JavaBean

A JavaBean is a Java class that should follow the following conventions:

- It should have a no-arg constructor.
- It should be Serializable.
- It should provide methods to set and get the values of the properties, known as getter and setter methods.

Why use JavaBean?

According to Java white paper, it is a reusable software component. A bean encapsulates many objects into one object so that we can access this object from multiple places. Moreover, it provides easy maintenance.

Java bean is a reusable soft component that can be visually manipulated in builder tools. Write - Once - run- anywhere. (WORA)

Simple example of JavaBean class

```
//Employee.java
package mypack;
public class Employee implements java.io.Serializable{
private int id;
private String name;
public Employee(){}
public void setId(int id){this.id=id;}
public int getId(){return id;}
public void setName(String name){this.name=name;}
public String getName(){return name;}
}
```

How to access the JavaBean class?

To access the JavaBean class, we should use getter and setter methods.

```
package mypack;
public class Test{
public static void main(String args[]){
Employee e=new Employee();//object is created
e.setName("Arjun");//setting value to the object
System.out.println(e.getName());
}}
```

Note: There are two ways to provide values to the object. One way is by constructor and second is by setter method.

JavaBean Properties

A JavaBean property is a named feature that can be accessed by the user of the object. The feature can be of any Java data type, containing the classes that you define.

A JavaBean property may be read, write, read-only, or write-only. JavaBean features are accessed through two methods in the JavaBean's implementation class:

1. getPropertyName ()

For example, if the property name is firstName, the method name would be getFirstName() to read that property. This method is called the accessor.

2. setPropertyName ()

For example, if the property name is firstName, the method name would be setFirstName() to write that property. This method is called the mutator.

Advantages of JavaBean

The following are the advantages of JavaBean:/p>

- The JavaBean properties and methods can be exposed to another application.
- It provides an easiness to reuse the software components.

Disadvantages of JavaBean

The following are the disadvantages of JavaBean:

- JavaBeans are mutable. So, it can't take advantages of immutable objects.
- Creating the setter and getter method for each property separately may lead to the boilerplate code.

Jar files in Java

A JAR (Java Archive) is a package file format typically used to aggregate many Java class files and associated metadata and resources (text, images, etc.) into one file to distribute application software or libraries on the Java platform.
In simple words, a JAR file is a file that contains a compressed version of .class files, audio files, image files, or directories. We can imagine a .jar file as a zipped file(.zip) that is created by using WinZip software. Even, WinZip software can be used to extract the contents of a .jar . So you can use them for tasks such as lossless data compression, archiving, decompression, and archive unpacking.

Let us see how to create a .jar file and related commands which help us to work with .jar files

1.1 Create a JAR file

In order to create a .jar file, we can use ***jar cf command*** in the following ways as discussed below:

Syntax:

```
jar cf jarfilename inputfiles
```

Here, cf represents to create the file. For example , assuming our package pack is available in C:\directory , to convert it into a jar file into the pack.jar , we can give the command as:

```
C:\> jar cf pack.jar pack
```

1. 2 View a JAR file

Now, *pack.jar* file is created. In order to view a JAR file '.jar' files, we can use the command as:

Syntax:

```
jar tf jarfilename
```

Here, tf represents the table view of file contents. For example, to view the contents of our pack.jar file, we can give the command:

```
C:/> jar tf pack.jar
```

Now, the contents of pack.jar are displayed as follows:

```
META-INF/
META-INF/MANIFEST.MF
pack/
pack/class1.class
pack/class2.class
..
..
```

Here class1, class2, etc are the classes in the package pack. The first two entries represent that there is a manifest file created and added to pack.jar. The third entry represents the sub-directory with the name pack and the last two represent the files name in the directory pack.

> ***Note:*** *When we create .jar files, it automatically receives the default manifest file. There can be only one manifest file in an archive, and it always has the pathname.*

```
META-INF/MANIFEST.MF
```

This manifest file is useful to specify the information about other files which are packaged.

1.3 Extracting a JAR file

In order to extract the files from a .jar file, we can use the commands below listed:

```
jar xf jarfilename
```

Here, xf represents extract files from the jar files. For example, to extract the contents of our pack.jar file, we can write:

```
C:\> jar xf pack.jar
```

This will create the following directories in C:\

```
META-INF
```

In this directory, we can see class1.class and class2.class.

```
pack
```

1.4 Updating a JAR File

The Jar tool provides a 'u' option that you can use to update the contents of an existing JAR file by modifying its manifest or by adding files. The basic command for adding files has this format as shown below:

Syntax:

```
jar uf jar-file input-file(s)
```

Here 'uf' represents the updated jar file. For example, to update the contents of our pack.jar file, we can write:

```
C:\>jar uf pack.jar
```

1.5 Running a JAR file

In order to run an application packaged as a JAR file (requires the Main-class manifest header), the following command can be used as listed:

Syntax:

```
C:\>java -jar pack.jar
```

Entity Bean in EJB 3.x

Entity bean represents the persistent data stored in the database. It is a server-side component.

In EJB 2.x, there was two types of entity beans: **bean managed persistence** (BMP) and container managed persistence (CMP).

Since EJB 3.x, it is deprecated and replaced by JPA (Java Persistence API) that is covered in the hibernate tutorial.

In hibernate tutorial, there are given hibernate with annotation examples where we are using JPA annotations. The JPA with Hibernate is widely used today.

Enterprise Java Beans (EJB)

Enterprise Java Beans (EJB) is one of the several Java APIs for standard manufacture of enterprise software. EJB is a server-side software element that summarizes business logic of an application. Enterprise Java Beans web repository yields a runtime domain for web related software elements including computer reliability, Java Servlet Lifecycle (JSL) management, transaction procedure and other web services. The EJB enumeration is a subset of the Java EE enumeration.

The EJB enumeration was originally developed by IBM in 1997 and later adopted by Sun Microsystems in 1999 and enhanced under the Java Community Process.

The EJB enumeration aims to provide a standard way to implement the server-side business software typically found in enterprise applications. Such machine code addresses the same types of problems, and solutions to these problems are often repeatedly re-implemented by programmers. Enterprise Java Beans is assumed to manage such common concerns as endurance, transactional probity and security in a standard way that leaves programmers free to focus on the particular parts of the enterprise software at hand.

To run EJB application we need an application server (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc. It performs:

1. Life cycle management
2. Security
3. Transaction management
4. Object pooling

Types of Enterprise Java Beans

There are **three** types of EJB:

1. ***Session Bean:*** Session bean contains business logic that can be invoked by local, remote or webservice client. There are two types of session beans: (i) Stateful session bean and (ii) Stateless session bean.

- **(i) Stateful Session bean :**
  Stateful session bean performs business task with the help of a state. Stateful session bean can be used to access various method calls by storing the information in an instance variable. Some of the applications require information to be stored across separate method calls. In a shopping site, the items chosen by a customer must be stored as data is an example of stateful session bean.

- **(ii) Stateless Session bean :**
  Stateless session bean implement business logic without having a persistent storage mechanism, such as a state or database and can used shared data. Stateless session bean can be used in situations where information is not required to used across call methods.

2. ***Message Driven Bean:*** Like Session Bean, it contains the business logic but it is invoked by passing message.
3. ***Entity Bean:*** It summarizes the state that can be remained in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API). There are two types of entity bean:

- **(i) Bean Managed Persistence :**
  In a bean managed persistence type of entity bean, the programmer has to write the code for database calls. It persists across multiple sessions and multiple clients.

- **(ii) Container Managed Persistence :**
  Container managed persistence are enterprise bean that persists across database. In container managed persistence the container take care of database calls.

When to use Enterprise Java Beans

1.**Application needs Remote Access.** In other words, it is distributed.
2.**Application needs to be scalable.** EJB applications supports load balancing, clustering and fail-over.
3.**Application needs encapsulated business logic.** EJB application is differentiated from demonstration and persistent layer.

Advantages of Enterprise Java Beans

1. EJB repository yields system-level services to enterprise beans, the bean developer can focus on solving business problems. Rather than the bean developer, the EJB repository is responsible for

system-level services such as transaction management and security authorization.
2. The beans rather than the clients contain the application's business logic, the client developer can focus on the presentation of the client. The client developer does not have to code the pattern that execute business rules or access databases. Due to this the clients are thinner which is a benefit that is particularly important for clients that run on small devices.
3. Enterprise Java Beans are portable elements, the application assembler can build new applications from the beans that already exists.

Disadvantages of Enterprise Java Beans

1. Requires application server
2. Requires only java client. For other language client, you need to go for webservice.
3. Complex to understand and develop EJB applications.

Remote Method Invocation in Java

Remote Method Invocation (RMI) is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an object running in a JVM present on a computer (Client-side) can invoke methods on an object present in another JVM (Server-side). RMI creates a public remote server object that enables client and server-side communications through simple method calls on the server object.

Stub Object: The stub object on the client machine builds an information block and sends this information to the server.

The block consists of

- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

Skeleton Object: The skeleton object passes the request from the stub object to the remote object. It performs the following tasks

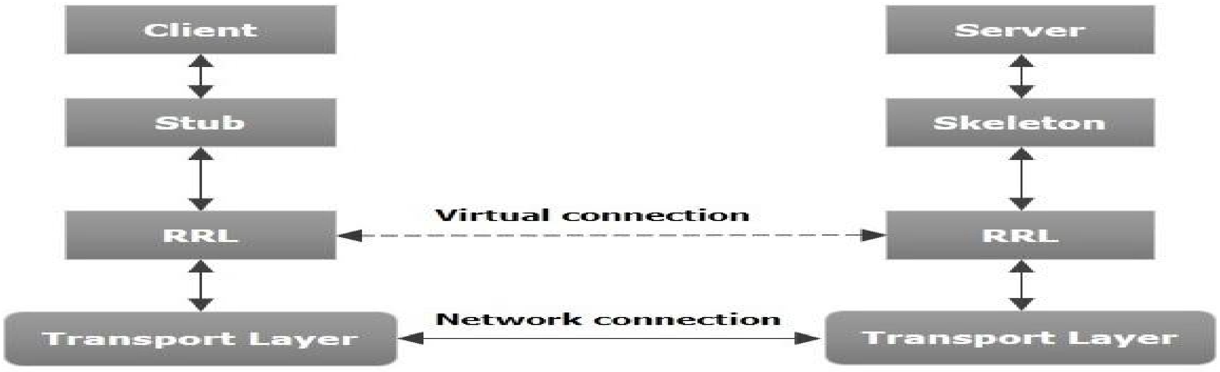
- It calls the desired method on the real object present on the server.
- It forwards the parameters received from the stub object to the method.

Architecture of an RMI Application(Sir wala Diagram)

In an RMI application, we write two programs, a **server program** (resides on the server) and a **client program** (resides on the client).

- Inside the server program, a remote object is created and reference of that object is made available for the client (using the registry).
- The client program requests the remote objects on the server and tries to invoke its methods.

The following diagram shows the architecture of an RMI application.

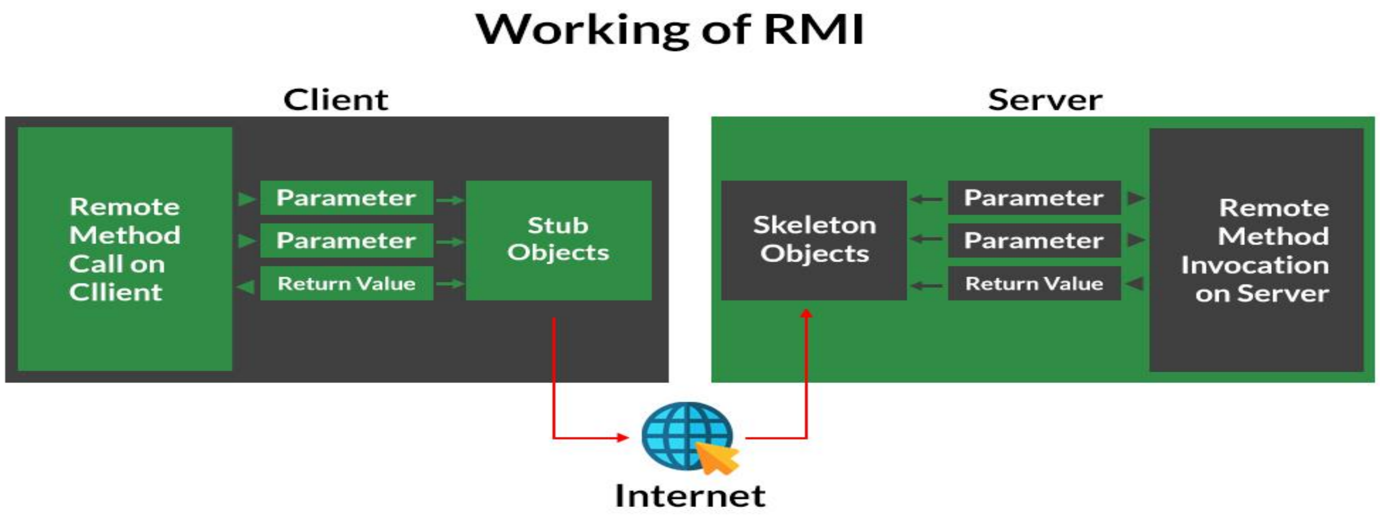


Let us now discuss the components of this architecture.

- **Transport Layer** – This layer connects the client and the server. It manages the existing connection and also sets up new connections.
- **RRL(Remote Reference Layer)** – It is the layer which manages the references made by the client to the remote object.

Working of RMI(Net wala)

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows:



These are the steps to be followed sequentially to implement Interface as defined below as follows:

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the implementation class using rmic (RMI compiler)
4. Start the rmiregistry
5. Create and execute the server application program
6. Create and execute the client application program.

Step 1: Defining the remote interface

The first thing to do is to create an interface that will provide the description of the methods that can be invoked by remote clients. This interface should extend the Remote interface and the method prototype within the interface should throw the RemoteException.

Example:

- Java

```
// Creating a Search interface
import java.rmi.*;
public interface Search extends Remote
{
    // Declaring the method prototype
    public String query(String search) throws RemoteException;
}
```

Step 2: Implementing the remote interface

The next step is to implement the remote interface. To implement the remote interface, the class should extend to UnicastRemoteObject class of java.rmi package. Also, a default constructor needs to be created to throw the java.rmi.RemoteException from its parent constructor in class.

- Java

```
// Java program to implement the Search interface
import java.rmi.*;
import java.rmi.server.*;
public class SearchQuery extends UnicastRemoteObject
                         implements Search
{
    // Default constructor to throw RemoteException
    // from its parent constructor
    SearchQuery() throws RemoteException
    {
        super();
    }

    // Implementation of the query interface
    public String query(String search)
                       throws RemoteException
    {
        String result;
        if (search.equals("Reflection in Java"))
            result = "Found";
        else
            result = "Not Found";

        return result;
    }
}
```

Step 3: Creating Stub and Skeleton objects from the implementation class using rmic

The rmic tool is used to invoke the rmi compiler that creates the Stub and Skeleton objects. Its prototype is rmic classname. For above program the following command need to be executed at the command prompt

rmic SearchQuery.

Step 4: Start the rmiregistry

Start the registry service by issuing the following command at the command prompt start rmiregistry

Step 5: Create and execute the server application program

The next step is to create the server application program and execute it on a separate command prompt.

- *The server program uses createRegistry method of LocateRegistry class to create rmiregistry within the server JVM with the port number passed as an argument.*
- *The rebind method of Naming class is used to bind the remote object to the new name.*

- Java

```
// Java program for server application
import java.rmi.*;
import java.rmi.registry.*;
public class SearchServer
{
    public static void main(String args[])
    {
        try
        {
            // Create an object of the interface
            // implementation class
            Search obj = new SearchQuery();

            // rmiregistry within the server JVM with
            // port number 1900
            LocateRegistry.createRegistry(1900);

            // Binds the remote object by the name
            // geeksforgeeks
            Naming.rebind("rmi://localhost:1900"+
                          "/geeksforgeeks",obj);
        }
        catch(Exception ae)
        {
            System.out.println(ae);
        }}}
```

Step 6: Create and execute the client application program

The last step is to create the client application program and execute it on a separate command prompt . The lookup method of the Naming class is used to get the reference of the Stub object.

- Java

```java
// Java program for client application
import java.rmi.*;
public class ClientRequest
{public static void main(String args[])
    { String answer,value="Reflection in Java";
        try
        {// lookup method to find reference of remote object
            Search access =
                (Search)Naming.lookup("rmi://localhost:1900"+ "/geeksforgeeks");
            answer = access.query(value);
            System.out.println("Article on " + value +
                            " " + answer+" at GeeksforGeeks");
        }
        catch(Exception ae)
        {
            System.out.println(ae);
        }}}
```

Note: The above client and server program is executed on the same machine so localhost is used. In order to access the remote object from another machine, localhost is to be replaced with the IP address where the remote object is present.

save the files respectively as per class name as

Search.java , SearchQuery.java , SearchServer.java & ClientRequest.java

Important Observations:

1. RMI is a pure java solution to Remote Procedure Calls (RPC) and is used to create the distributed applications in java.
2. Stub and Skeleton objects are used for communication between the client and server-side.

Remote Procedure Call (RPC)

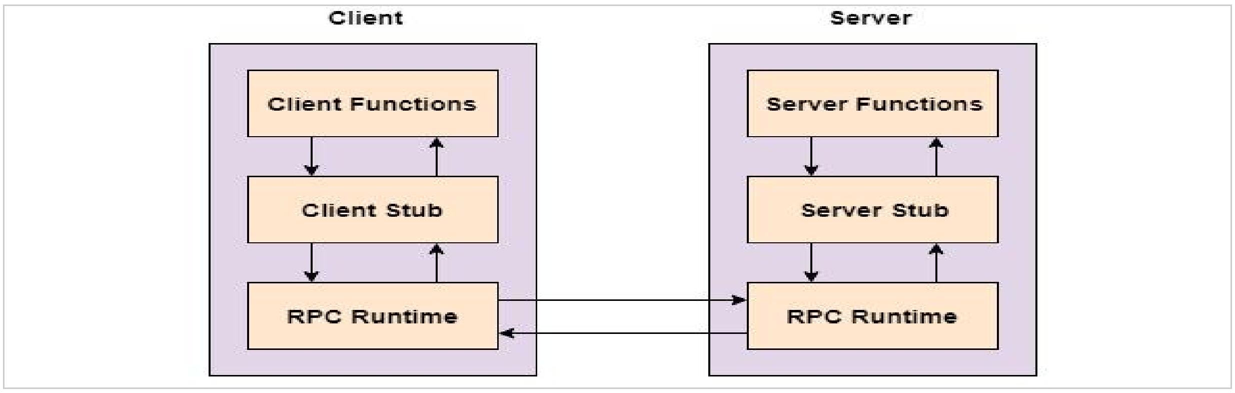
A remote procedure call is an interprocess communication technique that is used for client-server based applications. It is also known as a subroutine call or a function call.

A client has a request message that the RPC translates and sends to the server. This request may be a procedure or a function call to a remote server. When the server receives the request, it sends the required response back to the client. The client is blocked while the server is processing the call and only resumed execution after the server is finished.

The sequence of events in a remote procedure call are given as follows −

- The client stub is called by the client.
- The client stub makes a system call to send the message to the server and puts the parameters in the message.
- The message is sent from the client to the server by the client's operating system.
- The message is passed to the server stub by the server operating system.
- The parameters are removed from the message by the server stub.
- Then, the server procedure is called by the server stub.

A diagram that demonstrates this is as follows −



Advantages of Remote Procedure Call

Some of the advantages of RPC are as follows −

- Remote procedure calls support process oriented and thread oriented models.
- The internal message passing mechanism of RPC is hidden from the user.
- The effort to re-write and re-develop the code is minimum in remote procedure calls.
- Remote procedure calls can be used in distributed environment as well as the local environment.
- Many of the protocol layers are omitted by RPC to improve performance.

Disadvantages of Remote Procedure Call

Some of the disadvantages of RPC are as follows −

- The remote procedure call is a concept that can be implemented in different ways. It is not a standard.
- There is no flexibility in RPC for hardware architecture. It is only interaction based.
- There is an increase in costs because of remote procedure call.