Technical University of Cluj-Napoca

Faculty of Automation and Computer Science

Queue management system

- Assignment 2 -

Ionuț Matiș

Group: 30421

Table of contents

1. Objective……………………………………………………. 3
2. Problem analysis, modelling, use cases ……........................ 3
   1. Analysis ………………………………………………… 3
   2. Modelling ………………………………………………. 3
   3. Use cases ……………………………………………….. 4
3. Design ……………………………………………………… 5
   1. Design decisions ………………………………………... 5
   2. UML diagram ………………………………………….... 5
   3. Data structures & packages …………………………….. 5
4. Implementation ……………………………………………. 6
5. Results ……………………………………………………… 11
6. Conclusions and future developments ……………………... 11

1. Bibliography ……………………………………………….. 12

# Objective

The main objective of this project is to design and implement a simulation application aiming to analyze queuing-based systems for determining and minimizing clients’ waiting time. More precisely, this application deals effectively with the customers (tasks) who want to move to the shortest queue (server). Furthermore, it also helps us understand the concept of multithreading.

# Problem analysis, modelling, use cases

2.1 Analysis

When we think about queues, we create a mental image of an action that is done in parallel. For example, think about a supermarket. It has a number of queues in which clients wait an amount of time to pay for the products they bought.

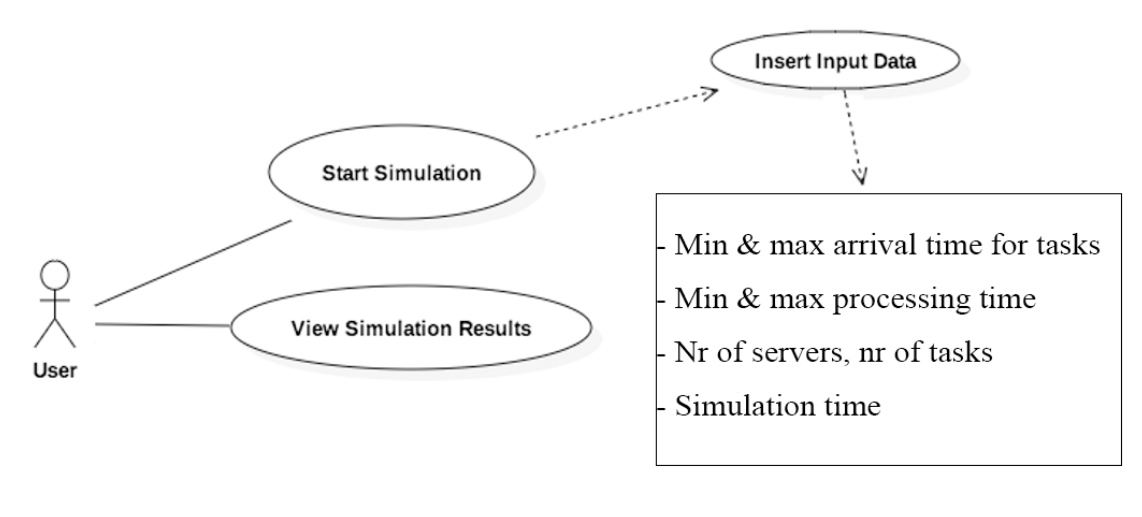
What is interesting is the fact that the clients in different queues do not care about one another. It seems normal, but when we want to model this using object oriented programming, we face a big challenge, because, at a first glance, it may seem there is no way we can perform this task. However, there is a solution – using Threads. This concept allows us to run multiple programs in parallel, so that we can distribute the clients in order to minimize their waiting time.

2.2 Modelling

The best way to model our problem is to use multiple queues, implemented as FIFO data structure (First In, First Out), because it resembles real life very well – the first client (task) to enter the queue (server) is the first to exit it. The operations to be done (push, pop) are already implemented in the JDK.

What is a thread? In computer science, a thread of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system. A thread is a component of a process. Multiple threads can exist within one process, executing concurrently and sharing resources such as memory, while different processes do not share these resources. The threads of a process share its executable code and the values of its variables at any given time.

2.3 Use cases



In order to use this program, the user has to do the following:

* 1. Run the application
  2. Enter valid data (integers, minimum smaller than maximum)
     1. Enter the minimum and maximum arrival time for tasks
     2. Enter the minimum and maximum service (processing time) for tasks
     3. Enter the number of servers
     4. Enter the number of tasks
     5. Enter the simulation time

If the entered values are not valid (the user entered characters or left empty spaces), he will get an error message and he will insert valid data to be processed

* 1. Press the button “Start” to run the simulation //start simulation
  2. The program will start and it will display in real time the queues and the global time of the application. It will also display a message with who entered or left the queue. In the end (when the timer reached its final value), the user will receive some statistics about the simulations (average time and peak time).
  3. The user can enter another set of data and press again “Start” (go to b.) or he can exit the application (pressing X in the right corner of the screen)

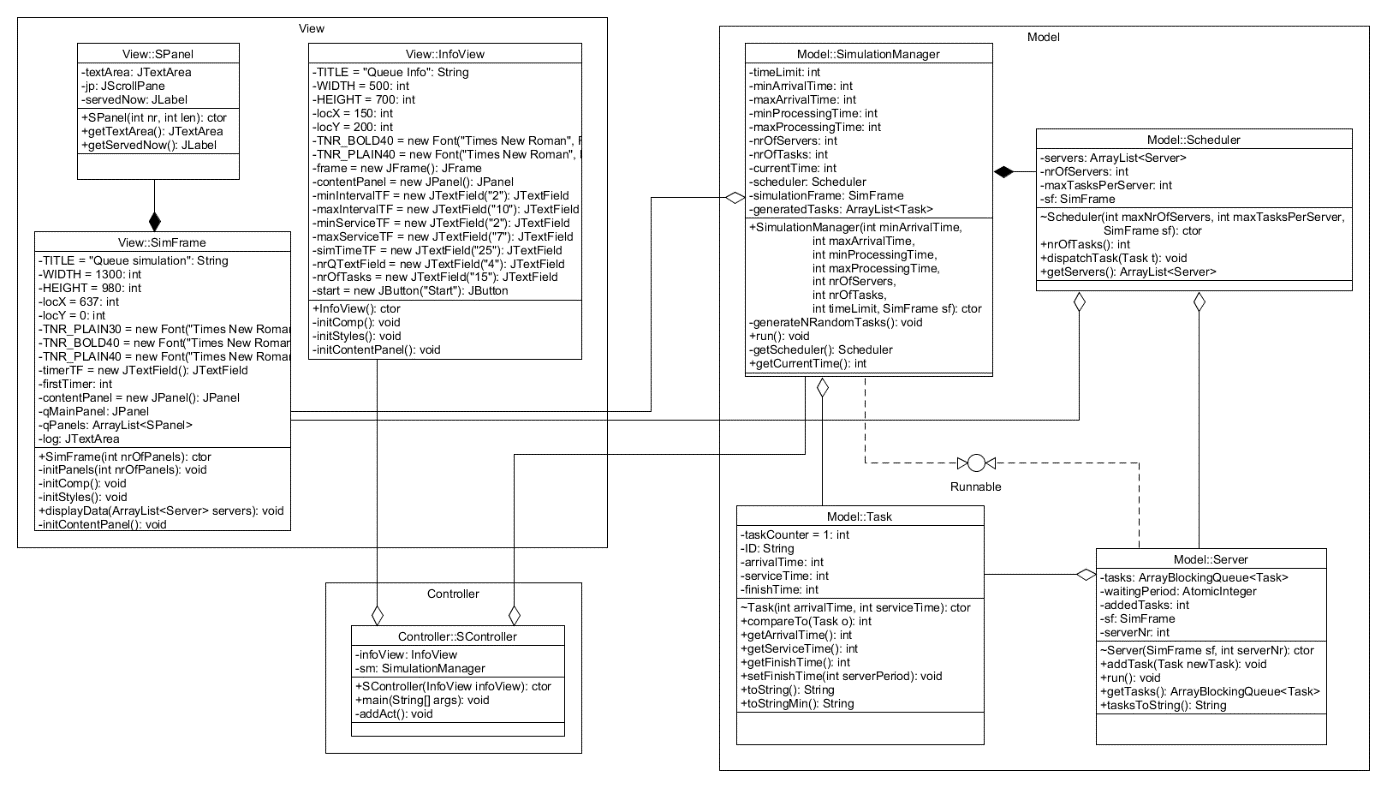
# Design

* 1. Decisions

Being a thread associated problem, I decided to make the Server class to implement the runnable interface, because instances of this class will be run in parallel. Another class that will implement the Runnable interface will be SimulationManger, which has the associated timer and starts all servers. The UI will consist of 2 frames. One will be used for data fetching and the other will be used to display data in real time (timer, queue evolution and logger).

* 1. CLASS Diagram !!!

Using UML diagram, we can see the final form of the project, highlighting the use of packages, classes, methods, and the relationship between them. All of this is done with respect to the Object Oriented Programming model.

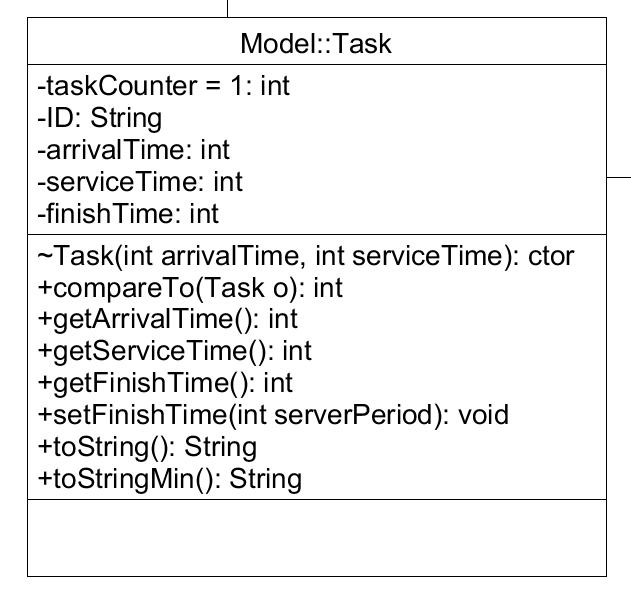


* 1. Data structures & packages

As data structures, I used ArrayLists to store various object, such as Tasks, Servers and even and array of custom JPanels. The project has a Model View Controller structure. The packages are the following: model contains the classes “Task”, “Server”, “Scheduler” and “SimulationManager”, view contains the classes “SPanel”, “InfoView” and “SimFrame”. Finally, the controller package contains the class “SController”. screenshot interfata

# Implementation //descrie relatii

4.1 Task class



It has special fields that are primarily used in multithreading (ArrayBlockingQueue and AtomicInteger). It also has an instance of simFrame so that it can write in the GUI.

The run method is presented below.

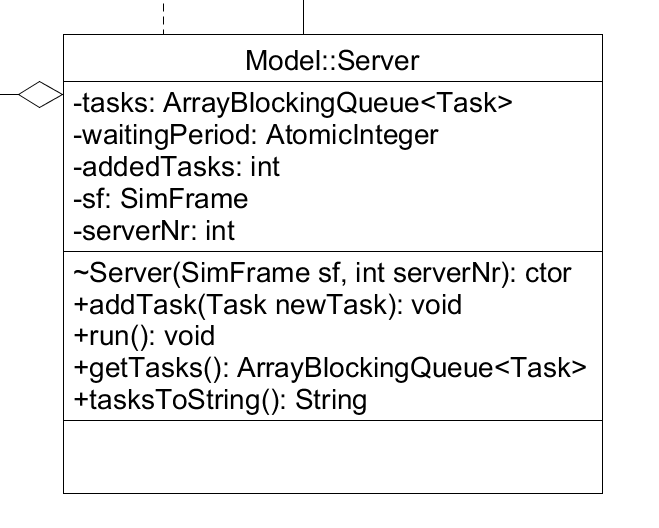
The to string method creates a string of all tasks of the current server so that they can be displayed in the UI. This is very important, because it is heavily used in the simulation manager run method.

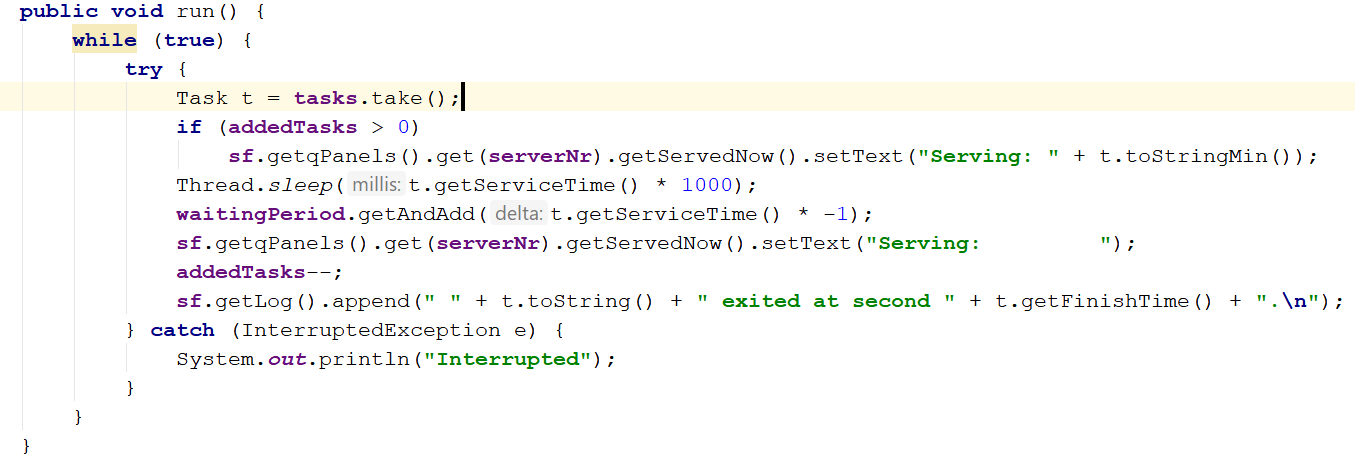
It has specific variables in order to perform the requirements. It also implements Comparable so that I can sort the tasks based on their arrival time.

It has a static counter so each time I create an object of this class, the counter increments and.

It has 2 toString methods – one for the long display (id + arrival time + service time) and one for displaying just the id.

4.2 Server class



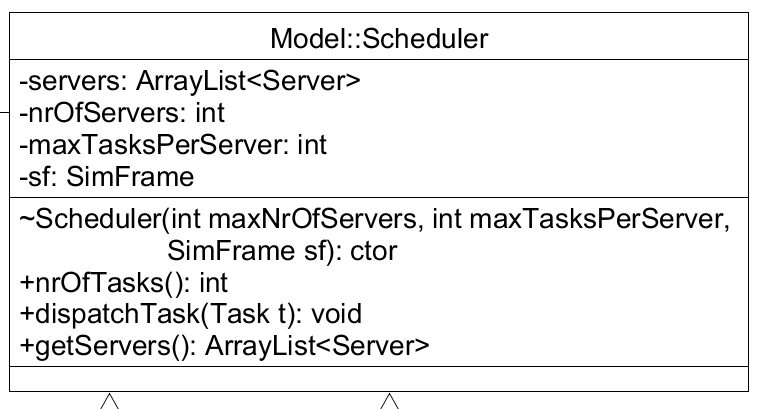


The run method implemented above takes the first element in the queue and updates the UI with the element it took form the queue. After this, I put the thread on sleep for the amount of time the task has service time. Next, I decrement the server waiting time and add to the log what server exited and at what time.

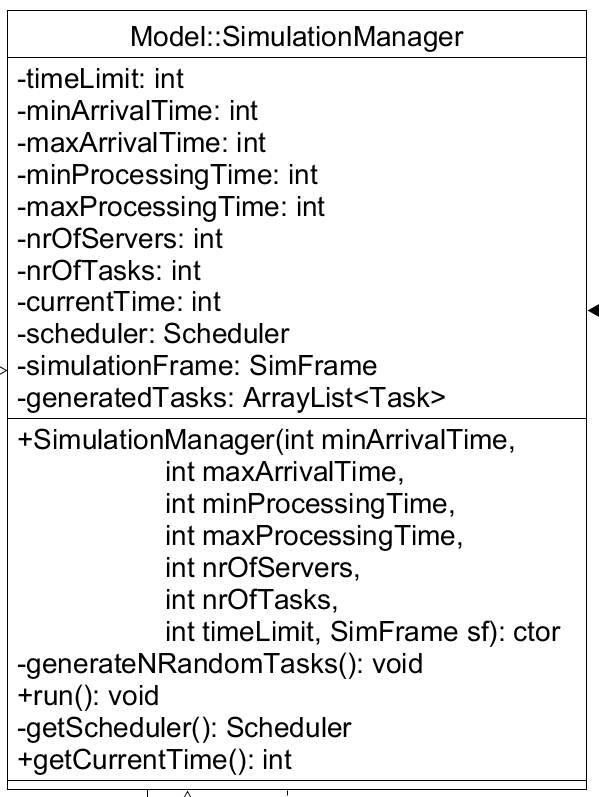
4.3 Scheduler class

It has the list of all servers in the queue (extracted from the interface) and an instance of SimFrame so that it can write in the log.

On special method here is the dispatchTask, which commands in which server the current task will go – i.e. in the server with the smallest amount of tasks (SHORTEST QUEUE policy). It does this by searching through the servers and returns the index of the server which has the minimum number of tasks. It also updates the log of the UI.

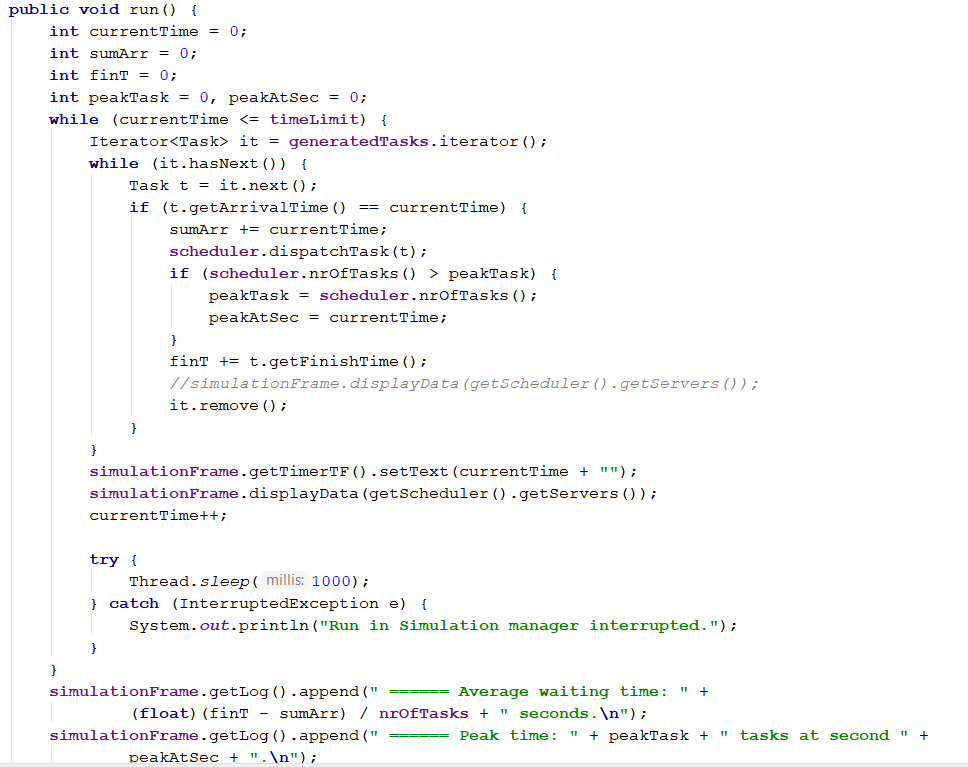


4.4 SimulationManager class



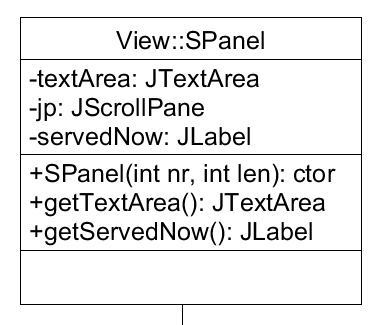
This class is the most important one in the project, because it coordinates all other objects. Its parameters are the ones read from the UI. In its constructor I initialize the scheduler and assign a thread for each one of the servers.

generateNRandomTasks method is the Random Client(Task) generator. Generates nrOfTasks tasks with a random arrival and service time and adds them to the list of generated tasks. In the end, it sorts the tasks based on their arrival time, using the sort method of Collection framework. The run method is presented beneath.

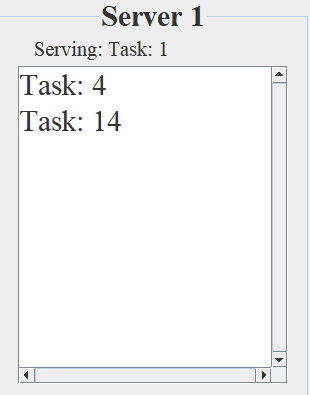


This thread has the main timer and will run for a specific amount of time, counting the seconds that have passed. It gets one task from the generated ones (remember that they are sorted based on their arriving time) and it the current timer is equal to the task’s timer, it sends the task to be added to one of the servers. In this method I also find the peak time, by checking every second the total number of tasks from all servers. After the processing, the task is removed, the timer is set and the data is updated in the UI. After the process ends, in the log area of the simulation frame will be displayed the average time of the simulation (by subtracting the sum of all arriving times from the sum of all finish times and then dividing the result to the total number of tasks created). After this, the peak time is also displayed (it was computed in the while on each iteration).

4.6 SPanel class

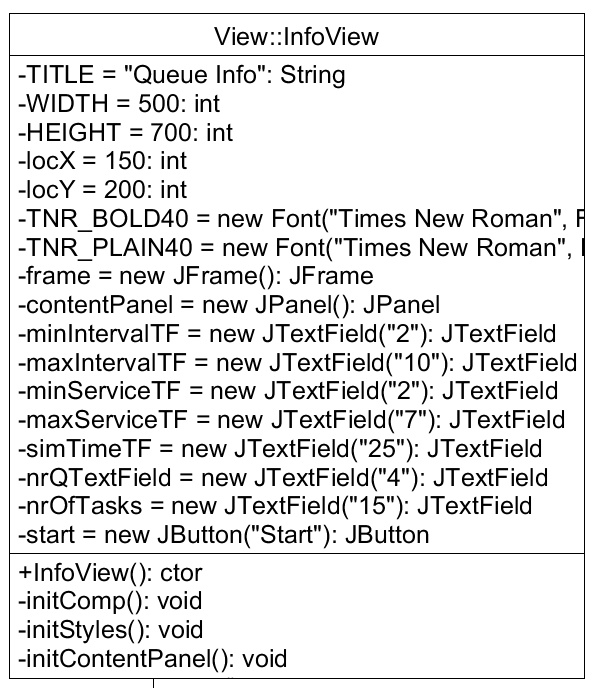


This panel extends JPanel and it has some special attributes. It has a Title (I used a TitledBorder), a label for currently serving client and a text area that displays the tasks that need to be processed. If there are too many tasks, the scroller will come in help so you can visualize all the elements in the server.



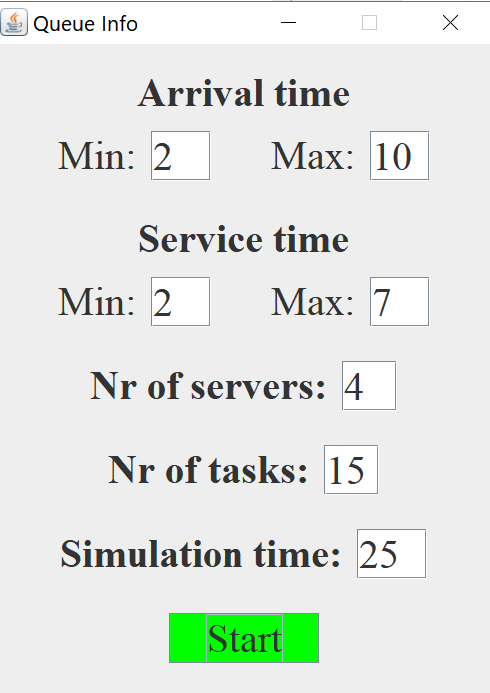
Visual representation of the SPanel

4.7 Infoview Class

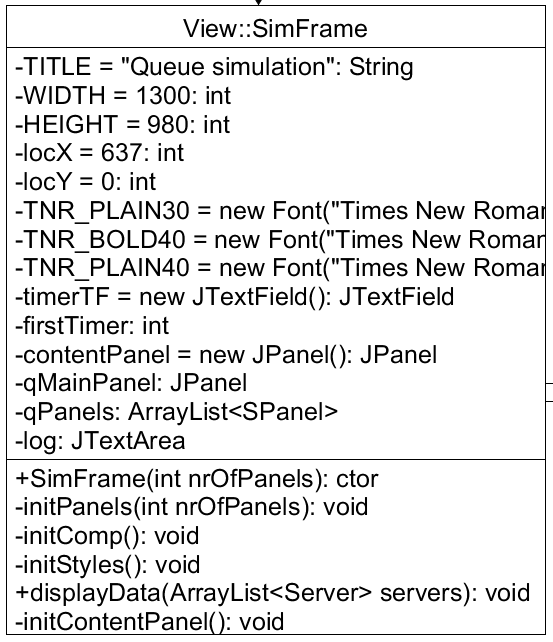


This class models the first frame that appears on the screen when the user starts the application. It has several text fields where the user can insert the data and a button to load it.

InitComp() sets the attributes for the frame (width, heigth etc), initStyles initializes the styles of the components (Fonts, Sizes etc) and initContentPanel() builds the frame, adding one component after another.

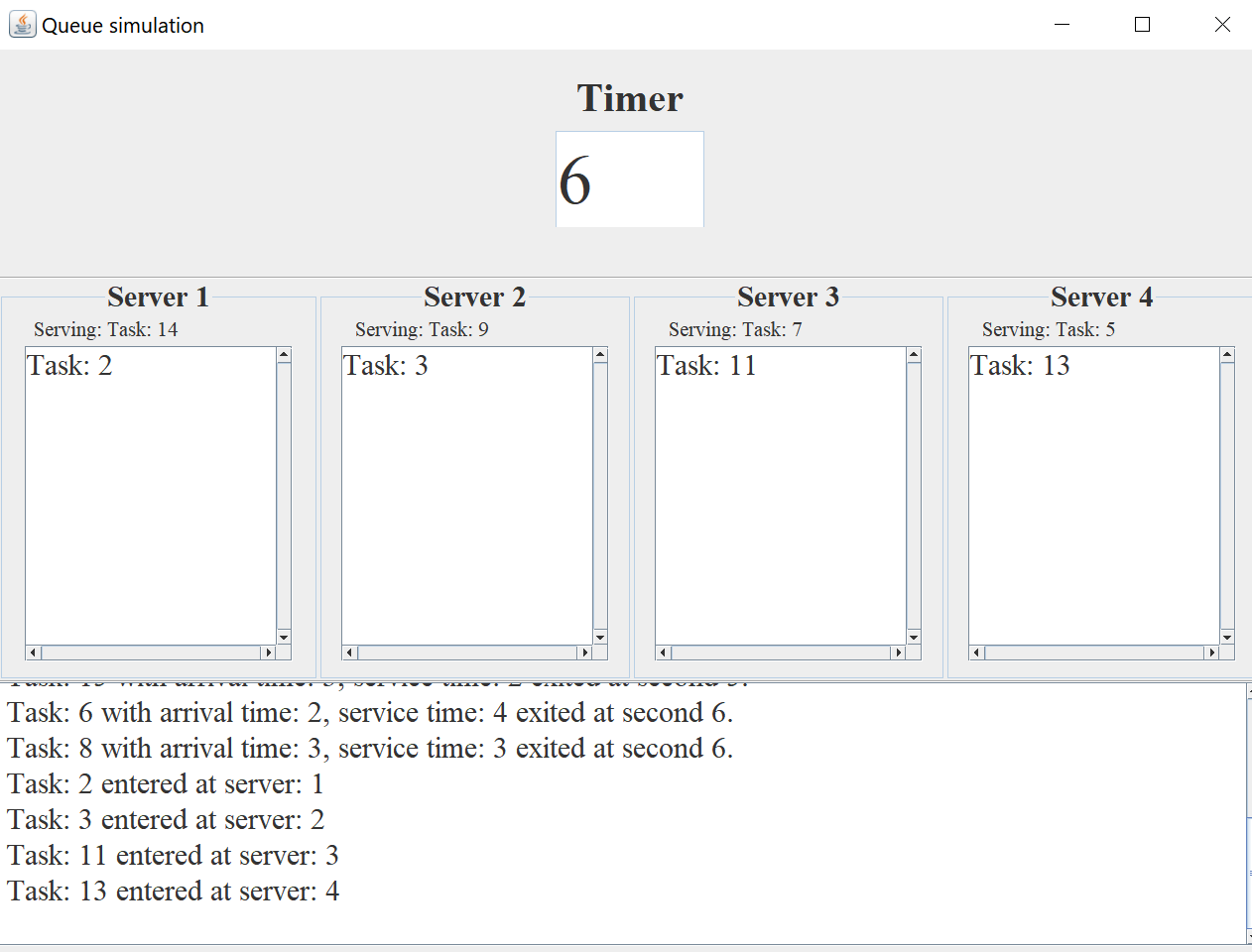


4.8 SimFrame class



This class shows how the simulation frame is built. It has a timer, an ArrayList of previously explained SPanels which are put into JPanel and a JTextField to display the log.

The constructor takes the nr of SPanels to be added to the frame and calls initPanels method. initComp, initStyles and initContentPanel do the same as the previous class (InfoView). displayData takes an argument of type ArrayList<Server> and updates each corresponding JTextField of the SPanel with the tasks they currently have. It does this iterating through each server and using its tasksToString method.



tHere you can see the layout and the log text area as it constantly updating. This is due to the fact that an instance of this class appears in both scheduler and Simulation Manager. Both of the classes have run methods (i.e. work concurrently) and have access to the fields of this class. The simulation frame constantly updates the timer and calls the method displayData for the remaining tasks so that they appear in their correct textbox. The server has access to the label of the Spanel (the one above the textfield) and it updates it after the client that was served exits the queue. It also has access to the JTextArea in the bottom part of the panel and appends a message based on what happened using this line of code:

sf.getLog().append(" " + t.toString() + " exited at second " + t.getFinishTime() + ".\n");

# Results

The results of this project can be seen only by starting the appl, because only in this way we can see how task enter the server (choosing always the one with the smallest nr of tasks), waiting to be processed and then removed from the queue.

After creating all the classes which I described in the above chapter and all the relationships between them, I can say that the application which I have developed simulates very well a queue-based system. It is built based on the skeleton received at the laboratory and has a simple and user-friendly interface which also warns the user if he tries to insert invalid data.

# Conclusions and further developments

The base of the project is the one received at the laboratory. It helped me a lot and I think that without it I could have been able to solve this project.

I have learnt a lot of useful things during the time I worked on this project, such as how to work with multiple threads in Java, how to build a Producer-Consumer application and it even taught me to manage my time more accurate.

For the further developments I would choose to make a graphical user interface with images, such as stickmen representing the tasks and a special stickman to be the cashier, instead of a server. Also, I would implement 2 strategies for distributing the clients so that the user could pick the one he or she prefers.

# Bibliography

1. <http://www.wideskills.com/java-tutorial/java-threads-tutorial>
2. <https://docs.oracle.com/javase/7/docs/api/javax/swing/border/TitledBorder.html>
3. <https://dzone.com/articles/multi-threading-java-swing>
4. <https://www.javamex.com/tutorials/threads/>
5. <https://stackoverflow.com/>
6. <https://www.wikipedia.org/>