**Documentation**

Assignment 2

Queue management simulation using threads

and synchronization mechanisms

Name: Știrb Călin-Alexandru

Group: 30422

**Contents:**

1. Assignment Objective............................................................................................3
2. Problem Analysis, Modeling, Scenariosm, Use Cases..........................................3
3. Design....................................................................................................................4
4. Implementation......................................................................................................8
5. Results....................................................................................................................12
6. Conclusions............................................................................................................13
7. Bibliography...........................................................................................................13
8. **Assignment Objective**

The main objective of this assignment is to work with multiple threads and synchronization methods in OOP. Thus we use the run() method from the interface Runnable in order to simulate queues of tasks inside of a server. In order for the threads to be secured we have to use *synchronus collections* such as: Vector and BlockingQueue.

**Sub-objectives:**

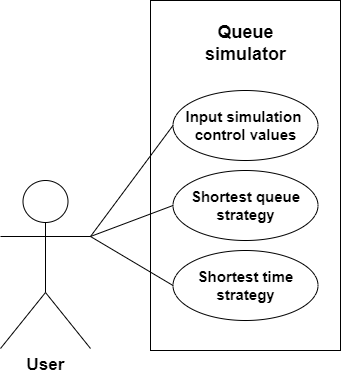
* Implementing the functional classes ( see the **design** section for more info):
  + Task: this class is the building block for this whole assignment as this class holds all the information needed to use the queues
  + Server: the server class is mostly used as an intermediary collection for the business logic of the project
  + SimulationManager: this class acts as the controller for the GUI
  + Gui: the visual element of the application
  + ShortestQueue: implements the run() method for the shortest queue implementation of the simulation
  + ShortestTime: implements the run() method for the shortest time implementation of the simulation

1. **Problem analysis, Modeling, Scenarios, Use cases**

For the functional part of the application, the user is required to enter a series of numbers into their respectable text fields, which represent the control values of the simulation:

A screenshot of a computer

Description automatically generated with medium confidenceThe screenshot to the left is the user-interactive part of the GUI, complete with a set of example values for a simulation to run.

 The use-case diagram illustrates the direct functionality of the application.

The primary actor for all the use cases is the ***user.***

Main success scenario for any of the illustrated use cases:

1. The user inserts a set of numbers into all of the text fields presented in the screenshot above.
2. The user selects which of the 2 methods they want the application to execute.
3. The user clicks the “start simulation” button.
4. The app performs the simulation and displays the info into the GUI’s text area.

Alternative sequence: Incorrect simulation data

* The user inserts anything other than numbers into the text field
* An error message will be displayed (currently only in the console)

**Functional requirements:**

* The app should allow the user to insert their preffered data for the simulation
* The app should allow the user to select their preffered execution strategy
* The app should display the output data in the GUI and also create a log.txt file containing the last simulation’s results

**Non-functional requirements:**

* The app should be easy to use by the user
* The output data should be formatted in such a way that it’s easy to understand

1. **Design**

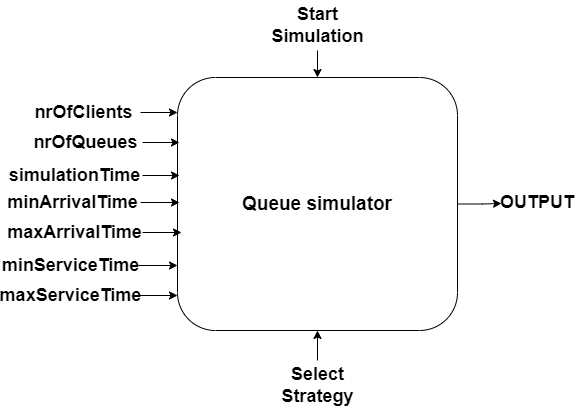
This section will provide a more in-depth view of the design choices taken during the development of the application.

* The most used data model implemented for this assignment is the Task class; this class is later used by the Server class in 2 different collections:

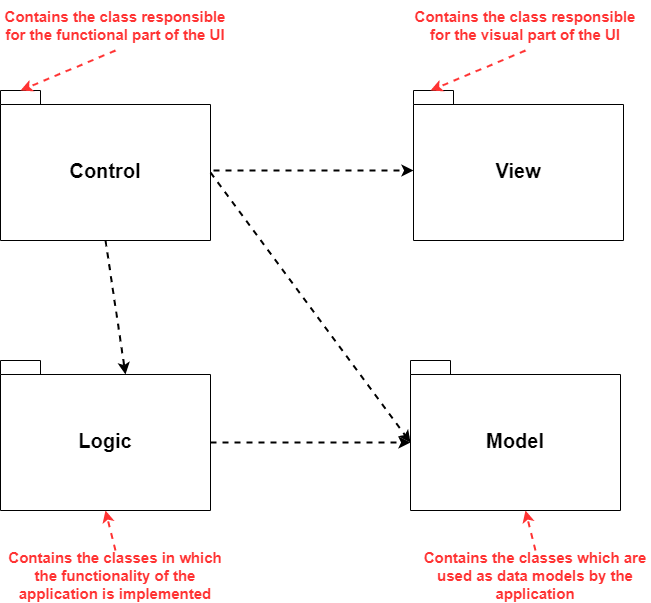
Vector<Task> tasks;

Vector<Queue<Task>> queues;

Level 1: Overall system design

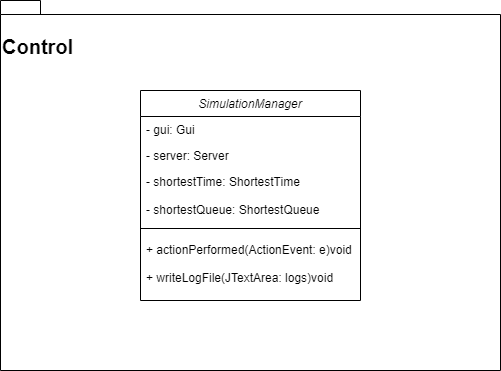


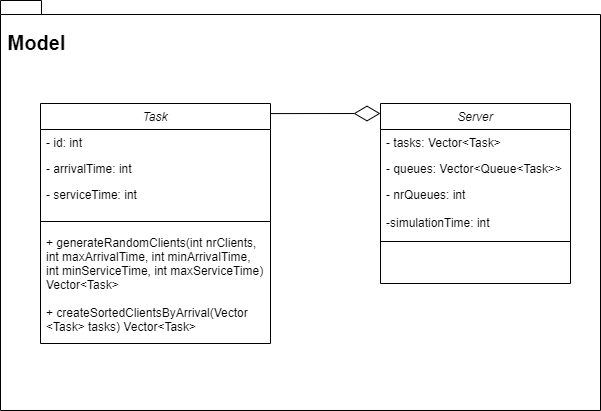
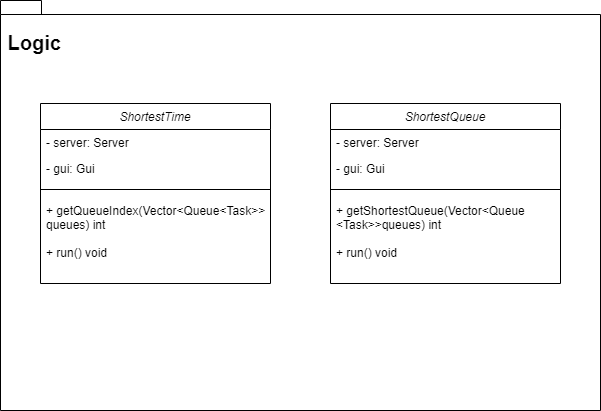
* In the diagram above are represented the main inputs and outputs of the app
* The series of inputs on the left represent the user’s own written input
* The input below is the drop down menu for the selection of the strategy
* The input above named “Start Simulation” is the button which the user clicks to start the simulation
* The output is the written logs into the GUI’s text area

Level2: Division into packages

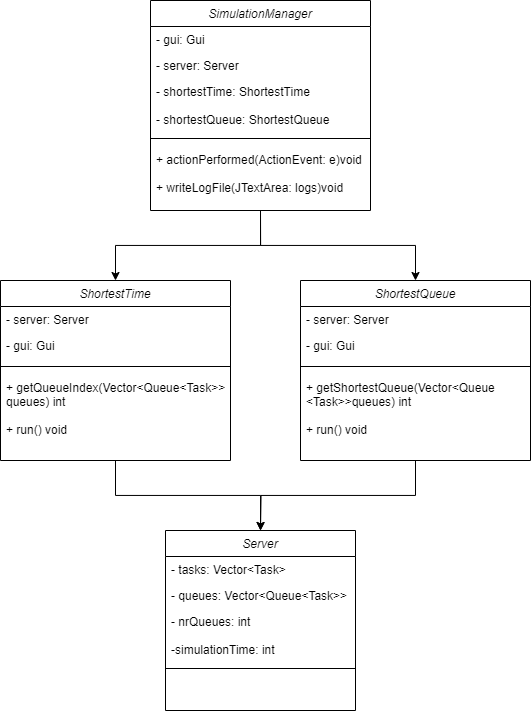
Level 3: Division into classes

A screenshot of a computer

Description automatically generated



Level 4: Division into routines



1. **Implementation**
2. **Task class**
   * This class contains 3 objects of type int:

Private int id;

Private int arrivalTime;

Private int serviceTime;

* + This class contains 2 constructors:

A picture containing text, screenshot, font

Description automatically generated

* + This class contains 2 methods:

A screen shot of a computer

Description automatically generated with medium confidence

1. **Server class**
   * This class contains 4 objects:
     + Private Vector<Task> tasks;
     + Private Vector<Queue<Task>> queues;
     + Private int nrQueues;
     + Private int simulationTime;
   * This class does not contain any methods besides getters and setters
2. **SimulationManager class**
   * The controller class represents the functionality of the GUI
   * This class implements the java Swing interface ActionListener
   * This class contains the following objects:
     + Private Gui gui;
     + Private Server server;
     + Private ShortestTime shortestTime;
     + Private ShortestQueue shortestQueue;
   * The main method implemented in this class is the override for the interface’s actionPerformed method:

A screen shot of a computer code

Description automatically generated with low confidence

1. **Gui class**
   * This class represents the visual side of the application
   * This class contains a bunch of javax.Swing elements, for example:
     + Public JFrame frame;
     + Public JPanel panel;
     + Public JTextArea logs;
     + Public JScrollPane scroll;
     + Etc.
   * This class also contains 2 methods:

A screen shot of a computer program

Description automatically generated with low confidence

1. **ShortestTime class**
   * This class is used for the implementation of the shortest time strategy of the simulator
   * This class implements the Runnable interface
   * This class contains 2 methods:
     + Private int getQueueIndex(Vector<Queue<Task>>queues)
     + Public void run()

A screen shot of a computer code

Description automatically generated with low confidence

A screen shot of a computer program

Description automatically generated with low confidence

* Screenshot with a part of the code of the run() method

1. **ShortestQueue class**
   * This class is used to implement the shortest queue strategy of the simulator
   * This class implements the Runnable interface
   * This class contains 2 methods:
     + Private int getShortestQueue(Vector<Queue<Task>> queues)
     + Public void run()

A picture containing text, screenshot, font

Description automatically generated

A screen shot of a computer program

Description automatically generated with low confidence

* A screenshot with a part of the run() method

1. **Results**

* The methods were tested both with and without the graphical elements.

A screenshot of a computer

Description automatically generated with medium confidence

* A screenshot with the output in the GUI;

A screenshot of a computer program

Description automatically generated with medium confidence

* Same output in the text file log.txt

1. **Conclusions**

In conclusion, this assignment helped with the understanding of threads and how to synchronize them. Moreover, by using synchronous collections it also helped with getting more experience with how to secure a thread and also just how to use different types of collections that are available for Java. It was also interesting to create the GUI since it is probably one of the more elaborate ui’s I’ve had to design in OOP thus far.

1. **Bibliography**
2. *Java swing questions on:*

[*https://stackoverflow.com*](https://stackoverflow.com)

1. *Thread and run() method information from:*

[*https://stackoverflow.com*](https://stackoverflow.com)

*and*

[*https://www.javatpoint.com/java-thread-run-method*](https://www.javatpoint.com/java-thread-run-method)

1. *run() method information:*

*https://www.javatpoint.com/java-thread-run-method*