Remote Method Invocation (RMI)

RMI allows a Java object running on one machine to invoke methods on a Java object running on another machine. This simplifies distributed application development compared to explicit socket handling.

RMI handles the network communication infrastructure automatically.

RMI vs. Socket Programming:

Feature	Socket Programming	RMI
Infrastructure	Explicitly managed	Automatically provided
Communication	Byte streams	Remote method calls

Export to Sheets

RMI uses **STUB** and **SKELETON** components to manage remote communication.

Component	Location	Role
STUB	Client Machine	Represents remote object, handles outbound calls & inbound results.
SKELETON	Server Machine	Handles inbound calls & outbound results, invokes local method.

Export to Sheets

Reasoning: STUB acts as a client-side proxy; SKELETON acts as a server-side helper.

RMI also uses a **Registry** (`rmiregistry`) to locate remote objects by name.
`java.rmi.Naming` class interacts with the registry.

RMI Architecture Flow

Steps when a client calls a remote method:

1. Server creates **Remote Object**, registers in **RMI Registry**.
2. Client **lookup** in Registry gets **STUB** reference.
3. Client calls method on STUB.
4. STUB **serializes/marshals** call details, sends over network.
5. Network transfers request to server.
6. SKELETON receives, **deserializes/unmarshals** details, finds Remote Object.
7. SKELETON invokes method on Remote Object.
8. Remote Object executes, returns result.
9. SKELETON **serializes/marshals** result, sends over network.
10. Network transfers result to client.
11. STUB receives, **deserializes/unmarshals** result.
12. STUB returns result to client.

Steps to Prepare RMI Application

1. **Declare Remote Interface:**
    - Extend `java.rmi.Remote`.
    - Declare remote methods `throws java.rmi.RemoteException`.

    Java

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

public interface YourRemoteInterface extends Remote {
    public String remoteMethod(String arg) throws RemoteException;
    public int anotherRemoteMethod(int x, int y) throws
RemoteException;
}
```

    - *Reasoning:* `Remote` marks interface for RMI; `RemoteException` indicates network issues.
2. **Declare Implementation Class:**
    - Implement Remote Interface.
    - Extend `java.rmi.server.UnicastRemoteObject`.
    - Provide method implementations.
    - Public no-argument constructor `throws RemoteException`.

    Java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class YourRemoteInterfaceImpl extends UnicastRemoteObject
implements YourRemoteInterface {
    public YourRemoteInterfaceImpl() throws RemoteException {
        super();
    }

    @Override
    public String remoteMethod(String arg) throws RemoteException {
        return "Server received: " + arg;
    }

    @Override
    public int anotherRemoteMethod(int x, int int y) throws
RemoteException {
         return x + y;
    }
}
```

    - *Reasoning:* `UnicastRemoteObject` makes object remotely callable. No-arg constructor needed for RMI.
3. **Prepare Registry Program (Server Side):**
    - Create Remote object instance.
    - Bind object in RMI Registry using `Naming.bind(String name, Remote obj)`.

    Java

```
import java.rmi.Naming;
// Import YourRemoteInterfaceImpl

public class YourRegistryProgram {
    public static void main(String[] args) throws Exception {
        YourRemoteInterface remoteObject = new
YourRemoteInterfaceImpl();
        Naming.bind("yourRemoteObjectName", remoteObject);
        System.out.println("Object 'yourRemoteObjectName' bound in
Registry.");
    }
}
```

4. **Prepare Client Application:**
   - Look up remote object (STUB) using `Naming.lookup(String name).`
   - Cast result to Remote Interface.
   - Call remote methods on STUB.

   Java

```
import java.rmi.Naming;
// Import YourRemoteInterface

public class YourClientApp {
    public static void main(String[] args) throws Exception {
         YourRemoteInterface remoteObjectStub = (YourRemoteInterface)
Naming.lookup("yourRemoteObjectName");

         String result = remoteObjectStub.remoteMethod("Hello from
Client!");
         System.out.println("Result: " + result);

         int sum = remoteObjectStub.anotherRemoteMethod(5, 7);
         System.out.println("Sum: " + sum);
    }
}
```

5. **Execute RMI Application:**
   - Compile: `javac *.java`
   - Start Registry: `start rmiregistry` (in a separate terminal)
   - Run Registry Program: `start java YourRegistryProgram` (in another terminal)
   - Run Client: `java YourClientApp` (in a third terminal)

RMI Code Examples

Example 1: Simple "Hello" Service

- `HelloRemote.java`:

  Java

```
import java.rmi.*;

public interface HelloRemote extends Remote {
```

```
    public String sayHello(String name) throws RemoteException;
}
```

- HelloRemoteImpl.java:

Java

```
import java.rmi.*;
import java.rmi.server.*;

public class HelloRemoteImpl extends UnicastRemoteObject implements
HelloRemote {
    public HelloRemoteImpl() throws RemoteException {
        super();
    }

    @Override
    public String sayHello(String name) throws RemoteException {
        return "Hello " + name + ", Good Morning!";
    }
}
```

- HelloRegistry.java:

Java

```
import java.rmi.*;

public class HelloRegistry {
    public static void main(String[] args) throws Exception {
        HelloRemote hr = new HelloRemoteImpl();
        Naming.bind("hello", hr);
        System.out.println("'hello' object bound.");
    }
}
```

- Test.java:

Java

```
import java.rmi.*;

public class Test {
    public static void main(String[] args) throws Exception {
        HelloRemote hr = (HelloRemote) Naming.lookup("hello");
        String msg = hr.sayHello("Durga");
        System.out.println(msg);
    }
}
```

- **Execution & Output:**
- D:\FullstackJava830\JAVA830\RMI\app01>javac *.java
- D:\FullstackJava830\JAVA830\RMI\app01>start rmiregistry
- D:\FullstackJava830\JAVA830\RMI\app01>start java HelloRegistry
- D:\FullstackJava830\JAVA830\RMI\app01>java Test
- Hello Durga, Good Morning!

Example 2: Simple Calculator Service

- `CalculatorRemote.java`:

  Java

```
import java.rmi.*;

public interface CalculatorRemote extends Remote {
    public int add(int fval, int sval) throws RemoteException;
    public int sub(int fval, int sval) throws RemoteException;
    public int mul(int fval, int sval) throws RemoteException;
    public int div(int fval, int sval) throws RemoteException;
}
```

- `CalculatorRemoteImpl.java`:

  Java

```
import java.rmi.*;
import java.rmi.server.*;

public class CalculatorRemoteImpl extends UnicastRemoteObject
implements CalculatorRemote {
    public CalculatorRemoteImpl() throws RemoteException {
        super();
    }

    @Override
    public int add(int fval, int sval) throws RemoteException {
        return fval + sval;
    }

    @Override
    public int sub(int fval, int sval) throws RemoteException {
        return fval - sval;
    }

    @Override
    public int mul(int fval, int sval) throws RemoteException {
        return fval * sval;
    }

    @Override
    public int div(int fval, int sval) throws RemoteException {
         if (sval == 0) throw new RemoteException("Division by
zero");
        return fval / sval;
    }
}
```

- `CalculatorRegistry.java`:

  Java

```
import java.rmi.*;
```

```
public class CalculatorRegistry {
    public static void main(String[] args) throws Exception {
        CalculatorRemote cr = new CalculatorRemoteImpl();
        Naming.bind("cal", cr);
        System.out.println("'cal' object bound.");
    }
}
```

- Test.java:

Java

```
import java.rmi.*;

public class Test {
    public static void main(String[] args) throws Exception {
        CalculatorRemote cr = (CalculatorRemote)
Naming.lookup("cal");
        System.out.println("ADD    : " + cr.add(10, 3));
        System.out.println("SUB    : " + cr.sub(10, 3));
        System.out.println("MUL    : " + cr.mul(10, 3));
        System.out.println("DIV    : " + cr.div(10, 3));
    }
}
```

- **Execution & Output:**
- D:\FullstackJava830\JAVA830\RMI\app02>javac *.java
-
- D:\FullstackJava830\JAVA830\RMI\app02>start rmiregistry
-
- D:\FullstackJava830\JAVA830\RMI\app02>start java CalculatorRegistry
-
- D:\FullstackJava830\JAVA830\RMI\app02>java Test
- ADD   : 13
- SUB   : 7
- MUL   : 30
  DIV   : 3