Institute of Distance and Open Learning

Vidya Nagari, Kalina, Santacruz East - 400098

A Practical Journal Submitted in fulfillment

of the degree of

MASTER OF SCIENCE

IN

COMPUTER SCIENCE

YEAR 2024-2025

Part 1 - Semester 2

PSCS201

"Advanced Operating System"

By

MR. Yogesh Vishwas Dhamke

Application ID:- 41752

Seat No:- 4500425

Under the Guidance of

Prof. Kavita M. Chouk

Institute of Distance and Open Learning

Vidya Nagari, kalina, Santacruz East – 400098

CERTIFICATE

This is to certify that, this practical journal entitled **"Advanced Operating System"** is a record of work carried out by **Mr. Yogesh Vishwas Dhamke (Seat No:- 4500425)**, student of **Master of Science in Computer Science Part 1** class and is submitted to University of Mumbai, in partial fulfillment of the requirement for the award of the degree of **Master of Science in Computer Science.** The practical journal has been approved.

----------	----------	----------
Guide	External Examiner	Coordinator- M.Sc.CS

Index

Sr.No	Practical	Signature
1	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.	
3	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.	
5	Assuming that a system has a 32-bit virtual address, write a Java program that is passed (1) the size of a page and (2) the virtual address. Your program will report the page number and offset of the given virtual address with the specified page size. Page sizes must be specified as a power of 2 and within the range 1024 — 16384 (inclusive). Assuming such a program is named Address, it would run as follows: java Address 4096 19986 and the correct output would appear as: The address 19986 contains: page number = 4 offset = 3602.	
6	Write a Java program that simulates the following disk-scheduling algorithms. Design separate classes that implement the following scheduling algorithms: a. FCFS b. SSTF c. SCAN	
7	Write a program that implements the FIFO and LRU page-replacement algorithms presented in this chapter. First, generate a random page reference string where page numbers range from 0 to 9. Apply the random page-reference string to each algorithm, and record the number of page faults incurred by each algorithm. Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used. Design and implement two classes—LRU and FIFO—that extend ReplacementAlgorithm. Each of these classes will implement the insert() method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm. Test your algorithm with suitable Java programs.	
9	Write Android activity that includes each of the fundamental lifecycle methods.	

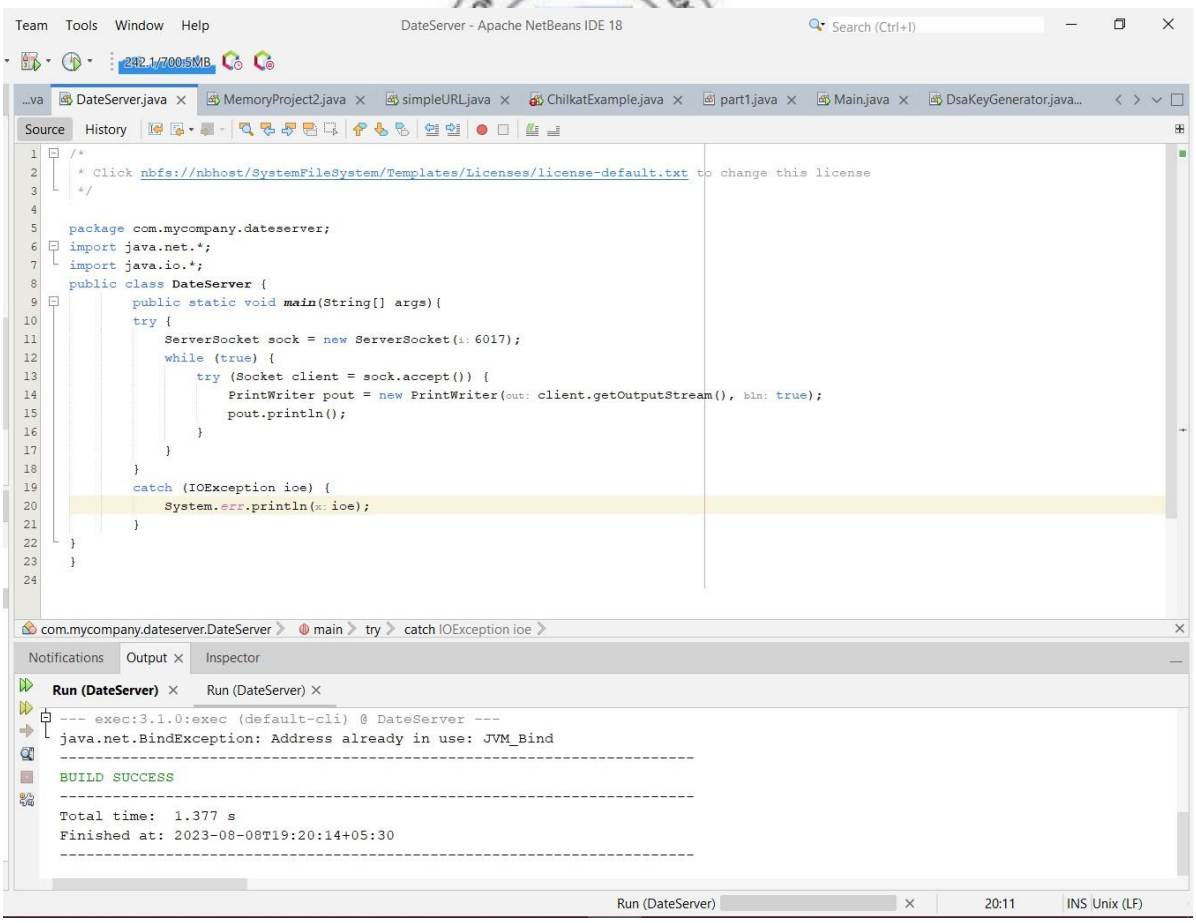
Practical 1

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.

Source Code:-

```
import java.net.*;
import java.io.*;
public class DateServer {
    public static void main(String[] args){
    try {
       ServerSocket sock = new ServerSocket(6017);
       while (true) {
          try (Socket client = sock.accept()) {
             PrintWriter pout = new PrintWriter(client.getOutputStream(), true);
             pout.println();
          }
       }
    }
    catch (IOException ioe) {
       System.err.println(ioe);
    }
}
}
```

Output:


Team Tools Window Help
DateServer - Apache NetBeans IDE 18
Search (Ctrl+I)
242.1/700.5MB
DateServer.java
MemoryProject2.java
simpleURL.java
ChilkatExample.java
part1.java
Main.java
DsaKeyGenerator.java...
Source History
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
*/
package com.mycompany.dateserver;
import java.net.*;
import java.io.*;
public class DateServer {
public static void main(String[] args){
try {
ServerSocket sock = new ServerSocket(i: 6017);
while (true) {
try (Socket client = sock.accept()) {
PrintWriter pout = new PrintWriter(out: client.getOutputStream(), bln: true);
pout.println();
}
}
}
catch (IOException ioe) {
System.err.println(x: ioe);
}
}
}
com.mycompany.dateserver.DateServer main try catch IOException ioe
Notifications
Output
Inspector
Run (DateServer)
Run (DateServer)
--- exec:3.1.0:exec (default-cli) @ DateServer ---
java.net.BindException: Address already in use: JVM_Bind
BUILD SUCCESS
Total time: 1.377 s
Finished at: 2023-08-08T19:20:14+05:30
Run (DateServer)
20:11
INS Unix (LF)

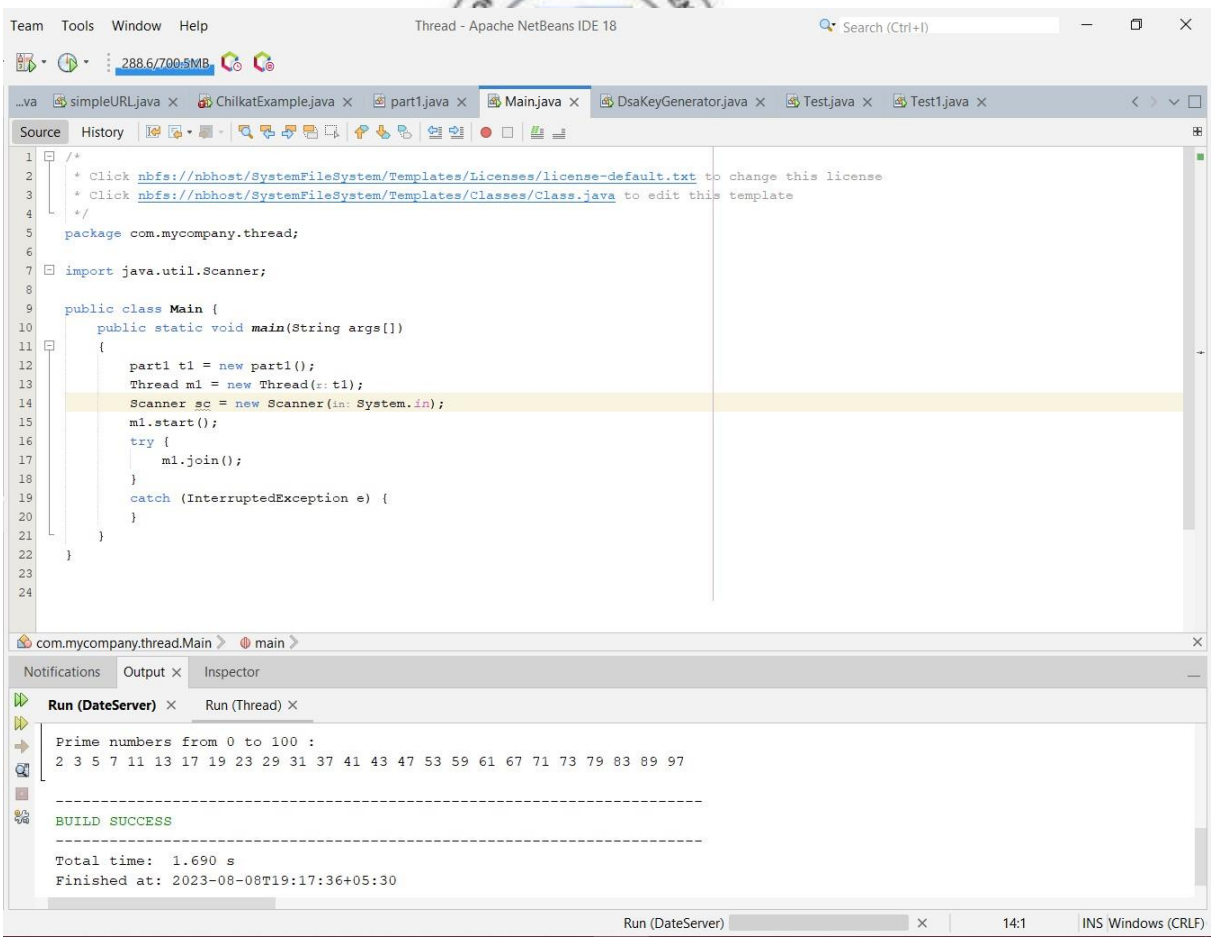
Practical 3

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.

Source Code:-

```
public class MemoryProject2 {
   public static void main(String[] args){
   }
   private final int virtualaddress ;
   private int pageNumber ;
   private int offSet;
   public MemoryProject2 (int va) {
      virtualaddress = va ;
   }
   public int pageNumber () {
      pageNumber =( virtualaddress / 4096);
      return pageNumber;
   }
   public int offSet() {
      offSet =(virtualaddress % 4096);
      return offSet;
}
}
```

Output:


Team Tools Window Help
Thread - Apache NetBeans IDE 18
Search (Ctrl+I)
288.6/700.5MB
simpleURL.java
ChilkatExample.java
part1.java
Main.java
DsaKeyGenerator.java
Test.java
Test1.java
Source History
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
* Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
*/
package com.mycompany.thread;
import java.util.Scanner;
public class Main {
public static void main(String args[])
{
part1 t1 = new part1();
Thread m1 = new Thread(r: t1);
Scanner sc = new Scanner(in: System.in);
m1.start();
try {
m1.join();
}
catch (InterruptedException e) {
}
}
}
com.mycompany.thread.Main
main
Notifications
Output
Inspector
Run (DateServer)
Run (Thread)
Prime numbers from 0 to 100 :
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89 97
BUILD SUCCESS
Total time: 1.690 s
Finished at: 2023-08-08T19:17:36+05:30
Run (DateServer)
14:1
INS Windows (CRLF)

Practical 5

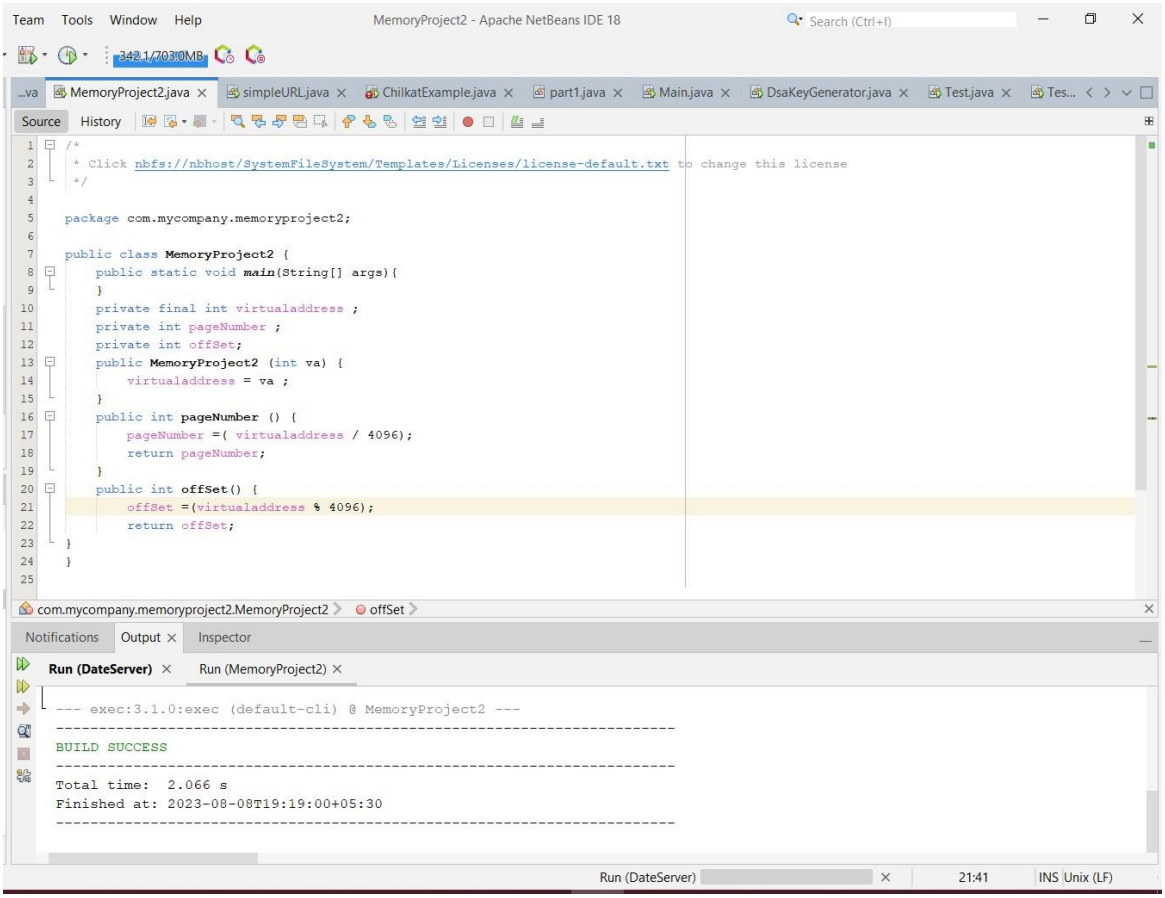
Aim: Assuming that a system has a 32-bit virtual address, write a Java program that is passed (1) the size of a page and (2) the virtual address. Your program will report the page number and offset of the given virtual address with the specified page size. Page sizes must be specified as a power of 2 and within the range 1024 — 16384 (inclusive). Assuming such a program is named Address, it would run as follows:
java Address 4096 19986
and the correct output would appear as:
The address 19986 contains:
page number = 4
offset = 3602.

Source Code:

```
public class part1 extends Thread {
   @Override
   public synchronized void run()
   {
      int i = 0;
      int num = 0;
      String primeNumbers = "";
      for (i = 1; i <= 100; i++) {
         int counter = 0;
         for (num = i; num >= 1; num--) {
            if (i % num == 0) {
               counter = counter + 1;
            }
         }

         if (counter == 2) {
            primeNumbers = primeNumbers + i + " ";
         }
      }
      System.out.println("\nPrime numbers from 0 to 100 : \n" + primeNumbers);
      System.out.println();
   }
}

import java.util.Scanner;

public class Main {
   public static void main(String args[])
   {
      part1 t1 = new part1();
```

```
        Thread m1 = new Thread(t1);
        Scanner sc = new Scanner(System.in);
        m1.start();
        try {
           m1.join();
        }
        catch (InterruptedException e) {
        }
    }
}
```

Output:



Practical 6

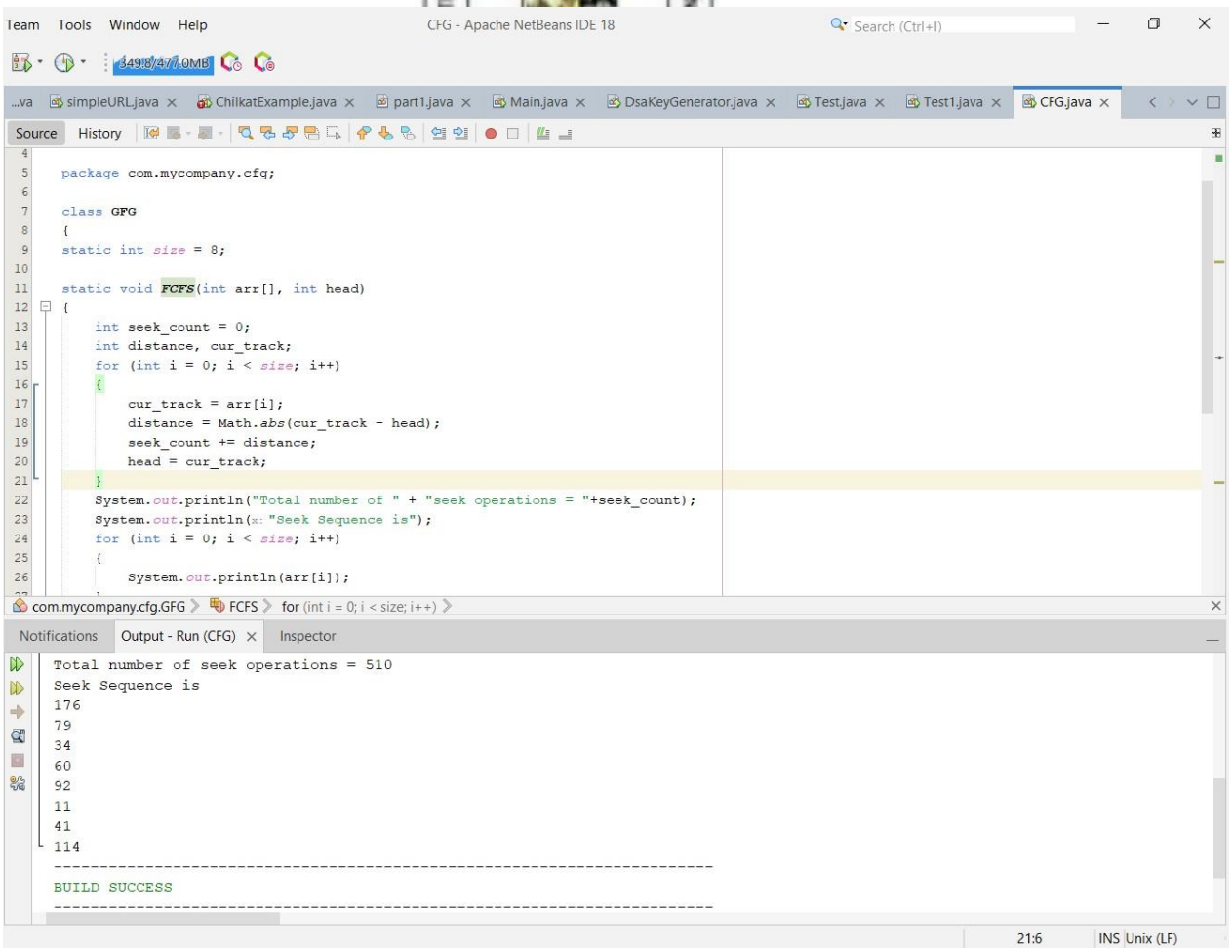
Aim: Write a Java program that simulates the following disk-scheduling algorithms. Design separate classes that implement the following scheduling algorithms:

a. FCFS

Source Code:

```
class GFG
{
static int size = 8;

static void FCFS(int arr[], int head)
{
   int seek_count = 0;
   int distance, cur_track;
   for (int i = 0; i < size; i++)
   {
      cur_track = arr[i];
      distance = Math.abs(cur_track - head);
      seek_count += distance;
      head = cur_track;
   }
   System.out.println("Total number of " + "seek operations = "+seek_count);
   System.out.println("Seek Sequence is");
   for (int i = 0; i < size; i++)
   {
      System.out.println(arr[i]);
   }
}

public static void main(String[] args)
{
   int arr[] = { 176, 79, 34, 60, 92, 11, 41, 114 };
   int head = 50;
   FCFS(arr, head);
}
}
```

Output:



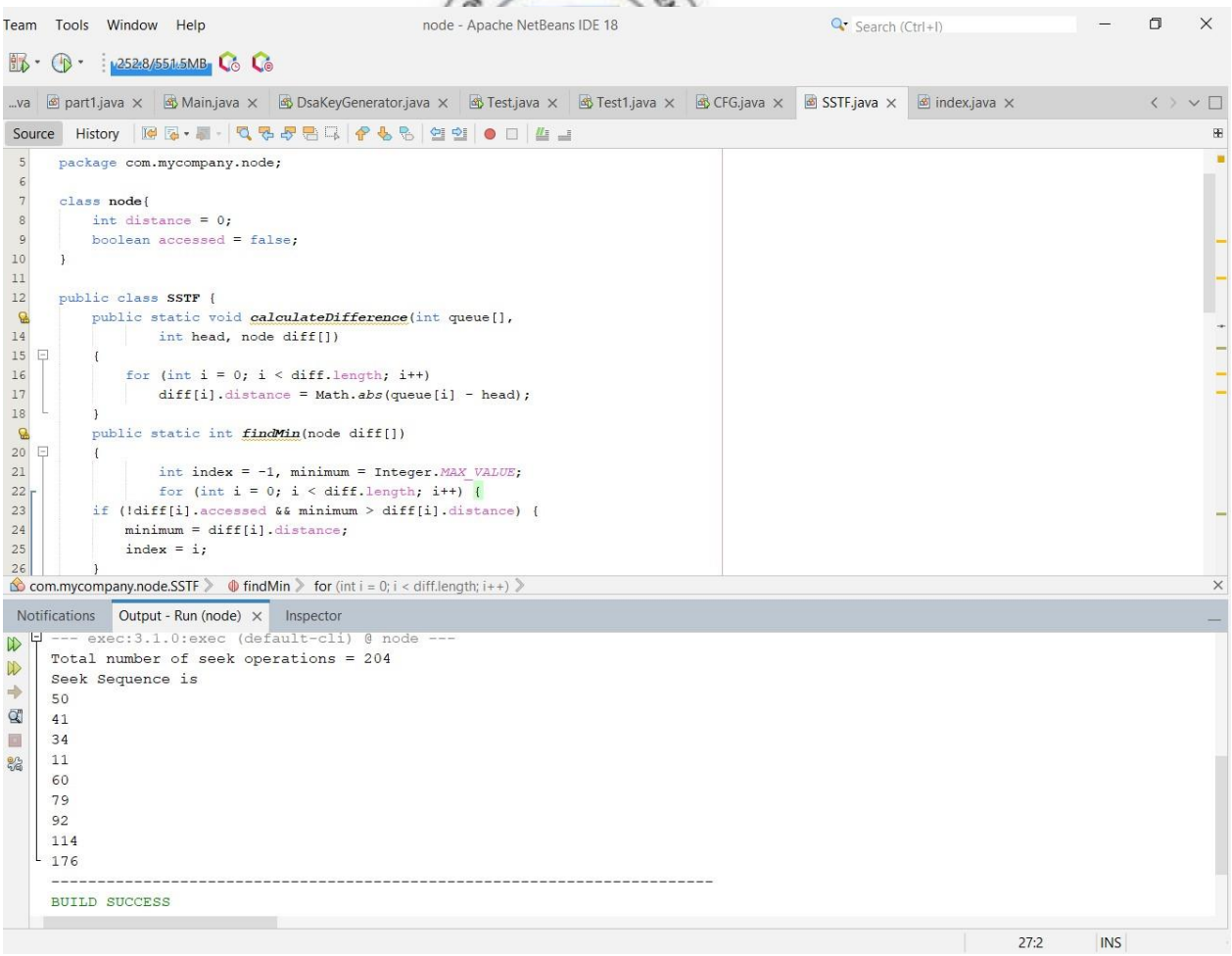
b. SSTF

Source Code:

```
package com.mycompany.node;

class node{
   int distance = 0;
   boolean accessed = false;
}

public class SSTF {
   public static void calculateDifference(int queue[],
         int head, node diff[])
   {
      for (int i = 0; i < diff.length; i++)
```

```
        diff[i].distance = Math.abs(queue[i] - head);
    }
    public static int findMin(node diff[])
    {
        int index = -1, minimum = Integer.MAX_VALUE;
        for (int i = 0; i < diff.length; i++) {
    if (!diff[i].accessed && minimum > diff[i].distance) {
      minimum = diff[i].distance;
      index = i;
    }
}
return index;
}

public static void shortestSeekTimeFirst(int request[], int head)
{
if (request.length == 0)
return;
node diff[] = new node[request.length];
for (int i = 0; i < diff.length; i++)
diff[i] = new node();
int seek_count = 0;
int[] seek_sequence = new int[request.length + 1];
for (int i = 0; i < request.length; i++) {
seek_sequence[i] = head;
calculateDifference(request, head, diff);
int index = findMin(diff);
diff[index].accessed = true;
seek_count += diff[index].distance;

head = request[index];
}
seek_sequence[seek_sequence.length - 1] = head;
System.out.println("Total number of seek operations = "+ seek_count);
System.out.println("Seek Sequence is");
for (int i = 0; i < seek_sequence.length; i++)
System.out.println(seek_sequence[i]);
}
public static void main(String[] args)
{
int arr[] = { 176, 79, 34, 60, 92, 11, 41, 114 };
shortestSeekTimeFirst(arr, 50);
}}
```

Output:



c.SCAN

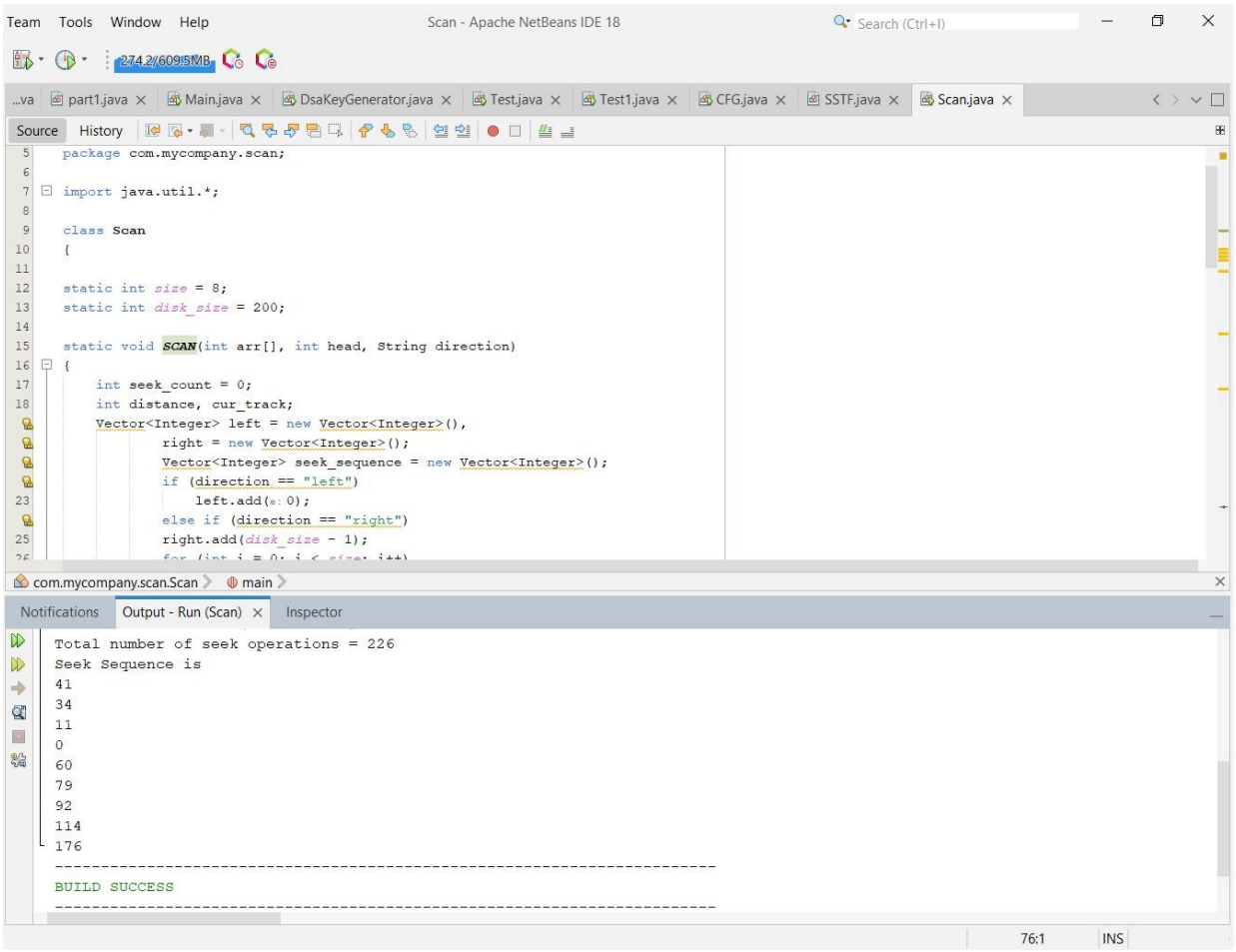
Source Code:

```
import java.util.*;

class Scan
{

static int size = 8;
static int disk_size = 200;

static void SCAN(int arr[], int head, String direction)
{
   int seek_count = 0;
   int distance, cur_track;
   Vector<Integer> left = new Vector<Integer>(),
         right = new Vector<Integer>();
```

```
        Vector<Integer> seek_sequence = new Vector<Integer>();
        if (direction == "left")
           left.add(0);
        else if (direction == "right")
        right.add(disk_size - 1);
        for (int i = 0; i < size; i++)
        {
           if (arr[i] < head)
              left.add(arr[i]);
           if (arr[i] > head)
              right.add(arr[i]);
        }
        Collections.sort(left);
Collections.sort(right);
int run = 2;
while (run-- >0)
{
   if (direction == "left")
   {
      for (int i = left.size() - 1; i >= 0; i--)
      {
         cur_track = left.get(i);
         seek_sequence.add(cur_track);
         distance = Math.abs(cur_track - head);
         seek_count += distance;
         head = cur_track;
      }
      direction = "right";
   }
   else if (direction == "right")
   {
      for (int i = 0; i < right.size(); i++)
      {
         cur_track = right.get(i);
         seek_sequence.add(cur_track);
         distance = Math.abs(cur_track - head);
         seek_count += distance;
         head = cur_track;
      }
      direction = "left";
   }
}
System.out.print("Total number of seek operations = "+ seek_count + "\n");
System.out.print("Seek Sequence is" + "\n");
for (int i = 0; i < seek_sequence.size(); i++)
{
```

```
    System.out.print(seek_sequence.get(i) + "\n");
}
}

public static void main(String[] args)
{
    int arr[] = {176, 79, 34, 60,92, 11, 41,114 };
int head = 50;
String direction = "left";
SCAN(arr, head, direction);
}
}
```

Output:



Practical 7

Aim: Write a program that implements the FIFO and LRU page-replacement algorithms presented in this chapter. First, generate a random page reference string where page numbers range from 0 to 9. Apply the random page-reference string to each algorithm, and record the number of page faults incurred by each algorithm. Implement the replacement algorithms so that the number of page frames can vary as well. Assume that demand paging is used. Design and implement two classes—LRU and FIFO—that extend ReplacementAlgorithm. Each of these classes will implement the insert() method, one class using the LRU page-replacement algorithm and the other using the FIFO algorithm. Test your algorithm with suitable Java programs.

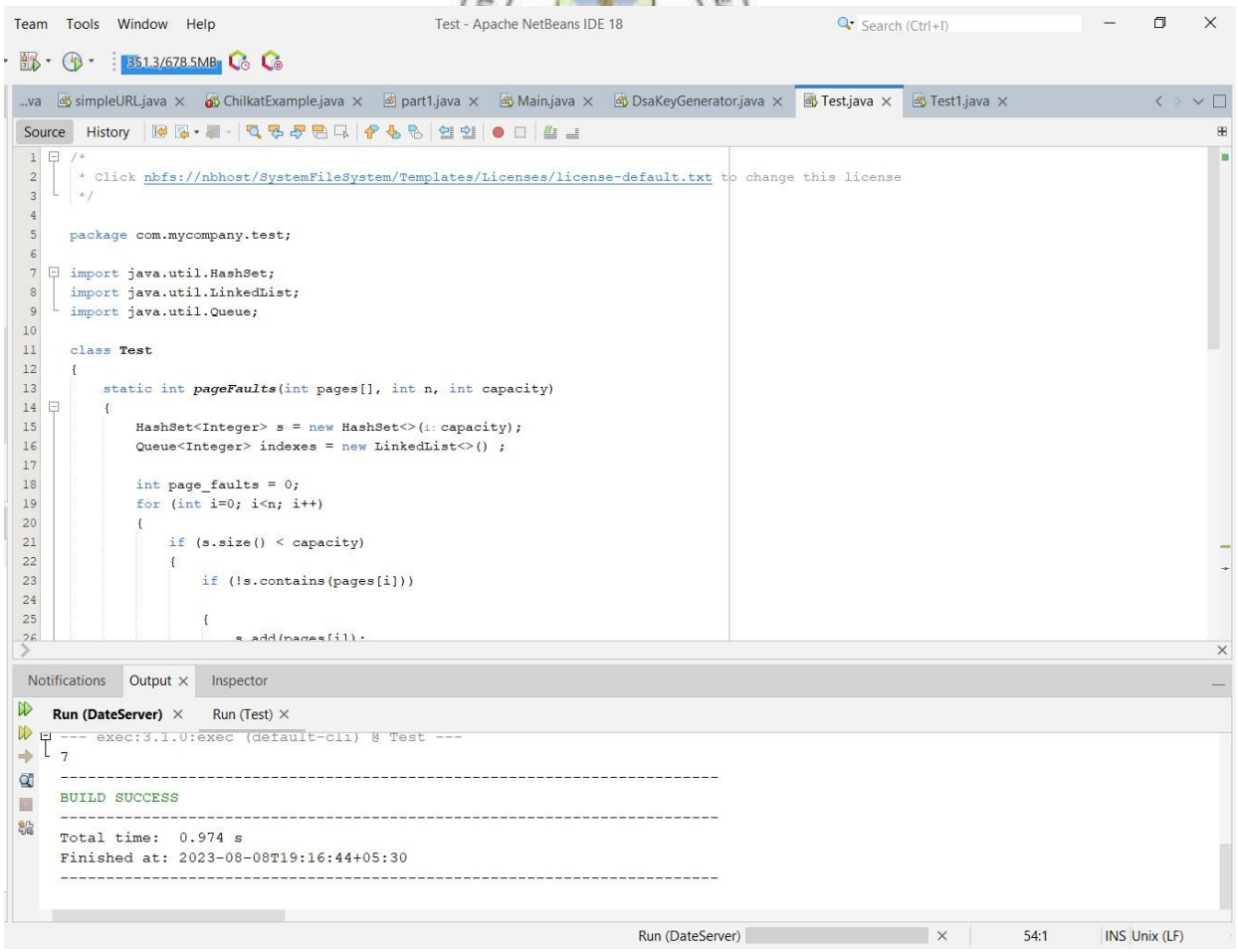
1) For FIFO

Source Code:

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;

class Test
{
   static int pageFaults(int pages[], int n, int capacity)
   {
      HashSet<Integer> s = new HashSet<>(capacity);
      Queue<Integer> indexes = new LinkedList<>() ;

      int page_faults = 0;
      for (int i=0; i<n; i++)
      {
         if (s.size() < capacity)
         {
            if (!s.contains(pages[i]))

            {
               s.add(pages[i]);
               page_faults++;
```

```
            indexes.add(pages[i]);
          }
        }
        else
        {
          if (!s.contains(pages[i]))

          {
          int val = indexes.peek();
          indexes.poll();
          s.remove(val);
          s.add(pages[i]);
          indexes.add(pages[i]);
          page_faults++;
          }
        }
      }
      return page_faults;
    }
    public static void main(String args[])
    {
      int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
      int capacity = 4;
      System.out.println(pageFaults(pages, pages.length, capacity));
    }
}
```

Output:



2) For LRU

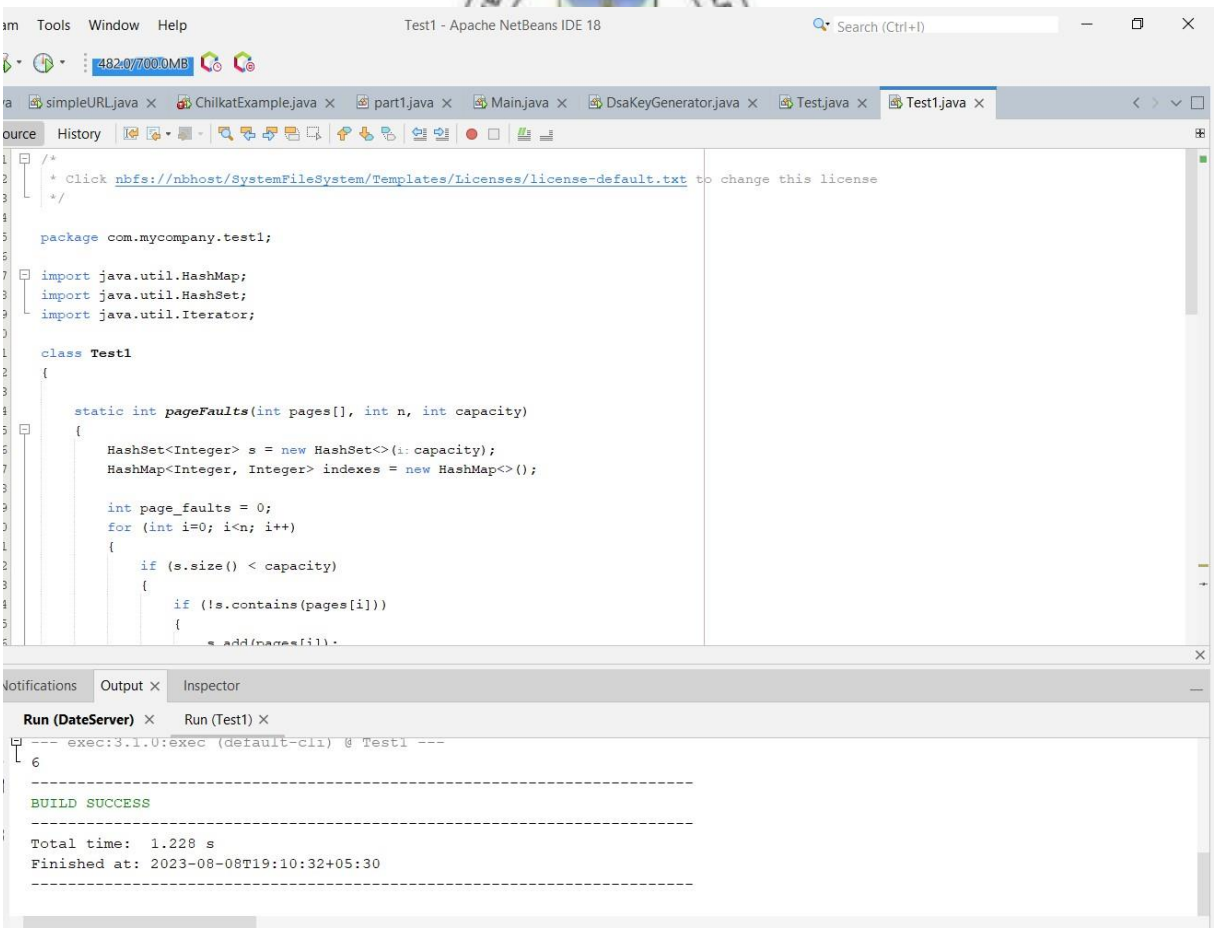
Source Code:

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;

class Test1
{

   static int pageFaults(int pages[], int n, int capacity)
```

```
{
   HashSet<Integer> s = new HashSet<>(capacity);
   HashMap<Integer, Integer> indexes = new HashMap<>();

   int page_faults = 0;
   for (int i=0; i<n; i++)
   {
      if (s.size() < capacity)
      {
         if (!s.contains(pages[i]))
         {
            s.add(pages[i]);

            page_faults++;

         }
         indexes.put(pages[i], i);
      }
      else
      {
         if (!s.contains(pages[i]))
         {

            int lru = Integer.MAX_VALUE, val=Integer.MIN_VALUE;
            Iterator<Integer> itr = s.iterator();
            while (itr.hasNext()) {
               int temp = itr.next();
               if (indexes.get(temp) < lru)
               {
                  lru = indexes.get(temp);
                  val = temp;
               }

            }
```

```
                s.remove(val);
                indexes.remove(val);
                s.add(pages[i]);
                page_faults++;
            }
            indexes.put(pages[i], i);
        }
    }
    return page_faults;
  }


public static void main(String args[])
  {
    int pages[] = {7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2};
    int capacity = 4;
    System.out.println(pageFaults(pages, pages.length, capacity));
  }
}
```

Output:


Tools Window Help
Test1 - Apache NetBeans IDE 18
Search (Ctrl+I)
482.0/700.0MB
simpleURL.java
ChilkatExample.java
part1.java
Main.java
DsaKeyGenerator.java
Test.java
Test1.java
History
/*
* Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
*/
package com.mycompany.test1;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Iterator;
class Test1
{
static int pageFaults(int pages[], int n, int capacity)
{
HashSet<Integer> s = new HashSet<>(i: capacity);
HashMap<Integer, Integer> indexes = new HashMap<>();
int page_faults = 0;
for (int i=0; i<n; i++)
{
if (s.size() < capacity)
{
if (!s.contains(pages[i]))
{
Notifications
Output
Inspector
Run (DateServer)
Run (Test1)
--- exec:3.1.0:exec (default-cli) @ Test1 ---
6
BUILD SUCCESS
Total time: 1.228 s
Finished at: 2023-08-08T19:10:32+05:30

Practical 9

Aim: Write Android activity that includes each of the fundamental lifecycle methods.

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities. Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it.

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity is not visible to the user.
onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed.

```
package tryexample.activitylifecycle;
import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d(“lifecycle”,“onCreate invoked”);

   }
   @Override
```

```
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
       super.onRestart();
       Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");
 }
}
```

Output: