Project Trimester

Team Members

Sally Ghonaim	2110266
Leen Aljabri	2113008
Rahaf Alghandi	2112014

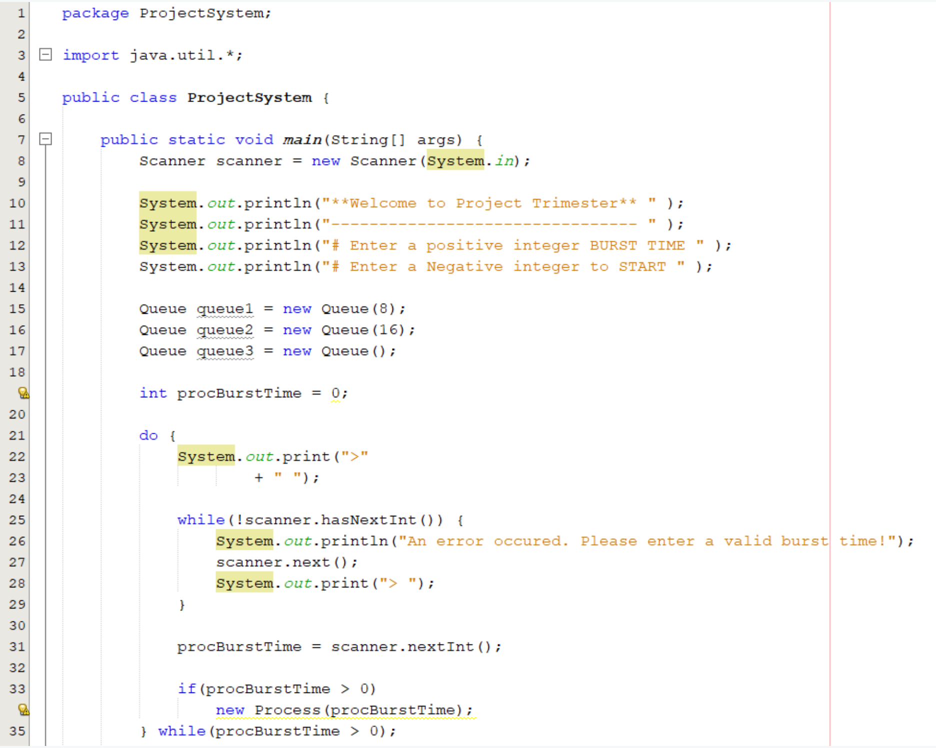
Tasks

Sally Ghonaim	:	Process Class
Leen Aljabri	:	Main Class
Rahaf Alghandi	:	Queue Class + Presentation

01

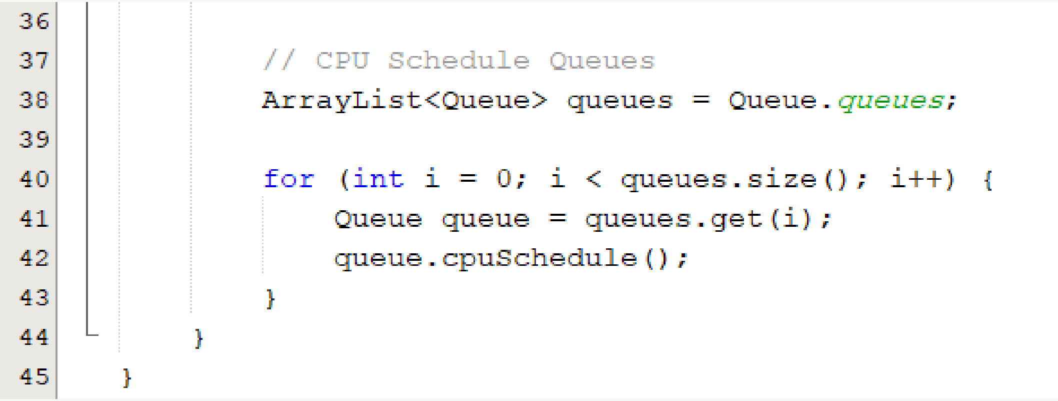
Project Code

ProjectSystem Main



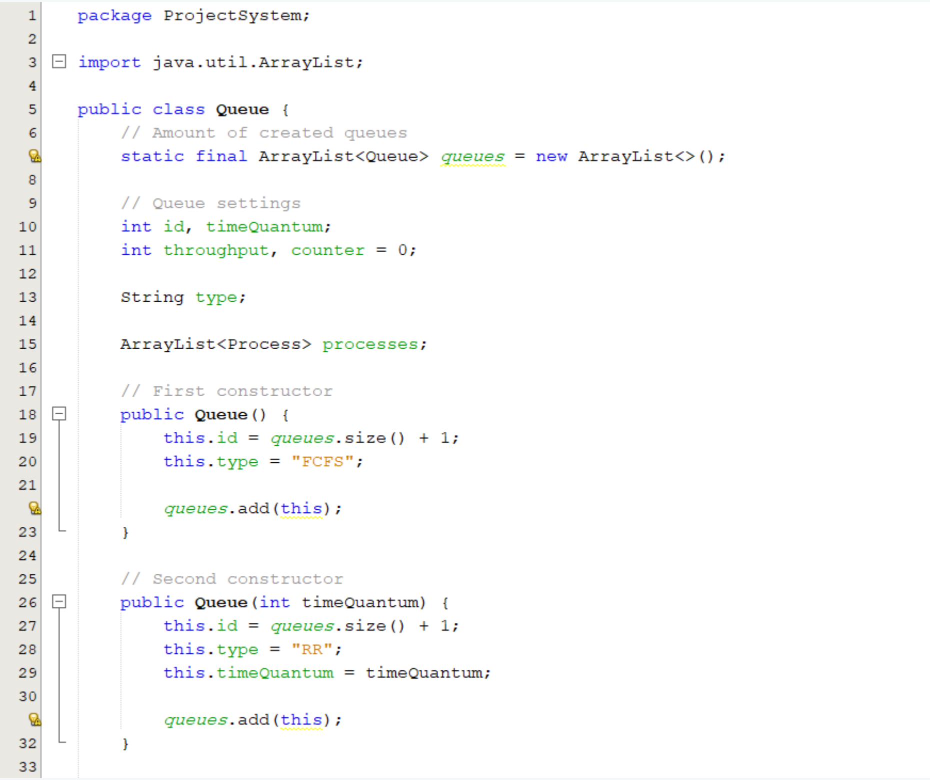
```java
package ProjectSystem;

import java.util.*;

public class ProjectSystem {

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println("**Welcome to Project Trimester** " );
        System.out.println("-------------------------------- " );
        System.out.println("# Enter a positive integer BURST TIME " );
        System.out.println("# Enter a Negative integer to START " );

        Queue queue1 = new Queue(8);
        Queue queue2 = new Queue(16);
        Queue queue3 = new Queue();

        int procBurstTime = 0;

        do {
            System.out.print(">"
                    + " ");

            while(!scanner.hasNextInt()) {
                System.out.println("An error occured. Please enter a valid burst time!");
                scanner.next();
                System.out.print("> ");
            }

            procBurstTime = scanner.nextInt();

            if(procBurstTime > 0)
                new Process(procBurstTime);
        } while(procBurstTime > 0);
```

ProjectSystem Main (Cont.)



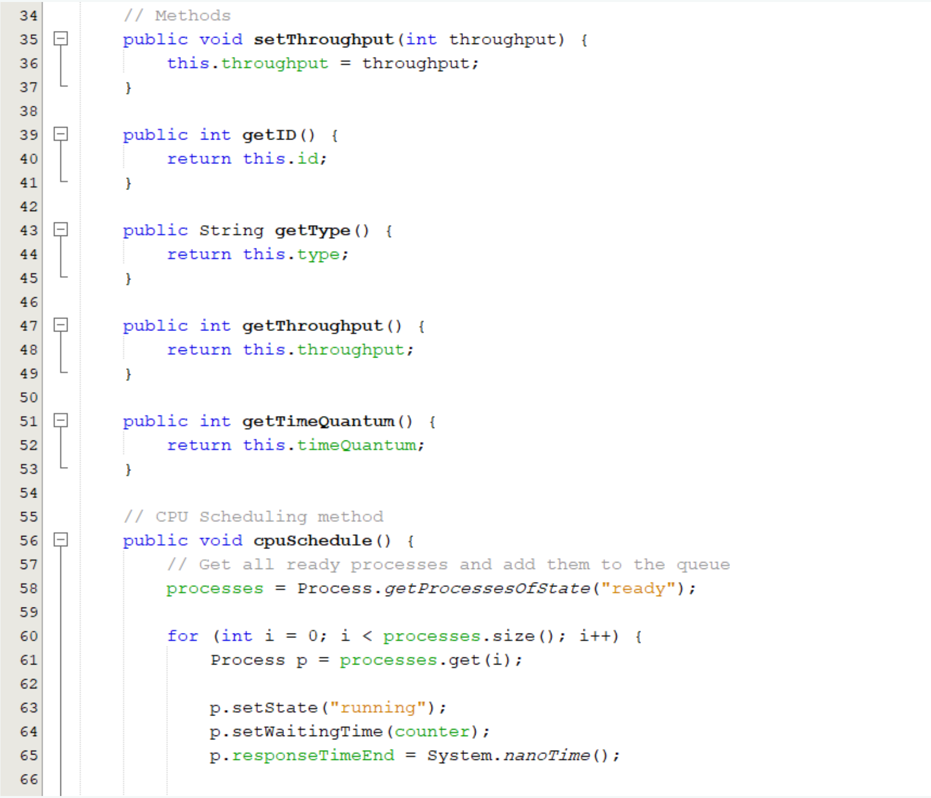
```

        // CPU Schedule Queues
        ArrayList<Queue> queues = Queue.queues;

        for (int i = 0; i < queues.size(); i++) {
            Queue queue = queues.get(i);
            queue.cpuSchedule();
        }
    }
}
```

Queue Class



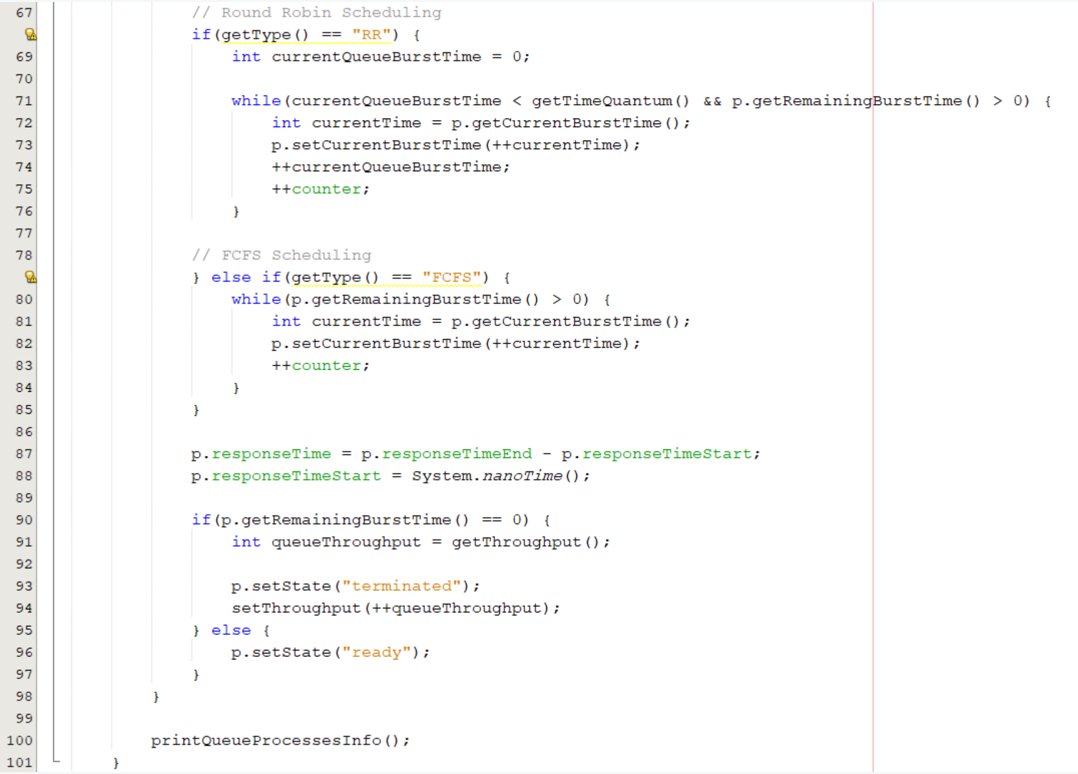
```java
package ProjectSystem;

import java.util.ArrayList;

public class Queue {
    // Amount of created queues
    static final ArrayList<Queue> queues = new ArrayList<>();

    // Queue settings
    int id, timeQuantum;
    int throughput, counter = 0;

    String type;

    ArrayList<Process> processes;

    // First constructor
    public Queue() {
        this.id = queues.size() + 1;
        this.type = "FCFS";

        queues.add(this);
    }

    // Second constructor
    public Queue(int timeQuantum) {
        this.id = queues.size() + 1;
        this.type = "RR";
        this.timeQuantum = timeQuantum;

        queues.add(this);
    }

```

Queue Class (Cont.)



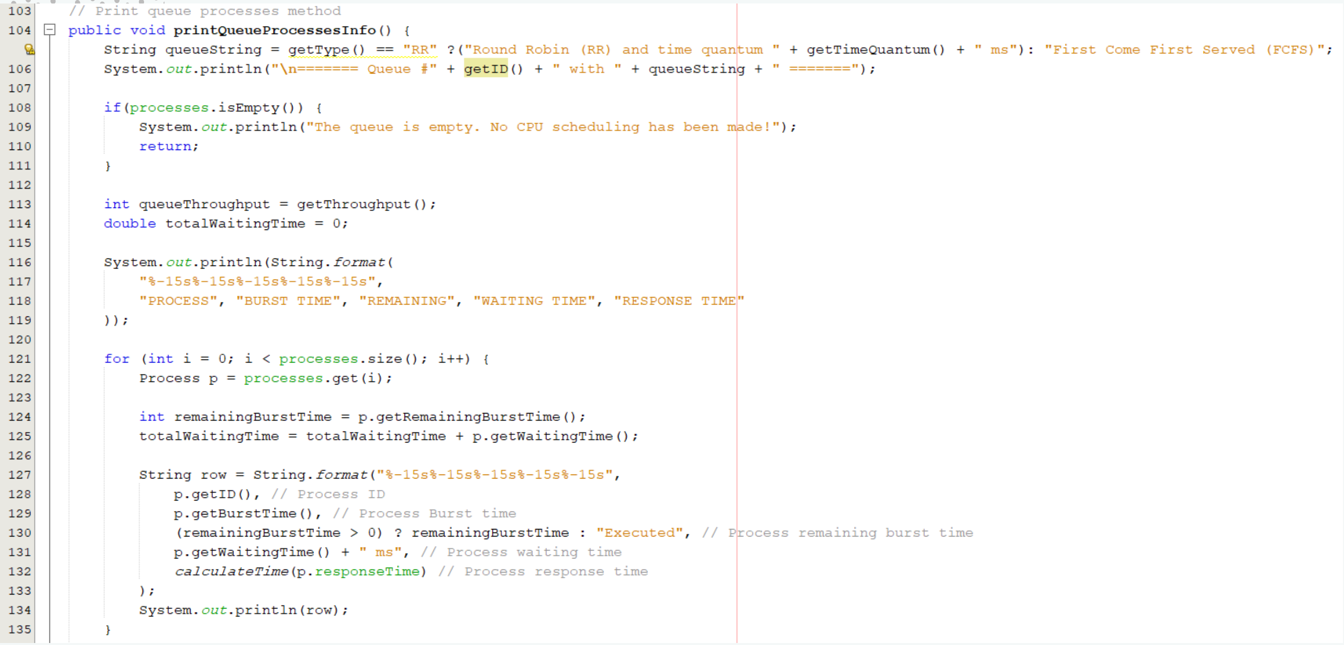
```java
    // Methods
    public void setThroughput(int throughput) {
        this.throughput = throughput;
    }

    public int getID() {
        return this.id;
    }

    public String getType() {
        return this.type;
    }

    public int getThroughput() {
        return this.throughput;
    }

    public int getTimeQuantum() {
        return this.timeQuantum;
    }

    // CPU Scheduling method
    public void cpuSchedule() {
        // Get all ready processes and add them to the queue
        processes = Process.getProcessesOfState("ready");

        for (int i = 0; i < processes.size(); i++) {
            Process p = processes.get(i);

            p.setState("running");
            p.setWaitingTime(counter);
            p.responseTimeEnd = System.nanoTime();

```

Queue Class (Cont.)



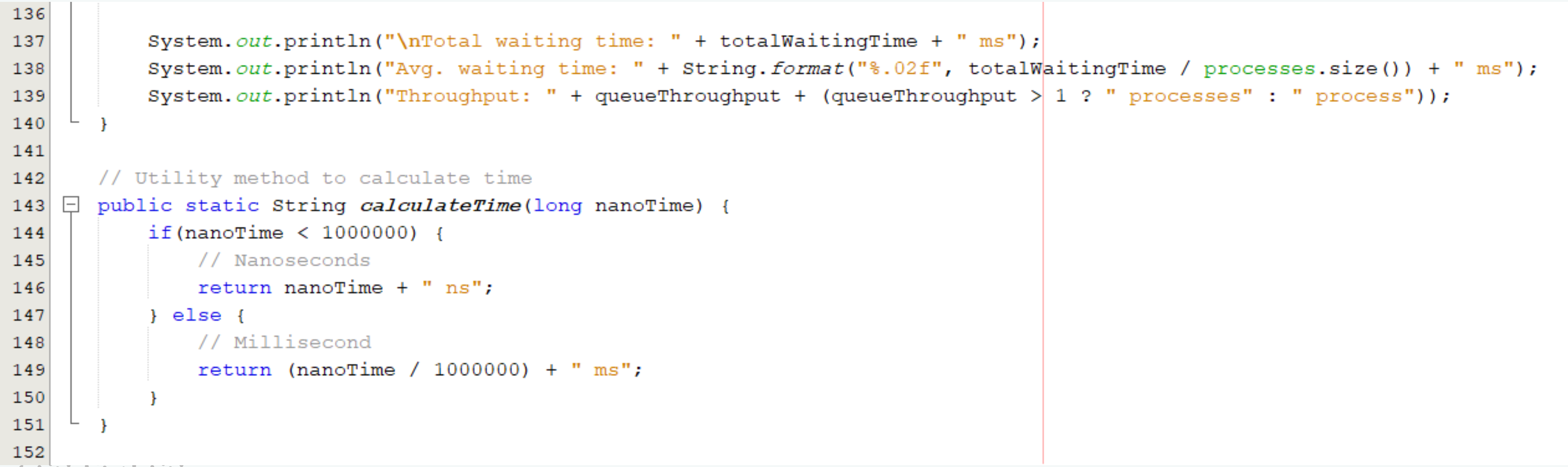
```
        // Round Robin Scheduling
        if(getType() == "RR") {
            int currentQueueBurstTime = 0;

            while(currentQueueBurstTime < getTimeQuantum() && p.getRemainingBurstTime() > 0) {
                int currentTime = p.getCurrentBurstTime();
                p.setCurrentBurstTime(++currentTime);
                ++currentQueueBurstTime;
                ++counter;
            }

        // FCFS Scheduling
        } else if(getType() == "FCFS") {
            while(p.getRemainingBurstTime() > 0) {
                int currentTime = p.getCurrentBurstTime();
                p.setCurrentBurstTime(++currentTime);
                ++counter;
            }
        }

        p.responseTime = p.responseTimeEnd - p.responseTimeStart;
        p.responseTimeStart = System.nanoTime();

        if(p.getRemainingBurstTime() == 0) {
            int queueThroughput = getThroughput();

            p.setState("terminated");
            setThroughput(++queueThroughput);
        } else {
            p.setState("ready");
        }
    }

    printQueueProcessesInfo();
}
```

Queue Class (Cont.)



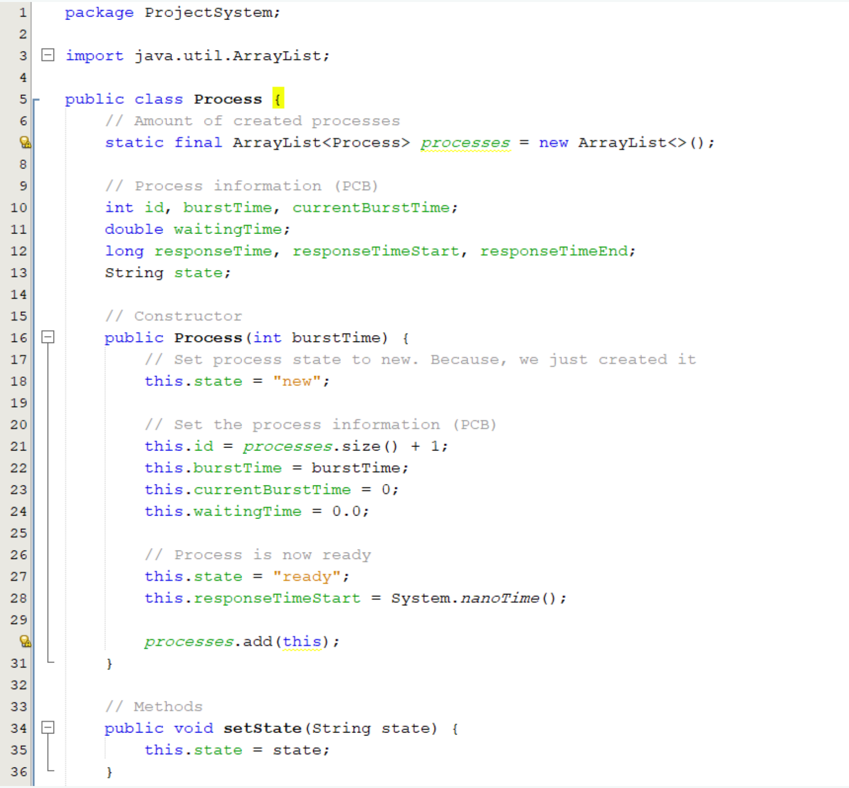
```java
// Print queue processes method
public void printQueueProcessesInfo() {
    String queueString = getType() == "RR" ?("Round Robin (RR) and time quantum " + getTimeQuantum() + " ms"): "First Come First Served (FCFS)";
    System.out.println("\n======= Queue #" + getID() + " with " + queueString + " =======");

    if(processes.isEmpty()) {
        System.out.println("The queue is empty. No CPU scheduling has been made!");
        return;
    }

    int queueThroughput = getThroughput();
    double totalWaitingTime = 0;

    System.out.println(String.format(
        "%-15s%-15s%-15s%-15s%-15s",
        "PROCESS", "BURST TIME", "REMAINING", "WAITING TIME", "RESPONSE TIME"
    ));

    for (int i = 0; i < processes.size(); i++) {
        Process p = processes.get(i);

        int remainingBurstTime = p.getRemainingBurstTime();
        totalWaitingTime = totalWaitingTime + p.getWaitingTime();

        String row = String.format("%-15s%-15s%-15s%-15s%-15s",
            p.getID(), // Process ID
            p.getBurstTime(), // Process Burst time
            (remainingBurstTime > 0) ? remainingBurstTime : "Executed", // Process remaining burst time
            p.getWaitingTime() + " ms", // Process waiting time
            calculateTime(p.responseTime) // Process response time
        );
        System.out.println(row);
    }
```

Queue Class (Cont.)



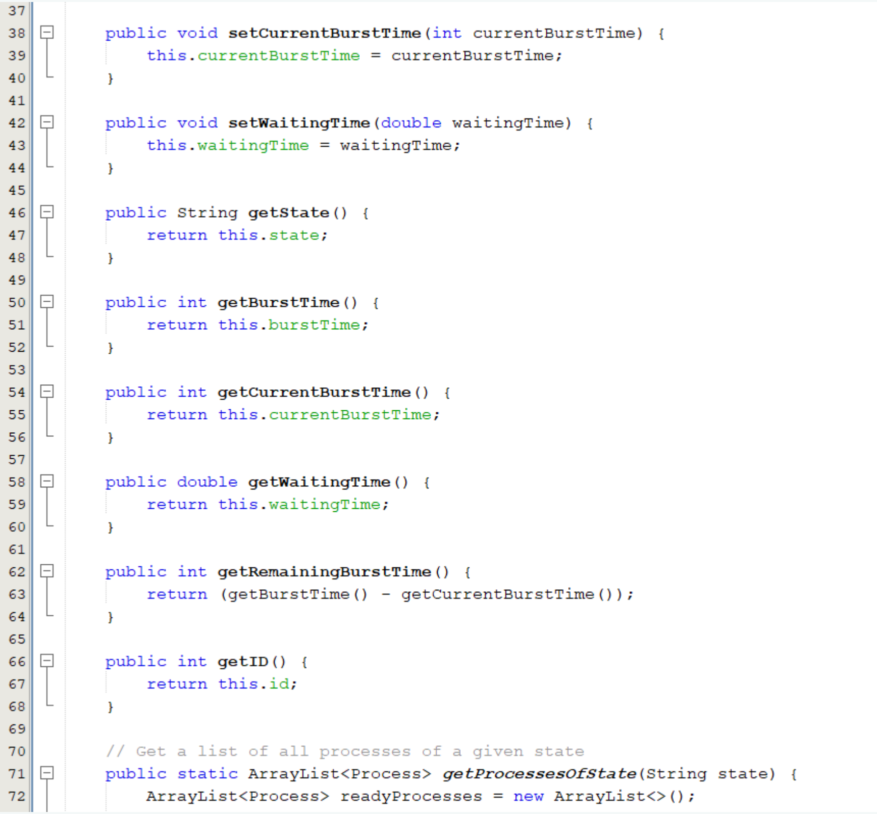
```java

        System.out.println("\nTotal waiting time: " + totalWaitingTime + " ms");
        System.out.println("Avg. waiting time: " + String.format("%.02f", totalWaitingTime / processes.size()) + " ms");
        System.out.println("Throughput: " + queueThroughput + (queueThroughput > 1 ? " processes" : " process"));
    }

    // Utility method to calculate time
    public static String calculateTime(long nanoTime) {
        if(nanoTime < 1000000) {
            // Nanoseconds
            return nanoTime + " ns";
        } else {
            // Millisecond
            return (nanoTime / 1000000) + " ms";
        }
    }

```

Process Class



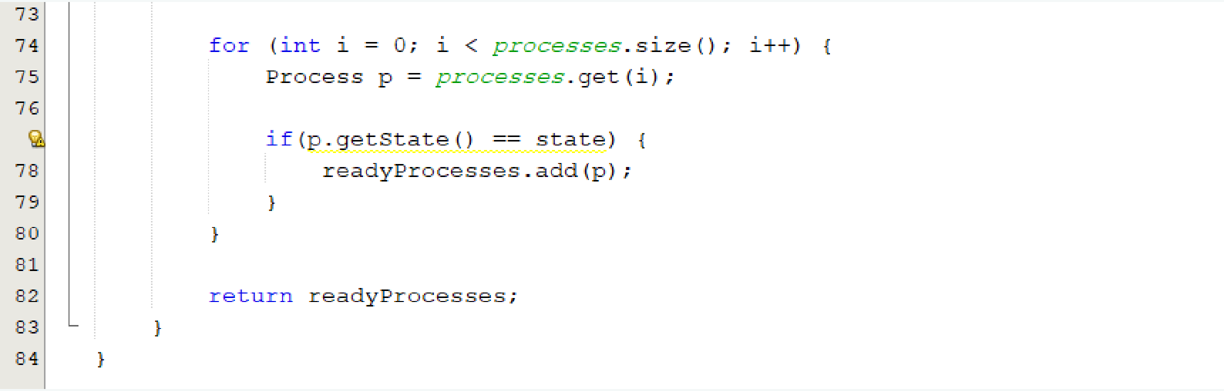
```java
package ProjectSystem;

import java.util.ArrayList;

public class Process {
    // Amount of created processes
    static final ArrayList<Process> processes = new ArrayList<>();

    // Process information (PCB)
    int id, burstTime, currentBurstTime;
    double waitingTime;
    long responseTime, responseTimeStart, responseTimeEnd;
    String state;

    // Constructor
    public Process(int burstTime) {
        // Set process state to new. Because, we just created it
        this.state = "new";

        // Set the process information (PCB)
        this.id = processes.size() + 1;
        this.burstTime = burstTime;
        this.currentBurstTime = 0;
        this.waitingTime = 0.0;

        // Process is now ready
        this.state = "ready";
        this.responseTimeStart = System.nanoTime();

        processes.add(this);
    }

    // Methods
    public void setState(String state) {
        this.state = state;
    }
```

Process Class (Cont.)



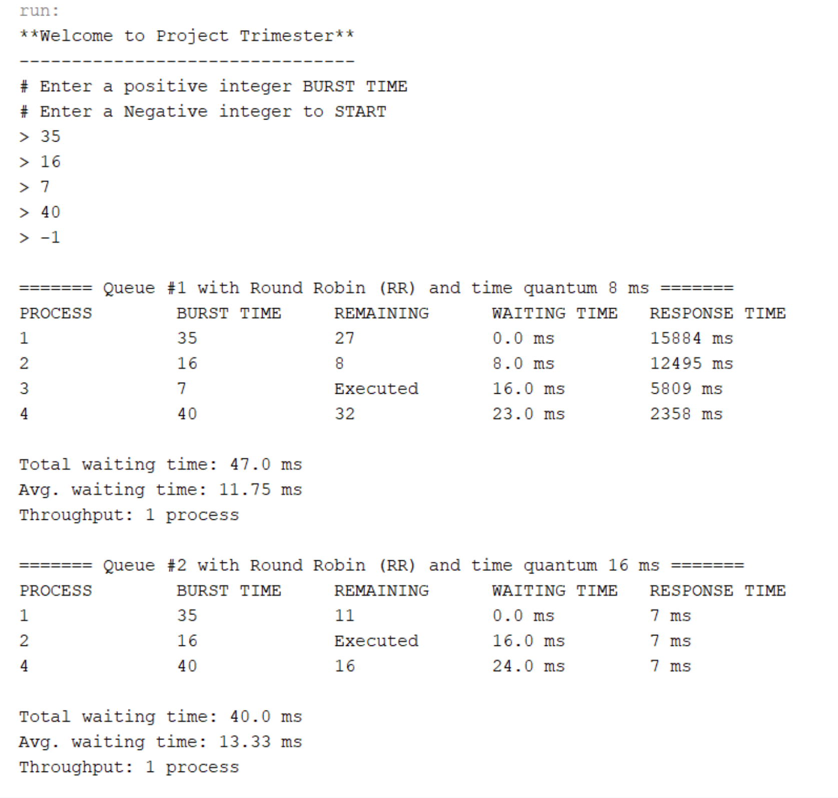
```java

    public void setCurrentBurstTime(int currentBurstTime) {
        this.currentBurstTime = currentBurstTime;
    }

    public void setWaitingTime(double waitingTime) {
        this.waitingTime = waitingTime;
    }

    public String getState() {
        return this.state;
    }

    public int getBurstTime() {
        return this.burstTime;
    }

    public int getCurrentBurstTime() {
        return this.currentBurstTime;
    }

    public double getWaitingTime() {
        return this.waitingTime;
    }

    public int getRemainingBurstTime() {
        return (getBurstTime() - getCurrentBurstTime());
    }

    public int getID() {
        return this.id;
    }

    // Get a list of all processes of a given state
    public static ArrayList<Process> getProcessesOfState(String state) {
        ArrayList<Process> readyProcesses = new ArrayList<>();
```

Process Class (Cont.)



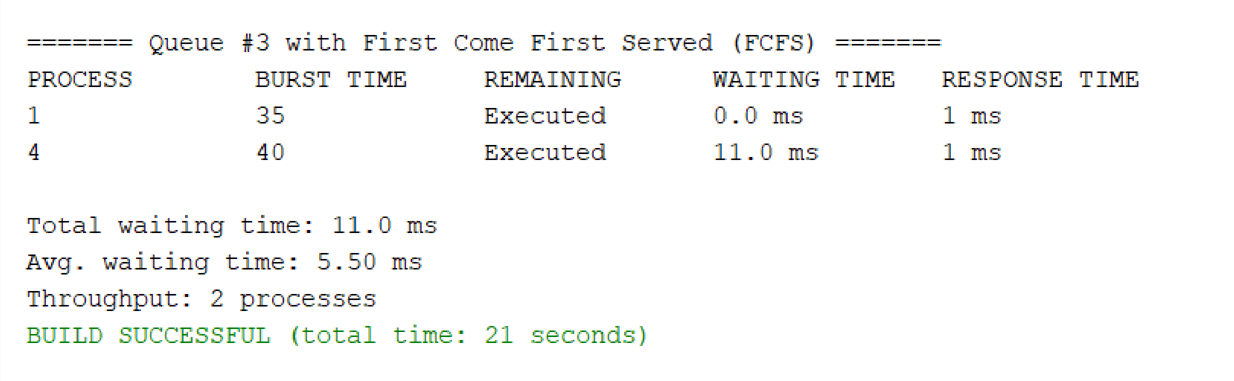
```

        for (int i = 0; i < processes.size(); i++) {
            Process p = processes.get(i);

            if(p.getState() == state) {
                readyProcesses.add(p);
            }
        }

        return readyProcesses;
    }
}
```

OutPut



```
run:
**Welcome to Project Trimester**
--------------------------------
# Enter a positive integer BURST TIME
# Enter a Negative integer to START
> 35
> 16
> 7
> 40
> -1

======= Queue #1 with Round Robin (RR) and time quantum 8 ms =======
PROCESS        BURST TIME     REMAINING      WAITING TIME   RESPONSE TIME
1              35             27             0.0 ms         15884 ms
2              16             8              8.0 ms         12495 ms
3              7              Executed       16.0 ms        5809 ms
4              40             32             23.0 ms        2358 ms

Total waiting time: 47.0 ms
Avg. waiting time: 11.75 ms
Throughput: 1 process

======= Queue #2 with Round Robin (RR) and time quantum 16 ms =======
PROCESS        BURST TIME     REMAINING      WAITING TIME   RESPONSE TIME
1              35             11             0.0 ms         7 ms
2              16             Executed       16.0 ms        7 ms
4              40             16             24.0 ms        7 ms

Total waiting time: 40.0 ms
Avg. waiting time: 13.33 ms
Throughput: 1 process
```

OutPut (Cont.)



```
======= Queue #3 with First Come First Served (FCFS) =======
PROCESS        BURST TIME     REMAINING      WAITING TIME   RESPONSE TIME
1              35             Executed       0.0 ms         1 ms
4              40             Executed       11.0 ms        1 ms

Total waiting time: 11.0 ms
Avg. waiting time: 5.50 ms
Throughput: 2 processes
BUILD SUCCESSFUL (total time: 21 seconds)
```

02

Feature & Capabilities

Flexible

More flexible than Multilevel queue scheduling.

Reduce Time

This algoritm helps in reducing the respone time.

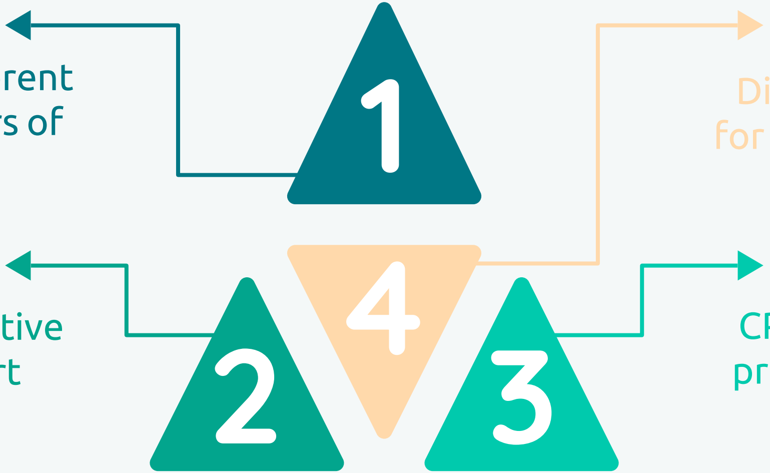
Multi Processor

This algoritm allows different processes to move between different queues.

03

Instructing

STEPS



STEP 1

User enters different positive numbers of burst time

STEP 2

User enters negative number to start

STEP 3

CPU schedule the processes into the given queues

STEP 4

Display the results for each process, and queue

THANKS!

Do you have any questions ??