Java RMI (Remote Method Invocation) – A Detailed Explanation

HAMMOUDI Sarra

1 Introduction to Java RMI

Java RMI (**Remote Method Invocation**) is a mechanism that allows an object residing in one Java Virtual Machine (**JVM**) to invoke methods on an object located in another JVM. It enables **distributed computing** by allowing remote communication between Java applications.

2 Architecture of Java RMI

Java RMI follows a client-server architecture and consists of the following key components:

- **Client** – The application that calls remote methods.
- **Server** – The application that implements and provides remote methods.
- **Remote Interface** – Defines the methods that can be invoked remotely.
- **Stub (Client-Side Proxy)** – Acts as a local representative for the remote object.
- **Skeleton (Server-Side Proxy, Deprecated in Java 5)** – Processes client requests on the server side.
- **RMI Registry** – A naming service that allows clients to find remote objects.

3 Steps to Implement Java RMI

To implement Java RMI, follow these steps:

1. Define a **Remote Interface**.
2. Implement the **Remote Interface** (Server).
3. Create a **Client** to call the remote methods.

4. Start the **RMI Registry** and run the server.
5. Run the client.

4 Implementation of Java RMI

Now, let's look at a complete Java RMI example where the server provides a method to **add two numbers remotely**.

4.1 Step 1: Define the Remote Interface

The remote interface declares methods that clients can invoke remotely. It must extend `java.rmi.Remote` and declare methods that throw `RemoteException`.

```java
// Define a Remote Interface
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Calculator extends Remote {
    int add(int a, int b) throws RemoteException;
}
```

4.2 Step 2: Implement the Remote Interface (Server)

The server implements the remote interface and extends `UnicastRemoteObject`.

```java
// Implement the Remote Interface
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class CalculatorImpl extends UnicastRemoteObject
   implements Calculator {

    protected CalculatorImpl() throws RemoteException {
        super();
    }

    @Override
    public int add(int a, int b) throws RemoteException {
        return a + b;
    }
}
```

4.3 Step 3: Create the Server

The server registers the remote object with the **RMI Registry**.

```java
// RMI Server
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class RMIServer {
    public static void main(String[] args) {
        try {
            LocateRegistry.createRegistry(1099);
            CalculatorImpl calculator = new CalculatorImpl()
                ;
            Naming.rebind("CalculatorService", calculator);
            System.out.println("Server is running...");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

4.4 Step 4: Create the Client

The client looks up the remote object in the **RMI Registry** and invokes remote methods.

```java
// RMI Client
import java.rmi.Naming;

public class RMIClient {
    public static void main(String[] args) {
        try {
            Calculator calculator = (Calculator) Naming.
                lookup("rmi://localhost/CalculatorService");
            int result = calculator.add(5, 10);
            System.out.println("Result of 5 + 10: " + result
                );
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

5 Running the Java RMI Application

Follow these steps to run the RMI program:

1. **Compile all Java files:**

```
    javac *.java
```

2. **Start the RMI Registry:**

```
    rmiregistry
```

3. **Start the Server:**

```
    java RMIServer
```

4. **Run the Client:**

```
    java RMIClient
```

6 Advantages of Java RMI

- **Ease of Use** – RMI simplifies distributed object interaction.
- **Supports Object Passing** – Unlike traditional RPC, it allows sending objects.
- **Built-in Garbage Collection** – Java RMI handles remote object lifecycle management.

7 Limitations of Java RMI

- **Java-Only** – RMI is limited to Java applications.
- **Complex Setup** – Requires RMI Registry and correct network configurations.
- **Performance Overhead** – Slower than lightweight alternatives like gRPC.

8 Conclusion

Java RMI is a powerful mechanism for Java-based distributed computing. It enables remote method calls between Java applications while handling serialization and networking internally.