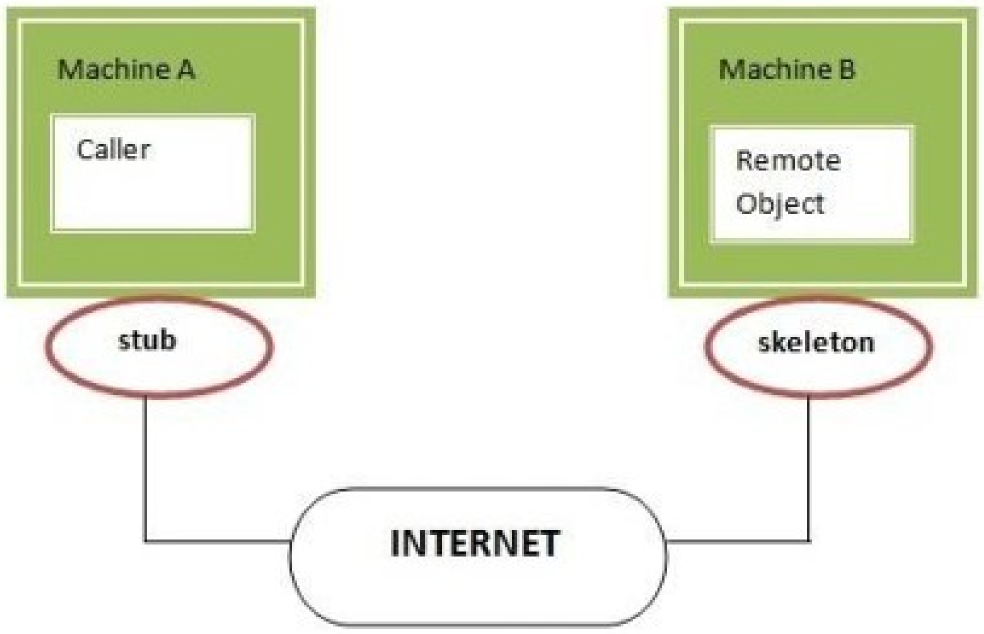
Distributed Computing

Experiment – 2

Program to demonstrate Client/Server application Using RMI	
Learning Objective:	To understand basic underlying concepts of forming distributed systems.
Learning Outcome:	Students will gain ability to demonstrate Client/Server application.
Course Outcome:	**CSL801.1**
Program Outcome:	(**PO**1) **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems. (**PO**2) **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences. **(PO**3) **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations
Theory:	*RMI, Advantages and Disadvantages of RMI, Types of RMI* **Remote Method Invocation (RMI)** The RMI is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM. The RMI provides remote communication between the applications using two objects stub and skeleton.



Stub

The stub is an object, acts as a gateway for the client side. All the outgoing requests are routed through it. It resides at the client side and represents the remote object. When the caller invokes method on the stub object, it does the following tasks:

1. It initiates a connection with remote Virtual Machine (JVM),
2. It writes and transmits (marshals) the parameters to the remote Virtual Machine (JVM),
3. It waits for the result
4. It reads (unmarshals) the return value or exception, and
5. It finally, returns the value to the caller.

Skeleton

The skeleton is an object, acts as a gateway for the server-side object. All the incoming requests are routed through it. When the skeleton receives the incoming request, it does the following tasks:

1. It reads the parameter for the remote method
2. It invokes the method on the actual remote object, and
3. It writes and transmits (marshals) the result to the caller.

Advantages of RMI:

Object-Oriented Nature: RMI is designed to work seamlessly with Java's object-oriented features, allowing developers to invoke methods on remote objects as if they were local.

Simplicity and Productivity: RMI abstracts much of the complexity associated with network communication, making it relatively easy to use compared to lower-level network programming.

Integration with Java: RMI is tightly integrated with the Java programming language, allowing developers to use familiar constructs and tools.

Dynamic Class Loading: RMI supports dynamic class loading, allowing objects to be downloaded and executed on the client side, which can be useful for updating applications without restarting them.

Security: RMI provides security features such as authentication and encryption, ensuring that communication between objects is secure.

Disadvantages of RMI:

Java-Centric: RMI is specific to Java, which limits its interoperability with systems implemented in other languages.

Complex Setup: Setting up RMI can be complex, especially for beginners. Configuring the RMI registry, dealing with class loading issues, and understanding the distributed object model can be challenging.

Performance Overhead: RMI may introduce performance overhead due to network communication, serialization/deserialization of objects, and other related processes.

Firewall Issues: RMI often requires special firewall configurations to allow communication between distributed components, which can be a security

	concern. **Limited Platform Support**: Although RMI is designed for Java, it may not be supported on all platforms, limiting its usability in heterogeneous environments. **Types of RMI:** **Simple RMI**: In this basic form of RMI, communication between client and server involves invoking methods on remote objects. **Parameter Passing in RMI**: Call by Value: Parameters are passed by value, meaning a copy of the parameter is sent to the remote object. Call by Reference: RMI also supports passing parameters by reference, allowing the remote object to directly access and modify the parameter. **RMI with Callbacks (Two-Way RMI)**: This involves the client invoking methods on the server and vice versa. Callbacks enable bidirectional communication between client and server. **Activatable RMI**: Activatable objects are server-side objects that are activated (instantiated) on-demand. They are only loaded into memory when needed, which can be useful for managing resources efficiently.
Algorithm :	**Creating and Running an RMI Application** The following steps are required to create and run an RMI application: 1. Create the remote interface 2. Provide the implementation class for the remote interface 3. Compile the implementation class and create the stub and skeleton objects using the rmic tool 4. Start the registry service by rmiregistry tool 5. Start the server 6. Create and start the client application

Program

1. RMI_Chat_Interface.java -

```
import java.rmi.Remote;

import java.rmi.RemoteException;

public interface RMI_Chat_Interface extends Remote {

    public void sendToServer(String message) throws RemoteException;

}
```

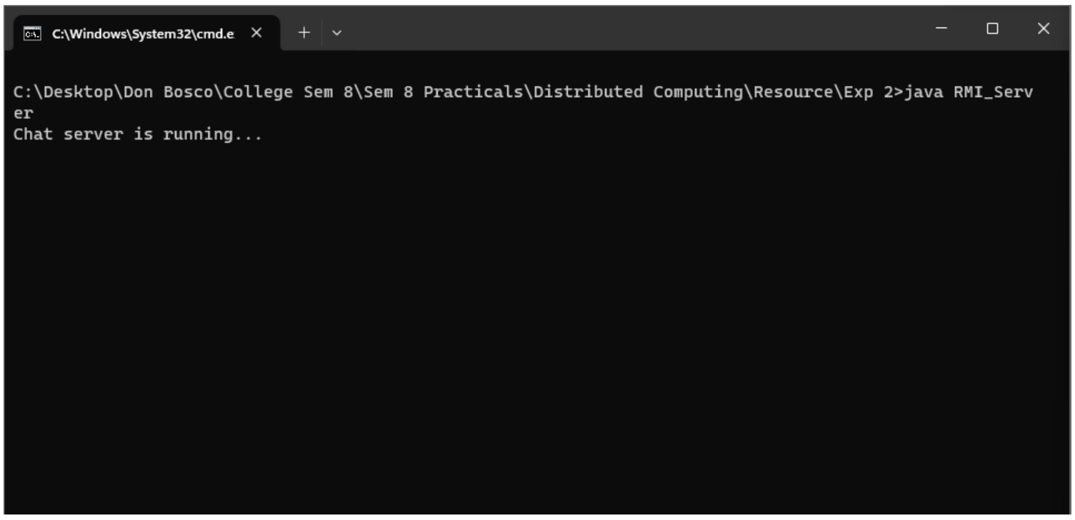
2. RMI_Server.java -

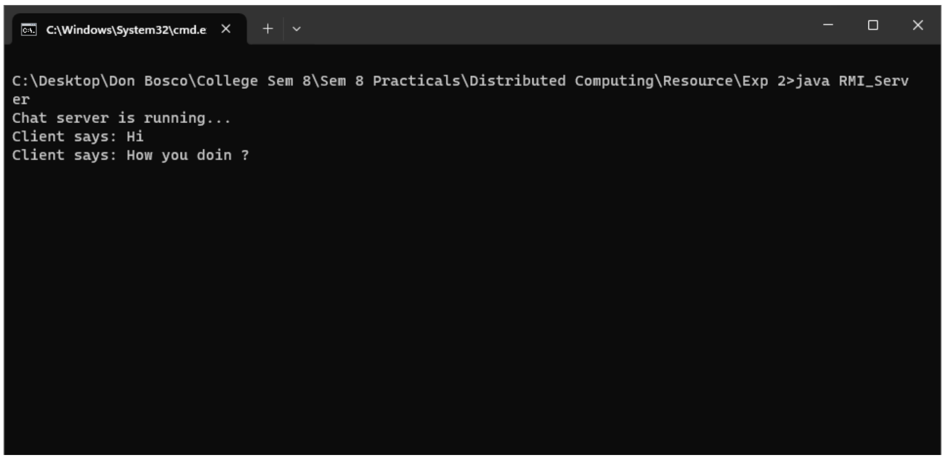
```
import java.rmi.RemoteException;

import java.rmi.registry.LocateRegistry;

import java.rmi.registry.Registry;

import java.rmi.server.UnicastRemoteObject;


public class RMI_Server extends UnicastRemoteObject implements
RMI_Chat_Interface {


    public RMI_Server() throws RemoteException {

        super();

    }

    @Override
    public void sendToServer(String message) throws RemoteException {
        System.out.println("Client says: " + message);
    }
    public static void main(String[] args) throws Exception {
        Registry rmiregistry = LocateRegistry.createRegistry(6000);
        rmiregistry.bind("chat", new RMI_Server());
        System.out.println("Chat server is running...");
    }
}
```

3. RMI_Client.java -

```
import java.rmi.Naming;
import java.util.Scanner;

public class RMI_Client {

    static Scanner input = null;

    public static void main(String[] args) throws Exception {
        RMI_Chat_Interface chatapi = (RMI_Chat_Interface)

        Naming.lookup("rmi://localhost:6000/chat");
        input = new Scanner(System.in);
```

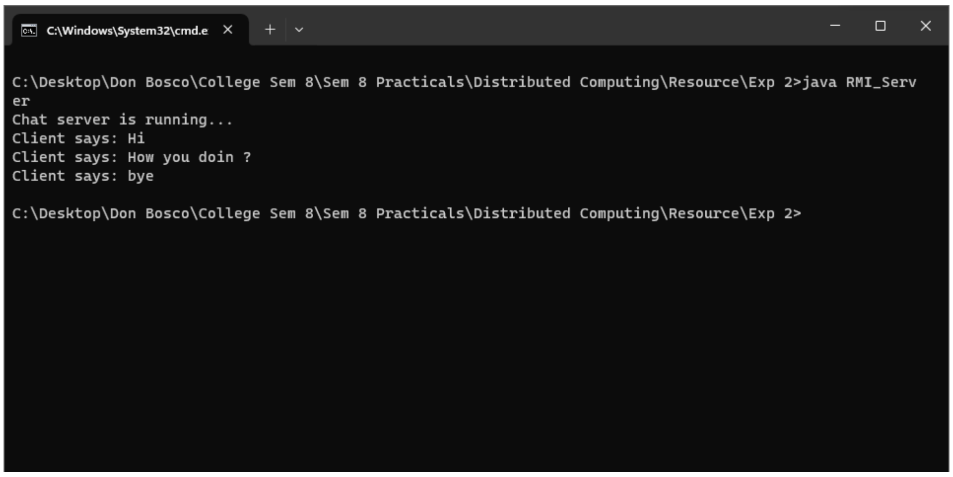
	```
	        System.out.println("Connected to server...");


	        System.out.println("Type a message for sending to server...");
	        String message = input.nextLine();
	        while (!message.equals("Bye")) {
	            chatapi.sendToServer(message);
	            message = input.nextLine();
	        }
	    }
	}
	```

Output	1 RMI_Server.java
	
	Server Start -
	
	
	C:\Windows\System32\cmd.e
	```
	C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>java RMI_Serv
	er
	Chat server is running...
	```
	Client Message Request -



```
C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>java RMI_Serv
er
Chat server is running...
Client says: Hi
Client says: How you doin ?
```

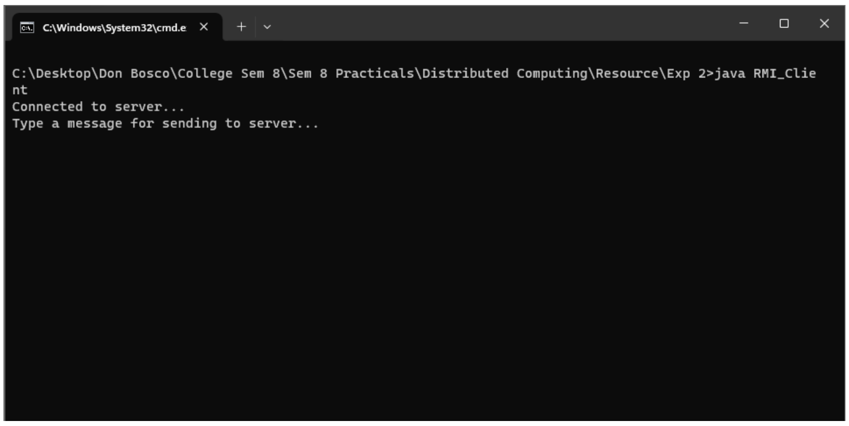
Server Stop Request – Message (Bye)



```
C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>java RMI_Serv
er
Chat server is running...
Client says: Hi
Client says: How you doin ?
Client says: bye

C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>
```

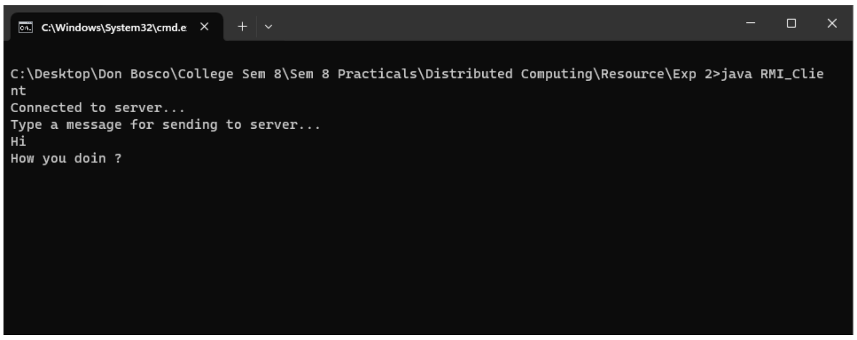
2. RMI_Client.java -

Client Start -



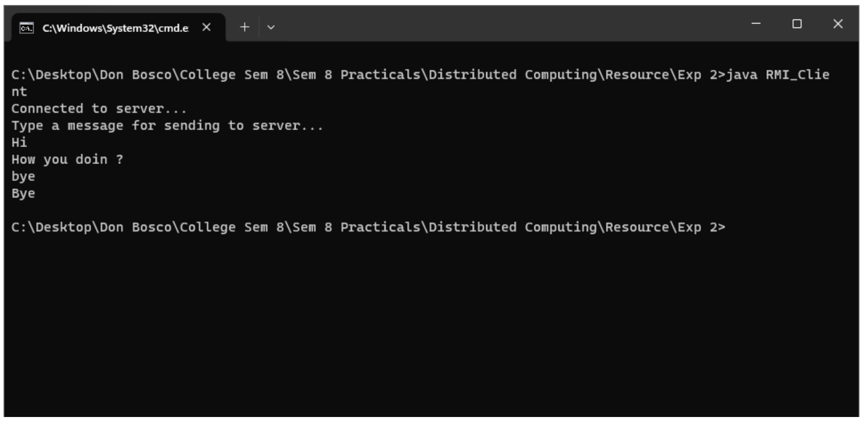
```
C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>java RMI_Clie
nt
Connected to server...
Type a message for sending to server...
```

Client Message Request -



```
C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>java RMI_Clie
nt
Connected to server...
Type a message for sending to server...
Hi
How you doin ?
```

Client Stop – Message(Bye)



```
C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>java RMI_Clie
nt
Connected to server...
Type a message for sending to server...
Hi
How you doin ?
bye
Bye

C:\Desktop\Don Bosco\College Sem 8\Sem 8 Practicals\Distributed Computing\Resource\Exp 2>
```

Conclusion :	Thus, we have studied basic underlying concepts of forming distributed systems and performed Implementation on Client/ Server Application with the help of Remote Method Invocation (RMI)
References:	https://en.wikipedia.org/wiki/Java_remote_method_invocation https://www.oracle.com/java/technologies/jpl1-remote-method-invocation.html http://infolab.stanford.edu/CHAIMS/Doc/Details/Protocols/rmi/rmi_description.html

Rubrics for Assessment

Timely Submission	Submitted after 2 weeks 0	Submitted after deadline 1	On time Submission 2
Understanding	Student is confused about the concept 0	Students has justifiably understood the concept 2	Students is very clear about the concepts 3
Performance	Students has not performed the Experiment 0	Student has performed with help 2	Student has independently performed the experiment 3
Development	Student struggles to write code 0	Student can write code the requirement stated 1	Student can write exceptional code with his own ideas 2