FR. CONCEICAO RODRIGUES COLLEGE OF ENGINEERING

Department of Computer Engineering

Course, Subject & Experiment Details

Practical No:	3
Title:	Remote Method Invocation
Name of the Student:	Warren Fernandes
Roll No:	8940
Date of Performance:	10/02/2023
Date of Submission:	17/02/2023

Evaluation:

Sr. No.	Rubric	Grade
1	Timeliness (1)	
2	Documentation (2)	
3	Preparation (3)	
4	Performance (4)	

Signature of the Teacher

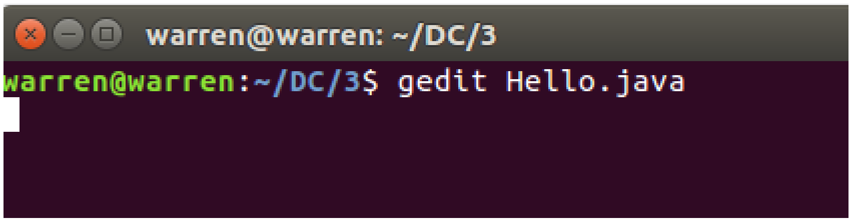
Aim:

The aim of a remote method invocation experiment is to learn how to create distributed applications using Java Remote Method Invocation (RMI). This experiment involves creating a client-server architecture where the client can invoke methods on the server, which may be running on a different machine. Through this experiment, we can gain hands-on experience in developing distributed applications, learning how to create and deploy RMI interfaces, and how to use the RMI registry to locate remote objects. Additionally, we can learn about the advantages and disadvantages of using RMI, such as the ability to create complex distributed systems and the potential for network latency issues. Overall, the aim of the remote method invocation experiment is to gain practical skills in creating and deploying distributed Java applications using RMI.

Steps:

Hello.java

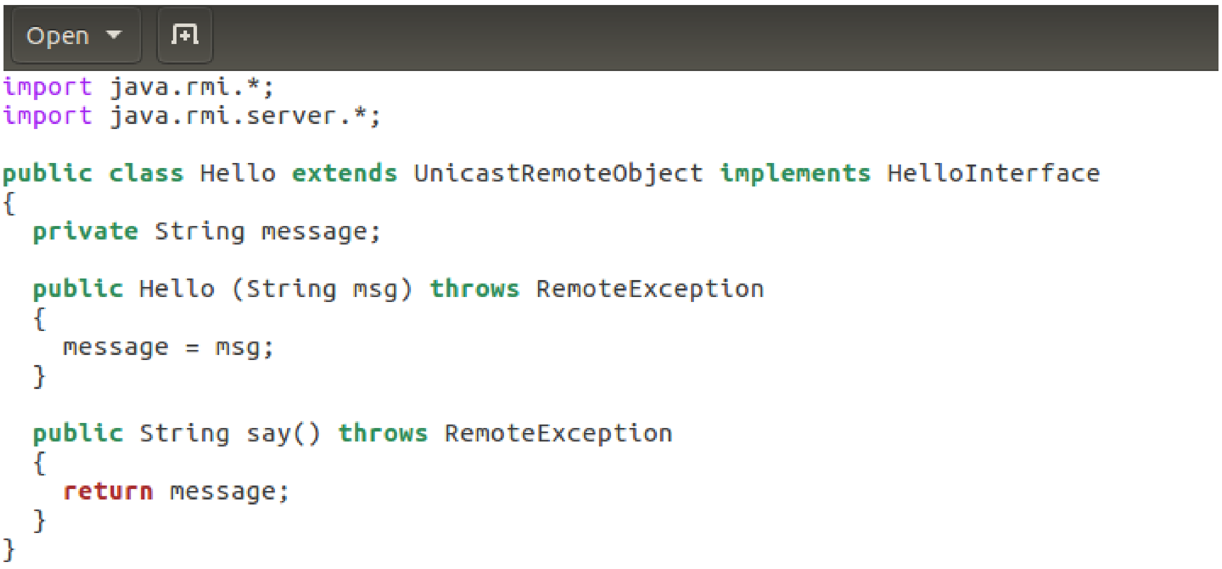
We can use gedit text editor to create a file named Hello.java for Java programming



```
warren@warren:~/DC/3$ gedit Hello.java
```

To create a Java program that uses Remote Method Invocation (RMI), we need to extend the UnicastRemoteObject class and implement a remote interface. Here's an example of how we can create a simple RMI program that returns a message:

In this example, we've created a Hello class that extends UnicastRemoteObject and implements a HelloInterface interface. The HelloInterface interface defines a single method, say(), that returns a String. In the Hello class, we've implemented the say() method to return the string message.

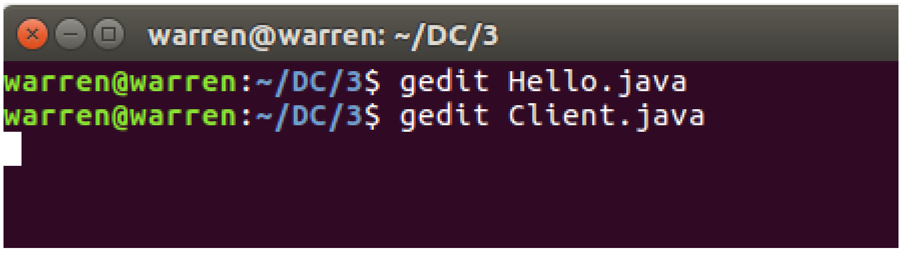


```
import java.rmi.*;
import java.rmi.server.*;

public class Hello extends UnicastRemoteObject implements HelloInterface
{
  private String message;

  public Hello (String msg) throws RemoteException
  {
    message = msg;
  }

  public String say() throws RemoteException
  {
    return message;
  }
}
```

Overall, this program demonstrates the basic structure of an RMI program using Java. We extend UnicastRemoteObject to create a remote object, and we implement a remote interface to define the methods that can be called remotely. We then bind the remote object to the RMI registry so that it can be accessed by clients running on other machines.

Client.java

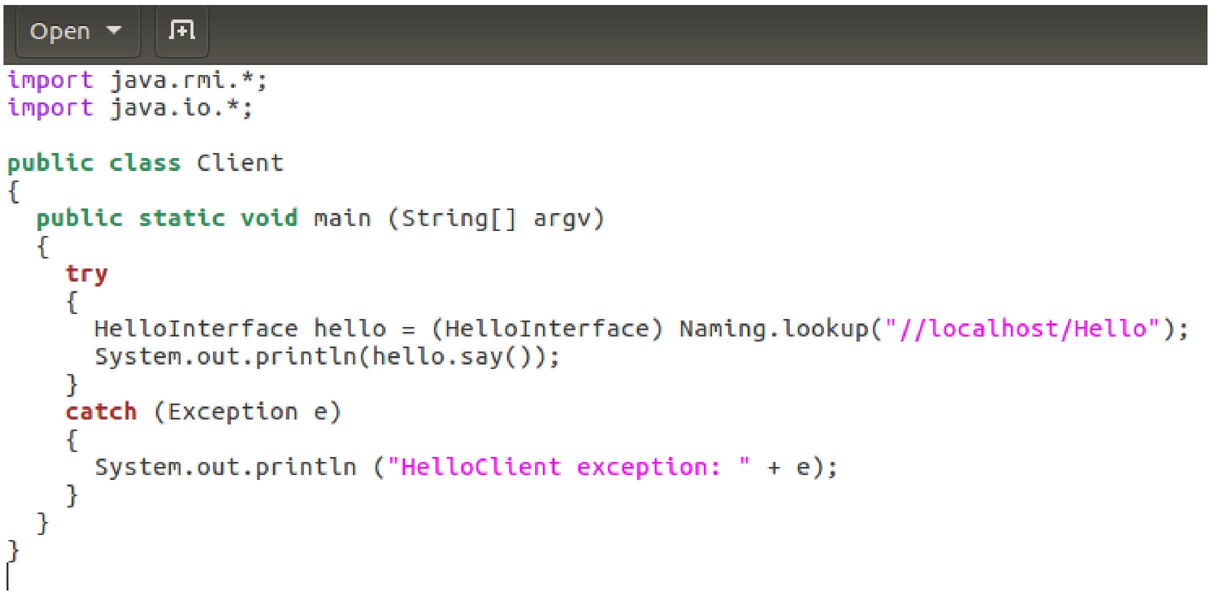
We can use gedit text editor to create a file named Client.java for Java programming



```
warren@warren: ~/DC/3
warren@warren:~/DC/3$ gedit Hello.java
warren@warren:~/DC/3$ gedit Client.java
```

In a Java program that uses Remote Method Invocation (RMI), we can create a client that can invoke methods on a remote server. To do this, we first need to create a client class that uses the Naming.lookup() method to locate the remote object in the RMI registry. Here's an example of how we can create a client program that uses the Naming.lookup() method to locate a remote object called "Hello" on the local machine:

In this example, we've created a HelloClient class that uses the Naming.lookup() method to locate a remote object called "Hello" on the local machine. We've also cast the remote object to an interface called HelloInterface, which defines the method that we want to call remotely (say()).



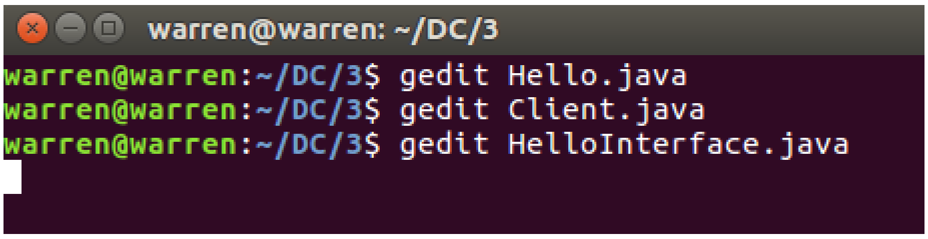
```
import java.rmi.*;
import java.io.*;

public class Client
{
  public static void main (String[] argv)
  {
    try
    {
      HelloInterface hello = (HelloInterface) Naming.lookup("//localhost/Hello");
      System.out.println(hello.say());
    }
    catch (Exception e)
    {
      System.out.println ("HelloClient exception: " + e);
    }
  }
}
```

Once we've located the remote object, we can call its say() method and print the result to the console. If everything is set up correctly, the HelloClient program should print the message "Hello, World!" to the console.

Overall, this program demonstrates the basic structure of an RMI client program using Java. We use the Naming.lookup() method to locate the remote object in the RMI registry, and we cast it to an interface so that we can call its methods remotely. We then call the remote method and handle any exceptions that may occur.

HelloInterface.java

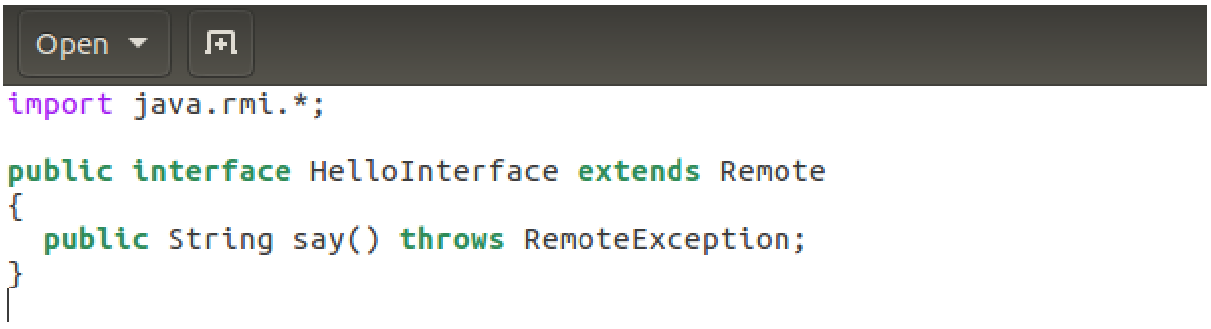
We can use gedit text editor to create a file named HelloInterface.java for Java programming



```
warren@warren:~/DC/3$ gedit Hello.java
warren@warren:~/DC/3$ gedit Client.java
warren@warren:~/DC/3$ gedit HelloInterface.java
```

In a Java program that uses Remote Method Invocation (RMI), we can define a remote interface that specifies the methods that can be called remotely by clients. This interface must extend the java.rmi.Remote interface, and each of its methods must declare a java.rmi.RemoteException in its throws clause.

Here's an example of how we can define a remote interface called HelloInterface that has a single method called say():



```java
import java.rmi.*;

public interface HelloInterface extends Remote
{
  public String say() throws RemoteException;
}
```

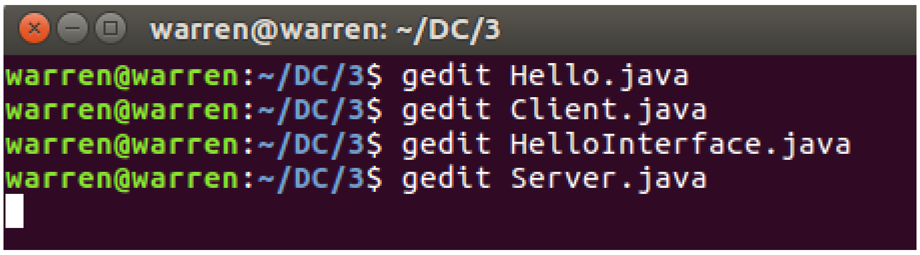
In this example, we've defined a remote interface called HelloInterface that extends the java.rmi.Remote interface. We've also defined a single method called say() that returns a String and declares a RemoteException in its throws clause.

Any class that implements this interface can be registered as a remote object in the RMI registry and can be accessed remotely by clients. The HelloInterface interface simply defines the methods that can be called remotely; the implementation of these methods is left up to the classes that implement the interface.

Overall, this program demonstrates how to define a remote interface in Java for use with RMI. We define the interface to extend java.rmi.Remote, and we declare any exceptions that may be thrown by its methods. We can then implement this interface in any class that we want to make available remotely.

Server.java

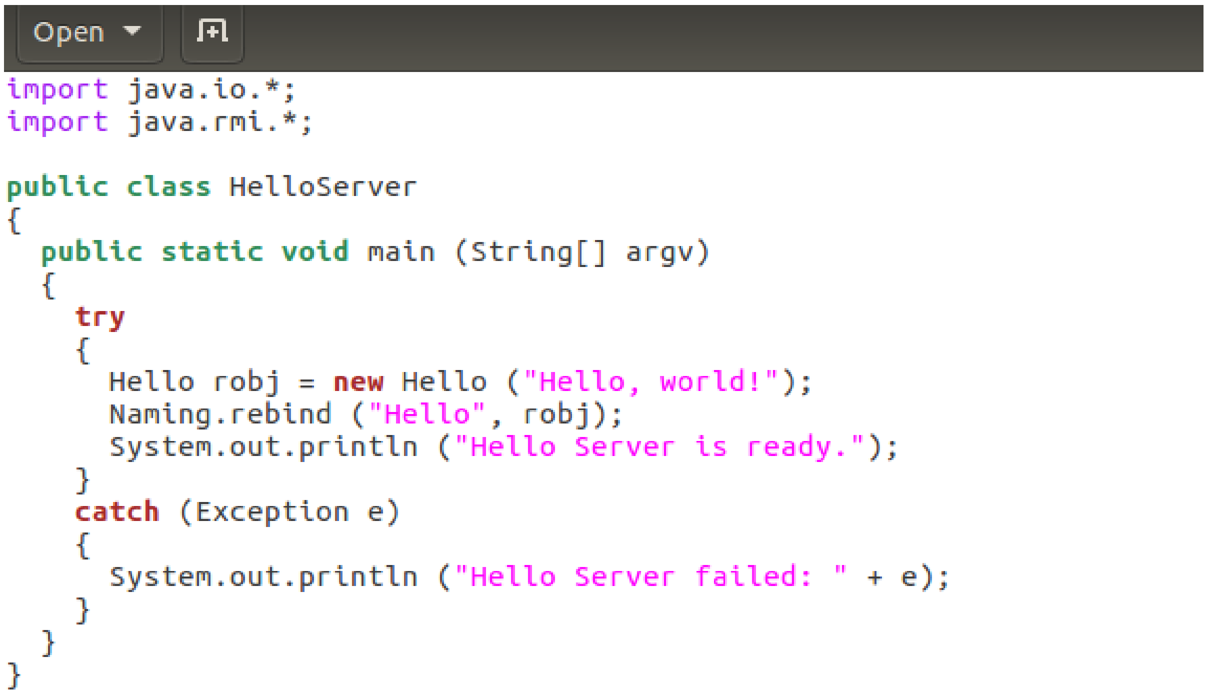
We can use gedit text editor to create a file named Server.java for Java programming



```
warren@warren:~/DC/3$ gedit Hello.java
warren@warren:~/DC/3$ gedit Client.java
warren@warren:~/DC/3$ gedit HelloInterface.java
warren@warren:~/DC/3$ gedit Server.java
```

In a Java program that uses Remote Method Invocation (RMI), we can define a server that exports remote objects and registers them with a naming service, such as the RMI registry. This allows clients to locate and invoke remote methods on these objects.

Here's an example of how we can define a server that exports a remote object and registers it with the RMI registry using the Naming.rebind() method:



```
import java.io.*;
import java.rmi.*;

public class HelloServer
{
  public static void main (String[] argv)
  {
    try
    {
      Hello robj = new Hello ("Hello, world!");
      Naming.rebind ("Hello", robj);
      System.out.println ("Hello Server is ready.");
    }
    catch (Exception e)
    {
      System.out.println ("Hello Server failed: " + e);
    }
  }
}
```

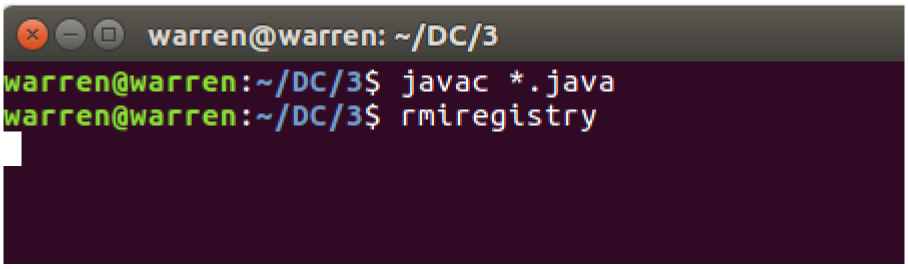
In this example, we've defined a server class called Server that creates a remote object called Hello and register it with the RMI registry using the Naming.rebind() method.

The Naming.rebind() method takes two arguments: a string that specifies the name under which the remote object should be registered, and the stub for the remote object. In this example, we've chosen the name "Hello" to identify the remote object.

Overall, this program demonstrates how to define a server in Java for use with RMI. We export a remote object and register it with the RMI registry Naming.rebind() methods. Once registered, the remote object can be accessed remotely by clients using its registered name.

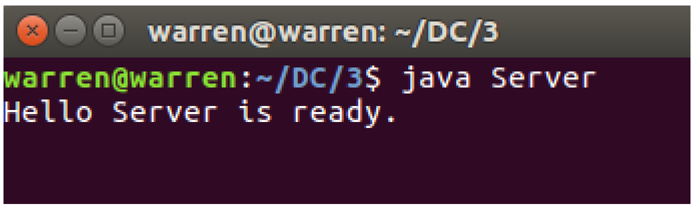
You can use javac to compile both the server and client Java files, which will generate bytecode files that can be executed using the Java Virtual Machine (JVM).

To start the rmiregistry, you can use the command "rmiregistry" in the terminal/command prompt.



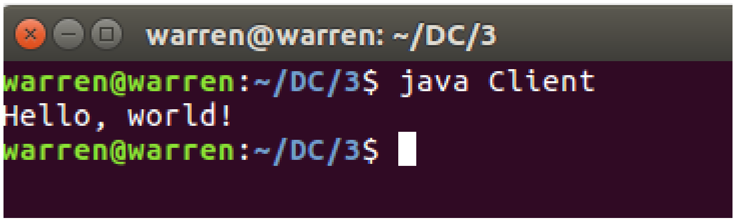
```
warren@warren: ~/DC/3
warren@warren:~/DC/3$ javac *.java
warren@warren:~/DC/3$ rmiregistry
```

To run the server, you can execute the compiled Server.class file using the following command in the terminal or command prompt:



```
warren@warren: ~/DC/3
warren@warren:~/DC/3$ java Server
Hello Server is ready.
```

To run the client, you can execute the compiled Client.class file using the following command in the terminal or command prompt:



```
warren@warren: ~/DC/3
warren@warren:~/DC/3$ java Client
Hello, world!
warren@warren:~/DC/3$
```

We have successfully run the Java RMI client and printed "Hello World" to the console. This confirms that the client was able to make a remote method invocation and receive a response from the server.

<u>Conclusion:</u>

In conclusion, the RMI experiment demonstrates how to use the Java Remote Method Invocation (RMI) framework to build distributed applications that communicate over a network. By implementing the Remote interface and using the UnicastRemoteObject class, it is possible to create objects that can be accessed remotely by clients. The RMI registry is used to keep track of these objects and provide a lookup service. Overall, RMI is a powerful tool for building distributed systems in Java.