**Setting up Prerequisites: *Path* and *Classpath***

**PATH is an Environment Variable used to locate JDK Binaries like 'java' or 'javac' Command used to run Java Program and compile Java Source File.**

**CLASSPATH is an Environment Variable used by System or Application ClassLoader to locate and load compile Java Bytecodes stored in the .class File.**

To set **PATH** \*permanently\*:

1. Go to Settings > Control Panel > System > Advanced System Settings > Environment Variable**.**

**Create a Variable: JAVA\_HOME**

Set its path to the latest installed **Java Development Kit** Folder **just short** of the inner **bin** Folder.

Eg: **C:\Program Files\Java\jdk1.8.0\_212**

Click on **Path** Variable > Select **Edit.** Then enter:

**%JAVA\_HOME%\bin OR**

1. Click on **Path** Variable > Select **Edit.** Then, enter: **C:\Program Files\Java\jdk1.8.0\_212\bin;**

To verify, at Command prompt, enter: **javac**

To set **CLASSPATH:**

Click on **Path** Variable > Select **Edit.** Then,

Set its path to the latest installed **Java Runtime Environment** Folder **until** the inner **lib** Folder.

**C:\Program Files\Java\jre1.8.0\_212\lib**

To set both \*temporarily\*:

Enter both the above Paths in the Command Prompt, Enter:

**set (path/classpath)=path-name;**

**PRACTICAL 01**

**Aim**: Port-17 is known as the ‘Quote of the Day Service’. When a Client connects to Port-17 on a Server, the Server responds with a Quote for that Day. Write a Server Program so that it delivers a Quote of the Day. The Quotes should be printable ASCII Characters and should contain fewer than 512 Characters, although multiple Lines are allowed. Since Port-17 is considered well known and therefore unavailable, have your Server listen to Port-6017. Write the Client Code used to read the Quotes returned by the Server.

**Server**

import java.io.\*; import java.net.\*; import java.util.\*; import java.util.Locale; import java.time.LocalDate;

import java.time.format.DateTimeFormatter; import java.time.format.TextStyle; import java.time.DayOfWeek; import java.time.ZonedDateTime; import java.time.ZoneId;

public class P01TCPServer {

private **ServerSocket** SrvrSckt; private **Socket** sckt = null; private **DataInputStream** DIS = null; private **DataOutputStream** DOS = null; private String s = null;

public P01TCPServer (int iPort) {

try {

//------------------------Sending Message back to Client...------------------------------------

// To capture Current Date:

/\* ZonedDateTime.now() retains time zone info. \*/

**ZonedDateTime zdt = ZonedDateTime.now(ZoneId.of("Asia/Kolkata")); s = String.valueOf(zdt);**  **String sDate = new**

**StringBuilder(s.substring(8,10)).append("/").append(s.substring(5,7)).append("/").append(s.su bstring(0,4)).toString();**

System.out.println(sDate);

**DateTimeFormatter DTFormatter = DateTimeFormatter.ofPattern ("dd/MM/yyyy");**

**LocalDate lclDate = LocalDate.parse(sDate, DTFormatter);**

/\* Extracts a `DayOfWeek` enum object.\*/  **DayOfWeek dow = lclDate.getDayOfWeek(); s = String.valueOf(dow);**

// String output = dow.getDisplayName(TextStyle.SHORT, Locale.US); // String = Tue

String sQuote = null;

switch(s)

{

case "SUNDAY":

sQuote = "Eagles love a Sunday.";

break; // break is optional

case "MONDAY":

sQuote = "Tigers love a Monday.";

break;

case "TUESDAY":

sQuote = "Lions love a Tuesday."; break;

case "WEDNESDAY":

sQuote = "Elephants love a Wednesday."; break;

case "THURSDAY":

sQuote = "Rats love a Thursday.";

break;

case "FRIDAY":

sQuote = "Guinea-Pigs love a Friday.";

break;

case "SATURDAY":

sQuote = "Dragons love a Saturday.";

break;

default :

}

// Part-1: Since this is a Server, create a Server Socket.

**SrvrSckt = new ServerSocket(iPort);**

System.out.println("Server started.");

System.out.println("Waiting for a Client Request...");

// Part-2: Accept the Client-Connection using the Server Socket on a Basic Socket. /\* accept( ) is a blocking call that will wait for a client to initiate communications and then ServerSocket returns with a normal Socket that is then used for communication with the client. \*/

**sckt = SrvrSckt.accept();**

System.out.println("Client accepted.");

// Part-3: Set up Output-Stream and Input-Stream using the Basic Socket. **DOS = new DataOutputStream(sckt.getOutputStream());**

**DIS = new DataInputStream (new BufferedInputStream**

**(sckt.getInputStream()));**

String sReply = "";

try {

// Part-4: Read the Input-Stream.  **sReply = DIS.readUTF();**

StringBuilder SB = new StringBuilder();

SB.append("Request from Client: ").append(sReply);

System.out.println(SB);

}

catch(IOException e) {System.out.println(e);}

if (sReply.length() > 0) {

try {

// Part-5: Write the Output-Stream.

**DOS.writeUTF(sQuote);**

System.out.println("Quote Sent.");

}

catch(IOException e) {System.out.println(e);}

} // IF

// Part-6: Close all Connections.

System.out.println("Closing All Connections...");

sckt.close();

DIS.close();

DOS.close();

System.out.println("Closed.");

}

catch(IOException i) { System.out.println(i); }

}

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

P01TCPServer srvr = new P01TCPServer(**6017**);

}

}

**Client**

import java.io.\*;

import java.net.\*;

public class P01TCPClient {

private **Socket** sckt = null; private **DataOutputStream** DOS = null; private **DataInputStream** DIS = null;

//Setting up Input from USER

private BufferedReader BR = new BufferedReader (new InputStreamReader (System.in));

private String sQuote = "";

public P01TCPClient(String sAddr, int iPort) {

// Part-1: Create a Basic Socket.

try {

**sckt = new Socket(sAddr, iPort);**  System.out.println("Connected.");

// Part-2: Create a Output-Stream using the Basic Socket.

**DOS = new DataOutputStream(sckt.getOutputStream());**

}

catch (UnknownHostException e) { System.out.println(e);

}

catch (IOException e) {

System.out.println(e);

}

// Part-3: Receive Input from User String sReply = null;

System.out.println("Do you want 'Quote Of The Day' ?: "); try {

sReply=BR.readLine();

if (sReply.equals("y") || sReply.equals("Y")) { // Part-4: Send Output-Stream to the Server.

**DOS.writeUTF("y");**

}

// Part-5: Receive Input-Stream from Server and print it.

**DIS = new DataInputStream (new BufferedInputStream**

# (sckt.getInputStream()));

sQuote = DIS.readUTF();

StringBuilder SB = new StringBuilder();

SB.append("Quote of the Day: ").append(sQuote);

System.out.println(SB);

} catch(IOException e) {

System.out.println(e);

}

// Part-6: Close all Connections. try {

System.out.println("Closing all Connections...");

DIS.close(); DOS.close(); sckt.close();

BR.close();

System.out.println("Closed.");

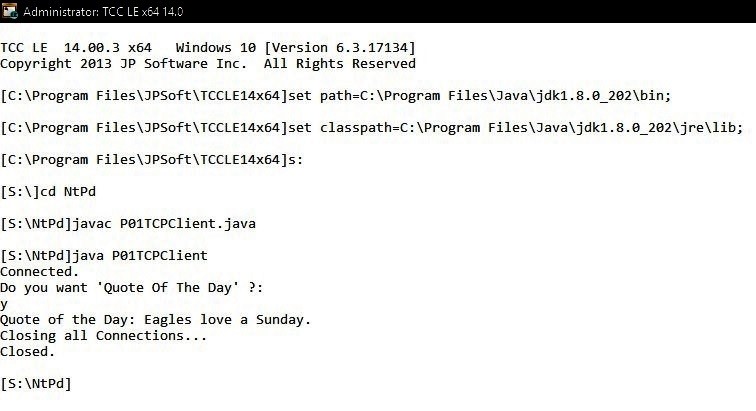
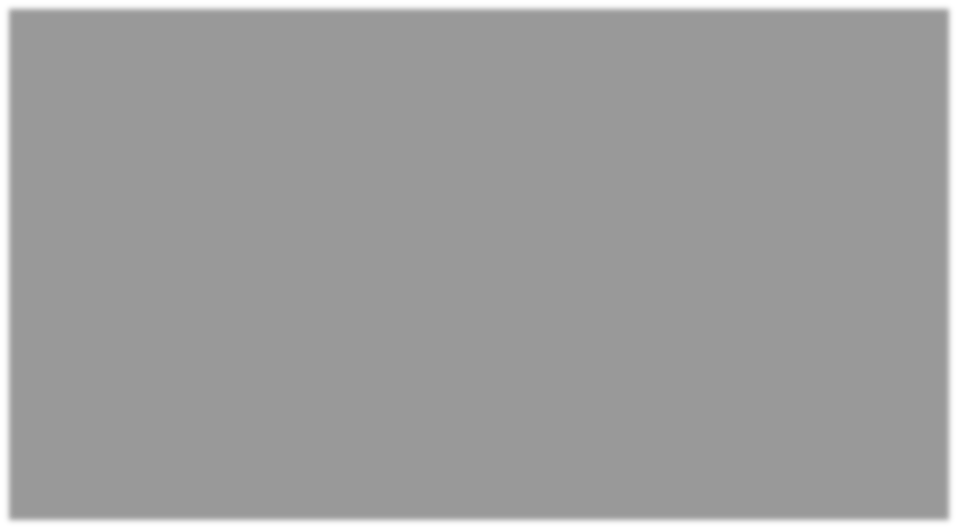
}

catch (Exception e) {System.out.println(e); }

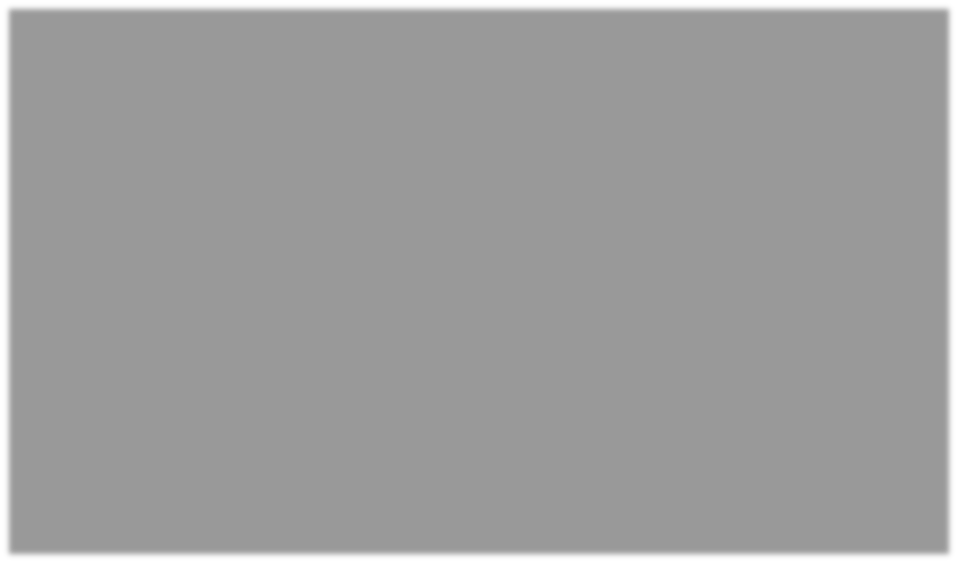
} // CONSTRUCTOR

public static void main(String[] args) throws UnknownHostException, IOException { P01TCPClient C = new P01TCPClient("127.0.0.1",6017);

} }



**Fig 1.2: Client-Side Execution**



**Fig 1.1: Server-Side Execution**

**PRACTICAL 02**

**Aim**: Write a Client–Server Application using Java Sockets that allows a Client to write a Message (as a String) to a Socket. A Server will read this message, count the Number of Characters and Digits in the Message, and send these two Counts back to the Client. The Server will listen to port 6100. The client can obtain the String Message that it is to pass to the Server either from the Command Line or by using a Prompt to the User. One Strategy for sending the two Counts back to the Client is for the Server to construct an Object containing:

The Message it receives from the Client

A Count of the Number of Characters in the Message

A Count of the Number of Digits in the Message.

**Class: *MyMessage.java***implements**Serializable**

import java.io.Serializable;

// We need to transfer a *'MyMessage'* **object** from client to server. So our *MyMessage*.java should implement the **Serializable** interface.

*public class* ***MyMessage*****implements****Serializable** *{*

**private static final long serialVersionUID = 777L;**

protected String sMsg; private int iTtlCharacters;

private int iTtlIntegers;

MyMessage() {}; MyMessage(String m) { this.sMsg = m;

}

MyMessage(String m, int tc, int ti) { this.sMsg = m;

this.iTtlCharacters = tc;

this.iTtlIntegers = ti;

}

public boolean equals(Object o) { if (this == o) return true;

if (o == null || getClass() != o.getClass()) return false; return true;

}

@Override

public String toString() {

StringBuilder SB = new StringBuilder();

SB.append(" Message:").append(sMsg).append("\n Total Characters:

").append(iTtlCharacters).append("\n Total Integers: ").append(iTtlIntegers); return SB.toString();

}

}

**Server**

import java.io.\*; import java.net.\*;

public class TCPServer {

private ServerSocket SrvrSckt; private Socket sckt = null; private **ObjectInputStream** OIS = null; private **ObjectOutputStream** OOS = null;

public TCPServer (int iPort) {

try {

// Part-1: Since this is a Server, create a Server Socket.  **SrvrSckt = new ServerSocket(iPort);**

System.out.println("Server started.");

System.out.println("Waiting for a Client Request...");

// Part-2: Accept the Client-Connection using the Server Socket on a Basic Socket.

**sckt = SrvrSckt.accept();**

System.out.println("Client accepted.");

// Part-3: Set up Object Input-Stream using the Basic Socket.

**OIS = new ObjectInputStream(sckt.getInputStream());**

try {

// Part-4: Receive the Object from the Client.

**MyMessage ReceivedMsg = (MyMessage) OIS.readObject();**

// Part-5: Extract the Message Part from the Object and process it.

**String sMsg = ReceivedMsg.sMsg;** /\* <-- For accessing the actual

Message, make sure it's Variable is **not** declared private in it's Definition. \*/

StringBuilder SB = new StringBuilder();

SB.append("Message received: ").append(sMsg);

System.out.println(SB);

System.out.println("Processing it..."); int iTtlChrctrs = 0;

int iTtlDigits = 0; for (int m=0; m<sMsg.length();m++) {

Character x = sMsg.charAt(m);

// System.out.println(Character.getNumericValue(x)); if (!x.equals(' ')) {

iTtlChrctrs ++;

if (Character.isDigit(x)) { iTtlDigits ++;

} //IF

} // IF

} // FOR

// Part-6: Create Object to be sent to the Client

MyMessage SentMsg = new MyMessage (sMsg, iTtlChrctrs,

iTtlDigits);

// Part-7: Set up Output-Stream for Object to be sent using the Basic

Socket.

**OOS = new ObjectOutputStream(sckt.getOutputStream());**

// Part-8: Send the Object to the Client

**OOS.writeObject(SentMsg);**

System.out.println("Processed Data sent back Successfully.");

}

catch (ClassNotFoundException e) {e.printStackTrace(); } catch (SocketException e) { e.printStackTrace(); } catch(IOException e) { e.printStackTrace(); }

// Part-9: Close all Connections.

System.out.println("Closing all Connections..."); sckt.close();

OIS.close();

OOS.close();

System.out.println("Closed.");

}

catch(IOException i) { System.out.println(i); }

}

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

TCPServer srvr = new TCPServer(6100);

}

}

**Client**

import java.io.\*;

import java.net.\*;

public class TCPClient {

private **Socket** sckt = null; private **ObjectInputStream** OIS = null; private **ObjectOutputStream** OOS = null;

//Setting up Input from USER

private BufferedReader BR = new BufferedReader (new InputStreamReader (System.in));

public TCPClient(String sAddr, int iPort) {

// Part-1: Creating a Socket, Setting up Output. try {

**sckt = new Socket(sAddr, iPort);**  System.out.println("Connected.");

}

catch (UnknownHostException e) {

System.out.println(e);

}

catch (IOException e) {

System.out.println(e);

}

// Part-2: Receive Input from User. String sLine = "";

try {

**sLine = BR.readLine();**

}

catch(IOException e) {

System.out.println(e);

}

try {

// Part-3: Create an Output-Stream using the Basic Socket.

**OOS = new ObjectOutputStream (sckt.getOutputStream());**  // Part-4: Create a Message Object using the User-Input and write it to the

Server using the Output-Stream.

System.out.println("Sending Message to the Server...");

MyMessage MyMsg = new MyMessage (sLine);

**OOS.writeObject(MyMsg);**

// Part-5: Create an Input-Stream using the Basic Socket.

System.out.println("Receiving processed Data from the Server...");

**OIS = new ObjectInputStream(sckt.getInputStream());**

// Part-6: Receive the Object from the Server using the Input-Stream.  **MyMessage ReceivedMsg = (MyMessage) OIS.readObject();**

// Part-7: Read/Process the Object.

# StringBuilder SB = new StringBuilder();

**SB.append("Object received: ").append(ReceivedMsg);**

**System.out.println(SB);** }

// thrown if Input-Stream fails to read an Object.

catch (ClassNotFoundException e) { System.out.println(e);} // thrown for any Failure in creating every Input/Output Stream.

catch(IOException e) { System.out.println(e);}

// Part-8: Close all Connections. try {

System.out.println("Closing all Connections..."); sckt.close(); BR.close();

OIS.close();

OOS.close();

System.out.println("Closed.");

}

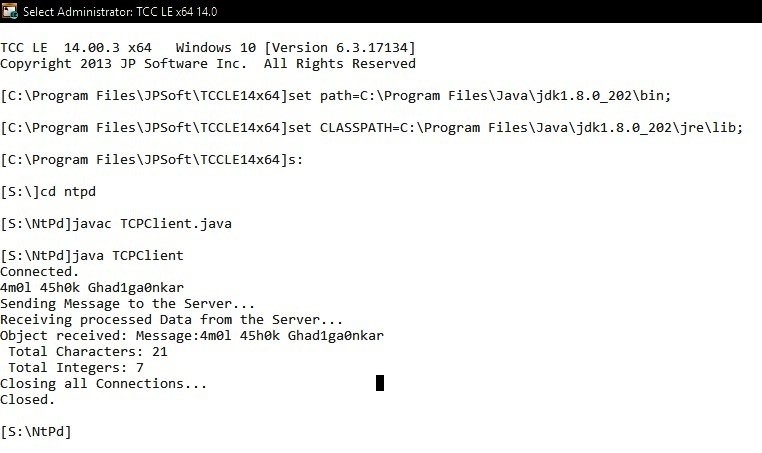
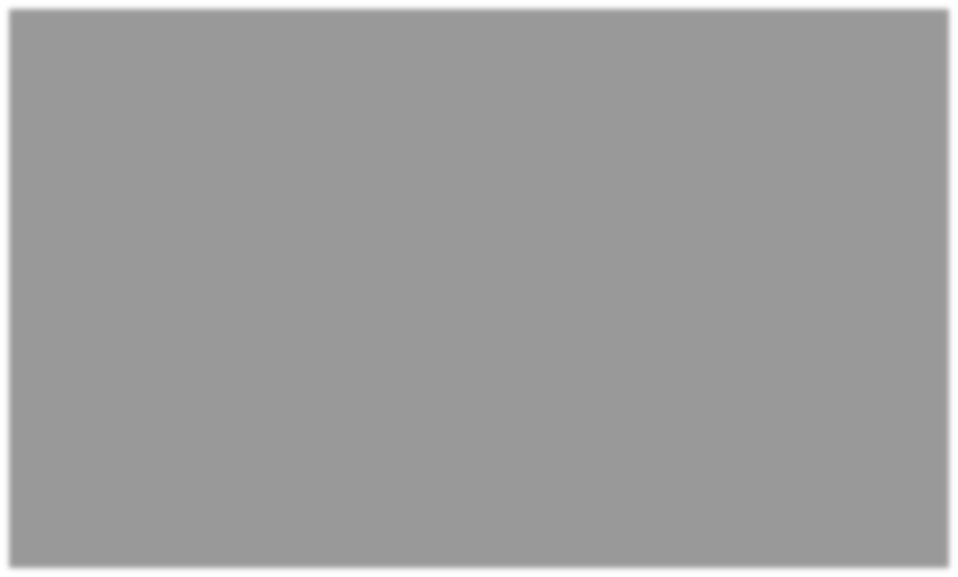
catch (Exception e) { System.out.println(e); }

} // CONSTRUCTOR

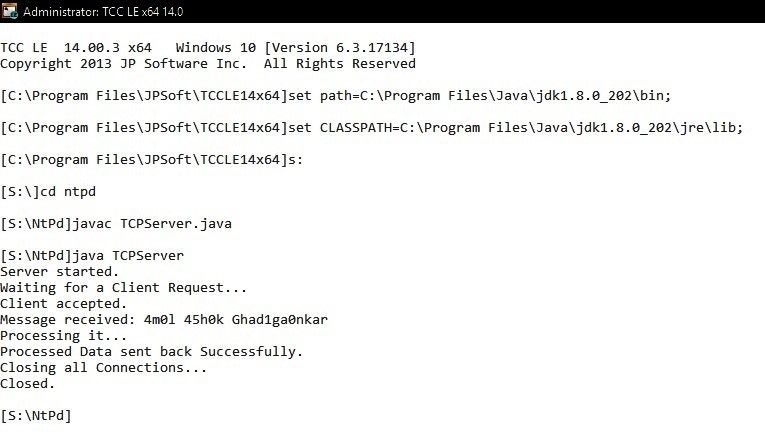
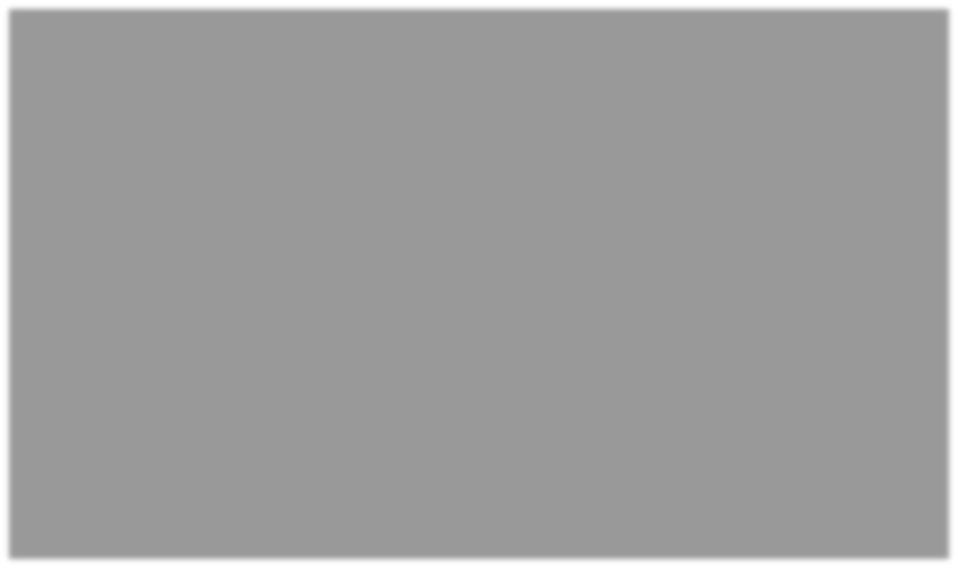
public static void main(String[] args) throws UnknownHostException, IOException { TCPClient C = new TCPClient("127.0.0.1",6100);

}

}



**Fig 2.2: Client-Side Execution**



**Fig 2.1: Server-Side Execution**

**Practical-03**

**Aim**: Write a *multithreaded* Java Program that outputs Prime Numbers. This Program should work as follows: The User will run the Program and will enter a Number on the Command Line. The Program will then create a separate Thread that outputs all the Prime Numbers less than or equal to the Number entered by the User.

## Class: PrimeNums\_1.java: extends Class Thread

public class PrimeNums\_1 extends Thread {

private int iNum;

boolean iNumIsAPrime = true;

public PrimeNums\_1 (int N) {

this.iNum=N;

start();

}

public void run() {

StringBuilder SB = new StringBuilder("\n THREAD-based Class for ");

SB.append(iNum).append(" has Begun.");

System.out.println(SB);

SB = new StringBuilder("\n THREAD-based Class: Prime Numbers lesser than or equal to ");

SB.append(iNum).append(": 2");

System.out.println(SB);

SB = new StringBuilder(); for (int i=2; i<=iNum; i++) {

iNumIsAPrime = true;

for (int j=2; j<iNum-1; j++) { if (i%j ==0 && i != j) { iNumIsAPrime = false; break;

} } // FOR - j

if (iNumIsAPrime == true) { if (i!=2) {

SB = new StringBuilder();

SB.append(", ").append(i);

System.out.println(SB);

}

} } // FOR - i

SB = new StringBuilder("\n THREAD-based Class Thread for ");

SB.append(iNum).append(" has exited.");

System.out.println(SB);

}

}

**Class: *PrimeNums\_2.java*:** implements **Interface *Runnable*.**

public class PrimeNums\_2 implements Runnable {

Thread Thrd; private int iNum;

boolean iNumIsAPrime = true;

public PrimeNums\_2 (int N) {

this.iNum=N;

Thrd = new Thread (this, "PrimeNumCalc"); Thrd.start();

}

public void run() {

StringBuilder SB = new StringBuilder("\n RUNNABLE Class Thread for "); SB.append(iNum).append(" has Begun.");

System.out.println(SB);

SB = new StringBuilder("\n RUNNABLE-based Class: Prime Numbers lesser than or equal to ");

SB.append(iNum).append(": 2");

System.out.println(SB);

SB = new StringBuilder(); for (int i=2; i<=iNum; i++) {

iNumIsAPrime = true;

for (int j=2; j<iNum-1; j++) { if (i%j ==0 && i != j) { iNumIsAPrime = false;

break;

} } // FOR - j

if (iNumIsAPrime == true) { if (i!=2) {

SB = new StringBuilder();

SB.append(", ").append(1);

System.out.println(SB);

}

} } // FOR - i

SB = new StringBuilder("\n RUNNABLE Class Thread for ");

SB.append(iNum).append(" has exited.");

System.out.println(SB);

}

}

**Class: *PrimeNums\_3.java*:** implements **Interface *Callable*.**

import java.util.concurrent.\*;

public class PrimeNums\_3 implements **Callable**<String> {

private int iNum;

private static StringBuilder sRslt = new StringBuilder(); boolean iNumIsAPrime = true;

public PrimeNums\_3 (int N) {

this.iNum=N;

}

// Call Method MUST return some Object. HERE: String public String call() throws Exception {

StringBuilder SB = new StringBuilder("\n CALLABLE Class Thread for ");

SB.append(iNum).append(" has Begun.");

System.out.println(SB);

SB = new StringBuilder("\n CALLABLE Class Thread: Prime Numbers lesser than or equal to ");

SB.append(iNum).append(": 2");

System.out.println(SB);

for (int i=2; i<=iNum; i++) {

iNumIsAPrime = true;

for (int j=2; j<iNum-1; j++) { if (i%j ==0 && i != j) { iNumIsAPrime = false;

break;

} } // FOR - j

if (iNumIsAPrime == true) { if (i!=2) {

sRslt.append(", ").append(i);

}

} } // FOR - i

SB = new StringBuilder("\n CALLABLE Class Thread for ");

SB.append(iNum).append(" has exited.");

System.out.println(SB);

return sRslt.toString();

}

}

**Main Class: *ThreadsDemo.java*: Main Thread**

import java.io.\*;

import java.util.concurrent.\*;

public class **ThreadsDemo** {

private static **ExecutorService** ES;

public static void main (String[] args) {

System.out.println("Main Thread: Finding Prime Numbers: ");

System.out.print("\n\n-------------- BOTH (Runnable Interface & Class Thread: based) Threads by CALLING A Join() on them [Ex: Numbers 47, 41]----------------------"); // create an Object of a Class extending Thread Class. PrimeNums\_1 ExThrd = new PrimeNums\_1(47);

// create an Object of a Class implementing Runnable Interface.

PrimeNums\_2 RnblePN1 = new PrimeNums\_2(41);

try {

**/\* NOTE: join() method ensures that the MAIN Method finishes last by waiting for the Child Threads to finish first. \*/**

**/\* 'PrimeNums\_1' Class, DIRECTLY extends the Thread Class. Hence the 'join' Method is DIRECTLY to be called on the**

**'PrimeNums\_1' itself. \*/**

ExThrd.**join()**;

**/\* In 'PrimeNums\_2' Runnable Class, a Thread Object is explicitly created. Hence the 'join' Method is to be called on that Object. \*/**

RnblePN1.***Thrd*.join()**;

}

catch (InterruptedException e) { e.printStackTrace();

}

//--------------------------------------------------------------------------

System.out.print("\n\n-------------- BOTH Threads implicitly WITHOUT CALLING

A Join() on them [Ex: 37, 31] ------------------");

// calls a Class extending Thread Class. new PrimeNums\_1(37);

// calls a Class implementing Runnable Interface. new PrimeNums\_2(31);

//--------------------------------------------------------------------------

System.out.print("\n\n---------------- RUNNABLE Interface: 1. with LAMBDA

Expression [Ex: 11] ----------------------------------");

**Runnable RnbleIntrfc= () -> {**

System.out.print("\n RUNNABLE Interface + LAMBA

Expression: 11: has Begun.");

System.out.print("\n RUNNABLE Interface + LAMBA Expression:

Prime Numbers lesser than or equal to 11: 2");

boolean iNumIsAPrime = true; for (int i=2; i<=11; i++) {

iNumIsAPrime = true;

for (int j=2; j<11-1; j++) { if (i%j ==0 && i != j) { iNumIsAPrime = false;

break;

}

}

if (iNumIsAPrime == true) { if (i!=2) {

System.out.print(", " + i);

}

}

}

System.out.print("\n\n RUNNABLE Interface + LAMBA Expression: 11: ...Done."); **}; // *rPrimeNums* ends**

**new Thread(RnbleIntrfc).start();**

//--------------------------------------------------------------------------

System.out.print("\n\n---------------- CALLABLE Interface: 1. with LAMBDA

Expression [Num: 13] ----------------------------------");

**Callable CllblIntrfc = () -> {**

System.out.print("\n CALLABLE Interface + LAMBA Expression: 13: has Begun.");

String sRslt = ""; boolean iNumIsAPrime = true; for (int i=2; i<=13; i++) {

iNumIsAPrime = true;

for (int j=2; j<13-1; j++) {

if (i%j ==0 && i != j) { iNumIsAPrime = false; break;

}

}

if (iNumIsAPrime == true) { if (i!=2) {

sRslt = sRslt + ", " + i;

}

} }// FOR

**return *sRslt***;

};

try {

System.out.print("\n\n CALLABLE Interface + LAMBDA Expression:

Prime Numbers lesser than or equal to 13: 2" + **CllblIntrfc.call()**);

}

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

System.out.print("\n\n-------------- CALLABLE Interface: with Future and

ExecutorService ----------------------");

// create an Object of a Class implementing CALLABLE Interface.

PrimeNums\_3 ClblePN = new PrimeNums\_3 (29);

**Future<**String**>**FT1 =

**Executors.newSingleThreadExecutor().submit(*ClblePN*);**

try {

String sOP = FT1.**get()**;

System.out.println("\n Future calculated for Number 29 with RUNNABLE: "+sOP);

}

catch (InterruptedException e) {e.printStackTrace();} catch (ExecutionException e) {e.printStackTrace();}

System.out.print("\n\n-------------- RUNNABLE Interface: 2. with FutureTask and ExecutorService [Ex: 23, 19] ----------------------");

// create Two Objects of a Class implementing RUNNABLE Interface.

RnblePN1 = new PrimeNums\_2(23); PrimeNums\_2 RnblePN2 = new PrimeNums\_2(19); // FutureTask used with RUNNABLE Interface.

**FutureTask <**String**>** FT3 = **new FutureTask** <String**>(*RnblePN1*,**

"\nFutureTask for Number "+ 23 + " with RUNNABLE: Done.");

**FutureTask** **<**String**>** FT4 **= new FutureTask<**String**>(RnblePN2**,

"\nFutureTask for Number "+ 19 + " with RUNNABLE: Done.");

//

**ExecutorService ES = Executors.newFixedThreadPool(2);**

**ES.submit(**FT3);

ES.submit(FT4);

while (true) { try {

**if (FT3.isDone() && FT4.isDone()) {**

System.out.println("Both FutureTasks Complete.");  **ES.shutdown();** // shut down executor service return;

}

if (!FT3.isDone()) {

// wait indefinitely for future task to complete

System.out.println("FT3 Output: " + **FT3.get()**);

}

System.out.println("Waiting for FT4 to complete...");

// Wait if necessary for the computation to complete, and then retrieves its result  **String s = FT4.get(250, TimeUnit.MILLISECONDS);**

if (s != null) {

System.out.println("FT4: Output: " + s);

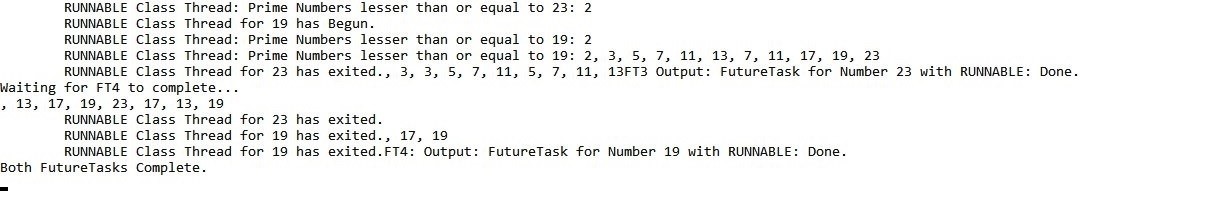
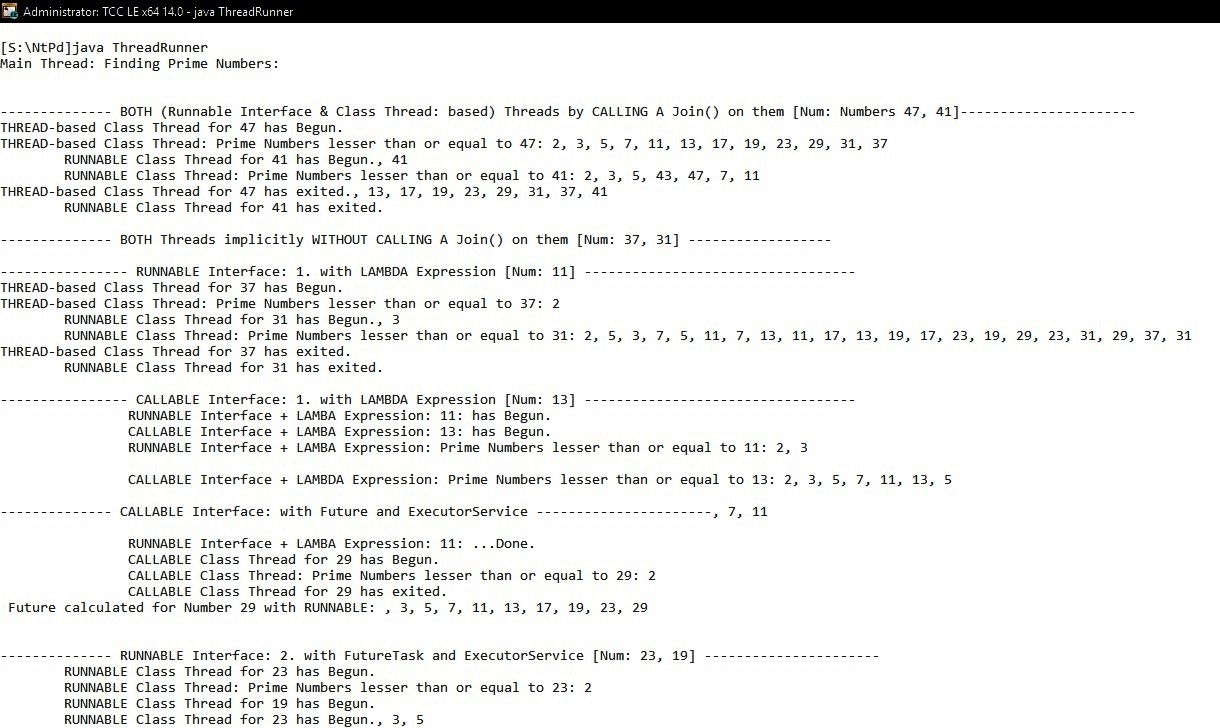
}

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

} // WHILE

} // main

}



**Fig 3.1: Running a Thread in various Ways.**

**PRACTICAL 04**

**Aim**: Servers can be designed to limited the Number of Open Connections. A Server may wish to have only N Connections open at any Point in Time. After N Connections have been made, the Server will not accept another incoming Connection until an existing Connection is released. Write a Java Program to demonstrate this Scenario.

**Class: *RnblClss.java*:** implements **Interface *Runnable*.**

import java.io.\*; import java.net.\*; import java.util.\*; import java.util.Locale; import java.util.concurrent.\*;

public class RnblClss implements Runnable {

Thread Thrd;

private ServerSocket SrvrSckt;

private Socket sckt = null;

private static int iFreeThrdsCtr = 0; private static int iBusyThrdsCtr = 0; private static int iTTLThrdsCtr = 0;

private DataInputStream DIS = null; private DataOutputStream DOS = null;

private int iNum; String sRslt ="2"; boolean iNumIsAPrime = true;

private static StringBuilder SB = new StringBuilder();

public RnblClss (ServerSocket SS) {

// Part-1: Since this is a Server, create a Server Socket. this.SrvrSckt = SS;

Thrd = new Thread (this, "MyService");

iTTLThrdsCtr++;

if (iFreeThrdsCtr<0) { iFreeThrdsCtr = 1; } else { iFreeThrdsCtr++; }

SB = new StringBuilder("\nWaiting for a Client Request: ");

SB.append(iFreeThrdsCtr).append("...");

System.out.println(SB);

Thrd.start();

}

public void run() {

//System.out.print("\n RUNNABLE Class Thread for " + iNum + " has

Begun.");

try {

// Part-2: Accept the Client-Connection using the Server Socket on a Basic Socket. /\* accept( ) is a blocking call that will wait for a client to initiate communications and then ServerSocket returns with a normal Socket that is then used for communication with the client. \*/

sckt = SrvrSckt.accept(); iBusyThrdsCtr++; iFreeThrdsCtr--;

if (iFreeThrdsCtr < 0) {iFreeThrdsCtr = 0;}

System.out.println("\nClient accepted: " + iBusyThrdsCtr);

System.out.println("Currently free Connection(s): "+ iFreeThrdsCtr);

// Part-3: Set up Data Input-Stream using the Basic Socket.

DIS = new DataInputStream (new BufferedInputStream (sckt.getInputStream()));

// Part-4: Receive the Data from the Client and Process it. iNum = Integer.parseInt(DIS.readUTF());

System.out.println("Number from Client "+iBusyThrdsCtr+ ": " + iNum);

for (int i=2; i<=iNum; i++) {

try {

Thread.sleep(1000);

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

iNumIsAPrime = true;

for (int j=2; j<iNum-1; j++) { if (i%j ==0 && i != j) { iNumIsAPrime = false;

break;

}

} if (iNumIsAPrime == true) { if (i!=2) {

sRslt = sRslt + ", " + i;

}

} } // FOR

// Part-5: Write the Output-Stream to the Client.

DOS = new DataOutputStream(sckt.getOutputStream());

DOS.writeUTF(sRslt);

System.out.println("\nResult Sent to Client: "+ iBusyThrdsCtr + ".");

// Part-6: Close all Connections.

sRslt ="2"; DIS.close(); DOS.close(); iBusyThrdsCtr--;

iTTLThrdsCtr--;

System.out.println("SOCKET CLOSED. Total RUNNING Connections: " + iTTLThrdsCtr);

sckt.close();

}

catch (SocketException se) {System.out.println("\n Unable to provide Service for Client: " + iBusyThrdsCtr + " until Next Free Connection is available.");

}

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

} // RUN

}

**Server**

import java.io.\*; import java.net.\*; import java.util.\*; import java.util.Locale;

import java.util.concurrent.\*;

public class P04Server {

private static ServerSocket SS; private static **ExecutorService** ES; private static **FutureTask <String>** FT; private static int iTTLThrdsCtr = 0; private static StringBuilder SB = new StringBuilder();

public void P04Server() {}

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

SS = new ServerSocket(6017);

SS = new ServerSocket(6017);

SB.append("Listening for connection...").append(SS.getLocalPort());

System.out.println(SB);

RnblClss[] Tasks = new RnblClss[3];

**FutureTask[]** FtrTsks = new FutureTask[3];

Tasks[0] = new RnblClss(SS); Tasks[1] = new RnblClss(SS);

Tasks[2] = new RnblClss(SS);

// FutureTask used with RUNNABLE Interface.

**FtrTsks[0] = new FutureTask <String>(Tasks[0], "FutureTask 1 RUNNABLE: Done.");**

**FtrTsks[1] = new FutureTask <String>(Tasks[1], "FutureTask 2 RUNNABLE: Done.");**

**FtrTsks[2] = new FutureTask <String>(Tasks[2], "FutureTask 3 RUNNABLE: Done.");**

**ES = Executors.newFixedThreadPool(3);**

**ES.submit(FtrTsks[0]);**

**ES.submit(FtrTsks[1]);**

**ES.submit(FtrTsks[2]);**

**while (true) { for (int i=0; i<=2; i++) {**

**if (FtrTsks[i].isDone()) { Tasks[i] = new RnblClss(SS);**

**SB.append("FutureTask ").append(i).append("**

# RUNNABLE: Done.");

**FtrTsks[i] = new FutureTask <String>(Tasks[i], SB);**

**ES.submit(FtrTsks[i]);**

**} // IF**

**} // FOR**

**}// WHILE**

} // MAIN

}

**Client**

import java.io.\*; import java.net.\*;

public class P04Client {

private Socket sckt = null; private **DataOutputStream** DOS = null; private **DataInputStream** DIS = null; private String sReply = "";cd

private BufferedReader BR = new BufferedReader (new InputStreamReader (System.in));

public P04Client(String sAddr, int iPort) {

// Part-1: Creating a Socket, Setting up Output.

try {

sckt = new Socket(sAddr, iPort); System.out.println("Connected.");

}

catch (UnknownHostException e) {e.printStackTrace();} catch (IOException e) {e.printStackTrace();}

int iNum = 0;

System.out.print("Enter a Number: "); // Part-2: Receive Input from User.

try {

iNum =Integer.parseInt(BR.readLine());

}

catch(IOException e) { iNum = 10;

e.printStackTrace();

}

try {

// Part-3: Create an Output-Stream using the Basic Socket.

**DOS = new DataOutputStream (sckt.getOutputStream());**  // Part-4: Create a Message Object using the User-Input and write it to the Server using the Output-Stream.

**DOS.writeUTF(Integer.toString(iNum));**

System.out.println("Number sent to the Server.");

// Part-5: Receive Input-Stream from Server and print it.

**DIS = new DataInputStream (new BufferedInputStream**

**(sckt.getInputStream()));**

**sReply = DIS.readUTF();**

System.out.println("RESULT: "+ sReply);

}

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

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

// Part-8: Close all Connections.

try {

System.out.println("Closing all Connections..."); sckt.close(); BR.close();

DIS.close();

DOS.close();

System.out.println("Closed.");

}

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

} // CONSTRUCTOR

public static void main(String[] args) throws UnknownHostException, IOException { P04Client C = new P04Client("127.0.0.1",6017);

}

}

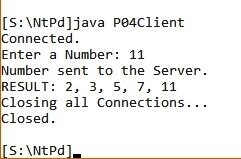
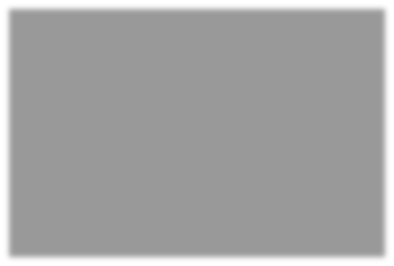
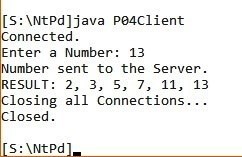
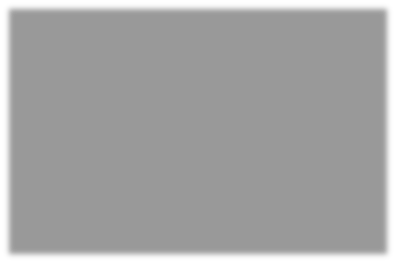
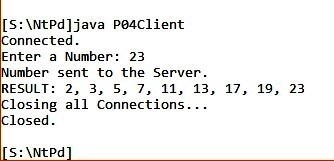
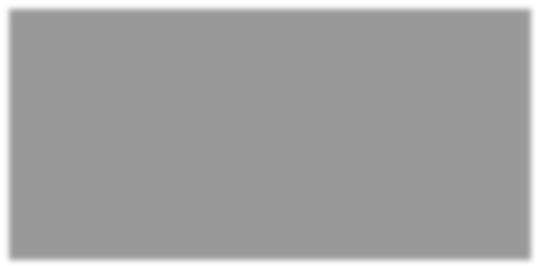
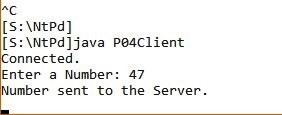
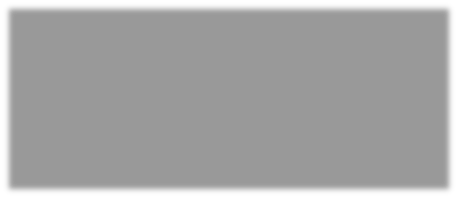


Fig 4.1: **Client** Connections Requests. **NOTE**: First Request (Number: 47) was **not** serviced because Connections were limited to just 3.

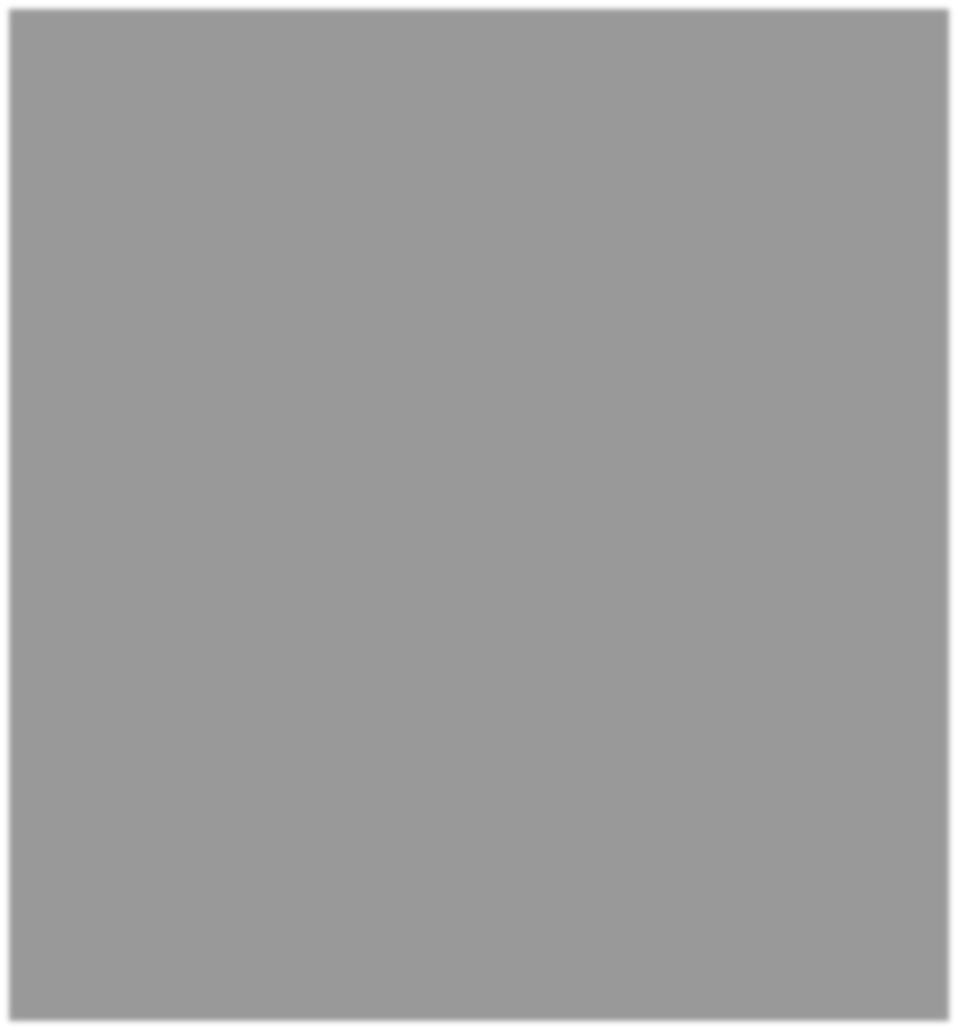


Fig 4.2: **Server** serving Requests and then waiting for more…

**PRACTICAL 05**

**Aim:** Assuming that a System has a 32-bit Virtual Address, write a Java Program that is passed: 1. Page-Size 2. the Virtual Address. The Page Number and the Offset of the given Virtual Address for the given Page-Size should be reported. Page-Size must be in Power of 2 and in the Range of 1024 and 16384.

## Class: AddrMapping

package NtPd;

import java.io.\*; import java.util.List; import java.util.ArrayList; import java.util.Arrays;

import java.util.Random;

public class AddrMapping {

private static BufferedReader BR = new BufferedReader (new InputStreamReader (System.in)); private static String sLine=""; private static String Words[]=null;

private static int iProduct = 1;

private static boolean IsPowerOfTwo = false;

private static int **iVirAddr** = 0; private static int **iPageSize** = 4; private static int **iVirPageIdx** = 0;

private static int **iOffset** = 0;

private static ArrayList<Integer> **iVirAddresses** = new ArrayList<Integer>();

private static String[] **LogicalMemory** = {"a","b","c",

"d","e","f","g","h","i","j","k","l","m","n","o","p","q","r","s","t","u","v","w","x"};

private static int[][] **PageTable**;

private static String[] **sData** = {"African", "Bonelli", "Crested", "Dragon", "Eastern", "Flyer", "Golden", "Harpy", "Indian", "Jaguar", "Kite", "Little", "Martial", "Nucleus", "Ornate", "Phillipine", "Queen", "Rain","Steppe", "Tawny", "Undertow", "Verreaux", "Whalberg", "Xenology"};

private static String[] **PhysicalMemory** = new String[24];

public static void main(String[] args) {

try {

System.out.print ("\n Enter Page-Size (2/4/8): "); iPageSize = Integer.parseInt(BR.readLine());

while (iPageSize > iProduct) { iProduct = iProduct \* 2; if (iPageSize == iProduct) {

IsPowerOfTwo = true;

break;

}

if (iPageSize < iProduct) { break;

}

}

if (!IsPowerOfTwo) {

System.out.println (" Page-Size is Invalid.");

} else {

System.out.println ("\n LOGICAL MEMORY DATA: "); System.out.print(" ");

StringBuilder SB = new StringBuilder(); for (String a : LogicalMemory) {

SB.append(a).append(" ");

}

System.out.print(SB);

int k;

List<Integer> LgclPgPos = new ArrayList<Integer>();

for (k=0; k< LogicalMemory.length; k+=iPageSize) {

LgclPgPos.add(k);

}

System.out.println ("\n\n POPULATING PHYSICAL MEMORY (RAM) - PAGE-WISE - RANDOMLY...");

Random Rndm = new Random(); int TtlPos = LgclPgPos.size(); int LgclPos = 0;

for (int l =0; l <TtlPos; l++) { int SomeIdx = Rndm.nextInt(LgclPgPos.size()); int PhyPgPos = LgclPgPos.get(SomeIdx); for (int p=PhyPgPos; p<(PhyPgPos+iPageSize); p++) {

PhysicalMemory[p] = sData[LgclPos];

LgclPos++;

}

LgclPgPos.remove(SomeIdx);

}

System.out.println (" DONE.");

System.out.println("\n PHYSICAL MEMORY - PAGE-WISE:");

for (int m=1; m<=PhysicalMemory.length; m++) { System.out.print(" ");

if (m%iPageSize != 0) {

System.out.print(m+ " - " + PhysicalMemory[m-1] + " ");

} else {

System.out.println(m + " - " + PhysicalMemory[m-1] + " ");

}

}

System.out.println ("\n GENERATING PAGE-TABLE AND IT'S ENTRIES BASED ON THE PAGE-SIZE...");

int TtlRows = LogicalMemory.length/iPageSize; PageTable = new int[TtlRows][2];

int LgclPageNum = 0;

for (int l=0; l<LogicalMemory.length; l+=iPageSize) { String s = LogicalMemory[l];

for (int p=0; p<=LogicalMemory.length; p++) {

if(s.equalsIgnoreCase(Character.toString(PhysicalMemory[p].charAt(0)))) {

PageTable[LgclPageNum][0] = LgclPageNum; PageTable[LgclPageNum][1] = p/iPageSize;

break;

}

}

LgclPageNum++;

} // FOR

System.out.println (" DONE.");

System.out.println ("\n PAGE TABLE: ");

System.out.println (" LogicaL Page Index\tPhysical Page Index");

System.out.println ("-------------------\t-------------------");

SB = new StringBuilder();

for (int t = 0; t< PageTable.length; t++) {

SB.append("\t").append(PageTable[t][0]).append("\t\t\t").append(PageTable[t][1]);

}

System.out.println ("\n Enter Logical (Virtual) Addresses (between 1 and 24): ");

for (sLine=BR.readLine(); sLine!=null; sLine=BR.readLine()) {

Words = sLine.split(" ");

iVirAddresses = getIntegerArray(new ArrayList<>(Arrays.asList(Words))); System.out.print ("\n PHYSICAL MAPPING: "); for (int iVirAddr : iVirAddresses) {

iVirPageIdx = iVirAddr/iPageSize;

if (iVirAddr%iPageSize == 0) { iVirPageIdx--; iOffset = iPageSize;

} else {

iOffset = iVirAddr%iPageSize;

}

int iPhyPageIdx= PageTable[iVirPageIdx][1]; int ActualIdx = (iPhyPageIdx\*iPageSize)+iOffset-1;

String sVal = PhysicalMemory[ActualIdx];

SB = new StringBuilder();

SB.append(" Logical Address: ").append(iVirAddr).append(",

Physical Page-Number: " ).append(iPhyPageIdx).append(", Offset:

").append(iOffset).append(", Value: ").append(sVal);

System.out.println(SB); } // FOREACH

} // FOR

} // VALID PAGE-SIZE

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

}// MAIN

private static ArrayList<Integer> getIntegerArray(ArrayList<String> arrStrs) {

ArrayList<Integer> arrInts = new ArrayList<Integer>(); for(String sVal : arrStrs) { try {

arrInts.add(Integer.parseInt(sVal));

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

} return arrInts;

}

} //CLASS

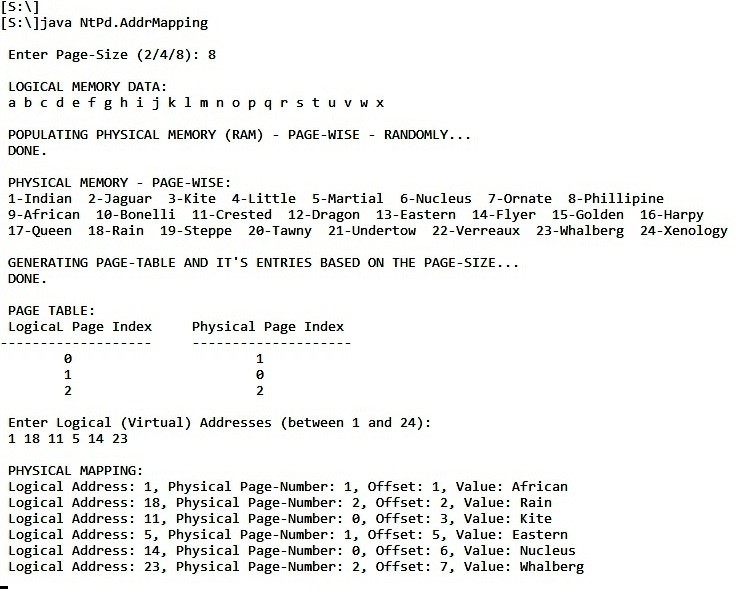
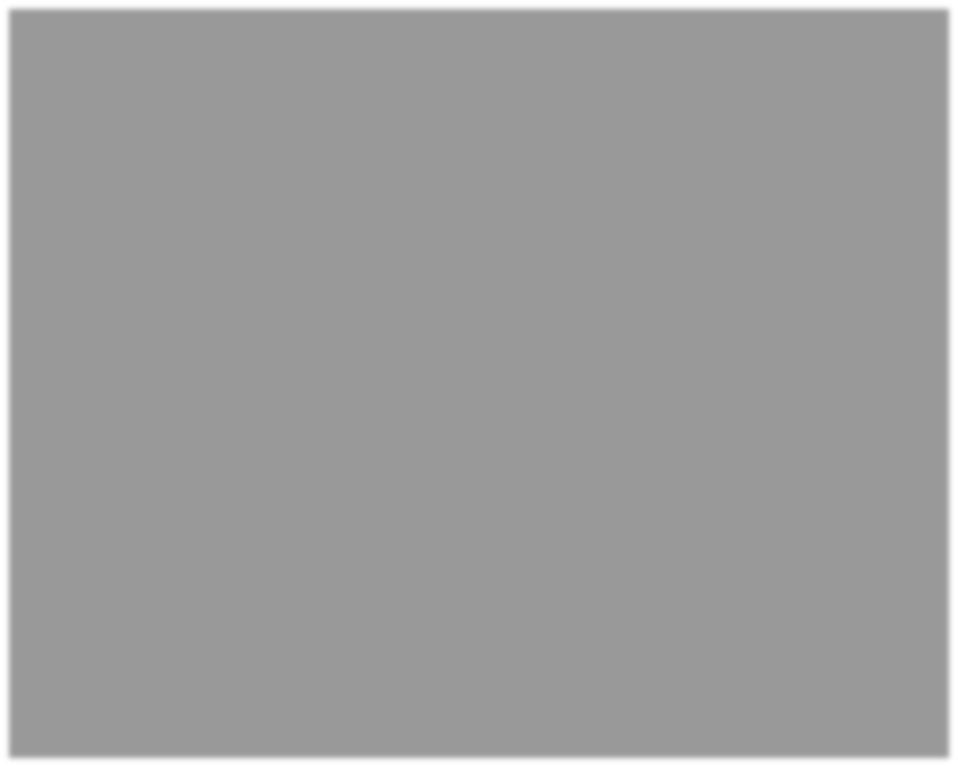
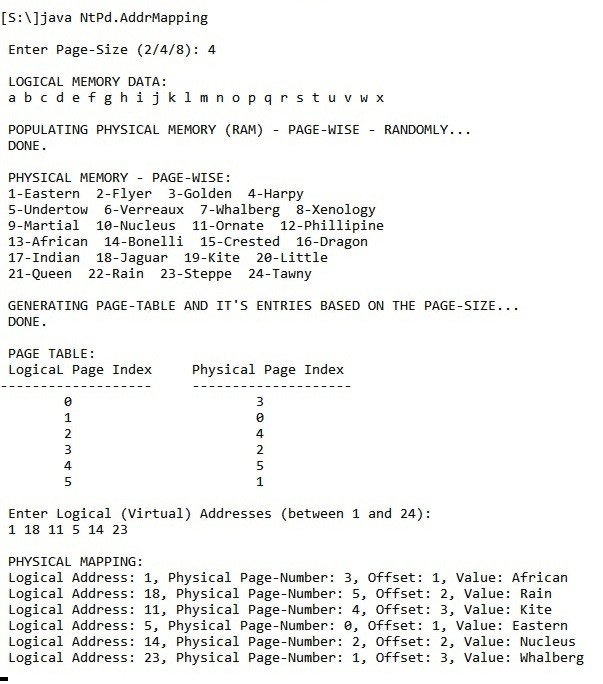
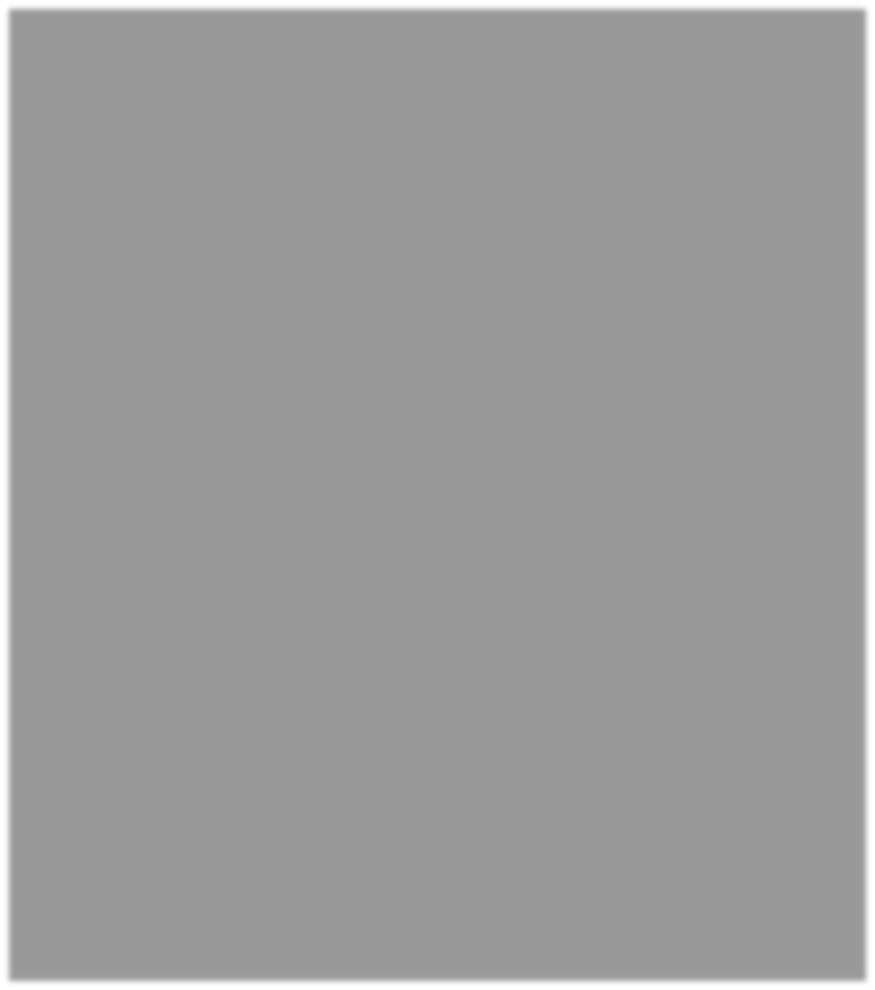


Fig: Calculating Page Numbers and Offsets for various Page-Sizes and various Physical Addresses.

**PRACTICAL 06**

**Aim:** Write a Java Program that simulate the Disk-Scheduling Algorithms: **FCFS, SSTF, SCAN, C-SCAN, LOOK, C-LOOK**. Design separate Classes to implement them. Each Algorithm will implement an Interface ***IDiskScheduler***that has one Method *public int serviceRequests()* that will return the Amount of Head Movement required by each Algorithm.

## Interface: IDiskScheduler

package NtPd;

public interface **IDiskScheduler** {

// service the requests and return the amount of head movement for the particular algorithm  **int serviceRequests();**

}

**Class: FCFS*.java*:** implements **Interface *IDiskScheduler*.**

package NtPd;

import java.lang.Math; import java.util.ArrayList;

public class FCFS implements IDiskScheduler {

private ArrayList<Integer> arrInts; private int len=0; private int iHeadMvmnt = 0;

private StringBuilder sOrder = new StringBuilder();

public FCFS (ArrayList<Integer> Reqs) { arrInts = Remove\_Duplicates(Reqs);

System.out.print("\nApplying FCFS Algorithm: \nPath: ");

}

public int serviceRequests(){

int iHeadMvmnt=0;

sOrder.append(Integer.toString(arrInts.get(0))); for (int f = 0; f <arrInts.size()-1; f++) {

iHeadMvmnt = iHeadMvmnt + Math.abs(arrInts.get(f)-arrInts.get(f+1)); sOrder.append(" >> ").append(Integer.toString(arrInts.get(f+1)));

}

System.out.print(sOrder);

return iHeadMvmnt;

}

public ArrayList<Integer> Remove\_Duplicates (ArrayList<Integer> arrInts) {

ArrayList<Integer> finalList = new ArrayList<Integer>();

for (int i : arrInts) { if (!finalList.contains(i)) { finalList.add(i);

}

}

return finalList;

}

}

**Class: SSTF*.java*:** implements **Interface *IDiskScheduler*.**

package NtPd;

import java.lang.Math; import java.util.ArrayList;

import java.util.Collections;

public class **SSTF** implements IDiskScheduler {

private ArrayList<Integer> Requests; private int iHeadMvmnt = 0; private int CURR\_Request=0; private int CURR\_Idx = 0;

private boolean PreviousIdxSelected = false;

private StringBuilder sOrder = new StringBuilder();

public SSTF (ArrayList<Integer> Reqs) {

CURR\_Request = Reqs.get(0);

sOrder.append(Reqs.get(0));

Requests = Remove\_Duplicates(Reqs);

Collections.sort(Requests);

// Collections.sort(Requests, Collections.reverseOrder()); << FOR REVERSE

ORDER

CURR\_Idx = Requests.lastIndexOf (CURR\_Request);

System.out.print("\nApplying SSTF Algorithm: \nPath: ");

}

public int **serviceRequests**() {

int NXT\_Idx= 0; int NXT\_Request = 0;

int iHeadMvmnt=0;

while (Requests.size() > 1) {

NXT\_Idx = getIdxNearestTo (CURR\_Idx); NXT\_Request = Requests.get(NXT\_Idx);

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - NXT\_Request); sOrder.append(" >> ").append(NXT\_Request);

Requests.remove(CURR\_Idx);

if ((PreviousIdxSelected && CURR\_Idx!=0) || (!PreviousIdxSelected &&

CURR\_Idx == Requests.size())) {

CURR\_Idx --;

}

CURR\_Request = Requests.get(CURR\_Idx);

} // WHILE

System.out.print(sOrder); return iHeadMvmnt;

}

public int **getIdxNearestTo** (int x) {

int LesserDistPos=0; if (x==0) { LesserDistPos = 1; }

else if (x==Requests.size()-1) { LesserDistPos = x-1; }

else {

if (Math.abs(Requests.get(x-1) - Requests.get(x)) < Math.abs(Requests.get(x)

- Requests.get(x+1))) {

LesserDistPos = x-1;

PreviousIdxSelected = true;

} else {

LesserDistPos = x+1;

PreviousIdxSelected = false;

}

}// OUTER ELSE

return LesserDistPos;

}

public ArrayList<Integer> **Remove\_Duplicates** (ArrayList<Integer> arrInts) {

ArrayList<Integer> finalList = new ArrayList<Integer>();

for (int i : arrInts) { if (!finalList.contains(i)) { finalList.add(i);

}

}

return finalList;

}

}

**Class: SCAN*.java*:** implements **Interface *IDiskScheduler*.**

package NtPd;

import java.lang.Math; import java.util.ArrayList; import java.util.Collections; import java.util.Set;

import java.util.LinkedHashSet;

public class **SCAN** implements IDiskScheduler {

private ArrayList<Integer> Requests; private int iHeadMvmnt = 0; private int CURR\_Request = 0; private int CURR\_Idx = 0; private int NXT\_Request = 0;

private boolean LASTCylinderReached = false; private boolean FIRSTCylinderReached = false; private int NearerEnd = 0;

private StringBuilder sOrder = new StringBuilder();

public SCAN (ArrayList<Integer> Reqs) {

CURR\_Request = Reqs.get(0); sOrder .append(Reqs.get(0));

NearerEnd = Math.abs(CURR\_Request - 0) < Math.abs (199 - CURR\_Request) ? 0: 199;

System.out.println("\nNearest End: " + NearerEnd); Requests = Remove\_Duplicates(Reqs);

Collections.sort(Requests);

CURR\_Idx = Requests.lastIndexOf (CURR\_Request);

System.out.print("\nApplying SCAN Algorithm: \nPath: ");

}

public int **serviceRequests**() {

while (Requests.size() > 1) {

if (NearerEnd == 0) {

// MOVING BACKWARD TO 0 while (CURR\_Idx > 0) { Calculate\_NXT (-1); CURR\_Idx --;

}

// THE FIRST CYLINDER: '0' SECTION:

if (CURR\_Request !=0 && !FIRSTCylinderReached) {

// MOVING FROM FIRST REQUEST TO 0...

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - 0); sOrder.append(" >> 0");

FIRSTCylinderReached = true;

// HERE: LAST REQUEST IS REMOVED AFTER BEING SERVICED.

Requests.remove(CURR\_Idx);

// COMING BACK FROM 0 TO THE NEW FIRST REQUEST... if (Requests.size() > 0) {

// NOW...NEW CURR\_Idx is the LAST INDEX. CURR\_Request = Requests.get(CURR\_Idx); iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - 0);

sOrder .append(" >> ").append(CURR\_Request);

}

}

// CHANGING DIRECTION: MOVING FROM FIRST CYLINDER (0) PROCESSING ALL REMAINING REQUESTS.

// IF THERE ARE STILL REQUESTS TO BE SERVED AFTER REACHING FIRST CYLINDER /\* GREATER THAN 1, THAT IS MINIMUM 2, BECAUSE WE NEED TO CALCULATE ...

...CURRENT INDEX AND ALSO NEXT INDEX. \*/ if (Requests.size() > 1) {

while (CURR\_Idx < Requests.size()-1) {

Calculate\_NXT (1);

/\* CURR\_Idx IS AUTOMATICALLY SET TO NEXT ONE WHEN REMOVED WHILE MOVING

FORWARD \*/

}

} // IF STILL REQUESTS LEFT

} else {

// MOVING FORWARD TO 199...

while (CURR\_Idx < Requests.size()-1) {

Calculate\_NXT (1);

/\* CURR\_Idx IS AUTOMATICALLY SET TO NEXT ONE WHEN REMOVED WHILE MOVING FORWARD \*/

}

// THE LAST CYLINDER: '199' SECTION:

if (CURR\_Request !=199 && !LASTCylinderReached) { // MOVING FROM LAST REQUEST TO 199...

iHeadMvmnt = iHeadMvmnt + Math.abs(199 - CURR\_Request); sOrder .append(" >> 199");

LASTCylinderReached = true;

// HERE: LAST REQUEST IS REMOVED AFTER BEING SERVICED.

Requests.remove(CURR\_Idx);

// COMING BACK FROM 199 TO THE NEW LAST REQUEST...

if (Requests.size() > 0) {

CURR\_Idx--;

// NOW...NEW CURR\_Idx is the LAST INDEX. CURR\_Request = Requests.get(CURR\_Idx); iHeadMvmnt = iHeadMvmnt + Math.abs(199 - CURR\_Request); sOrder .append(" >> ").append(CURR\_Request);

}

}

// IF THERE ARE STILL REQUESTS TO BE SERVED...MOVING BACKWARD TO 0...

if (Requests.size() > 1) {

while (CURR\_Idx > 0) { Calculate\_NXT (-1); CURR\_Idx --;

}

} // IF STILL REQUESTS LEFT

} // IF 0 OR 199

} // WHILE REQUESTS TO BE PROCESSED, EXIST

System.out.println(sOrder);

return iHeadMvmnt;

}

public void **Calculate\_NXT** (int MoveBy) {

NXT\_Request = Requests.get(CURR\_Idx + MoveBy);

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - NXT\_Request); sOrder .append(" >> ").append(NXT\_Request);

Requests.remove(CURR\_Idx);

CURR\_Request = NXT\_Request;

}

public ArrayList<Integer> **Remove\_Duplicates** (ArrayList<Integer> arrInts) {

// Create a new LinkedHashSet

Set<Integer> LHS = new LinkedHashSet<>();

// Add the elements to set LHS.addAll(arrInts);

// Clear the list arrInts.clear();

// add the elements of set with no duplicates to the list arrInts.addAll(LHS);

// return the list

return arrInts;

}

}

**Class: C\_SCAN*.java*:** implements **Interface *IDiskScheduler*.**

package NtPd;

import java.lang.Math; import java.util.ArrayList;

import java.util.Collections;

public class **C\_SCAN** implements IDiskScheduler {

private ArrayList<Integer> Requests; private int iHeadMvmnt = 0; private int CURR\_Request = 0; private int CURR\_Idx = 0; private int NXT\_Request = 0;

private boolean LASTCylinderReached = false; private boolean FIRSTCylinderReached = false; private int NearerEnd = 0;

private StringBuilder sOrder = new StringBuilder();

public C\_SCAN (ArrayList<Integer> Reqs) {

CURR\_Request = Reqs.get(0); sOrder.append(Reqs.get(0));

NearerEnd = Math.abs(CURR\_Request - 0) < Math.abs (199 - CURR\_Request) ? 0 : 199;

System.out.println("\nNearest End: " + NearerEnd);

Collections.sort(Reqs);

Requests = Remove\_Duplicates(Reqs);

CURR\_Idx = Requests.lastIndexOf (CURR\_Request);

System.out.print("\nApplying C-SCAN Algorithm: \nPath: ");

}

public int serviceRequests() {

while (Requests.size() > 1) {

if (NearerEnd == 0) {

while (CURR\_Idx > 0) {

Calculate\_NXT (-1); CURR\_Idx --;

}

if (CURR\_Request !=0 && !FIRSTCylinderReached) {

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - 0); sOrder.append(" >> 0");

FIRSTCylinderReached = true;

Requests.remove(CURR\_Idx);

if (Requests.size() > 0) {

sOrder.append(" >> 199"); LASTCylinderReached = true;

CURR\_Idx = Requests.size()-1; CURR\_Request = Requests.get(CURR\_Idx); if (CURR\_Request !=199) {

iHeadMvmnt = iHeadMvmnt + (199 - CURR\_Request);

sOrder.append(" >> ").append(CURR\_Request);

}

}

}

} else {

while (CURR\_Idx < Requests.size()-1) {

Calculate\_NXT (1);

}

if (CURR\_Request !=199 && !LASTCylinderReached) {

iHeadMvmnt = iHeadMvmnt + Math.abs(199 - CURR\_Request); sOrder.append(" >> 199");

LASTCylinderReached = true;

Requests.remove(CURR\_Idx);

if (Requests.size() > 0) {

sOrder.append(" >> 0"); FIRSTCylinderReached = true;

CURR\_Idx = 0;

CURR\_Request = Requests.get(CURR\_Idx);

if (CURR\_Request !=0 ) {

iHeadMvmnt = iHeadMvmnt + CURR\_Request;

sOrder.append(" >> " ).append(CURR\_Request);

}

}

}

}

}

System.out.println(sOrder); return iHeadMvmnt;

} public void Calculate\_NXT (int MoveBy) { NXT\_Request = Requests.get(CURR\_Idx + MoveBy);

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - NXT\_Request); sOrder.append(" >> ").append(NXT\_Request); Requests.remove(CURR\_Idx);

CURR\_Request = NXT\_Request;

}

public ArrayList<Integer> Remove\_Duplicates (ArrayList<Integer> arrInts) {

ArrayList<Integer> finalList = new ArrayList<Integer>();

for (int i : arrInts) { if (!finalList.contains(i)) {

finalList.add(i);

}

}

return finalList;

}

}

**Class: LOOK*.java*:** implements **Interface *IDiskScheduler*.**

package NtPd;

import java.lang.Math; import java.util.ArrayList;

import java.util.Collections;

public class LOOK implements IDiskScheduler {

private ArrayList<Integer> Requests; private int iHeadMvmnt = 0; private int CURR\_Request = 0; private String sOrder = "";

private int CURR\_Idx = 0;

private StringBuilder sOrder = new StringBuilder();

public LOOK (ArrayList<Integer> Reqs) {

CURR\_Request = Reqs.get(0);

sOrder = sOrder + Reqs.get(0);

Collections.sort(Reqs);

Requests = Remove\_Duplicates(Reqs);

CURR\_Idx = Requests.lastIndexOf (CURR\_Request);

System.out.print("\nApplying LOOK Algorithm: \nPath: ");

}

public int serviceRequests() {

int NXT\_Request = 0;

int iHeadMvmnt=0;

while (Requests.size() > 1) {

while (CURR\_Idx > 0) {

NXT\_Request = Requests.get(CURR\_Idx - 1);

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - NXT\_Request);

sOrder.append(" >> ").append(NXT\_Request);

Requests.remove(CURR\_Idx);

CURR\_Request = NXT\_Request; CURR\_Idx --;

}

if (Requests.size() > 1) {

iHeadMvmnt = iHeadMvmnt + Math.abs(Requests.get(0) - Requests.get(1));

sOrder.append(" >> ").append(Requests.get(1));

Requests.remove(0);

}

}

System.out.println();

System.out.print(sOrder);

return iHeadMvmnt;

}

public ArrayList<Integer> Remove\_Duplicates (ArrayList<Integer> arrInts) { ArrayList<Integer> finalList = new ArrayList<Integer>();

for (int i : arrInts) { if (!finalList.contains(i)) {

finalList.add(i);

}

}

return finalList;

}

}

**Class: C\_LOOK*.java*:** implements **Interface *IDiskScheduler*.**

package NtPd;

import java.lang.Math; import java.util.ArrayList;

import java.util.Collections;

public class C\_LOOK implements IDiskScheduler {

private ArrayList<Integer> Requests; private int iHeadMvmnt = 0;

private int CURR\_Request = 0;

private int CURR\_Idx = 0;

private StringBuilder sOrder = new StringBuilder();

public C\_LOOK (ArrayList<Integer> Reqs) {

CURR\_Request = Reqs.get(0); sOrder.append(Reqs.get(0)); Requests = Remove\_Duplicates(Reqs);

Collections.sort(Requests);

CURR\_Idx = Requests.lastIndexOf (CURR\_Request);

System.out.print("\nApplying C-LOOK Algorithm: \nPath: ");

}

public int serviceRequests() {

int NXT\_Request = 0; int iHeaMvmnt=0;

while (Requests.size() > 1) {

while (CURR\_Idx < Requests.size()-1) { NXT\_Request = Requests.get(CURR\_Idx + 1);

iHeadMvmnt = iHeadMvmnt + Math.abs(CURR\_Request - NXT\_Request);

sOrder.append(" >> ").append(NXT\_Request);

Requests.remove(CURR\_Idx);

CURR\_Request = NXT\_Request;

}

if (Requests.size() > 1) {

iHeadMvmnt = iHeadMvmnt + (Requests.get(CURR\_Idx) - Requests.get(0));

sOrder.append(" >> ").append(Requests.get(0)); Requests.remove(CURR\_Idx);

CURR\_Idx = 0;

CURR\_Request = Requests.get(CURR\_Idx);

}

}

System.out.println(sOrder);

return iHeadMvmnt;

}

public ArrayList<Integer> Remove\_Duplicates (ArrayList<Integer> arrInts) {

ArrayList<Integer> finalList = new ArrayList<Integer>(); for (int i : arrInts) { if (!finalList.contains(i)) {

finalList.add(i);

}

}

return finalList;

}

}

**Class: DskSchdl*.java***

package NtPd;

import java.io.\*; import java.util.\*;

public class **DskSchdl** {

private static BufferedReader BR = new BufferedReader (new InputStreamReader (System.in));

public DskSchdl() {}

public static void main(String[] args) {

String sLine="";

String Words[]=null;

ArrayList<Integer> iDARs = new ArrayList<>();

System.out.println("Enter all Disk Access Requests: ");

try {

for (sLine = BR.readLine(); sLine != null; sLine = BR.readLine()) {

Words=sLine.split(" ");

iDARs = getIntegerArray(new ArrayList<>(Arrays.asList(Words)));

FCFS F = new FCFS(iDARs);

System.out.print("\nTotal Seek Count: " + F.serviceRequests());

SSTF S = new SSTF(iDARs);

System.out.print("\nTotal Seek Count: " + S.serviceRequests());

SCAN SC = new SCAN(iDARs);

System.out.print("\nTotal Seek Count: " + SC.serviceRequests());

C\_SCAN CS = new C\_SCAN(iDARs);

System.out.print("\nTotal Seek Count: " + CS.serviceRequests());

LOOK L = new LOOK(iDARs);

System.out.print("\nTotal Seek Count: " + L.serviceRequests());

C\_LOOK CL = new C\_LOOK(iDARs);

System.out.print("\nTotal Seek Count: " + CL.serviceRequests());

} // FOR

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

} //MAIN

private static ArrayList<Integer> getIntegerArray(ArrayList<String> arrStrs) {

ArrayList<Integer> arrInts = new ArrayList<Integer>();

for(String sVal : arrStrs) { try {

arrInts.add(Integer.parseInt(sVal));

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

} return arrInts;

}

}

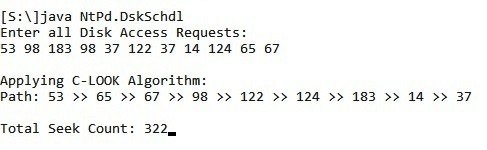
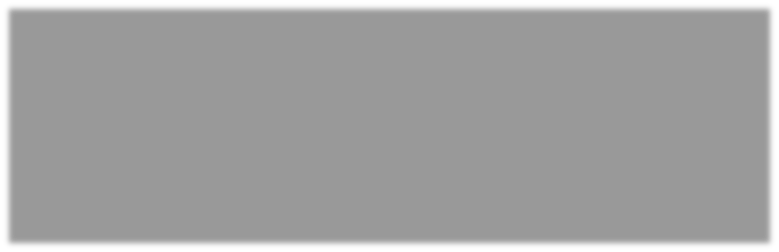
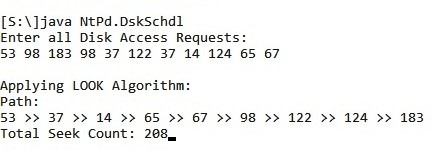
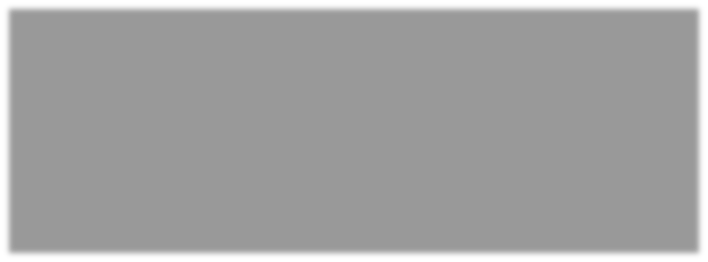
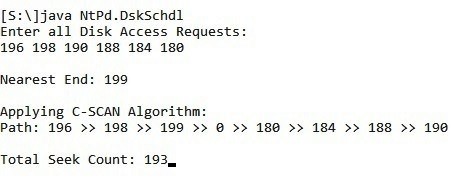
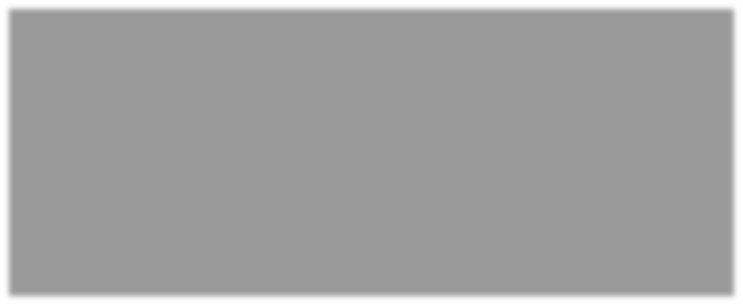
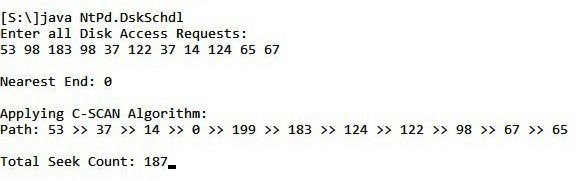
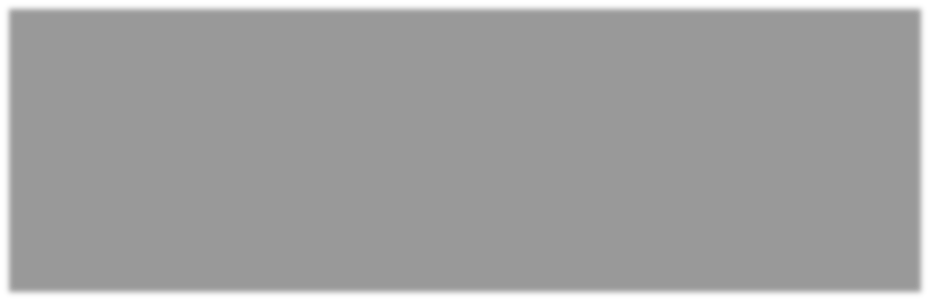
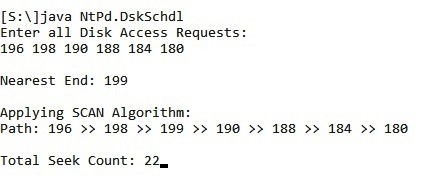
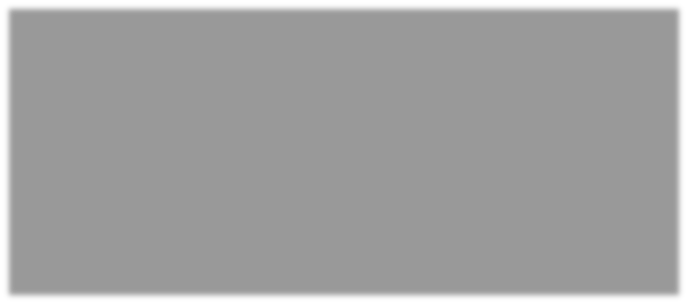
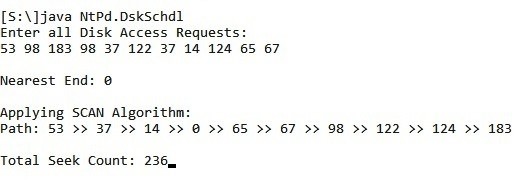
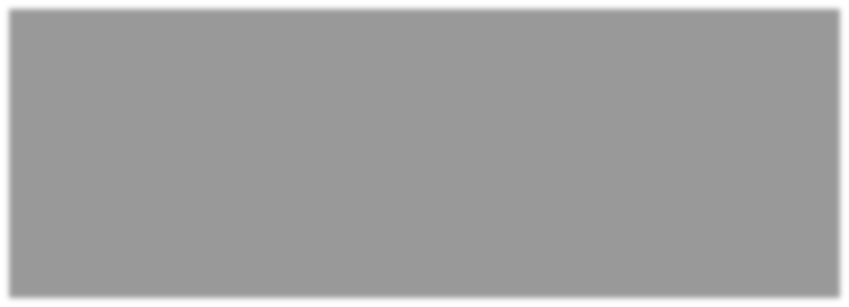
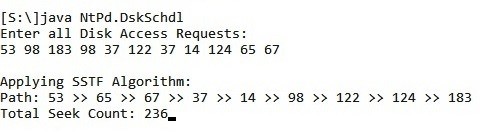
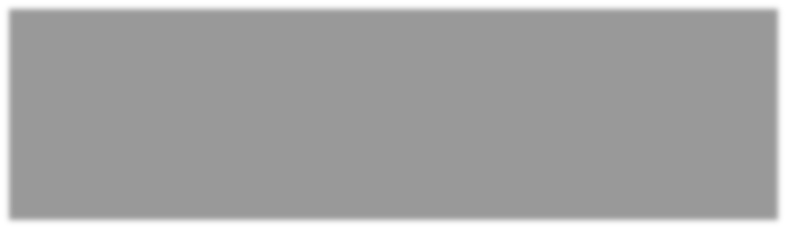
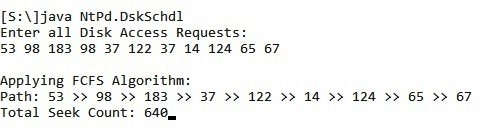
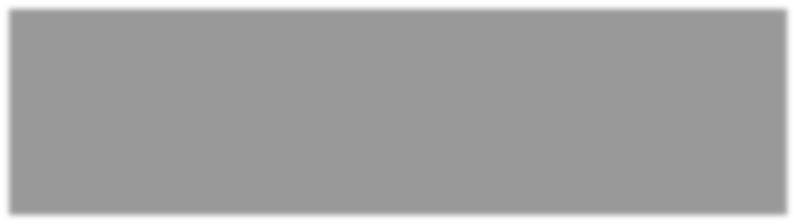


Fig. 6.01: **Various** **Head Movements** for various Algorithms for the **same** **Disk Access Requests.**

**PRACTICAL 07:**

**Aim**: Write a Java Program that implements the **FIFO** and **LRU Page-**

**Replacement Algorithms**. Apply a random Page-Reference String to each Algorithm and record the Number of Page-Faults in each Algorithm. Vary the Page-Frames and record the Results. Classes that implement the above Page-Replacement algorithms must implement the Interface *ReplacementAlgorithm* that has an insert() Method to implement.

## Interface: ReplacementAlgorithm

package NtPd;

public interface ***ReplacementAlgorithm* {**

public void insert();

**}**

## Class: FIFOAlgo

package NtPd;

import java.util.\*; public class FIFOAlgo {

ArrayList<Integer> iPages;

ArrayList<Integer> iFrames;

int TtlFrames,TtlPages, PageFault, PageHit; boolean IsAPgFault;

String sFaultOrHit = "";

int IncomingPage=0; int PageInMemory=0;

int CurrPos = 0;

private StringBuilder SB = new StringBuilder();

public FIFOAlgo (ArrayList<Integer> Pgs, ArrayList<Integer> Frms) {

iPages = Pgs;

TtlPages = iPages.size(); iFrames=Frms;

TtlFrames = PageFault=iFrames.size();

PageHit=0;

System.out.println("\nApplying FIFO Algorithm...");

}

public void insert() {

// THE FRAME WHEN RECEIVED HAS IT'S ALL SPACES FILLED UP ALREADY. for(int p=TtlFrames; p<TtlPages; p++) {

IsAPgFault=true;

IncomingPage=iPages.get(p);

PageInMemory=0; sFaultOrHit= "";

for (int j=0;j<TtlFrames;j++) {

PageInMemory = iFrames.get(j); if(PageInMemory==IncomingPage) {

PageHit ++;

sFaultOrHit= " Page Hit. Going back to Frame-1 again...";

IsAPgFault=false;

break; // Stop checking as soon as any one Page-In-Memory

(in Frame) matches the Incoming Page.

}

}

if(CurrPos==TtlFrames) {

System.out.println("\nComparison till the Last Frame completed. Going back to Frame-1 again...");

CurrPos=0;

}

if (!IsAPgFault) {

SB = new StringBuilder();

SB.append("\n ").append(IncomingPage).append(" matches ")

.append(PageInMemory);

System.out.print(SB);

}

if(IsAPgFault) {

SB = new StringBuilder();

SB.append("\n ").append(IncomingPage).append(" replace")

.append(iFrames.get(CurrPos));

System.out.print(SB);

iFrames.set(CurrPos,IncomingPage); //Incoming Page replaces Page-In-Memory

PageFault++;

sFaultOrHit= " Page Fault.";

CurrPos++;

} // IS A FAULT

System.out.print(". Frames: "); for (int k=0;k<TtlFrames;k++) {

SB = new StringBuilder();

SB.append(iFrames.get(k)).append(" ");

System.out.print(SB);

}

System.out.print(sFaultOrHit);

} // ALL PAGES - FOR

SB = new StringBuilder();

SB.append("\n\nTotal Page Faults: ").append(PageFault).append(", Total Page Hits: ")

.append(PageHit);

System.out.print(SB);

} // INSERT FUNC

}

## Class: LRUAlgo

package NtPd;

import java.util.\*;

public class LRUAlgo {

ArrayList<Integer> iPages;

ArrayList<Integer> iFrames;

int TtlFrames,TtlPages, PageFault, PageHit; boolean IsAPgFault;

String sFaultOrHit = "";

int IncomingPage=0;

int PageInMemory=0;

private StringBuilder SB = new StringBuilder();

public LRUAlgo (ArrayList<Integer> Pgs, ArrayList<Integer> Frms) {

iPages = Pgs;

TtlPages = iPages.size(); iFrames=Frms;

TtlFrames = PageFault=iFrames.size();

PageHit=0;

System.out.print("\n Applying LRU Algorithm...");

}

public void insert() {

// THE MAIN MEMORY BLOCK 'Frms' WHEN RECEIVED HAS ALL ITS SPACES FILLED UP ALREADY.

for(int p=TtlFrames; p<TtlPages; p++) {

IsAPgFault=true;

IncomingPage=iPages.get(p); PageInMemory=0;

sFaultOrHit= "";

for (int j=0;j<TtlFrames;j++) {

PageInMemory = iFrames.get(j); if(PageInMemory==IncomingPage) {

PageHit ++;

sFaultOrHit= "Page Hit."; IsAPgFault=false;

break; // Stop checking as soon as any one Page-In-Memory

(in Frame) matches the Desired Page.

}

}

if (!IsAPgFault) { SB = new StringBuilder();

SB.append("\n\n ").append(IncomingPage).append(" matches

").append(PageInMemory);

System.out.print(SB);

}

if(IsAPgFault) {

/\* LOOKING FOR THE PREVIOUS LEAST USED PAGE TO REPLACE. \*/

SB = new StringBuilder();

SB.append("\n\n Incoming Page: ").append(iPages.get(p))

.append(".");

System.out.print(SB); int LeastUsedPg\_Pos = p-1;

int LeastUsedFrm\_Pos = 0;

for (int f=0; f<TtlFrames; f++) {

for (int g=p-1; g>=0; g--) {

if (iPages.get(g) == iFrames.get(f)) {

if (g < LeastUsedPg\_Pos) {

LeastUsedPg\_Pos = g;

LeastUsedFrm\_Pos=f;

}

break;

}

} } // FOR

SB = new StringBuilder();

SB.append(" Least Recently Used Page: ").append(iPages.get(LeastUsedPg\_Pos));

System.out.print(SB);

SB.append("\n").append(IncomingPage).append(" replaces ")

.append(iFrames.get(LeastUsedFrm\_Pos));

iFrames.set(LeastUsedFrm\_Pos,IncomingPage); // Incoming Page replaces Page-In-Memory

PageFault++;

sFaultOrHit= "Page Fault.";

} // IS A FAULT

System.out.print("\n Frames: ");

for (int k=0;k<TtlFrames;k++) { System.out.print(iFrames.get(k));

System.out.print(" ");

}

System.out.print(sFaultOrHit);

} // ALL PAGES - FOR

System.out.print("\n\n Total Page Faults: " + PageFault + ", Total Page Hits: " + PageHit);

} // INSERT FUNC

}

**Class: *PgRplc***

package NtPd;

import java.io.\*; import java.util.\*;

public class PgRplc {

private static BufferedReader BR = new BufferedReader (new InputStreamReader

(System.in));

private static String sLine=""; private static String Words[]=null;

private static int TtlFrames, TtlPages;

private static ArrayList<Integer> iPages = new ArrayList<>(); private static ArrayList<Integer> iFrames; private static int IncomingPage=0;

StringBuilder SB = new StringBuilder();

public static void main(String[] args) {

try {

System.out.print(" Enter Main-Memory Size: ");

TtlFrames= Integer.parseInt(BR.readLine());

System.out.println(" Enter all Page-References: "); System.out.print(" ");

for (sLine = BR.readLine(); sLine != null; sLine = BR.readLine()) {

Words=sLine.split(" ");

iPages = getIntegerArray(new ArrayList<>(Arrays.asList(Words)));

TtlPages = iPages.size();

iFrames = new ArrayList<Integer> (TtlFrames); for(int f=0; f<TtlFrames; f++) {

iFrames.add(f,-1);

}

System.out.println("\n All Frames Initialized to -1 (Empty). ");

System.out.println( " NOTE: If the Incoming Page and the Page-InMemory MATCH: Page-Hit. If not then: Page-Fault.");

for(int p=0; p<TtlFrames; p++) {

IncomingPage=iPages.get(p);

SB.append("\n Incoming Page: ").append(IncomingPage)

.append(". ");

System.out.println(SB);

// ROUND-1: ALL 'N' EMPTY FRAMES ARE FILLED FIRST... // Incoming Page fills up Empty Space. iFrames.set(p,IncomingPage);

System.out.print(" Frames: ");

int k=0;

for (int Frm : iFrames) { SB = new StringBuilder();

SB.append(iFrames.get(k)).append(" "); System.out.print(SB); k++;

} // FOR

System.out.print("Page Fault.");

}

System.out.println("\n\n NOTE: ALL ALGORITHMS ONLY

START WORKING FROM HERE ONWARDS, THAT IS,...");

System.out.println(" ...AFTER ALL EMPTY FRAMES HAVE BEEN FILLED UP.");

FIFOAlgo F = new FIFOAlgo(iPages, iFrames);

F.insert();

LRUAlgo L = new LRUAlgo(iPages, iFrames);

L.insert();

} // FOR

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

} //MAIN

private static ArrayList<Integer> getIntegerArray(ArrayList<String> arrStrs) { ArrayList<Integer> arrInts = new ArrayList<Integer>();

for(String sVal : arrStrs) { try {

arrInts.add(Integer.parseInt(sVal));

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

}

return arrInts;

}

}

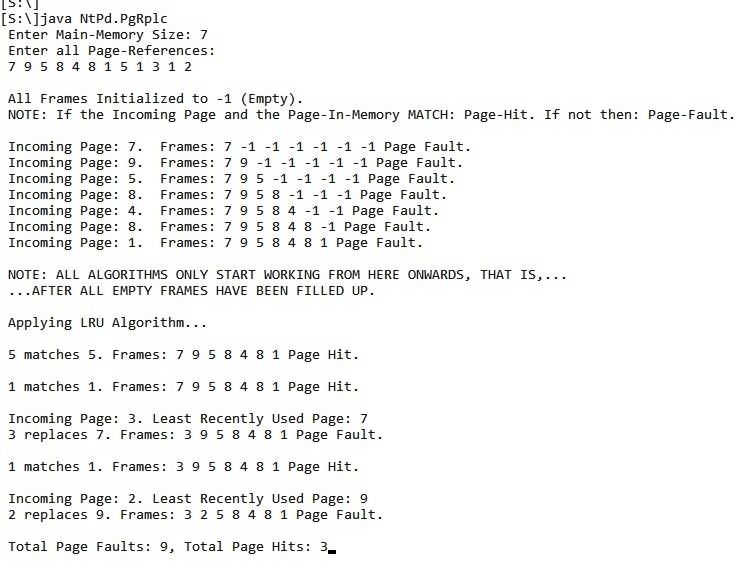
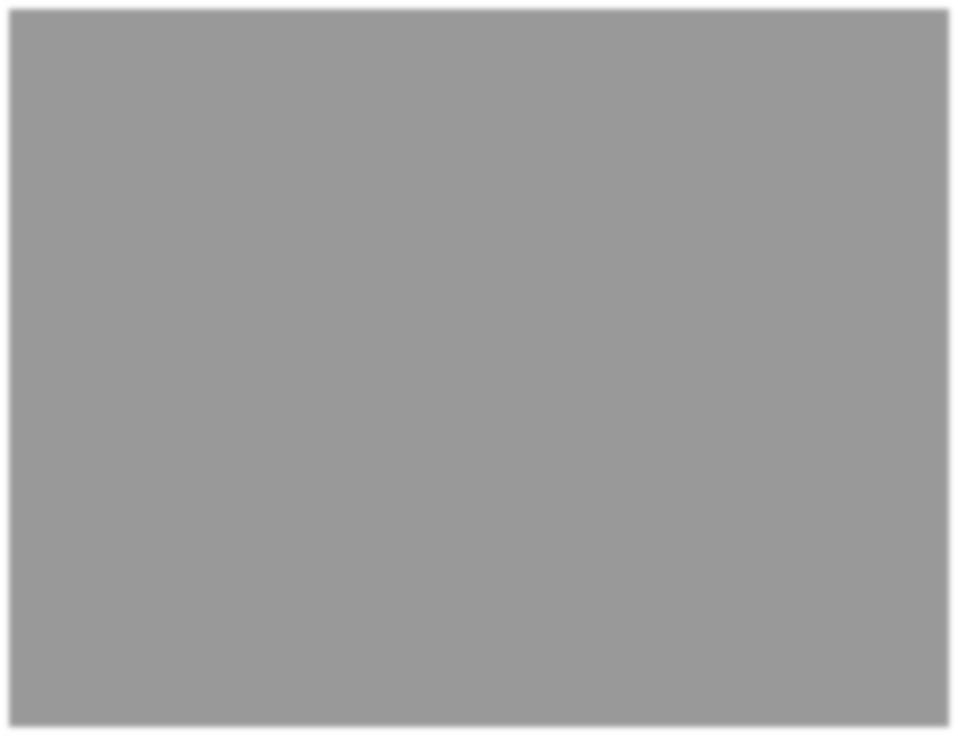
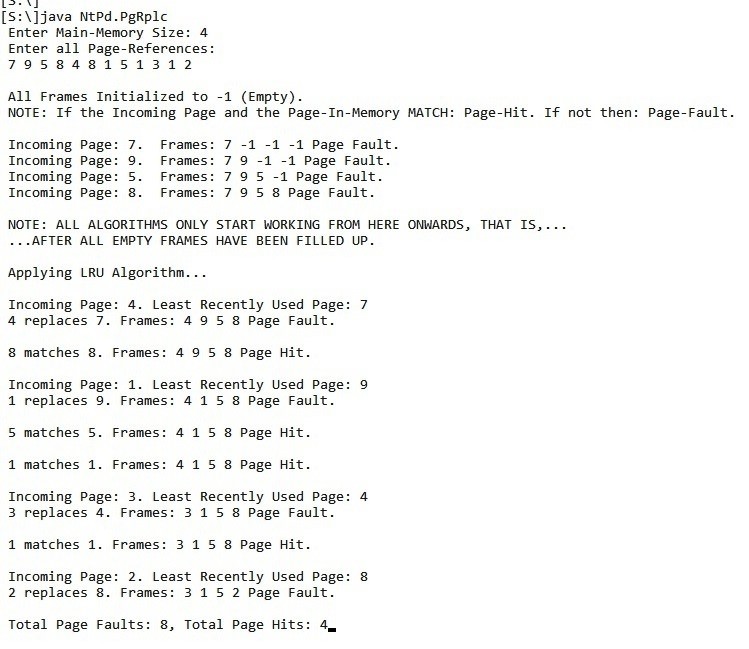
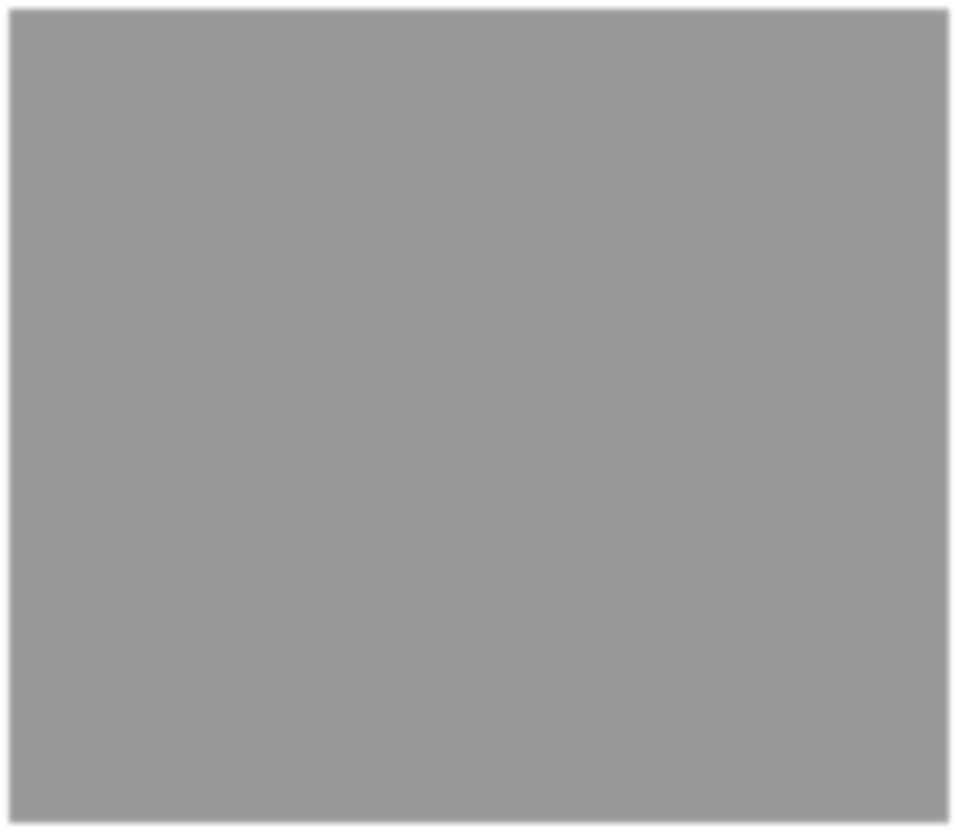
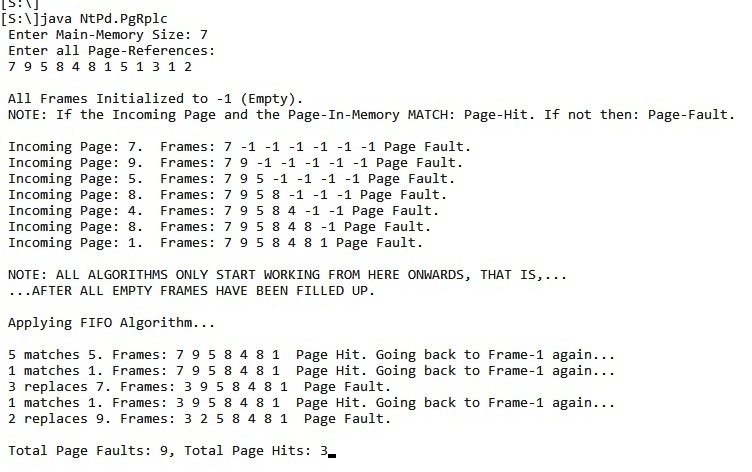
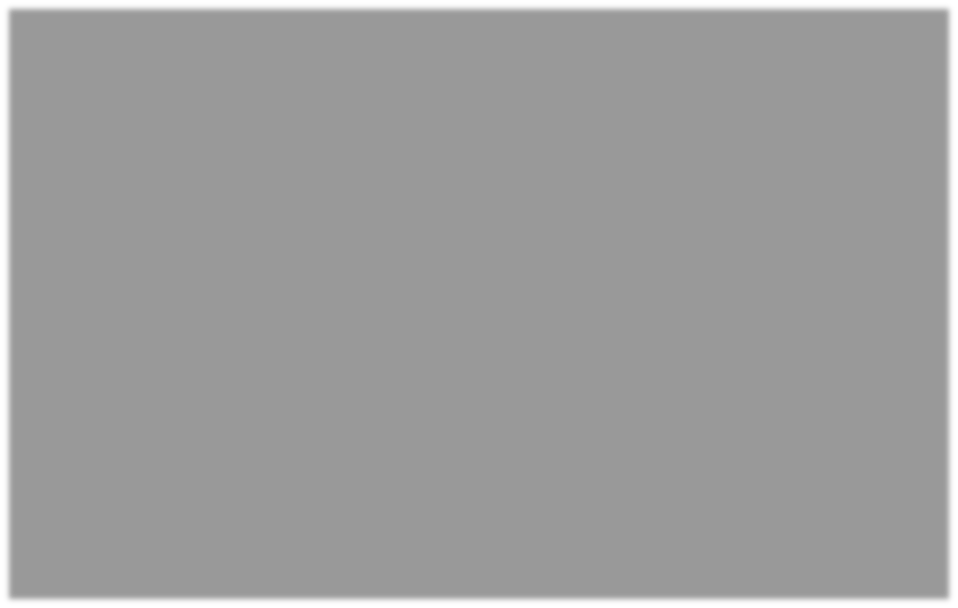
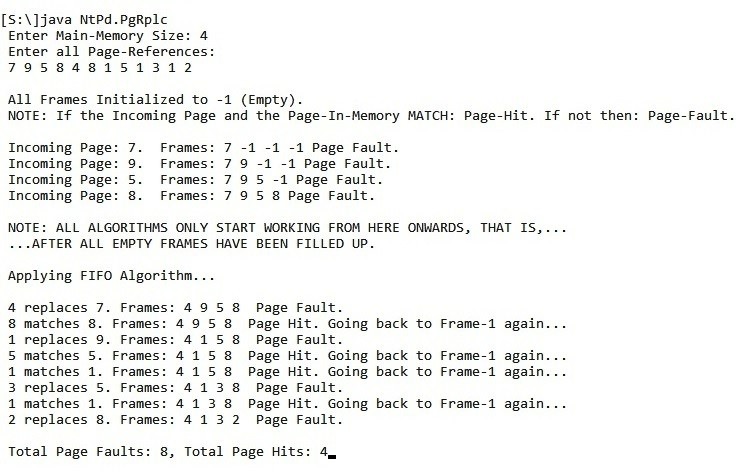
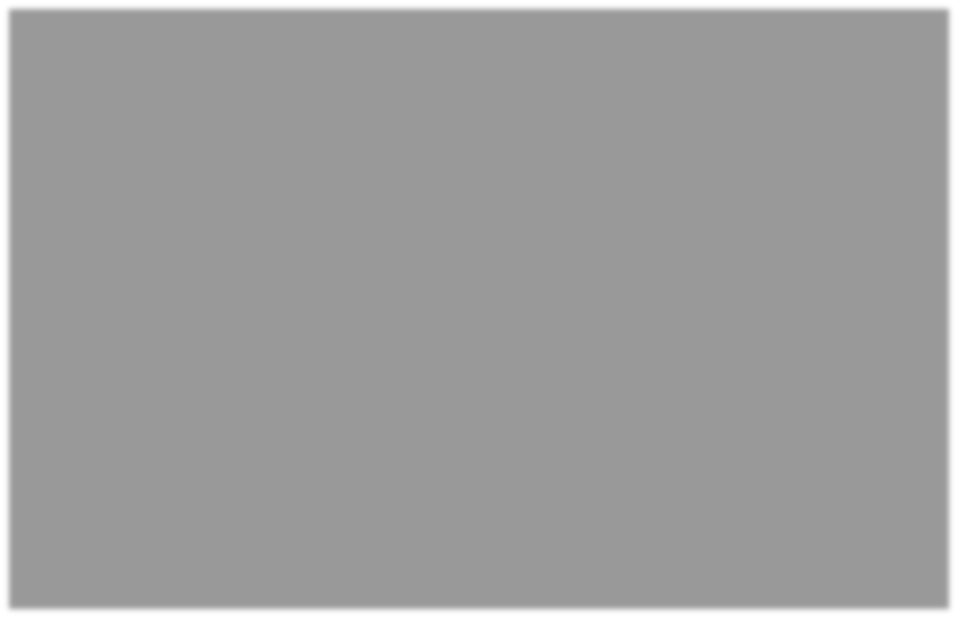


Fig: FIFO & LRU Page-Replacements for same Page-References using Main-Memory-Sizes of 4 & 7.