|  |  |
| --- | --- |
| Ex.No.8 | IMPLEMENTATION OF CONCURRENT SERVERS USING MULTITHREADING |

|  |
| --- |
| **AIM:** |

To develop Concurrent server/Client application for Student Query System in Java

|  |
| --- |
| **THEORY:** |

**Concurrent Server:**

A concurrent server has to be able to serve more than one client at a time. i.e. it iterates through each client and serves one request at a time. Alternatively, a server can handle multiple clients at the same time in parallel, and this type of a server is called a concurrent server.

**Differences between Concurrent Server and Iterative Server:**

|  |  |
| --- | --- |
| **Concurrent Server** | **Iterative Server** |
| Handles multiple client at a time | Handles one at a time |
| It waits for client through accept() and it closes the connection when done | It waits for client through accept() and it goes back to listen for further upcoming clients |

**APIs and methods required to implement Concurrent Servers:**

ServerSocket() – To get server up and running

Socket() – To accept clients and create clients

Thread() – To allot each client the server’s functionality

PrintStream() – To write to socket stream

InputStream() – Socket’s input stream

OutputStream() – Socket’s Output stream

InetAddress() – Used to get the internet address

**Adding Comments to Summation Server and Summation Client:**

**SummationServer.java:**

import java.io.\*;

import java.net.\*;

class summationThread extends Thread{

Socket clientSocket;

summationThread(Socket cs){ clientSocket = cs; } //Copying incoming client connection

public void run( ){

try{

BufferedReader br = new BufferedReader(new

InputStreamReader(clientSocket.getInputStream( ))); //Getting socket input stream

int count = Integer.parseInt(br.readLine( ));

int sum = 0;

for (int ctr = 1; ctr <= count; ctr++){

sum += ctr;

Thread.sleep(200);

}

PrintStream ps = new PrintStream(clientSocket.getOutputStream( )); //getting socket

//output stream and prints the result

ps.println(sum);

ps.flush( );

clientSocket.close( ); //Closing connection

}

catch(Exception e){e.printStackTrace( );}

}

}

class summationServer{

public static void main(String[ ] args){

try{

int serverPort = Integer.parseInt(args[0]);

ServerSocket calcServer = new ServerSocket(serverPort); //Initializing server at port

while (true){

Socket clientSocket = calcServer.accept( ); //waiting for incoming connection

summationThread thread = new summationThread(clientSocket);

thread.start( ); //new thread process for client

}

}

catch(Exception e){e.printStackTrace( );}

}

}

**SummationClient.java:**

import java.io.\*;

import java.net.\*;

class summationClient{

public static void main(String[ ] args){

try{

InetAddress serverHost = InetAddress.getByName(args[0]);

int serverPort = Integer.parseInt(args[1]);

long startTime = System.currentTimeMillis( );

int count = Integer.parseInt(args[2]);

Socket clientSocket = new Socket(serverHost, serverPort); //creating new client with given

//ip and port

PrintStream ps = new PrintStream(clientSocket.getOutputStream()); //getting out stream

ps.println(count);

BufferedReader br = new BufferedReader(new

InputStreamReader(clientSocket.getInputStream())); //getting input stream

int sum = Integer.parseInt(br.readLine());

System.out.println(" sum = "+sum);

long endTime = System.currentTimeMillis();

System.out.println(" Time consumed for receiving the feedback from the server:

"+(endTime-startTime)+" milliseconds");

clientSocket.close( ); //close connection

}

catch(Exception e){e.printStackTrace( );}

}

}

|  |
| --- |
| **ALGORITHM:** |

**Server:**

1. Setting port address and running server in localhost
2. Storing it in a non primitive data type variable
3. Establishing the Connection
4. Server socket object is initialized and inside a while loop a socket object continuously accepts incoming connection.
5. Obtaining the Streams
6. The inputstream object and outputstream object is extracted from the current requests’ socket object
7. Invoking the start()
8. Accepting further connections
9. Displaying served connection names

**Client:**

1. Creating a client socket with ip and port number
2. Obtaining the input and output stream of the socket
3. Query the server
4. Print the result
5. Close connection if needed

|  |
| --- |
| **CODING:** |

**Server Side Code:**

import java.io.\*;

import java.net.\*;

import java.util.\*;

class summationThread extends Thread{

Socket clientSocket;

summationThread(Socket cs){ clientSocket = cs; }

public void run( ){

try{

BufferedReader br = new BufferedReader(new

InputStreamReader(clientSocket.getInputStream( )));

String name=br.readLine();

String i1=br.readLine();

String i2=br.readLine();

int gst=Integer.parseInt(i1);

int total=Integer.parseInt(i2);

float a=(gst\*total)/100;

float sum=a+total;

Thread.sleep(200);

PrintStream ps = new PrintStream(clientSocket.getOutputStream( ));

ps.println(sum);

System.out.println("Served : "+name);

ps.flush( );

clientSocket.close( );

}

catch(Exception e){e.printStackTrace( );}

}

}

class summationServer{

public static void main(String[ ] args){

try{

System.out.print("Enter the port address");

Scanner s=new Scanner(System.in);

int serverPort =s.nextInt();

ServerSocket calcServer = new ServerSocket(serverPort);

while (true){

Socket clientSocket = calcServer.accept( );

summationThread thread = new summationThread(clientSocket);

thread.start( );

}

}

catch(Exception e){e.printStackTrace( );}

}

}

**Client Side Code:**

import java.io.\*;

import java.util.\*;

import java.net.\*;

class summationClient{

public static void main(String[ ] args){

try{

InetAddress serverHost = InetAddress.getByName("localhost");

System.out.print("Enter the port address");

Scanner s=new Scanner(System.in);

int serverPort =s.nextInt();

long startTime = System.currentTimeMillis( );

System.out.println("Enter the name");

String name=s.next();

System.out.println("Enter the product id:\n1.Electronics\n2.Furnitures\n3.Garments");

String types=s.next();

System.out.print("Enter the gst");

int gst=s.nextInt();

System.out.println("Enter the total amount");

int total=s.nextInt();

Socket clientSocket = new Socket(serverHost, serverPort);

PrintStream ps = new PrintStream(clientSocket.getOutputStream());

ps.println(name);

ps.println(String.valueOf(gst));

ps.println(String.valueOf(total));

BufferedReader br = new BufferedReader(new

InputStreamReader(clientSocket.getInputStream()));

String sum = br.readLine();

System.out.println(" sum = "+sum);

long endTime = System.currentTimeMillis();

System.out.println(" Time consumed for receiving the feedback from the server:"+(endTime-startTime)+" milliseconds");

clientSocket.close( );

}

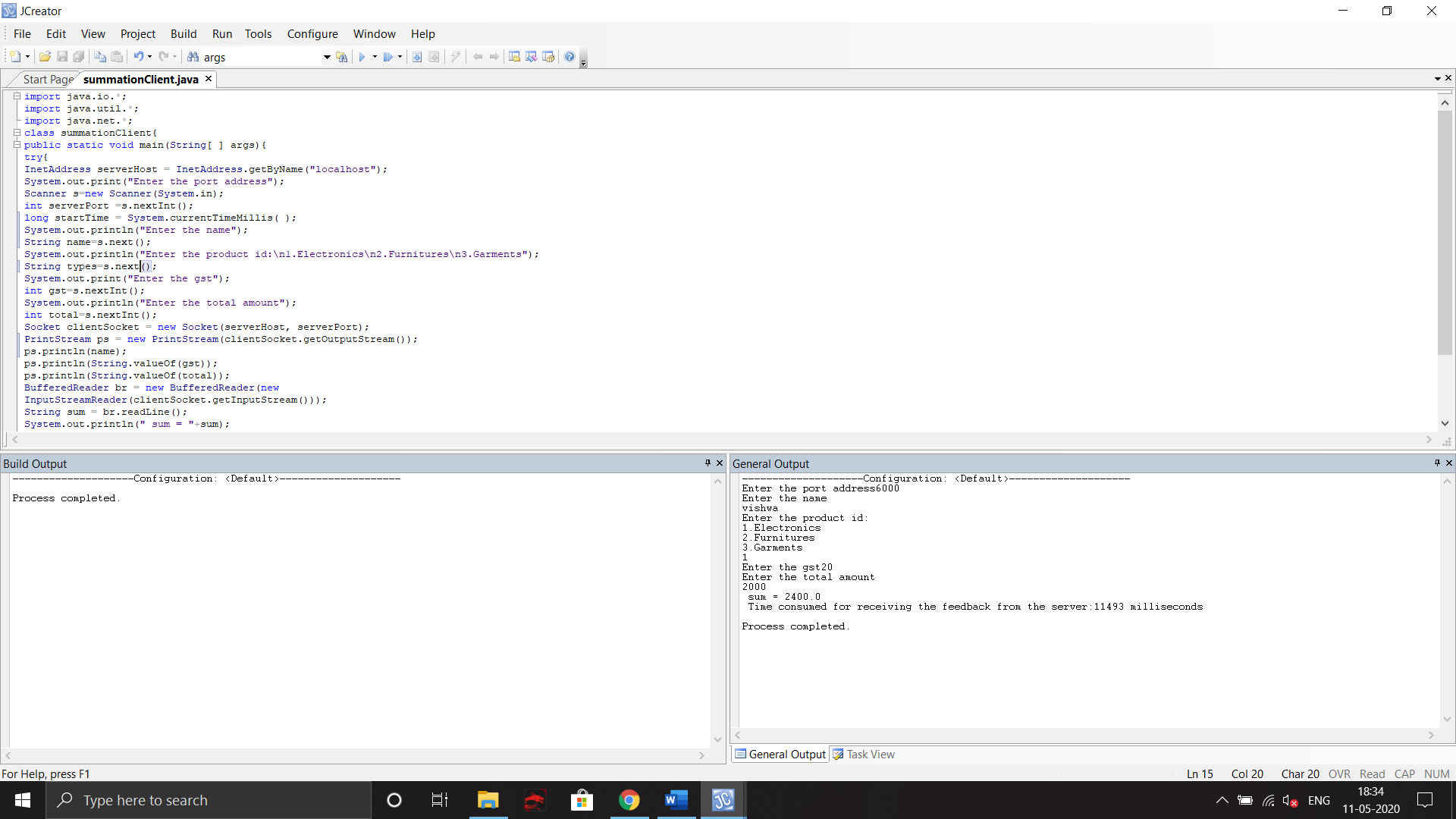
catch(Exception e){e.printStackTrace( );}

}

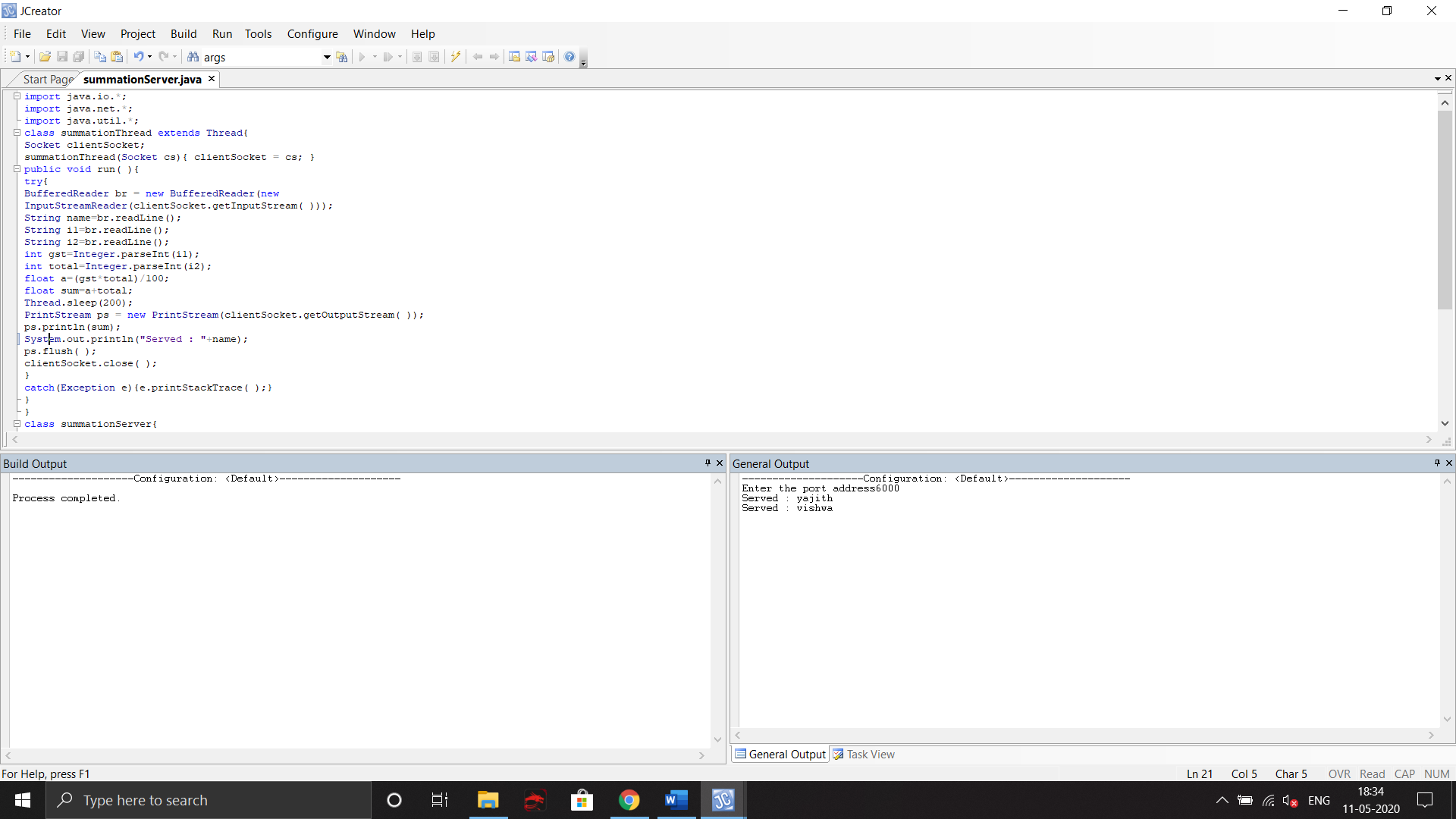
}

|  |
| --- |
| **SCREEN SHOTS:** |

**Clients:**



**Server:**



|  |
| --- |
| **RESULT:** |

Thus, the application for Student Query System is deployed as a Concurrent Server/Client in Java and the results are verified.

**Evaluation:**

|  |  |  |
| --- | --- | --- |
| Parameter | Max Marks | Marks Obtained |
| Complexity of the Application | 10 |  |
| Clarity of Algorithm | 3 |  |
| Use of Comment lines and standard coding practices | 2 |  |
| Viva | 5 |  |
| Sub Total | 20 |  |
| Completion of experiment on time | 3 |  |
| Documentation | 7 |  |
| Sub Total | 10 |  |
| Signature of the faculty with Date |  |  |