EXPERIMENT NO: 1

AIM: - Implement a client/server using RPC/RMI

THEORY:

Remote Method Invocation (RMI) is an API that allows an object to invoke a method on an object that exists in another address space, which could be on the same machine or on a remote machine. Through RMI, an 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.

Stub Object: The stub object on the client machine builds an information block and sends this information to the server. The block consists of

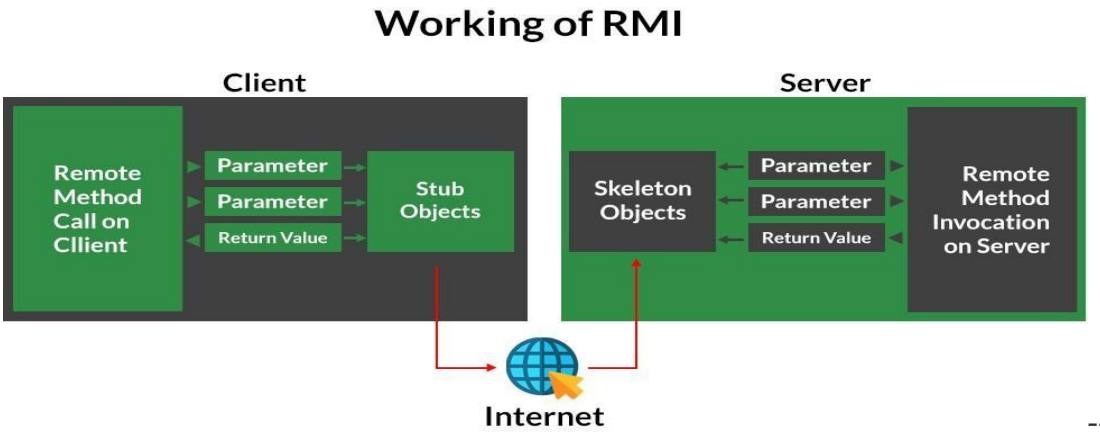
- An identifier of the remote object to be used
- Method name which is to be invoked
- Parameters to the remote JVM

Skeleton Object: The skeleton object passes the request from the stub object to the remote object. It performs the 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

The communication between client and server is handled by using two intermediate objects: Stub object (on client side) and Skeleton object (on server-side) as also can be depicted from below media as follows:



--

These are the steps to be followed sequentially to implement Interface as defined below as follows:

1. Defining a remote interface
2. Implementing the remote interface
3. Creating Stub and Skeleton objects from the implementation class using rmic (RMI compiler)
4. Start the RMI registry
5. Create and execute the server application program
6. Create and execute the client application program.

PROGRAM:

Search.java

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

SearchQuery.java

```
// Java program to implement the Search interface
import java.rmi.*; import java.rmi.server.*;
public class SearchQuery extends UnicastRemoteObject implements
Search
{
        // Default constructor to throw RemoteException
        // from its parent constructor
        SearchQuery() throws RemoteException
        { super();
        }

        // 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;
        }
}
```

SearchServer.java

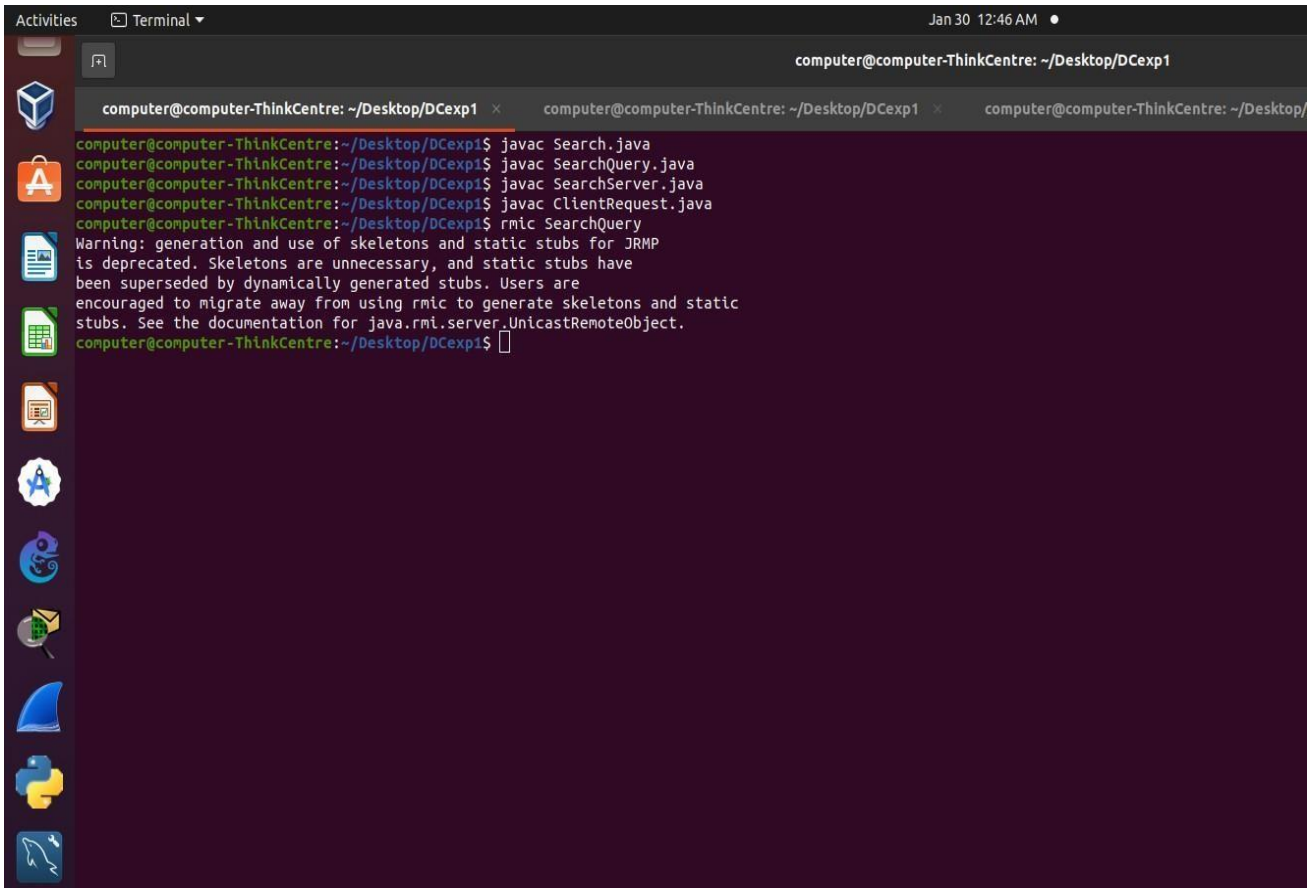
```
// Java program for server application
import java.rmi.*; import
java.rmi.registry.*; public class
SearchServer
{ public static void main(String args[])
        { try
                {
                        // Create an object of the interface
// implementation class
                        Search obj = new SearchQuery();

                        // rmiregistry within the server JVM with
// port number 1900
                        LocateRegistry.createRegistry(1900);

                        // Binds the remote object by the name
// geeksforgeeks
                        Naming.rebind("rmi://localhost:1900"+
"/geeksforgeeks",obj);
                }
                catch(Exception ae)
                {
System.out.println(ae);
                }
        }
}
```

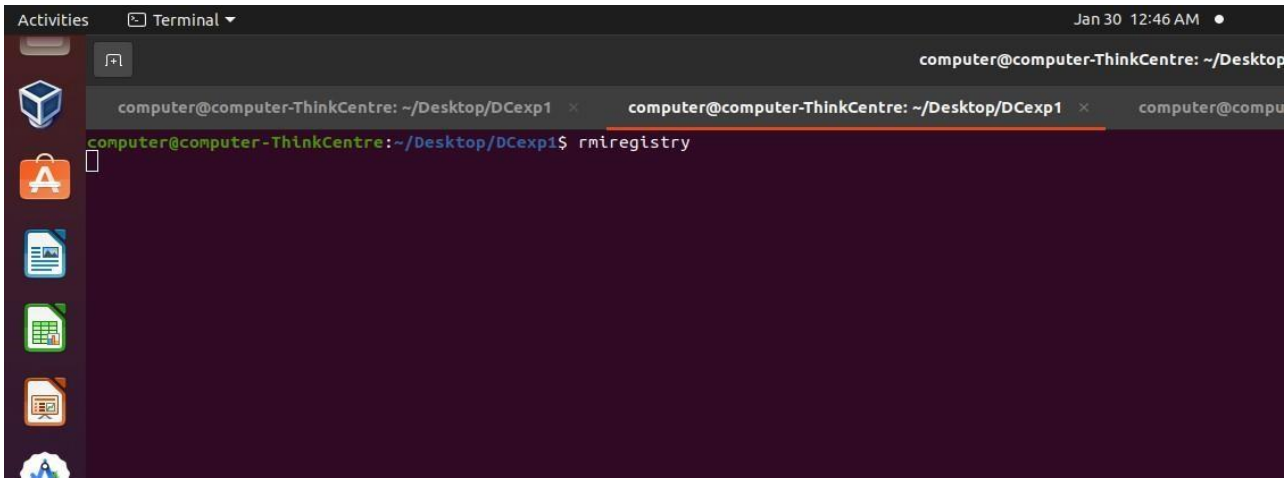
ClientRequest.java

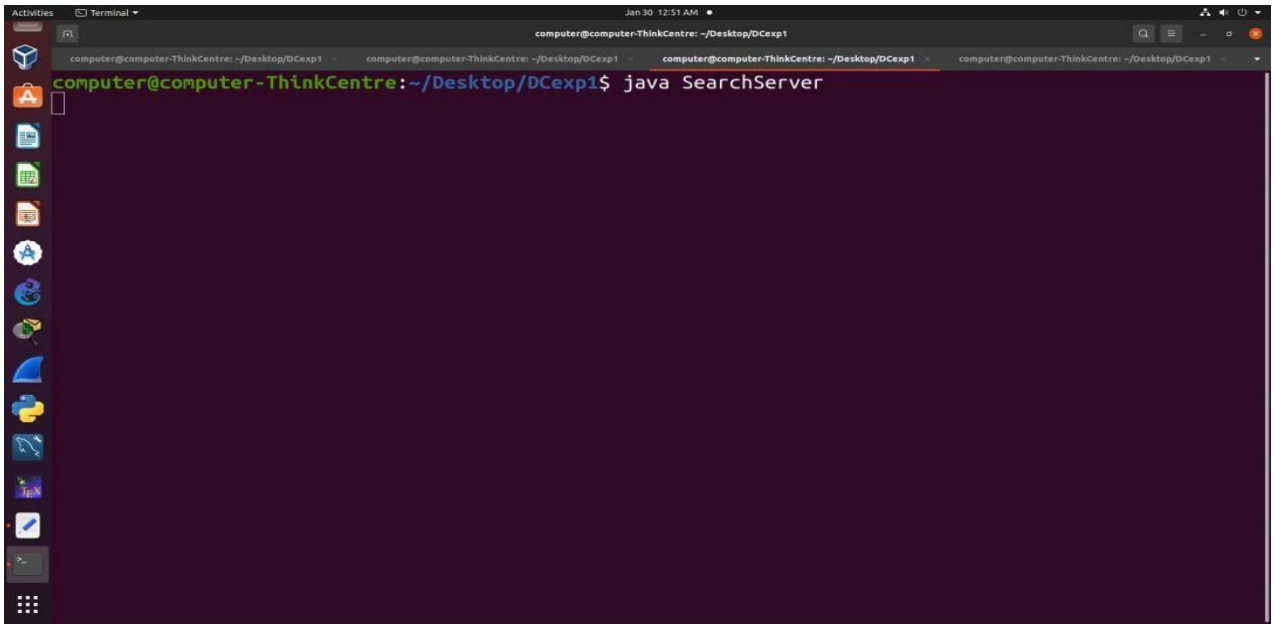
```
// Java program for client application
import java.rmi.*; public class
ClientRequest
{ public static void main(String args[])
        {
                String answer,value="Reflection in Java";
                try {
                        // lookup method to find reference of remote object Search
                        access =
                                (Search)Naming.lookup("rmi://localhost:1900"+
"/geeksforgeeks");
                        answer = access.query(value);
```

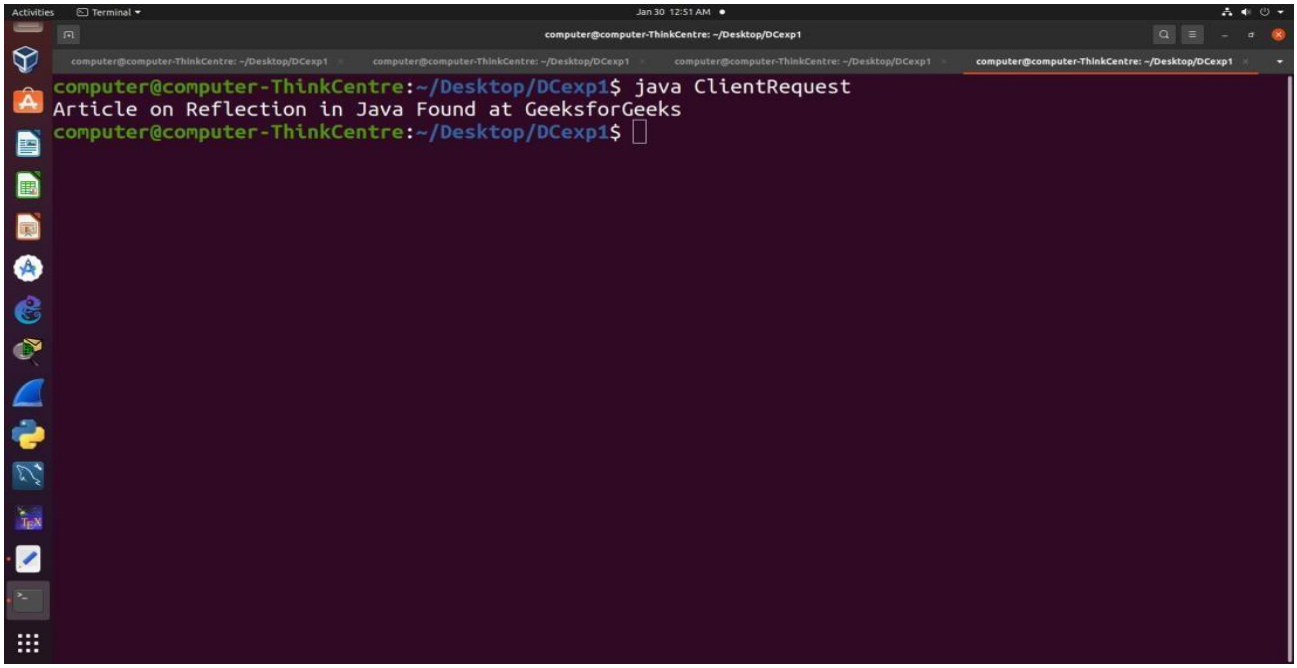
```
System.out.println("Article on " + value +"
"+answer+" at GeeksforGeeks");
                }
                catch(Exception ae)
                {
                        System.out.println(ae);
                }
        }
}
```

OUTPUT:









CONCLUSION: Therefore, this is how Client/server using RPC/RMI is implemented.