Experiment 05

Learning Objectives: Implement Remote Method Invocation(RMI) for a system.

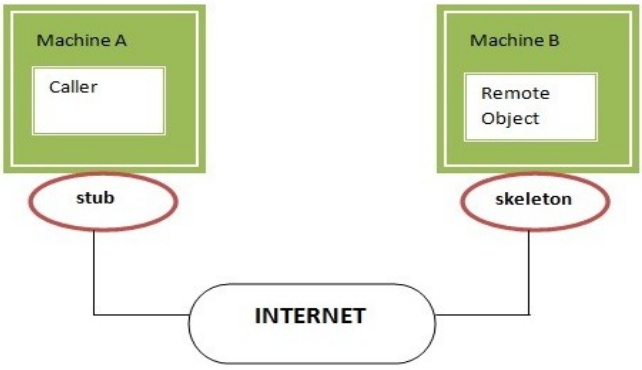
Tools : Software: Java Development kit, Eclipse or IntelliJ IDE.

Theory :

Remote Method Invocation is a Java-based API that allows an object residing in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM, typically on a different machine over a network. This makes it possible to create distributed applications, where objects can communicate with each other as if they were running in the same JVM, even though they may be on different physical machines. The RMI provides remote communication between the applications using two objects stub and skeleton.

stub: The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.



Skeleton:

The skeleton is an object, acts as a gateway for the server side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

In the Java 2 SDK, a stub protocol was introduced that eliminates the need for skeletons. If any application performs these tasks, it can be distributed application.

1. The application need to locate the remote method
2. It need to provide the communication with the remote objects, and
3. The application need to load the class definitions for the objects.

The RMI application have all these features, so it is called the distributed application.

Java RMI example for a marketplace Project

Step 1 : Define the Remote Interface

The interface declares the methods that can be called remotely.

remoteInterface.java

```java
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;

// Remote Interface for the Marketplace
public interface Marketplace extends Remote {
    List<String> listProducts() throws RemoteException;
    String purchaseProduct(String productName) throws RemoteException;
}
```

Step 2: Implement the Remote Interface (Server-Side Logic) : This class implements the remote interface and provides the actual logic for listing and purchasing products.

MarketplaceImpl.java 3

MarketplaceImpl.java > ...

```java
import java.rmi.server.UnicastRemoteObject;
import java.rmi.RemoteException;
import java.util.ArrayList;
import java.util.List;

// Implementation of Marketplace interface
public class MarketplaceImpl extends UnicastRemoteObject implements Marketplace {
    private List<String> products;

    protected MarketplaceImpl() throws RemoteException {
        super();
        products = new ArrayList<>();
        products.add(e:"Laptop");
        products.add(e:"Smartphone");
        products.add(e:"Headphones");
        products.add(e:"Camera");
    }

    @Override
    public List<String> listProducts() throws RemoteException {
        System.out.println(x:"Client requested product list.");
        return products;
    }

    @Override
    public String purchaseProduct(String productName) throws RemoteException {
        if (products.contains(productName)) {
            products.remove(productName);
            System.out.println("Client purchased: " + productName);
            return "You have successfully purchased " + productName;
        } else {
            return "Product not available!";
```

Step 3: Create the Server

The server creates an instance of the remote object and binds it to the RMI registry so that clients can locate and invoke methods on it.

MarketplaceImpl.java 3

MarketplaceImpl.java > ...

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class MarketplaceServer {
    Run | Debug
    public static void main(String[] args) {
        try {
            // Create an instance of the Marketplace service
            Marketplace marketplace = new MarketplaceImpl();

            // Bind the service to the RMI registry
            Registry registry = LocateRegistry.createRegistry(port:1099);
            registry.rebind(name:"MarketplaceService", marketplace);

            System.out.println(x:"Marketplace Server is ready.");
        } catch (Exception e) {
            System.out.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Step 4: Create the Client

The client looks up the remote object from the RMI registry and invokes methods on it.

MarketplaceServer.java 3

MarketplaceServer.java > ...

```java
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.util.List;

public class MarketplaceClient {
    Run | Debug
    public static void main(String[] args) {
        try {
            // Locate the RMI registry
            Registry registry = LocateRegistry.getRegistry(host:"localhost", port:1099);

            // Look up the remote service by name
            Marketplace marketplace = (Marketplace) registry.lookup(name:"MarketplaceService");

            // List available products
            List<String> products = marketplace.listProducts();
            System.out.println("Available Products: " + products);

            // Purchase a product
            String response = marketplace.purchaseProduct("Laptop");
            System.out.println(response);

            // List products again after the purchase
            products = marketplace.listProducts();
            System.out.println("Available Products after purchase: " + products);

        } catch (Exception e) {
            System.out.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
```

For running this rmi example,

1) compile all the java files - javac *.java

2)create stub and skeleton object by rmic tool - rmic AdderRemote

3)start rmi registry in one command prompt - rmiregistry 5000

4)start the server in another command prompt - java MyServer

5)start the client application in another command prompt - java MyClient

Result & Discussion

The RMI-based marketplace project successfully demonstrated remote interaction between a client and server. The server registered the marketplace service with the RMI registry, allowing the client to list products and make purchases remotely. The product inventory was updated on the server, reflecting real-time changes.

The project highlighted the ease of distributed computing with Java RMI, but also pointed out challenges like network dependencies and the need for security measures.

Future enhancements could include concurrency control, database integration, and user management for a more robust marketplace system.

Learning Outcomes: The student should have the ability :

- LO1: To understand the RMI architecture and its communication process.
- LO2: To learn how to implement remote interfaces and services.
- LO3: To apply and handle network and security aspects of RMI.

Course Outcomes

- CO1: Understand the RMI architecture and its communication process.
- CO2: Develop and learn how to implement remote interfaces and services.
- CO3: Application & Handling the network and security aspects of RMI.

Conclusion

This project showcases how Java RMI can be effectively used to build a simple distributed system. The separation of client and server logic, along with the abstraction provided by RMI, makes it easier to develop and manage applications that need remote communication. This approach lays the foundation for building more complex distributed systems, where multiple clients can interact with shared resources over a network.

For Faculty Use

Correction Parameters	**Formative Assessment [40%]**	**Timely completion of Practical [40%]**	**Attendance / Learning Attitude [20%]**	
Marks Obtained				