# Experiment 05- *RMI Server*

**Learning Objective:** Create a RMI Server and implement for client and server side.

**Tools:** MS Word, VS Code.

**Language Used:** Java

**Theory:**

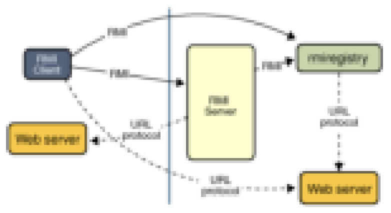
## An Overview of RMI Applications:

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a *distributed object application*.

Distributed object applications need to do the following:

- **Locate remote objects.** Applications can use various mechanisms to obtain references to remote objects. For example, an application can register its remote objects with RMI's simple naming facility, the RMI registry. Alternatively, an application can pass and return remote object references as part of other remote invocations.
- **Communicate with remote objects.** Details of communication between remote objects are handled by RMI. To the programmer, remote communication looks similar to regular Java method invocations.
- **Load class definitions for objects that are passed around.** Because RMI enables objects to be passed back and forth, it provides mechanisms for loading an object's class definitions as well as for transmitting an object's data.

The following illustration depicts an RMI distributed application that uses the RMI registry to obtain a reference to a remote object. The server calls the registry to associate (or bind) a name with a remote object. The client looks up the remote object by its name in the server's registry and then invokes a method on it. The illustration also shows that the RMI system uses an existing web server to load class definitions, from server to client and from client to server, for objects when needed.



## Creating Distributed Applications by Using RMI

Using RMI to develop a distributed application involves these general steps:

1. Designing and implementing the components of your distributed application.
2. Compiling sources.
3. Making classes network accessible.
4. Starting the application.

Designing and Implementing the Application Components

First, determine your application architecture, including which components are local objects and which components are remotely accessible. This step includes:

- **Defining the remote interfaces.** A remote interface specifies the methods that can be invoked remotely by a client. Clients program to remote interfaces, not to the implementation classes of those interfaces. The design of such interfaces includes the determination of the types of objects that will be used as the parameters and return values for these methods. If any of these interfaces or classes do not yet exist, you need to define them as well.
- **Implementing the remote objects.** Remote objects must implement one or more remote interfaces. The remote object class may include implementations of other interfaces and methods that are available only locally. If any local classes are to be used for parameters or return values of any of these methods, they must be implemented as well.
- **Implementing the clients.** Clients that use remote objects can be implemented at any time after the remote interfaces are defined, including after the remote objects have been deployed.

## Starting the Application

- Starting the application includes running the RMI remote object registry, the server, and the client.
- The rest of this section walks through the steps used to create a compute engine.

In a real-world scenario in which a service such as the compute engine is deployed, a developer would likely create a Java Archive (JAR) file that contains the Compute and Task interfaces for server classes to implement and client programs to use. Next, a developer, perhaps the same developer of the interface JAR file, would write an implementation of the Compute interface and deploy that service on a machine available to clients. Developers of client programs can use the Compute and the Task interfaces, contained in the JAR file, and independently develop a task and client program that uses a Compute service.

In this section, you learn how to set up the JAR file, server classes, and client classes. You will

see that the client's Pi class will be downloaded to the server at runtime. Also,
the Compute and Task interfaces will be downloaded from the server to the registry at runtime.

**Building a Server Class:**

Microsoft Windows:

```
cd c:\home\ann\src
javac -cp c:\home\ann\public_html\classes\compute.jar
 engine\ComputeEngine.java
```

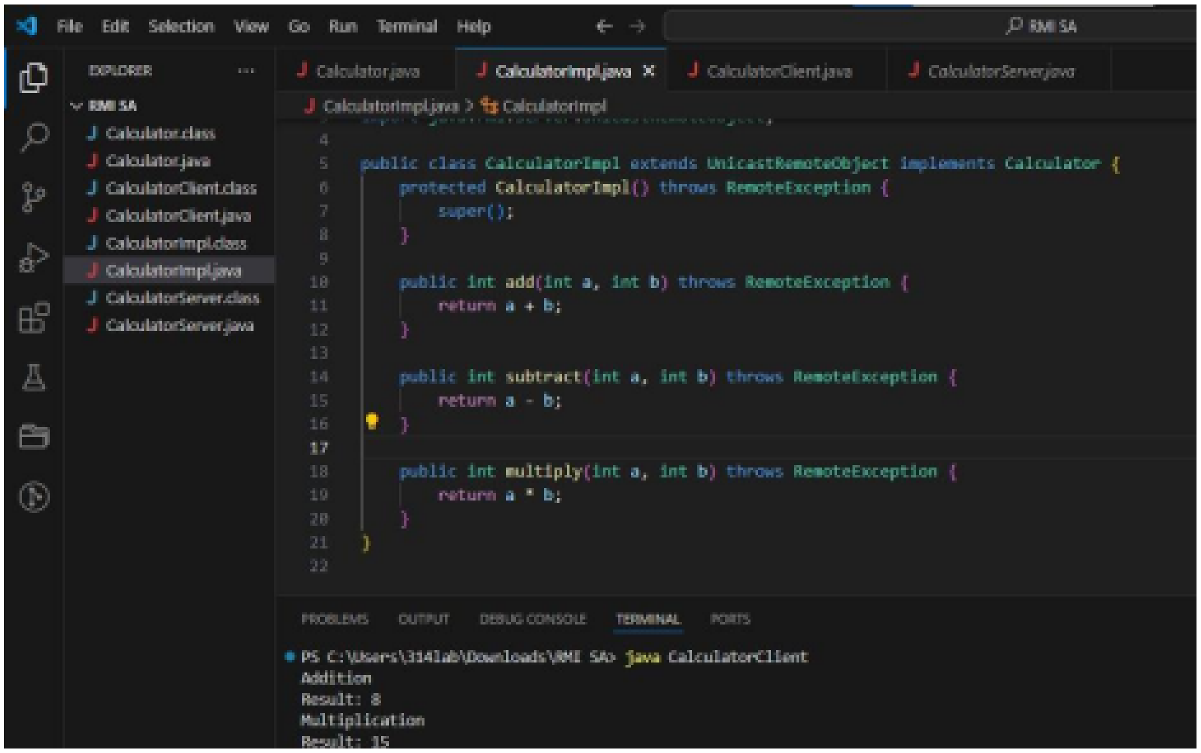
**Building a Client Class:**

Microsoft Windows:

```
cd c:\home\jones\src
javac -cp c:\home\jones\public_html\classes\compute.jar
 client\ComputePi.java client\Pi.java
mkdir c:\home\jones\public_html\classes\client
cp client\Pi.class
 c:\home\jones\public_html\classes\client
```

*Only the `Calc` class needs to be placed in the directory `public_html\classes\client` because only the `Calc` class needs to be available for downloading to the compute engine's Java virtual machine. Now, you can run the server and then the client.*

**Result and Discussion:**



```java
public class CalculatorImpl extends UnicastRemoteObject implements Calculator {
    protected CalculatorImpl() throws RemoteException {
        super();
    }

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

    public int subtract(int a, int b) throws RemoteException {
        return a - b;
    }

    public int multiply(int a, int b) throws RemoteException {
        return a * b;
    }
}
```

```
PS C:\Users\3141lab\Downloads\RMI SA> java CalculatorClient
Addition
Result: 8
Multiplication
Result: 15
```

**Learning Outcomes**: The student should have the ability to:

LO 1: Identify the importance of class diagrams.

LO 2: Draw class diagrams for a given scenario.

LO 3: Identify design patterns applicable to your project.

**Course Outcomes**: Upon completion of the course students will be able to understand RMI Server and their working.

**Conclusion**:

**For Faculty Use**

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