Student: Ardelean Eugen-Richard

Group: 30423

Programming Techniques

Homework 2

Contents:

1. Problem Specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
2. Analyzing the problem, modelling, scenarios, use cases . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
3. Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface) . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
4. Implementation and testing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
5. Results . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
6. Conclusion and future developments . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
7. Bibliography . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
8. **Problem Specification**

**Objective**

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

**Description**

Queues are commonly used to model real world domains. 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 their "clients" are waiting in queues before they are served. 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 service supplier. When a new server is added the waiting customers 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 customers 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 clients in the queue and their service needs.

**Input data:**

- Minimum and maximum interval of arriving time between customers;

- Minimum and maximum service time;

- Number of queues;

- Simulation interval;

- Other information you may consider necessary;

**Minimal output:**

- The average of waiting time, service time and empty queue time for 1, 2 and 3 queues for the simulation interval and for a specified interval (other useful information may be also considered);

- Log of events and main system data;

- Queue evolution;

- Peak hour for the simulation interval;

1. **Analyzing the problem, modelling, scenarios, use cases**
   1. **Problem analysis**

This application should simulate customers waiting to receive a service (e.g. supermarket, bank, etc.). Like in the real world, they have to wait in queues, each queue processing clients simultaneously. The idea is to analyze how many clients can be served in a certain simulation interval, by entering parameters in an intuitive, user-friendly, application graphical interface.

The clients are generated randomly, each having his own service time and arrival time. How many clients are generated, depends on the input values.

The user can set:

* the maximum number of queues available to process customers. At the beginning, only one queue will be open, and, depending on how many clients are generated and their service time, additional queues will be opened so that the clients are put in the smallest queue
* minimum and maximum arrival interval: the delay between customers arriving to receive a service. When generating clients, the arrival time will be chosen randomly between the 2 values using the data type Random
* minimum and maximum service time: the number of minutes needed for a client to be processed, a value is chosen randomly between the minimum and maximum
* simulation interval: the starting and finishing time of the simulation

The user can read:

* How many customers were served (not generated), during the simulation interval;
* The average service time of the served customers (in minutes);
* The average waiting time of the served customers (in minutes): how much the customers have waited in queue to receive the service;
* The total service time of all served customers;
* The total waiting time of all served customers;
* The “peak hour”, when the most clients were served;
* Additional details and statistics can be read in the detailed log, by clicking the dedicated button.
  1. **Modelling**

Modelling is the activity to make an abstract concept easier to understand by finding its main characteristics and defining some laws which make the given phenomenon quantifiable. This process also includes the decomposition of a complex problem into smaller and simpler problems which will be easier to implement.

The main part of the problem is the creation of the queues, but the most important part is generating the clients randomly, which is done using the Thread class.

* 1. **Scenarios**

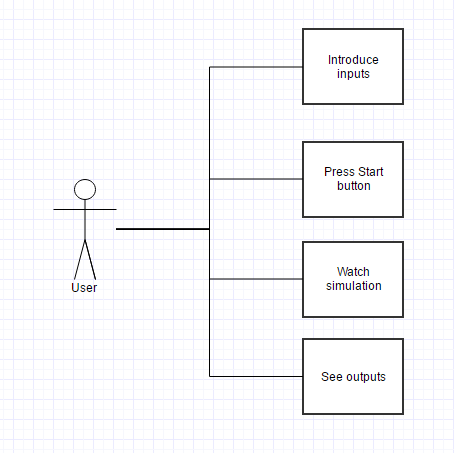
A success scenario looks like this:

* User successfully inserts the input parameters
* User pushes the "Start" button to start the simulation
* Application displays the output

While a failure scenario looks like this:

* User types wrong data in the input fields
* User pushes the "Start" button attempting to start the simulation
* An error message is displayed
* Another change of inserting/correcting the input data is provided
  1. **Use cases**

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors).



1. **Design (design decisions, UML diagrams, data structures, class design, interfaces, relationships, packages, algorithms, user interface)**
   1. **Design decisions**

I chose to use the object-oriented approach with a class for clients, one for queues, one that generates the randomness of the added clients and the graphical user interface as one class.

The queue class is made so that it tries to delete clients every time it can, but it has to respect the rules given by the input. This is done using the notify functions and the wait.

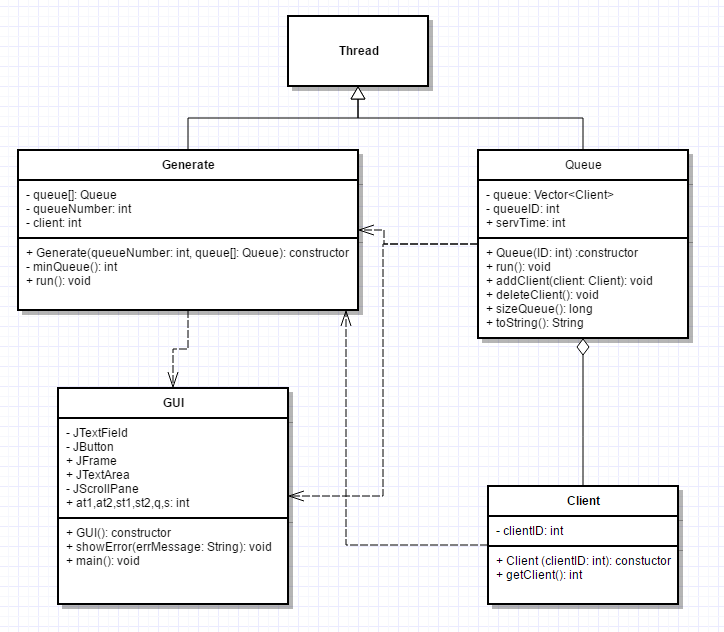
While the generate class creates clients with random service time and random arrival time and add the client to the queue with the minimum number of elements (clients).

The graphical user interface is explained in the user interface part of the documentation.

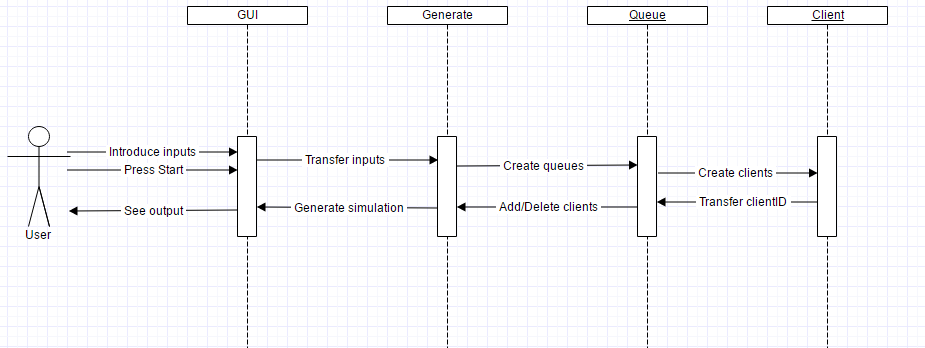
* 1. **Diagrams**

We prepare UML diagrams to understand a system in better and simple way. A single diagram is not enough to cover all aspects of the system. So, UML defines various kinds of diagrams to cover most of the aspects of a system.

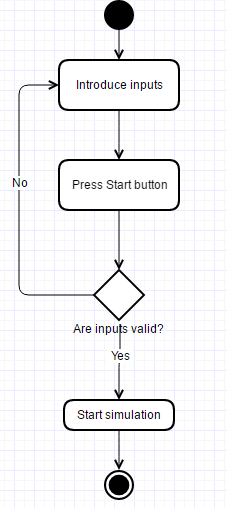
Class diagrams are the most common diagrams used in UML. Class diagram consists of classes, interfaces, associations and collaboration. Class diagrams basically represent the object-oriented view of a system which is static in nature.



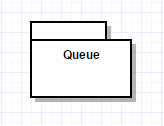
A sequence diagram is a diagram that shows the interaction between the different objects of the application. For Queue Simulation it would look something like that:



Activity diagram is basically a flow chart to represent the flow from one activity to another activity. The activity can be described as an operation of the system.



A package diagram in the Unified Modeling Language depicts the dependencies between the packages that make up a model. And in our case we have only one package.



Deployment diagram is a structure diagram which shows architecture of the system as deployment (distribution) of software artifacts to deployment targets. Artifacts represent concrete elements in the physical world that are the result of a development process.

Component diagram is a special kind of diagram in UML. It does not describe the functionality of the system but it describes the components used to make those functionalities.

* 1. **Data Structures**

The main problem to be taken into account when designing a class is choosing which data structures are needed. Choosing the wrong data structures, can result in less efficient performance.

I chose to use the Vector java class because it implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a Vector can grow or shrink as needed to accommodate adding and removing items after the Vector has been created.

This vector has to be able to behave like a queue, in order to use this we will look at the FIFO structure (first in, first out), in order to do this we model the vector so that when you add you always add a new element to the end of the vector and always remove the first. We simulate queues by storing clients assigned to a queue in a vector, practically each queue is a vector.

* 1. **Class Design**

As you can see in the class diagram I chose to implement this application using 4 classes: one that represents clients, one that represents queues, one that generates the simulation and the last that represents the graphical user interface.

Client is an individual entity, which has to be modeled separately, because each client has its own, unique properties (arrival time, service time). Customers are generated randomly during the simulation, and assigned to the shortest queue (the queue with the smallest number of customers).

Queue represents a queue from the real world, where clients wait in line to receive a service. Clients can be assigned to a queue by using the dedicate method addClient(), but they can also be removed after they were served or in order to move them to a less crowded queue.

Generate class is the class that generates the new clients so that their service time and arrival time is random.

GUI represents the graphical user interface and it will be explained in the User interface part of the documentation.

* 1. **Interfaces**

For the implementation of this project, there were no interfaces used as there wasn’t any need to implement them. All classes have different methods, even if they have the same name, the way they operate or / and the parameters are different.

There was the possibility of using the interface Runnable but I chose to implements the classes using inheritance from the class Thread.

* 1. **Relationships**

For this application, there are some inheritance relationships between the classes Queue and Generate from the class Thread, this is used in the implementation of threads but it is not the only possibility.

Between the class Client and Queue there is a relationship of aggregation because a queue is a vector of clients in our case and clients can exist without a queue.

And between Generate and Queue again an aggregation relationship. The class Generate uses an array of queues and generates a random client. Generate also has an aggregation relationship with client because of this.

Between the GUI and the Generate and Queue classes is a dependency relationship in both ways because they use attributes, methods and objects from one another to work.

* 1. **Packages**

For the current implementation, all classes have been saved in one package. The GUI class and the rest of the classes are easier to understand if they are all in one package, because they are tightly connected between each other.

* 1. **Algorithms**

Multithreading refers to two or more tasks executing concurrently within a single program. A thread is an independent path of execution within a program. Many threads can run concurrently within a program. Every thread in Java is created and controlled by the Thread class.

Each tread is an instance of class thread. The implementation of threads in a class can be done in 2 ways: the first one by implementing the interface Runnable or by extending Thread (that implements Runnable).

To create a thread, you have to follow the next steps:

* create a class that extends the Thread class
* override the method run() from class Thread
* create an object of type thread using new
* you can start the newly created thread by calling the method start(), also inherited from class Thread

An example of implementing the thread class:

public class Example

{

public static void main(String args[])

{

ThreadClass th = new ThreadClass();

th.start();

System.out.println("Return to main");

}

}

class ThreadClass extends Thread

{

public void run()

{

for(int i=0; i<10; i++)

System.out.println("Step " + i );

System.out.println("Run is done");

}

}

* 1. **User Interface**

The User Interface was built using Java Swing elements (buttons, panels, frames, labels, textfields). The interface can be called user-friendly because it needs only the most basic of actions and knowledge to make it work, the inputs will be read from textfields that have labels with their individual meaning positioned so that they cannot be mistaken. While starting the application, making it simulate a queue, is done by pressing a button called “start”.

A window will open when running the application. This window will allow the user to introduce the minimum and maximum arriving time, the minimum and maximum service time, the number of queues and the simulation interval in textfields appropriately-named using labels. After introducing the desired data the user will be able to press a button called “Start” to start the simulation of the queue. The Simulation will be able to be visualized in 2 textareas bellow the start button, each textarea having different information about the simulation. The first textarea contains the log of events, it shows whenever a client arrives at or leaves a queue, and specifies which client and queue.

1. **Implementation and testing**
   1. **Implementation**

The application has been implemented in Java Programming language, using Eclipse. For the GUI, all the buttons, textfields and other components were added by code without any “drag and drop” plugins. Listeners are placed for specific component to catch events (usually a simple button press), and respond accordingly.

All the implemented classes, including their methods and attributes were documented with comments to be easier to understand.

One of the hardest parts of the project implementation is removing elements of the queues, this was done in the following way:

**public** **synchronized** **void** deleteClient() **throws** InterruptedException

{

**while** (queue.size() == 0) // verifies if the queue is empty

wait(); // if it is empty it waits for the notify

Client client = (Client) queue.elementAt(0); //saves the removed element in order to print it later

queue.removeElementAt(0); // removes the beginning of the vector

// prints in the textarea every time a client leaves a queue

GUI.*logs*.setText( GUI.*logs*.getText() + "Client " + Integer.*toString*(client.getClient()) + " left Queue " + queueID + "\n");

// wakes all threads in the waiting set

notifyAll();

}

// overrided method of the Thread class

**public** **void** run()

{

//try - catch structure to avoid interruption errors

**try**

{

**while** ( **true** ) // the condition is always true because you want to delete as fast as you are allowed

{

// servTime is a random value between the minimum and maximum service time

servTime = (**int**) ( Math.*random*() \*( GUI.*st2* - GUI.*st1* ) + GUI.*st1* );

// sleep waits for the indicated amount of seconds before going to the next line

*sleep*( servTime \* 1000 );

// deleteClient deletes according to FIFO behaviour

deleteClient();

}

}

**catch** (InterruptedException e)

{

System.***out***.println("Intrerupere");

System.***out***.println(e.toString());

}

}

* 1. **Testing**

The testing in this homework is quite different from the previous one where it was required to do the testing using JUnit. There is no mandatory way of doing the tests, so I chose to do them using try-catch structures and by printing errors if any occur during the input or execution part of the application.

Errors appear for leaving certain inputs as their initial values, 0, like the number of queues or the simulation interval (because it would make no sense otherwise). Also, there are some conditions that have to be met, like the maximum should always be bigger than the minimum.

1. **Results**

Through hard work and intensive testing, I have been able to create an application for Queue Simulation.

The final result looks something like this:

1. **Conclusion and future developments**
   1. **Conclusions**

At the beginning, it seems hard because our knowledge of threads is practically zero. But as you start to understand them and use them it starts to get easier. A big help was the examples given in the laboratory, a simple example that makes you understand the idea of queues and threads and how they work.

* 1. **Future developments**

There are many improvements that could be made to this application, for example the graphical interface could show animations of clients arriving at queues and leaving them. There could be introduced a database of products and the number of products to determine the service time.

1. **Bibliography**

* Example and usage of vectors in java:

<https://docs.oracle.com/javase/7/docs/api/java/util/Vector.html>

* To draw the diagrams:

<https://www.gliffy.com/>

* Examples of how to create and use threads:

<https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html>

* Answers to questions regarding JAVA syntax:

<https://stackoverflow.com>

* Great example of a project that shows the implementation of a queue:

<http://users.utcluj.ro/~crisb_pop/TP2017.html>

* Explanations for diagrams:

<https://www.tutorialspoint.com/uml/uml_standard_diagrams.htm>