DOCUMENTATION

ASSIGNMENT *ASSIGNMENT\_02*

STUDENT NAME: Baraian Tudor

GROUP: 30424

# Assignment Objective

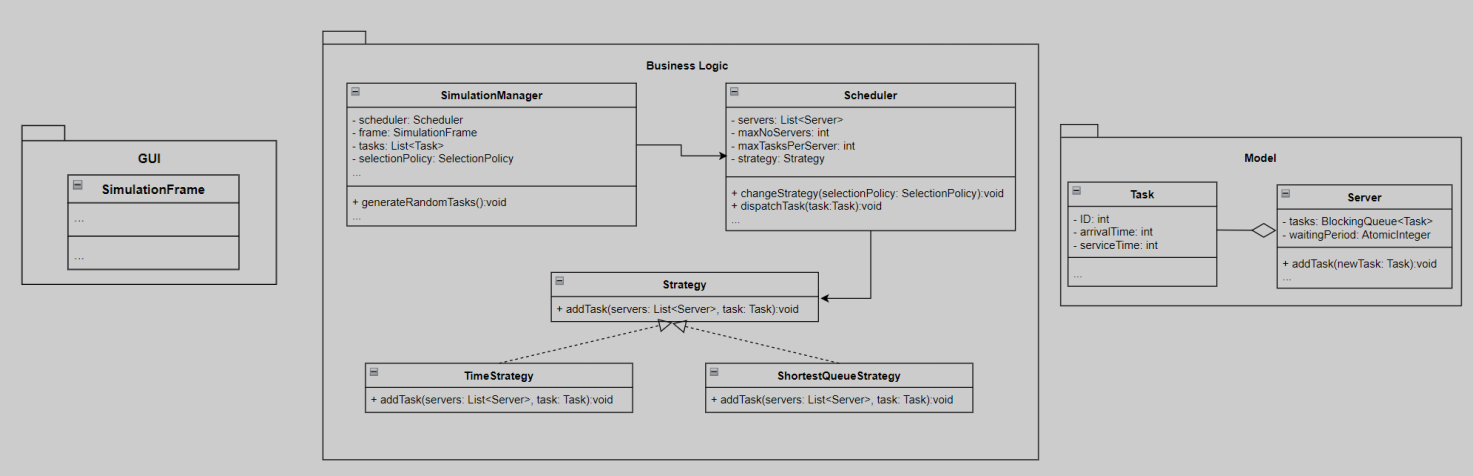
1. *-The main Objective: Create a Thread managing application with a UI.(not console)*
2. *-sub-objectives: .create a model component of the project, which is responsible for implementing the data/objects used for the logical part of the application, in this case: Task, Server, Task Comparator*
3. *.Create a view component. This consists of the Simulation Frame class that is responsible for creating all the views and managing the displayed content on the UI and the written data in the text file.*

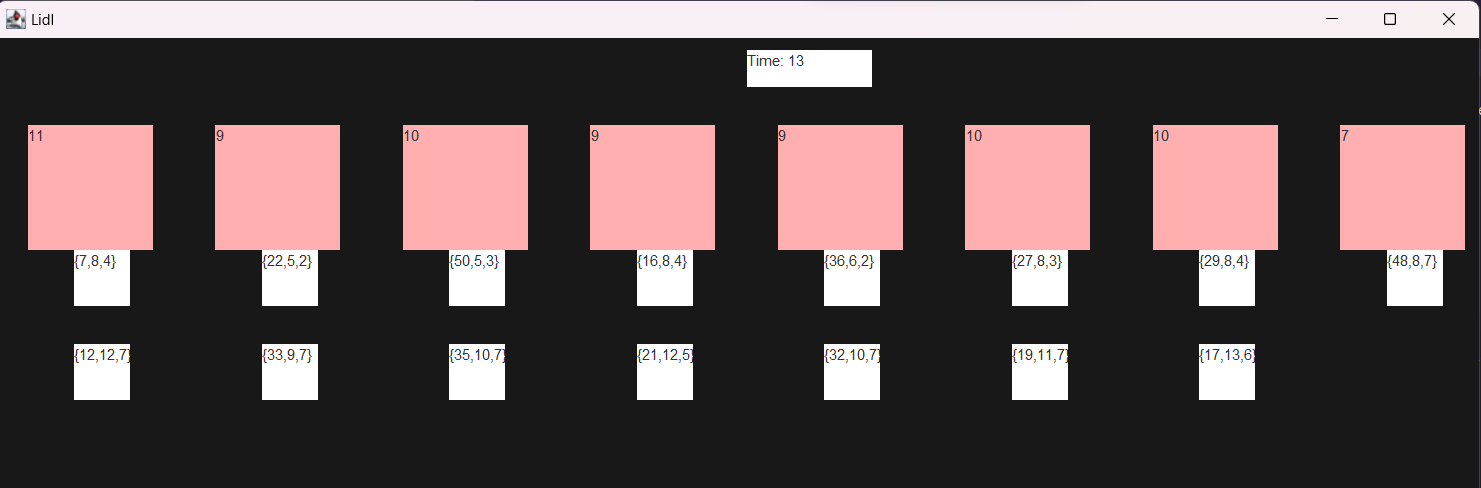
# Problem Analysis, Modeling, Scenarios, Use Cases

*The functional requirements should be presented together with the use cases (use case diagrams and use case description). The use cases’ descriptions can be done as a flow-chart or as a list containing the execution steps of each use case.*

# Design

*I used Atomic Integer, Blocking Queue for modifying(from more thread contexts) the data in the server objects*





# Implementation

*Each class will be described (fields, important methods). Also, the implementation of the graphical user interface will be described.*

*The* ***Server*** *class:*

*-core components of this class are:*

*\*the tasks, that are kept in a blocking queue, so that one cannot cancel a modify operation on the queue, as well as an atomic integer keeping the time remaining for the queue to be empty.*

*The constructor starts a new thread that runs until the main thread(the one assigned to the Simulation Frame) is finished. The running method keeps updating the remaining time for the current task, and when it is done, it removes it from the blocking queue.*

*There is one more significant method that adds the task to the queue. It adds to the total waiting time variable, either how much time the queue has left until the end of the program time(if the tasks won’t finish by then), or the remaining time it has before it reaches the new task.*

*The* ***Task*** *class:*

*-in this class, there are kept the id, when the task will arrive to the server pool, and the time that will take for the task to complete once it is designated to a server.*

*The* ***Task Comparator*** *class:*

*-the tasks are sorted using the task comparator, which takes tasks with smaller arrival time before those with higher arrival times.*

*The* ***Simulation Frame*** *class:*

*-Holds the frames in which the application is displayed. Keeps a reference to the scheduler object created in the simulation manager. Has a function update that is called every time one second passes, a file object and a file writer that are used to update the content of a file*

*There are some text areas created, that will represent the time elapsed/average waiting time, the servers, and the tasks designated to those servers.*

*The* ***Scheduler*** *class:*

*-this class contains references to the servers(creates them), changes the strategy, and delivers the task to the desired strategy*

*The* ***Selection Policy*** *class:*

*-it is an enum that has only two elements associated to the strategy the user will pick:*

*\*Shortest Queue, and Shortest Time*

*The* ***Shortest Queue Strategy*** *class:*

*-implements the Strategy interface. It chooses the server that has the lowest number of elements, then adds the designated task to it.*

*The* ***Simulation Manager*** *class:*

*-this class contains the main method, creates the Simulation Frame Object, and when the constraints of the application are set(by the UI component), it starts a thread, in which the tasks are distributed to the scheduler, and the frame gets updated. It generates the application tasks randomly.*

*The* ***Strategy*** *class:*

*-it is an interface that will be used in choosing the right strategy(shortest queue or Time)*

*-has one method(add Task)*

*The* ***Time Strategy*** *class:*

*-implements the Strategy interface. It chooses the server that has the smallest waiting time, then adds the designated task to it.*

# Results

*The testing scenarios will be presented. If in the homework requirements it is specified to perform unit testing with JUnit, the results of the JUnit tests will be integrated in this section.*

# Conclusions

*In this assignment, I learned how to manage more threads from different contexts, while maintaining synchronized data, How to use enums in certain contexts for better readability, how to manage faster a file using the File Writer class, How to design an animation like display process on the UI.*

# Bibliography

1. [*www.tutorialspoint.com*](http://www.tutorialspoint.com)
2. http://docs.oracle.com/javase/tutorial/essential/concurrency/index.html
3. http://www.tutorialspoint.com/java/util/timer\_schedule\_period.html