**PRACTICAL NO. 11**

**AIM: DEMONSTRATATION OF VARIOUS READER AND WRITER SUBCLASSES IN LISTING**

**Program:**

import java.io.\*;

public class ReaderWriter

{

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

{

System.out.println("With InputStreamReader");

int a;

String s;

InputStreamReader inr=new InputStreamReader(System.in);

System.out.println("enter a line");

while((a=inr.read())!=13)

System.out.println((char)a);

System.out.println();

System.out.println("With BufferedReader and InputStreamReader");

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

System.out.println("enter a line");

System.out.println(br.readLine());

System.out.println("Output With PrintWriter and FileWriter");

BufferedReader br1 = new

BufferedReader(new

InputStreamReader(System.in));

PrintWriter p=new PrintWriter(new FileWriter("demo.out"));

while((s=br1.readLine())!=null)

p.println("output "+s);

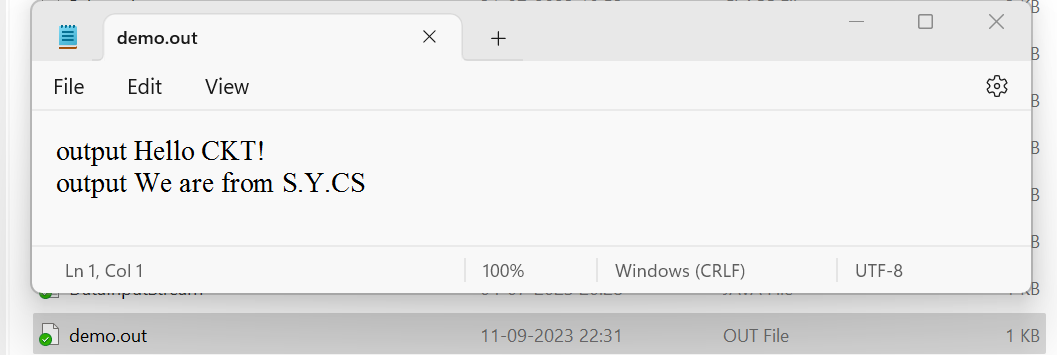
p.close();

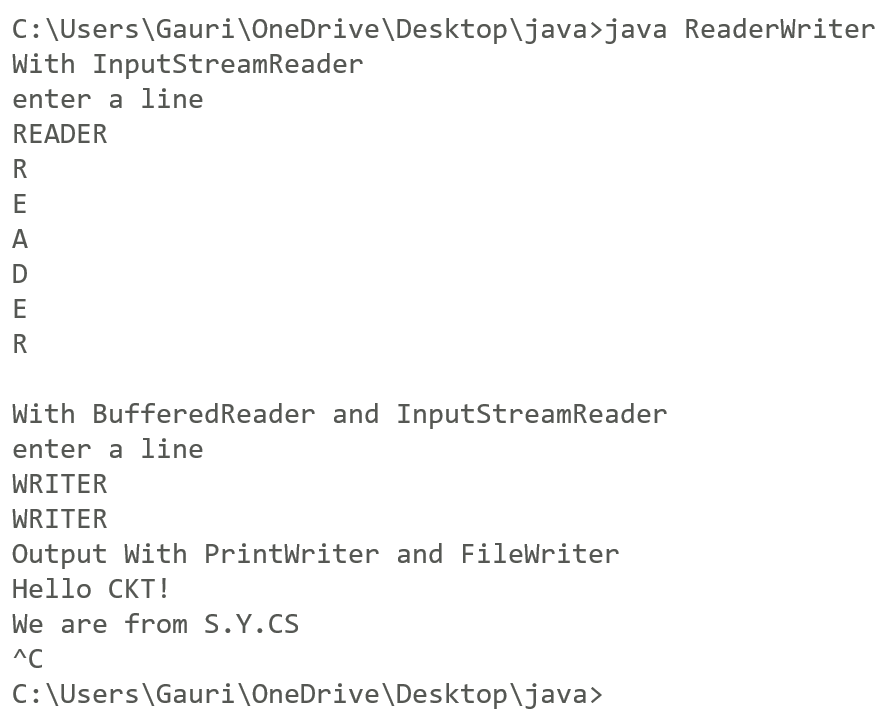
}

}

**Output:**

Created file-





**PRACTICAL NO. 12**

AIM : WRITE A JAVA PROGRAM USING RUNNABLE INTERFACE AND WITH THE HELP OF THREAD CLASS , CREATE THREE THREADS. RUN EACH THREAD 10 TIMES AND THEN STOP THREAD EXCUTION.

Program:

class A implements Runnable

{

public void run()

{

int i;

for(i=1;i<=10;i++)

{

System.out.println("Thread A : "+i);

}

}

}

class B implements Runnable

{

public void run()

{

int i;

for(i=1;i<=10;i++)

{

System.out.println("Thread B : "+i);

}

}

}

class C implements Runnable

{

public void run()

{

int i;

for(i=1;i<=10;i++)

{

System.out.println("Thread C : "+i);

}

}

}

class RunnableDemo

{

public static void main(String hello[])throws Exception

{

System.out.println("Main starts");

Thread t1 = new Thread(new A());

Thread t2 = new Thread(new B());

Thread t3 = new Thread(new C());

t1.start();

t2.start();

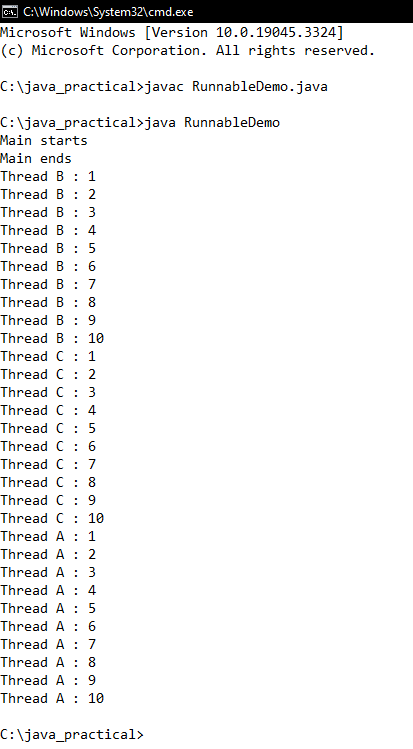
t3.start();

System.out.println("Main ends");

}//end main()

}//end class

**Output**:



**PRACTICAL NO. 13**

**AIM:** Write a program to create 4 threads to perform 4 different arithmetic operations like addition, subtraction, multiplication and division. Accept two numbers from command line arguments and perform the operations using thread.

**Program:**

import java.io.\*;

class Add extends Thread

{

int n1,n2;

public Add(int x, int y)

{

n1=x;

n2=y;

}

public void run()

{

System.out.println("Addition is : "+(n1+n2));

}

}

class Sub extends Thread

{

int n1,n2;

public Sub(int x, int y)

{

n1=x;

n2=y;

}

public void run()

{

System.out.println("Subtraction is : "+(n1-n2));

}

}

class Mul extends Thread

{

int n1,n2;

public Mul(int x, int y)

{

n1=x;

n2=y;

}

public void run()

{

System.out.println("Multplication is : "+(n1\*n2));

}

}

class Div extends Thread

{

int n1,n2;

public Div(int x, int y)

{

n1=x;

n2=y;

}

public void run()

{

System.out.println("Division is : "+(n1/n2));

}

}

class ThreadDemo

{

public static void main(String ar[])

{

try

{

DataInputStream dis = new DataInputStream(System.in);

System.out.println("Enter two values");

int a = Integer.parseInt(dis.readLine());

int b = Integer.parseInt(dis.readLine());

new Add(a,b).start();

new Sub(a,b).start();

new Mul(a,b).start();

new Div(a,b).start();

}

catch(Exception e)

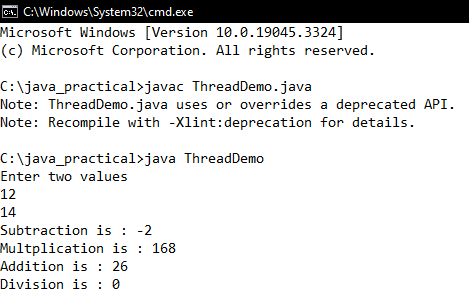
{

}

}

}

**Output:**



**PRACTICAL NO. 14**

**AIM:** WRITE A CLIENT SOCKET THAT WILL ACCEPT N NAMES FROM USER AND SEND THEM TO THE SERVER. AFTER RECEIVING THE NAMES , THE SERVER SOCKET SHOULD SEND THE MESSAGE “NAMES RECEIVED: AND CLOSE THE CONNECTION.

**Program:**

**Client code:**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class Client

{

public static void main(String[] args)

{

final String serverAddress = "localhost"; // Change to the server's IP if needed

final int serverPort = 12345; // Specify the server's port

try

{

Socket socket = new Socket(serverAddress, serverPort);

System.out.println("Connected to server: " + serverAddress + ":" + serverPort);

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);

Scanner scanner = new Scanner(System.in);

System.out.println("Enter names (separated by commas):");

String names = scanner.nextLine();

// Send names to server

writer.println(names);

// Receive and display the server's confirmation message

String confirmationMessage = reader.readLine();

System.out.println("Server says: " + confirmationMessage);

socket.close();

reader.close();

writer.close();

}

catch (IOException e)

{

e.printStackTrace();

}

}

}

**Server code:**

import java.io.\*;

import java.net.\*;

public class Server

{

public static void main(String[] args)

{

final int port = 12345; // Specify the port you want to use

try

{

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Server is listening on port " + port);

while (true)

{

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress());

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);

String receivedNames = reader.readLine();

System.out.println("Received names from client: " + receivedNames);

// Send confirmation message to client

writer.println("NAMES RECEIVED: " + receivedNames);

writer.close();

reader.close();

clientSocket.close();

}

}

catch (IOException e)

{

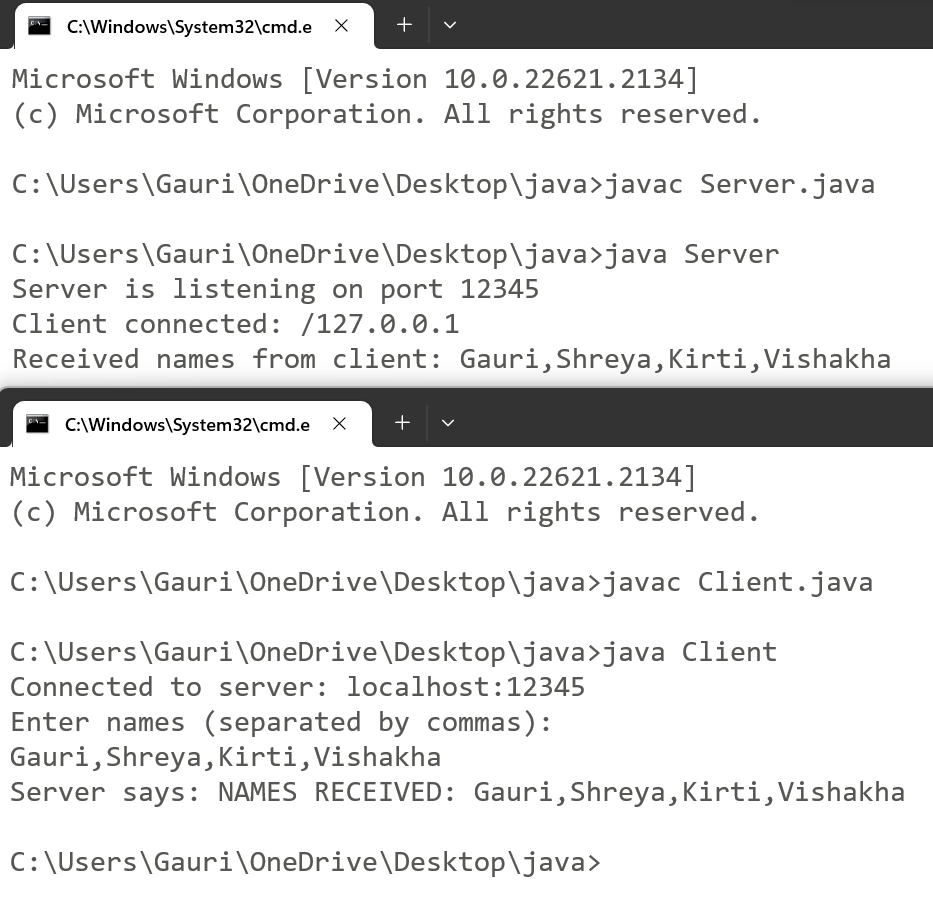
e.printStackTrace();

}

}

}

**Output:**



**PRACTICAL NO. 15**

**AIM:** CREATE A CLIENT SOCKET WHICH SENDS A NUMBER TO THE SERVER. THE SERVER SOCKET RETURNS THE SUM OF DIGITS OF THE NUMBER IF THE NUMBER IS POSITIVE, OTHERWISE IT SENDS AN ERROR MESSAGE AND CLOSE THE CONNECTION.

Program:

**Server code:**

import java.io.\*;

import java.net.\*;

public class Server

{

public static void main(String[] args)

{

final int port = 12345; // Specify the port you want to use

try

{

ServerSocket serverSocket = new ServerSocket(port);

System.out.println("Server is listening on port " + port);

while (true)

{

Socket clientSocket = serverSocket.accept();

System.out.println("Client connected: " + clientSocket.getInetAddress());

BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

PrintWriter writer = new PrintWriter(clientSocket.getOutputStream(), true);

String clientInput = reader.readLine();

try

{

int number = Integer.parseInt(clientInput);

if (number >= 0)

{

int sumOfDigits = calculateSumOfDigits(number);

writer.println("Sum of digits: " + sumOfDigits);

}

else

{

writer.println("Error: Negative number not allowed");

}

}

catch (NumberFormatException e)

{

writer.println("Error: Invalid input");

}

writer.close();

reader.close();

clientSocket.close();

}

}

catch (IOException e)

{

e.printStackTrace();

}

}

private static int calculateSumOfDigits(int number)

{

int sum = 0;

while (number != 0)

{

sum += number % 10;

number /= 10;

}

return sum;

}

}

**Client code :**

import java.io.\*;

import java.net.\*;

import java.util.Scanner;

public class Client

{

public static void main(String[] args)

{

final String serverAddress = "localhost"; // Change to the server's IP if needed

final int serverPort = 12345; // Specify the server's port

try

{

Socket socket = new Socket(serverAddress, serverPort);

System.out.println("Connected to server: " + serverAddress + ":" + serverPort);

BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));

PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);

Scanner scanner = new Scanner(System.in);

System.out.print("Enter a number: ");

String input = scanner.nextLine();

writer.println(input);

String serverResponse = reader.readLine();

System.out.println("Server says: " + serverResponse);

socket.close();

reader.close();

writer.close();

}

catch (IOException e)

{

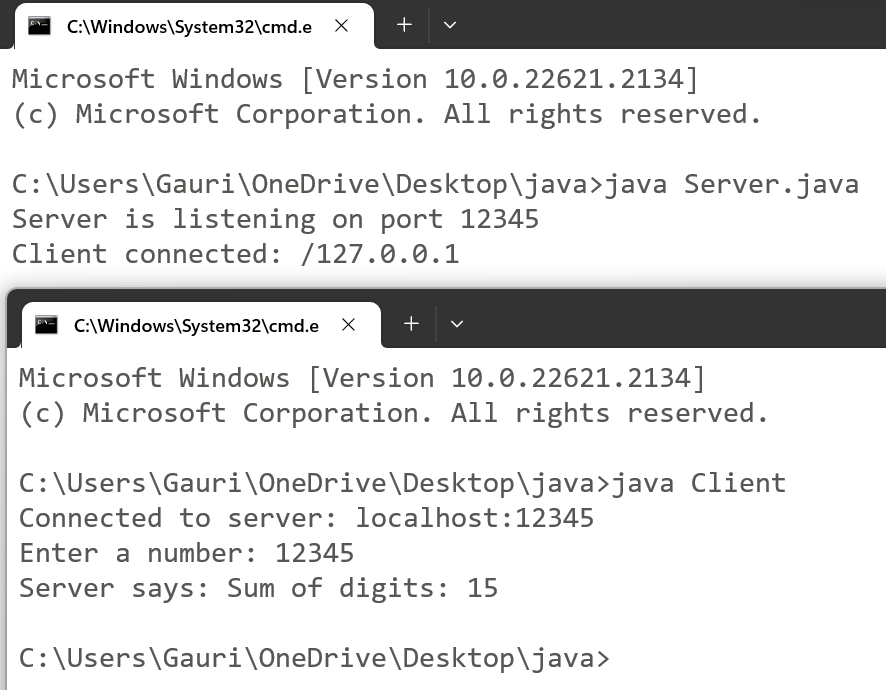
e.printStackTrace();

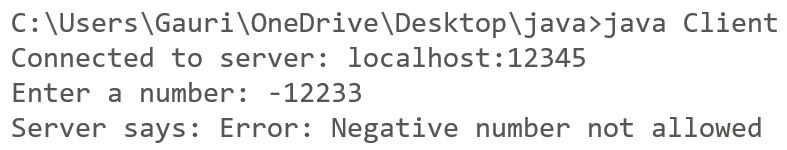
}

}

}

**Output:**

****

****