1. To simulate FCFS CPU Scheduling Algorithm.

Aim: To Write a java Program to simulate FCFS CPU Scheduling Algorithm.

Description: First Come First Serve (FCFS) is an operating system scheduling algorithm that automatically executes queued requests and processes in order of their arrival. It is the easiest and simplest CPU scheduling algorithm. In this type of algorithm, processes which requests the CPU first get the CPU allocation first. This is managed with a FIFO queue. The full form of FCFS is First Come First Serve.

Procedure:

1. Input the processes along with their burst time (bt).
2. Find waiting time (wt) for all processes.
3. As first process that comes need notto

   wait so waiting time for process 1 will

   be 0 i.e. wt[0] = 0.
4. Find waiting time for all other

   processes i.e. for process i ->

   wt[i] = bt[i-1] + wt[i-1] .
5. Find turnaround time = waiting_time +

   burst_time for all processes.
6. Find average waiting time = total_waiting_time

   / no_of_processes.
7. Similarly, find average turnaround time

   = total_turn_around_time /

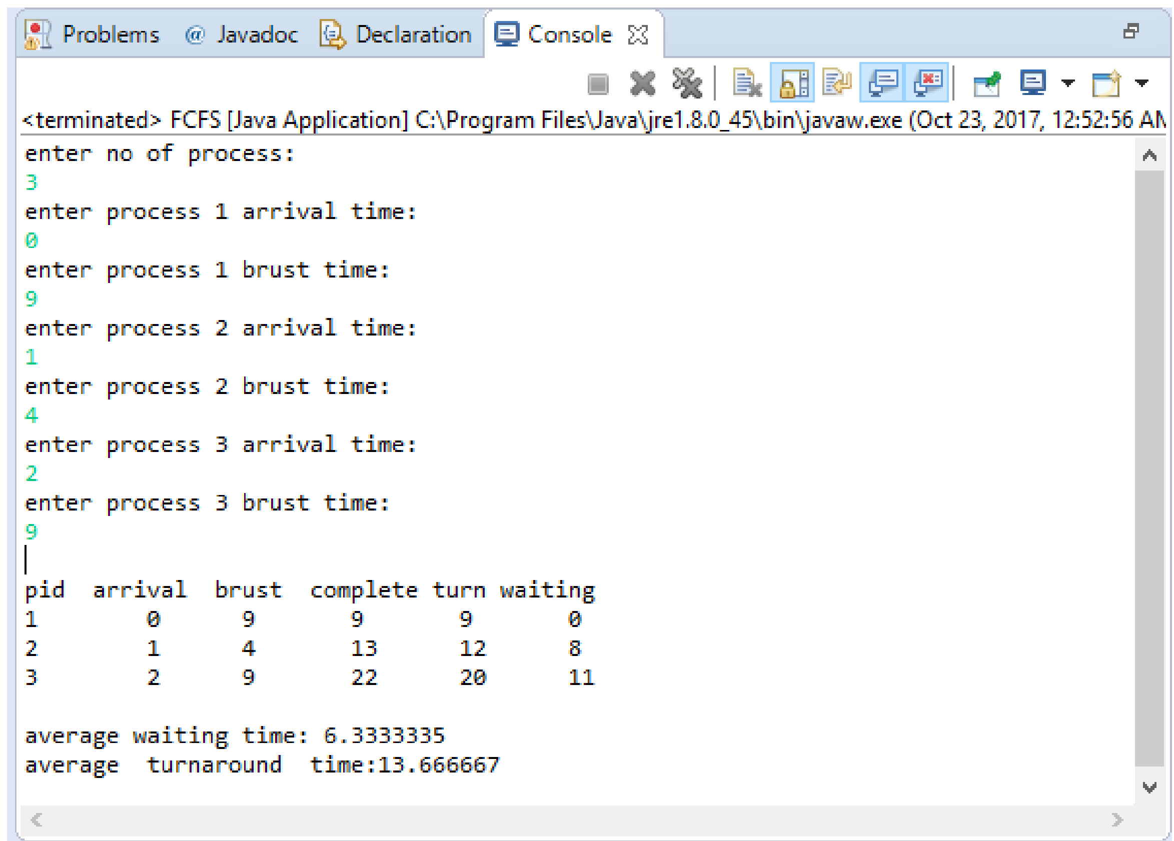
   no_of_processes.

Program:

```
import java.util.*;

public class FCFS {
	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]; // process ids
		int ar[] = new int[n];      // arrival times
		int bt[] = new int[n];      // burst or execution times
		int ct[] = new int[n];      // completion times
		int ta[] = new int[n];      // turn around times
		int wt[] = new int[n];      // waiting times
		int temp;
		float avgwt=0,avgta=0;

		for(int i = 0; i < n; i++)
		{
			System.out.println("enter process " + (i+1) + " arrival time: ");ar[i]
			= sc.nextInt();
			System.out.println("enter  process  "  +  (i+1)  +  "  brust  time:  ");bt[i]
			= sc.nextInt();
			pid[i] = i+1;
		}

		//sorting  according  to  arrival  times
		for(int i = 0 ; i <n; i++)
		{
			for(int  j=0;  j < n-(i+1) ; j++)
			{
				if( ar[j] > ar[j+1] )
				{
					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;
				}
			}
		}
```

```
		// finding completion times
		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] ; // turnaround time= completion time- arrival time

			wt[i] = ta[i]  - bt[i] ;          // waiting time= turnaround time- burst time
			avgwt += wt[i] ;                  // total waiting time
			avgta += ta[i] ;                 // total turnaround time
		}

		System.out.println("\npid  arrival  brust  complete turn 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" + wt[i] ) ;
		}
		sc.close();
		System.out.println("\naverage  waiting  time: "+ (avgwt/n));          // printing
average waiting time.
		System.out.println("average  turnaround  time:"+(avgta/n)); // printing  average
turnaround time.
	}
}
```

OUTPUT:



2. To Simulate Shortest Job First (SJF) CPU SchedulingAlgorithm.

Aim: To write a Java Program To Simulate Shortest Job First (SJF) CPU Scheduling Algorithm.

Description: Shortest job first (SJF) or shortest job next, is a scheduling policy that selects the waiting process with the smallest execution time to execute next. SJN is a non-preemptive algorithm.

Procedure:

1. Traverse until all process gets completely executed.
2. Find process with minimum remaining time at every single time lap.
3. Reduce its time by 1.
4. Check if its remaining time becomes 0
5. Increment the counter of process completion.
6. Completion time of current process = current_time +1;
7. Calculate waiting time for each completed

   process. wt[i]= Completion time -

   arrival_time-burst_time
8. Increment time lap by one.

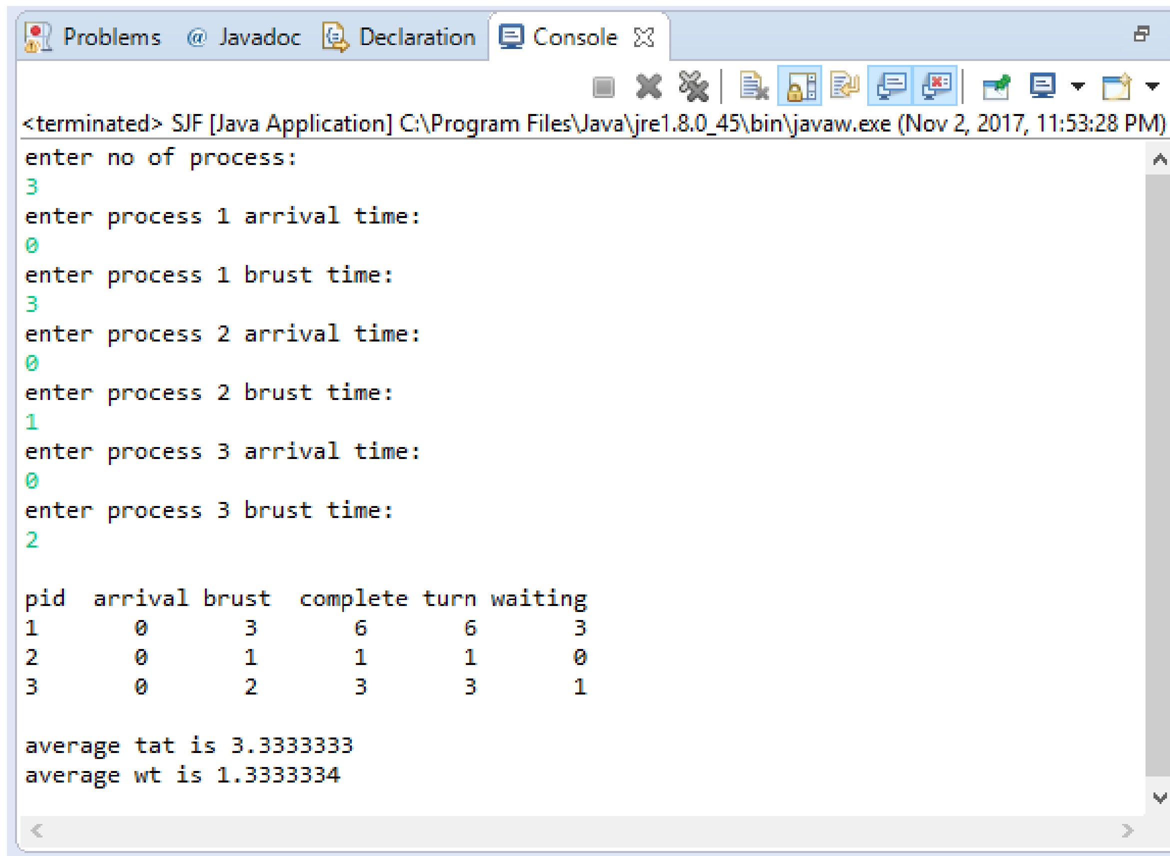
   2- Find turnaround time (waiting_time+burst_time).

Program:

```
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 mea ns 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 bebreak;
terminated
			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 ca n not updated because no process arrivaltime<
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();
        }
}
```

OUTPUT:



3. To simulate ROUND ROBIN (RR) CPU scheduling Algorithm.

Aim: To write a Java program to simulate ROUND ROBIN (RR) CPU scheduling Algorithm.

Description: Round Robin is a CPU scheduling algorithm where each process is assigned a fixed time slot in a cyclic way. It is simple, easy to implement, and starvation-free as all processes get fair share of CPU. One of the most commonly used technique in CPU scheduling as a core.

Procedure:

1. Create an array rem_bt[] to keep track of

   remaining burst time of processes. This

   array is initially a

   copy of bt[] (burst times array)

2. Create another array wt[] to store

   waiting times of processes. Initialize

   this array as 0.

3. Initialize time : t = 0

4. Keep traversing the all processes while all

   processes are not done. Do following for i'th

   process if it is

   not done yet.

   a. If rem_bt[i] > quantum

      (i) t = t + quantum

      (ii)bt_rem[i] -= quantum;

   b. Else // Last cycle for this process

      (i) t = t + bt_rem[i];

      (ii)wt[i] = t - bt[i]

      (iii)bt_rem[i] = 0; // This process is over

Program:

```
public class GFG
{
   // 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 repeatedlyfor
         (int i = 0 ; i < n; i++)
         {
            // If burst time of a process is greater than 0
            // then only need to process furtherif
            (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 processedt +=
                  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 processelse
               {
                  // Increase the value of t i.e. shows
```

```
                // how much time a process has been processedt = 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 doneif
    (done == true)
      break;
  }
}

// Method to calculate turn around time
static void fi ndTurnAroundTime(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);
    }
}
```

Output:

Processes	Burst time	Waiting time	Turn around time
1	10	13	23
2	5	10	15
3	8	13	21

Average waiting time = 12

Average turn around time = 19.6667

4. Priority Scheduling With Different ArrivalTime Priority Scheduling.

Aim: To write a Java Implementation for Priority Scheduling with Different Arrival Time Priority Scheduling.

Description: Priority scheduling is a non-preemptive algorithm and one of the most common scheduling algorithms in batch systems. Each process is assigned first arrival time (less arrival time process first) if two processes have same arrival time, then compare to priorities (highest process first).

Procedure:

1. First input the processes with their arrival time, burst time and priority.
2. First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
3. Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). If two process priority are same then sort according to process number. Note: In the question, They will clearly mention, which number will have higher priority and which number will have lower priority.
4. Once all the processes have been arrived, we can schedule them based on their priority.

Program:

```
import java.util.*;

/// Data Structure
class Process {
    int at, bt, pri, pno;
    Process(int pno, int at, int bt, int pri)
    {
        this.pno = pno;
        this.pri = pri;
        this.at = at;
        this.bt = bt;
    }
}

/// Gantt chart structure
class GChart {
    // process number, start time, complete time,
    // turn around time, waiting time
    int pno, stime, ctime, wtime, ttime;
}

// user define comparative method (first arrival first serve,
// if arrival time same then heigh priority first)
class MyComparator implements Comparator {

    public int compare(Object o1, Object o2)
    {

        Process  p1  =  (Process)o1;
        Process p2 = (Process)o2; if
        (p1.at < p2.at)
            return (-1);

        else if (p1.at == p2.at && p1.pri > p2.pri)
            return (-1);


        else
            return (1);
    }
}



// class to find Gantt chart
class FindGantChart {
    void findGc(LinkedList queue)
    {

        // initial time = 0
```

```
        int time = 0;

        // priority Queue sort data according
        // to arrival time or priority (ready queue)
        TreeSet prique = new TreeSet(new MyComparator());

        // link list for store processes data
        LinkedList result = new LinkedList();

        // process in ready queue from new state queue
        while (queue.size() > 0)
            prique.add((Process)queue.removeFirst()); Iterator it

        = prique.iterator();

        // time set to according to first process
        time = ((Process)prique.first()).at;

        // scheduling process
        while (it.hasNext()) {

            // dispatcher dispatch the
            // process ready to running state
            Process obj = (Process)it.next();

            GChart gc1 = new GChart();
            gc1.pno = obj.pno;
            gc1.stime = time;
            time += obj.bt;
            gc1.ctime = time;
            gc1.ttime = gc1.ctime - obj.at;
            gc1.wtime = gc1.ttime - obj.bt;

            /// store the exxtreted process
            result.add(gc1);
        }

        // create object of output class and call methodnew
        ResultOutput(result);
    }
}
```

OUTPUT:

Process_no	Start_time	Complete_time	Turn_Around_Time	Waiting_Time
1	1	4	3	0
2	5	10	8	3
3	4	5	2	1
4	10	17	13	6
5	17	21	16	12

Average Waiting Time is : 4.4

Average Turn Around time is : 8.4

5.To Simulate page replacement algorithms using FIFO.

Aim: To write a Java Program to Simulate page replacement algorithms usingFIFO.

Description: page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page.

Procedure:

1. Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

b) Simultaneously maintain the pages in the queue to perform FIFO.

c) Increment page fault

ii) Else

If current page is present in set, do nothing.

Else

a) Remove the first page from the queue as it was the first to be entered inthe memory

b) Replace the first page in the queue with the current page in the string.

c) Store current page in the queue.

d) Increment page faults.

2. Return page faults.

Program:

```
import java.util.HashSet;
import java.util.LinkedList;
import java.util.Queue;
class Test
{
    // Method to find page faults using FIFO
    static int pageFaults(int pages[], int n, int capacity)
```

```
// To represent set of current pages. We use
            // an unordered_set so that we quickly check
            // if a page is present in set or not HashSet<Integer>  s
            = new  HashSet<>(capacity);

            // To store the pages in FIFO manner Queue<Integer>
            indexes = new LinkedList<>() ;

            // Start from initial page
            int page_faults = 0; for
            (int i=0; i<n; i++)
            {
                // Check if the set can hold more pagesif
                (s.size() < capacity)
                {
                    // Insert it into set if not present
                    // already which represents page faultif
                    (!s.contains(pages[i]))
                    {
                        s.add(pages[i]);

                        // increment page fault
                        page_faults++;

                        // Push the current page into the queue
                        indexes.add(pages[i]);
                    }
                }

                // If the set is full then need to perform FIFO
                // i.e. remove the first page of the queue from
                // set and queue both and insert the current pageelse
                {
                    // Check if current page is not already
                    // present in the set
                    if (!s.contains(pages[i]))
                    {
                        //Pop the first page from the queueint
                        val = indexes.peek();

                        indexes.poll();

                        // Remove the indexes page
                        s.remove(val);

                        // insert the current page
                        s.add(pages[i]);
```

```
                    // push the current page into
                    // the queue
                    indexes.add(pages[i]);

                    // Increment page faults
                    page_faults++;
                }
            }
        }

        return page_faults;
    }

    // Driver method
    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));
    }
}
// This code is contributed by Gaurav Miglani
```

Output:
7

6.To simulate page replacement algorithms using LRU.

Aim: To write a Java Program to Simulate page replacement algorithms usingLRU.

Description: Page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page.

Procedure:

1. Start traversing the pages.

i) If set holds less pages than capacity.

a) Insert page into the set one by one until the size of set reaches capacity or all page requests are processed.

b) Simultaneously maintain the recent occurred index of each page in a mapcalled indexes.

c) Increment page fault

ii) Else

If current page is present in set, do nothing.

Else

a) Find the page in the set that was least recently used. We find it usingindex array.

We basically need to replace the page with minimum index.

b) Replace the found page with current page.

c) Increment page faults.

d) Update index of current page.

2. Return page faults.

Program:

```
import    java.util.HashMap;
import     java.util.HashSet;
import   java.util.Iterator;
class Test
{
    // Method to find page faults using indexes
    static int pageFaults(int pages[], int n, int capacity)
    {
        // To represent set of current pages. We use
        // an unordered_set so that we quickly check
        // if a page is present in set or not HashSet<Integer>  s
        = new  HashSet<>(capacity);

        // To store least recently used indexes
        // of pages.
        HashMap<Integer,  Integer> indexes = new HashMap<>();

        // Start from initial page
        int page_faults = 0; for
        (int i=0; i<n; i++)
        {
            // Check if the set can hold more pagesif
            (s.size() < capacity)
            {
                // Insert it into set if not present
                // already which represents page faultif
                (!s.contains(pages[i]))
                {
                    s.add(pages[i]);

                    // increment page fault
                    page_faults++;
                }

                // Store the recently used index of
                // each page
                indexes.put(pages[i], i);
            }

            // If the set is full then need to perform lru
            // i.e. remove the least recently used page
            // and insert the current page
            else
            {
                // Check if current page is not already
                // present in the set
                if (!s.contains(pages[i]))
                {
```

```
                    // Find the least recently used pages
                    // that is present in the set
                    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;
                        }
                    }
                    // Remove the indexes page
                    s.remove(val);
                    //remove lru from hashmap
                    indexes.remove(val);
                    // insert the current page
                    s.add(pages[i]);
            // Increment page faults
                    page_faults++;
                }
           // Update the current page index
                indexes.put(pages[i], i);
            }
        }
        return page_faults;
    }
   // Driver method
    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:

6

7. To Demonstrate FCFS Disk Scheduling Algorithm.

Aim: To write a Write a Java Program To Demonstrate FCFS Disk Scheduling Algorithm.

Description: FCFS is the simplest disk scheduling algorithm. As the name suggests, this algorithm entertains requests in the order they arrive in the disk queue. The algorithm looks very fair and there is no starvation (all requests are serviced sequentially) but generally, it does not provide the fastest service.

Procedure:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let us one by one take the tracks in default order and calculate the absolute distance of the track from the head.
3. Increment the total seek count with this distance.
4. Currently serviced track position now becomes the new head position.
5. Go to step 2 until all tracks in request array have not been serviced.

Program:

```
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];

        // calculate absolute distance
        distance = Math.abs(cur_track - head);

        // increase the total count
        seek_count += distance;

        // accessed track is now new headhead =
        cur_track;
    }

    System.out.println("Total number of " +
                    "seek operations = " +
                    seek_count);

    // Seek sequence would be the same
    // as request array sequence
    System.out.println("Seek Sequence is");

    for (int i = 0; i < size; i++)
    {
        System.out.println(arr[i]);
    }
}

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

```
    FCFS(arr, head);
}
}

// This code is contributed by 29AjayKumar
```

Output:

Total number of seek operations = 510

Seek Sequence is

176

79

34

60

92

11

41

114

8.To Demonstrate SCAN Disk Scheduling Algorithm.

Aim: To write a Java Program to Demonstrate SCAN Disk Scheduling Algorithm.

Description: In SCAN disk scheduling algorithm, head starts from one end of the disk and moves towards the other end, servicing requests in between one by one and reach the other end. Then the direction of the head is reversed and the process continues as head continuously scan back and forth to access the disk. So, this algorithm works as an elevator and hence also known as the elevator algorithm.

Procedure:

1. Let Request array represents an array storing indexes of tracks that have been requested in ascending order of their time of arrival. 'head' is the position of disk head.
2. Let direction represents whether the head is moving towards left or right.
3. In the direction in which head is moving service all tracks one by one.
4. Calculate the absolute distance of the track from the head.
5. Increment the total seek count with this distance.
6. Currently serviced track position now becomes the new head position.
7. Go to step 3 until we reach at one of the ends of the disk.
8. If we reach at the end of the disk reverse the direction and go to step 2 until all tracks in request array have not been serviced.

Program:

```
import  java.util.*;class  GFG
{

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>();

    // appending end values
    // which has to be visited
    // before reversing the direction
```

```
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]);
}

// sorting left and right vectors
Collections.sort(left);
Collections.sort(right);

// run the while loop two times.
// one by one scanning right
// and left of the headint
run = 2;
while (run-- >0)
{
    if (direction == "left")
    {
        for (int i = left.size() - 1; i >= 0; i--)
        {
            cur_track = left.get(i);

            // appending current track to seek sequence
            seek_sequence.add(cur_track);

            // calculate absolute distance
            distance = Math.abs(cur_track - head);

            // increase the total count
            seek_count += distance;


            // accessed track is now the new head
            head = cur_track;
        }
        direction = "right";
    }
    else if (direction == "right")
    {
        for (int i = 0; i < right.size(); i++)
        {
            cur_track = right.get(i);
```

```
                // appending current track to seek sequence
                seek_sequence.add(cur_track);

                // calculate absolute distance
                distance = Math.abs(cur_track - head);

                // increase the total count
                seek_count += distance;


                // accessed track is now new head
                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");
    }
}

// Driver code
public static void main(String[] args)
{

    // request array
    int arr[] = { 176, 79, 34, 60,
                92, 11, 41, 114 };
    int head = 50;
    String direction = "left";


    SCAN(arr, head, direction);
}
}
```

Output:

Total number of seek operations = 226
Seek Sequence is
41
34
11
0
60
79
92
114
176

9.To implement HEAD command in Unix/Linux

Aim: To write a java Program to implement HEAD command in Unix/Linux.

Description: The head command is a command-line utility for outputting the first part of files given to it via standard input. It writes results to standard output. By default head returns the first ten lines of each file that it is given.

Procedure:

1. Start

2. To store character data into File

3. TO DO Auto-generated method stub

   FileInputStream fstream = new FileInputStream(args[0]);

   BufferedReaderbr = new BufferedReader(new

InputStreamReader(fstream)); 4.Read File Line By Line

   if(count>10)

5. Print the content on

   the console count++;

6. Close the input

   stream

   fstream.close(

   );

7. Stop

Program:

```
importjava.io.BufferedReader;

importjava.io.FileInputStream;

importjava.io.FileReader; importjava.io.FileWriter;

importjava.io.InputStreamReader;

    // to store character data into File

    public class JHead

    {

            public static void main(String[] args) {

                    // TODO Auto-generatedmethod stub

                    try {

    FileInputStreamfstream = new FileInputStream(args[0]);

    BufferedReaderbr = new BufferedReader(new

            InputStreamReader(fstream)); String strLine;int

    count=1;

            //Read File Line By Line

    while ((strLine = br.readLine()) != null)                   {

             // Print the content on the

            console if(count>10)

                    break;

    System.out.println (strLine);count++;

            }
```

```
        //Close the input stream
fstream.close();


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


        }
}
```

Output

C:\jdk1.8\bin>java JHead testfile.txt

Synchronized Methods

The Java programming language provides two basic synchronization idioms: sTo make a method synchronized, simply add the synchronized keyword to its declaration:

```
public class SynchronizedCounter {
privateint c = 0;
public synchronized void
increment() { c++;
  }
public synchronized void
decrement() { c--;
```

10. To simulate TAIL command in Unix/Linux.

Aim: To write a java program to simulate TAIL command in Unix/Linux

Description: Tail command is used to display the last ten lines of one or more files. Its main purpose is to read the error message. By default, it displays the last ten lines of a file. Additionally, it is used to monitor the file changes in real-time. It is a complementary command of the head command.

Procedure:

4. Start

5. To store character data into File

6. TO DO Auto-generated method
   stub int
   n=Integer.parseInt(args[
   1]);

7. Read File Line By Line.

8. Close the input
   stream
   fstream.close(
   );
9. Stop

Program:

```
importjava.io.BufferedReader;
importjava.io.FileInputStream;
importjava.io.FileReader;
importjava.io.FileWriter;
importjava.io.InputStreamReader
; importjava.util.*;
// to store character data into
File public class JTail {
        public static void main(String[] args) {
                // TODO Auto-generatedmethod
stub int n=Integer.parseInt(args[1]);
                try {

FileInputStreamfstream = new FileInputStream(args[0]);
BufferedReaderbr = new BufferedReader(new
        InputStreamReader(fstream)); String strLine;
Stack<String> lines = new Stack<String>();

        //Read File Line By Line
while ((strLine = br.readLine()) != null) {
lines.push(strLine);
              }
           int
    count=1;

while(! lines.empty()) {
System.out.println(lines.pop());


        if (count>=n)
        break;
        count++;
     }
        //Close the input stream
   fstream.clos
       e();

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

         }
         }
```

Output:

Displays 5 lines from tail of the file(ie.. from end of file);

C:\jdk1.8\bin>java JTail testfile.txt 5

Synchronized methods enable a simple strategy for preventing thread inBut then

other threads can use instances to access the object before construction

instances.add(this);

Warning: When constructing an objecontaining every instance of class.

You might Note that constructors cannot be synchronizedâ?? using the

synchronized

C:\jdk1.8\bin>java JTail testfile.txt 3

Synchronized methods enable a simple strategy for preventing threadinBut then
other threads can use instances to access the object before construction
instances.add(this);

11. Synchronizing tasks.

Aim: To write a java program for Synchronizing tasks.

Description: Synchronization in java is the capability to control the access of multiple threads to any shared resource. Java Synchronization is better option where we want to allow only one thread to access the shared resource.

Procedure:

10. Only one thread can execute at a time.
11. sync_object is a reference to an object
12. whose lock associates with the monitor.
13. The code is said to be synchronized on
14. The monitor object

    synchronized(sync_object)
15. Access shared variables and other
16. shared resources

Program:

```
class Table{
void printTable(int n){//method not synchronizedfor(int
   i=1;i<=5;i++){
     System.out.println(n*i);
     try{
     Thread.sleep(400);
     }catch(Exception e){ System.out.println(e);}
   }

 }
}

class MyThread1 extends Thread{
Table t;
MyThread1(Table t){
this.t=t;
}
public void run(){
t.printTable(5);
}

}
class MyThread2 extends Thread{
Table t;
MyThread2(Table t){
this.t=t;
}
public void run(){
t.printTable(100);
}
}

class TestSynchronization1{
public static void main(String args[]){
Table obj = new Table();//only one object
MyThread1 t1=new MyThread1(obj);
MyThread2 t2=new MyThread2(obj);
t1.start();
t2.start();
}
}
```

Output:

```
100
10
200
15
300
20
400
25
500
```

12.To simulate is cat and ls commands in Linux.

Aim: To Implement a Java program to simulate is cat command in Linux.

Description: The Linux command is a utility of the Linux operating system. All basic and advanced tasks can be done by executing commands. The commands are executed on the Linux terminal. The terminal is a command-line interface to interact with the system, which is similar to the command prompt in the Windows OS. Commands in Linux are case- sensitive**.**

Procedure:

1. Start

2. Next, Provides a stream for a file, supporting both synchronous and asynchronous read and write operations.

```
FileReaderfileReader = new FileReader(args[0]);
```

3. Next, BufferedReader is a class which simplifies reading text from acharacter input stream. It buffers the characters in order to enable efficient reading of text data.

```
BufferedReader in = new BufferedReader(fileReader);
```

4. The try statement allows you to define a block of code to be tested for errors while it is being executed.

5. The catch statement allows you to define a block of code to be executed,if an error occurs in the try block.

6. Finally it prints the input and output cat file

7. Stop

Program:

12. (a) CAT command

```
importjava.io.BufferedReader;

importjava.io.FileNotFoundExcepti on;

importjava.io.FileReader;

importjava.io.IOException;


public class Jcat {

public static void main(String[] args) {

if(args.length==1){

try {

FileReaderfileReader  =  new FileReader(args[0]);

BufferedReader in = new

BufferedReader(fileReader);

            String line;

while((line = in.readLine())!= null){

System.out.println(line);

            }

         } catch (FileNotFoundException ex) {

System.out.println(args[0]+",  file  not  found.");

         }

catch (IOException ex) {

System.out.println(args[0]+",  input/outputerror.");

         }

      }

   }
```

```
}
```

Output:

Input: file.txt

Java Reader is an abstract class for reading character streams.

The java command-line argument is an argument i.e. passed at the time ofrunning the java program.

The arguments passed from the console can be received in the java programand it can be used as an input.

C:\jdk1.8\bin>java Jcat file.txt

Java Reader is an abstract class for reading character streams.

The java command-line argument is an argument i.e. passed at the time ofrunning the java program.

The arguments passed from the console can be received in the java programand it can be used as an input.

(b) LS command

```
importjava.io.File;

public class Jls {

public static void main(String[] args) {File

      dir = new

      File(System.getProperty("user.dir"));       String

      childs[] = dir.list();

for(String    child:    childs){

System.out.println(child);

      }

   }

}
```

Output:-

C:\jdk1.8\bin>javaJls

appexample.html

applet.html

appletfile.html

appletviewer.exe

AxisBank

CommandLineExample.class

CommandLineExample

.java data DReceiver.class