PROJECT 3: FUNCTIONAL CALCULATOR USING REMOTE METHOD INVOCATION

* RMI stands for Remote Method Invocation.
* RMI allows the clients to invoke methods of a remote interface on remote objects.
* RMI does not require any connection protocol like one in CORBA.

The project has been implemented in the following steps:

STEP1: The project starts with defining an INTERFACE which contains the required methods, parameters, events, arguments, exceptions etc. In order to make the interface REMOTE your interface must extend java.rmi.Remote. The file is saved as “Calculator.java”

STEP2: Implement the Remote Interface. This class CalculatorImpl writes the definitions of all methods defined in the remote interface. The class must extend UnicastRemoteObject to make the objects remotely available.

STEP3: Implement the Server. The class CalculatorServer contains the main(). Server Class performs the following functionalities:

* The server registers its remote objects with the RMIRegistry.
* It passes the URL and the object reference.
* The first argument is the URL. The URL contains the IP address, default port number and the name of Remote Object.
* The second argument is the remote object reference.
* Calls the constructor from the main().

STEP4:

Implement the Client. Client class contains the main() and performs the following functionalities:

* Calls the lookup() of the Naming class.
* Passes the URL to identify the reference to the remote object by name.
* Calls the methods of the remote interface on Client requests.
* Fetches the results back to the Clients.

STEP5:

This part describes how I made a working calculator.

* Accept a string from the user in the form “+\_num1\_ num2”.
* Separate the operator and the two operands using substring().
* Using charAt(), store them in three different characters.
* Convert the operator into its ASCII value and the operators into their integer values using Character.getNumericValue().
* Pass the ASCII value of the operator and the two operands to the calculate().
* Store the returned result in a variable and display the result.
* If the client wants to continue then accept another string else invoke the exit().

STEP6:

Running the Calculator Application:

* Compile the interface using the following command: javac Calculator.java
* Compile the Server implementation using the following command: javac CalculatorImpl.java
* Compile the Server with main() using the following command: javac CalculatorServer.java
* Compile the Client using the following command: javac CalculatorClient.java
* Generate the stub and the skeleton using the following command: rmic CalculatorImpl
* Now open three different consoles to run the application.
* On the first console, start the RMI Registry using the following command: rmiregistry
* On the second console, start the Server using the following command: java CalculatorServer
* On the third console, start the Client using the following command: java CalculatorClient.

Interface:

//A container which holds all the interfaces,methods,parameters,

//exceptions,events etc.

//This interface is a remote interface

public interface Calculator extends java.rmi.Remote

{

//RemoteException occurs when the RMI procedure is unsuccessful.

//The methods handle Network Error Exceptions

public int calculate(int opcode,int op1,int op2)

throws java.rmi.RemoteException;

public void exit()

throws java.rmi.RemoteException;

}

Remote Interface:

//This class has the definitions of the interface methods.

//This class runs on the remote server

//UnicastRemoteObject is the base class for all remote objects

//It provides point to point TCP connectivity

public class CalculatorImpl

extends

java.rmi.server.UnicastRemoteObject

implements

Calculator

{

//The constructor declares the RemoteException exception

public CalculatorImpl()

throws java.rmi.RemoteException

{

//call to the superclass constructor

super();

}

public int calculate(int opcode,int op1,int op2)

throws java.rmi.RemoteException

{

//ADDITION

if(opcode==43)

{

return (op1+op2);

}

//SUBTRACTION

else if(opcode==45)

{

return (op1-op2);

}

//MULTIPLICATION

else if(opcode==42)

{

return (op1\*op2);

}

//DIVISION

else if(opcode==47)

{

return (op1/op2);

}

//DEFAULT

else

{

return 0;

}

}

public void exit()

throws java.rmi.RemoteException

{

System.out.println("The server is out and exiting");

}

}

Server:

import java.rmi.Naming;

public class CalculatorServer

{

public CalculatorServer()

{

try

{

Calculator c = new CalculatorImpl();

//The server registers its remote objects with the RMIRegistry

//It passes the URL and the object reference

//The first argument is the URL

//The URL contains the IP address,default port number,

//name of Remote Object.

//The second argument is the remote object reference.

Naming.rebind("rmi://localhost:1099/PrathamVasaCalculator", c);

System.out.println("Server is ready and waiting");

}

catch (Exception e)

{

System.out.println("There is an exception: " + e);

}

}

public static void main(String args[])

{

//call to the constructor

new CalculatorServer();

}

}

Client

import java.rmi.Naming;

import java.rmi.RemoteException;

import java.net.MalformedURLException;

import java.rmi.NotBoundException;

import java.io.\*;

import java.util.\*;

public class CalculatorClient

{

public static void main(String[] args)throws IOException

{

try

{

//Client passes the same URL as registered by the Server

//to get the access of the remote objects.

Calculator c = (Calculator)

Naming.lookup

("rmi://localhost/PrathamVasaCalculator");

int x=1;

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

while(x==1)

{

System.out.println("Enter the string:");

String abc=br.readLine();

//If the Client wants to discontinue then Server closes.

if(abc.equalsIgnoreCase("exit"))

{

x=0;

c.exit();

break;

}

int i;

for(i=0;i<abc.length();i++)

{

//Find out the index of the space

if(Character.isWhitespace(abc.charAt(i))&&(i!=1))

{

break;

}}

//Splitting the operator and the two operands.

int index=i;

String a1=abc.substring(0,1);

String a2=abc.substring(2,index);

String a3=abc.substring(index+1);

//Obtain the ASCII value of the operator.

char ch1=abc.charAt(0);

int b1=(int)ch1;

//Obtain the Integer values of the operands.

int b2=Integer.parseInt(a2);

int b3=Integer.parseInt(a3);

int result=c.calculate(b1,b2,b3);

System.out.println("The result is:"+result);

}}

//If a malformed URL has been discovered

//or the string has not been parsed properly.

catch (MalformedURLException murle)

{

System.out.println();

System.out.println

("MalformedURLException");

System.out.println(murle);

}

//If there is some problem in the network communication.

catch (RemoteException re)

{

System.out.println();

System.out.println("RemoteException");

System.out.println(re);

}

//If you try to unbind a name which has not yet been bound.

catch (NotBoundException nbe)

{

System.out.println();

System.out.println("NotBoundException");

System.out.println(nbe);

}

//If an exceptional arithmetic condition occurs.

catch (java.lang.ArithmeticException ae)

{

System.out.println();

System.out.println("java.lang.ArithmeticException");

System.out.println(ae);

}

}

}