Name of Student: Pushkar Sane		
Roll Number: 45	**Lab Assignment Number: 3**	
Title of Lab Assignment: Remote Method Invocation		
DOP: 02-09-2024	**DOS: 05-09-2024**	
CO Mapped:	**PO Mapped:**	**Signature:**

Practical No. 3

Aim:

1. To find date and time using Remote Method Invocation (Use stubs and skeletons).
2. Implementation of equation solver using Remote Method Invocation (RMI).

Description:

1. Create a client and server application where the client invokes methods via an interface. These methods are implemented on the server side. Create the necessary STUBS and SKELETONS. Retrieve time and date function from server to client. This program should display server date and time. (Use the concept of JDBC and RMI for accessing multiple data access objects).
2. Equation solver. The client should provide an equation to the server through an interface. The server will solve the expression given by the client.
   (a-b)2 = a2 –2ab + b2;
   If a = 5 and b = 2 then return value = 52 – 2.5.2 + 22 = 9

Theory:

1. **RMI (Remote Method Invocation)**
   Remote Method Invocation (RMI) is a Java API that allows an object running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM. This is often referred to as remote communication. RMI enables the construction of distributed applications, where different parts of the application can be located on different machines.
   RMI abstracts much of the complexity of network programming, allowing developers to focus on the application logic rather than the details of network communication.

2. **Stub**
   A stub is a client-side proxy that represents the remote object. When a client wants to invoke a method on a remote object, it actually calls the corresponding method on the stub. The stub then handles the communication over the network, forwarding the call to the actual remote object on the server.

Key Responsibilities of the Stub:

- Marshalling: The stub converts (serializes) the method arguments into a format that can be transmitted over the network.
- Sending the Request: The stub sends the method invocation request to the server via the network.
- Receiving the Response: After the server processes the request and sends back a response, the stub receives it.
- Unmarshalling: The stub converts (deserializes) the response back into a form that the client can use.

3. Skeleton

The skeleton is the server-side counterpart to the stub. In older versions of Java (pre-Java 1.2), the skeleton was responsible for receiving method invocation requests from the stub, unmarshalling the parameters, invoking the appropriate method on the remote object, and sending the result back to the stub.

Key Responsibilities of the Skeleton:

- Receiving the Request: The skeleton listens for incoming method invocation requests from the client.
- Unmarshalling the Request: The skeleton deserializes the method arguments sent by the stub.
- Invoking the Method: The skeleton invokes the corresponding method on the actual remote object on the server.
- Marshalling the Response: The skeleton serializes the result and sends it back to the stub.

Code:

1. **Client and server application where the client invokes methods via an interface.**

   DateTimeService.java

```
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface DateTimeService extends Remote {
        String getDateTime() throws RemoteException;
}
```

DateTimeServiceImpl.java

```
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
import java.text.SimpleDateFormat;
import java.util.Date;
public class DateTimeServiceImpl extends UnicastRemoteObject implements DateTimeService {
	protected DateTimeServiceImpl() throws RemoteException {
		super();
	}

	@Override
	public String getDateTime() throws RemoteException {
		SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
		return sdf.format(new Date());
	}
}
```

DateTimeServer.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;

public class DateTimeServer {
	public static void main(String[] args) {
		try {
			// Create and export a remote object
			DateTimeServiceImpl obj = new DateTimeServiceImpl();

			// Create and start the RMI registry on port 1099
			LocateRegistry.createRegistry(1099);

			// Bind the remote object in the registry
			Naming.rebind("DateTimeService", obj);
```

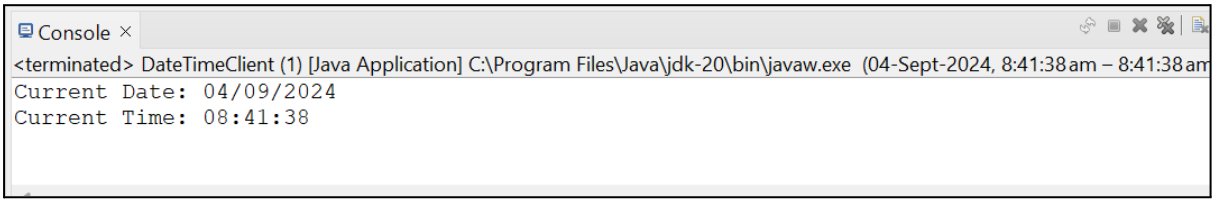
```
                System.out.println("DateTimeService bound to registry");
            } catch (Exception e) {
                System.err.println("Server exception: " + e.toString());
                e.printStackTrace();
            }
        }
}
```

DateTimeClient.java

```
import java.rmi.Naming;
import java.rmi.RemoteException;

public class DateTimeClient {
    public static void main(String[] args) {
        try {
            // Lookup the remote object
            DateTimeService service = (DateTimeService) Naming.lookup("rmi://localhost/DateTimeService");
            // Invoke the remote method
            String dateTime = service.getDateTime();
            System.out.println("Current Date and Time from Server: " + dateTime);
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Output:



Code: Implementation of equation solver using Remote Method Invocation (RMI).

EquationSolverInterface.java:

```
package server;
import java.rmi.Remote;
import java.rmi.RemoteException;
public interface EquationSolverInterface extends Remote {
  int solveEquation(String equation, int a, int b, int c) throws RemoteException;
}
```

EquationSolverImpl.java:

```
package server;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;
public class EquationSolverImpl extends UnicastRemoteObject implements EquationSolverInterface {
  protected EquationSolverImpl() throws RemoteException {
    super();
  }
  @Override
  public int solveEquation(String equation, int a, int b, int c) throws RemoteException {
    int result = 0;
    switch (equation) {
      case "(a+b)^2":
        result = (int) (Math.pow(a + b, 2));
        break;
      case "(a-b)^2":
        result = (int) (Math.pow(a, 2) - 2 * a * b + Math.pow(b, 2));
        break;
      case "(a+b+c)^2":
        result = (int) (Math.pow(a + b + c, 2));
```

```
            break;
        case "(a+b)^3":
            result = (int) (Math.pow(a + b, 3));
            break;
        default:
            throw new RemoteException("Equation not supported.");
    }
    return result;
  }
}
```

EquationSolverServer.java:

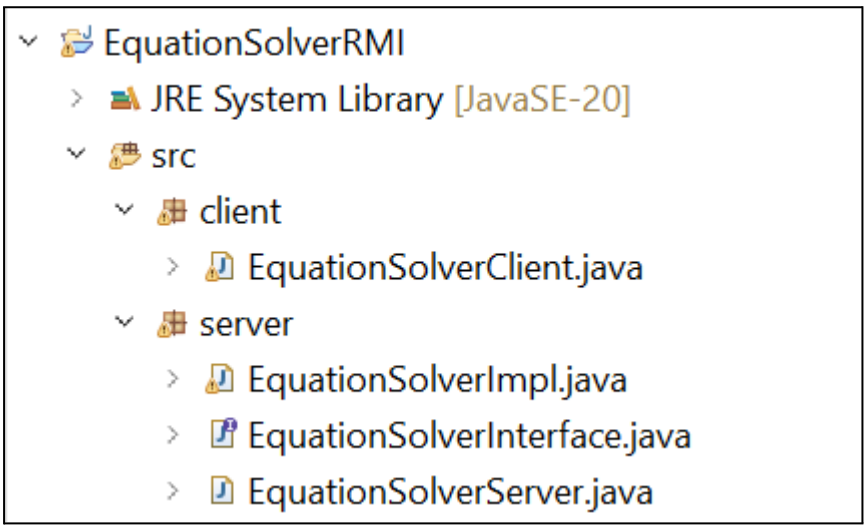
```
package server;
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;
public class EquationSolverServer {
  public static void main(String[] args) {
    try {
      LocateRegistry.createRegistry(1099);
      EquationSolverImpl obj = new EquationSolverImpl();
      Naming.rebind("rmi://localhost/EquationSolverServer", obj);
      System.out.println("Server started, waiting for client to enter equations...");
    } catch (Exception e) {
      e.printStackTrace();
    }
  }
}
```

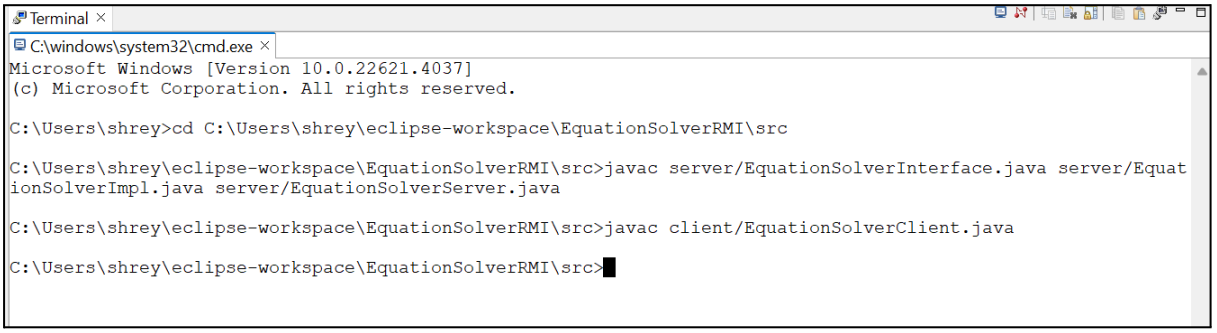
EquationSolverClient.java:

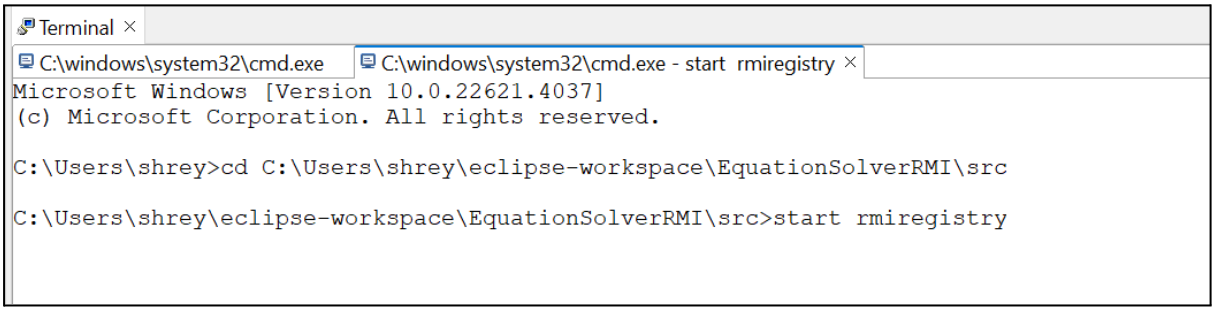
```
package client;
import server.EquationSolverInterface;
import java.rmi.Naming;
import java.util.Scanner;
public class EquationSolverClient {
  public static void main(String[] args) {
    try {
```

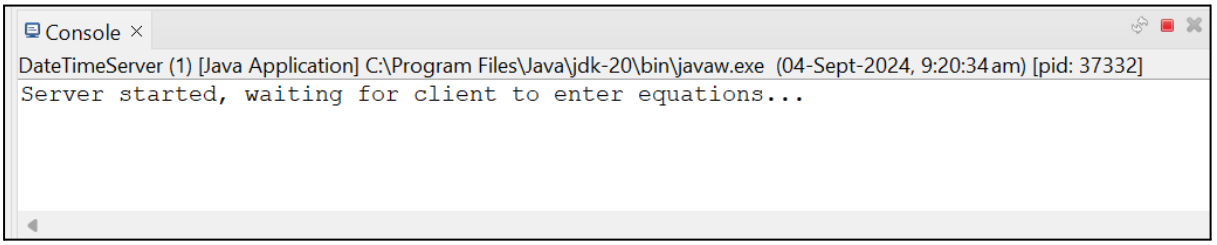
```
                EquationSolverInterface obj = (EquationSolverInterface)
Naming.lookup("rmi://localhost/EquationSolverServer");
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter equation: ");
        String equation = sc.nextLine();
        System.out.print("Enter value for a: ");
        int a = sc.nextInt();
        System.out.print("Enter value for b: ");
        int b = sc.nextInt();
        int c = 0;
        if (equation.contains("c")) {
            System.out.print("Enter value for c: ");
            c = sc.nextInt();
        }
        int result = obj.solveEquation(equation, a, b, c);
        System.out.println("Answer: " + result);
    } catch (Exception e) {
        e.printStackTrace();
    }
  }
}
```

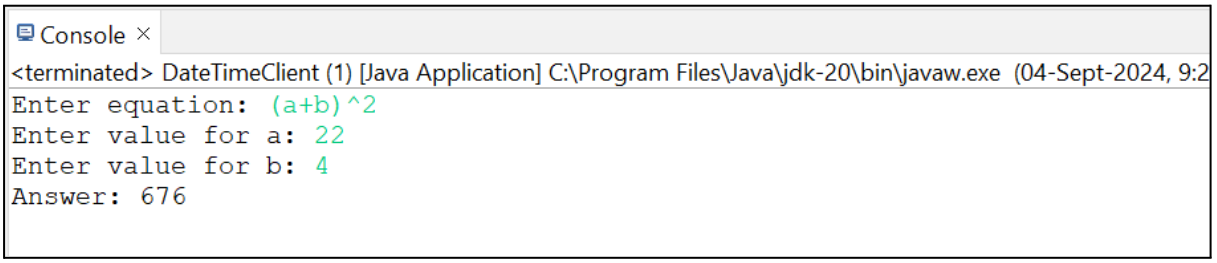
Output:

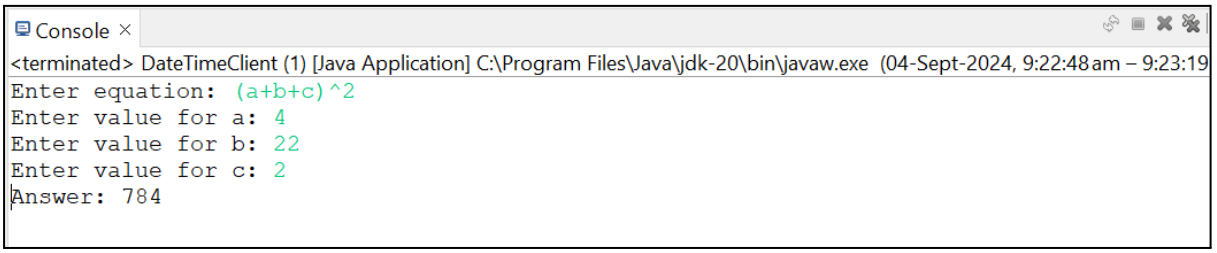












Conclusion: Implemented finding date and time and solving equations using Remote Method Invocation successfully.