**Practical no 1   
Producer consumer problem**

public class ProducerConsumer

{

public static void main(String[] args)

{

Shop c = new Shop();

Producer p1 = new Producer(c, 1);

Consumer c1 = new Consumer(c, 1);

p1.start();

c1.start();

}

}

class Shop

{

private int materials;

private boolean available = false;

public synchronized int get()

{

while (available == false)

{

try

{

wait();

}

catch (InterruptedException ie)

{

}

}

available = false;

notifyAll();

return materials;

}

public synchronized void put(int value)

{

while (available == true)

{

try

{

wait();

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

materials = value;

available = true

notifyAll();

}

}

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++)

{

value = Shop.get();

System.out.println("Consumed value " + this.number+ " got: " + value);

}

}

}

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++)

{

Shop.put(i);

System.out.println("Produced value " + this.number+ " put: " + i);

try

{

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

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

**Practical no 2 summation**

import java.io.\*;

import java.util.Scanner;

class job implements Runnable

{

int a1;

Thread t;

job(int a)

{

a1=a;

t = new Thread(this);

t.start();

}

public void run()

{

int b=0;

try

{

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

{

b=b+i;

Thread.sleep(100);

}

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

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

}

catch(InterruptedException e)

{

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

}

}

}

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();

job j1 = new job(n);

}

catch(Exception e)

{

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

System.out.println("Plz contact system admin...");

}

}

}

**Practical 3 prime no**

import java.io.\*;

import java.util.Scanner;

class job implements Runnable

{

int a1;

Thread t;

job(int a)

{

a1=a;

t = new Thread(this);

t.start();

}

public void run()

{

try

{

int i,k,j;

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

{

Thread.sleep(100);

k=0;

for(j=2;j<i;j++)

{

if(i%j==0)

{

k=1;

break;

}

}

if(k==0)

{

System.out.println(i);

}

}

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

}

catch(InterruptedException e)

{

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

}

}

}

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();

job j1 = new job(n);

}

catch(Exception e)

{

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

System.out.println("Plz contact system admin...");

}

}

}

**Practical no 4 fibonacci series**

import java.io.\*;

import java.util.Scanner;

class job implements Runnable

{

int a1;

Thread t;

job(int a)

{

a1=a;

t = new Thread(this);

t.start();

}

public void run()

{

int t1 = 0, t2 = 1;

try

{

int i,k,j;

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

{

Thread.sleep(100);

System.out.println(t1+" ");

int sum = t1 + t2;

t1 = t2;

t2 = sum;

}

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

}

catch(InterruptedException e)

{

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

}

}

}

class Fibonacc

{

public static void main(String args[])

{

try

{

int n;

Scanner s=new Scanner(System.in);

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

n=s.nextInt();

job j1 = new job(n);

}

catch(Exception e)

{

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

System.out.println("Plz contact system admin...");

}

}

}

**Practical no 5 sleeping barber**

import java.util.Date;

import java.util.LinkedList;

import java.util.List;

import java.util.concurrent.TimeUnit;

public class SleepingBarber1 {

public static void main(String a[])

{

Bshop shop = new Bshop();

Barber barber = new Barber(shop);

CustomerGenerator cg = new CustomerGenerator(shop);

Thread thbarber = new Thread(barber);

Thread thcg = new Thread(cg);

thcg.start();

thbarber.start();

}

}

class Barber implements Runnable

{

Bshop shop;

public Barber(Bshop shop)

{

this.shop = shop;

}

public void run()

{

try

{

Thread.sleep(10000);

}

catch(InterruptedException iex)

{

iex.printStackTrace();

}

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

while(true)

{

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()

{

goForHairCut();

}

private synchronized void goForHairCut()

{

shop.add(this);

}

}

class CustomerGenerator implements Runnable

{

Bshop shop;

public CustomerGenerator(Bshop shop)

{

this.shop = shop;

}

public void run()

{

while(true)

{

Customer customer = new Customer(shop);

customer.setInTime(new Date());

Thread thcustomer = new Thread(customer);

customer.setName("Customer Thread "+thcustomer.getId());

thcustomer.start();

try

{

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

}

catch(InterruptedException iex)

{

iex.printStackTrace();

}

}

}

}

class Bshop

{

int nchair;

List<Customer> listCustomer;

public Bshop()

{

nchair = 3;

listCustomer = new LinkedList<Customer>();

}

public void cutHair()

{

Customer customer;

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

synchronized (listCustomer)

{

while(listCustomer.size()==0)

{

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

try

{

listCustomer.wait();

}

catch(InterruptedException iex)

{

iex.printStackTrace();

}

}

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

customer = (Customer)((LinkedList<?>)listCustomer).poll();

}

long duration=0;

try

{

System.out.println("Cuting hair of Customer : "+customer.getName());

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

TimeUnit.SECONDS.sleep(duration);

}

catch(InterruptedException iex)

{

iex.printStackTrace();

}

System.out.println("Completed Cuting hair of Customer : "+customer.getName() + " in "+duration+ " seconds.");

}

public void add(Customer customer)

{

System.out.println("Customer : "+customer.getName()+ " entering the shop at "+customer.getInTime());

synchronized (listCustomer)

{

if(listCustomer.size() == nchair)

{

System.out.println("No chair available for customer "+customer.getName());

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

return ;

}

((LinkedList<Customer>)listCustomer).offer(customer);

System.out.println("Customer : "+customer.getName()+ " got the chair.");

if(listCustomer.size()==1)

listCustomer.notify();

}

}

}

**Practical 6 FCFS**

import java.io.\*;

class MemMgmt

{

String frame\_sequence;

int mem\_block[];

MemMgmt(String fs,int n)

{

frame\_sequence = fs;

mem\_block = new int[n];

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

mem\_block[i]=-1;

}

void dispMemBlock()

{

System.out.print("|");

for(int i=0;i<mem\_block.length;i++)

{

System.out.print(mem\_block[i]+"|");

}

System.out.println();

}

void fcfs()

{

System.out.println("======= FCFS ===========");

String strpages[] = frame\_sequence.split(" ");

int pages[] = new int[strpages.length];

for(int i=0; i<strpages.length; i++)

pages[i] = Integer.parseInt(strpages[i]);

int mem\_block\_num=0, page\_faults=0;

System.out.println("Initial Memory layout...");

dispMemBlock();

for(int i=0; i<pages.length; i++)

{

boolean present=false;

for(int j=0;j<mem\_block.length;j++)

{

if(mem\_block[j] == pages[i])

{

present=true;

break;

}

}

if(!present)

{

mem\_block[mem\_block\_num] = pages[i];

mem\_block\_num++;

page\_faults++;

}

if(mem\_block\_num==mem\_block.length)

mem\_block\_num=0;

System.out.println("Loading page no." + (i+1) + ":" + pages[i]);

dispMemBlock();

}

System.out.println("Total Number of page faults:" + page\_faults);

}

}

class PFCFS

{

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

{

String frame\_sequence = "7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1";

MemMgmt m = new MemMgmt(frame\_sequence,3);

m.fcfs();

}

}

**Practical 7 SJF**

import java.util.\*;

public class SJF {

public static void main(String args[])

{

Scanner sc = new Scanner(System.in);

System.out.println ("enter no of process:");

int n = sc.nextInt();

int pid[] = new int[n];

int at[] = new int[n]; // at means arrival time

int bt[] = new int[n]; // bt means burst time

int ct[] = new int[n]; // ct means complete time

int ta[] = new int[n]; // ta means turn around time

int wt[] = new int[n]; //wt means waiting time

int f[] = new int[n]; // f means it is flag it checks process is completed or not

int st=0, tot=0;

float avgwt=0, avgta=0;

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

{

System.out.println ("enter process " + (i+1) + " arrival time:");

at[i] = sc.nextInt();

System.out.println ("enter process " + (i+1) + " brust 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) // total no of process = completed process loop will be terminated

break;

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

{

/\*

\* If i'th process arrival time <= system time and its flag=0 and burst<min

\* That process will be executed first

\*/

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

{

min=bt[i];

c=i;

}

}

/\* If c==n means c value can not updated because no process arrival time< system time so we increase the system time \*/

if (c==n)

st++;

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++;

}

}

System.out.println("\npid arrival brust complete turn waiting");

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

}

System.out.println ("\naverage tat is "+ (float)(avgta/n));

System.out.println ("average wt is "+ (float)(avgwt/n));

sc.close();

}

}

**Practical 8 round robin**

public class RR

{

// Method to find the waiting time for all

// processes

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

int bt[], int wt[], int quantum)

{

// Make a copy of burst times bt[] to store remaining

// burst times.

int rem\_bt[] = new int[n];

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

rem\_bt[i] = bt[i];

int t = 0; // Current time

// Keep traversing processes in round robin manner

// until all of them are not done.

while(true)

{

boolean done = true;

// Traverse all processes one by one repeatedly

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

{

// If burst time of a process is greater than 0

// then only need to process further

if (rem\_bt[i] > 0)

{

done = false; // There is a pending process

if (rem\_bt[i] > quantum)

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t += quantum;

// Decrease the burst\_time of current process

// by quantum

rem\_bt[i] -= quantum;

}

// If burst time is smaller than or equal to

// quantum. Last cycle for this process

else

{

// Increase the value of t i.e. shows

// how much time a process has been processed

t = t + rem\_bt[i];

// Waiting time is current time minus time

// used by this process

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

// As the process gets fully executed

// make its remaining burst time = 0

rem\_bt[i] = 0;

}

}

}

// If all processes are done

if (done == true)

break;

}

}

// Method to calculate turn around time

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

int bt[], int wt[], int tat[])

{

// calculating turnaround time by adding

// bt[i] + wt[i]

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

tat[i] = bt[i] + wt[i];

}

// Method to calculate average time

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;

// Function to find waiting time of all processes

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

// Function to find turn around time for all processes

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

// Display processes along with all details

System.out.println("Processes " + " Burst time " +

" Waiting time " + " Turn around time");

// Calculate total waiting time and total turn

// around time

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

{

total\_wt = total\_wt + wt[i];

total\_tat = total\_tat + tat[i];

System.out.println(" " + (i+1) + "\t\t" + bt[i] +"\t " +

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

}

System.out.println("Average waiting time = " +

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

System.out.println("Average turn around time = " +

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

}

// Driver Method

public static void main(String[] args)

{

// process id's

int processes[] = { 1, 2, 3};

int n = processes.length;

// Burst time of all processes

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

// Time quantum

int quantum = 2;

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

}

}

**Practical 9 FIFO**

import java.io.\*;

public class fifo

{

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

{

int f,p,num=0,pageHit=0;

int pages[];

int frame[];

boolean flag = true;

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

System.out.println("Enter number of frames :");

f=Integer.parseInt(br.readLine());

System.out.println("Enter number of pages :");

p = Integer.parseInt(br.readLine());

frame = new int[f];

pages = new int[p];

for (int i=0;i<f;i++)

{

frame[i]=-1;

}

System.out.println("Enter page number :");

for(int i=0;i<p;i++)

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

for(int i=0;i<p;i++)

{

flag=true;

int page = pages[i];

for(int j=0;j<f;j++)

{

if(frame[j]==page)

{

flag=false;

pageHit++;

break;

}

}

if(num==f)

num=0;

if(flag)

{

frame[num]=page;

num++;

}

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

for(int k=0;k<f;k++)

System.out.print(frame[k]+" ");

System.out.println();

}

System.out.println("No. of pages hit :"+pageHit);

}

}

**Practical 10 LRU**

import java.util.\*;

class LRU

{

Scanner sc=new Scanner(System.in);

int[] frame,page,present;

int size,pages,pf=0,flag=0,flag1=0;

LRU(int size,int pages)

{

this.size=size;

this.pages=pages;

frame=new int[size];

present=new int[size];

page=new int[pages];}

void get()

{

System.out.println("Enter pages");

for(int i=0;i<pages;i++)

page[i]=sc.nextInt();

for(int i=0;i<size;i++)

frame[i]=-1;}

int check(int x)

{

flag=-1;

for(int i=0;i<size;i++)

if(frame[i]==x)

{

flag=i;

break;}

return flag;}

int replace(int x)

{

int i;

for(i=0;i<size;i++)

present[i]=0;

flag1=0;

for(i=x-1;i>=0;i--)

{

if(check(page[i])>=0)

{

flag1++;

present[check(page[i])]=1;}

if(flag1==(size-1)) break;

}

for(i=0;i<size;i++)

if(present[i]==0)

{

flag1=i;

break;}

return i;

}

void lru()

{

for(int i=0;i<pages;i++)

{

if(i<size)

{

frame[i]=page[i];

pf++;

for(int j=0;j<size;j++)

System.out.print(frame[j]+" ");

System.out.println();}

else

{

if(check(page[i])==-1)

{

int r=replace(i);

frame[r]=page[i];

pf++;

for(int j=0;j<size;j++)

System.out.print(frame[j]+" ");

System.out.println();}

else

{

for(int j=0;j<size;j++)

System.out.print(frame[j]+" ");

System.out.println();}}}

System.out.println("PAGE FAULT: "+pf);}}

class LRU1

{

public static void main(String arg[])

{

Scanner s=new Scanner(System.in);

System.out.print("Enter frame size:");

int n=s.nextInt();

System.out.print("Enter no of pages:");

int p=s.nextInt();

LRU obj=new LRU(n,p);

obj.get();

obj.lru();

}

}

**Practical 11 banker’s algorithm**

import java.io.\*;

public class Bankers {

static int safe\_sequence[];

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

{

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

System.out.println("Please enter the total number of Resources: ");

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

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

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

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

{

System.out.println("Enter total number of instances for Resource " + (i+1) + ":");

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

cur\_avail[i] = res[i];

}

System.out.println("Enter 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];

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

{

System.out.println("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)));

}

}

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

{

System.out.println("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];

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

{

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

{

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

}

}

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

System.out.println();

if(safe)

{

System.out.println("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.println("The system is not in a Safe State.");

if(safe)

{

System.out.println();

System.out.println("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.println("Please enter the Request Matrix: ");

String ip = br.readLine();

int need\_count = 0, avl\_count = 0;

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

System.out.println();

}

else

System.out.println("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];

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

avail[i] = cur\_avail[i];

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;

}

}

**Practical 12 reader’s writer problem**

class Database

{

private int readers;

public Database()

{

this.readers = 0;

}

public void read(int number)

{

synchronized(this)

{

this.readers++;

System.out.println("Reader " + number + " starts reading.");

}

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

synchronized(this)

{

System.out.println("Reader " + number + " stops reading.");

this.readers--;

if (this.readers == 0)

{

this.notifyAll();

}

}

}

public synchronized void write(int number)

{

while (this.readers != 0)

{

try

{

this.wait();

}

catch (InterruptedException e) {}

}

System.out.println("Writer " + number + " starts writing.");

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

System.out.println("Writer " + number + " stops writing.");

this.notifyAll();

}

}

class Reader extends Thread

{

private static int readers = 0; // number of readers

private int number;

private Database database;

/\*\*

Creates a Reader for the specified database.

@param database database from which to be read.

\*/

public Reader(Database database)

{

this.database = database;

this.number = Reader.readers++;

}

public void run()

{

while (true)

{

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

this.database.read(this.number);

}

}

}

class Writer extends Thread

{

private static int writers = 0; // number of writers

private int number;

private Database database;

/\*\*

Creates a Writer for the specified database.

@param database database to which to write.

\*/

public Writer(Database database)

{

this.database = database;

this.number = Writer.writers++;

}

/\*\*

Writes.

\*/

public void run()

{

while (true)

{

final int DELAY = 5000;

try

{

Thread.sleep((int) (Math.random() \* DELAY));

}

catch (InterruptedException e) {}

this.database.write(this.number);

}

}

}

/\*\*

This app creates a specified number of readers and

writers and starts them.

\*/

public class Simulator

{

/\*\*

Creates the specified number of readers and writers and starts them.

@param args[0] The number of readers.

@param args[1] The number of writers.

\*/

public static void main(String[] args)

{

if (args.length < 2)

{

System.out.println("Usage: java Simulator <number of readers> <number of writers>");

}

else

{

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

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

Database database = new Database();

for (int i = 0; i < READERS; i++)

{

new Reader(database).start();

}

for (int i = 0; i < WRITERS; i++)

{

new Writer(database).start();

}

}

}

}

**Practical 13**

**Demonstrate the concept of synchronized access to shared resource**

class room

{

public synchronized void takesLecture(String name) throws InterruptedException

{

System.out.println(name + " enters...");

Thread.sleep(250);

System.out.println(name + " starts lecture...");

Thread.sleep(500);

System.out.println(name + " exits...");

Thread.sleep(250);

}

}

class lecturer implements Runnable

{

String name;

room r;

Thread t;

lecturer(String n,room r)

{

name = n;

t = new Thread(this, n);

this.r = r;

}

public void start()

{

t.start();

}

public void run()

{

try

{

r.takesLecture(name);

}

catch(InterruptedException e)

{

System.out.println(name + " operation interrupted...");

}

}

}

class os

{

public static void main(String args[])

{

room r = new room();

lecturer kamlakar = new lecturer("Kamlakar Sir",r);

lecturer madhavi = new lecturer("Madhavi Madam",r);

lecturer prajisha = new lecturer("Prajisha Madam",r);

kamlakar.start();

madhavi.start();

prajisha.start();

}

}