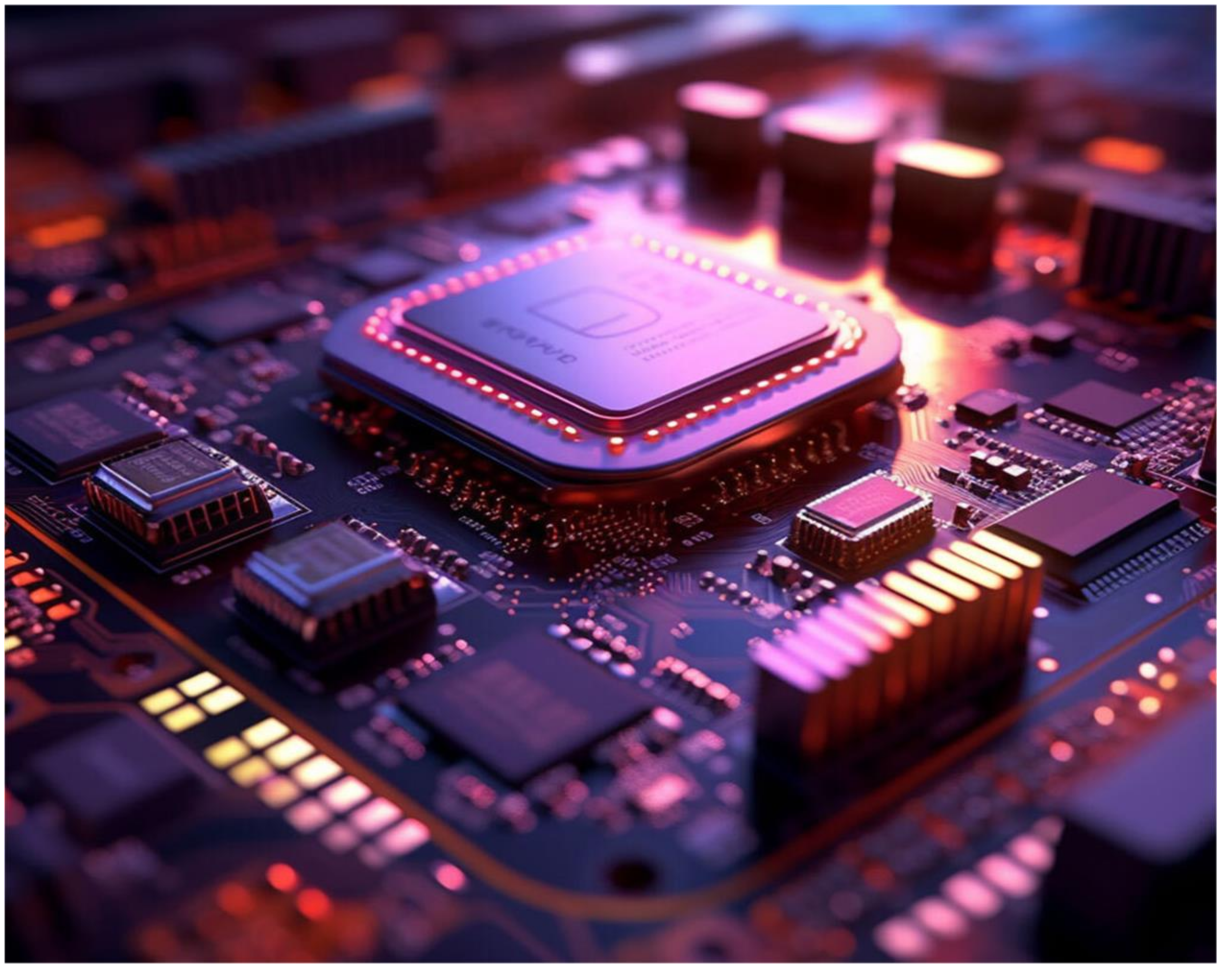
Processor Execution Simulation

Simulating task execution across multiple processors

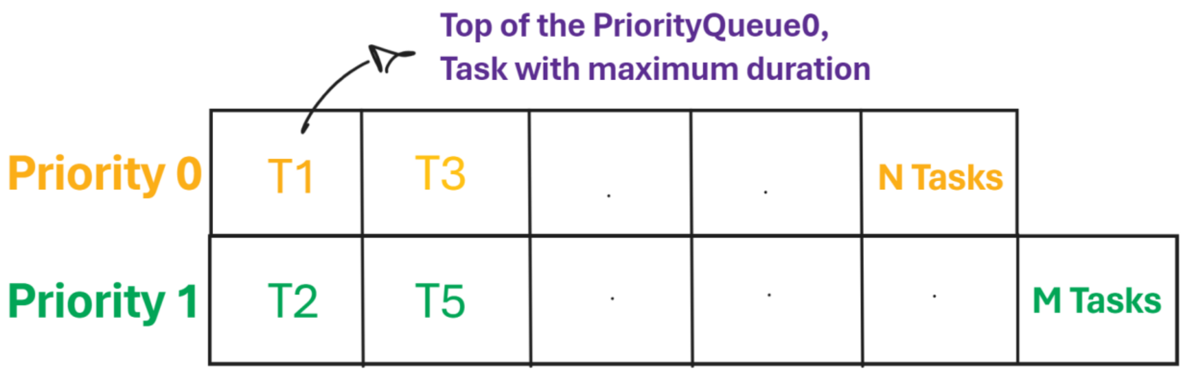


- **Outlines:**
  - **Algorithm explanation & Execution flow.**
  - **Software Design with UML diagram.**
  - **Other details.**

Algorithm Explanation & Execution Flow

- **Schedulers:**

  Each **Scheduler** object has a **PriorityQueue<**Task**>**, where “**Task**” is an object that has a “**duration**” to be completed (in cycles), and a “**priority**” (0 or 1 in our case), and the top (Head) of each priority queue will be the Task with the **maximum duration** in it.
  So, we will create an **N-Schedulers** (N is the number of priorities, which is **2** in our case) and put them inside a **List<Scheduler>**, where the first item in the list contains the Tasks with the **lower** priority (priority 0), and the last item contains the tasks with the **upper** priority.

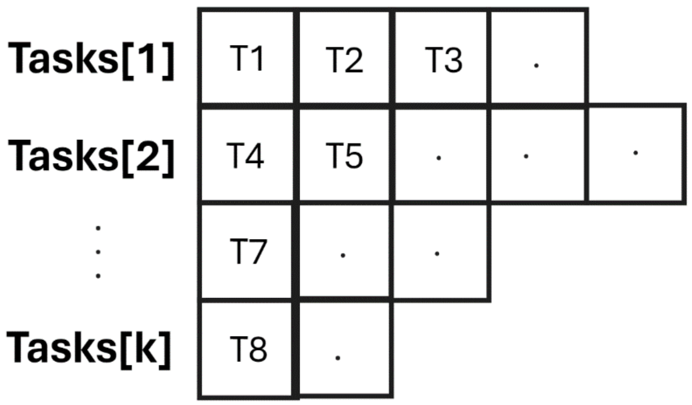


- **Tasks preparing:**

  Which is creating an **Array of List<Task>**, where the **i-th** list in the array will be the **list of tasks** that are **created at the i-th clock cycle**.

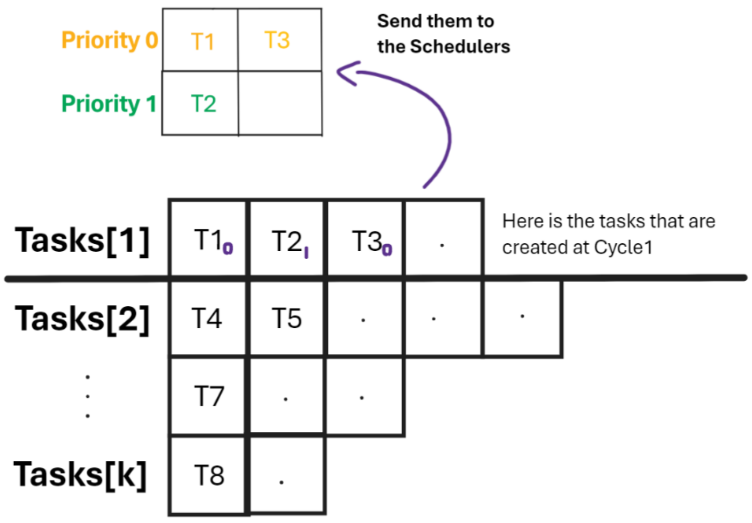
  So, **Tasks[1]** is the list of tasks that are created at the first clock cycle, **Tasks[2]** is the list of tasks that are created at the second clock cycle, and so on.

  The order of tasks that belong to the same list (i.e. have the same creation time) is not important, all we care about is that they are created at the i-th clock cycle.



- **Cycle processing:**
  We will start processing **each cycle independently**, at the **i-th cycle** we will do the following:

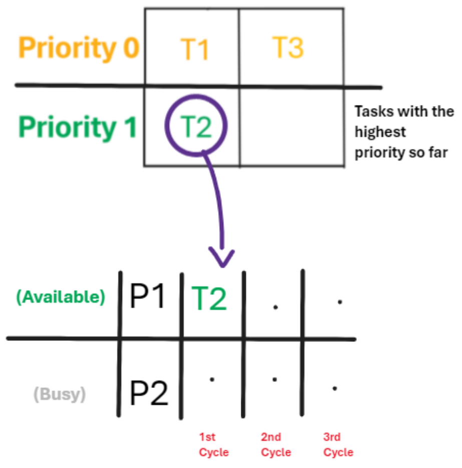
  1. **Add** the tasks that are **created at the i-th cycle** (if any such tasks exist) to the **schedulers**. Each task will be sent to the appropriate Scheduler (PriorityQueue) based on its **priority** (e.g., if we are at the first clock cycle and tasks T1, T2 and T3 are created, then **T1** with priority **0** will be sent to the first Scheduler in the list as shown, **T2** with priority **1** to the second one, and so on).

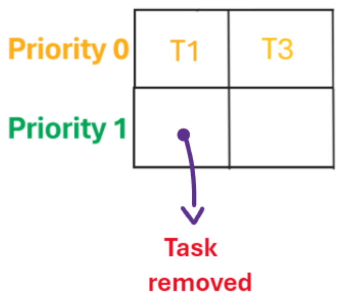

Priority 0
T1
T3
Priority 1
T2
Send them to the Schedulers
Tasks[1]
T1₀
T2₁
T3₀
Here is the tasks that are created at Cycle1
Tasks[2]
T4
T5
T7
Tasks[k]
T8

  2. **Assign tasks** to each of the **available processors** (assuming that we can't assign a task to processor that is already busy with some other task).

     We will assign tasks to processors using the **schedulers.** This will be done by iterating through the **List<Scheduler>** (that we have so far) starting from the last Scheduler to the First Scheduler (from the **highest priority to the lowest priority**). If there are any tasks in the Scheduler that we are currently iterating over, we will **remove** this task from this Scheduler, **assign it to one of the available processors**, and then continue with the next available processor, and so on.

     However, if there are no tasks in the current Scheduler, we will move to the scheduler that has less priority than the current one.

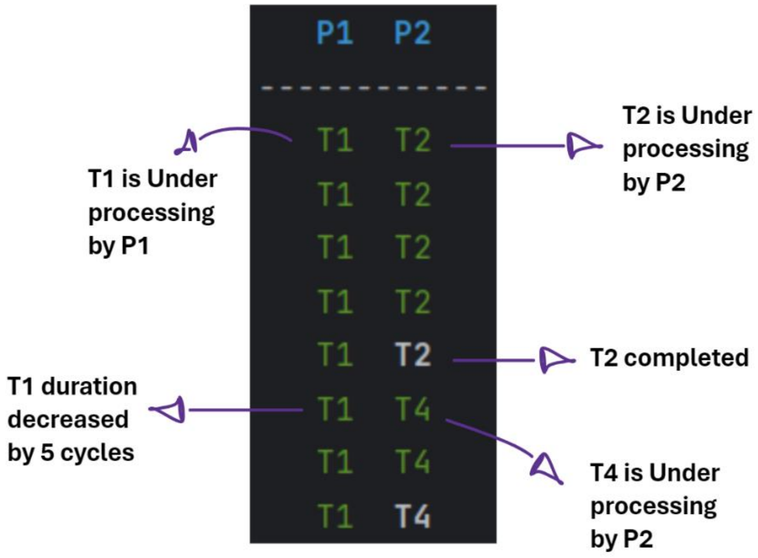




- **Processors execution:**

  Each processor will **continue** to perform its current task during the cycles.

  In each cycle, the **duration** (number of cycles needed to complete the task) of the current task will be **decreased by 1** cycle (**duration = duration - 1**), if the current task has **completed** its duration (duration = 0), **another task** will be assigned to the processor.



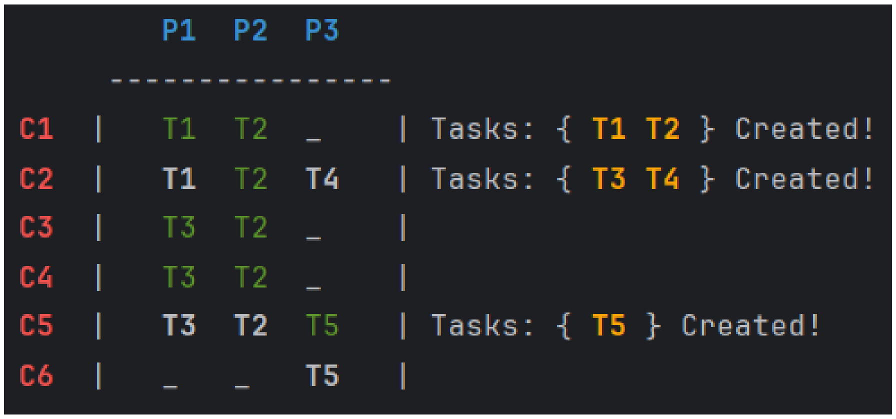
```
P1   P2
------------
T1   T2
T1   T2
T1   T2
T1   T2
T1   T2
T1   T4
T1   T4
T1   T4
```

T1 is Under processing by P1
T1 duration decreased by 5 cycles
T2 is Under processing by P2
T2 completed
T4 is Under processing by P2

- **Report:**

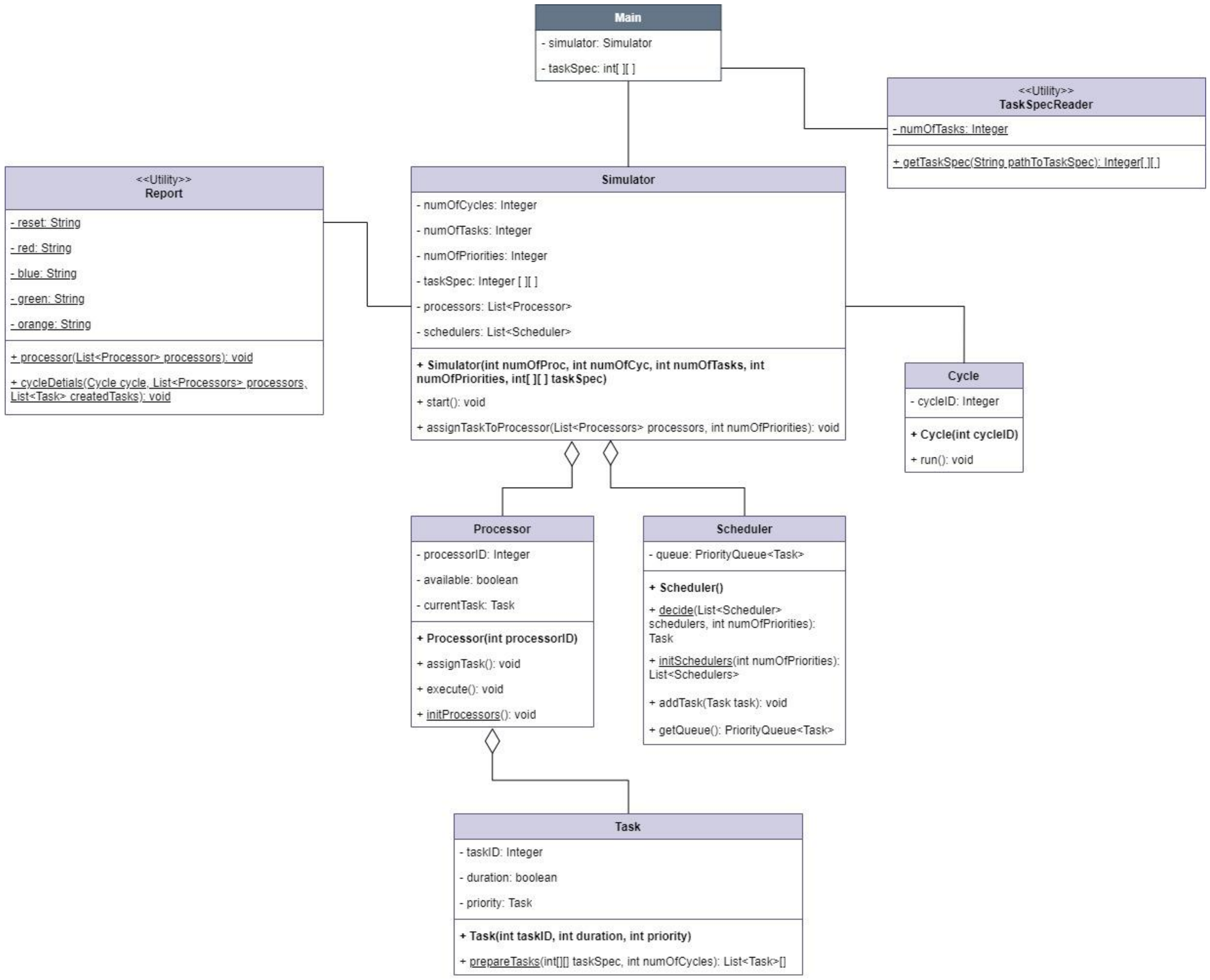
  The report will be printed in the console, explaining how the processors will execute the tasks. I've used the **ANSI** escape codes to control the colors printed in the console to clarify the process, Where:

Blue	: Color of the **Processor**
Red	: Color of the **Cycle**
Orange	: Color of the **Created Task**
Green	: Color of the **Task being processed**
White	: Color of the **Completed Task**



```
        P1   P2   P3
      ----------------
C1  |   T1   T2   _    | Tasks: { T1 T2 } Created!
C2  |   T1   T2   T4   | Tasks: { T3 T4 } Created!
C3  |   T3   T2   _    |
C4  |   T3   T2   _    |
C5  |   T3   T2   T5   | Tasks: { T5 } Created!
C6  |   _    _    T5   |
```

Software design with UML Diagram


Main
- simulator: Simulator
- taskSpec: int[ ][ ]
<<Utility>>
TaskSpecReader
- numOfTasks: Integer
+ getTaskSpec(String pathToTaskSpec): Integer[ ][ ]
<<Utility>>
Report
- reset: String
- red: String
- blue: String
- green: String
- orange: String
+ processor(List<Processor> processors): void
+ cycleDetials(Cycle cycle, List<Processors> processors, List<Task> createdTasks): void
Simulator
- numOfCycles: Integer
- numOfTasks: Integer
- numOfPriorities: Integer
- taskSpec: Integer [ ][ ]
- processors: List<Processor>
- schedulers: List<Scheduler>
+ Simulator(int numOfProc, int numOfCyc, int numOfTasks, int numOfPriorities, int[ ][ ] taskSpec)
+ start(): void
+ assignTaskToProcessor(List<Processors> processors, int numOfPriorities): void
Cycle
- cycleID: Integer
+ Cycle(int cycleID)
+ run(): void
Processor
- processorID: Integer
- available: boolean
- currentTask: Task
+ Processor(int processorID)
+ assignTask(): void
+ execute(): void
+ initProcessors(): void
Scheduler
- queue: PriorityQueue<Task>
+ Scheduler()
+ decide(List<Scheduler> schedulers, int numOfPriorities): Task
+ initSchedulers(int numOfPriorities): List<Schedulers>
+ addTask(Task task): void
+ getQueue(): PriorityQueue<Task>
Task
- taskID: Integer
- duration: boolean
- priority: Task
+ Task(int taskID, int duration, int priority)
+ prepareTasks(int[][] taskSpec, int numOfCycles): List<Task>[]

Software Design with UML Diagram

Let us review each class and discuss the main purpose for each one:

1. **Report class:**
   Which is a **Utility class** (i.e. all its attributes and methods are **static**) that will control the output in the console and will be responsible for giving the report about the processing.
   Will be **invoked** by the **Simulator** class to make the output for each cycle.

2. **Task class:**
   Instances of this class have three attributes:
   - **taskId**: uniquely identifies the tasks.
   - **duration**: time (in cycles) needed to complete the task.
   - **priority**: indicates the priority of the task (0 or 1 in our case).

   The class has an **Aggregation** relationship with the **Processor** class (Processor is the whole and Task is the part), each Processor instance maintains a reference to its current task, which is the task being executed at a specific clock cycle.

   Also, Task class includes a static method named "**prepareTasks**" which is responsible for preparing the tasks in an Array of lists, so that it becomes more efficient to know what are the tasks that have been created at some cycle.

3. **Processor class:**
   Instances of this class have the following attributes:
   - **processorID**: unique for each task (integer)
   - **available**: true if the object has no currentTask, false otherwise (Boolean)
   - **currentTask**: which is the task that is being executed at the current cycle (Task)

   Tasks can be assigned to processors using the method "**assignTask**", so we can track the current task being executed by the processor.

   This class has an **Aggregation** relationship with the **Simulator** class.

4. **Scheduler class:**
   Instances of this class have a **PriorityQueue<**Task**>**, and they are responsible for deciding which task should be sent to the available processors based on their priority (scheduler with higher priority will decide first).
   This class has an **Aggregation** relationship with the **Simulator** class.

5. **Cycle class:**
   Instances of this class are responsible for **managing the time** of the clock cycles, and each cycle will be initiated in the Simulator class (this class has an **Association** relationship with the **Simulator** class).

6. **Simulator class:**
   The class that coordinates the simulation process and utilizes other classes (**Processor**, **Scheduler**, **Cycle**) to start the processor simulation. Has an Aggregation relationship with classes (Processor, Scheduler, Cycle) and **uses** the **Report** (Utility) class.

7. **Main class:**
   This class takes the input from the user and uses the Simulator class to achieve the simulation process.

8. **TaskSpecReader:**
   This utility class reads the input from a file (given by the file path), and it will be used to read the Task Specifications from a .txt file.
   The Main class will use this utility class to get the task specifications.

Other details

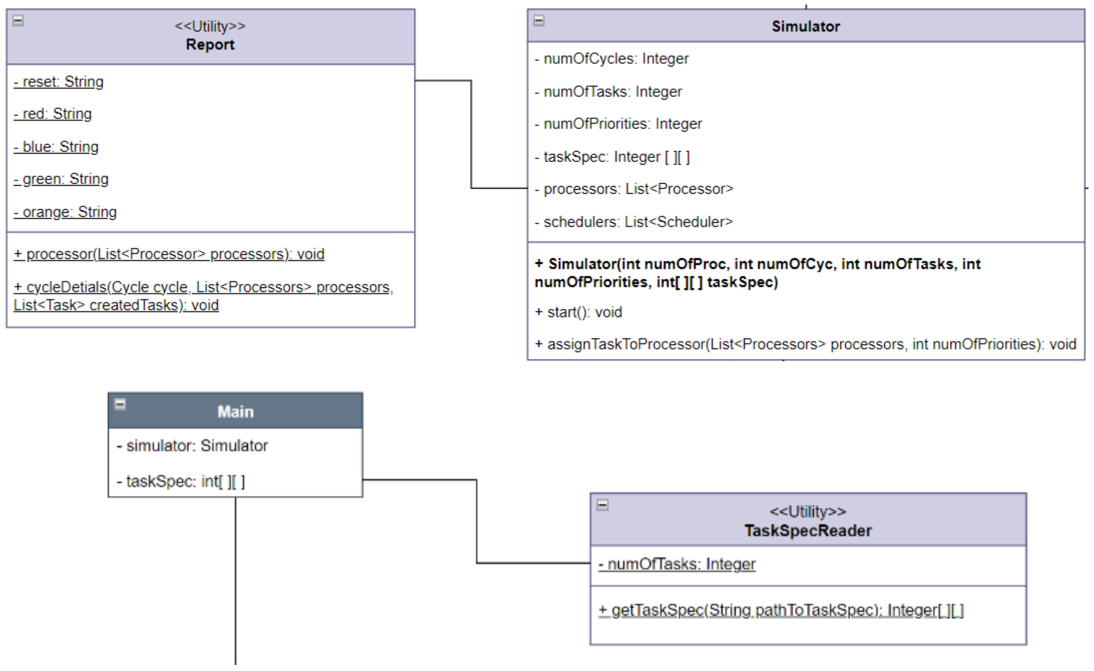
- ### Reusability & Efficiency:

  I've designed my code to be as reusable as possible and simplify the simulation process, making it more maintainable and reusable. For example, the **number of the priorities** (which is 2 in our case, either 0 or 1) can easily be adjusted just by telling the simulator that we have **N** priorities instead **2**, and the simulator will work smoothly with the new priorities.

  In terms of efficiency, the PriorityQueue is highly efficient. It can push and delete in **O(log$_2$(N))** time complexity and retrieve the top element (which is the task with the maximum duration in our case) in **O(1)** time complexity, which is very efficient.

- ### Utility classes:

  Classes "**Report**" and "**TaskSpecReader**" are **Utility Classes** (all the methods and attributes are static in these classes), and the classes "**Main**" and "**Simulator**" utilize these two utility classes.



<<Utility>>
Report
- reset: String
- red: String
- blue: String
- green: String
- orange: String
+ processor(List<Processor> processors): void
+ cycleDetials(Cycle cycle, List<Processors> processors, List<Task> createdTasks): void
Simulator
- numOfCycles: Integer
- numOfTasks: Integer
- numOfPriorities: Integer
- taskSpec: Integer [ ][ ]
- processors: List<Processor>
- schedulers: List<Scheduler>
+ Simulator(int numOfProc, int numOfCyc, int numOfTasks, int numOfPriorities, int[ ][ ] taskSpec)
+ start(): void
+ assignTaskToProcessor(List<Processors> processors, int numOfPriorities): void
Main
- simulator: Simulator
- taskSpec: int[ ][ ]
<<Utility>>
TaskSpecReader
- numOfTasks: Integer
+ getTaskSpec(String pathToTaskSpec): Integer[ ][ ]

Thanks

Ahmad Nabeel Jaber