**CODE**:

package os;

import java.lang.\*;

import java.util.Scanner;

class Memory

{

int start, end, size;

boolean empty;

public Memory(int start, int end, boolean empty) {

this.start = start;

this.end = end;

this.size = end - start;

this.empty = empty;

}

}

public class MemoryMgmt {

public static void main(String args[])

{

int memorySize, no, start, end,nStart, nEnd, size,i,j;

Scanner sc = new Scanner(System.in);

System.out.println("Enter memory size: ");

memorySize = sc.nextInt();

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

no = sc.nextInt();

Memory slice[] = new Memory[1000];

j = 0; start = 0; end = 0;

System.out.println("Enter start and end of processes: ");

for(i = 0; i < 1000; i++)

{

if(j != no){

System.out.print("Process "+j+": ");

j++;

nStart = sc.nextInt();

nEnd = sc.nextInt();

if(end == nStart)

{

slice[i] = new Memory(nStart, nEnd, false);

end = nEnd;

}

else

{

slice[i] = new Memory(end, nStart, true);

slice[++i] = new Memory(nStart, nEnd, false);

end = nEnd;

}

}

else

break;

}

int totalSlices = i;

if(end != memorySize)

slice[totalSlices++] = new Memory(end, memorySize, true);

display(slice, totalSlices);

System.out.println("Enter new process and its size ");

size = sc.nextInt();

System.out.println("Enter your choice: \n1.BEST FIT \n2.WORST FIT\n3.FIRST FIT");

int op = sc.nextInt();

switch(op)

{

case 1: //BEST FIT;

bestFit(slice, totalSlices,size);break;

case 2: //WORST FIT;

worstFit(slice, totalSlices, size);break;

case 3: //FIRST FIT;

firstFit(slice,totalSlices, size);break;

}

}

private static void display(Memory[] slice, int totalSlices) {

int i;

System.out.println("ID\tSTART\tEND\tTYPE");

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

System.out.println(i+"\t"+slice[i].start+"\t"+slice[i].end+"\t"+(slice[i].empty?"Free":"Occupied"));

}

private static void sort(Memory[] slice, int totalSlices){

int i, j;

Memory temp;

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

{

for(j = 0; j < totalSlices-i-1; j++)

{

if(slice[j].start > slice[j+1].start)

{

temp = slice[j];

slice[j] = slice[j+1];

slice[j+1] = temp;

}

}

}

display(slice, totalSlices);

}

private static void bestFit(Memory[] slice, int totalSlices, int size) {

int i; int diff = 9999; int pos = -1;

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

{

if(slice[i].empty && slice[i].size >= size && ((slice[i].size - size) <= diff))

{

pos = i;

diff = slice[i].size - size;

}

}

if(pos != -1){

slice[pos].size = size;

slice[pos].end = slice[pos].start + size;

slice[pos].empty = false;

if(diff != 0)

slice[totalSlices++] = new Memory(slice[pos].end,slice[pos+1].start, true);

}

else

System.out.println("Cannot fit");

sort(slice, totalSlices);

}

private static void worstFit(Memory[] slice, int totalSlices, int size) {

int i; int diff = -1; int pos = -1;

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

{

if(slice[i].empty && (slice[i].size >= size) && ((slice[i].size - size) >= diff))

{

pos = i;

diff = slice[i].size - size;

}

}

if(pos != -1){

slice[pos].size = size;

slice[pos].end = slice[pos].start + size;

slice[pos].empty = false;

if(diff != 0)

slice[totalSlices++] = new Memory(slice[pos].end,slice[pos+1].start, true);

}

else

System.out.println("Cannot fit");

sort(slice, totalSlices);

}

private static void firstFit(Memory[] slice, int totalSlices, int size) {

int i; int pos = -1, diff = -1;

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

{

if(slice[i].empty && (slice[i].size >= size))

{

pos = i;

diff = slice[i].size - size;

break;

}

}

if(pos != -1){

slice[pos].size = size;

slice[pos].end = slice[pos].start + size;

slice[pos].empty = false;

if(diff != 0)

slice[totalSlices++] = new Memory(slice[pos].end,slice[pos+1].start, true);

}

else

System.out.println("Cannot fit");

sort(slice, totalSlices);

}

}

**OUTPUT**:

***1.BEST FIT***

Enter memory size:

1000

Enter number of processes:

3

Enter start and end of processes:

Process 0: 100 200

Process 1: 400 600 900 1000

Process 2: ID START END TYPE

0 0 100 Free

1 100 200 Occupied

2 200 400 Free

3 400 600 Occupied

4 600 900 Free

5 900 1000 Occupied

Enter **new** process and its size

150

Enter your choice:

1.BEST FIT

2.WORST FIT

3.FIRST FIT

1

ID START END TYPE

0 0 100 Free

1 100 200 Occupied

2 200 350 Occupied

3 350 400 Free

4 400 600 Occupied

5 600 900 Free

6 900 1000 Occupied

***2. WORST FIT***

Enter memory size:

1000

Enter number of processes:

3

Enter start and end of processes:

Process 0: 100 200

Process 1: 400 600

Process 2: 900 1000

ID START END TYPE

0 0 100 Free

1 100 200 Occupied

2 200 400 Free

3 400 600 Occupied

4 600 900 Free

5 900 1000 Occupied

Enter **new** process and its size

150

Enter your choice:

1.BEST FIT

2.WORST FIT

3.FIRST FIT

2

ID START END TYPE

0 0 100 Free

1 100 200 Occupied

2 200 400 Free

3 400 600 Occupied

4 600 750 Occupied

5 750 900 Free

6 900 1000 Occupied

***3. FIRST FIT***

Enter memory size:

1000

Enter number of processes:

3

Enter start and end of processes:

Process 0: 100 200

Process 1: 400 600

Process 2: 900 1000

ID START END TYPE

0 0 100 Free

1 100 200 Occupied

2 200 400 Free

3 400 600 Occupied

4 600 900 Free

5 900 1000 Occupied

Enter **new** process and its size

175

Enter your choice:

1.BEST FIT

2.WORST FIT

3.FIRST FIT

3

ID START END TYPE

0 0 100 Free

1 100 200 Occupied

2 200 375 Occupied

3 375 400 Free

4 400 600 Occupied

5 600 900 Free

6 900 1000 Occupied