Processor Execution Simulator

Written by: Moayad AL-Saleh

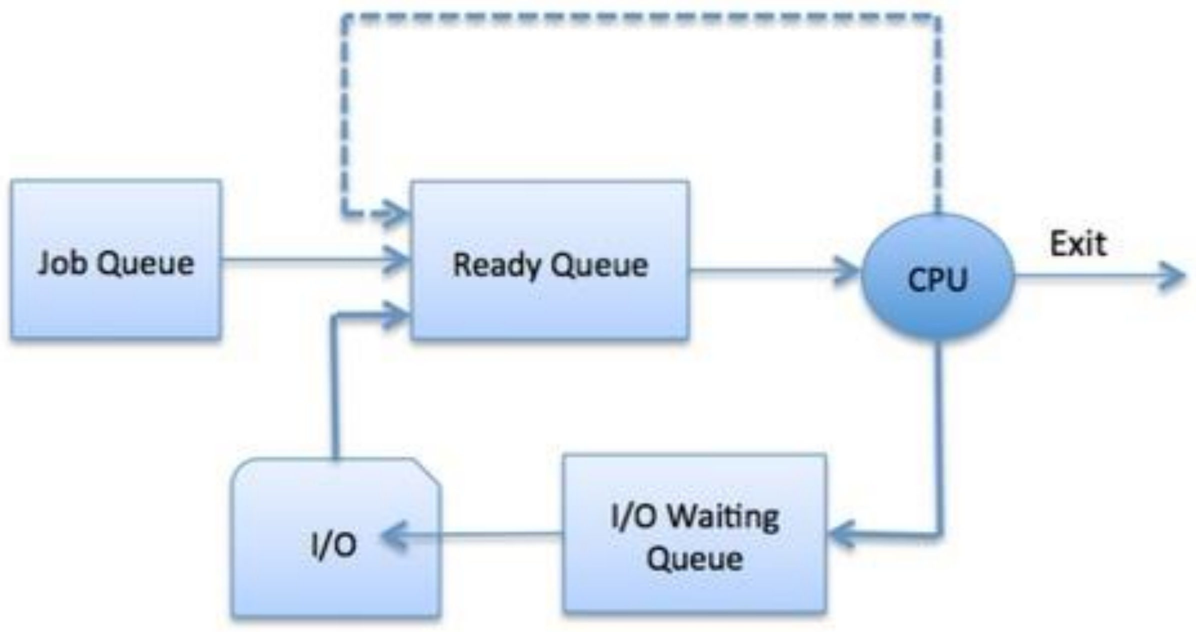
Contents

What is the problem?

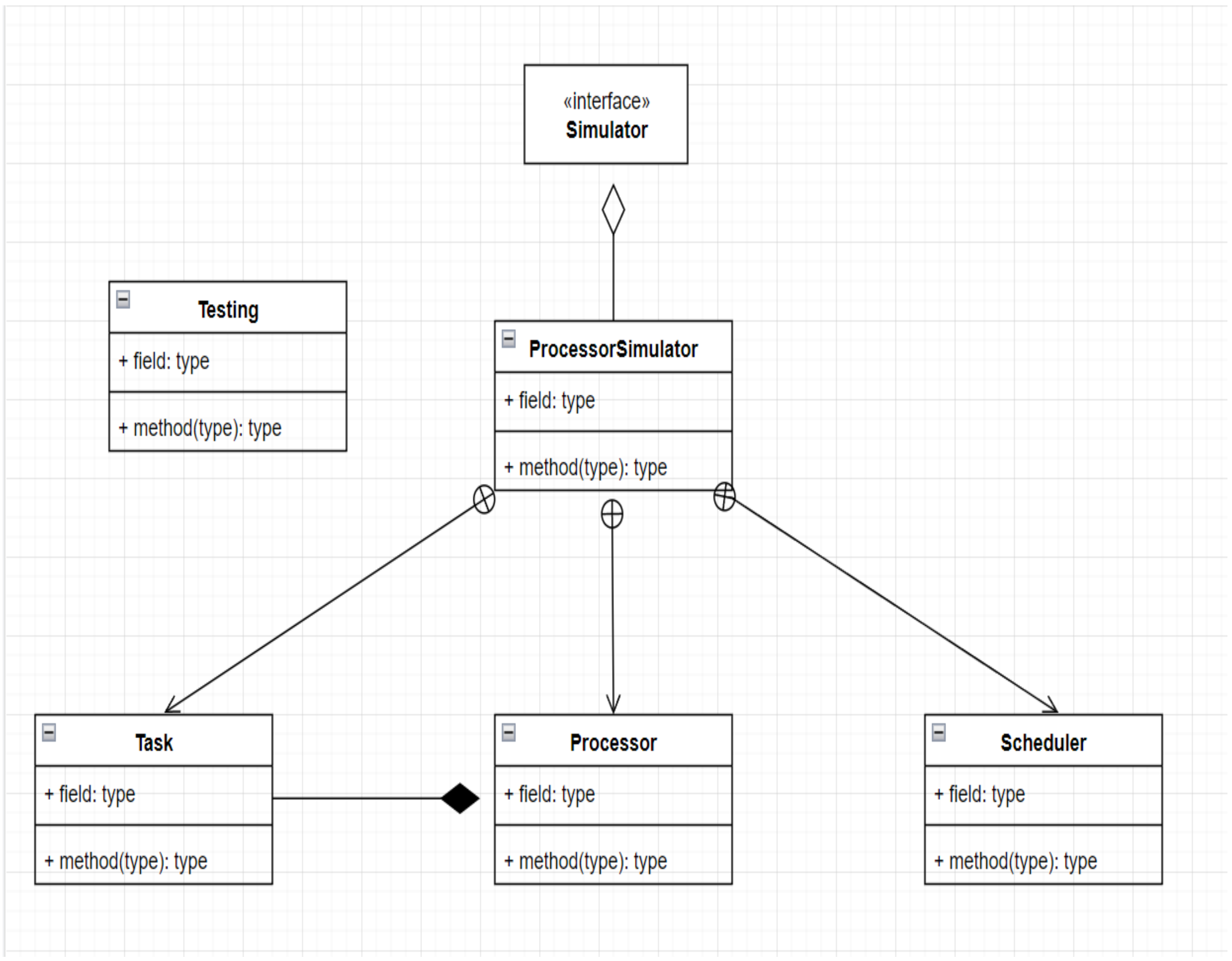
The required build a simulator that simulates processor execution for processes (i.e., tasks).

Consider the following notes about the processor execution:

- A processor can only execute one task at a time.
- The number of processors is fixed during the whole simulation.
- All processors are synchronized, i.e., they all have the same clock.
- Processors are given unique ids as follows: P1, P2, P3, P4, etc.
- Tasks are given unique ids as follows: T1, T2, T3, T4, etc.
- Clock cycles are given unique ids as follows: C1, C2, C3, C4, etc.



UML Diagram


«interface»
Simulator
Testing
+ field: type
+ method(type): type
ProcessorSimulator
+ field: type
+ method(type): type
Task
+ field: type
+ method(type): type
Processor
+ field: type
+ method(type): type
Scheduler
+ field: type
+ method(type): type

The plan to solve the problem :

I read Processor Execution Simulator Assignment very well.

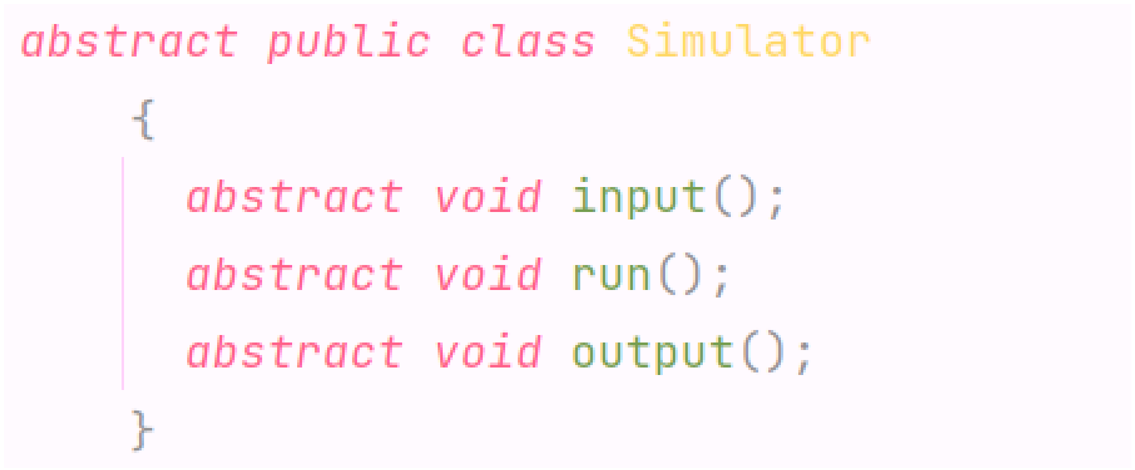
Find the best possible way to create processor simulator.

Apply the solution programmatically so that it is practical and covers all cases.

The Core Classes in the project

Simulator:

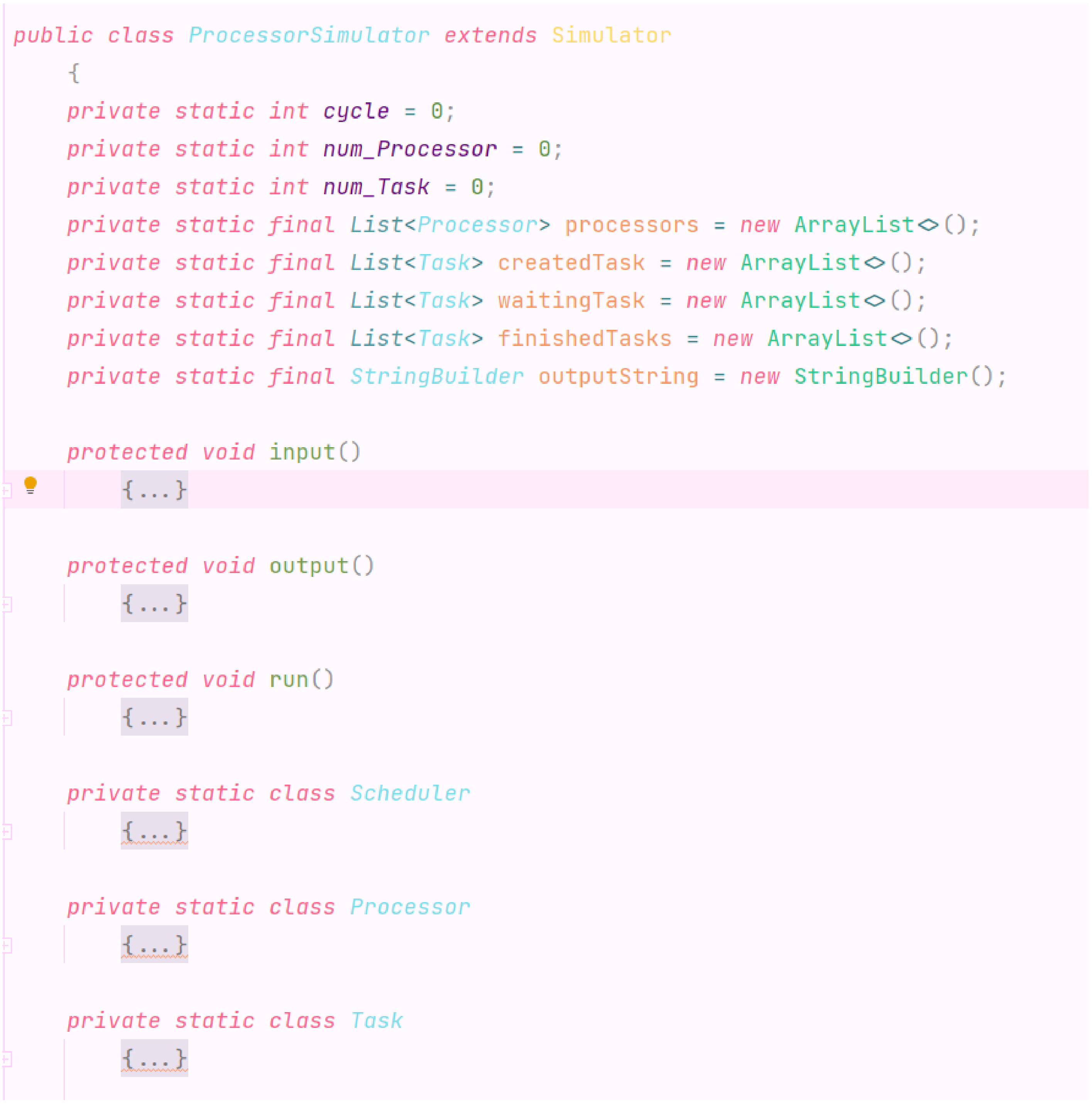
The Simulator is interface, which contains the most important methods that should be in any simulator.



```
abstract public class Simulator
    {
        abstract void input();
        abstract void run();
        abstract void output();
    }
```

Processor Simulator:

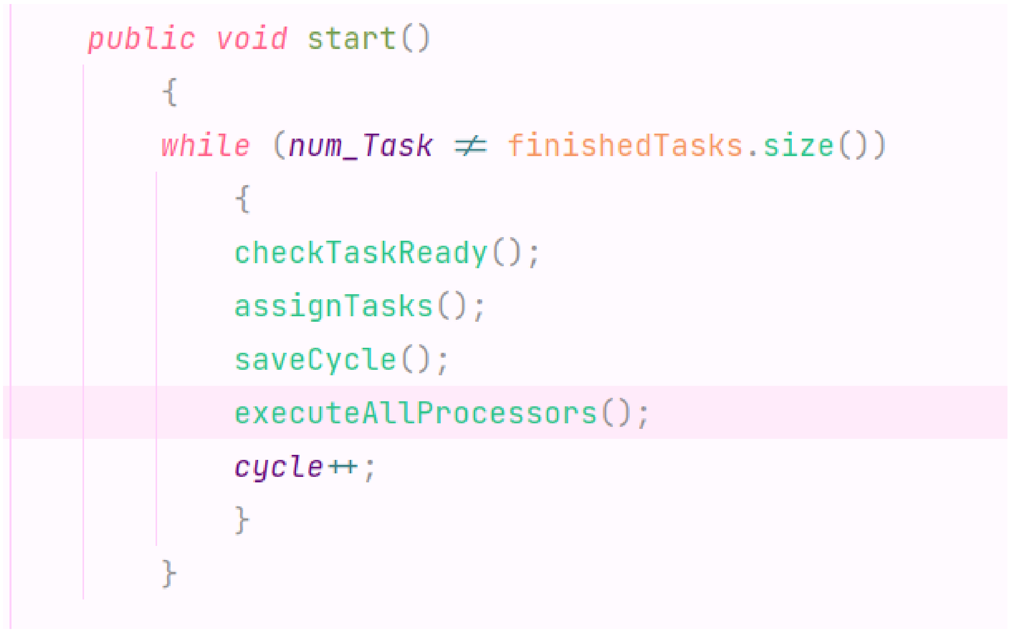
This class contains the simulation of processors scheduling, it determines how tasks are entered and how they are displayed.



```java
public class ProcessorSimulator extends Simulator
    {
    private static int cycle = 0;
    private static int num_Processor = 0;
    private static int num_Task = 0;
    private static final List<Processor> processors = new ArrayList<>();
    private static final List<Task> createdTask = new ArrayList<>();
    private static final List<Task> waitingTask = new ArrayList<>();
    private static final List<Task> finishedTasks = new ArrayList<>();
    private static final StringBuilder outputString = new StringBuilder();

    protected void input()
        {...}

    protected void output()
        {...}

    protected void run()
        {...}

    private static class Scheduler
        {...}

    private static class Processor
        {...}

    private static class Task
        {...}
```

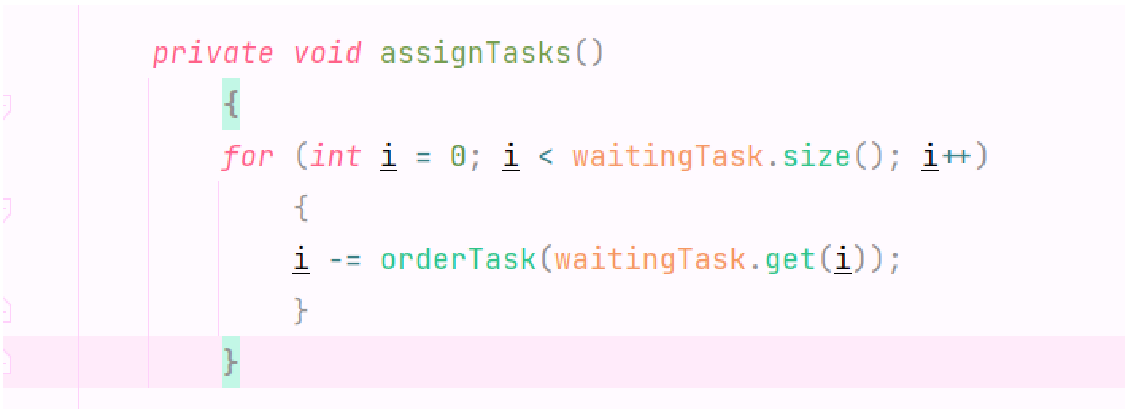
Scheduler:

This class orders the task and determines the way tasks are entered to the processors, so the most important class in the project.



```
public void start()
    {
    while (num_Task ≠ finishedTasks.size())
        {
        checkTaskReady();
        assignTasks();
        saveCycle();
        executeAllProcessors();
        cycle++;
        }
    }
```

Here the class start receiving tasks.



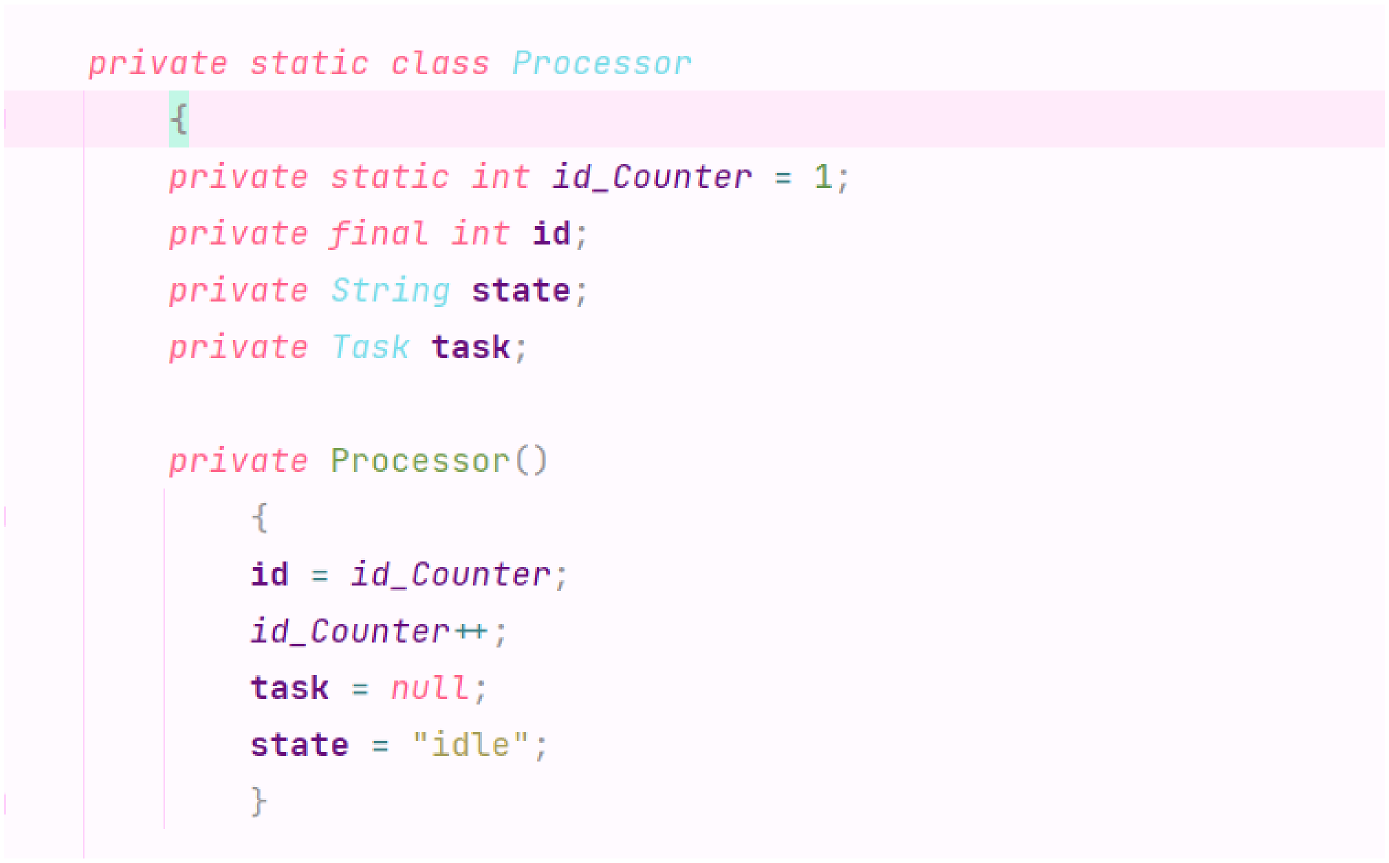
```
private void assignTasks()
    {
    for (int i = 0; i < waitingTask.size(); i++)
        {
        i -= orderTask(waitingTask.get(i));
        }
    }
```

Here it starts sending tasks to the processor and order it inside.

Note: that in case a processor receiving any task, 1 is returned to indicate that a task was assigned successfully, otherwise 0 is returned.

Processor:

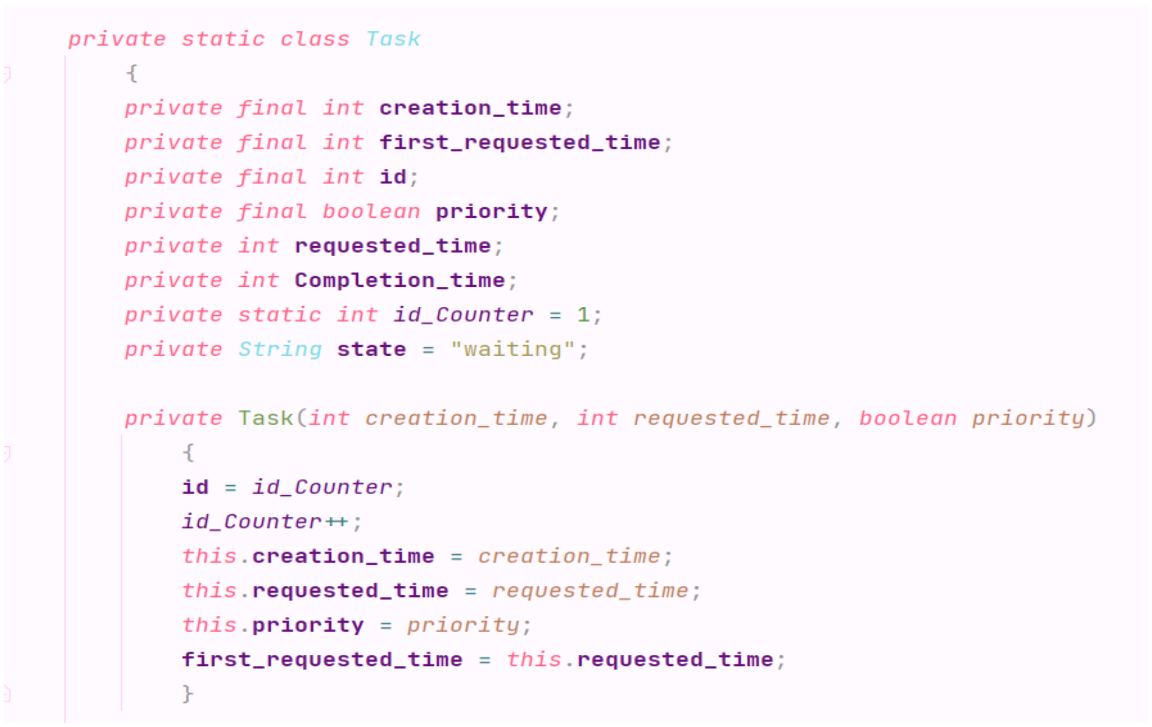
This class represents the processor and its attributes such as id, state.



```
private static class Processor
    {
    private static int id_Counter = 1;
    private final int id;
    private String state;
    private Task task;

    private Processor()
        {
        id = id_Counter;
        id_Counter++;
        task = null;
        state = "idle";
        }
```

Task:

This class represents the Task and its attributes such as id, state and creation time etc.

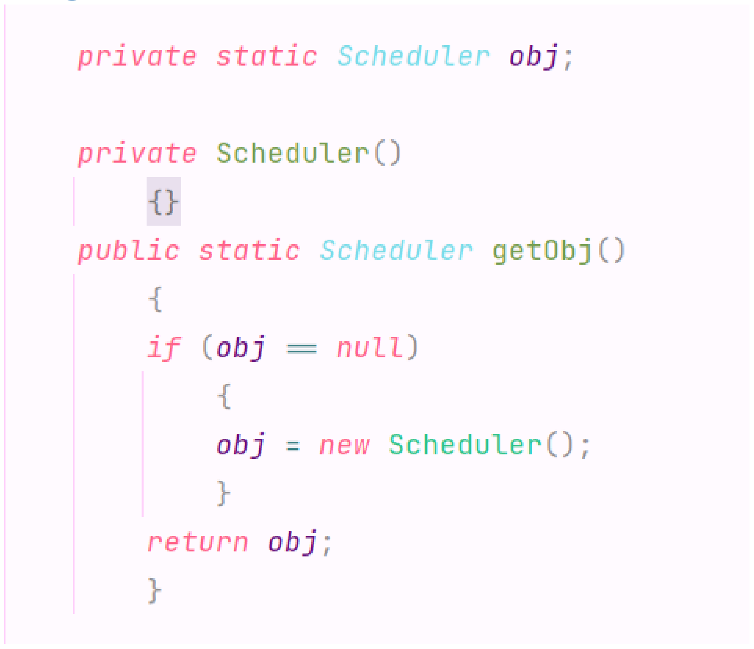


```
private static class Task
    {
    private final int creation_time;
    private final int first_requested_time;
    private final int id;
    private final boolean priority;
    private int requested_time;
    private int Completion_time;
    private static int id_Counter = 1;
    private String state = "waiting";

    private Task(int creation_time, int requested_time, boolean priority)
        {
        id = id_Counter;
        id_Counter++;
        this.creation_time = creation_time;
        this.requested_time = requested_time;
        this.priority = priority;
        first_requested_time = this.requested_time;
        }
```

Design Pattern

I used Singleton as a design pattern in my project:

Singleton



```
    private static Scheduler obj;

    private Scheduler()
        {}
    public static Scheduler getObj()
        {
        if (obj == null)
            {
            obj = new Scheduler();
            }
        return obj;
        }
```

Clean Code

In my project I tried to apply the rules of clean code such as:

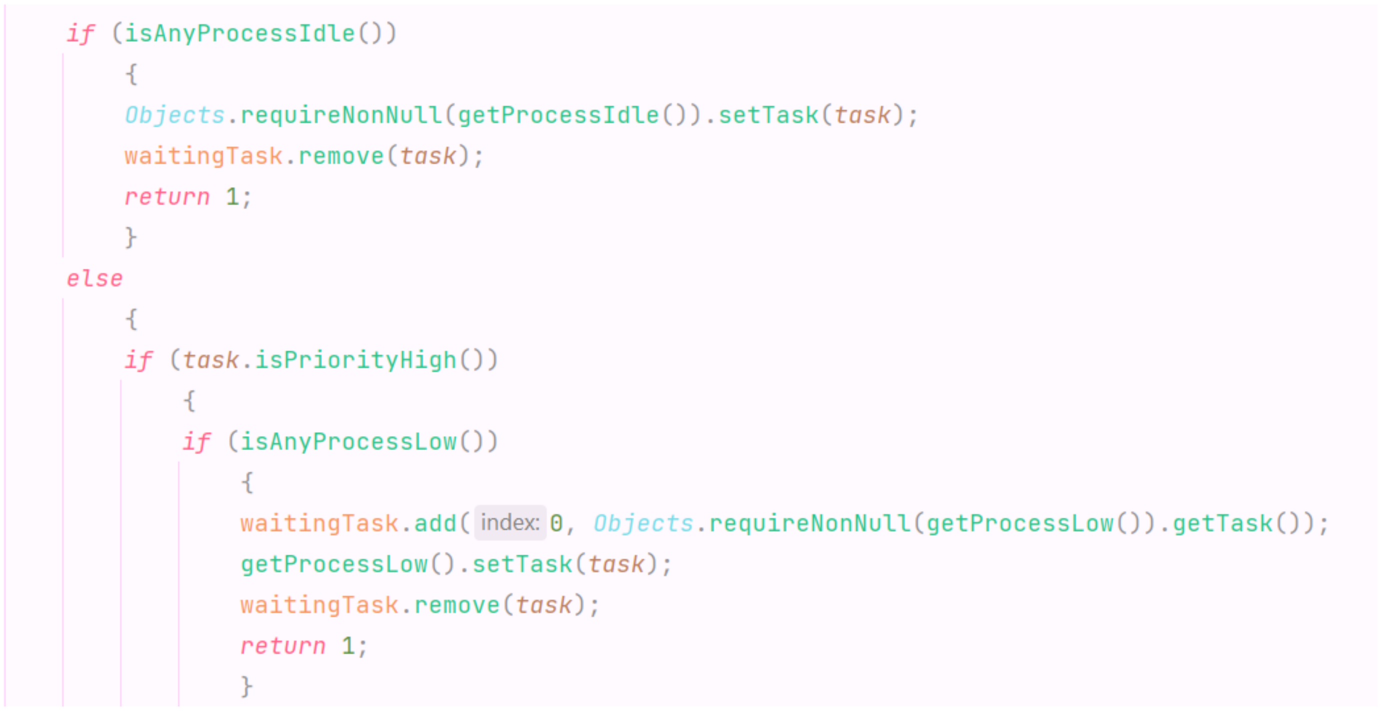
- Readability
- Meaningful Names
- Handling exceptions
- Eliminating useless comments

Tie Breaking

Low vs. high:

I have ordered the tasks so that the tasks with higher priority are entered the processor before the tasks that have lower priority.

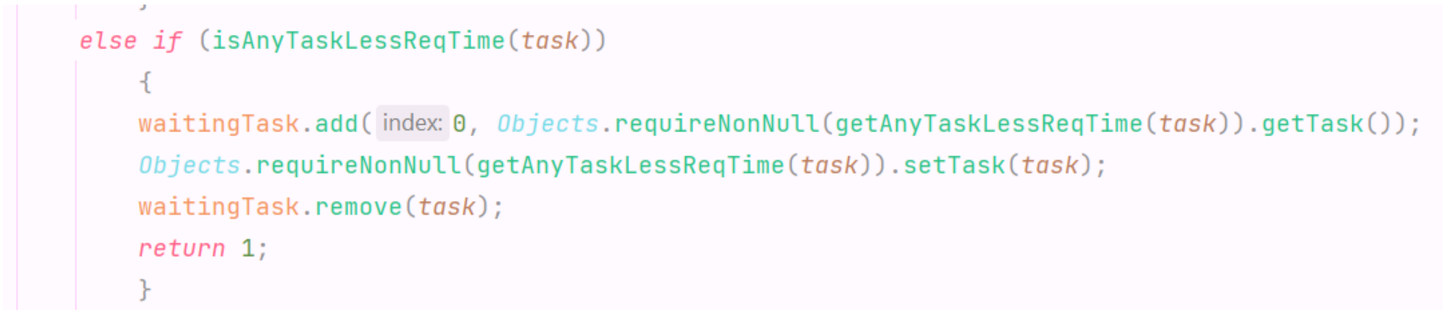
Low priority tasks are interrupted if there are high priority tasks waiting.



```
if (isAnyProcessIdle())
    {
    Objects.requireNonNull(getProcessIdle()).setTask(task);
    waitingTask.remove(task);
    return 1;
    }
else
    {
    if (task.isPriorityHigh())
        {
        if (isAnyProcessLow())
            {
            waitingTask.add( index: 0, Objects.requireNonNull(getProcessLow()).getTask());
            getProcessLow().setTask(task);
            waitingTask.remove(task);
            return 1;
            }
```

High vs. high:

In the case of tying two high-priority tasks, we compare them based on requested time (descendingly), then Based on creation time (ascendingly).

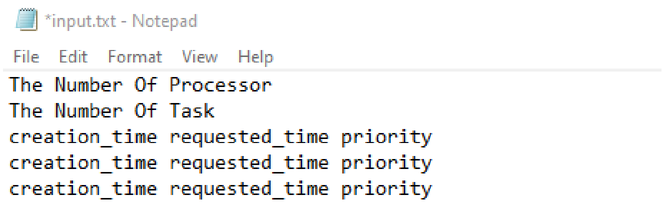


```
else if (isAnyTaskLessReqTime(task))
    {
    waitingTask.add( index: 0, Objects.requireNonNull(getAnyTaskLessReqTime(task)).getTask());
    Objects.requireNonNull(getAnyTaskLessReqTime(task)).setTask(task);
    waitingTask.remove(task);
    return 1;
    }
```

Input and Output Sample

I chose the text file for input and output.

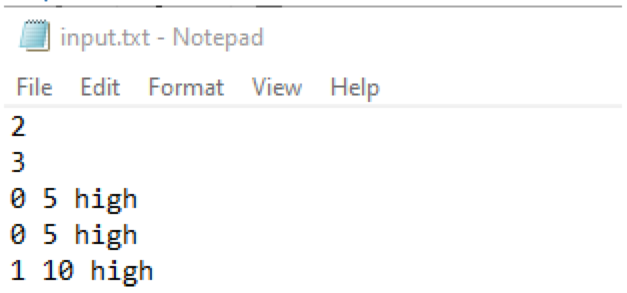
This is the format of my input:



```
The Number Of Processor
The Number Of Task
creation_time requested_time priority
creation_time requested_time priority
creation_time requested_time priority
```

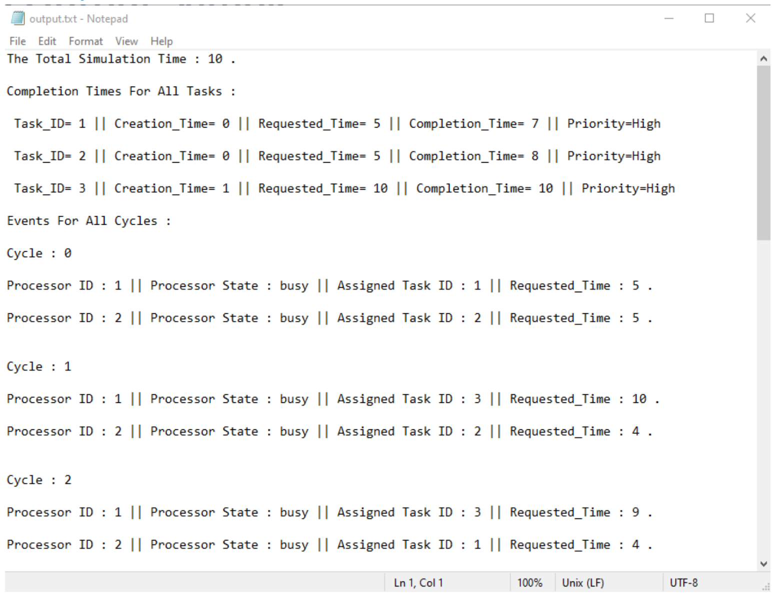
Ex:

Input:



```
2
3
0 5 high
0 5 high
1 10 high
```

Output:



```
The Total Simulation Time : 10 .

Completion Times For All Tasks :

 Task_ID= 1 || Creation_Time= 0 || Requested_Time= 5 || Completion_Time= 7 || Priority=High

 Task_ID= 2 || Creation_Time= 0 || Requested_Time= 5 || Completion_Time= 8 || Priority=High

 Task_ID= 3 || Creation_Time= 1 || Requested_Time= 10 || Completion_Time= 10 || Priority=High

Events For All Cycles :

Cycle : 0

Processor ID : 1 || Processor State : busy || Assigned Task ID : 1 || Requested_Time : 5 .

Processor ID : 2 || Processor State : busy || Assigned Task ID : 2 || Requested_Time : 5 .


Cycle : 1

Processor ID : 1 || Processor State : busy || Assigned Task ID : 3 || Requested_Time : 10 .

Processor ID : 2 || Processor State : busy || Assigned Task ID : 2 || Requested_Time : 4 .


Cycle : 2

Processor ID : 1 || Processor State : busy || Assigned Task ID : 3 || Requested_Time : 9 .

Processor ID : 2 || Processor State : busy || Assigned Task ID : 1 || Requested_Time : 4 .
```