Chapter -6 Java Beans

A **Java Bean** is a software component that has been designed to be **reusable** in a variety of different environments. There is no restriction on the capability of a Bean. It may perform a simple function, such as checking the spelling of a document, or a complex function, such as forecasting the performance of a stock portfolio.

Beans are important, because they allow you to build complex systems from software components. These components may be provided by you or supplied by one or more different vendors. Java Beans defines an architecture that specifies how these building blocks can operate together. To better understand the value of Beans, consider the following. Hardware designers have a wide variety of components that can be integrated together to construct a system. Resistors, capacitors, and inductors are examples of simple building blocks. Integrated circuits provide more advanced functionality. All of these different parts can be reused. It is not necessary or possible to rebuild these capabilities each time a new system is needed. Also, the same pieces can be used in different types of circuits. This is possible because the behavior of these components is understood and documented.

Unfortunately, the software industry has not been as successful in achieving the benefits of reusability and interoperability. Large applications grow in complexity and become very difficult to maintain and enhance. Part of the problem is that, until recently, there has not been a standard, portable way to write a software component. To achieve the benefits of component software, a component architecture is needed that allows programs to be assembled from software building blocks, perhaps provided by different vendors. It must also be possible for a designer to select a component, understand its capabilities, and incorporate it into an application. When a new version of a component becomes available, it should be easy to incorporate this functionality into existing code.Fortunately, Java Beans provides just such an architecture.

Advantages of Java Beans

A software component architecture provides standard mechanisms to deal with software building blocks. The following list enumerates some of the specific benefits that Java technology provides for a component developer:

- Bean obtains all the benefits of Java's "write-once, run-anywhere" paradigm.
- The properties, events, and methods of a Bean that are exposed to an application builder tool can be controlled.
- A Bean may be designed to operate correctly in different locales, which makes it useful in global markets.
- Auxiliary software can be provided to help a person configure a Bean. This software is only needed when the design-time parameters for that component are being set. It does not need to be included in the run-time environment.
- The configuration settings of a Bean can be saved in persistent storage and restored at a later time.
- A Bean may register to receive events from other objects and can generate events that are sent to other objects.

Steps for creating a new Bean using netbeans

Here are the steps that you must follow to create a new Bean:

1. Create a directory for the new Bean.

2. Create the Java source file(s).

3. Compile the source file(s).

4. Create a manifest file.

5. Generate a JAR file.

6. Start the BDK (Bean Development Kit).

7. Test.

Explanation

1. Create a Directory for the New Bean (You can use netbeans and create a new project for that)

2. Create the Source File for the New Bean

```
//Colors.java
import java.awt.*;
import java.awt.event.*;

public class Colors extends Canvas {
transient private Color color;
private boolean rectangular;
public Colors() {
addMouseListener(new MouseAdapter() {
public void mousePressed(MouseEvent me) {
change();
}
});
rectangular = false;
setSize(200, 100);
change();
}
public boolean getRectangular() {
return rectangular;
```

```
}
public void setRectangular(boolean flag) {
this.rectangular = flag;
repaint();
}
public void change() {
  color = randomColor();
repaint();
}
private Color randomColor() {
int r = (int)(255*Math.random());
int g = (int)(255*Math.random());
int b = (int)(255*Math.random());
return new Color(r, g, b);
}
public void paint(Graphics g) {
Dimension d = getSize();
int h = d.height;
int w = d.width;
g.setColor(color);
if(rectangular) {
g.fillRect(0, 0, w-1, h-1);
}
else {
g.fillOval(0, 0, w-1, h-1);
}
}
}
```

3. Compile the Source Code for the New Bean

Compile the source code to create a class file. Type the following:

javac Colors.java.

4. Create a Manifest File

You must now create a manifest file. First, switch to the c:\bdk\demo directory. This is the directory in which the manifest files for the BDK demos are located. Put the source code for your manifest file in the file colors.mft. It is shown here:

Name: sunw/demo/colors/Colors.class

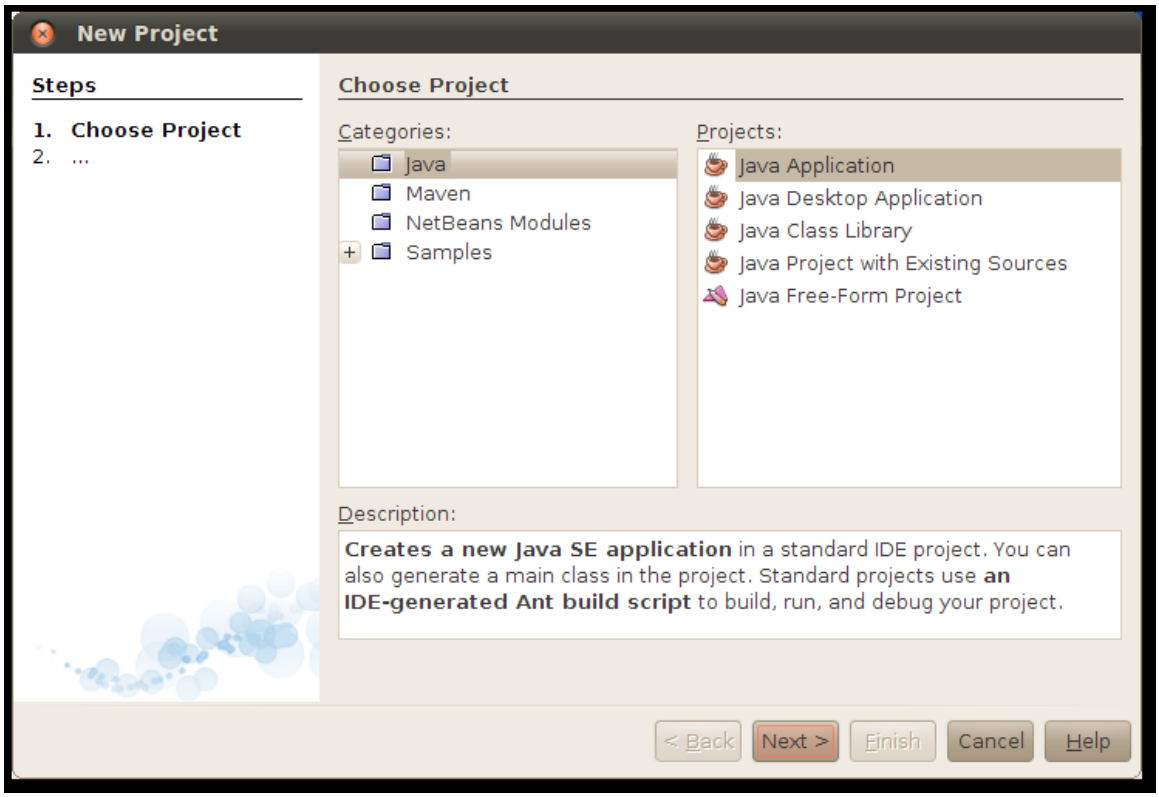
Java-Bean: True

This file indicates that there is one .class file in the JAR file and that it is a Java Bean. Notice that the Colors.class file is in the package sunw.demo.colors and in the

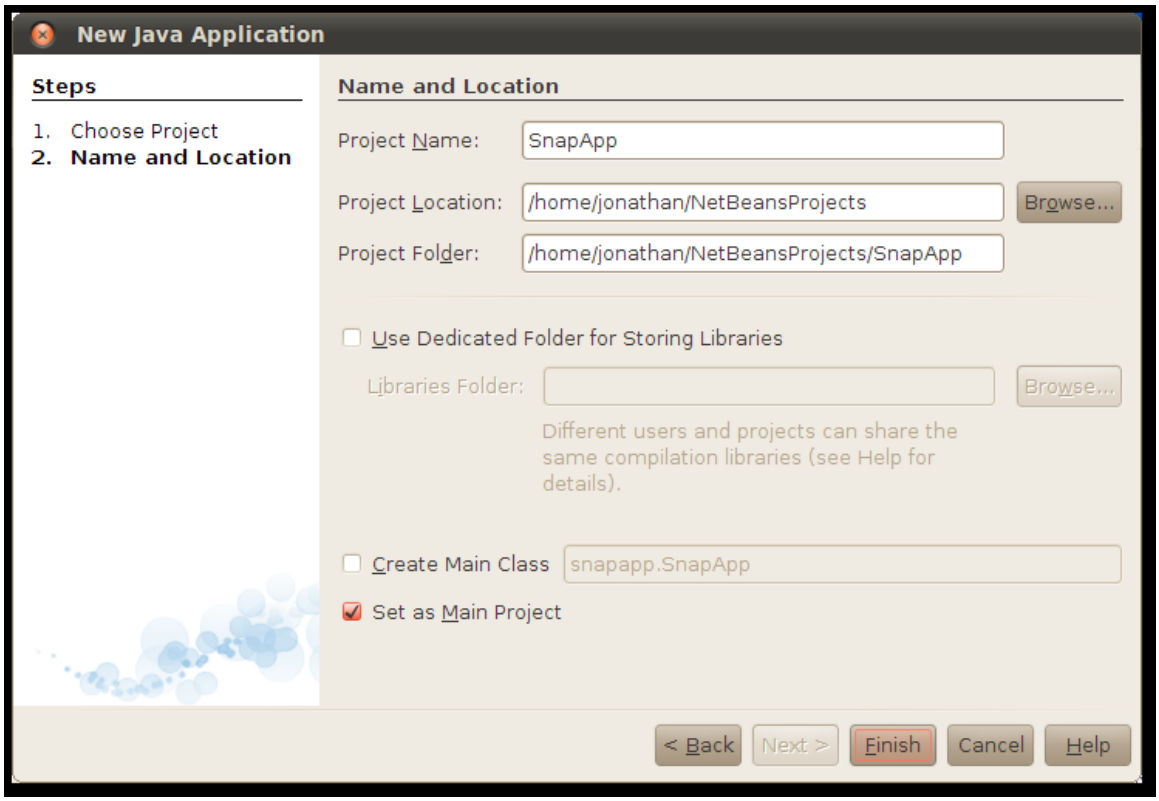
subdirectory sunw\demo\colors relative to the current directory.

Creating Java Beans using NetBeans

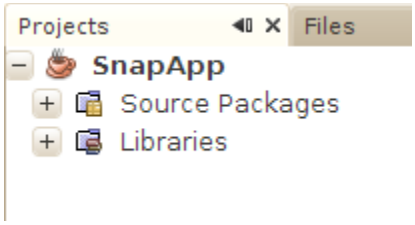
1. Start NetBeans. Choose File > New Project... from the menu.



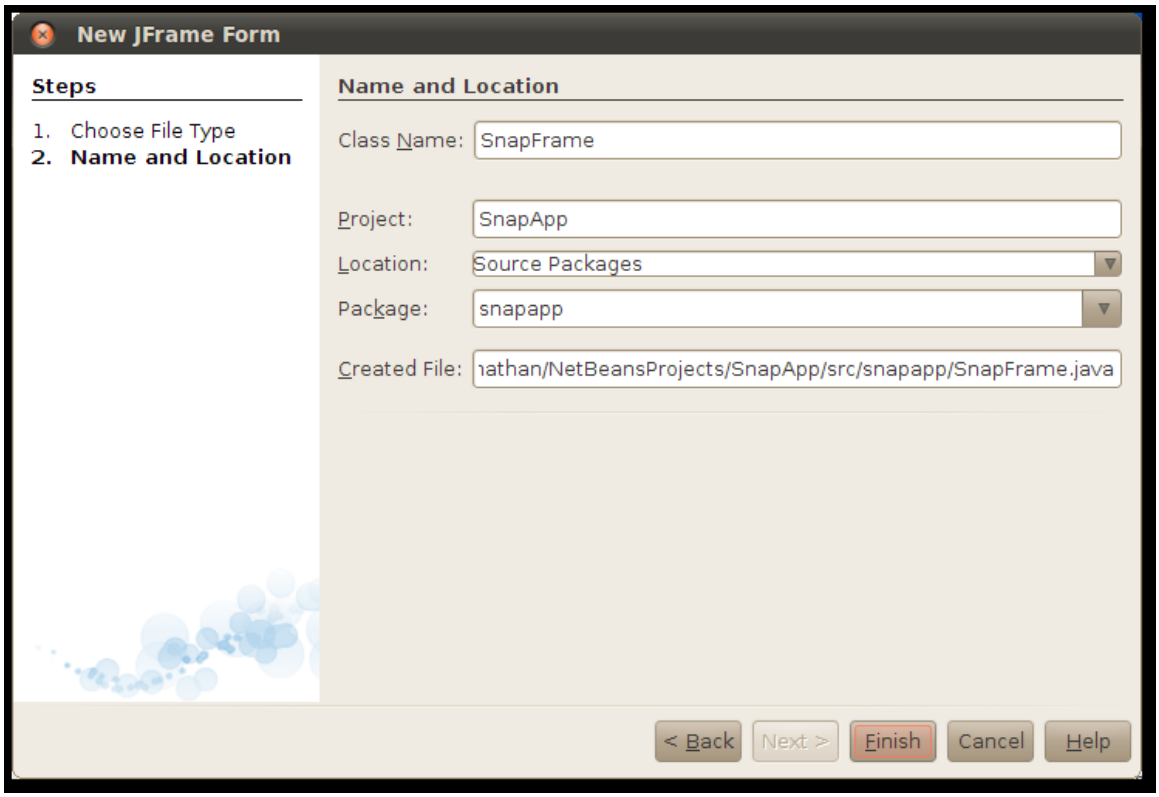
2. Select Java from the Categories list and select Java Application from the Projects list. Click Next



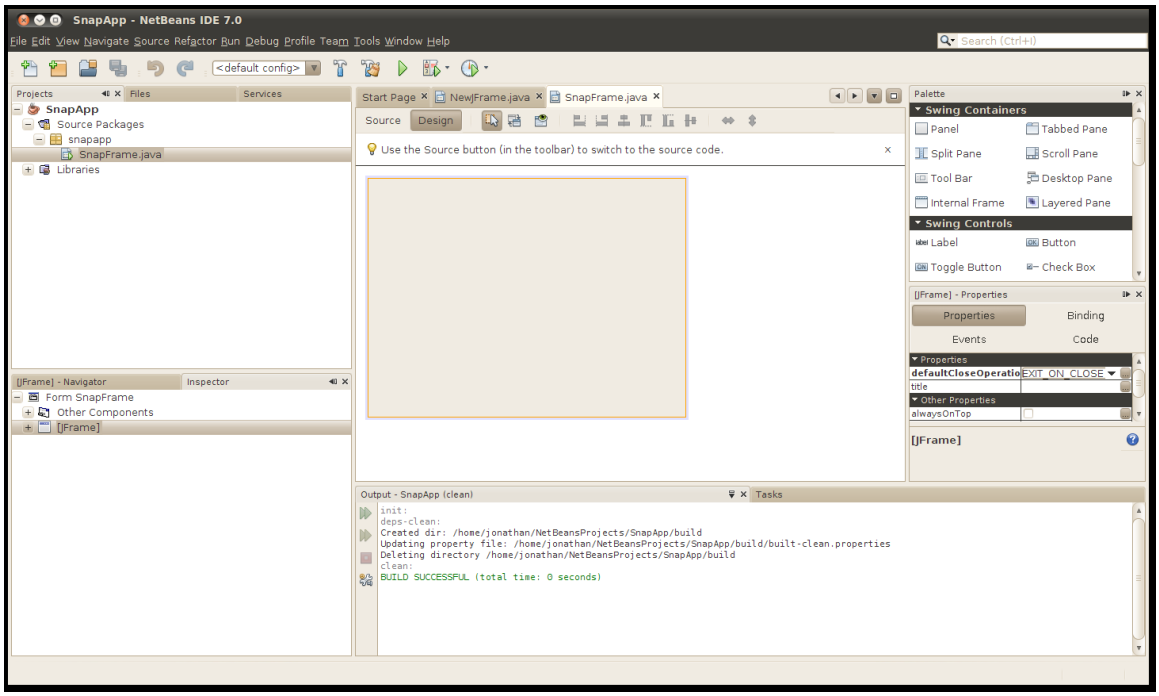
3. Enter SnapApp as the application name. Uncheck Create Main Class and click Finish. NetBeans creates the new project and you can see it in NetBeans' Projects pane:



4. Right-click on the SnapApp project and choose New > JFrame Form... from the popup menu.



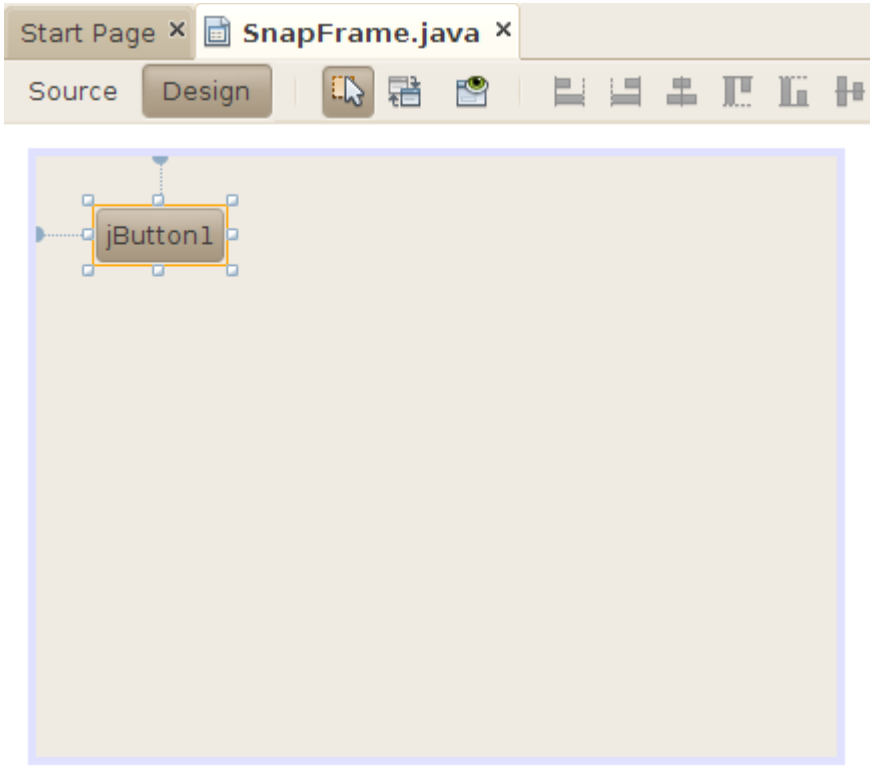
5. Fill in SnapFrame for the class name and snapapp as the package. Click Finish. NetBeans creates the new class and shows its visual designer:



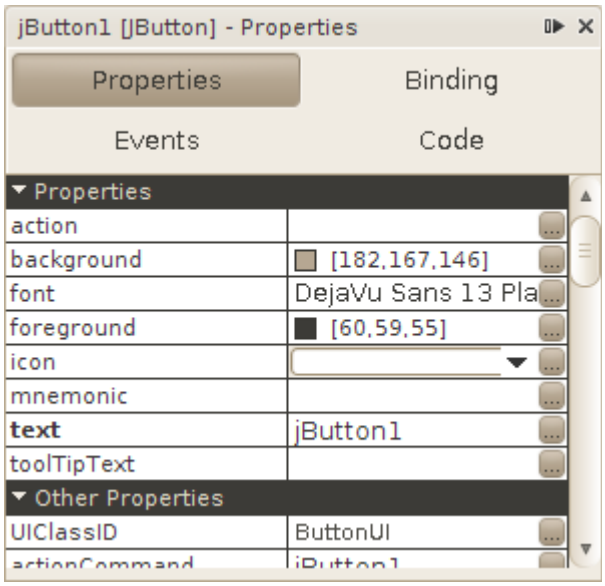
In the Projects pane on the left, you can see the newly created SnapFrame class. In the center of the screen is the NetBeans visual designer. On the right side is the Palette, which contains all the

components you can add to the frame in the visual designer.

6. Take a closer look at the Palette. All of the components listed are beans. The components are grouped by function. Scroll to find the Swing Controls group, then click on Button and drag it over into the visual designer. The button is a bean!



Under the palette on the right side of NetBeans is an inspector pane that you can use to examine and manipulate the button. Try closing the output window at the bottom to give the inspector pane more space.



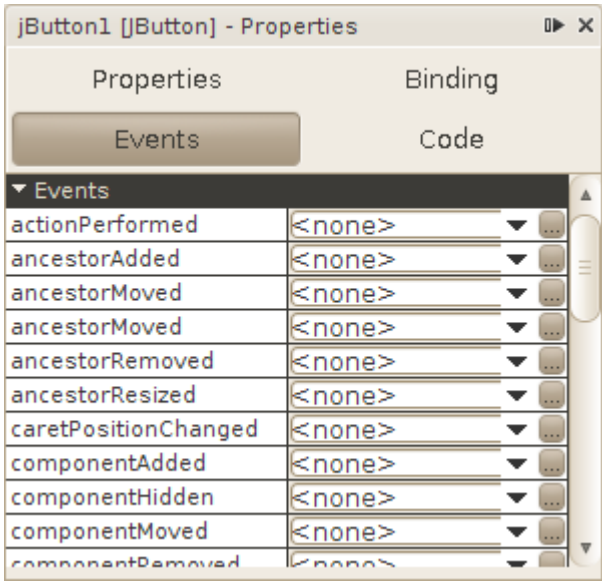
Properties

The properties of a bean are the things you can change that affect its appearance or internal state. For the button in this example, the properties include the foreground color, the font, and the text that appears on the button. The properties are shown in two groups. Properties lists the most frequently used properties, while Other Properties shows less commonly used properties.

Go ahead and edit the button's properties. For some properties, you can type values directly into the table. For others, click on the ... button to edit the value. For example, click on ... to the right of the foreground property. A color chooser dialog pops up and you can choose a new color for the foreground text on the button. Try some other properties to see what happens. Notice you are not writing any code.

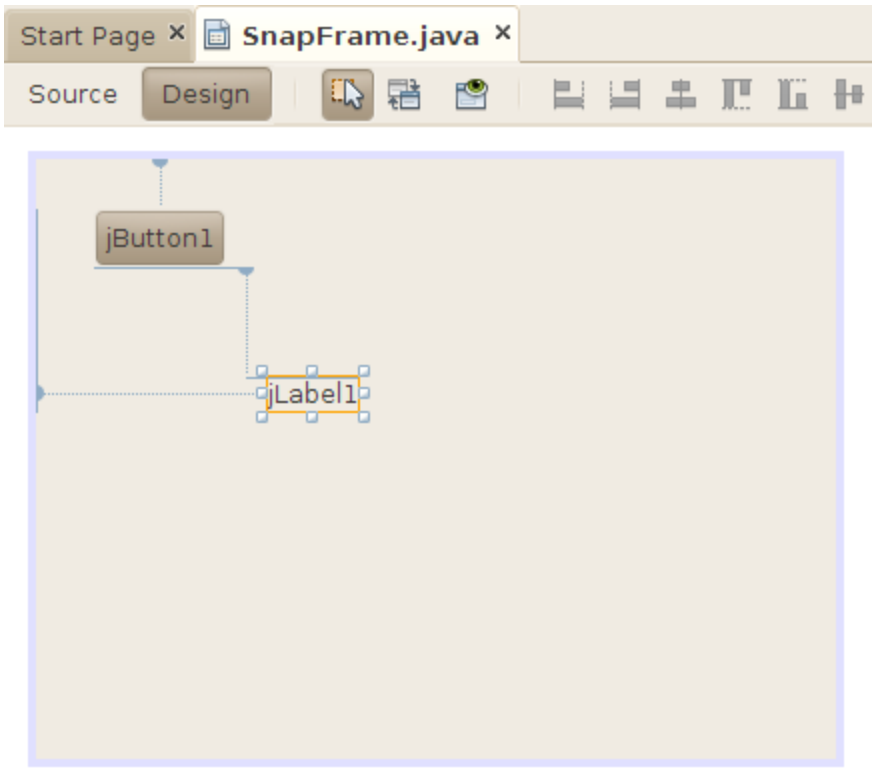
Events

Beans can also fire events. Click on the Events button in the bean properties pane. You'll see a list of every event that the button is capable of firing.



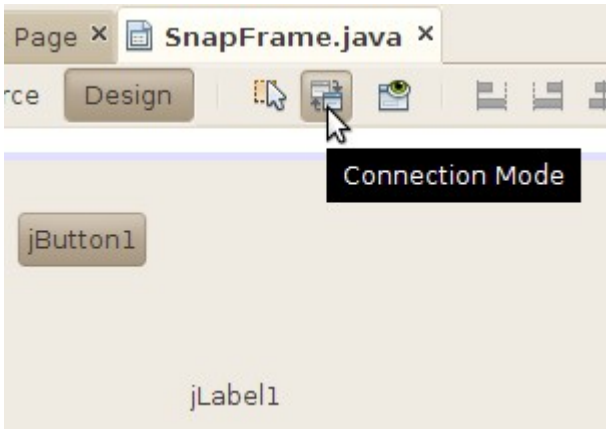
You can use NetBeans to hook up beans using their events and properties. To see how this works, drag a Label out of the palette into the visual designer for SnapFrame.

Add a label to the visual designer



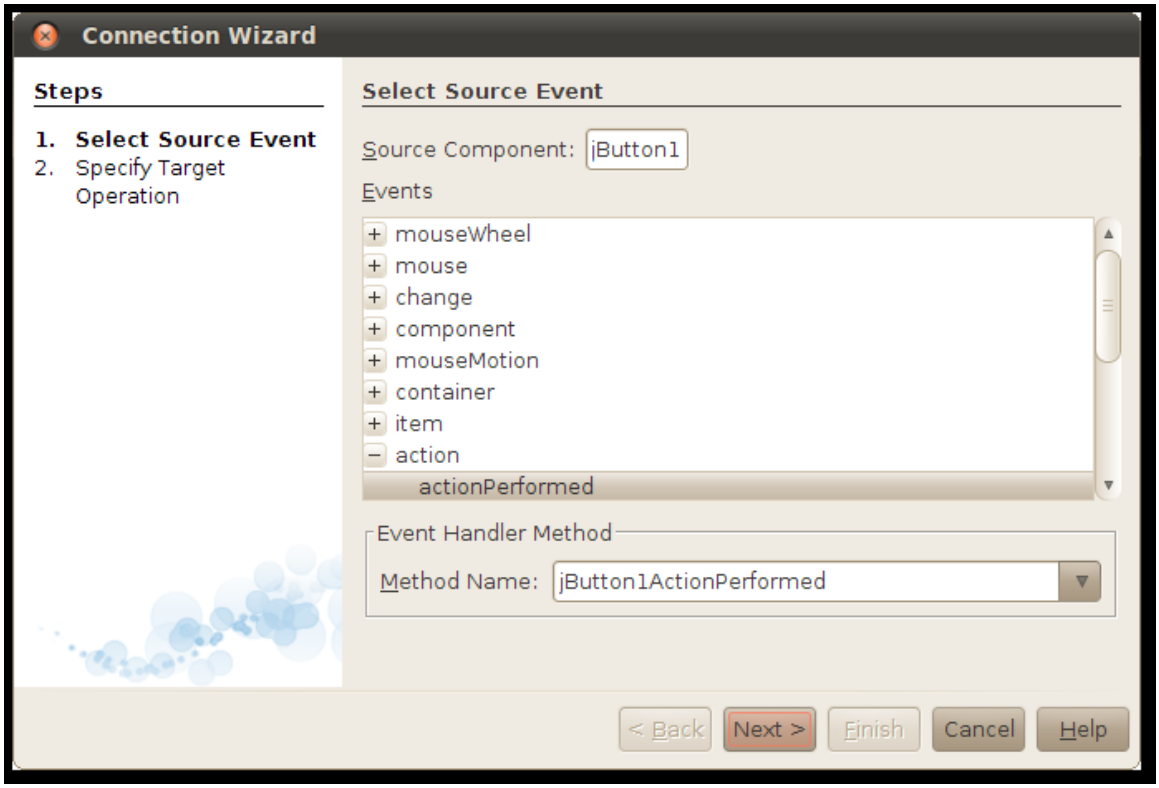
Wiring the Application

To wire the button and the label together, click on the Connection Mode button in the visual designer toolbar.

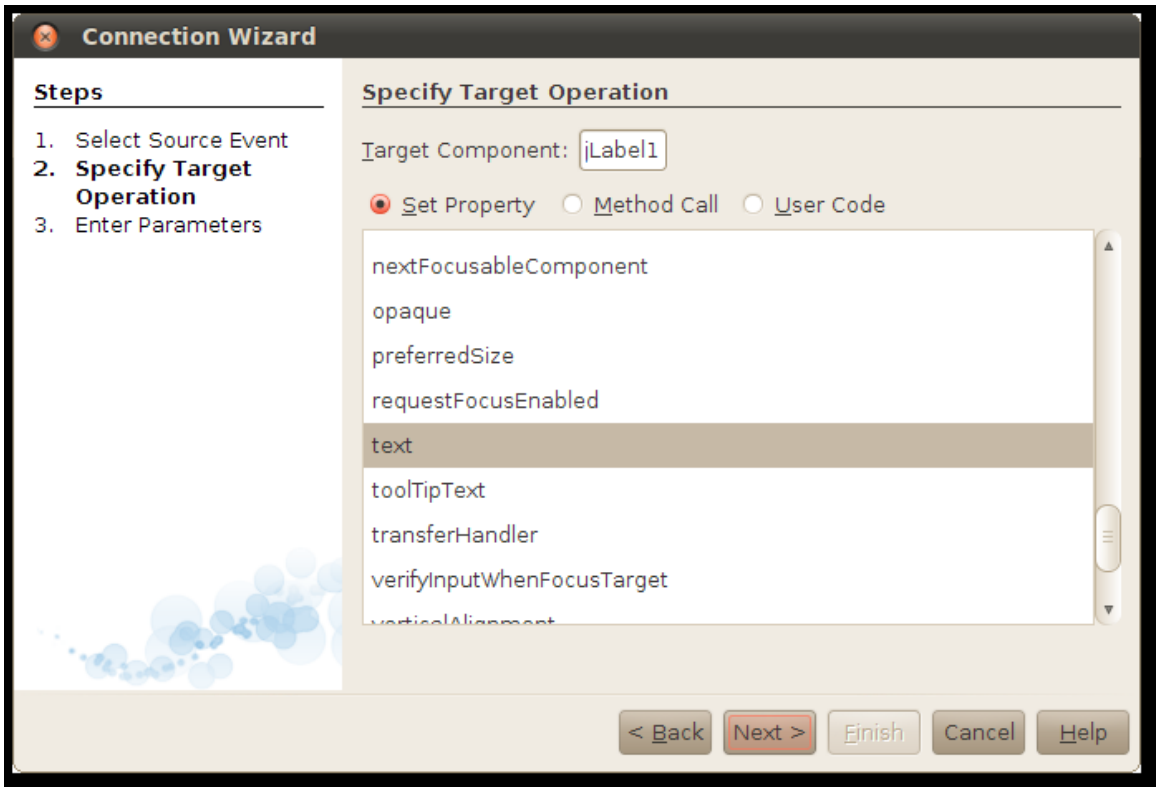


Click on the button in the SnapFrame form. NetBeans outlines the button in red to show that it is the component that will be generating an event.

Click on the label. NetBeans' Connection Wizard pops up. First you will choose the event you wish to respond to. For the button, this is the action event. Click on the + next to action and select actionPerformed. Click Next >.



Now you get to choose what happens when the button fires its action event. The Connection Wizard lists all the properites in the label bean. Select text in the list and click Next.

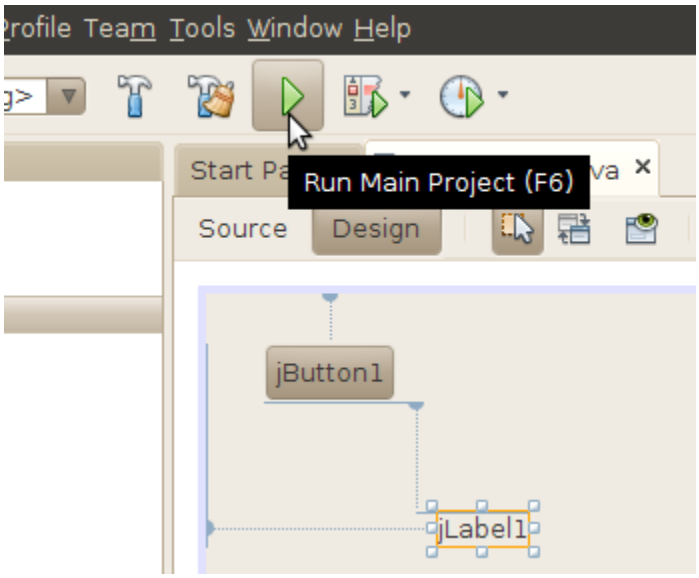


In the final screen of the Connection Wizard, fill in the value you wish to set for the text property. Click on Value, then type You pressed the button! or something just as eloquent. Click Finish.

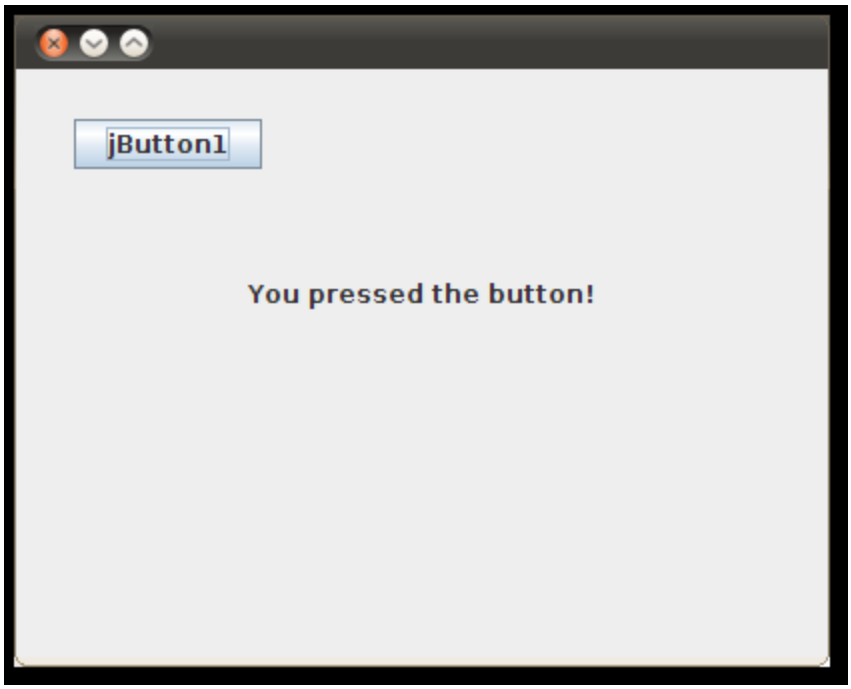
NetBeans wires the components together and shows you its handiwork in the source code editor.

```
    @SuppressWarnings("unchecked")
    Generated Code

    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        jLabel1.setText("You pressed the button!");
    }

    /**
```

Click on the Design button in the source code toolbar to return to the UI designer. Click Run Main Project or press F6 to build and run your project.



NetBeans builds and runs the project. It asks you to identify the main class, which is SnapFrame. When the application window pops up, click on the button. You'll see your immortal prose in the label.



Notice that you did not write any code. This is the real power of JavaBeans — with a good builder tool like NetBeans, you can quickly wire together components to create a running application.

Using a Third-Party Bean

Almost any code can be packaged as a bean. The beans you have seen so far are all visual beans, but beans can provide functionality without having a visible component.

The power of JavaBeans is that you can use software components without having to write them or understand their implementation.

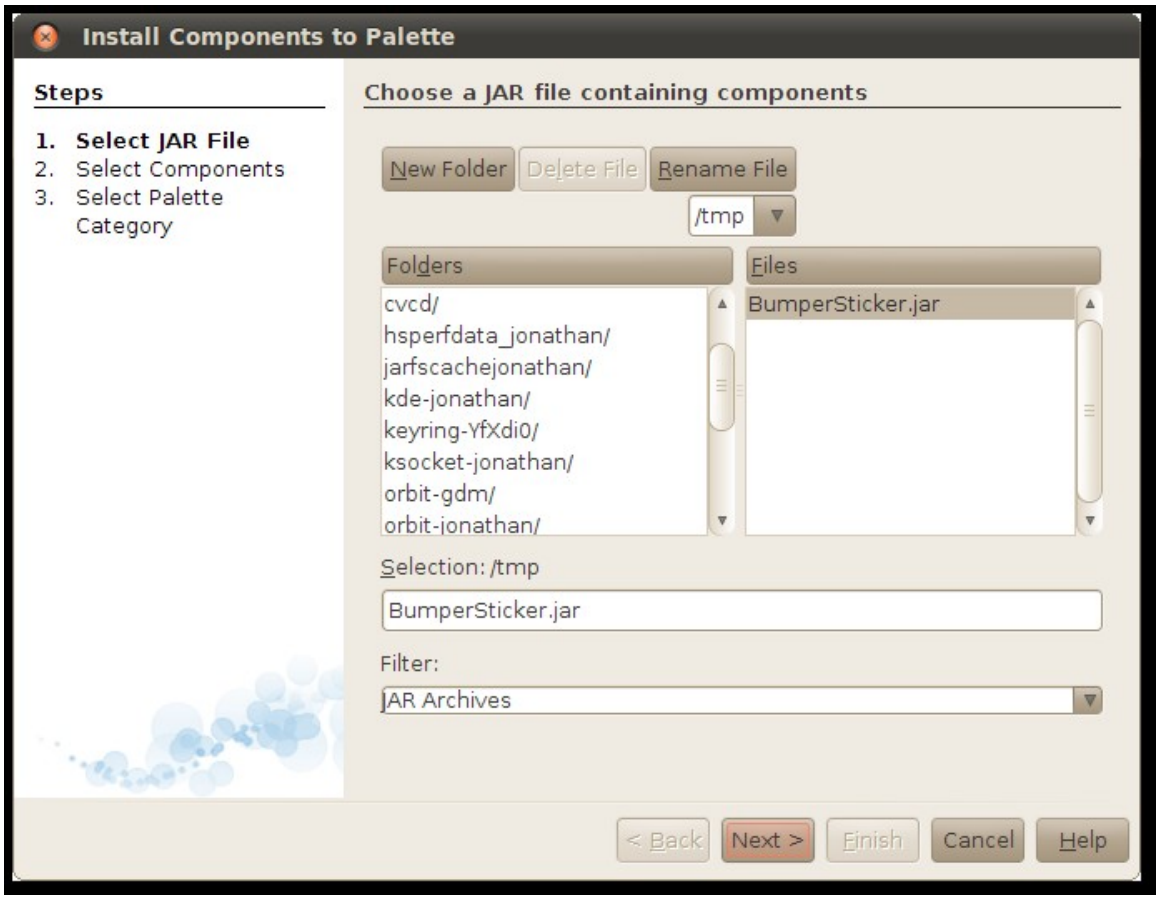
This page describes how you can add a JavaBean to your application and take advantage of its functionality.

Adding a Bean to the NetBeans Palette

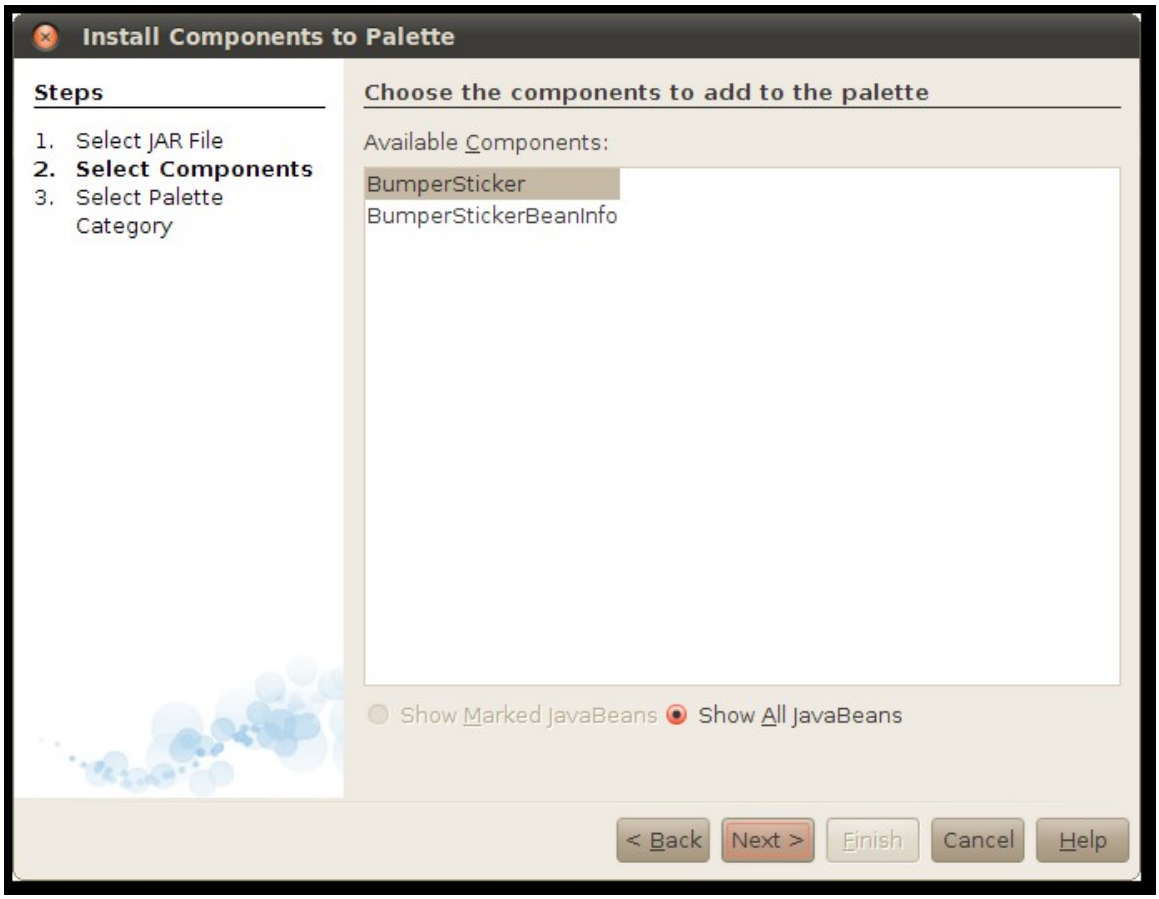
Download an example JavaBean component, BumperSticker. Beans are distributed as JAR files. Save the file somewhere on your computer. BumperSticker is graphic component and exposes one method, go(), that kicks off an animation.

To add BumperSticker to the NetBeans palette, choose Tools > Palette > Swing/AWT Components from the NetBeans menu.

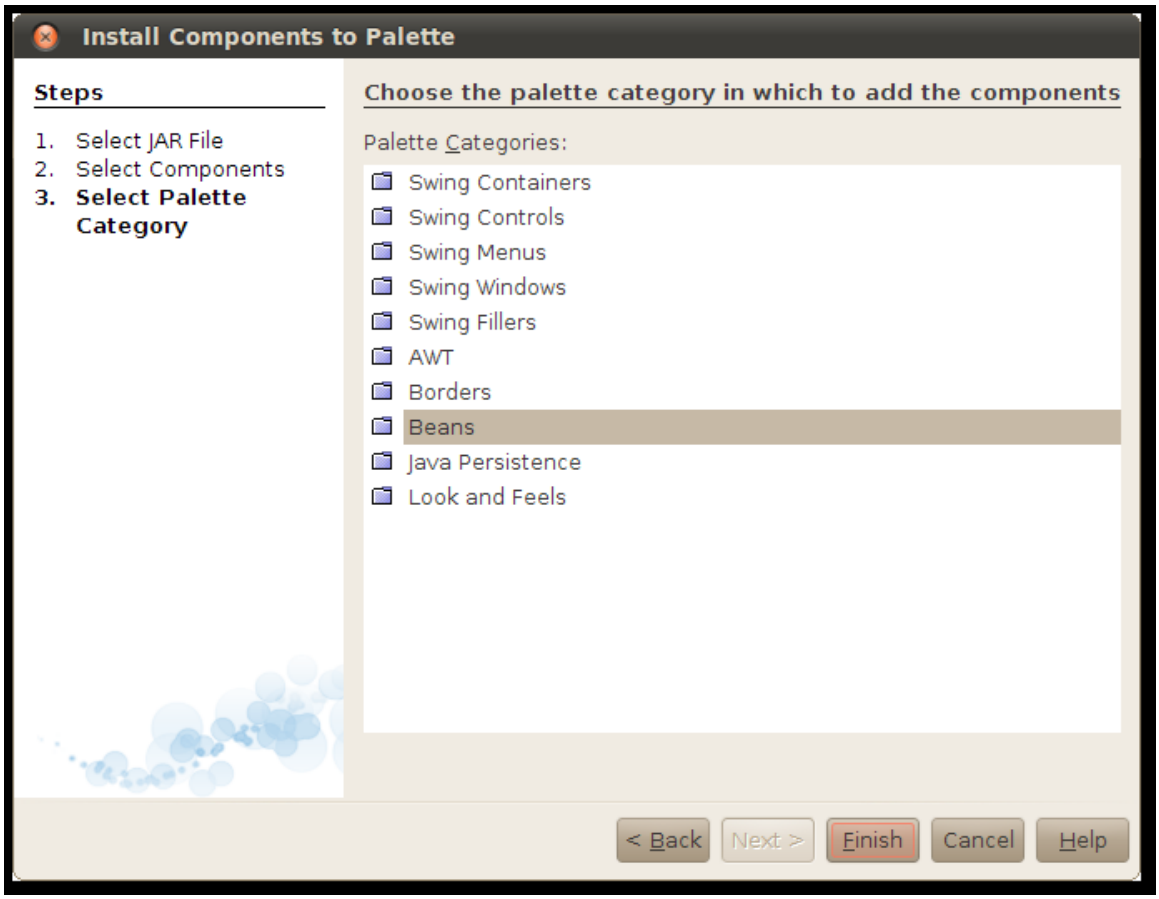
Click on the Add from JAR... button. NetBeans asks you to locate the JAR file that contains the beans you wish to add to the palette. Locate the file you just downloaded and click Next.



NetBeans shows a list of the classes in the JAR file. Choose the ones you wish you add to the palette. In this case, select BumperSticker and click Next



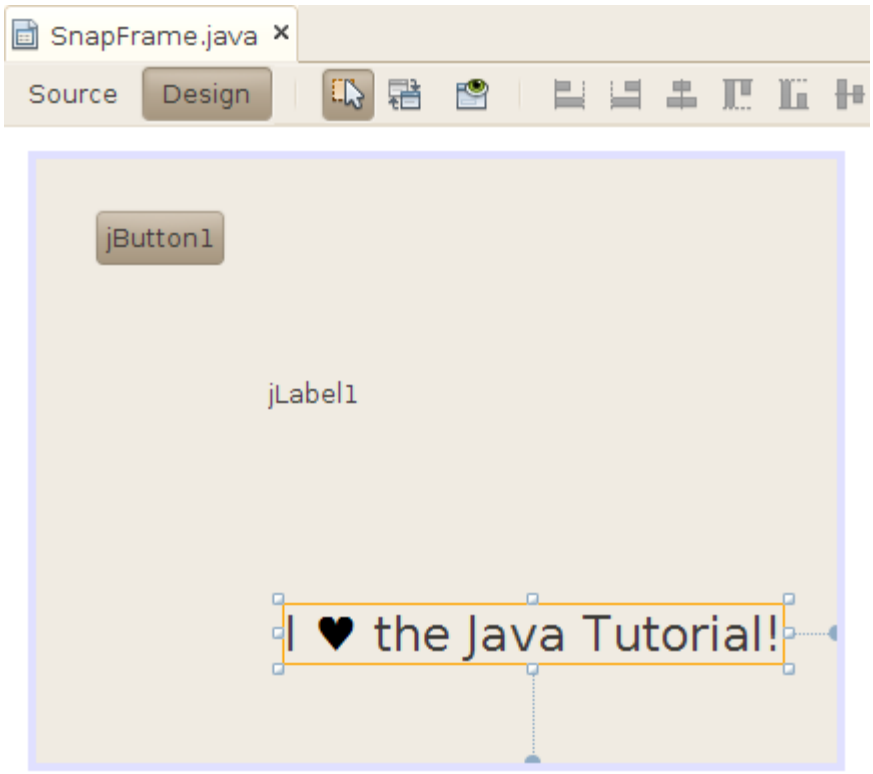
Finally, NetBeans needs to know which section of the palette will receive the new beans. Choose Beans and click Finish.



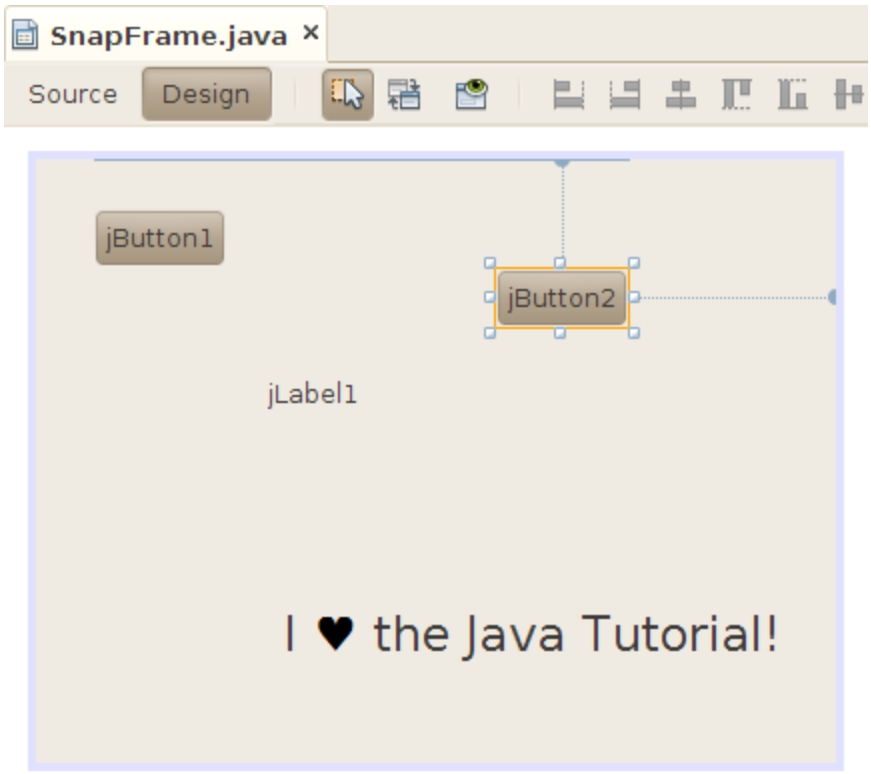
Click Close to make the Palette Manager window go away. Now take a look in the palette. BumperSticker is there in the Beans section.

Using Your New JavaBean

Go ahead and drag BumperSticker out of the palette and into your form.



You can work with the BumperSticker instance just as you would work with any other bean. To see this in action, drag another button out into the form. This button will kick off the BumperSticker's animation.



Wire the button to the BumperSticker bean, just as you already wired the first button to the text field.

Begin by clicking on the Connection Mode button.

Click on the second button. NetBeans gives it a red outline.

Click on the BumperSticker component. The Connection Wizard pops up.

Click on the + next to action and select actionPerformed. Click Next >.

Select Method Call, then select go() from the list. Click Finish.

Run the application again. When you click on the second button, the BumperSticker component animates the color of the heart.

Again, notice how you have produced a functioning application without writing any code.