Threads

Programming Techniques – Third Project

Group: 30424

First\_Name: Daiana

Second\_Name: Spatacean

Teacher: Claudia Pop

Summary

1**. Introduction**

1.1 Problem specification . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2. **Description of the project**

2. 1 Problem analysis . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

2. 2 Modeling . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

3. **Projection**

3. 1 UML diagrams . . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . .

3. 1. 1 Use-case diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . .

3. 1.2 Class diagram . . . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . .

4**. Implementation**

Packages . . . . . . . . . . . . . . . . . . . . . . . .. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

5. **Implementation** **and testing** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

6. **Further developments .** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

7. **Conclusions** . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

8. **References**  . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

1. **Intoduction**

Java's creators have graciously designed two ways of creating threads: implementing an interface and extending a class. Extending a class is the way Java inherits methods and variables from a parent class. In this case, one can only extend or inherit from a single parent class. This limitation within Java can be overcome by implementing interfaces, which is the most common way to create threads. (Note that the act of inheriting merely allows the class to be run as a thread. It is up to the class to start() execution, etc . ) .

public class Counter extends Thread

{

public void run()

{

....

}

}

**2.Description of the project**

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.

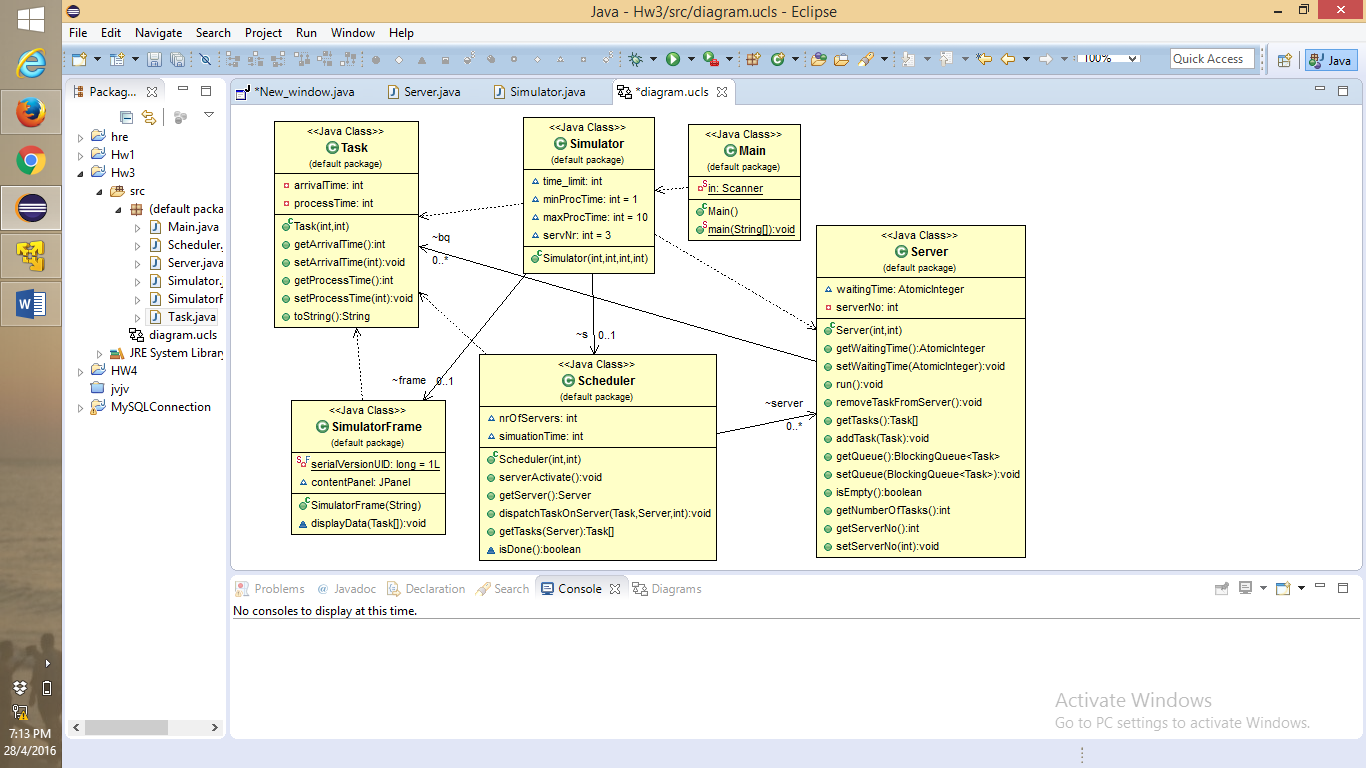
**3.Diagrams**

**3.1 Use-case diagram**

A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different [use cases](https://en.wikipedia.org/wiki/Use_case) in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well.

The user can be any person that wants to use the application and the interface must be pretty friendly. In this case the use case diagram is when the user inputs from the keyboard the values for the: min process time, max process time, simulation time and numbers of servers.

**3.2 Class Diagram**



**4.Implementation**

This project is made following the OOP structure(classes, methods) and principes:

- **inheritance** when one object acquires all the properties and behaviours of parent object i.e. known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism

- **polymorphism** when one task is performed by different ways i.e. known as polymorphism. For example: to convense the customer differently, to draw something e.g. shape or rectangle etc.

- **abstraction** hiding internal details and showing functionality is known as abstraction. For example: phone call, we don't know the internal processing

- **encapsulation** binding (or wrapping) code and data together into a single unit is known as encapsulation. For example: capsule, it is wrapped with different medicines.

5. **Implementation** **and testing**

The simulation time , minimum processing time , maximum processing time ,the number of servers are given by the user from the keyboard . After this in the console of the project will appear the output of the project .

My project has only one package. This package contains all the classes from the project.

* 1. **Class Task**

**public class Task**

* This class has only two private variables, which are vital for each task and they are: the time in which the task arrives and the time in which it is processed and getters and setters for each variable
  1. **Class Scheduler**

**public class Scheduler**

* As variable in this class I have a list of servers, the number of servers and the time for the simulation
* I have a method called **serverActivate(),**  which starts a thread for each server
* Another method called **getServer()** which searches for the server with minim number of tasks because when a task arrives it must be placed on the servers where is the minim numbers of tasks because like at the store when we want to pay for our products that we want to buy we are interested to waist not much time when we get and pay for our products
* the method **dispatchTaskOnServer()** splits the tasks on the write servers
  1. **Class Server**

**public class Server implements Runnable**

* here is a blocking queue where the task generated is put here and it waits to be processed to know in which server to go so it will have a minimum time to wait its turn
* in the run method, which is overridden I take each task from the blocking queue to be processed
* also I have a method which adds the tasks and one which gets the tasks
  1. **Simulator**

**public class Simulator implements Runnable**

* this class has a method called is done which

* The **user** should:

Regarding the implementation process, I used as program Eclipse IDE. During the implementation of the project I made a lot of changes. My program has only one package. The classes had many methods. I also start the project not based on OOP principles. Before the interface was made I print the results in the console to see if they are correct and to know if I continue in that manner or I had to make some changes.

* 1. **SimulatorFrame**

**public class SimulatorFrame extends JFrame**

* This class was made for the interface of the project but I didn’t manage to do it. The only thing that is visible is a server and is not correct I think.
  1. **Main**

**public static void main**

* Here I put the simulation time, minim process time and maxim process time and the number of servers and the user can input from the keyboard the dates that he or she wants.

5. **Results**

I am pretty satisfied with my work and I think that the project is very easy to understand and the most important thing is that any type of user is able to use the interface and will have no problem with it. And in the end I think that I implemented all the petitions of the “client”.

6. **Further developments**

I think that I can make a very long list with the things that I want to improve related to my project with the polynomial calculator:

* starting from the “back” of the project: I like to make more operations like multiplication, division and so on
* I also want to implement multi-threading
* More OOP with more classes and method with less lines of code and not so many duplicate code in some
* I want to have the structure for the packages: Model( with all the operations, term class and so on ), GUI( with the interface ) and Controller( in which the model and the GUI are tied )
* I definitely want to make an interface for this project like when we go to the supermarket and we wait to pay for the food that we buy
* It will be nice if the size of the window can be modified from the user

7. **Conclusions**

To conclude, I can say that this project meant hard work, a lot of new things learned, focusing, development and creativity. Even if I encountered a lot of problems, I was able to fix them after all, by searching on the internet or asking a colleague for advice. I think that my application satisfies the requirements and the users will have at their disposal all its functionalities.

8. **References**

* <https://en.wikipedia.org/wiki/Polynomial>
* <http://stackoverflow.com>
* <http://www.oracle.com/technetwork/articles/java/index-137868.html>
* <http://www.tutorialspoint.com/java/java_data_structures.htm>

And others!

.