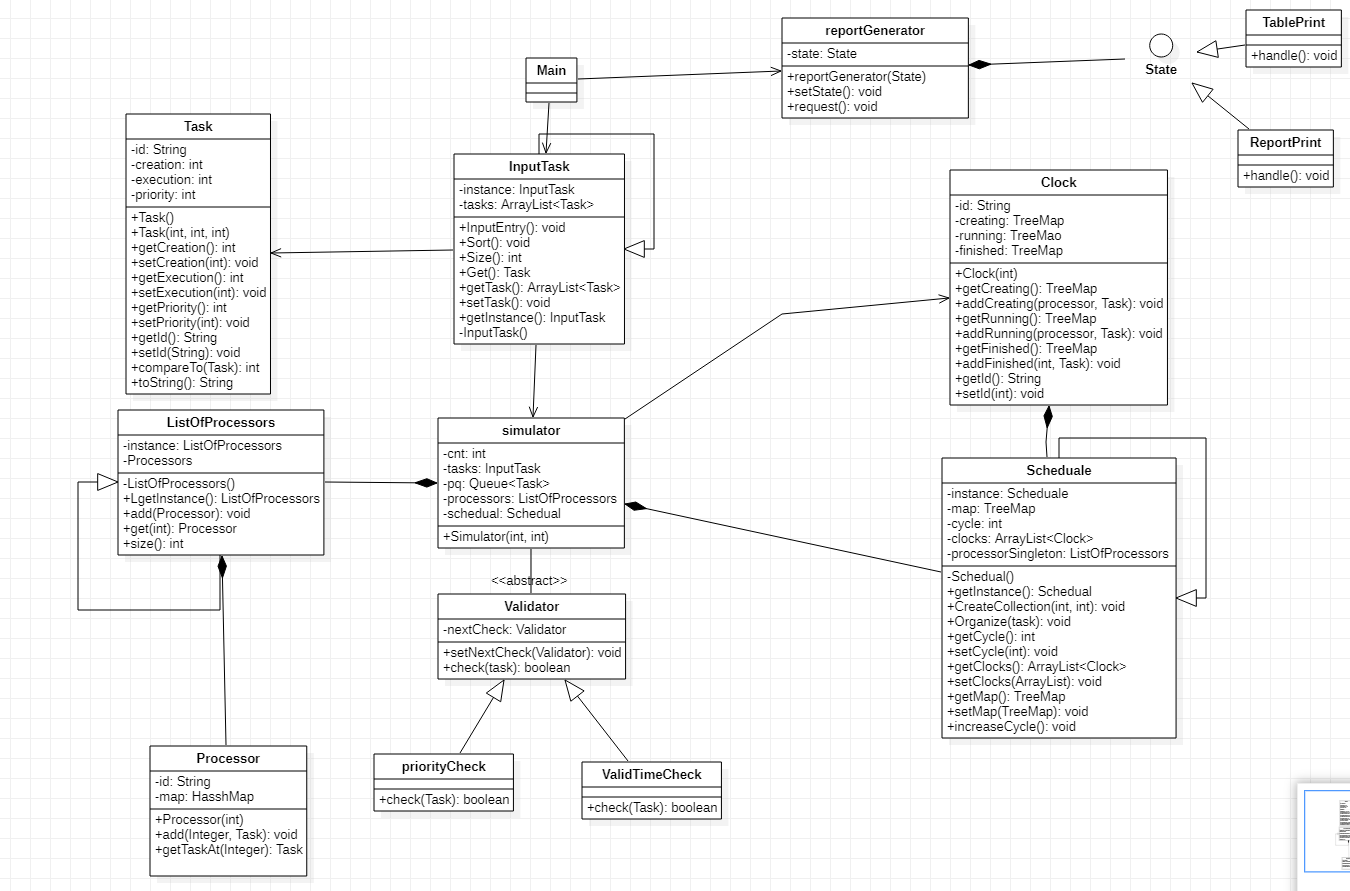
**Processor Execution Simulation Assignment**

Student : Diya Momani

* Overview:

In this assignment we are required to build a simulator that simulates processor execution for processes. I build nine main classes: main, InputTask, Simulator, Clock, Task, Schedule, ListOfProcessors, validator and Processor . And I used collections such as Tree Map, ArrayList, PriorityQueue.

* Classes:



1. Main:

This class is received the inputs: number of processors, number of clocks and path of text file.

1. InputTask:

This class open the text file and stores the tasks from text file .

1. Simulator:

This class contain five variables:

1. Cnt : index over the tasks array.
2. Array of tasks.
3. Pq: priorityQueue to store the tasks based on the priority and creation time.
4. Processor: to invoke the ListOfProcessors singleton instance.
5. Schedule: to invoke the Schedule singleton instance.
6. Clock:

This class contain the id of the cycle and three tree maps the key of them is integer represent the processor that stores the task and the value is task:

1. Created :which represent the tasks that created in this cycle.
2. Running : which represent the tasks that is running in this cycle.
3. Finished : which represent the tasks that finished in this cycle.
4. Task:

Represent the task and contain id, creation , execution and priority. This class override compareTo method that sort them in priorityQueue based on priority and creation.

1. Schedule:

This is a singleton design pattern and it work on take the tasks from queue and add them to processor in certain cycle, I used tree map , its key is integer represent the cycle that has empty processors, and the value is linked list of processors in this cycle. When the task is sent to organizer method in Scheduler the first processors (the first element in the linked list of the first key) will be removed and the task will be put in this processor at this cycle , and will add the new empty space by take the place that the task has put in and increase it by the execution time .

Note : this whole process toke O(logn), as the deletion and addition into tree map toke O(logn), and the deletion and addition into the first of linked list toke O(1)

1. ListOfProcessors:

This is a singleton design pattern that has list of processors and I used it to print the table.

1. Processor:

This class has id and hash map of tasks

1. reportGenerator: this is a state design pattern and associate it with state interface and the state has two classes that implement it : TablePrint and ReportPrint
2. Validator: this is chain of responsibility design pattern and it has two childs priorityCheck and validTimeCheck. It used to ensure that tasks input is always valid.

* Design Patterns:

I used three design patterns in five classes:

1. Singleton: I used it in ListOfProcessors, InputTasks and Schedule classes , and I used it to ensure that a class has only one instance, while providing a global point of access to this instance for the entire system.
2. Chain Of Responsibility: I used it in Validator class to ensure that tasks input is always valid and it makes the design and code more improvable and flexible.
3. State : I used it in resportGenerator to handle state-dependent behavior in a clean and organized manner, making it easier to maintain and modify the code.

* SOLID principles:

1. Single Responsibility Principle (SRP):I apply it by chain of responsibility and state design patterns and every class in my code do specific thing and solve specific problem ,even the name of classes and functions are clear and you can know what classes are and what functions do from their names, and do what expected to do , thus ,I increase cohesion by applying this principle.
2. Open/ Closed principle :I apply it by chain of responsibility and state design patterns and the classes are open for extension, and closed for modifications.
3. Interface Segregation principle: I apply it by chain of responsibility and state design patterns and I avoid making general interface contains all methods, every interface is specialized with one subject and contains related functions, thus ,I increase cohesion by applying this principle.
4. Liskov Substitution Principle (LSP): I apply it by state design pattern and the objects of a derived class in my code are able to be used wherever objects of the base class are used.