

**PROGRAMMING TECHNIQUES**

Homework 3

**Student:Bresan Tudor Vlad**

**Group:** 30423

**Table of Contents**

**1. Objective 3**

**2. Problem analysis, modeling, scenarios, use cases 3**

**2.1. Problem Analysis 3**

**2.2. Modeling 4**

**2.3. Scenario and use cases 4**

**3. Design 5**

**3.1. UML Diagram 5**

**3.2. Data Structures 6**

**3.3. Class Projection 6**

**3.4. User Interface 8**

**4. Implementation and use of cases 9**

**5. Results 10**

**6. Conclusion and possible updates 10**

**7. References 10**

**Documentation Queuing Based Systems**

1. Objective

Design and implement a simulation application aiming to analyze queuing based systems for determining and minimizing clients’ waiting time.

1. Problem analysis, scenarios, use cases

Queues are commonly seen both in real world and in the models. The main objective of a queue is to provide a place for a "client" to wait before receiving a "service". The management of queue based systems is interested in minimizing the time amount its "clients" are waiting in queues. One way to minimize the waiting time is to add more servers, i.e. more queues in the system (each queue is considered as having an associated processor) but this approach increases the costs of the supplier. When a new server is added the waiting clients will be evenly distributed to all current available queues.

The application should simulate a series of clients arriving for service, entering queues, waiting, being served and finally leaving the queue. It tracks the time the clients spend waiting in queues and outputs the average waiting time. To calculate waiting time we need to know the arrival time, finish time and service time. The arrival time and the service time depend on the individual clients – when they show up and how much service they need. The finish time depends on the number of queues, the number of other clients in the queue and their service needs.

2.1 Problem analysis

If we take a closer look to the problem of order management we shall find that it is a more complex problem that it may seem at first glance.

First of all, we need to find a way to make all the queues run in the same time, so that it will simulate a genuine real life situation.

Another problem concerning the queue system would be the way the user gives us the data and the way we tell him the result of the processing. After doing some research on the internet and looking at online applications that provide these services that cover the order processing problem I have reached the conclusion that the best way in terms of displaying data which comes from tables is using naturally a text field containing all the queues which is put in the Graphic User Interface.

Further on we shall analyze all the aspects which needed to be managed in order for the right functioning of this project.

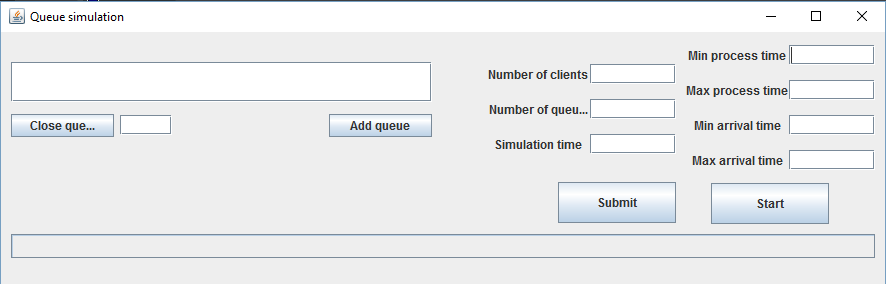
* 1. Modeling

The implementation of this project is the same one as the one presented in the section 2.1. There is only one package which is used for storing all the classes. There are Task (which stores the data of every Task), TaskGenerator(which generates tasks for our queues), Queue(which stores the data of every queue), Manager(is a class that manages our queues), Time(is a class where we keep track of passed time) and GUI(which implements our interface). We shall analyze the role of each of these classes further on.

2.3 Scenarios and use cases

The scenarios were already mentioned, but I will present the details here. Firstly I thought about how the tasks will be managed by the application and my idea was: Tasks stored in the Queue , Queues stored in the Manager , a TaskGenerator in order to create Tasks, a Time to keep track of the execution timeline and a GUI to display the running of the application. I followed this plan and I did not have any unexpected surprise when it comes to the implementation and the structure of the program.

The use cases are strictly dependent on the user, and finally I order to make the application as user friendly as possible I decided to implement the following user interface:



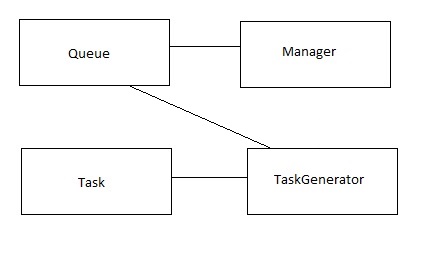
From now on I will present the functionality of the application using the “lifetime” of a Task. A Task is created in the TaskGenerator class using the Manager class. The Manager’s class role is finding the best Queue to add the Task, it opens servers if the corresponding button is pressed and the number of servers does not have a limit , this is going to be presented in further implementetions. Further on, a Task is given its properties which are: Arrival time, Process time. It is popped from the Queue from the user interface when the Task ends (its processing time ends after he reaches the top of the Queue). This process is done between and interval of time in which the Queues are open. After the end of the OPEN status of the Queues the application prints the peak hour , the peak processing time and the Average empty queue time, which repsresents the time a queue took to empty itself.

1. Design

3.1 UML Diagram

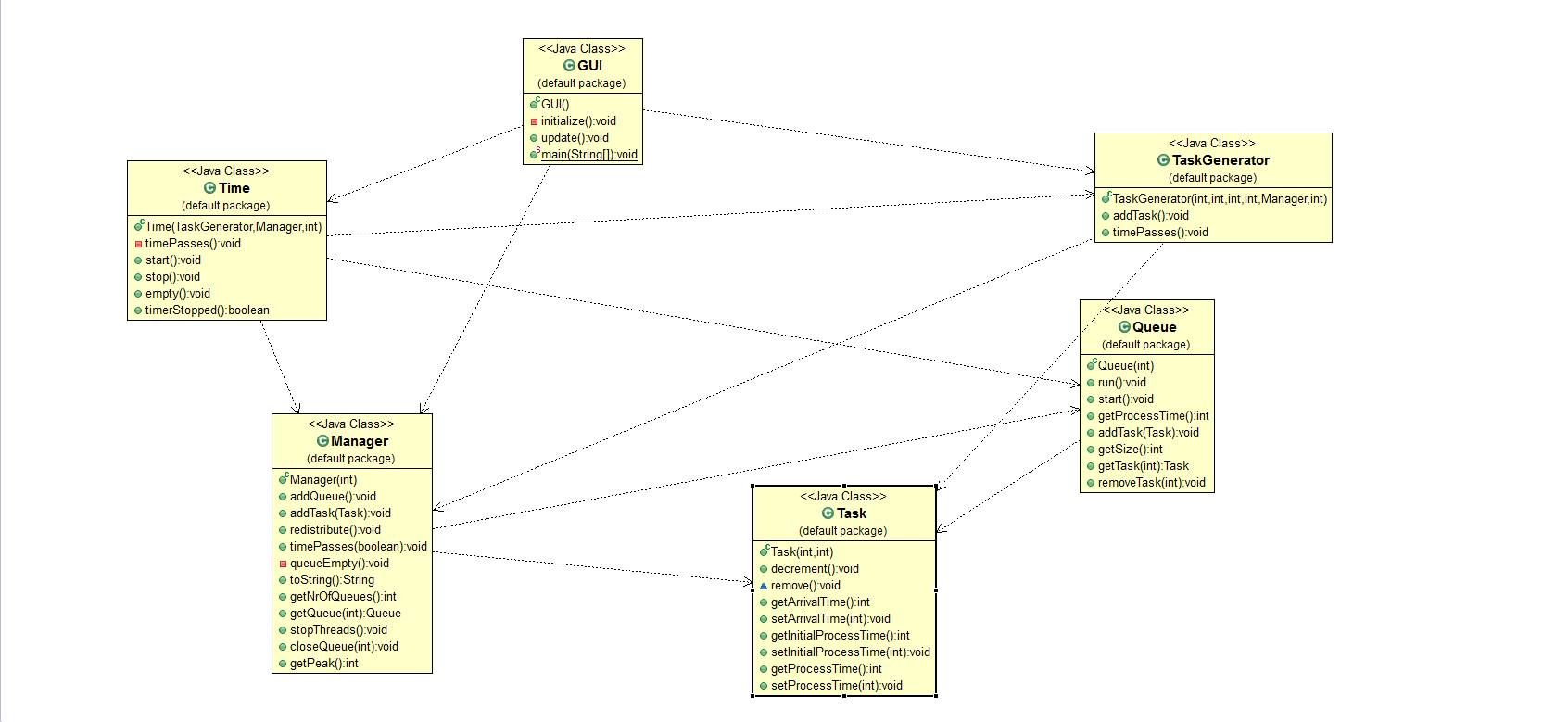
The UML diagram is a class diagram in which we can find the relationship between classes and also the elements that the specified class contains.

Design phase diagram



Final stage diagram

The following diagram represents the final stage of the project. This diagram was created using a plug-in after the application was running successfully.



3.2 Data structures

In this application I have used various data types including JButtons, JPanels, JFrame,JTextField for the GUI package, Server, Task, for transferring data between classes and the classic int, String, Boolean for regular operations.

3.3 Class projection

Class projection refers mainly to how the model was thought, how the problem was divided in sub-problems, each sub-problem representing more or less the introduction of a new class. First I will start by mentioning exactly how my problem was divided into classes. I begin by creating the main method which starts the execution of the application. Then, as you can see from the UML diagram, I called the Manager class which starts the actual execution by a repetitive loop while(true). Further on, everything is done mostly by Threads, being the most important feature of the application.

This main package contains all the classes which make this application run as it does. We shall start explaining the code from the GUI class.

**public class GUI {**

**private JFrame frmQueueSimulation;**

**Manager manag;**

**TaskGenerator taskGen;**

**Time t;**

**Timer t1;**

**int simProgress=0;**

**private JTextField nrOfClientsField;**

**private JTextField nrOfQueuesField;**

**private JTextField simulationField;**

**private JTextField minProcessTimeField;**

**private JTextField maxProcessTimeField;**

**private JTextField minArrivalTimeField;**

**private JTextField maxArrivalTimeField;**

**private JProgressBar progressBar;**

**private JButton btnAddQueue;**

**private JButton btnNewButton;**

**private JTextField textArea;**

**private JButton btnCloseQueue;**

**private JTextField textField;**

**public GUI() {**

**initialize();**

**}**. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . .. . . .. . . . . . . . . .

This is the main code of the GUI class which describes perfectly how everything is done. In the GUI class we create the user interface which allows the user to input the range of arrival time for the customers , the range of process time for each customer , the number of queues , the initial number of clients that are in the queues. The interface also allows the user to add an additional queue or to close a queue ,based on it’s index.

The Manager class is the most important of the application, as it contains most of the logic on which the program is based. We shall analyze some of it’s essential code.

**public class Manager {**

**ArrayList<Queue> queues = new ArrayList<Queue>();**

**private int simulationTime;**

**public static int peakVal;**

**public static int peakHour;**

**public static int queueEmptyTime=0;**

**private int[] closedQueues=new int[100];**

**public Manager(int nrOfQueues) {**

**for (int i = 0; i < nrOfQueues; i++) {**

**queues.add(new Queue(i));**

**queues.get(i).start();**

**closedQueues[i] = 0;**

**}**

**}**

**public void addQueue() ;**

**public void addTask(Task tsk) ;**

**public void redistribute() ;**

**public void timePasses(boolean simulationOver) ;**

**private void queueEmpty() ;**

**public String toString() ;**

**public int getNrOfQueues() ;**

**public Queue getQueue(int i) ;**

**public void stopThreads() ;**

**public void closeQueue(int i) ;**

**public int getPeak() ;**

This is the main Thread which keeps on running until the final time is equal with the current time. I have used a special method timePasses() in order to simulate as good as possible the passing of the time. This class implements methods that close the queue, stops a thread , gets the number of queues etc.

The Task class only stored the data of each Task and therefore there is no important code in it only setters and getters.

**public** **class** **Task** {

**public** **int** arrivalTime=0;

**public** **int** initialProcessTime;

**private** **int** processTime=0;

**public** **Task**(**int** minProcessTime,**int** maxProcessTime){

processTime = minProcessTime + (**int**) (**Math**.*random*()\*(maxProcessTime - minProcessTime));

initialProcessTime = processTime;

}

The TaskGenerator class is used to generate random arrival times for each new Task. We shall analyze its code:

**public** **TaskGenerator**(**int** minArrivalTime,**int** maxArrivalTime,**int** minProcessTime,**int** maxProcessTime,**Manager** manag,**int** init){

**this**.minArrivalTime = minArrivalTime;

**this**.maxArrivalTime = maxArrivalTime;

**this**.minProcessTime = minProcessTime;

**this**.maxProcessTime = maxProcessTime;

random = (minArrivalTime + (**int**)(**Math**.*random*()\*(maxArrivalTime-minArrivalTime)));

**this**.manag = manag;

**if**(init>0){

**for**(**int** **i**=0;i<init;i++){

addTask();

}

}

The Time class keeps track of time by using timers such that when the final time is reach , a delay is applied if some of the customers are still in the queue. After the delay reaches the set value the final data log is displayed.

The Queue class is very important as it is the core of the application. Here we have its code:

**public** **class** **Queue** **extends** **Thread** {

**ArrayList**<Task> queue = **new** ArrayList<Task>();

**int** queueNr;

**Thread** t;

**public** **static** **int** *waiting*;

**public** **boolean** stopThread = **false**;

**public** **Queue**(**int** queueNr) {

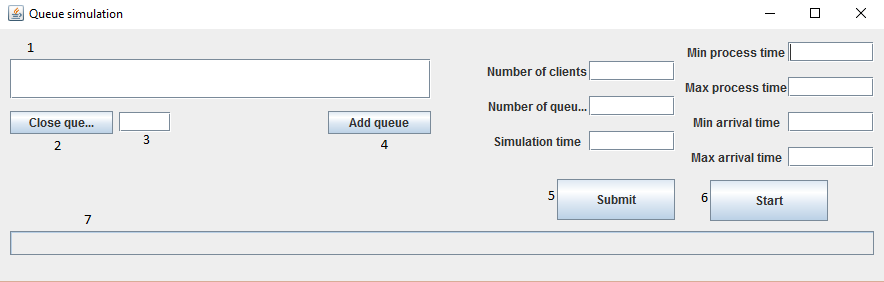
**this**.queueNr = queueNr;

}

It extends Thread class so that we can override Thread methods such as: Run. In this class we also have setters and getters in order to access the desired data through objects. We can also add a task or remove one from a desired queue.

3.4 The interface

The interface is made out of a frat which includes several elements such as : JTextField , JLabel , JButton , JProgressBar ( this object helps the user see the progress of the simulation time ) . It also provides the action listeners for the buttons/text fileld which are available for the user. We shall describe each component in order to understand how the user interface is created.



This photo represents the GUI and we’re going to describe each indexed component.  
1. Stands for the area where our queues are going to be displayed

2. Represents the button to close a queue

3. Represents the text field in which the user has to input the index of the queue which is going to be closed

4. Represents the button to add a queue

5. Represents the button which send the values from the textfields to the main logic of the project

6. Represents the button which needs to be pressed in order for the simulation to start.

7. Represents the progress bar for our simulation time

3.5 Implementation

In what the implementation is concerned this project was developed in Eclipse and it was only tested in this environment. However the program should maintain its portability. Concerning the code implementation I did not make use of laborious algorithms, but I have rather stayed faithful to the classical algorithms of computing polynomials learned in high school. However I have tried to implement my problem in a way that appears to me as being the most efficient one. Testing implies checking for any errors in the program or limitations of this program. Due to the fact that the program is rather simplistic, they are few errors that might generate this program to work wrong or to stop. These errors are mostly related to the interface or the number of queues, the total number of clients etc. I have assumed that the user is aware of basic concepts of mathematics and realizes that is not possible for a store to have a large amount of queues (i.e. 500) or a large amount of customers (i.e. 5000) , otherwise if he enters data with invalid format the program will probably generate some bugs and will stop. Hence this part with checking all the possible scenarios will be seen as future development.

1. Results

The application is an user friendly and useful application to analyze queuing based systems for determining and minimizing clients’ waiting time. As the application is developed on a Java platform, it is highly portable and allows it to run on several operating systems (as long as they have the Java SDK installed). The application is straightforward an easy to understand and to use by any user who respects the basic concepts of real world examples . Even though being limited, this application can be considered as being a helpful tool that can be used when dealing with such data storing situations.

1. Conclusions

All in all the application provides a close to reality approach of a supermarket. The application can open a new queue , close a desired queue and it generates new clients at different times. By implementing this application , even though it was pretty hard , I’ve started learning about Threads in java but it’s a long way to go to master such knowledge. As a further development some bugs need to be fixed such as: reopening a previously closed queue, announcing clients that the queue is going to close in 5 minutes and printing all the clients in the queue(so far the application only prints the total process time of a queue, the process time being modified over as time passes due to the fact that some clients may enter or leave the queue).

1. References
2. <http://stackoverflow.com/>
3. <https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html>
4. <http://www.tutorialspoint.com/java/java_multithreading.htm>l