DOCUMENTATION

ASSIGNMENT *2*

STUDENT NAME: Cupsa Bogdan

GROUP: 30431

# CONTENTS

[1. Assignment Objective 3](#_Toc128043139)

[2. Problem Analysis, Modeling, Scenarios, Use Cases 3](#_Toc128043140)

[3. Design 3](#_Toc128043141)

[4. Implementation 3](#_Toc128043142)

[5. Results 3](#_Toc128043143)

[6. Conclusions 3](#_Toc128043144)

[7. Bibliography 3](#_Toc128043145)

# Assignment Objective

The main objective is to design and implement an application aiming to analyze queuing-based systems by (1) simulating a series of N clients arriving for service, entering Q queues, waiting, being served, and finally leaving the queues, and (2) computing the average waiting time, average service time and peak hour.

The secondary objectives are:

* Analyze the problem and identify the requirements.
* Design the simulation application.
* Implement the simulation application.
* Test the simulation application.

# Problem Analysis, Modeling, Scenarios, Use Cases

For this project, the functional requirements are:

* The simulation application should allow users to setup the simulation.
* The simulation application should allow users to start the simulation.
* The simulation application should display the real time evolution of queues.
* The simulation application should write everything about the simulation into a text file.

The non-functional requirements:

* The simulation application should be intuitive and easy to use by each user.

Use cases:

A diagram of a person with text

Description automatically generated with low confidence

As use cases, we have the possibility to modify the parameters of the simulation, we can start the simulation and see the simulation being displayed.

Actor: User.

Use Case: Change parameters of simulation.

Main Scenario:

1. The user inserts all the data necessary for the simulation.

Use Case: Start the simulation.

Main Scenario:

1. The data is found in their given fields.
2. The user presses the start button, and the simulation starts.

# Design

The design of this application consists of 10 classes and 1 interface. I used a layered architecture with 3 packages: gui, model and business logic. In the model, we find Task and Server classes. In the gui we have the SimulationFrame (where the simulation is displayed) and InputFrame (where data is entered). In the business logic, we find the Strategy interface, implemented by TimeStrategy and ShortestQueueStrategy. Also, we have the Scheduler and the SimulationManager, responsible for the whole working of the project.

Relations:

* Aggregation between Task and Server. Server has as attribute a LinkedBlockingQueue of Tasks.
* Realization between TimeStrategy and ShortestQueueStrategy and Strategy (Implemented interface).
* Dependency SimulationManager and Scheduler.
* Dependency between Scheduler and Strategies classes.
* Dependency between SimulationManager and the classes from the gui package.

A picture containing text, diagram, line, screenshot

Description automatically generated

Data structures used:

* LinkedBlockingQueue: for the tasks in the server. It is thread safety.
* CopyOnWriteArrayList: used for storing the generated tasks.
* ArrayList: for the servers list and the inputs list.
* List: for the servers in the scheduler.

# Implementation

For the implementation of the polynomial calculator, I have used the following classes:

* Main
* Task
* Server
* SimulationManager
* Scheduler
* Strategy (Interface)
* TimeStrategy
* ShortestQueueStrategy
* SimulationFrame
* InputFrame

Main:

In the main class we check if the simulation started and sleep if not. If it starts, we can start the thread. Also, SimulationManager is instantiated.

Task:

Class from the model package. Basically, representing the person from the queue. It has an id, arrivalTime and serviceTime.

Server:

Class from the model package. Has a waiting period and a LinkedBlockingQueue of tasks. We use this data structure for thread safety. It implements Runnable so it can be instantiated by different threads. We have the add task method. Also, as it implements Runnable, we have the run method where we get a task, suspend for as long as it works, then pop it out of the queue.

Strategy:

Interface containing the addTask abstract method. To be implemented by TimeStrategy and ShortestQueueStrategy.

TimeStrategy:

Implements Strategy. From the list of servers, it checks for the one with minimal waiting time.

ShortestQueueStrategy:

Implements Strategy. From the list of servers, get the one with the shortest length for the queue of tasks.

Scheduler:

Initializes the servers list. It changes the strategy based on the Policy (using policy enumeration). Based on this strategy, is dispatches the task to one of the servers (dispatchTask method).

SimulationManager:

Get the inputs from the graphical user interface. Generates n random tasks. Updated the frame as time goes on. Also calls the scheduler and perfom operations on the generatedTasks list, which is of type CopyWriteArrayList. The SimulationManager is also responsible for the writing in the .txt file.

InputFrame:

Class from the gui package responsible for storing the inputs given by the user regarding the simulation (maxProcessingTime, number of servers, etc.).

SimulationFrame:

Class responsible for displaying the simulation. It has the update method, where everything in changing as time goes on.

# Results

In this section I will show the correctness of the implementation by providing some results.

A screenshot of a computer program

Description automatically generated with medium confidence

A picture containing text, screenshot, font, menu

Description automatically generated

A screenshot of a computer

Description automatically generated

# Conclusions

In conclusion, after implementing the polynomial calculator, I have learned or deepened the following:

* A better understanding of OOP principles.
* Understanding of Thread principles.
* Code organization in packages.
* More organized using the strategy pattern.
* Better at designing the graphical user interface.
* Getting more experience of working on documentations for my projects.
* Being more organized, by first working on the class diagram and only then starting to code.
* Better organization of the code.

# Bibliography

<https://www.wiley.com/en-us/Pattern+Oriented+Software+Architecture%2C+Volume+1%2C+A+System+of+Patterns-p-9780471958697>

<https://www.baeldung.com/parameterized-tests-junit-5>

<https://docs.oracle.com/javase/tutorial/uiswing/>

<https://www.baeldung.com/java-arraylist>

<https://www.baeldung.com/java-string-operations>

<https://dsrl.eu/>

<https://docs.oracle.com/javase/tutorial/essential/concurrency/>

<https://www.javacodegeeks.com/2013/01/java-thread-pool-example-using-executors-and-threadpoolexecutor.html>

<http://www.tutorialspoint.com/java/util/timer_schedule_period.htm>