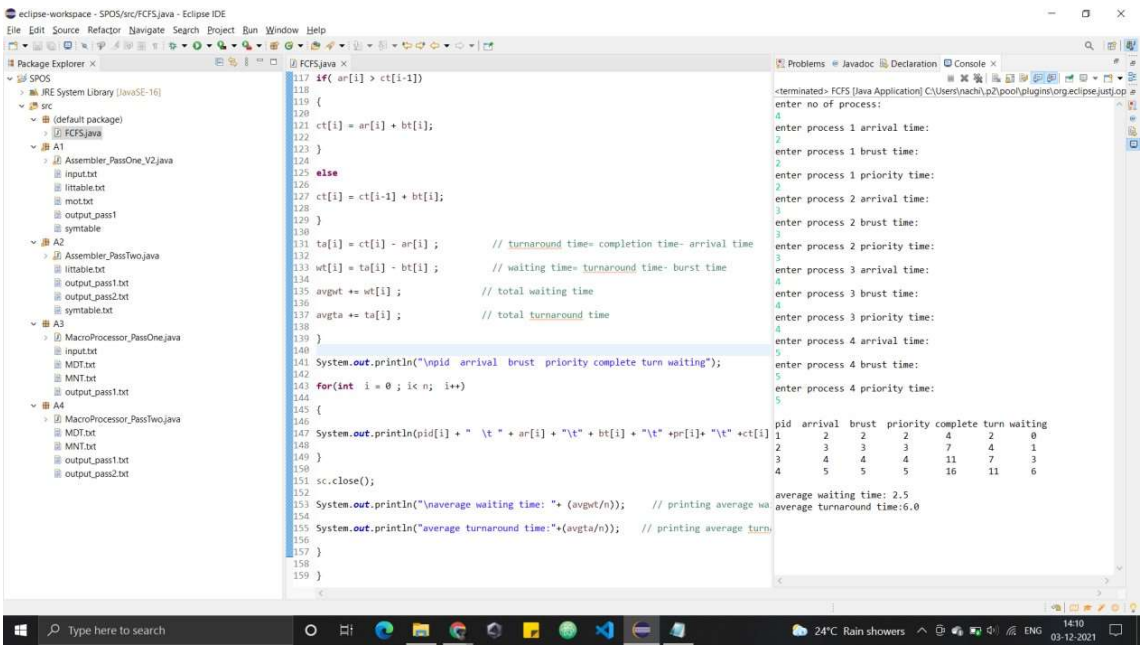
Source code:
A.FCFS:

```
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];
int bt[] = new int[n];
int ct[] = new int[n];
int ta[] = new int[n];
int wt[] = new int[n];
int pr[]= new int[n];
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();
System.out.println("enter process " + (i+1) + " priority time: ");
pr[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  priority complete turn waiting");
for(int i = 0 ; i< n;  i++)
{
System.out.println(pid[i] + "  \t " + ar[i] + "\t" + bt[i] + "\t" +pr[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:

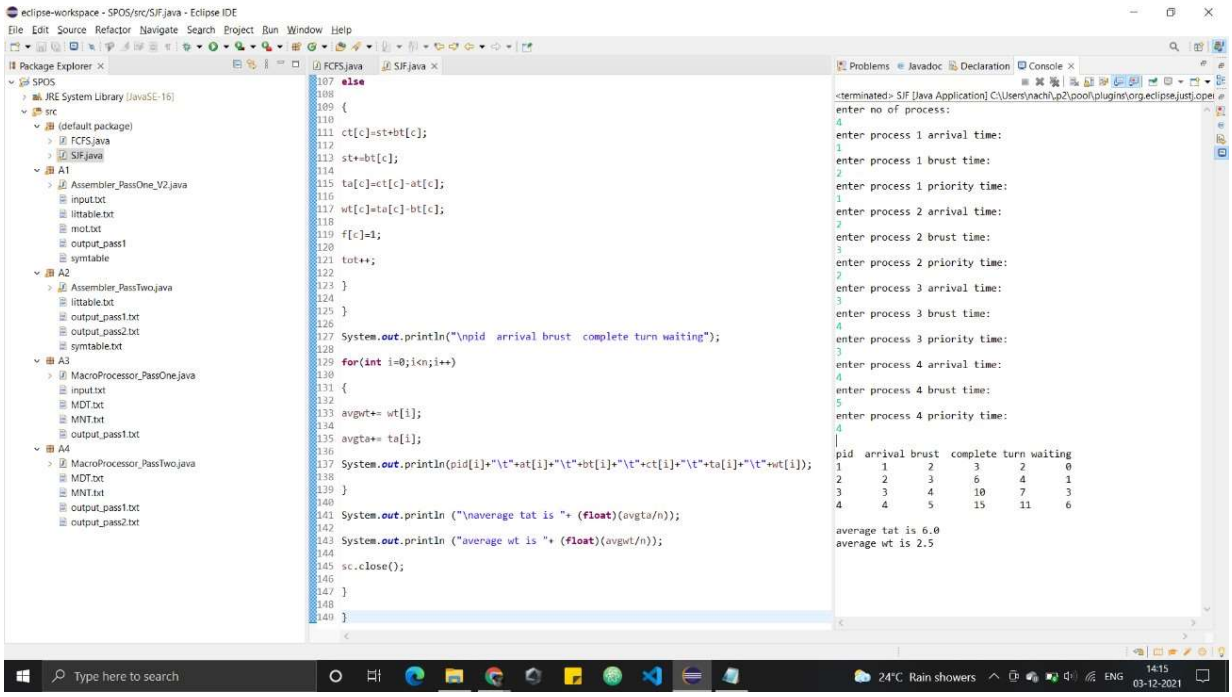


B.SJF:

```
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];
int bt[] = new int[n];
int ct[] = new int[n];
int ta[] = new int[n];
int wt[] = new int[n];
int pr[]= new int[n];
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();
System.out.println("enter process " + (i+1) + " priority time: ");
pr[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();
}
}
```

OUTPUT:



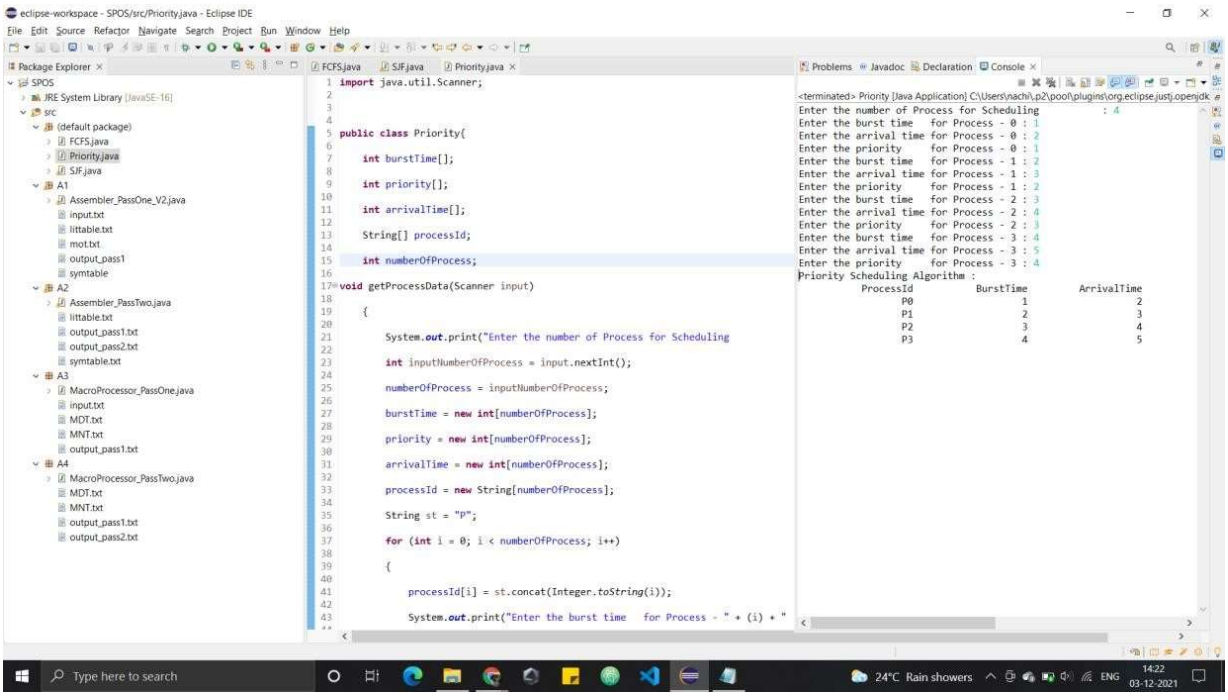
C. Priority:

```
import java.util.Scanner;
public class Priority{
int burstTime[];
int priority[];
int arrivalTime[];
String[] processId;
int numberOfProcess;
void getProcessData(Scanner input)
{
System.out.print("Enter the number of Process for Scheduling           : ");
int inputNumberOfProcess = input.nextInt();
numberOfProcess = inputNumberOfProcess;
burstTime = new int[numberOfProcess];
priority = new int[numberOfProcess];
arrivalTime = new int[numberOfProcess];
processId = new String[numberOfProcess];
String st = "P";
for (int i = 0; i < numberOfProcess; i++)
{
processId[i] = st.concat(Integer.toString(i));
System.out.print("Enter the burst time    for Process - " + (i) + " : ");
burstTime[i] = input.nextInt();
System.out.print("Enter the arrival time for Process - " + (i) + " : ");
arrivalTime[i] = input.nextInt();
System.out.print("Enter the priority     for Process - " + (i) + " : ");
priority[i] = input.nextInt();
}
}
void sortAccordingArrivalTimeAndPriority(int[] at, int[] bt, int[] prt, String[] pid)
{
int temp;
String stemp;
for (int i = 0; i < numberOfProcess; i++)
{
for (int j = 0; j < numberOfProcess - i - 1; j++)
{
if (at[j] > at[j + 1])
{
//swapping arrival time
temp = at[j];
at[j] = at[j + 1];
at[j + 1] = temp;
//swapping burst time
temp = bt[j];
bt[j] = bt[j + 1];
bt[j + 1] = temp;
//swapping priority
temp = prt[j];
prt[j] = prt[j + 1];
prt[j + 1] = temp;
//swapping process identity
stemp = pid[j];
pid[j] = pid[j + 1];
```

```
pid[j + 1] = stemp;
}
//sorting according to priority when arrival timings are same
if (at[j] == at[j + 1])
{
if (prt[j] > prt[j + 1])
{
//swapping arrival time
temp = at[j];
at[j] = at[j + 1];
at[j + 1] = temp;
//swapping burst time
temp = bt[j];
bt[j] = bt[j + 1];
bt[j + 1] = temp;
//swapping priority
temp = prt[j];
prt[j] = prt[j + 1];
prt[j + 1] = temp;
//swapping process identity
stemp = pid[j];
pid[j] = pid[j + 1];
pid[j + 1] = stemp;
}
}
}
}
}
void priorityNonPreemptiveAlgorithm()
{
int finishTime[] = new int[numberOfProcess];
int bt[] = burstTime.clone();
int at[] = arrivalTime.clone();
int prt[] = priority.clone();
String pid[] = processId.clone();
int waitingTime[] = new int[numberOfProcess];
int turnAroundTime[] = new int[numberOfProcess];
sortAccordingArrivalTimeAndPriority(at, bt, prt, pid);
//calculating waiting & turn-around time for each process
finishTime[0] = at[0] + bt[0];
turnAroundTime[0] = finishTime[0] - at[0];
waitingTime[0] = turnAroundTime[0] - bt[0];
for (int i = 1; i < numberOfProcess; i++)
{
finishTime[i] = bt[i] + finishTime[i - 1];
turnAroundTime[i] = finishTime[i] - at[i];
waitingTime[i] = turnAroundTime[i] - bt[i];
}
float sum = 0;
for (int n : waitingTime)
{
sum += n;
}
float averageWaitingTime = sum / numberOfProcess;
sum = 0;
```

```
for (int n : turnAroundTime)
{
sum += n;
}
float averageTurnAroundTime = sum / numberOfProcess;
//print on console the order of processes along with their finish time & turn around time
System.out.println("Priority Scheduling Algorithm : ");
System.out.format("%20s%20s%20s%20s%20s%20s%20s\n", "ProcessId", "BurstTime",
"ArrivalTime", "Priority", "FinishTime", "WaitingTime", "TurnAroundTime");
for (int i = 0; i < numberOfProcess; i++) {
System.out.format("%20s%20d%20d%20d%20d%20d%20d\n", pid[i], bt[i], at[i],
prt[i], finishTime[i], waitingTime[i], turnAroundTime[i]);
}
System.out.format("%100s%20f%20f\n", "Average", averageWaitingTime,
averageTurnAroundTime);
}
public static void main(String[] args)
{
Scanner input = new Scanner(System.in);
Priority obj = new Priority();
obj.getProcessData(input);
obj.priorityNonPreemptiveAlgorithm();
}
}
```

OUTPUT:

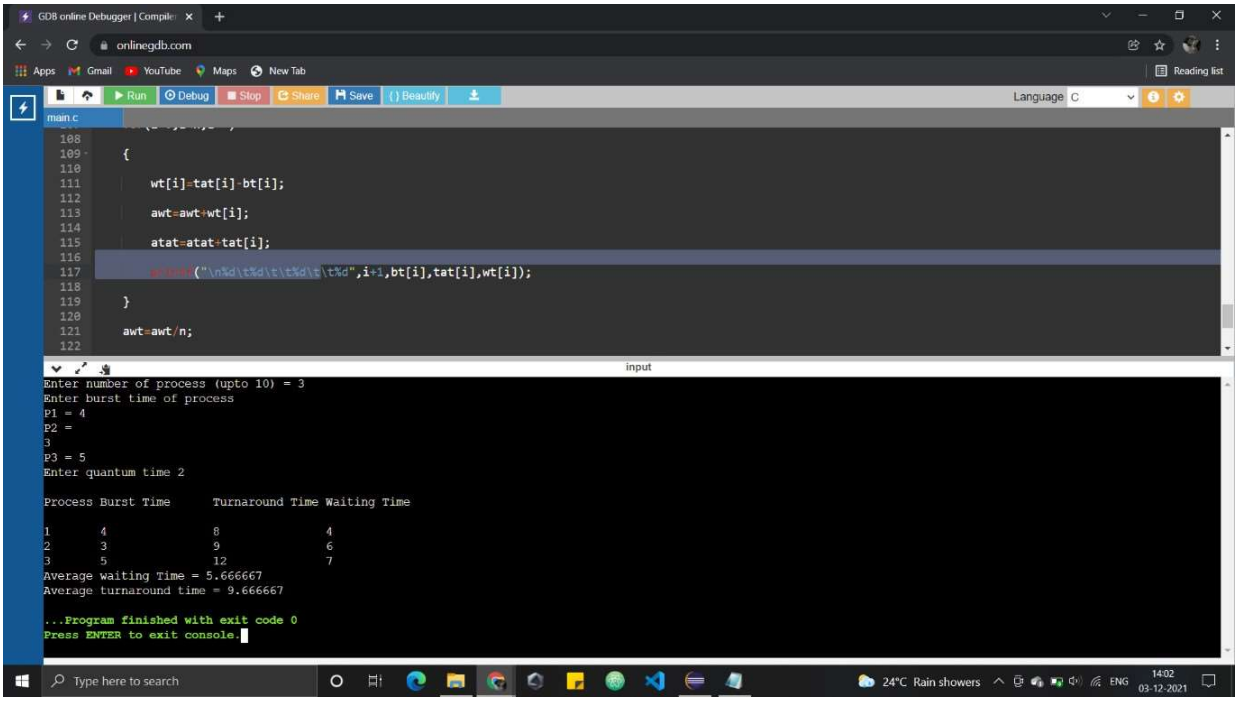


D.Round Robin:

```
#include<stdio.h>
#include<conio.h>
int main()
{
int n,i,qt,count=0,temp,sq=0,bt[10],wt[10],tat[10],rem_bt[10];
//n signifies number of process
//i is for using loops
//qt denotes Quantum Time
//count denotes when one process is completed
//temp and sq are temproray variables
//bt[10] denotes burst time
//wt[10] denotes waiting time
//tat[10] denotes turnaround time
//rem_bt[10] denotes remaining burst time
float awt=0,atat=0;
//awt represents average waiting time
//atat represents average turnaround time
printf("Enter number of process (upto 10) = ");
scanf("%d",&n);
printf("Enter burst time of process\n");
for (i=0;i<n;i++)
{
printf("P%d = ",i+1);
scanf("%d",&bt[i]);
rem_bt[i]=bt[i];
}
printf("Enter quantum time ");
scanf("%d",&qt);
while(1)
{
for (i=0,count=0;i<n;i++)
{
temp=qt;
if(rem_bt[i]==0)
{
count++;
continue;
}
if(rem_bt[i]>qt)//changing the value of remaining burst time
rem_bt[i]=rem_bt[i]-qt;
else
if(rem_bt[i]>=0)//if process is exhausted then setting remaining burst time
{
temp=rem_bt[i];
rem_bt[i]=0;
}
sq=sq+temp; //calculating turnaround time
tat[i]=sq;
}
if(n==count)//breaking the loop when all process are exhausted
break;
}
printf("\nProcess\tBurst Time\tTurnaround Time\tWaiting Time\n");
```

```
for(i=0;i<n;i++)
{
wt[i]=tat[i]-bt[i];
awt=awt+wt[i];
atat=atat+tat[i];
printf("\n%d\t%d\t\t%d\t\t%d",i+1,bt[i],tat[i],wt[i]);
}
awt=awt/n;
atat=atat/n;
printf("\nAverage waiting Time = %f\n",awt);
printf("Average turnaround time = %f",atat);
return 0;
}
```

OUTPUT:



Conclusion:

CPU policies implemented successfully.

Input:
• No. of jobs (js) & No. of blocks (bs)
• Job size of all jobs & Block size of all blocks
For Example:
js=4
bs=5
block[] = {100, 500, 200, 300, 600};
jobs[] = {212, 417, 112, 426};

Code:

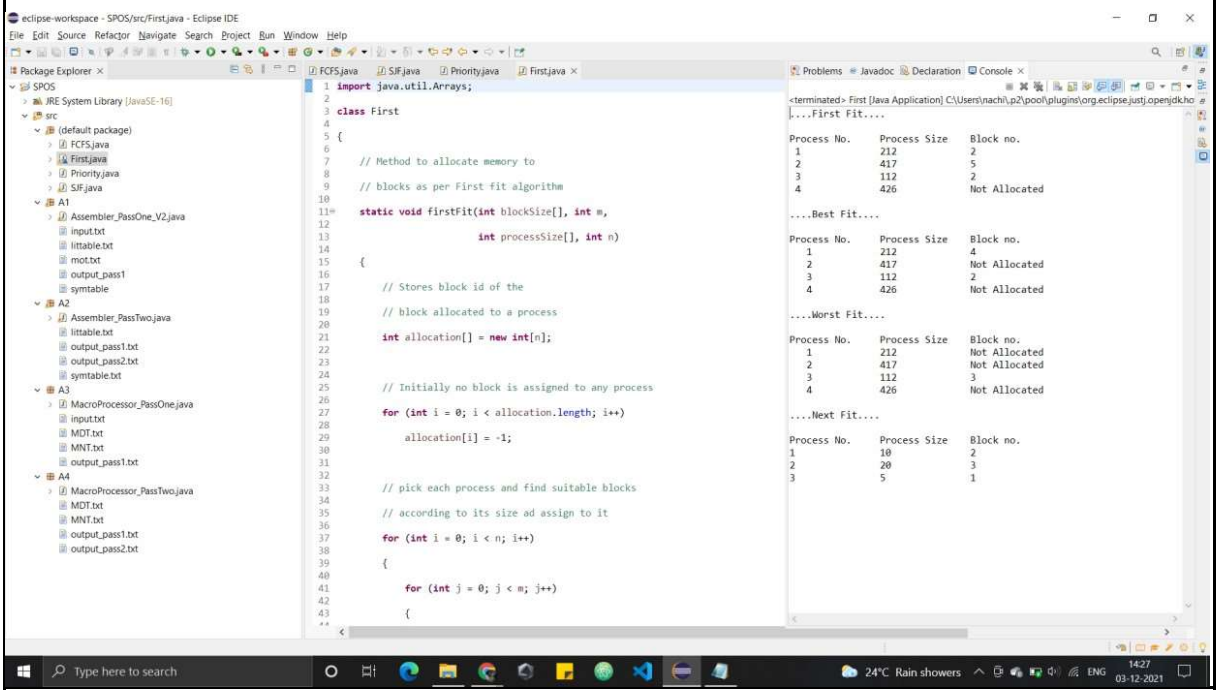
```
import java.util.Arrays;
class First
{
// Method to allocate memory to
// blocks as per First fit algorithm
static void firstFit(int blockSize[], int m,
int processSize[], int n)
{
// Stores block id of the
// block allocated to a process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i = 0; i < n; i++)
{
for (int j = 0; j < m; j++)
{
if (blockSize[j] >= processSize[i])
{
// allocate block j to p[i] process
allocation[i] = j;
// Reduce available memory in this block.
blockSize[j] -= processSize[i];
break;
}
}}
System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
System.out.print(" " + (i+1) + "\t\t" +
processSize[i] + "\t\t");
if (allocation[i] != -1)
System.out.print(allocation[i] + 1);
else
System.out.print("Not Allocated");
System.out.println();
}
}
```

```
static void bestFit(int blockSize[], int m, int processSize[],
int n)
{
// Stores block id of the block allocated to a
// process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i=0; i<n; i++)
{
// Find the best fit block for current process
int bestIdx = -1;
for (int j=0; j<m; j++)
{
if (blockSize[j] >= processSize[i])
{
if (bestIdx == -1)
bestIdx = j;
else if (blockSize[bestIdx] > blockSize[j])
bestIdx = j;
}
}
// If we could find a block for current process
if (bestIdx != -1)
{
// allocate block j to p[i] process
allocation[i] = bestIdx;
// Reduce available memory in this block.
blockSize[bestIdx] -= processSize[i];
}
}
System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
System.out.print("   " + (i+1) + "\t\t" + processSize[i] + "\t\t");
if (allocation[i] != -1)
System.out.print(allocation[i] + 1);
else
System.out.print("Not Allocated");
System.out.println();
}
}
static void worstFit(int blockSize[], int m, int processSize[],
int n)
{
// Stores block id of the block allocated to a
// process
int allocation[] = new int[n];
// Initially no block is assigned to any process
for (int i = 0; i < allocation.length; i++)
allocation[i] = -1;
// pick each process and find suitable blocks
```

```
// according to its size ad assign to it
for (int i=0; i<n; i++)
{
// Find the best fit block for current process
int wstIdx = -1;
for (int j=0; j<m; j++)
{
if (blockSize[j] >= processSize[i])
{
if (wstIdx == -1)
wstIdx = j;
else if (blockSize[wstIdx] < blockSize[j])
wstIdx = j;
}
}
// If we could find a block for current process
if (wstIdx != -1)
{
// allocate block j to p[i] process
allocation[i] = wstIdx;
// Reduce available memory in this block.
blockSize[wstIdx] -= processSize[i];
}
}
System.out.println("\nProcess No.\tProcess Size\tBlock no.");
for (int i = 0; i < n; i++)
{
System.out.print("   " + (i+1) + "\t\t" + processSize[i] + "\t\t");
if (allocation[i] != -1)
System.out.print(allocation[i] + 1);
else
System.out.print("Not Allocated");
System.out.println();
}
}
static void NextFit(int blockSize1[], int m1, int processSize1[], int n1) {
// Stores block id of the block allocated to a
// process
int allocation[] = new int[n1], j = 0;
// Initially no block is assigned to any process
Arrays.fill(allocation, -1);
// pick each process and find suitable blocks
// according to its size ad assign to it
for (int i = 0; i < n1; i++) {
// Do not start from beginning
int count =0;
while (j < m1) {
count++;    //makes sure that for every process we traverse through entire
array maximum once only.This avoids the problem of going into infinite loop if memory is
not available
if (blockSize1[j] >= processSize1[i]) {
// allocate block j to p[i] process
allocation[i] = j;
// Reduce available memory in this block.
blockSize1[j] -= processSize1[i];
```

```
break;
}
// mod m will help in traversing the blocks from
// starting block after we reach the end.
j = (j + 1) % m1;
}
}
System.out.print("\nProcess No.\tProcess Size\tBlock no.\n");
for (int i = 0; i < n1; i++) {
System.out.print( i + 1 + "\t\t" + processSize1[i]
+ "\t\t");
if (allocation[i] != -1) {
System.out.print(allocation[i] + 1);
} else {
System.out.print("Not Allocated");
}
System.out.println("");
}
}
// Driver Code
public static void main(String[] args)
{
System.out.println("....First Fit....");
int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m = blockSize.length;
int n = processSize.length;
firstFit(blockSize, m, processSize, n);
/* int blockSize[] = {100, 500, 200, 300, 600};
int processSize[] = {212, 417, 112, 426};
int m = blockSize.length;
int n = processSize.length;*/
System.out.println(" ");
System.out.println("....Best Fit....");
bestFit(blockSize, m, processSize, n);
System.out.println(" ");
System.out.println("....Worst Fit....");
worstFit(blockSize, m, processSize, n);
System.out.println(" ");
System.out.println("....Next Fit....");
int blockSize1[] = {5, 10, 20};
int processSize1[] = {10, 20, 5};
int m1 = blockSize1.length;
int n1 = processSize1.length;
NextFit(blockSize1, m1, processSize1, n1);
}
}
```

Output:



Conclusion:

successfully implemented simulation of memory placement strategies.

```
****LRU****
*/
import java.io.*;
    class lru
     {
     public static void main(String args[])throws IOException
      {
                  BufferedReader obj=new BufferedReader(new InputStreamReader(System.in));
                  int f,page=0,ch,pgf=0,n,chn=0;
                  boolean flag;
                  int pages[];                //pgf-page fault

                System.out.println("1.LRU");
                int pt=0;
          System.out.println("enter no. of frames: ");
                 f=Integer.parseInt(obj.readLine());
                int frame[]=new int[f];


                for(int i=0;i<f;i++)
                {
                        frame[i]=-1;
                }

                System.out.println("enter the no of pages ");
                n=Integer.parseInt(obj.readLine());

              pages=new int[n];
                System.out.println("enter the page no ");

                for(int j=0;j<n;j++)
                pages[j]=Integer.parseInt(obj.readLine());

                int pg=0;
                for(pg=0;pg<n;pg++)
          {
                        page=pages[pg];
                        flag=true;
                        for(int j=0;j<f;j++)
                        {
                                if(page==frame[j])
                                {
                                        flag=false;
                                        break;
                                 }
                        }
                        int temp,h=3,i;
                        if(flag)
                {
                        if( frame[1]!=-1 && frame[2]!=-1 && frame[0]!=-1)
                                {
                                        temp=pages[pg-3];
                                        if(temp==pages[pg-2] || temp==pages[pg-1])
                                                temp=pages[pg-4];
```

```
                                        for(i=0;i<f;i++)
                                                if(temp==frame[i])
                                                        break;
                                        frame[i]=pages[pg];
                                }
                                else
                                {
                                        if(frame[0]==-1)
                                                frame[0]=pages[pg];
                                        else if(frame[1]==-1)
                                                frame[1]=pages[pg];
                                        else if(frame[2]==-1)
                                                frame[2]=pages[pg];
                                }


                                        System.out.print("frame :");
                                        for(int j=0;j<f;j++)
                                        System.out.print(frame[j]+"   ");
                                        System.out.println();
                                        pgf++;
                        }
                        else
                        {
                                System.out.print("frame :");
                                for(int j=0;j<f;j++)
                                System.out.print(frame[j]+"  ");
                                System.out.println();
                        }

                }//for

        System.out.println("Page fault:"+pgf);

}//main
}//class



****Optimal****
*/
import java.util.*;
import java.io.*;
class Optimal
{
        public static void main(String args[])throws IOException
        {
                BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
                int numberOfFrames, numberOfPages, flag1, flag2, flag3, i, j, k, pos = 0, max;
                int faults = 0;
                int temp[] = new int[10];

                System.out.println("Enter number of Frames: ");
                numberOfFrames = Integer.parseInt(br.readLine());
                int frame[] = new int[numberOfFrames];
```

```
    System.out.println("Enter number of Pages: ");
    numberOfPages = Integer.parseInt(br.readLine());

    int pages[] = new int[numberOfPages];
    System.out.println("Enter the pages: ");
    for(i=0; i<numberOfPages; i++)
             pages[i] = Integer.parseInt(br.readLine());

    for(i = 0; i < numberOfFrames; i++)
frame[i] = -1;


     for(i = 0; i < numberOfPages; ++i){
          flag1 = flag2 = 0;

          for(j = 0; j < numberOfFrames; ++j){
             if(frame[j] == pages[i]){
                   flag1 = flag2 = 1;
                   break;
               }
          }

          if(flag1 == 0){
             for(j = 0; j < numberOfFrames; ++j){
                if(frame[j] == -1){
                   faults++;
                   frame[j] = pages[i];
                   flag2 = 1;
                   break;
                }
             }
          }

          if(flag2 == 0){
             flag3 =0;

             for(j = 0; j < numberOfFrames; ++j){
                temp[j] = -1;

                for(k = i + 1; k < numberOfPages; ++k){
                   if(frame[j] == pages[k]){
                      temp[j] = k;
                      break;
                   }
                }
             }

             for(j = 0; j < numberOfFrames; ++j){
                if(temp[j] == -1){
                   pos = j;
                   flag3 = 1;
                   break;
                }
```

```
                    }

                    if(flag3 ==0){
                        max = temp[0];
                        pos = 0;

                        for(j = 1; j < numberOfFrames; ++j){
                            if(temp[j] > max){
                                max = temp[j];
                                pos = j;
                            }
                        }
                    }

                    frame[pos] = pages[i];
                    faults++;
                }

//                  System.out.print();

                    for(j = 0; j < numberOfFrames; ++j){
                        System.out.print("\t"+ frame[j]);
                    }
                }

                System.out.println("\n\nTotal Page Faults: "+ faults);

        }
}
```

Optimal Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

LRU Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6