ASSIGNMENT NO. 1B)

Assignment No. 1 B)	To develop any distributed application through implementing client-server communication programs based on **Java RMI.**
Objective(s):	By the end of this assignment, the student will be able to implement any distributed applications based on RMI.
Tools	Eclipse, Java 8, rmiregistry

REMOTE METHOD INTERFACE (RMI)

- **Remote Method Invocation (RMI)** is an API which allows an object to invoke a method of an object that exists in another address space, which could be on the same machine or on a remote machine.
- Through RMI, 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.
- The communication between client and server is handled by using two int-ermediate objects:

  Stub object (on client side) and **Skeleton object** (on server side).
- **Stub Object:**
  The stub object on the client machine **builds an information block and sends this information to the server**.
- The block consists of:

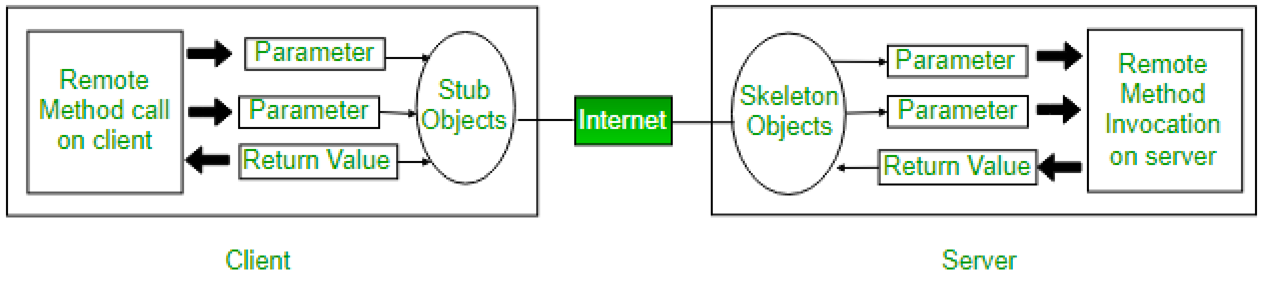
  An identifier of the remote object to be used

  Method name which is to be invoked

  Parameters to the remote JVM.

RMI IMPLEMENTATION

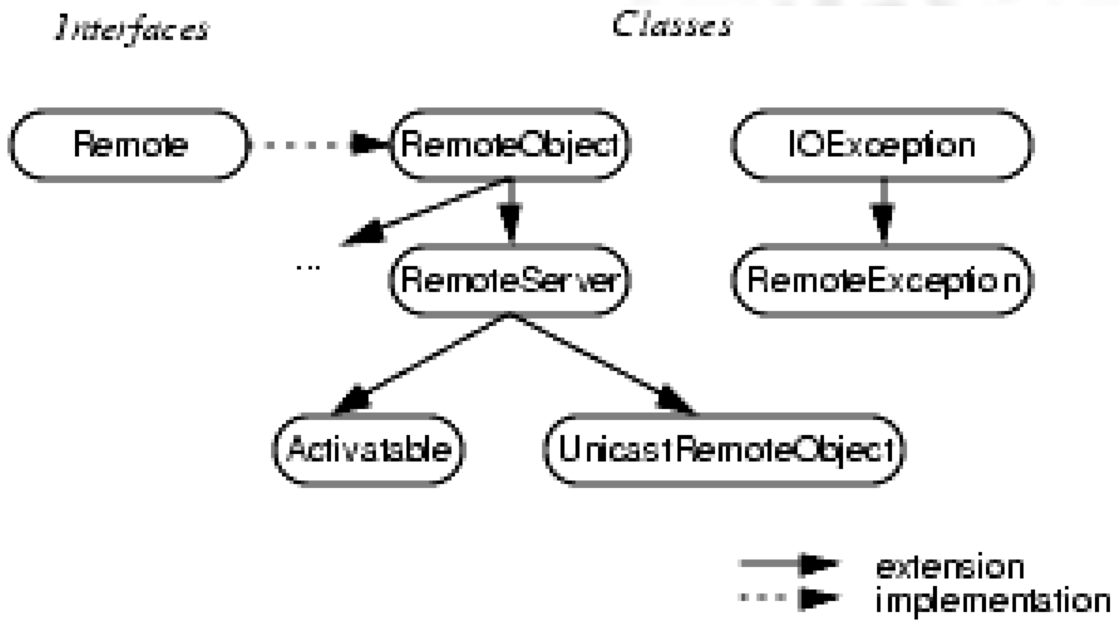
- **Skeleton Object**
- The skeleton object passes the request from the stub object to the remote object. It performs following tasks:
  - It calls the desired method on the real object present on the server.
  - It forwards the parameters received from the stub object to the method .



Working of RMI

REMOTE METHOD INTERFACE (RMI)

The interfaces and classes that are responsible for specifying the remote behavior of the RMI system are defined in the java.rmi package hierarchy. The following figure shows the relationship between several of these interfaces and classes:



REMOTE METHOD INTERFACE (RMI)

- `java.rmi.Remote` **Interface:**
- In RMI, a *remote* interface is an interface that declares a set of methods that may be invoked from a remote Java virtual machine.
- **The** `RemoteObject` **Class and its Subclasses**:
- RMI server functions are provided by `java.rmi.server.RemoteObject` and its subclasses, `java.rmi.server.RemoteServer` and `java.rmi.server.UnicastRemoteObject`.
- The class `java.rmi.server.RemoteObject` provides implementations for the `java.lang.Object` methods that are sensible for remote objects.
- The methods needed to create remote objects and make them available to remote clients are provided by the class `UnicastRemoteObject`.
- The `java.rmi.server.UnicastRemoteObject` class defines a singleton (unicast) remote object whose **references are valid only while the server process is alive**.

REMOTE METHOD INTERFACE (RMI)

- **Locating Remote Objects:** A simple **name server is provided for storing named references to remote objects.**
- A remote object reference can be stored using the URL-based methods of the class `java.rmi.Naming.`
- For a client to invoke a method on a remote object, that client must first obtain a reference to the object.
- The `java.rmi.Naming` class provides Uniform Resource Locator (URL) based methods to look up, bind, rebind, unbind, and list the name-object pairings maintained on a particular host and port.

- **Stub hides the serialization of parameters and the network-level communication** in order to present a simple invocation mechanism to the caller. In the remote JVM, each remote object may have a corresponding skeleton.
- **The skeleton** is responsible for dispatching the call to the actual remote object implementation.

RMI IMPLEMENTATION

Steps to implement RMI:

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

- Remote interfaces: Every remote object has a remote interface that specifies which of its methods can be invoked remotely.

RMI IMPLEMENTATION

- **Step 1: Defining the remote interface:**
- To create an interface which 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.

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

RMI IMPLEMENTATION

Step 2: Implementing the remote interface

- To implement the remote interface, the class should extend to `UnicastRemoteObject` class of `java.rmi` package.

```
// Java program to implement the Search interface
(SearchQuery.java)
import java.rmi.*;
import java.rmi.server.*;
public class SearchQuery extends UnicastRemoteObject
                                     implements Search
{      // 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; } }
```

RMI IMPLEMENTATION

- **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. The command need to be executed at the command prompt

```
# rmic SearchQuery
```

- **STEP 4: Start the rmiregistry**
- Start the registry service by issuing the command at the command prompt :

```
# start rmiregistry
```

RMI IMPLEMENTATION

- **STEP 5: Create and execute the server application program** 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 argument.
- The rebind method of Naming class is used to bind the remote object to the new name.

RMI IMPLEMENTATION

```
//program for server application (SearchServer.java)
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 cl9
                Naming.rebind("rmi://localhost:1900"+
                                          "/cl9",obj);
        }
        catch(Exception ae)
        {
                System.out.println(ae);
        }  }  }
```

SAVITRIBAI PHULE PUNE UNIVERSITY
सावित्रीबाई फुले पुणे विद्यापीठ
॥ यः क्रियावान् स पण्डितः ॥

RMI IMPLEMENTATION

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 Naming class is used to get the reference of the Stub object.

```
//program for client application  (ClientRequest.java)
import java.rmi.*;
public class ClientRequest
{
        public static void main(String args[])
        {       String answer,value= "RMI in Java";
                try
                {  // lookup method to find reference of remote object
                Search access =
(Search)Naming.lookup("rmi://localhost:1900/cl9");
                        answer = access.query(value);
                        System.out.println("Article on " + value +
                                                  " " + answer+" at
cl9");
                }
                catch(Exception ae)
                {
                        System.out.println(ae);
                }     }  }
```

RMI IMPLEMENTATION

Step 6: Compile and execute application programs:

```
#Javac SearchQuery.java
#rmic SearchQuery
#rmiregistry on console
```

On console-1:

Compile Server Application:

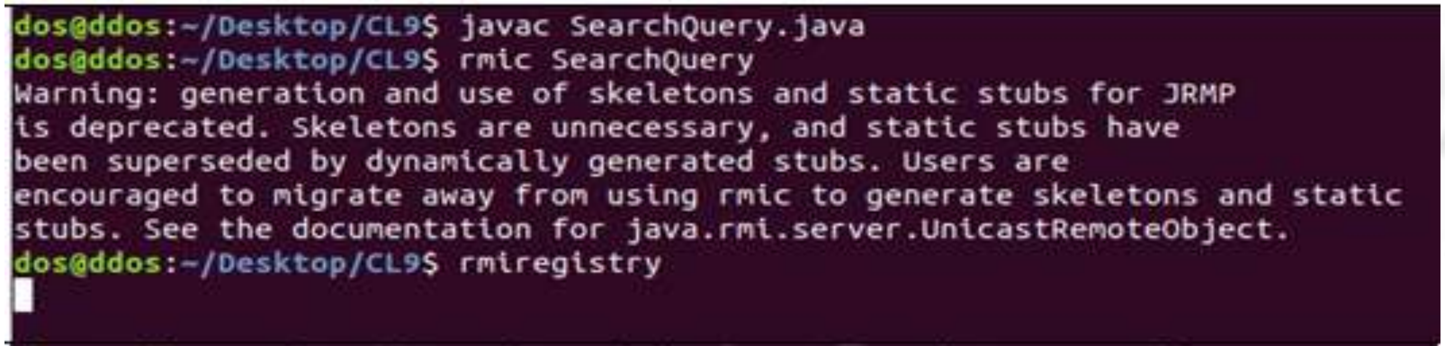
```
#javac SearchServer.java
#java SearchServer
```

On console-2:

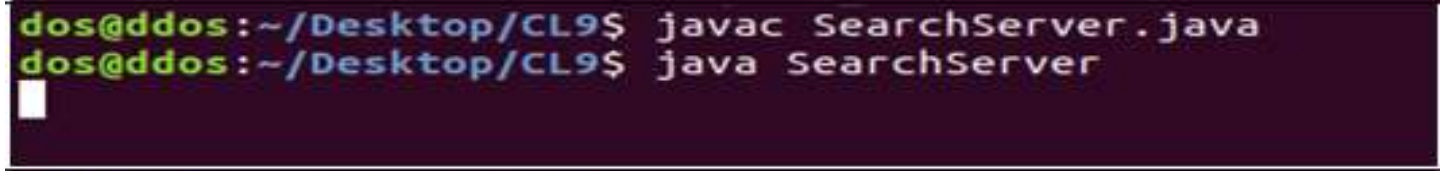
Compile ClientRequet Application:

```
#Javac ClientRequest.java
#java ClientRequest
```

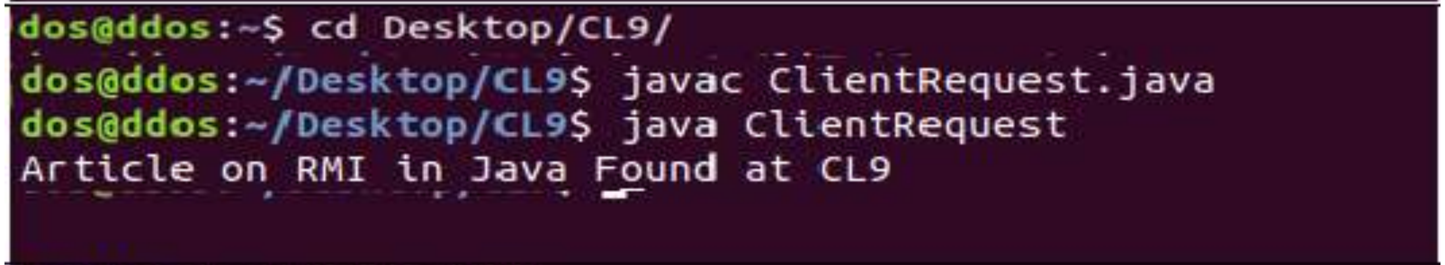
OUTPUT



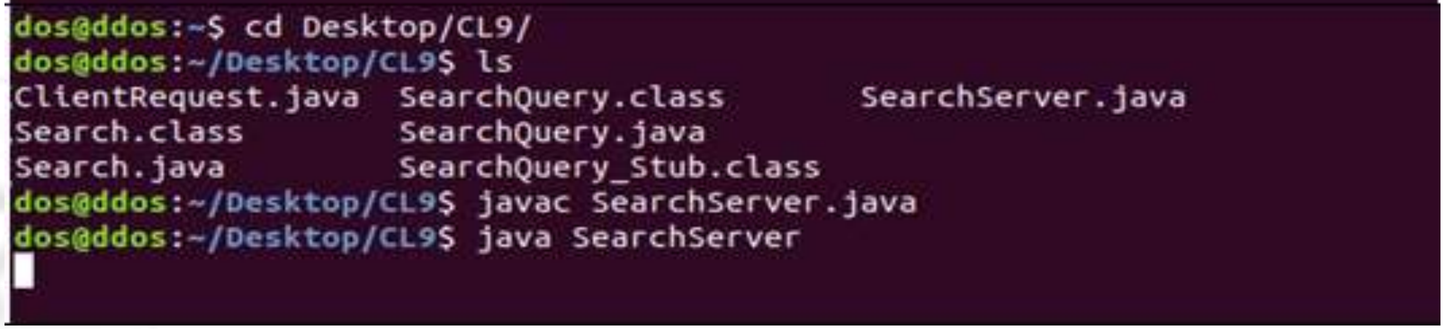
```
dos@ddos:~/Desktop/CL9$ javac SearchQuery.java
dos@ddos:~/Desktop/CL9$ rmic SearchQuery
Warning: generation and use of skeletons and static stubs for JRMP
is deprecated. Skeletons are unnecessary, and static stubs have
been superseded by dynamically generated stubs. Users are
encouraged to migrate away from using rmic to generate skeletons and static
stubs. See the documentation for java.rmi.server.UnicastRemoteObject.
dos@ddos:~/Desktop/CL9$ rmiregistry
█
```



```
dos@ddos:~/Desktop/CL9$ javac SearchServer.java
dos@ddos:~/Desktop/CL9$ java SearchServer
█
```



```
dos@ddos:~$ cd Desktop/CL9/
dos@ddos:~/Desktop/CL9$ javac ClientRequest.java
dos@ddos:~/Desktop/CL9$ java ClientRequest
Article on RMI in Java Found at CL9
```



```
dos@ddos:~$ cd Desktop/CL9/
dos@ddos:~/Desktop/CL9$ ls
ClientRequest.java  SearchQuery.class      SearchServer.java
Search.class        SearchQuery.java
Search.java         SearchQuery_Stub.class
dos@ddos:~/Desktop/CL9$ javac SearchServer.java
dos@ddos:~/Desktop/CL9$ java SearchServer
█
```

SAVITRIBAI PHULE PUNE UNIVERSITY
सावित्रीबाई फुले पुणे विद्यापीठ
॥ यः क्रियावान् स पण्डितः ॥

REFERENCES

- https://www.geeksforgeeks.org/remote-method-invocation-in-java/
- https://docs.oracle.com/javase/7/docs/platform/rmi/spec/rmiTOC.html