| **Name of Student:** Ajay Karthikesan | | | |
| --- | --- | --- | --- |
| **Roll Number:** 57 | | **Practical Number:** 4 | |
| **Aim of Practical:**  To find date and time using Remote Method Invocation (Use stubs and skeletons). | | | |
| **DOP:** 14.9.23 | | **DOS:** 10.10.23 | |
| **CO Mapped:**  CO1 | **PO Mapped:**  - | **Faculty Signature:** | **Marks:** |

## 

## Practical No. 4

**Aim:** To find date and time using Remote Method Invocation (Use stubs and skeletons).

**Theory:**

RMI (Remote Method Invocation):

Definition: RMI is a Java API that provides a mechanism for invoking methods on objects that are located in remote Java virtual machines (JVMs).

Purpose: RMI enables distributed computing by allowing Java objects in one JVM to invoke methods on Java objects located in other JVMs, making it possible to build distributed and networked applications.

Usage: To use RMI, you define remote interfaces that extend the java.rmi.Remote interface and annotate them with java.rmi.RemoteException. Implementations of these remote interfaces are provided on the server side, and client programs can invoke methods on remote objects as if they were local.

Stub:

Definition: A stub is a client-side proxy for a remote object. It implements the same set of remote interfaces as the remote object it represents.

Purpose: Stubs act as intermediaries between the client and server, allowing clients to invoke methods on remote objects without needing to know the details of remote communication.

Generation: Stubs are generated automatically by the RMI compiler (rmic) based on the remote interfaces. They encapsulate the details of the communication, including object serialization and network transport.

Usage: In a client program, you interact with the stub as if it were a local object. The stub handles the serialization of method calls and parameters, sends them over the network to the server, and receives and deserializes the results.

Skeleton:

Definition: A skeleton is a server-side object that receives incoming method calls from clients, dispatches them to the appropriate remote object, and returns the results to the clients.

Purpose: Skeletons act as intermediaries between the network and the actual remote objects. They handle the deserialization of incoming method calls and parameters and forward the calls to the appropriate remote objects for execution.

Generation: Skeletons are generated automatically by the RMI runtime system. However, in modern RMI, they are mostly transparent, and developers don't need to write or deal with skeleton classes directly.

Usage: Skeletons are part of the RMI runtime system and are responsible for unmarshalling and routing incoming method calls to the correct remote objects. Developers primarily focus on implementing remote interfaces and server-side logic.

**Code:**

File: DateTimeService.java

package vesit.ajayk57.practical4;

import java.rmi.Remote;

import java.rmi.RemoteException;

import java.util.Date;

public interface DateTimeService extends Remote {

Date getDate() throws RemoteException;

String getTime() throws RemoteException;

}

File: DateTimeServerImpl.java

package vesit.ajayk57.practical4;

import java.rmi.RemoteException;

import java.rmi.server.UnicastRemoteObject;

import java.util.Date;

public class DateTimeServiceImpl extends UnicastRemoteObject implements DateTimeService {

public DateTimeServiceImpl() throws RemoteException {

super();

}

@Override

public Date getDate() throws RemoteException {

return new Date(); // Return the current date.

}

@Override

public String getTime() throws RemoteException {

return new Date().toString(); // Return the current time as a string.

}

}

File: DateServer.java

package vesit.ajayk57.practical4;

import java.rmi.Naming;

import java.rmi.registry.LocateRegistry;

public class DateTimeServer {

public static void main(String[] args) {

try {

DateTimeService dateTimeService = new DateTimeServiceImpl();

// Create and start the RMI registry on port 1099 (default).

LocateRegistry.createRegistry(1099);

// Bind the remote object to a name in the RMI registry.

Naming.rebind("DateTimeService", dateTimeService);

System.out.println("Server is running and ready to serve...");

} catch (Exception e) {

e.printStackTrace();

}

}

}

File: DateClient.java

package vesit.ajayk57.practical4;

import java.rmi.Naming;

public class DateTimeClient {

public static void main(String[] args) {

try {

// Lookup the remote object in the RMI registry using its name.

DateTimeService dateTimeService = (DateTimeService) Naming.lookup("rmi://localhost/DateTimeService");

// Call the remote methods to get date and time.

System.out.println("Server Date: " + dateTimeService.getDate());

System.out.println("Server Time: " + dateTimeService.getTime());

} catch (Exception e) {

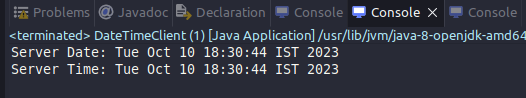
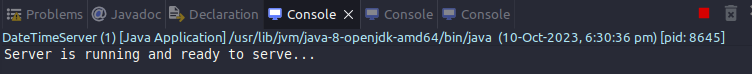
e.printStackTrace();

}

}

}

**Output:**



**Conclusion:**

I learnt how to create a client and server application where the client invokes methods via an interface.