## **Practical No. – 1**

**Aim:** Producer and consumer problem

### **Program:**

public class ProducerConsumer {

public static void main(String[] args) {

// Create a shared Shop object

Shop c = new Shop();

// Create and start a Producer thread

Producer p1 = new Producer(c, 1);

p1.start();

// Create and start a Consumer thread

Consumer c1 = new Consumer(c, 1);

c1.start();

}

}

// Shop class represents a shared resource where producers put materials and consumers get materials

class Shop {

private int materials;

private boolean available = false;

// Consumer method to get materials from the shop

public synchronized int get() {

while (!available) {

try {

// If materials are not available, the consumer waits

until materials are put by the producer

wait();

} catch (InterruptedException ie) {

// Handle interrupted exception if it occurs during

waiting

ie.printStackTrace();

}

}

// When materials are available, the consumer takes them and

notifies waiting threads (producers)

available = false;

notifyAll();

return materials;

}

// Producer method to put materials into the shop

public synchronized void put(int value) {

while (available) {

try {

// If materials are available, the producer waits

until they are consumed by the consumer

wait();

} catch (InterruptedException ie) {

// Handle interrupted exception if it occurs during

waiting

ie.printStackTrace();

}

}

// When materials are consumed, the producer puts new

materials and notifies waiting threads (consumers)

materials = value;

available = true;

notifyAll();

}

}

// Consumer class represents a thread that consumes materials from the shop

class Consumer extends Thread {

private Shop shop;

private int number;

public Consumer(Shop c, int number) {

shop = c;

this.number = number;

}

public void run() {

int value = 0;

for (int i = 0; i < 10; i++) {

// The consumer gets materials from the shop and prints

the output

value = shop.get();

System.out.println("Consumer consumed " + this.number + "

value and got: " + value);

}

}

}

// Producer class represents a thread that produces and puts materials into the shop

class Producer extends Thread {

private Shop shop;

private int number;

public Producer(Shop c, int number) {

shop = c;

this.number = number;

}

public void run() {

for (int i = 0; i < 10; i++) {

// The producer puts materials into the shop and prints

the output

shop.put(i);

System.out.println("Producer produced " + this.number + "

value and put: " + i);

try {

// The producer sleeps for a random time (up to 100

milliseconds) to simulate production time

sleep((int) (Math.random() \* 100));

} catch (InterruptedException ie) {

// Handle interrupted exception if it occurs during

sleeping

ie.printStackTrace();

}

}

}

}

### **Output:**

Producer produced 1 value and put: 0

Consumer consumed 1 value and got: 0

Consumer consumed 1 value and got: 1

Producer produced 1 value and put: 1

Producer produced 1 value and put: 2

Consumer consumed 1 value and got: 2

Producer produced 1 value and put: 3

Consumer consumed 1 value and got: 3

Consumer consumed 1 value and got: 4

Producer produced 1 value and put: 4

Producer produced 1 value and put: 5

Consumer consumed 1 value and got: 5

Consumer consumed 1 value and got: 6

Producer produced 1 value and put: 6

Consumer consumed 1 value and got: 7

Producer produced 1 value and put: 7

Producer produced 1 value and put: 8

Consumer consumed 1 value and got: 8

Consumer consumed 1 value and got: 9

Producer produced 1 value and put: 9

## **Practical No. - 2**

**Aim:** Determine submission of non- negative number using multithreading

### **Program:**

import java.util.Scanner;

public class Summation {

    public static void main(String[] args) {

        try {

            int n;

            Scanner s = new Scanner(System.in);

            System.out.print("Enter the value: ");

            n = s.nextInt(); // Read the user input

            Job j1 = new Job(n); // Create a new Job object with the

user input as the parameter

        } catch (Exception e) {

            // If any exception occurs during input or job creation,

this block will execute

            System.out.println("Some process failed to complete...");

            System.out.println("Please contact the system admin...");

        }

    }

}

// Class representing a job that calculates the summation of numbers from 1 to a given input value

class Job implements Runnable {

    int a1; // Variable to store the input value

    Thread t; // Thread to run the job

    // Constructor to create a Job object and start a new thread for

this job

    Job(int a) {

        a1 = a;

        t = new Thread(this);

        t.start(); // Start the thread and execute the run() method

    }

    // The run() method is called when the thread starts running

    public void run() {

        int b = 0; // Variable to store the summation result

        try {

            // Calculate the summation of numbers from 1 to the input

value (a1)

            for (int i = 1; i <= a1; i++) {

                b = b + i;

                Thread.sleep(100); // Add a delay of 100 milliseconds

to simulate some processing

            }

            // Print the result inside the 'try' block

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

            System.out.println("Job is over");

        } catch (InterruptedException e) {

            // If the thread is interrupted during the sleep, this

block will execute

            System.out.println("The job has been interrupted...");

        }

    }

}

### **Output:**

Enter the value: 10

The summation is: 55

Job is over

Enter the value: 5

The summation is: 15

Job is over

## **Practical No. - 3**

**Aim:** Write a multithread program that outputs prime number

### **Program:**

import java.util.Scanner;

// Create a class named "Job" that implements the Runnable interface

class Job implements Runnable {

    int a1;        // Declare an integer variable to store a number

    Thread t;      // Declare a Thread object for concurrent execution

    // Constructor to initialize the number and start a new thread

    Job(int a) {

        a1 = a;       // Assign the input number to the instance

        variable

        t = new Thread(this);  // Create a new thread that runs the

        "run" method of this class

        t.start();   // Start the thread's execution

    }

    // The "run" method is called when the thread starts executing

    public void run() {

        try {

            int i, k = 0;

            for (i = 2; i < a1; i++) {

                Thread.sleep(100);  // Pause the thread for 100

                milliseconds

                if (a1 % i == 0) {

                    System.out.println("Number is not prime");

                    k = 1;         // Set "k" to 1 to indicate the

                    number is not prime

                    break;         // Exit the loop since we found a

                    factor

                }

            }

            if (k == 0) {

                System.out.println("Number is prime");  // If "k" is

                still 0, the number is prime

            }

            System.out.println("Job is over");  // This message is

            printed when the thread completes its task

        } catch (InterruptedException e) {

            System.out.println("The job has been interrupted"); //

            Handle interruptions gracefully

        }

    }

}

public class Prime {

    public static void main(String args[]) {

        try {

            int n;

            Scanner s = new Scanner(System.in);

            System.out.print("Enter the value: ");

            n = s.nextInt();  // Read an integer from the user

            Job ji = new Job(n);  // Create a Job object with the

            user-provided number

        } catch (Exception e) {

            System.out.println("Some process failed to complete...");

            System.out.println("Please contact the system admin...");

        }

    }

}

### **Output:**

Enter the value: 11

Number is prime

Job is over

Enter the value: 12

Number is not prime

Job is over

## **Practical No. - 4**

**Aim:** Write a multithread program that outputs finocchi series

### **Program:**

import java.io.\*;

import java.util.Scanner;

// Create a class named "job" that implements the Runnable interface

class job implements Runnable {

int a1; // Declare an integer variable to store the

    number of Fibonacci numbers to generate

Thread t; // Declare a Thread object for concurrent

    execution

job(int a) {

a1 = a; // Assign the input number to the instance

        variable

t = new Thread(this); // Create a new thread that runs the

        "run" method of this class

t.start(); // Start the thread's execution

}

// The "run" method is called when the thread starts executing

public void run() {

int t1 = 0, t2 = 1;

try {

int i;

for (i = 1; i <= a1; ++i) {

Thread.sleep(100); // Pause the thread for 100

                milliseconds

System.out.print(t1 + " "); // Print the current

                Fibonacci number

int sum = t1 + t2;

t1 = t2;

t2 = sum;

}

System.out.println("\nJob is over!!");

} catch (InterruptedException e) {

System.out.println("The job has been interrupted"); //

            Handle interruptions gracefully

}

}

}

public class fibonacci {

public static void main(String[] args) {

try {

int n;

Scanner s = new Scanner(System.in);

System.out.print("Enter the value: ");

n = s.nextInt(); // Read an integer from the user

job j1 = new job(n); // Create a job object with the user-

            provided number

} catch (Exception e) {

System.out.println("Some process failed to complete");

System.out.println("Please contact the system admin");

}

}

}

### **Output:**

Enter the value: 10

0 1 1 2 3 5 8 13 21 34

Job is over!!

Enter the value: 15

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377

Job is over!!

## **Practical No. - 5**

**Aim:** Write program to contradict the barber and customer using java  synchronization (sleeping barber problem)

### **Program:**

import java.util.Date;

import java.util.LinkedList;

import java.util.List;

import java.util.concurrent.TimeUnit;

public class SleepingBarber {

    public static void main(String args[]) {

        // Create an instance of the barbershop.

        Bshop shop = new Bshop();

        // Create a barber and a customer generator, passing the shop

        instance.

        Barber barber = new Barber(shop);

        CustomerGenerator cg = new CustomerGenerator(shop);

        // Create threads for the barber and customer generator.

        Thread thbarber = new Thread(barber);

        Thread thcg = new Thread(cg);

        // Start the threads.

        thcg.start();

        thbarber.start();

    }

}

class Barber implements Runnable {

    Bshop shop;

    public Barber(Bshop shop) {

        this.shop = shop;

    }

    public void run() {

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

        while (true) {

            // The barber keeps cutting hair as long as there are

            customers in the shop.

            shop.cutHair();

        }

    }

}

class Customer implements Runnable {

    String name;

    Date inTime;

    Bshop shop;

    public Customer(Bshop shop) {

        this.shop = shop;

    }

    public String getName() {

        return name;

    }

    public Date getIntime() {

        return inTime;

    }

    public void setName(String name) {

        this.name = name;

    }

    public void setIntime(Date inTime) {

        this.inTime = inTime;

    }

    public void run() {

        // When a customer runs, they go for a hair cut.

        goForHairCut();

    }

    // This method adds the customer to the barbershop.

    private synchronized void goForHairCut() {

        shop.add(this);

    }

}

class CustomerGenerator implements Runnable {

    Bshop shop;

    private static int customerCount = 0;

    public CustomerGenerator(Bshop shop) {

        this.shop = shop;

    }

    public void run() {

        while (true) {

            // Generate a new customer and set their arrival time.

            Customer customer = new Customer(shop);

            customer.setIntime(new Date());

            // Generate a unique customer name.

            String customerName = "Customer " + customerCount++;

            customer.setName(customerName);

            Thread thcustomer = new Thread(customer);

            // Start the customer thread.

            thcustomer.start();

            try {

                // Wait for a random time (up to 10 seconds) before

                generating the next customer.

                TimeUnit.SECONDS.sleep((long) (Math.random() \* 10));

            } catch (InterruptedException iex) {

                iex.printStackTrace();

            }

        }

    }

}

class Bshop {

    int nchair;

    List<Customer> listCustomer;

    public Bshop() {

        // Initialize the barbershop with 2 chairs and an empty

        customer list.

        nchair = 2;

        listCustomer = new LinkedList<Customer>();

    }

    public void cutHair() {

        Customer customer;

        // The barber waits for the lock on the customer list.

        System.out.println("Barber is waiting for lock");

        synchronized (listCustomer) {

            // If the customer list is empty, the barber waits for a

            customer.

            while (listCustomer.isEmpty()) {

                System.out.println("Barber is waiting for customer");

                try {

                    listCustomer.wait();

                } catch (InterruptedException iex) {

                    iex.printStackTrace();

                }

            }

            // When a customer is found in the queue, remove them from

            the list.

            System.out.println("Barber found a customer in the

            queue");

            customer = listCustomer.remove(0);

        }

        long duration = 0;

        try {

            // The barber simulates cutting hair for a random duration

            (up to 10 seconds).

            System.out.println("Cutting hair of customer: " +

            customer.getName());

            duration = (long) (Math.random() \* 10);

            TimeUnit.SECONDS.sleep(duration);

        } catch (InterruptedException iex) {

            iex.printStackTrace();

        }

        // After cutting hair, the barber informs that the customer's

        hair is cut.

        System.out.println("Completed cutting hair of customer: " +

        customer.getName() + " in " + duration + " seconds.");

    }

    public void add(Customer customer) {

        // When a customer enters the shop, their arrival time is

        displayed.

        System.out.println("Customer: " + customer.getName() + "

        entering the shop at " + customer.getIntime());

        synchronized (listCustomer) {

            // If there are no available chairs, the customer leaves

            the shop.

            if (listCustomer.size() == nchair) {

                System.out.println("No chair available for customer "

                + customer.getName());

                System.out.println("Customer " + customer.getName() +

                " exits..");

                return;

            }

            // If there is an available chair, the customer takes it

            and is added to the list.

            listCustomer.add(customer);

            System.out.println("Customer: " + customer.getName() + "

            got the chair.");

            // If this is the first customer in the list, notify the

            barber that a customer is waiting.

            if (listCustomer.size() == 1) {

                listCustomer.notify();

            }

        }

    }

}

### **Output:**

Barber started..

Barber is waiting for lock

Barber is waiting for customer

Customer: Customer 0 entering the shop at Tue Sep 19 15:17:51 IST 2023

Customer: Customer 0 got the chair.

Barber found a customer in the queue

Cutting hair of customer: Customer 0

Customer: Customer 1 entering the shop at Tue Sep 19 15:17:53 IST 2023

Customer: Customer 1 got the chair.

Customer: Customer 2 entering the shop at Tue Sep 19 15:17:59 IST 2023

Customer: Customer 2 got the chair.

Completed cutting hair of customer: Customer 0 in 8 seconds.

Barber is waiting for lock

Barber found a customer in the queue

Cutting hair of customer: Customer 1

Customer: Customer 3 entering the shop at Tue Sep 19 15:18:08 IST 2023

Customer: Customer 3 got the chair.

Customer: Customer 4 entering the shop at Tue Sep 19 15:18:09 IST 2023

No chair available for customer Customer 4

Customer Customer 4 exits..

Completed cutting hair of customer: Customer 1 in 9 seconds.

Barber is waiting for lock

Barber found a customer in the queue

Cutting hair of customer: Customer 2

Completed cutting hair of customer: Customer 2 in 7 seconds.

Barber is waiting for lock

Barber found a customer in the queue

Cutting hair of customer: Customer 3

Completed cutting hair of customer: Customer 3 in 0 seconds.

## **Practical No. - 6**

**Aim:** Implement FCFS scheduling algorithm in java

### **Program:**

import java.util.\*;

public class FCFS {

    public static void main(String args[]) {

        Scanner sc = new Scanner(System.in);

        System.out.print("Enter the number of processes: ");

        int n = sc.nextInt();

        int pid[] = new int[n];

        int ar[] = new int[n];

        int bt[] = new int[n];

        int ct[] = new int[n];

        int ta[] = new int[n];

        int wt[] = new int[n];

        int temp;

        float avgwt = 0, avgta = 0;

        // Input process arrival time and burst time

        for (int i = 0; i < n; i++) {

            System.out.print("Enter process " + (i + 1) + " arrival

            time: ");

            ar[i] = sc.nextInt();

            System.out.print("Enter process " + (i + 1) + " burst

            time: ");

            bt[i] = sc.nextInt();

            pid[i] = i + 1; // Assign process IDs

        }

        // Sort processes based on their arrival times using bubble

        sort

        for (int i = 0; i < n; i++) {

            for (int j = 0; j < n - (i + 1); j++) {

                if (ar[j] > ar[j + 1]) {

                    // Swap arrival time, burst time, and process IDs

                    temp = ar[j];

                    ar[j] = ar[j + 1];

                    ar[j + 1] = temp;

                    temp = bt[j];

                    bt[j] = bt[j + 1];

                    bt[j + 1] = temp;

                    temp = pid[j];

                    pid[j] = pid[j + 1];

                    pid[j + 1] = temp;

                }

            }

        }

        // Calculate completion time, turnaround time, and waiting

        time for each process

        for (int i = 0; i < n; i++) {

            if (i == 0) {

                ct[i] = ar[i] + bt[i];

            } else {

                if (ar[i] > ct[i - 1]) {

                    ct[i] = ar[i] + bt[i];

                } else

                    ct[i] = ct[i - 1] + bt[i];

            }

            ta[i] = ct[i] - ar[i];

            wt[i] = ta[i] - bt[i];

            avgwt += wt[i];

            avgta += ta[i];

        }

        // Display the process details

        System.out.println("\nPID  Arrival  Burst  Complete  Turnaroun

        d  Waiting");

        for (int i = 0; i < n; i++) {

            System.out.println(pid[i] + "\t" + ar[i] + "\t" + bt[i] +

            "\t" + ct[i] + "\t" + ta[i] + "\t\t" + wt[i]);

        }

        sc.close();

        // Calculate and display average waiting time and average

        turnaround time

        System.out.println("\nAverage Waiting Time: " + (avgwt / n));

        System.out.println("Average Turnaround Time: " + (avgta / n));

    }

}

### **Output:**

Enter the number of processes: 4

Enter process 1 arrival time: 1

Enter process 1 burst time: 2

Enter process 2 arrival time: 0

Enter process 2 burst time: 2

Enter process 3 arrival time: 3

Enter process 3 burst time: 4

Enter process 4 arrival time: 5

Enter process 4 burst time: 6

PID Arrival Burst Complete Turnaround Waiting

2 0 2 2 2 0

1 1 2 4 3 1

3 3 4 8 5 1

4 5 6 14 9 3

Average Waiting Time: 1.25

Average Turnaround Time: 4.75

## **Practical No. - 7**

**Aim:** Implement shortest job fast scheduling algorithm in java

### **Program:**

import java.util.\*;

public class SJF {

public static void main(String args[]) {

Scanner sc = new Scanner(System.in);

System.out.print("Enter the number of processes: ");

int n = sc.nextInt();

int pid[] = new int[n];

int at[] = new int[n];

int bt[] = new int[n];

int ta[] = new int[n];

int wt[] = new int[n];

int ct[] = new int[n];

int f[] = new int[n];

int st = 0, tot = 0;

float avgwt = 0, avgta = 0;

// Input process arrival time and burst time

for (int i = 0; i < n; i++) {

System.out.print("Enter process " + (i + 1) + " arrival

            time: ");

at[i] = sc.nextInt();

System.out.print("Enter process " + (i + 1) + " burst

            time: ");

bt[i] = sc.nextInt();

pid[i] = i + 1;

f[i] = 0;

}

boolean a = true;

while (true) {

int c = n, min = 999;

if (tot == n)

break;

// Find the shortest job that has arrived and not yet

            completed

for (int i = 0; i < n; i++) {

if (at[i] <= st && f[i] == 0 && bt[i] < min) {

min = bt[i];

c = i;

}

}

if (c == n)

st++; // If no eligible job found, increment time

else {

ct[c] = st + bt[c];

st += bt[c];

ta[c] = ct[c] - at[c];

wt[c] = ta[c] - bt[c];

f[c] = 1;

tot++;

}

}

// Display the process details and calculate averages

System.out.println("\nPID\tArrival\tBurst\tComplete\tTurnaround\tWaiting");

for (int i = 0; i < n; i++) {

avgwt += wt[i];

avgta += ta[i];

System.out.println(pid[i] + "\t" + at[i] + "\t" + bt[i] +

            "\t" + ct[i] + "\t\t" + ta[i] + "\t\t" + wt[i]);

}

System.out.println("\nAverage turnaround time is " + (float)

        (avgta / n));

System.out.println("Average waiting time is " + (float) (avgwt

        / n));

sc.close(); // Close the scanner

}

}

### **Output:**

Enter no of processes: 3

Enter process 1 Arrival time: 1

Enter process 1 Burst time: 2

Enter process 2 Arrival time: 3

Enter process 2 Burst time: 4

Enter process 3 Arrival time: 5

Enter process 3 Burst time: 6

PID Arrival Burst Complete Turnaround Waiting

1 1 2 3 2 0

2 3 4 7 4 0

3 5 6 13 8 2

Average turnaround time is 4.6666665

Average waiting time is 0.6666667

Enter no of processes: 4

Enter process 1 Arrival time: 1

Enter process 1 Burst time: 2

Enter process 2 Arrival time: 3

Enter process 2 Burst time: 4

Enter process 3 Arrival time: 5

Enter process 3 Burst time: 6

Enter process 4 Arrival time: 7

Enter process 4 Burst time: 8

PID Arrival Burst Complete Turnaround Waiting

1 1 2 3 2 0

2 3 4 7 4 0

3 5 6 13 8 2

4 7 8 21 14 6

Average turnaround time is 7.0

Average waiting time is 2.0

## **Practical No. - 8**

**Aim:** Implement round robin scheduling algorithm in java

### **Program:**

import java.util.Arrays;

public class RR {

    static void findWaitingTime(int processes[], int n, int bt[], int

    wt[], int quantum) {

        int rem\_bt[] = new int[n];

        // Initialize remaining burst times as the original burst

        times

        for (int i = 0; i < n; i++)

            rem\_bt[i] = bt[i];

        int t = 0; // Current time

        while (true) {

            boolean done = true; // To check if all processes are done

            // Traverse all processes

            for (int i = 0; i < n; i++) {

                if (rem\_bt[i] > 0) {

                    done = false;

                    // If remaining burst time is more than the

                    quantum, decrease it by quantum

                    if (rem\_bt[i] > quantum) {

                        t += quantum;

                        rem\_bt[i] -= quantum;

                    }

                    // If remaining burst time is less than or equal

                    to the quantum, finish the process

                    else {

                        t = t + rem\_bt[i];

                        wt[i] = t - bt[i];

                        rem\_bt[i] = 0;

                    }

                }

            }

            // If all processes are done, exit the loop

            if (done == true)

                break;

        }

    }

    static void findTurnAroundTime(int processes[], int n, int bt[],

    int wt[], int tat[]) {

        for (int i = 0; i < n; i++)

            tat[i] = bt[i] + wt[i];

    }

    static void findavgTime(int processes[], int n, int bt[], int

    quantum) {

        int wt[] = new int[n], tat[] = new int[n];

        int total\_wt = 0, total\_tat = 0;

        // Calculate waiting time for all processes

        findWaitingTime(processes, n, bt, wt, quantum);

        // Calculate turnaround time for all processes

        findTurnAroundTime(processes, n, bt, wt, tat);

        // Print the table

        System.out.println("Processes\tBurst Time\tWaiting

        Time\tTurnaround Time");

        for (int i = 0; i < n; i++) {

            total\_wt += wt[i];

            total\_tat += tat[i];

            System.out.println(processes[i] + "\t\t" + bt[i] + "\t\t"

            + wt[i] + "\t\t" + tat[i]);

        }

        // Calculate and print average waiting time and average

        turnaround time

        float avg\_wt = (float) total\_wt / n;

        float avg\_tat = (float) total\_tat / n;

        System.out.println("\nAverage Waiting Time = " + avg\_wt);

        System.out.println("Average Turnaround Time = " + avg\_tat);

    }

    public static void main(String[] args) {

        int processes[] = {1, 2, 3};

        int n = processes.length;

        int burst\_time[] = {10, 5, 8};

        int quantum = 2;

        findavgTime(processes, n, burst\_time, quantum);

    }

}

### **Output:**

Processes Burst Time Waiting Time Turnaround Time

1 10 13 23

2 5 10 15

3 8 13 21

Average Waiting Time = 12.0

Average Turnaround Time = 19.666666

## **Practical No. - 9**

**Aim:** Implement fifo page in java

### **Program:**

### **Output:**

## **Practical No. - 10**

**Aim:** Implement LrU page replacement in java

### **Program:**

### **Output:**

## **Practical No. - 11**

**Aim:** Implement bankers algorithm in java

### **Program:**

import java.io.\*;

public class BankersAlogrithm {

    static int safe\_sequence[];

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

        BufferedReader br = new BufferedReader(new

        InputStreamReader(System.in));

        System.out.print("Please enter the total number of resources:

        ");

        // Input: Total number of resources

        int res\_n = Integer.parseInt(br.readLine());

        int res[] = new int[res\_n];

        int cur\_avail[] = new int[res\_n];

        // Input: Total instances for each resource and initialize

        current available resources

        for (int i = 0; i < res\_n; i++) {

            System.out.print("Enter total number of instances for

            resources " + (i + 1) + ": ");

            res[i] = Integer.parseInt(br.readLine());

            cur\_avail[i] = res[i];

        }

        System.out.print("\nEnter number of processes: ");

        int pros\_n = Integer.parseInt(br.readLine());

        safe\_sequence = new int[pros\_n];

        int max[][] = new int[res\_n][pros\_n];

        int alloc[][] = new int[res\_n][pros\_n];

        // Input: Maximum resource allocation for each process

        for (int i = 0; i < pros\_n; i++) {

            System.out.print("Enter the maximum string for process " +

            (i + 1) + ": ");

            String ip = br.readLine();

            for (int j = 0; j < res\_n; j++)

                max[j][i] =

                Integer.parseInt(String.valueOf(ip.charAt(j)));

        }

        // Input: Allocation matrix for each process and update

        current available resources

        for (int i = 0; i < pros\_n; i++) {

            System.out.print("Enter the allocation string for process

            " + (i + 1) + ": ");

            String ip = br.readLine();

            for (int j = 0; j < res\_n; j++) {

                alloc[j][i] =

                Integer.parseInt(String.valueOf(ip.charAt(j)));

                cur\_avail[j] = cur\_avail[j] - alloc[j][i];

            }

        }

        int need[][] = new int[res\_n][pros\_n];

        // Calculate the resource needs of each process

        for (int i = 0; i < pros\_n; i++) {

            for (int j = 0; j < res\_n; j++)

                need[j][i] = max[j][i] - alloc[j][i];

        }

        // Check if the system is in a safe state

        boolean safe = check\_state(need, alloc, cur\_avail, res\_n,

        pros\_n);

        System.out.println();

        if (safe) {

            System.out.print("The system is in a safe state.");

            System.out.print("The safe sequence is: ");

            for (int i = 0; i < pros\_n; i++)

                System.out.print("P" + (safe\_sequence[i] + 1) + " ");

            System.out.println();

        } else

            System.out.print("The system is not in a safe state.");

        if (safe) {

            System.out.println();

            System.out.print("Please enter the number of the process

            that is requesting more resources: ");

            int req\_n = Integer.parseInt(br.readLine()) - 1;

            int req[] = new int[res\_n];

            System.out.print("Please enter the request matrix: ");

            String ip = br.readLine();

            int need\_count = 0;

            int avl\_count = 0;

            // Input: Resource request for a process

            for (int i = 0; i < res\_n; i++) {

                req[i] =

                Integer.parseInt(String.valueOf(ip.charAt(i)));

                if (req[i] <= need[i][req\_n])

                    need\_count++;

                if (req[i] <= cur\_avail[i])

                    avl\_count++;

            }

            if (need\_count != res\_n)

                System.out.println("The request cannot be granted

                since requested resources are more than previously

                declared maximum.");

            if (avl\_count != res\_n)

                System.out.println("The request cannot be granted

                since the amount of resources requested are not

                available.");

            if (need\_count == res\_n && avl\_count == res\_n) {

                for (int i = 0; i < res\_n; i++) {

                    alloc[i][req\_n] += req[i];

                    need[i][req\_n] -= req[i];

                    cur\_avail[i] -= req[i];

                }

                safe = check\_state(need, alloc, cur\_avail, res\_n,

                pros\_n);

                System.out.println();

                if (safe) {

                    System.out.print("The system will be in a safe

                    state if the request is granted.");

                    System.out.print("The safe sequence is: ");

                    for (int i = 0; i < pros\_n; i++)

                        System.out.println("p" + (safe\_sequence[i] +

                        1) + " ");

                    System.out.println();

                } else

                    System.out.print("The system will not be in a safe

                    state if the request is granted.");

            }

        }

    }

    static boolean check\_state(int need[][], int alloc[][], int

    cur\_avail[], int res\_n, int pros\_n) {

        boolean marked[] = new boolean[pros\_n];

        int safe\_pos = 0;

        boolean safe = true;

        int avail[] = new int[res\_n];

        // Copy current available resources

        for (int i = 0; i < res\_n; i++)

            avail[i] = cur\_avail[i];

        // Check if the system is in a safe state

        while (safe\_pos < pros\_n && safe) {

            for (int i = 0; i < pros\_n; i++) {

                int c = 0;

                for (int j = 0; j < res\_n; j++) {

                    if (need[j][i] <= avail[j])

                        c++;

                }

                if ((c == res\_n) && (marked[i] == false)) {

                    for (int j = 0; j < res\_n; j++) {

                        avail[j] += alloc[j][i];

                    }

                    marked[i] = true;

                    safe\_sequence[safe\_pos] = i;

                    safe\_pos++;

                    break;

                }

                if (i == pros\_n - 1 && c < res\_n) {

                    safe = false;

                }

            }

        }

        return safe;

    }

}

### **Output:**

Please enter the total number of resources: 3

Enter total number of instances for resources 1: 10

Enter total number of instances for resources 2: 5

Enter total number of instances for resources 3: 7

Enter number of processes: 5

Enter the maximum string for process 1: 753

Enter the maximum string for process 2: 322

Enter the maximum string for process 3: 902

Enter the maximum string for process 4: 422

Enter the maximum string for process 5: 533

Enter the allocation string for process 1: 010

Enter the allocation string for process 2: 200

Enter the allocation string for process 3: 302

Enter the allocation string for process 4: 211

Enter the allocation string for process 5: 002

The system is in a safe state.

The safe sequence is:

P2 P4 P1 P3 P5

Please enter the number of the process that is requesting more resources: 4

Please enter the request matrix: 000

The system will be in a safe state if the request is granted.

The safe sequence is:

p2

p4

p1

p3

p5

Please enter the number of the process that is requesting more resources: 3

Please enter the request matrix: 903

The request cannot be granted since requested resources are more than previously declared maximum.

The request cannot be granted since the amount of resources requested are not available.