Callbacks

Callbacks refer to a mechanism in which a library or utility class provides a service to clients that are unknown to it when it is defined.

- Suppose, for example, that a server class creates a window and recognizes events in that window.
- Suppose, in addition, that various clients need to be informed of these events so that they can perform actions of their own.

When such an event occurs, the server class needs to invoke methods in one or more of the clients associated with the server, but the server has no prior knowledge of which clients will be attached and the names of the methods that need to be called.

In C and C++, this situation is handled by having the clients register with the server by sending it function pointers that refer to the methods that need to be executed when an event occurs.

These functions are called callback functions.

When an appropriate event happens, the server calls the methods in the appropriate clients by invoking the formal parameters that denote the function pointers that were register by the clients.

But Java has no user-defined function pointers.

Java provides the callback mechanism with interfaces using the following steps:

1. An interface is defined with an abstract method whose name is known to the server.

2. Each client that has actions to be performed when a server event occurs implements the interface, thereby giving code for the abstract method.
3. When a client registers with the server, the server holds an instance variable that refers to the client and whose type is the interface type.
4. The server class invokes the client action by calling the interface method for that client.

Example

We define a server that creates a window with four buttons.

The server maintains a list that can hold references to any number of clients.

When any of the first three buttons is clicked, an action in one of the first three clients will be performed.

When the fourth button is pressed, the actions in all of the clients will be executed.

The connection between the server and the clients is provided by an interface called Callback.

```
interface Callback
{
        void performCallback(String str);
}
```

The entire program is controlled by a class that creates the server and three clients.

```
import java.util.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class Callbacks
{
   public static void main(String [] args)
   {
         WindowServer ws = new WindowServer();
         ws.setSize(300,200);
         ws.setVisible(true);
         ClientOne client1 = new ClientOne(ws);
         ClientTwo client2 = new ClientTwo(ws);
         ClientThree client3 = new ClientThree(ws);
   }
}
```

Note that the clients are made aware of the server by passing a reference to the server to their constructors.

The server class WindowServer contains:

- An instance variable of type List that contains references to the clients that register.
- A constructor that creates the window with the four buttons.
- An inner class ButtonHandler for handling the button clicks.
- A method for registering clients, adding them to the List.

- A method for unregistering clients, deleting them from the List.

Note that all references to clients are as instances of the interface Callback.

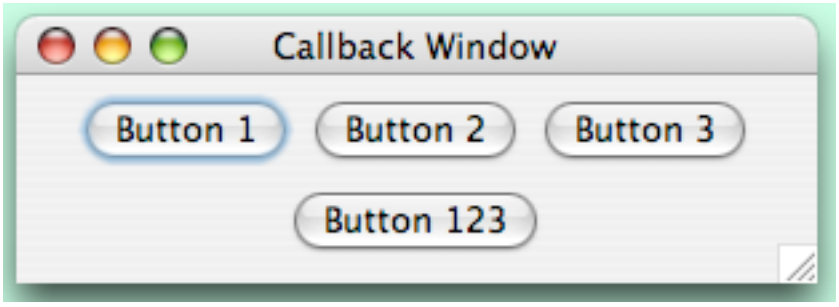
```
class WindowServer extends JFrame
{
   private java.util.List clientObjs = new ArrayList();
   private JButton button1, button2, button3, button123;
   WindowServer()
   {
      super("Callback Window");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      Container cp = getContentPane();
      cp.setLayout(new FlowLayout());
      button1 = new JButton("Button 1");
      button1.addActionListener(new ButtonHandler(1));
      button2 = new JButton("Button 2");
      button2.addActionListener(new ButtonHandler(2));
      button3 = new JButton("Button 3");
      button3.addActionListener(new ButtonHandler(3));
      button123 = new JButton("Button 123");
      button123.addActionListener(new ButtonHandler(123));
      cp.add(button1); cp.add(button2);
      cp.add(button3); cp.add(button123);
   }
```

```
class ButtonHandler implements ActionListener
{
   private int value;

   ButtonHandler(int n)
   {
         value = n;
   }

   public void actionPerformed(ActionEvent evt)
   {
      if (value>100)
      {
         System.out.println("Button 123 pressed.");

         notifyClients("Button 123");
      }
      else
      {
         System.out.println("Button " + value + " pressed.");

         ((Callback)clientObjs.get(value-1)).
                                 performCallback("says Hello");
      }
   }
}


void registerCallback(Callback client)
{
   clientObjs.add(client);
}
```

```
    void unregisterCallback(Callback client)
    {
        clientObjs.remove(client);
    }

    void notifyClients(String str)
    {
        for (Iterator it = clientObjs.iterator(); it.hasNext();  )
            ((Callback)(it.next())).performCallback(str);
    }
}
```

Observe that Objects in the List need to be cast into Callback objects before the interface method *performCallback* can be called.

Window Server



One client is defined below.

The other two clients can be defined in an analogous manner.

```
class ClientOne implements Callback
{
   private WindowServer server;

   ClientOne(WindowServer ws)
   {
         System.out.println("Creating Client One");
         server = ws;
         server.registerCallback(this);
   }

   public void performCallback(String str)
   {
         System.out.println("Client One " + str);
   }
}
```

Sample Output

```
Creating Client One
Creating Client Two
Creating Client Three

Button 1 pressed.
Client One says Hello
Button 2 pressed.
Client Two says Hello
Button 3 pressed.
Client Three says Hello
Button 123 pressed.
Client One Button 123
Client Two Button 123
Client Three Button 123
```

Note: The event handling mechanism in Java works in a similar manner.

Objects that implement "listener" interfaces are registered with the objects that generate events (buttons, text fields, and so on) so that they can be notified when the particular events occur.