Unit 12 RMI, CORBA and Java Beans

Structure:

12.1 Introduction

In the last unit, we have discussed database management concepts in Java. In this unit, we shall discuss some advanced Java concepts like Remote Method Invocation, Common Object Request Broker Architecture (CORBA) and Java Beans.

There are different approaches used for developing distributed applications. The Internet and the Web are examples of distributed systems that have been developed using the Client/Server approach. The transfer of data is one of the most important processes in distributed applications. The default implementation of the message passing method in Java transfers data from the calling object to the called object within a single Java Virtual Machine (JVM). Another way to achieve the same is using ***Remote Method Invocation (RMI)***, which allows objects in different hosts to send and receive messages. Both the methods achieve the same goal. However, the method call approach used by RMI is much easier to use. RMI allows objects in different JVMs belonging to different hosts to send and receive message. CORBA stands for **Common Object Request Broker Architecture**. CORBA is a distributed computing technology where the participating objects need not only be written in Java. **JavaBeans** is the

software component architecture for Java. It allows constructing applications efficiently by configuring and connecting components called Beans. The Beans need to be written and tested with a rich set of mechanisms for interaction between objects, along with common actions that most objects will need to support such as persistence and event handling. ***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.

Objectives:

After studying this unit, you should be able to:

- describe Remote Method Invocation (RMI)
- implement RMI
- describe Common Object Request Broker Architecture (CORBA) and Java Interface Definition Language (IDL)
- explain Java Beans

12.2 Remote Method Invocation

Distributed applications are applications that execute across multiple host systems. Objects executing on one host system can invoke the methods of objects on remote hosts. The remotely invoked methods can return values to the local objects.

There are different approaches used for developing distributed applications. The Internet and the Web are examples of distributed systems that have been developed using the Client/Server approach.

The transfer of data is one of the most important processes in distributed applications. The default implementation of the message passing method in Java transfers data from the calling object to the called object within a single Java Virtual Machine (JVM). You have learned how to cross process boundaries to transfer the data from one host to the other using sockets. Another way to achieve the same is using Remote Method Invocation (RMI), which allows objects in different hosts to send and receive messages. Both the methods achieve the same goal. However, the method calls approach used by RMI in much easier to use. RMI allows objects in different JVMs belonging to different hosts to send and receive message.

12.2.1 RMI Terminology

RMI is built upon the specification of how local and remote objects interoperate. **Local objects** are objects that execute on the local machine. **Remote objects** are objects that execute on all the other machines. Objects on remote hosts are exported so that they can be invoked remotely. An object exports itself by registering itself with a **Remote Registry Server.** A **Remote Registry Server** is a service that runs on a server and helps the objects on other hosts to remotely access its registered objects. The *registry* service maintains a database of all the named remote objects.

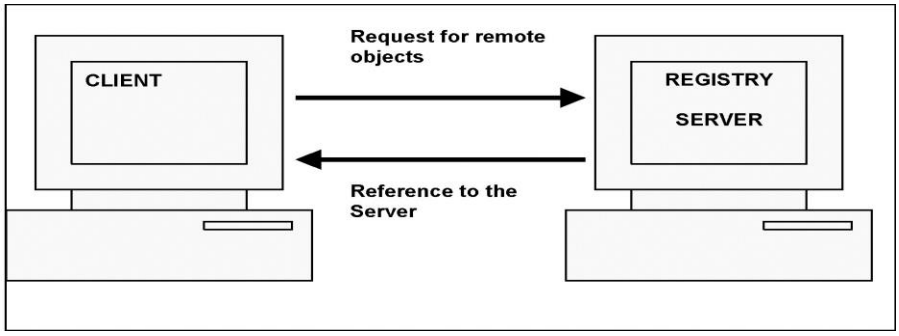

Figure 12.1: RMI Mechanism

A Distributed Application Created Using RMI

Objects that are exported for remote access must implement the interface **RemoteInterface**. This interface identifies the object to be accessed remotely. All the methods that are to be invoked remotely must throw a **RemoteException.** This exception is used to handle the errors that may occur during the invocation of a remote method.

Java's RMI approach is organized into a **client/server** framework. A local object that invokes a method of a remote object is referred to as a **client** object, and the remote object whose methods are invoked is referred to as a **server** object.

Java's RMI approach makes use of **stubs** and **skeletons.** A **stub** is a **local object** on the client's machine that acts as a proxy for a **remote object.** The stub provides the methods of the remote object. Local objects invoke the methods of the stubs as if they were methods of the remote objects. The stub communicates this method invocation to the remote object through a

skeleton that is implemented on a remote host. The skeleton is the proxy of the client machine's object that is located on the remote host.

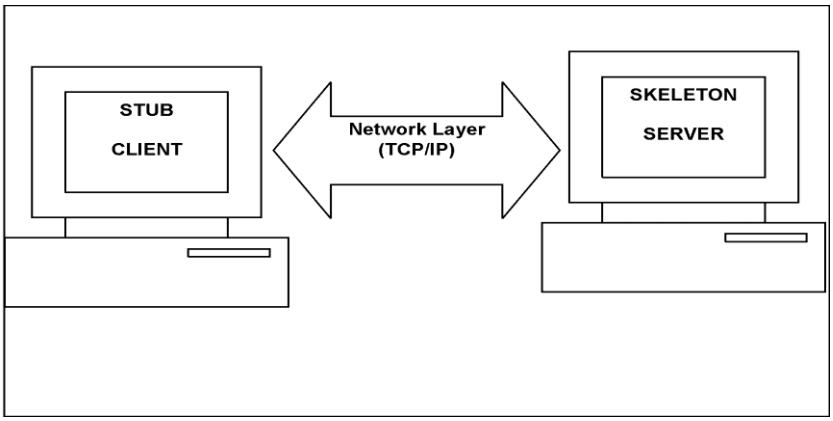

Figure 12.2: Use of Stubs and Skeletons

Usage of Stubs and Skeletons to support Client/Server Communication

The stub and the skeleton communicate through a **remote reference layer.** This layer provides stubs the capability to communicate with skeletons through a transport protocol. RMI uses the **TCP protocol** for transporting information.

Self Assessment Questions

1. RMI stands for _____.
2. A _____ is a service that runs on a server and helps the objects on other hosts to remotely access its registered objects.
3. Objects that are exported for remote access must implement the interface called _____.
4. All the methods that are to be invoked remotely must throw the exception called _____.
5. RMI uses the _____ protocol for transporting information.

12.3 Common Object Request Broker Architecture (CORBA)

CORBA stands for Common Object Request Broker Architecture. RMI, discussed in previous section needs the 2 objects participating in communication be written in Java. CORBA is a distributed computing technology where the participating objects need not only be written in Java.

12.3.1 What is Java IDL?

- Java IDL is a technology for distributed objects-that is, objects interacting on different platforms across a network.
- Java IDL is similar to RMI (Remote Method Invocation), which supports distributed objects written entirely in the Java programming language. However, Java IDL enables objects to interact regardless of whether they're written in the Java programming language or another language such as C, C++, COBOL, or others.
- Java IDL is based on the Common Object Request Brokerage Architecture (CORBA), an industry-standard distributed object model.
- A key feature of CORBA is IDL, a language-neutral **Interface Definition Language.** Each language that supports CORBA has its own IDL mapping and as its name implies, Java IDL supports the mapping for Java. CORBA and the IDL mappings are the work of an industry consortium known as the OMG, or **Object Management Group.**
- To support interaction between objects in separate programs, Java IDL provides an Object Request Broker, or ORB. The ORB is a class library that enables low-level communication between Java IDL applications and other CORBA-compliant applications.

12.3.2 Example: The Hello Client-Server

This tutorial teaches the basic tasks in building a CORBA distributed application using Java IDL. You will build the classic Hello World program as a distributed application. The Hello World program has a single operation that returns a string to be printed.

- The client (applet or application) invokes the sayHello operation of the HelloServer.
- The ORB transfers that invocation to the server object registered for IDL interface.
- The server's sayHello method runs, returning a Java String.
- The ORB transfers that String back to the client.
- The client prints the value of the String.

Self Assessment Questions

6. CORBA stands for Common _____.
7. IDL stands for _____.
8. CORBA and the IDL mappings are the work of an industry consortium known as the _____.

12.4 Java Beans

JavaBeans is the software component architecture for Java. It allows constructing applications efficiently by configuring and connecting components called Beans. The Beans need to be written and tested with a rich set of mechanisms for interaction between objects, along with common actions that most objects will need to support such as persistence and event handling.

12.4.1 The BeanBox

The BeanBox is a very simple test container. It allows you to try out both the BDK example beans and your own newly created beans.

The BeanBox allows you to:

- Drop beans onto a composition window.
- Resize and move beans around.
- Edit the exported properties of a bean.
- Connect a bean event source to an event handler method.
- Connect together bound properties on different beans.
- Save and restore sets of beans.
- Make applets from beans.
- Get an introspection report on a bean.
- Add new beans from JAR files.

Note that the BeanBox is intended as a test container and as a reference base, but it is not intended as a serious application development tool.

Advantages of Java Beans

Java Beans is a software component architecture that 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:

A 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.

12.4.2 Running the BeanBox

To start the BeanBox first change to the beanbox directory and then run beanbox. Note that you need to have the JDK "bin" directory on your path and you should be using JDK 1.1 or later.

The BeanBox comes up as three separate windows:

- The middle window is the main beanbox composition window.
- The left-hand window is the Toolbox palette displaying available beans that can be dropped onto the composition window.
- The right-hand window is a ***PropertySheet*** showing the properties for the currently selected bean.

- **Dropping beans onto the composition window :**
  To add a bean to the composition window first click on the bean's name or icon in the left-hand ToolBox window. Then click on the location in the center composition window where you want the new bean to appear. The chosen bean will appear centered at the new location and will become the current selected bean for editing.

- **Selecting a bean in the composition window:**
  The currently selected bean is marked with a black-and-white hashed boundary. The right-hand PropertySheet window shows its properties and the "edit" menu provides access to its events, bound properties, etc. To select a bean you must click the mouse just outside of the bean in the boundary area where the black-and-white hashed boundary appears. Some beans won't notice a click on the bean itself, and require you to click in the surrounding border area.

- **Moving or resizing a bean:**
  You can move the currently selected bean by clicking the mouse on one of the black-and-white hashed borders and the holding the mouse down and dragging the bean. Some beans, such as the BeanBox bean or the ExplicitButton, also allow themselves to be resized. You can try to resize the currently selected bean by clicking on one of the corners of the black-and-white boundary and holding the mouse down and stretching the corner out.

- **Editing a beans properties :**
  You can edit the public properties of the currently selected bean using the right-hand PropertySheet window. For each editable property the PropertySheet window shows the name of the property and its current value. The current value may be shown as
  - An editable text field.
  - Or a selection in a choice menu
  - Or a painted value, where you can click on the painted value to bring up a small model dialog for that property.

- **Using a bean Customizer :**
  For those beans that have customizers, the BeanBox allows you to run the customizer to configure your beans. If the currently selected bean has a customizer, then the “edit” menu on the central window will include a “Customize…….” entry. If you select that array, then a dialog window will come up with the bean’s customizer.

- **Connecting an event handler :**
  The BeanBox allows you to connect an event from the currently selected bean to a target event handling method on any other bean. The “edit” menu on the central composition window has an “events” menu that has a sub-menu for all the different kinds of events that the currently selected bean fires. These events are grouped and named according to their EventListener interfaces.

- **Connecting a bound property :**
  The BeanBox allows you to connect a bound property from a source bean to a target property on some other bean. Then when the bound property on the source bean is changed, the associated target property is also automatically updated.

If the currently selected bean supports bound properties, then the “edit” menu will include a “Bind properly……” item. If you chose this menu item, then the BeanBox will bring up a dialog showing the bound properties available on the source bran. You can then select a source property and press “OK”. The beanbox will then draw a “rubber band” line. You can then click on the border of the target bean to which you want the property bound.

The BeanBox will then bring up a dialog listing the target properties on the target bean that match the of the source property. You can select a target property and press “ok”. Now when you change the source property on the source bean then the target property on the target bean will also be updated.

- **Saving and restoring beans:**
  Once you have set up some beans in the BeanBox, you can use the “File” menu’s “Save…” sub-menu to save away the current state of the BeanBox. This uses Java Object Serialization to automatically store away all the state of the beans into a named file. You can then use the “File” menu’s “Clear” item to discard the current set of beans and use the “Load…” item to read in and recreate all the serialized beans. This provides a simple way to test the use of serialization with your beans.

- **Getting an introspection report on a beans :**
  If you want to see all the properties, methods, and events that the Beans introspector has found on a selected bean, you can use the BeanBox’s “report” menu item under “edit” menu. This generates a summary report to the standard output of the introspection information for the selected bean.

- **Adding your bean to the BeanBox :**
  When the BeanBox starts it loads all the JAR files that it finds in the “jars” directory. The BeanBox uses the manifest file in each JAR file to identify any bean classes in the JAR file, and adds those beans to the ToolBox palatte in the BeanBox. To add your bean to the BeanBox, you must wrap it up in a JAR file which contains a suitable manifest file describing the bean. Once you have a suitable JAR file, simply add it to the “jars” directory and restart the BeanBox, or load the JAR directly using the “LoadJar….”item in the “File” menu.

Example Beans:

As part of the BDK we include a number of example beans that demonstrate various aspect of the JavaBeans architecture. The following beans are all present in the default ToolBox palatte for the BeanBox. Their sources are in the "demo" directory.

- BridgeTester.
- CahngeReporter.
- ExplicitButton
- EventMonitor
- JellyBean
- Juggler
- Molecule
- OurButton
- TickTock
- Voter

Create and Configure an Instance of the Molecule Bean

Follow these steps to create and configure an instance of the **Molecule** Bean:

1. Position the cursor on the ToolBox entry labeled **Molecule** and click the left mouse button. You should see the cursor change to a cross.
2. Move the cursor to the BeanBox display area and click the left mouse button in approximately the area where you wish the Bean to be displayed. You should see a rectangular region appear which contains a 3-D display of a molecule. This area is surrounded by a hatched border, indicating that it is currently selected.
3. You can reposition the **Molecule** Bean by positioning the cursor over one of the hatched borders and dragging the Bean.
4. You can change the molecule that is displayed by changing the selection in the Properties window. Notice that the Bean display changes immediately when you change the selected molecule.

Create and Configure an Instance of the OurButton Bean

Follow these steps to create and configure an instance of the **OurButton** Bean and connect it to the **Molecule** Bean:

1. Position the cursor on the ToolBox entry labeled **OurButton** and click the left mouse button. You should see the cursor change to a cross.

2. Move the cursor to the BeanBox display area and click the left mouse button in approximately the area where you wish the Bean to be displayed. You should see a rectangular region appear that contains a button. This area is surrounded by a hatched border indicating that it is currently selected.
3. You may reposition the **OurButton** Bean by positioning the cursor over one of the hatched borders and dragging the Bean.
4. Go to the Properties window and change the label of the Bean to "Rotate X". The button appearance changes immediately when this property is changed.
5. Go to the menu bar of the BeanBox and select Edit | Events | action | actionPerformed. You should now see a line extending from the button to the cursor. Notice that one end of the line moves as the cursor moves. However, the other end of the line remains fixed at the button.
6. Move the cursor so that it is inside the **Molecule** Bean display area, and click the left mouse button. You should see the Event Target Dialog dialog box.
7. The dialog box allows you to choose a method that should be invoked when this button is clicked. Select the entry labeled "rotateOnX" and click the OK button. You should see a message box appear very briefly, stating that the tool is "Generating and compiling adaptor class." Test the application. Each time you press the button, the molecule should move a few degrees around one of its axes. Now create another instance of the **OurButton** Bean. Label it "Rotate Y" and map its action event to the "rotateY" method of the **Molecule** Bean. The steps to do this are very similar to those just described for the button labeled "Rotate X".

   Test the application by clicking these buttons and observing how the molecule moves.

Self Assessment Questions

9. The _____ is a very simple test container for JavaBeans.
10. A Bean obtains all the benefits of Java's "_____" paradigm.
11. The right-hand window of the beanbox contains _____showing the properties for the currently selected bean.

12.5 Summary

- Distributed applications are applications that execute across multiple host system.
- RMI allows objects in different JVMs belonging to different hosts to send and receive message.
- A Remote Registry Server is a service that runs on a server and helps objects on other hosts to remotely access its registered objects.
- Java's RMI approach makes use of stubs and skeletons.
- CORBA stands for Common Object Request Broker Architecture. RMI, discussed in previous section with RMI is that, it needs the 2 objects participating in communication be written in Java CORBA is a distributed computing technology where the participating objects need not only be written in Java.
- JavaBeans is the software component architecture for Java. It allows efficiently constructing applications by configuring and connecting components called Beans.
- The BeanBox comes up as three separate windows:
  - The middle window is the main beanbox composition window.
  - The left-hand window is the Toolbox palette displaying available beans that can be dropped onto the composition window.
  - The right-hand window is a PropertySheet showing the properties for the currently selected bean.

12.6 Terminal Questions

1. What are the different RMI terminologies?
2. What are the uses of CORBA?
3. Explain the working of BeanBox?

12.7 Answers

Self Assessment Questions

1. Remote Method Invocation.
2. Remote Registry Server.
3. RemoteInterface.
4. RemoteException.
5. TCP
6. Common Object Request Broker Architecture.
7. Interface Definition Language.

8. Object Management Group
9. BeanBox.
10. write-once, run-anywhere.
11. PropertySheet.

Terminal Questions

1. Local Objects, Remote Objects, Remote Registry Server.
   (Refer Section 12.2)
2. A key feature of CORBA is IDL, a language-neutral **Interface Definition Language.** Each language that supports CORBA has its own IDL mapping and as its name implies, Java IDL supports the mapping for Java. CORBA and the IDL mappings are the work of an industry consortium known as the OMG, or **Object Management Group. (**Refer Section 12.3)
3. The BeanBox comes up as three separate windows:

   The middle window is the main beanbox composition window.

   The left-hand window is the Toolbox palette displaying available beans that can be dropped onto the composition window.

   The right-hand window is a ***PropertySheet*** showing the properties for the currently selected bean. (Refer Section 12.4)