following design patterns are regarded as indicating a property of this type.

```
public <PropertyElementType> get<PropertyName>(int a)

public void set<PropertyName>(int a, < PropertyElementType> b)
```

where the value passed to the integer parameter is the index of the element of interest.

The following example program defines a property according to this design pattern. Application of the **Introspector** using JDK 1.1.1 found but did not successfully report the accessor methods for this property. This may be a bug in the JDK 1.1.1 Introspector, or may be an error in my program. I'm not certrain as of this writing (April 14, 1997).

It is also possible to have accessor methods which read and/or write the entire array. This results in design patterns of the following form:

```
public <PropertyType>[] get<PropertyName>()

public void set<PropertyName>(<PropertyType> a[])
```

Taking all of this into account might lead to the following four methods in the design pattern for an indexed property of type **MyType** named **myProperty**.

```
public MyType getMyProperty(int a) //return an element
```

```
public void setMyProperty(int a, MyType b) //set an element

public MyType[] getMyProperty() //return an array

public void setMyProperty(MyType a[]) //set an array
```

Sample Bean Program

In this lesson, we will deal with two different programs. One program is a program that creates a bean. The other program is a program used to partially test the bean. This section deals with the program used to create the bean.

This program was designed to be compiled and executed under JDK 1.1.1. It was tested using JDK 1.1.1 and the Apr97 version of the BDK 1.0 under Win95. As mentioned earlier, this is important information. Testing the same program using a later version of the JDK or BDK may yield different results.

The purpose of this program is to illustrate *simple* and *indexed* properties. An illustration of *bound* and *constrained* properties is deferred until a future lesson.

This bean contains three properties: a *simple* **Color** property named **color**, a *simple* **boolean** property named **ready**, and an *indexed* **int** property named **indexedProperty**. (It also contains a read-only property named **preferredSize** which resulted from providing that information for the benefit of the layout manager.)

The behavior of the properties was first tested by installing the bean in the BeanBox and observing the behavior.

The bean contains a method named **makeBlue()** which sets the background color of the bean to blue by invoking the *set* method of the **color** property. This method was included for the sole purpose of illustrating the setting of a property by invoking its *set* method under program control.

A button was installed in the BeanBox along with this bean. By linking the **actionPerformed()** event of the button to the **makeBlue()** method of the bean, that button was used to invoke the **makeBlue()** method which in turn demonstrated the ability to set the color property under program control. When the button was clicked, the background color of the bean burned blue.

It was also observed that the **color** property and the **ready** property both appeared in the *property editor* of the BeanBox and behaved as would be expected, both for setting and getting the values of the properties. This was considered to be an adequate test of both of the *simple* properties.

The fact that the bean appeared in the BeanBox at the correct size was considered to be an adequate test of the simple read-only property named **preferredSize**.

The *indexed* property did not appear in the *property editor* of the BeanBox. I was unable to find anything in the documentation on the BeanBox regarding the use of the property editor with *indexed* properties, so I really am not certain what we should expect in this regard.

Primarily because we were unable to test the *indexed* property using the BeanBox, the bean was tested further using two other approaches.

First, a special test program named **Beans02Test** was written to test the *indexed* property of the bean. This program tested the ability to *set* and *get* the individual elements of the *indexed* property under program control. It also tested the ability to *set* and *get* the entire indexed property array under program control.

The results of those tests appeared to be successful and are contained in the comments in the program listing for that program. They will be discussed in a later section.

Next, an **Introspection** program named **Introspect01** was applied to the bean to report on its properties. The program named **Introspect01** is a program developed in an earlier lesson that applies introspection and reports on the *properties*, *events*, and *methods* of a bean. Application of the **Introspect01** program to this bean produced the following output (since this bean was developed to illustrate properties, only that portion of the report dealing with properties is shown).

```
Name of bean:  Beans02
Class of bean: class Beans02

==== Properties: ====
Name: indexedProperty
 Type:       class [I
 Get method: public synchronized int[] Beans02.getIndexedProperty()
 Set method: public synchronized void Beans02.setIndexedProperty(int[])
Name: preferredSize
 Type:       class java.awt.Dimension
 Get method: public synchronized java.awt.Dimension
Beans02.getPreferredSize()
 Set method: null
Name: color
 Type:       class java.awt.Color
 Get method: public synchronized java.awt.Color Beans02.getColor()
 Set method: public synchronized void Beans02.setColor(java.awt.Color)
Name: ready
 Type:       boolean
 Get method: public synchronized boolean Beans02.isReady()
 Set method: public synchronized void Beans02.setReady(boolean)
================================================================================
```

The report on all of the *simple* properties look correct.

The **Type** specification for the *indexed* property looks a little strange. Otherwise, the report looks correct insofar as the *set* and *get* methods for setting and getting the entire array of indexed property values is concerned.

However, it is bothersome that the **Introspector** did not also report the *set* and *get* methods for setting and getting a single indexed element for the *indexed* property.

To further investigate this, the methods that *set* and *get* the entire indexed property were removed from the program. Then the program was recompiled and tested again using the **Introspector**. When this was done, the **Introspector** reported the following regarding the indexed property:

```
Name: indexedProperty
 Type:       null
 Get method: null
 Set method: null
--------------------------------------------------------------------------
```

This is definitely not what we would hope to see. The **Introspector** appears to be able to recognize the existence of the *indexed* property, but does not properly report on the accessor methods for the individual elements. I don't know if this is a bug in the system or an error in my program. Again, for the record, these results were obtained using JDK 1.1.1. Performing the same tests with a later version may produce different results.

Note: This same test was performed again in October 1997 using JDK 1.1.3 and the results were the same. The **Introspector** still didn't seem to be able to properly report on the accessor methods for the individual elements.

With respect to getting different results from later versions of the JDK and BDK, note that the undocumented requirement for a special **sizeToFit()** method that existed in the Feb97 version of BDK 1.0 was eliminated in the Apr97 version. For more details on this, see the earlier lesson on the skeleton bean. It now appears that this was a bug in the Feb97 version that was eliminated in the Apr97 version.

For the record, the batch file used to install this bean in the beanbox contained the following:

```
jar cfm ..\jars\Beans02.jar Beans02.mft sunw\demo\beans02\*.class
```

The manifest file named **Beans02.mft** used in conjunction with the batch file to install the bean in the beanbox contained the following:

```
Name: sunw/demo/beans02/Beans02.class
Java-Bean: True
```

The following package specification was required to install the Bean in the BeanBox. Note that the package specification changes depending on where the Bean is installed in the directory structure.

```
package sunw.demo.beans02;
```

Sample Test Program

As mentioned above, the BeanBox could not be adequately used to test the *indexed* property of this bean. Therefore, a special test program was written to test just that aspect of the bean.

This program was designed to be compiled and executed under JDK 1.1.1. The program was tested using JDK 1.1.1 under Win95.

This is a minimal test program for **Beans02.java**. It is designed to test only the *indexed* property of **Beans02**. The other properties of **Beans02** can be tested adequately in the BeanBox.

The program executes a series of statements that invoke the *set* and *get* methods of the *indexed* property of the bean both for individual elements and for the entire indexed property array.

The test appeared to be completely successful. The code in this test program is almost trivial and is interesting only because it is being used to test an *indexed* property of a bean. Since it is so simple, it won't be discussed in detail at this point. However, the entire program listing as well as the output produced by the program is provided at the end of this lesson for your review.

Interesting Code Fragments from the Bean Program

The first interesting code fragment is the declaration of the three instance variables that are used to maintain the property values. This is followed immediately by the constructor that, among other things, instantiates an array object for maintenance of the values of the *indexed* property. Recall that in Java, you must declare a reference variable for an array and then instantiate an actual array object which is referenced by that reference variable.

```
protected Color myColor;
protected boolean readyPropertyValue = true;
protected int[] myIndexedInstanceVariable;

public Beans02(){//constructor
  myColor = Color.yellow;//initialize the background color to yellow
  setBackground(myColor);
  //Instantiate an array object to maintain the indexed property
  myIndexedInstanceVariable = new int[3];
}//end constructor
```

The next interesting code fragment is the pair of *set* and *is* methods which, along with the instance variable named **readyPropertyValue,** constitutes the *simple* property named **ready**. Recall that for the special case of **boolean** properties, either a *get* method or an *is* method will satisfy the *design pattern* requirement.

```
public synchronized boolean isReady(){
  return readyPropertyValue;
}//end isDummyInstanceVariable()

public synchronized void setReady(boolean data){
  readyPropertyValue = data;
}//end setReady
```

The next interesting code fragment is the pair of *set* and *get* methods which, along with the instance variable named **myColor,** constitute the *simple* property named **color**.

```
public synchronized void setColor(Color inColor){
  myColor = inColor;
  this.setBackground(myColor);
}//end setColor()
```

```
  public synchronized Color getColor(){
    return myColor;
  }//end getColor
```
The next interesting code fragment is the method named **makeBlue()** that was included in the bean solely to illustrate the ability to modify the **color** property under program control by invoking its *set* method. Properties are often modified at design time using property editors in builder tools, and are often modified at runtime by invoking *set* methods under program control.

```
  public synchronized void makeBlue(){
    this.setColor(Color.blue);
  };//end makeBlue()
```

That brings us to the interesting code fragments for the *indexed* properties. The following two methods are intended to satisfy the *design pattern* requirement for setting and getting individual elements in an *indexed* property. Although these methods seem to match the specifications, they don't seem to be properly recognized by the **Introspector** in JDK 1.1.1 as discussed in an earlier section.

They do respond properly to the test program which is listed at the end of the lesson. It seems as if the *design pattern* matching capability of the **Introspector** doesn't work correctly for the case where the *get* method signature contains an argument and the *set* method signature contains two arguments which is the requirement for getting and setting individual indexed values for an *indexed* property.

```
  public synchronized int getIndexedProperty(int a){
    return myIndexedInstanceVariable[a];
  }//end getIndexedProperty

  public synchronized void setIndexedProperty(int a, int b){
    myIndexedInstanceVariable[a] = b;
  }//end setIndexedProperty
```
The next interesting code fragment is the following pair of methods that satisfy the *design pattern* requirement for setting and getting the entire array that maintains the values of an *indexed* property. Although the *design pattern* for setting and getting individual elements in the indexed property don't seem to work properly, the *design pattern* for setting and getting all of the elements as a group does seem to work properly. See the discussion in an earlier section.

```
  public synchronized int[] getIndexedProperty(){
    return myIndexedInstanceVariable;
  }//end int[] getIndexedProperty()

  public synchronized void setIndexedProperty(int a[]){
    myIndexedInstanceVariable = a;
  }//
```
The next interesting code fragment is interesting only because it represents a bug fix in the Apr97 version of BDK 1.0. This is a relatively standard method for establishing the size of a

component at display time. The Feb97 version of the BDK did not properly use this method to set the size of the bean, but that problem has been fixed in the Apr97 version.

```
  public synchronized Dimension getPreferredSize(){
    return new Dimension(50,50);
  }//end getPreferredSize()
```

Interesting Code Fragments from the Test Program

As mentioned earlier, the test program is so simple that it really doesn't contain any interesting code fragments. The program simply invokes the *set* and *get* methods of the *indexed* property, both for individual elements and for the complete set of *indexed* property values. The results are entirely as would be expected. A listing of the program and the output produced by the program is included at the end of the lesson.

Program Listing for the Bean Program

This section contains a commented listing of the bean program. See the previous sections for an operational description of the program.

```
/*File Beans02.java.java Copyright 1997, R.G.Baldwin
This program was designed to be compiled and executed under JDK 1.1.1.
It was tested using JDK 1.1.1 and the Apr97 version of the BDK 1.0
under Win95.

The purpose of this program is to illustrate simple and indexed
properties.

The following package specification was required to install the Bean in
the BeanBox.  Note that the package specification changes depending on
where the Bean is installed in the directory structure.
*/
package sunw.demo.beans02;

import java.awt.event.*;
import java.awt.*;
import java.io.Serializable;
//============================================================================
public class Beans02 extends Canvas implements Serializable{

  //The following three instance variables are used for properties
  protected Color myColor;
  protected boolean readyPropertyValue = true;
  protected int[] myIndexedInstanceVariable;

  public Beans02(){//constructor
    myColor = Color.yellow;//initialize the background color to yellow
    setBackground(myColor);
    //Instantiate an array object to maintain the indexed property
    myIndexedInstanceVariable = new int[3];
  }//end constructor
```

```
//-----------------------------------------------------------------------
//The following "set" and "is" methods in conjunction with the instance
// variable readyPropertyValue constitute a boolean property named ready.
// For boolean properties, either a "get" method or an "is" method will
// support the design pattern requirement.
public synchronized boolean isReady(){
  return readyPropertyValue;
}//end isDummyInstanceVariable()

public synchronized void setReady(boolean data){
  readyPropertyValue = data;
}//end setReady

//-----------------------------------------------------------------------
//The following "set" and "get" methods in conjunction with the instance
// variable myColor constitute a property of type Color named color.
public synchronized void setColor(Color inColor){
  myColor = inColor;
  this.setBackground(myColor);
}//end setColor()

public synchronized Color getColor(){
  return myColor;
}//end getColor

//-----------------------------------------------------------------------
//The following method is included to demonstrate that the color
// property can be set under program control by invoking its set access
// method.
public synchronized void makeBlue(){
  this.setColor(Color.blue);
};//end makeBlue()
//-----------------------------------------------------------------------

//The following two methods are intended to satisfy the design pattern
// for setting and getting elements from an indexed property.  See the
// comments at the beginning of the program for a further discussion
// on this.

public synchronized int getIndexedProperty(int a){
  return myIndexedInstanceVariable[a];
}//end getIndexedProperty

public synchronized void setIndexedProperty(int a, int b){
  myIndexedInstanceVariable[a] = b;
}//end setIndexedProperty
//-----------------------------------------------------------------------

//The following two methods satisfy the design pattern for setting and
// getting the entire array for an indexed property.

public synchronized int[] getIndexedProperty(){
  return myIndexedInstanceVariable;
}//end int[] getIndexedProperty()

public synchronized void setIndexedProperty(int a[]){
```

```
    myIndexedInstanceVariable = a;
  }//

  //-----------------------------------------------------------------------
  //This method defines the display size of the object when it appears
  // in the BeanBox.  It is called automatically by the layout manager
  // or whatever is controlling the placement and size of components
  // in the display window.  Note that the Feb97 version of BDK 1.0 did
  // not properly support this approach but the Apr97 version does
  // support it.
  public synchronized Dimension getPreferredSize(){
    return new Dimension(50,50);
  }//end getPreferredSize()

}//end class Beans02.java
```

Program Listing for the Test Program

This section contains a listing of a test program written to test the *indexed* property of the bean program. The output from running the test is also contained in the listing.

```
/*File Beans02Test.java.java Copyright 1997, R.G.Baldwin
This program was designed to be compiled and executed under JDK 1.1.1.
The program was tested using JDK 1.1.1 and the Apr97 version of
BDK 1.0 under Win95.

This is a minimal test program for Beans02.java.  It is designed to
test only the indexed property of Beans02.  The other properties of
Beans02 can be tested adequately in the BeanBox.

Output from the program is:

Set, get, and display value of 30 in index 2
30
Get, display, and modify entire property array
0
0
30
Using modified array, set, get, and display entire property array
0
1


*/

import java.awt.*;
import java.awt.event.*;
//============================================================
public class Beans02Test extends Frame{
  public static void main(String[] args){
    new Beans02Test();
  }//end main

  public Beans02Test(){//constructor
    setTitle("Copyright 1997, R.G.Baldwin");
```

```
    setLayout(new FlowLayout());
    Beans02 myBean = new Beans02();//instantiate a Bean object
    add(myBean);//Add it to the Frame
    setSize(250,200);
    setVisible(true);

    System.out.println("Set, get, and display value of 30 in index 2");
    myBean.setIndexedProperty(2,30);//store value of 30 in index 2
    System.out.println(myBean.getIndexedProperty(2));//get and display it

    System.out.println("Get, display, and modify entire property array");
    int[] myArray = myBean.getIndexedProperty();//get the entire property
array
    for(int cnt = 0; cnt < myArray.length; cnt++){
      System.out.println(myArray[cnt]); //display it
      myArray[cnt] = cnt;//modify it
    }//end for loop

    System.out.println(
      "Using modified array, set, get, and display entire property
array");
    myBean.setIndexedProperty(myArray);//set property with modified array
    myArray = myBean.getIndexedProperty();//get the entire property array
    for(int cnt = 0; cnt < myArray.length; cnt++)
      System.out.println(myArray[cnt]); //display the modified version

    this.addWindowListener(new Terminate());//terminate when Frame is
closed
  }//end constructor
}//end class Beans02Test.java

//=====================================================================
class Terminate extends WindowAdapter{
  public void windowClosing(WindowEvent e){
    System.exit(0);//terminate the program when the window is closed
  }//end windowClosing
}//end class Terminate
//=====================================================================
```

-end-