Page No.
Date: / /

SPOSL C1

Title : Scheduling Algorithms

Problem Statement:
Write a Java program (using OOP) to implement following Scheduling Algorithms : FCFS, SJF (Premeptive), Priority (Non-preemptive) & Round Robin (premeptive).

Objective:
To implement scheduling Algorithms

Outcomes:
Students will be able to implement OS scheduling algorithms for CPU scheduling.

Theory:
Scheduler:
An operating system program that selects the job to ~~do~~be admitted for execution.
The performance criteria are:
1) Throughput
2) CPU utillzation
3) Turnaround Time
4) Response Time.
5) Priority
6) Avg. TA
7) Avg. waiting time
8) Avg. Response time.

Scheduling Policy:
It determines when it is time for process to be removed from CPU & which should be allocated next.

Page No.
Date: / /

CPU scheduling is divided into 2 types:

1) Premptive:
It integrates the processing of a process & transfers the CPU to another process.

2) Non-premptive:
Uninterrupted completion of process execution takes place.

Types of Scheduling Algorithms

1) FCFS
First Come First Serve
- non-premptive scheduler

Process	P1	P2	P3
Arrival Time	0	1	2

execution: P1 → P2 → P3

2) SJF
Shortest Job First.
- Premptive / Non-Premtive
- Smaller burst times

Process	P1	P2	P3
Burst Time	6	4	2

execution: P3 → P2 → P1

3) Round Robin Premptive
- CPU time divided into quantum time.

Process	P1	P2	P3
Burst time	6	4	2

time quantum = 2 units
execution: P1 → P2 → P3 → P1 → P2 → P1

4) Priority Scheduling
- Premptive / Non-Premptive
- Process with higher priority executed.
- Processes with same priority: RR / FCFS is used.

Process	P1	P2	P3
BT	3	4	2
Priority	2	1	1

execution: P2 → P3 → P1

Algorithms:
1) FCFS:
1) I/P processes with Burst Time.
2) I/P arrival times
3) Sort according to arrival time
4) Perform processes in that order
5) end.

2) SJF
1) Calc. burst time.
2) Sort burst time in ascending order
3) Apply FCFS, perform processes, stop.

3) Round Robin

1) Get i/p for processes with arrival & burst time.
2) Sort according to arrival time
3) Process till all are done
4) end.

4) Priority

1) Get input with arrival time, burst time & priority
2) Sort according to arrival time.
3) If arrival is same sort by priority
4) end.

Test Cases:

1) FCFS

Process	Input		Output			
	BT	AT	BT	AT	TAT	WT
P1	6	0	6	0	6	0
P2	4	4	4	4	9	5
P3	3	2	3	2	7.	4

2) SJF

Process	Input		Output			
	BT	AT	BT	AT	TAT	WT
P1	4	0	4	0	4	0
P2	7	2	2	2	13	6
P3	2	3	2	3	8	.1
P4	2	3	2	3	5	3

Page No.
Date: / /

3) Round Robin:

Process	Input		Output			
	BT	AT	BT	AT	TAT	WT
P1	5	1	5	1	12	7
P2	4	0	4	0	8	4
P3	7	2	7	2	14	7

4) Priority

Process	Input			Output				
	AT	BT	Priority	AT	BT	Priority	TAT	WT
P1	0	3	2	0	3	2	3	0
P2	2	5	6	2	5	6	16	11
P3	1	4	3	1	4	3	6	2
P4	4	2	5	4	2	5	9	7
P5	6	9	7	6	9	7	21	12
P6	5	4	4	5	4	4	6	7
P7	7	10	10	7	10	10	30	18

Result = Success

Conclusion:
Hence we successfully implemented all the scheduling algorithms in JAVA

```
import java.util.*;

class Process
{
	int pNo=0;
	int arrivalTime=0;
	int burstTime=0;
	int turnAroundTime=0;
	int waitTime=0;
}

class FCFS
{
	Queue<Process> arrivalQueue = new LinkedList<Process>();
	Queue<Process> readyQueue = new LinkedList<Process>();
	Queue<Process> finishQueue= new LinkedList<Process>();

	int numProcess;
	int elapsed=0;
	Process p[];

	FCFS()
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter no. of processes : ");
		numProcess = sc.nextInt();
		p = new Process[numProcess];
		for(int i=0;i<numProcess;i++)
		{
			p[i] = new Process();
			p[i].pNo=i;
			System.out.println("Enter Arrival Time for Process: "+i);
			p[i].arrivalTime = sc.nextInt();
			System.out.println("Enter Burst Time for Process: "+i);
			p[i].burstTime=sc.nextInt();
		}
	}

	void setArrivalQueue()
	{
		Process temp;
		for(int i=0; i<numProcess-1 ;i++)
		{
			for(int j=i+1; j<numProcess ; j++)
			{
				if(p[j].arrivalTime < p[i].arrivalTime)
				{
					temp = p[j];
					p[j] = p[i];
```

```
                                p[i] = temp;
                        }
                }
        }

        for(int i=0;i<numProcess;i++)
        {
                arrivalQueue.add(p[i]);
        }
}

void setReadyQueue()
{
        for(int i=0 ; i<arrivalQueue.size() ;i++)
        {
                Process p = arrivalQueue.remove();
                if(p.arrivalTime <= elapsed)
                {
                        readyQueue.add(p);
                }
                else
                {
                        arrivalQueue.add(p);
                }
        }
}

void calculateFCFS()
{
        setArrivalQueue();
        while(!arrivalQueue.isEmpty() || !readyQueue.isEmpty())
        {
                if(!arrivalQueue.isEmpty())
                {
                        setReadyQueue();
                }

                if(!readyQueue.isEmpty())
                {
                        Process p = readyQueue.remove();
                        elapsed = elapsed + p.burstTime;
                        p.turnAroundTime = elapsed - p.arrivalTime;
                        p.waitTime = p.turnAroundTime - p.burstTime;
                        finishQueue.add(p);
                }
                else
                {
                        elapsed = elapsed + 1;
                }
```

```
		}
	}

	void display()
	{
		double avgWaitTime=0;
		double avgTurnAroundTime=0;
		System.out.println("PN \t AT \t BT \t TAT \t WT");
		while(!finishQueue.isEmpty())
		{
			Process p = finishQueue.remove();
			System.out.println(p.pNo + " \t " + p.arrivalTime + " \t " + p.burstTime + " \t "+
p.turnAroundTime + " \t "+ p.waitTime);
			avgWaitTime = avgWaitTime + p.waitTime;
			avgTurnAroundTime = avgTurnAroundTime + p.turnAroundTime;
		}
		avgWaitTime = avgWaitTime/numProcess;
		avgTurnAroundTime = avgTurnAroundTime/numProcess;
		System.out.println("Average Waiting Time is: "+avgWaitTime);
		System.out.println("Average Turn Around Time is: "+avgTurnAroundTime);
	}

	public static void main(String args[])
	{
		FCFS f = new FCFS();
		f.calculateFCFS();
		f.display();
	}
}



import java.util.*;

class Process
{
	int pNo=0;
	int arrivalTime=0;
	int burstTime=0;
	int turnAroundTime=0;
	int waitTime=0;
	int priority=-1;
}

class Priority
{
	Queue<Process> arrivalQueue = new LinkedList<Process>();
	Queue<Process> readyQueue = new LinkedList<Process>();
	Queue<Process> finishQueue= new LinkedList<Process>();
```

```
int numProcess;
int elapsed=0;
Process p[];

Priority()
{
	Scanner sc = new Scanner(System.in);
	System.out.println("Enter no. of processes : ");
	numProcess = sc.nextInt();
	p = new Process[numProcess];

	for(int i=0;i<numProcess;i++)
	{
		p[i] = new Process();
		p[i].pNo=i;
		System.out.println("Enter arrival Time for Process: "+i);
		p[i].arrivalTime = sc.nextInt();
		System.out.println("Enter Burst Time for Process: "+i);
		p[i].burstTime=sc.nextInt();
		System.out.println("Enter Priority for Process: "+i);
		p[i].priority=sc.nextInt();
	}
}

void setArrivalQueue()
{
	Process temp;
	for(int i=0; i<numProcess-1 ;i++)
	{
		for(int j=i+1; j<numProcess ; j++)
		{
			if(p[j].priority < p[i].priority)
			{
				temp = p[j];
				p[j] = p[i];
				p[i] = temp;
			}
		}
	}

	for(int i=0;i<numProcess;i++)
	{
		arrivalQueue.add(p[i]);
	}
}

void setReadyQueue()
{
	int asize = arrivalQueue.size();
```

```
	for(int i=0;i<asize;i++)
	{
		Process p = arrivalQueue.remove();
		if(p.arrivalTime <= elapsed)
		{
			readyQueue.add(p);
		}
		else
		{
			arrivalQueue.add(p);
		}
	}
}

void cal()
{
	setArrivalQueue();

	while(!arrivalQueue.isEmpty() || !readyQueue.isEmpty())
	{
		if(!arrivalQueue.isEmpty())
		{
			setReadyQueue();
		}

		if(!readyQueue.isEmpty())
		{
			//System.out.println("In readyQueue Queue");
			Process p = readyQueue.remove();
			elapsed = elapsed + p.burstTime;
			p.turnAroundTime = elapsed - p.arrivalTime;
			p.waitTime = p.turnAroundTime - p.burstTime;
			finishQueue.add(p);
		}
		else
		{
			elapsed = elapsed + 1;
		}
	}
}

void display()
{
	double avgWaitTime=0, avgTurnAroundTime=0;
	System.out.println("PN \t AT \t BT \t TAT \t WT \t PT");
	while(!finishQueue.isEmpty())
	{
		Process p = finishQueue.remove();
```

```
                        System.out.println(p.pNo + " \t " + p.arrivalTime + " \t " + p.burstTime + " \t "+
p.turnAroundTime + " \t "+ p.waitTime + " \t "+p.priority);
                        avgWaitTime = avgWaitTime + p.waitTime;
                        avgTurnAroundTime = avgTurnAroundTime + p.turnAroundTime;
                }
                avgWaitTime = avgWaitTime/numProcess;
                avgTurnAroundTime = avgTurnAroundTime/numProcess;
                System.out.println("Average Waiting Time is: "+avgWaitTime);
                System.out.println("Average Turn Around Time is: "+avgTurnAroundTime);
        }
        public static void main(String args[])
        {
                Priority p = new Priority();
                p.cal();
                p.display();
        }
}


import java.util.*;

class Process
{
        int pNo=0;
        int arrivalTime=0;
        int burstTime=0;
        int turnAroundTime=0;
        int waitTime=0;
        int rburst=0;
}

class RR
{
        Queue<Process> arrivalQueue = new LinkedList<Process>();
        Queue<Process> readyQueue = new LinkedList<Process>();
        Queue<Process> finishQueue= new LinkedList<Process>();
        int numProcess;
        int elapsed=0;
        int quantum=0;
        Process p[];
        String seq = "";
        RR()
        {
                Scanner sc = new Scanner(System.in);
                System.out.println("Enter no. of processes:");
                numProcess = sc.nextInt();
                p = new Process[numProcess];

                System.out.println("Enter size of Quantum:");
```

```
        quantum = sc.nextInt();

        for(int i=0;i<numProcess;i++)
        {
                p[i] = new Process();
                p[i].pNo=i;
                System.out.println("Enter arrival Time for Process: "+i);
                p[i].arrivalTime = sc.nextInt();
                System.out.println("Enter Burst Time for Process: "+i);
                p[i].burstTime=sc.nextInt();
                p[i].rburst=p[i].burstTime;
        }
}

void setArrival()
{
        Process temp;
        for(int i=0; i<numProcess-1 ;i++)
        {
                for(int j=i+1; j<numProcess ; j++)
                {
                        if(p[j].arrivalTime < p[i].arrivalTime)
                        {
                                temp = p[j];
                                p[j] = p[i];
                                p[i] = temp;
                        }
                }
        }

        for(int i=0;i<numProcess;i++)
        {
                arrivalQueue.add(p[i]);
        }
}

void setReadyQueue()
{
        int asize = arrivalQueue.size();
        for(int i=0;i<asize;i++)
        {
                Process p = arrivalQueue.remove();
                if(p.arrivalTime <= elapsed)
                {
                        readyQueue.add(p);
                }
                else
                {
                        arrivalQueue.add(p);
```

```
                }
        }
}

void cal()
{
        setArrival();
        int rrflag=0;
        Process p1=p[0];
        while(!arrivalQueue.isEmpty() || !readyQueue.isEmpty())
        {
                if(!arrivalQueue.isEmpty())
                {
                        setReadyQueue();
                }

                if(rrflag == 1)
                {
                        readyQueue.add(p1);
                        rrflag = 0;
                }

                if(!readyQueue.isEmpty())
                {
                        p1 = readyQueue.remove();
                        if(p1.rburst <= quantum)
                        {
                                elapsed = elapsed + p1.rburst;
                                p1.turnAroundTime = elapsed - p1.arrivalTime;
                                p1.waitTime = p1.turnAroundTime - p1.burstTime;
                                finishQueue.add(p1);
                                seq = seq + p1.pNo + " -> ";
                        }
                        else
                        {
                                elapsed = elapsed + quantum;
                                p1.rburst = p1.rburst - quantum;
                                if(!arrivalQueue.isEmpty())
                                {
                                        rrflag=1;
                                }
                                else
                                {
                                        readyQueue.add(p1);
                                }
                                seq = seq + p1.pNo + " -> ";
                        }
                }
                else
```

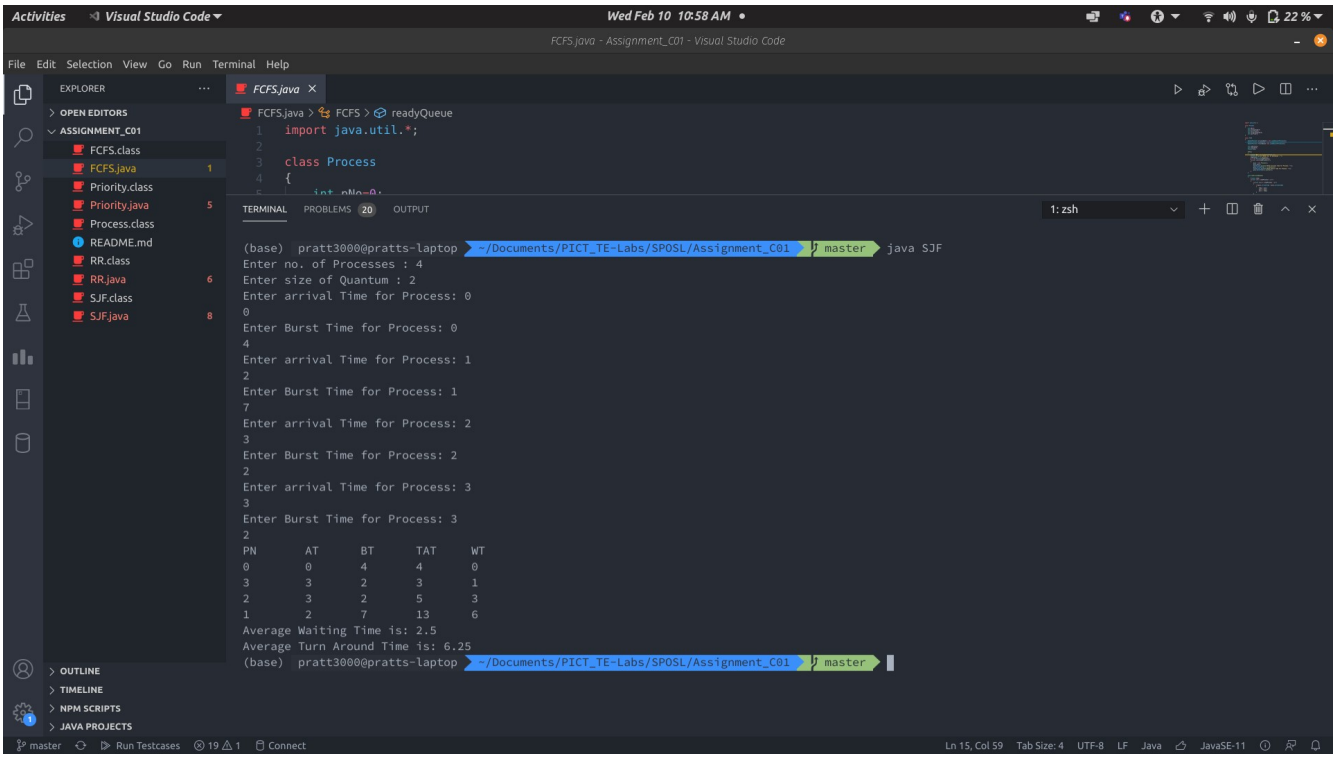
```
                {
                        elapsed = elapsed + 1;
                }
            }
        }

        void display()
        {
            double avgWaitingTime = 0,avgTurnAroundTime = 0;
            System.out.println("PN \t AT \t BT \t TAT \t WT");
            while(!finishQueue.isEmpty())
            {
                Process p = finishQueue.remove();
                System.out.println(p.pNo + " \t " + p.arrivalTime + " \t " + p.burstTime + " \t "+
p.turnAroundTime + " \t "+ p.waitTime);
                avgWaitingTime = avgWaitingTime + p.waitTime;
                avgTurnAroundTime =avgTurnAroundTime + p.turnAroundTime;
            }
            avgWaitingTime = avgWaitingTime/numProcess;
            avgTurnAroundTime = avgTurnAroundTime/numProcess;
            System.out.println("Average Waiting Time is: "+avgWaitingTime);
            System.out.println("Average Turn Around Time is: "+avgTurnAroundTime);
            System.out.println(seq);
        }
        public static void main(String args[])
        {
            RR r = new RR();
            r.cal();
            r.display();
        }
}


import java.util.*;

class Process
{
        int pNo=0;
        int arrivalTime=0;
        int burstTime=0;
        int turnAroundTime=0;
        int waitTime=0;
        int rburst=0;
}

class SJF
{
        Queue<Process> arrivalQueue = new LinkedList<Process>();
        Queue<Process> readyQueue = new LinkedList<Process>();
```

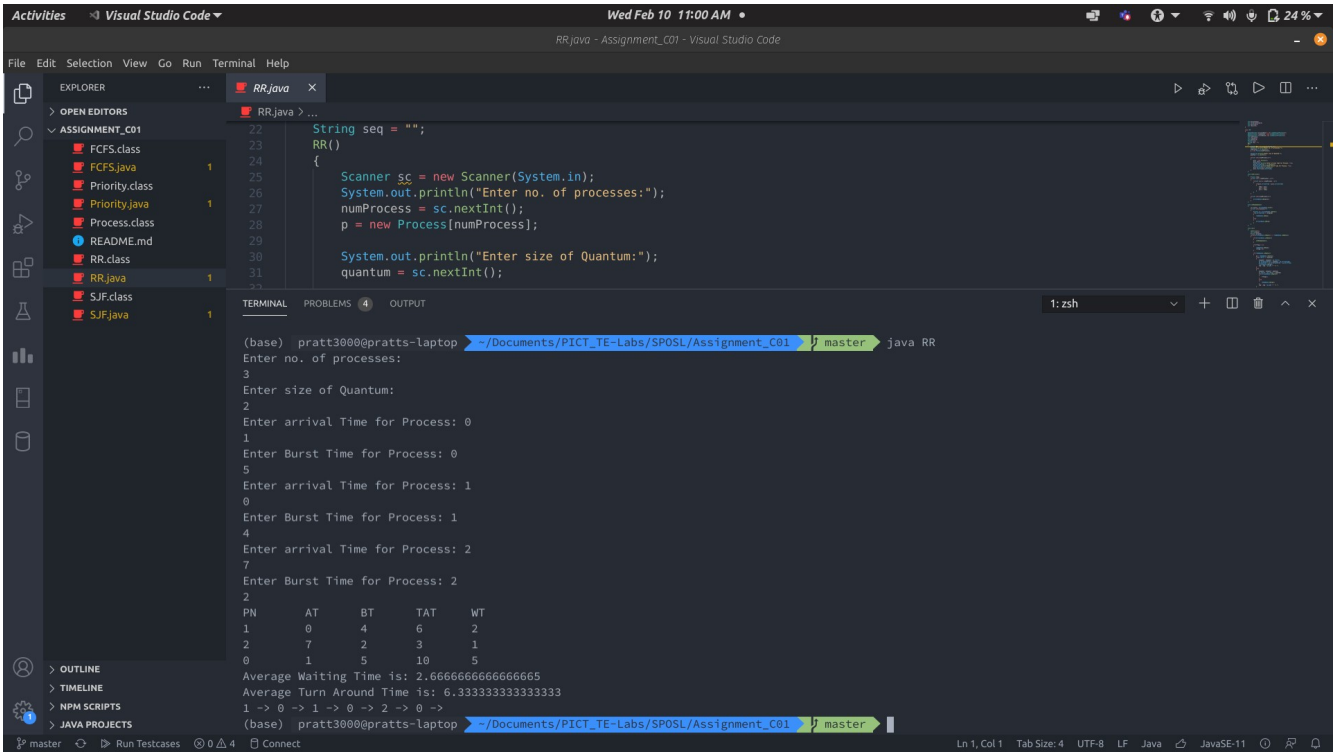
```
	Queue<Process> finishQueue= new LinkedList<Process>();
	int numProcess;
	int elapsed=0;
	int quantum=0;
	Process p[];

	SJF()
	{
		Scanner sc = new Scanner(System.in);
		System.out.print("Enter no. of Processes : ");
		numProcess = sc.nextInt();
		p = new Process[numProcess];

		System.out.print("Enter size of Quantum : ");
		quantum = sc.nextInt();

		for(int i=0;i<numProcess;i++)
		{
			p[i] = new Process();
			p[i].pNo=i;
			System.out.println("Enter arrival Time for Process: "+i);
			p[i].arrivalTime = sc.nextInt();
			System.out.println("Enter Burst Time for Process: "+i);
			p[i].burstTime=sc.nextInt();
			p[i].rburst=p[i].burstTime;
		}
	}

	void setArrival()
	{
		Process temp;
		for(int i=0; i<numProcess-1 ;i++)		//Sorting it in acsending order according to
burst time
		{
			for(int j=i+1; j<numProcess ; j++)
			{
				if(p[j].burstTime < p[i].burstTime)
				{
					temp = p[j];
					p[j] = p[i];
					p[i] = temp;
				}
			}
		}

		for(int i=0;i<numProcess;i++)
		{
			arrivalQueue.add(p[i]);
		}
```

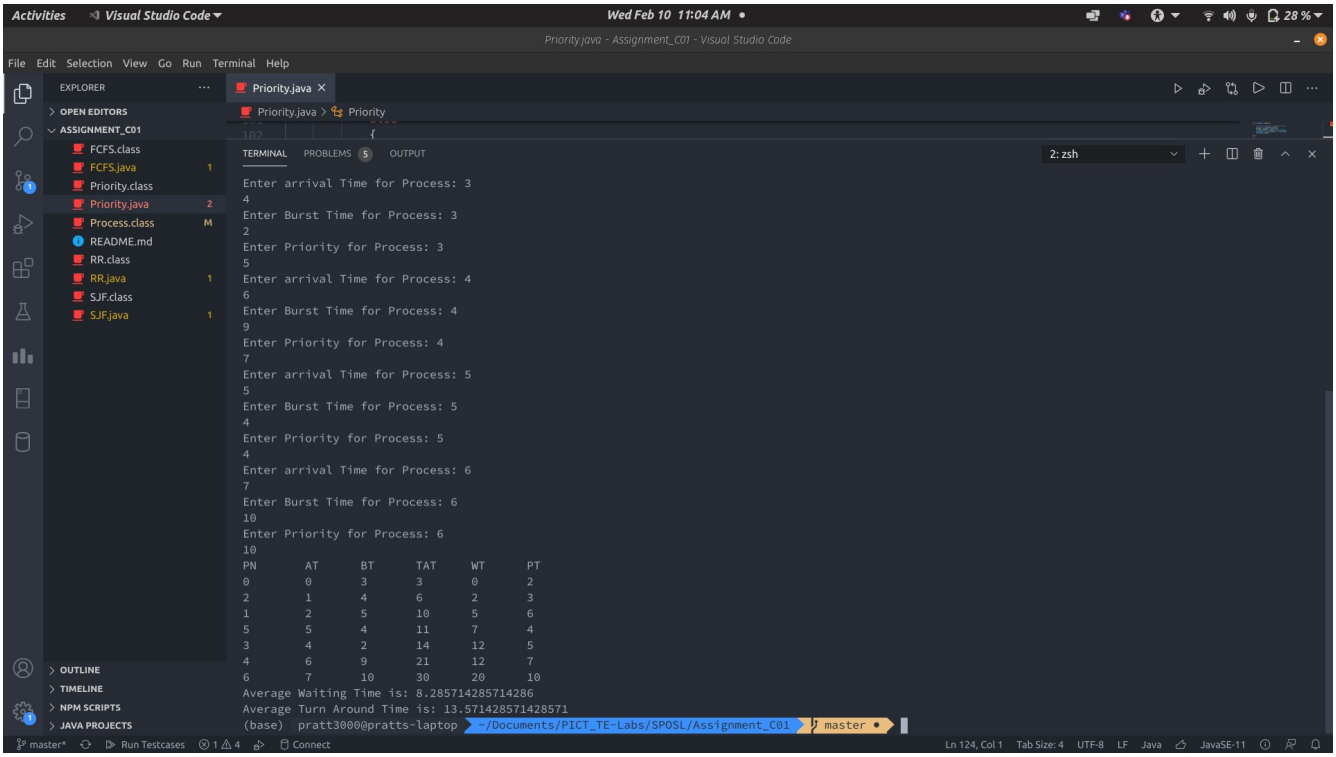
```
}

void setReadyQueue()
{
	for(int i=0;i<arrivalQueue.size();i++)
	{
		Process p = arrivalQueue.remove();
		if(p.arrivalTime <= elapsed)
		{
			readyQueue.add(p);
		}
		else
		{
			arrivalQueue.add(p);
		}
	}
}

void sortReadyQueue()
{
	Process p1[] = new Process[numProcess];    //Temporary array used for sorting
	int queueSize = readyQueue.size();

	for(int i=0;i<queueSize;i++)
	{
		p1[i] = readyQueue.remove();
	}

	Process temp;
	for(int i=0; i<queueSize-1 ;i++)
	{
		for(int j=i+1; j<queueSize ; j++)
		{
			if(p1[j].rburst < p1[i].rburst)
			{
				temp = p1[j];
				p1[j] = p1[i];
				p1[i] = temp;
			}
		}
	}
	for(int i=0;i<queueSize;i++)
	{
		readyQueue.add(p1[i]);
	}
}

void cal()
{
```

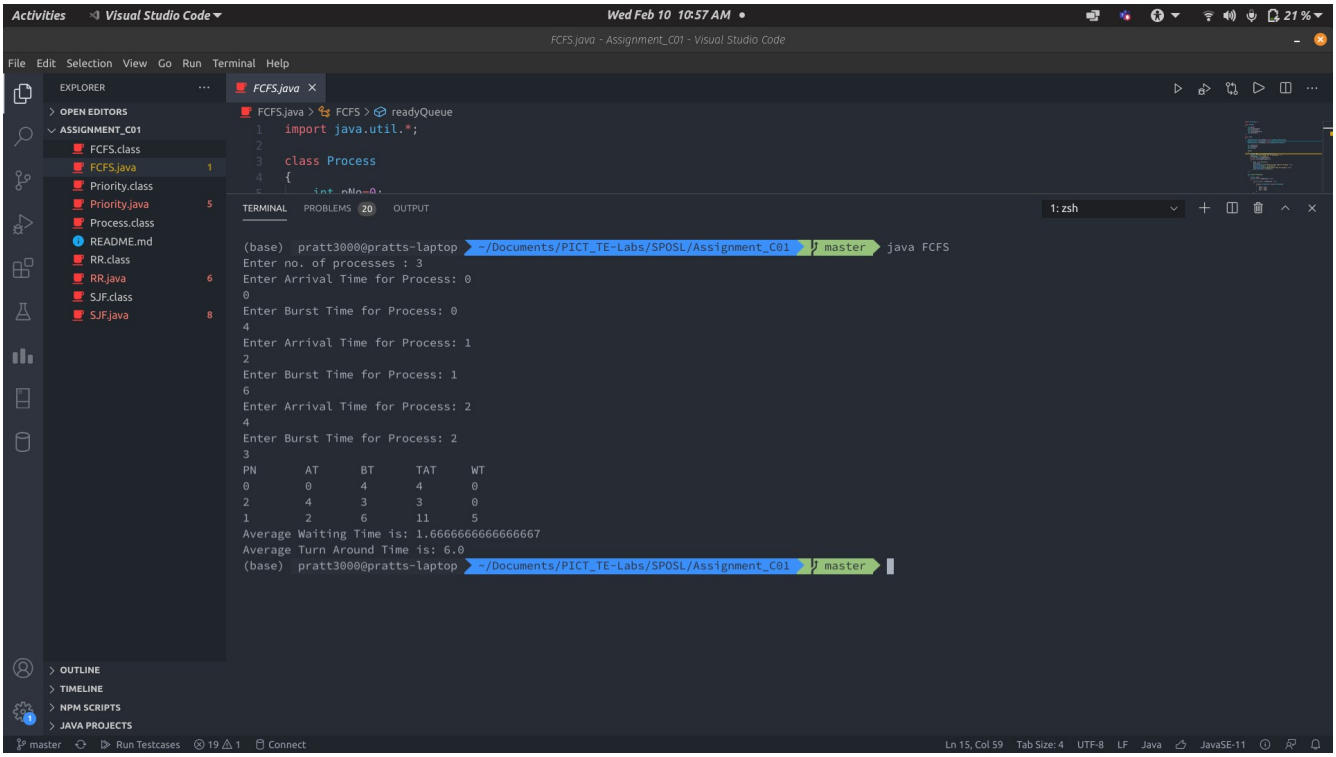
```
		setArrival();

		while(!arrivalQueue.isEmpty() || !readyQueue.isEmpty())
		{
			if(!arrivalQueue.isEmpty())
			{
				setReadyQueue();
			}

			if(!readyQueue.isEmpty())
			{
				if(readyQueue.size() > 1)
				{
					sortReadyQueue();
				}
				Process p = readyQueue.remove();
				if(p.rburst <= quantum)
				{
					elapsed = elapsed + p.rburst;
					p.turnAroundTime = elapsed - p.arrivalTime;
					p.waitTime = p.turnAroundTime - p.burstTime;
					finishQueue.add(p);
				}
				else
				{
					elapsed = elapsed + quantum;
					p.rburst = p.rburst - quantum;
					readyQueue.add(p);
				}
			}
			else
			{
				elapsed = elapsed + 1;
			}
		}
	}

	void display()
	{
		double avgWaitingTime=0,avgTurnAroundTime=0;
		System.out.println("PN \t AT \t BT \t TAT \t WT");
		while(!finishQueue.isEmpty())
		{
			Process p = finishQueue.remove();
			System.out.println(p.pNo + " \t " + p.arrivalTime + " \t " + p.burstTime + " \t "+
p.turnAroundTime + " \t "+ p.waitTime);
			avgWaitingTime = avgWaitingTime + p.waitTime;
			avgTurnAroundTime = avgTurnAroundTime + p.turnAroundTime;
		}
```

```
                avgWaitingTime = avgWaitingTime/numProcess;
                avgTurnAroundTime = avgTurnAroundTime/numProcess;
                System.out.println("Average Waiting Time is: "+avgWaitingTime);
                System.out.println("Average Turn Around Time is: "+avgTurnAroundTime);
        }
        public static void main(String args[])
        {
                SJF s = new SJF();
                s.cal();
                s.display();
        }
}
```




Activities
Visual Studio Code
Wed Feb 10 11:00 AM
RR.java - Assignment_C01 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
ASSIGNMENT_C01
FCFS.class
FCFS.java 1
Priority.class
Priority.java 1
Process.class
README.md
RR.class
RR.java 1
SJF.class
SJF.java 1
RR.java
String seq = "";
RR()
{
Scanner sc = new Scanner(System.in);
System.out.println("Enter no. of processes:");
numProcess = sc.nextInt();
p = new Process[numProcess];
System.out.println("Enter size of Quantum:");
quantum = sc.nextInt();
TERMINAL PROBLEMS 4 OUTPUT
1: zsh
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL/Assignment_C01 master java RR
Enter no. of processes:
3
Enter size of Quantum:
2
Enter arrival Time for Process: 0
1
Enter Burst Time for Process: 0
5
Enter arrival Time for Process: 1
0
Enter Burst Time for Process: 1
4
Enter arrival Time for Process: 2
7
Enter Burst Time for Process: 2
2
PN AT BT TAT WT
1 0 4 6 2
2 7 2 3 1
0 1 5 10 5
Average Waiting Time is: 2.6666666666666665
Average Turn Around Time is: 6.333333333333333
1 -> 0 -> 1 -> 0 -> 2 -> 0 ->
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL/Assignment_C01 master
OUTLINE
TIMELINE
NPM SCRIPTS
JAVA PROJECTS
master Run Testcases 0 4 Connect
Ln 1, Col 1 Tab Size: 4 UTF-8 LF Java JavaSE-11


Activities
Visual Studio Code
Wed Feb 10 11:04 AM
Priority.java - Assignment_C01 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
ASSIGNMENT_C01
FCFS.class
FCFS.java 1
Priority.class
Priority.java 2
Process.class M
README.md
RR.class
RR.java 1
SJF.class
SJF.java 1
Priority.java
Priority.java > Priority
{
TERMINAL PROBLEMS 5 OUTPUT
2: zsh
Enter arrival Time for Process: 3
4
Enter Burst Time for Process: 3
2
Enter Priority for Process: 3
5
Enter arrival Time for Process: 4
6
Enter Burst Time for Process: 4
9
Enter Priority for Process: 4
7
Enter arrival Time for Process: 5
5
Enter Burst Time for Process: 5
4
Enter Priority for Process: 5
4
Enter arrival Time for Process: 6
7
Enter Burst Time for Process: 6
10
Enter Priority for Process: 6
10
PN AT BT TAT WT PT
0 0 3 3 0 2
2 1 4 6 2 3
1 2 5 10 5 6
5 5 4 11 7 4
3 4 2 14 12 5
4 6 9 21 12 7
6 7 10 30 20 10
Average Waiting Time is: 8.285714285714286
Average Turn Around Time is: 13.571428571428571
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL/Assignment_C01 master
OUTLINE
TIMELINE
NPM SCRIPTS
JAVA PROJECTS
master* Run Testcases 1 4 Connect
Ln 124, Col 1 Tab Size: 4 UTF-8 LF Java JavaSE-11


Activities
Visual Studio Code
Wed Feb 10 10:57 AM
FCFS.java - Assignment_C01 - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS
ASSIGNMENT_C01
FCFS.class
FCFS.java 1
Priority.class
Priority.java 5
Process.class
README.md
RR.class
RR.java 6
SJF.class
SJF.java 8
OUTLINE
TIMELINE
NPM SCRIPTS
JAVA PROJECTS
FCFS.java
FCFS.java > FCFS > readyQueue
1 import java.util.*;
3 class Process
4 {
TERMINAL PROBLEMS 20 OUTPUT
1: zsh
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL/Assignment_C01 master java FCFS
Enter no. of processes : 3
Enter Arrival Time for Process: 0
0
Enter Burst Time for Process: 0
4
Enter Arrival Time for Process: 1
2
Enter Burst Time for Process: 1
6
Enter Arrival Time for Process: 2
4
Enter Burst Time for Process: 2
3
PN AT BT TAT WT
0 0 4 4 0
2 4 3 3 0
1 2 6 11 5
Average Waiting Time is: 1.6666666666666667
Average Turn Around Time is: 6.0
(base) pratt3000@pratts-laptop ~/Documents/PICT_TE-Labs/SPOSL/Assignment_C01 master
master Run Testcases 19 1 Connect
Ln 15, Col 59 Tab Size: 4 UTF-8 LF Java JavaSE-11