**Queue system**

***-Project documentation-***

Student: Cocoş Tudor-Adrian

Group: 30423

**Contents:**

1. Objective of the assignment
2. Problem analysis
   1. Assumptions
   2. Modelling
   3. Scenarios
   4. Use cases
3. Design
   1. Design choices
   2. UML diagram
   3. Class design
   4. Algorithms
   5. User Interface
4. Implementation
5. Testing
6. Results
7. Conclusions
8. Bibliography

**1) Objective of the assignment**

The main objective of this assignment is to design and implement a simulation application aiming to analyse queuing-based systems for determining and minimizing clients’ waiting time.

The secondary objectives are the steps needed to be taken to achieve the main objective. First of all, we need to analyse the concept of shop queues and what do they imply in terms of registers with queues, cashiers which process the clients and, of course, the clients, which have individual needs and amounts of shopping. After that we need to break the elements of the problem into implementable features like classes and methods needed. The next steps include class design, method implementation, User Interface development and testing. Finally, we evaluate the results and think about further improvements.

**2) Problem analysis**

*a) Assumptions*

Given the assignment, we assume that a user must input the number of customer that will go in the shop in a given time-span, the simulation interval, the number of registers (queues) available in the shop, the maximal length of the queue at one time (may there is a space constraint in the shop that doesn’t allow long queues), the minimal and maximal arrival time at the queue (an interval from which we will get the time when the customers that were shopping go to find a queue) and in a similar fashion the minimal and maximal servicing time (that implies a period span from which the individual servicing time at the register for each client will be determined).

*b) Modelling*

To make the data as simple as possible to comprehend and follow step by step, each client is characterized by its id, arrival time and servicing time (or processing time). All of these are simple integers. In a similar way, each queue will be identified using an id.

*c) Scenarios*

The main scenario and the perfect one is when all the inputted data is correct, and the program may start to function. Otherwise the user will be prompted to change the inputted data and try again. The faulty input data may reside in choosing the number of queues greater than five as this is a limitation given by the UI. Another thing that may go wrong is when the user inputs a maximum arrival time combined with a maximum service time that exceeds the simulation interval because that may be like a client that gets late to the shop and while the store is closing he expects to be serviced by the cashier at the register. Something that the program allows is when even if the queues are working at full capacity and it’s closing time, the customers that were at the queue will be serviced.

The main scenario is also broken into several smaller ones given by the multiple choices that can be made when inputting data at the beginning of the program. These could be scenarios of shops with a small number of register (one or two), shops having a limited space inside for clients to queue up (given by the number of clients per queue inputted as a limit for the queue), clients that show up closer to the opening of the shop or closer to the end of the store’s opening hours or of clients that may have a few items to buy and by doing so they would have a minimum servicing time where the scanning of the produces is almost negligible and the main factor is the payment processing speed (which is constant for most customers) and finally the scenario of clients with filled shopping trolleys or baskets that have a large item processing and for which the payment time is almost negligible.

*d) Use cases*

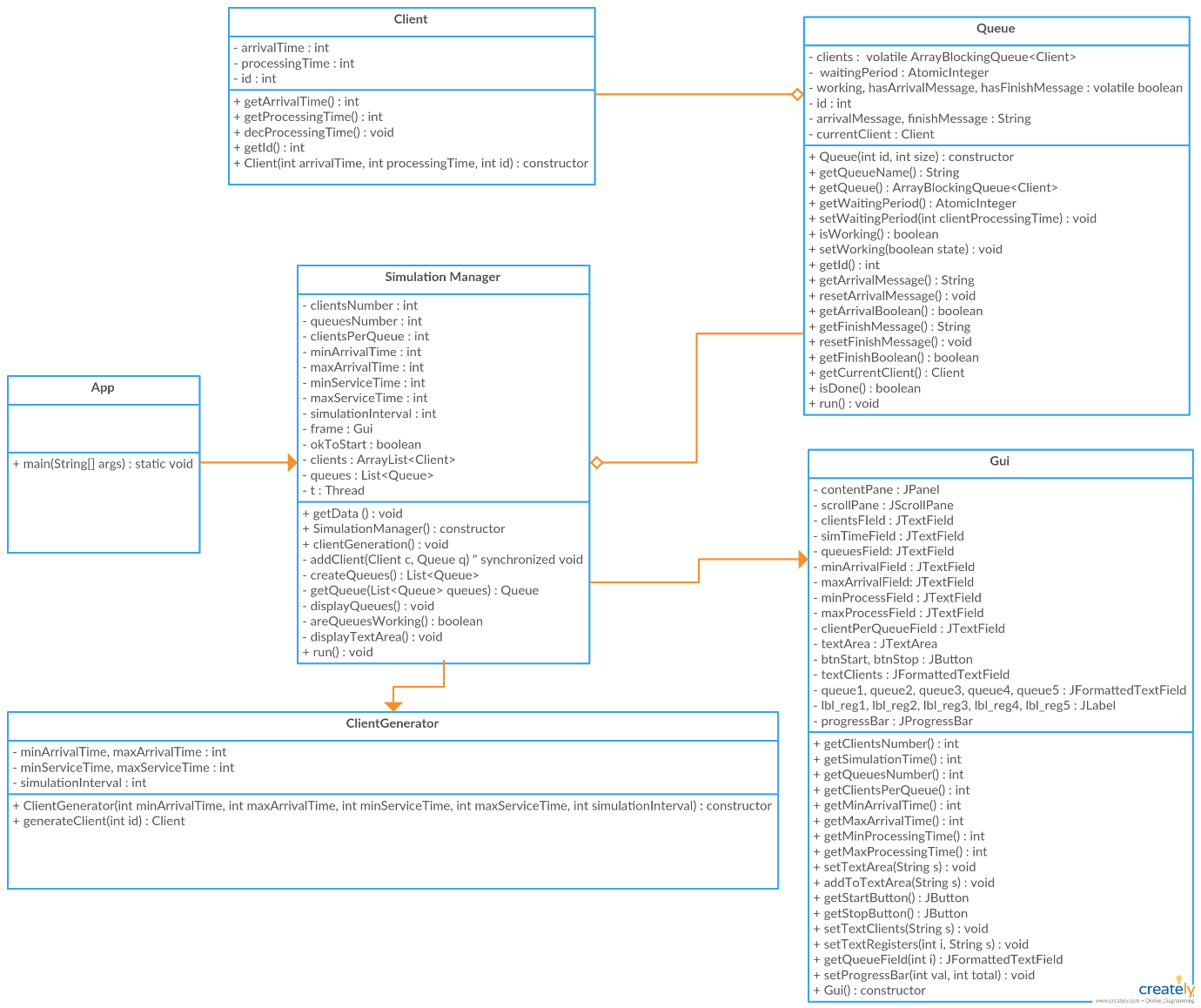
As seen previously in the Scenarios section, the application may be used to simulate various types of shops and their customers. It may be used to increase the sales by being able to service more customers if the number of queues may be a problem. Where the number of registers is not a problem, maybe another use case may be to optimize the schedule of the cashiers by observing the part of the opening hours where there are the most customers in contrast to parts of the day when there are very few. These are just a few ideas of what this program is able to do.

**3) Design**

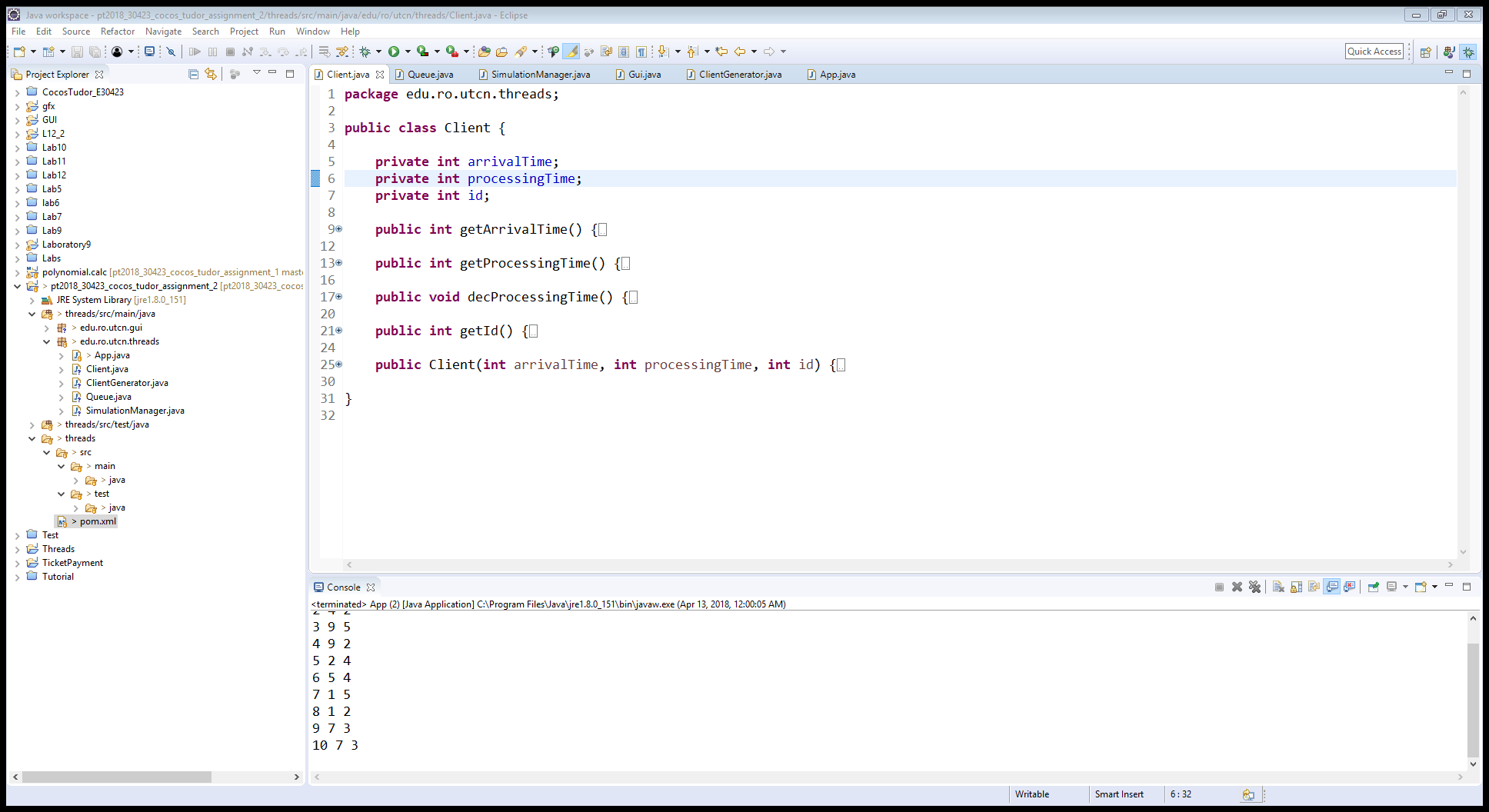
*a) Design choices*

We start class design with the Client class as from an “atomic” point of view, this is the smallest part of the whole simulation environment (the actual shop) that the clients will fill. Each one of the clients presents an id, an arrival time at the queue and a servicing period. Up next is the Queue class. Each queue will be defined by its own id just like the registers in a store that have their own number. A queue consists mainly of the list of clients that it is made of and added to this there is also a waiting period defined for each one of the queues. The waiting period is defined by the sum of all the processing time intervals of the clients at the queue and will be used to find the most suitable queue for a new client to be attached to. To be able to simulate a working cashier that services a series of clients, each Queue is also an individual Thread essentially by implementing Runnable. After that the Simulation Manager was created which will act as a controller of the whole application. It also implements Runnable and it will be the main Thread of this program. Here, the list of clients will be generated, when the time is right they will be placed at the right queue, it will supervise the evolution of each client in the queues and display the according messages in the UI or change different parts of the UI depending on the evolution of the application. To do that, this class will have a list of the queues, a list of the clients that are still in the shop and an entity of the GUI. To be able to generate the clients out of the inputted data, the Client Generation class is used which using Random will return clients based on the intervals given by the minimum and maximum of arrival or servicing time. The GUI class contains all the graphical elements that are displayed and used in the program and finally, there is the small App class that is used to start the whole application.

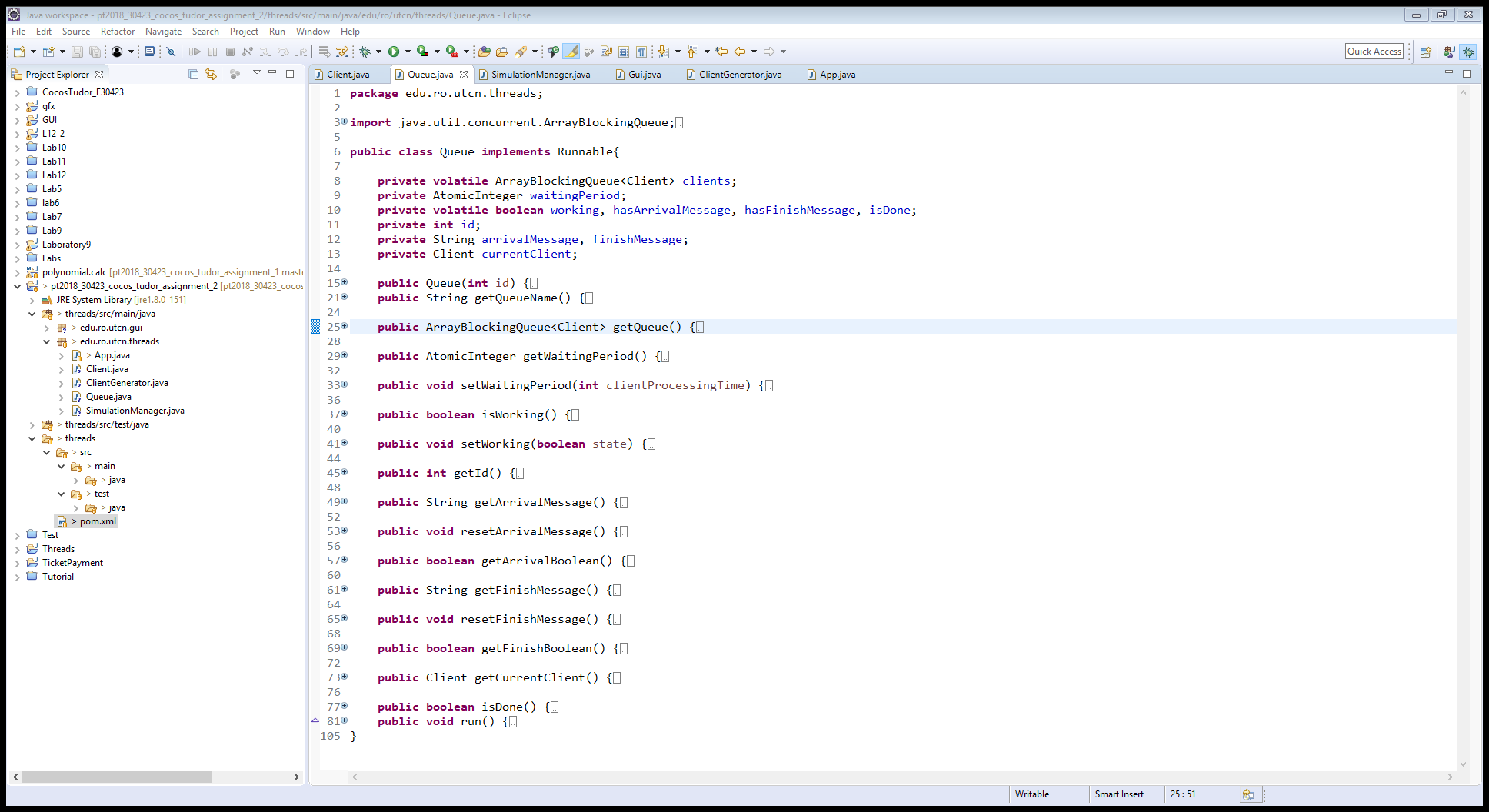
*b) UML Diagrams*

**

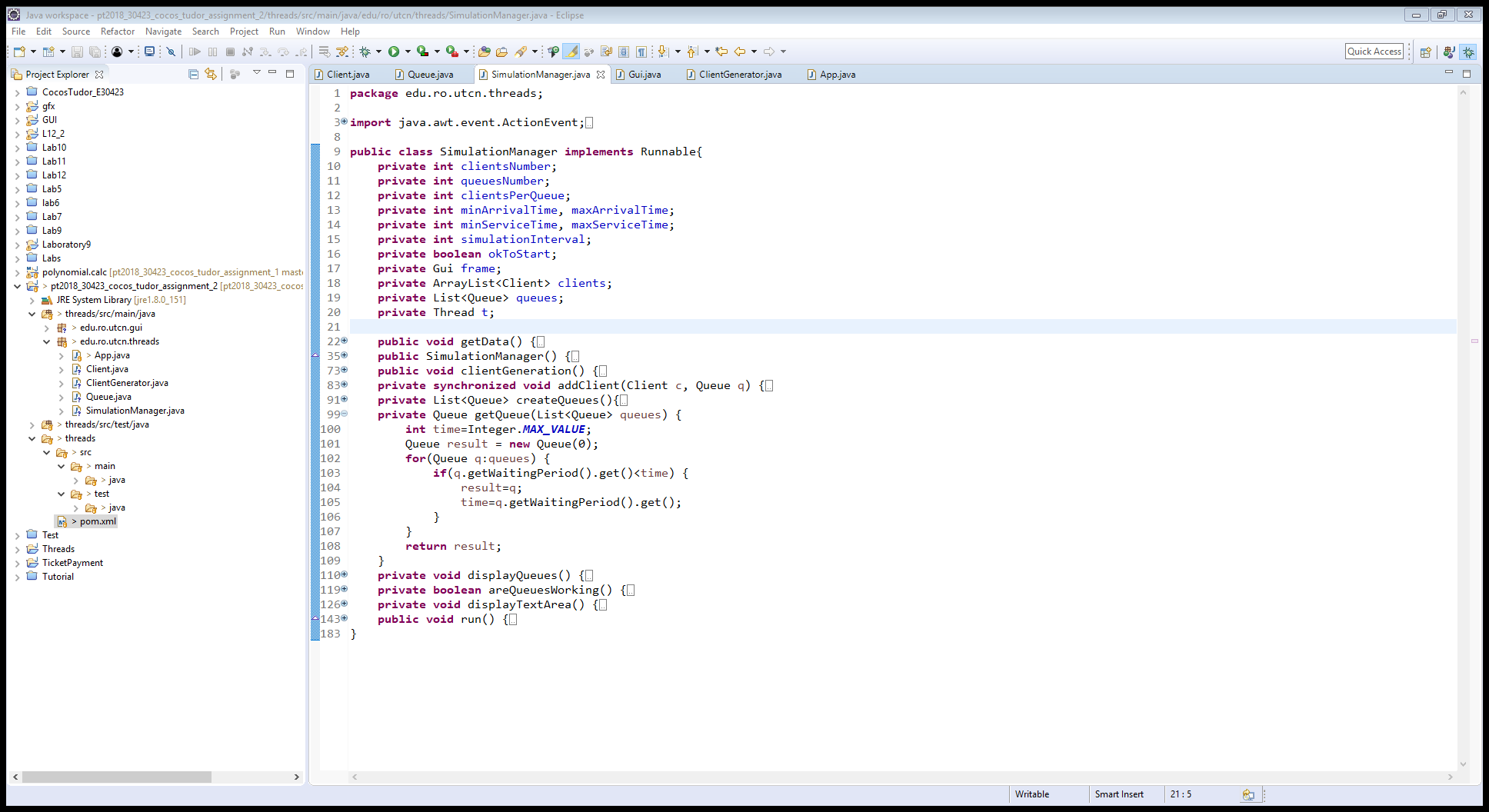
*c) Class design*



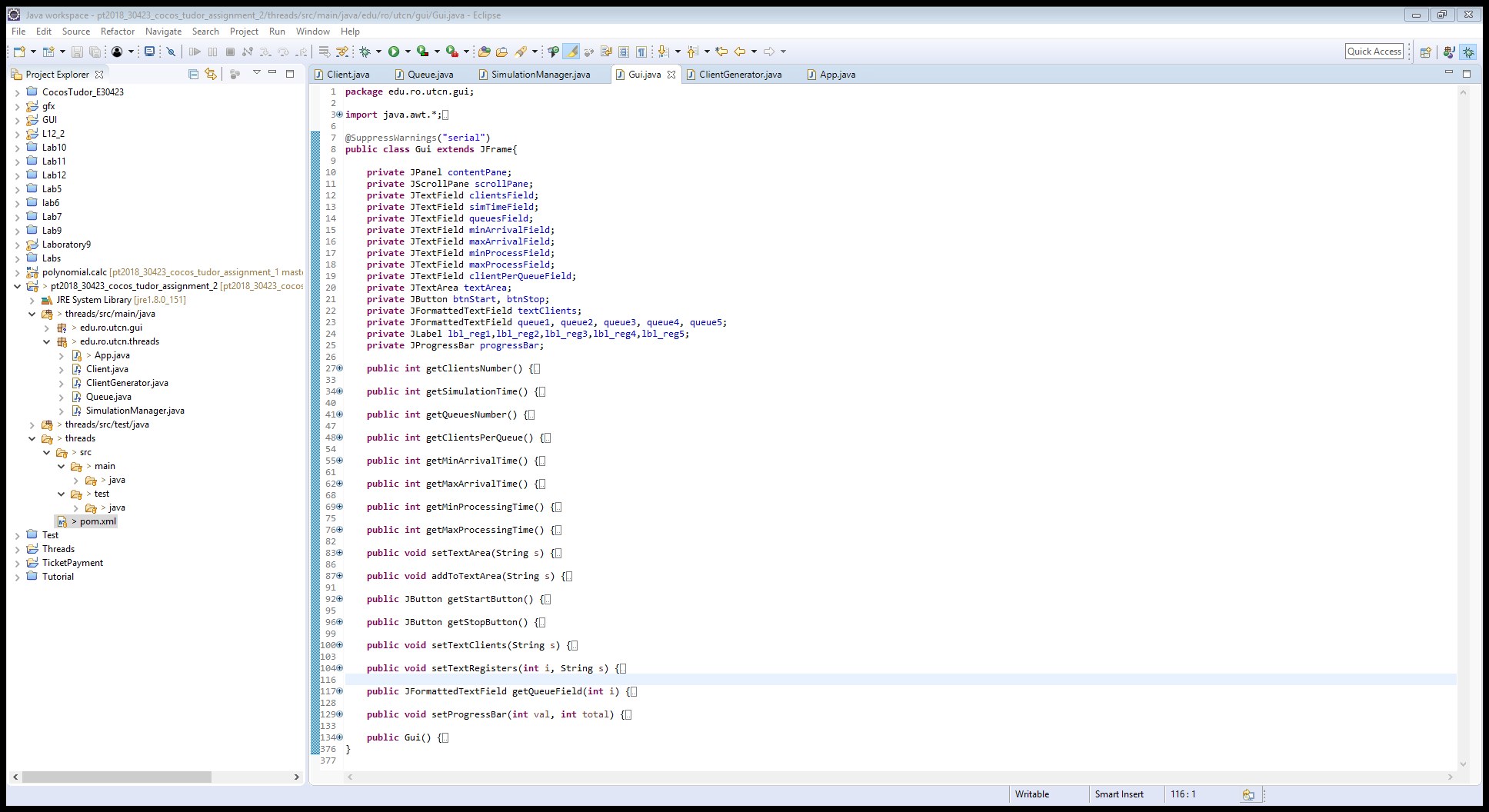
***Client class***

******

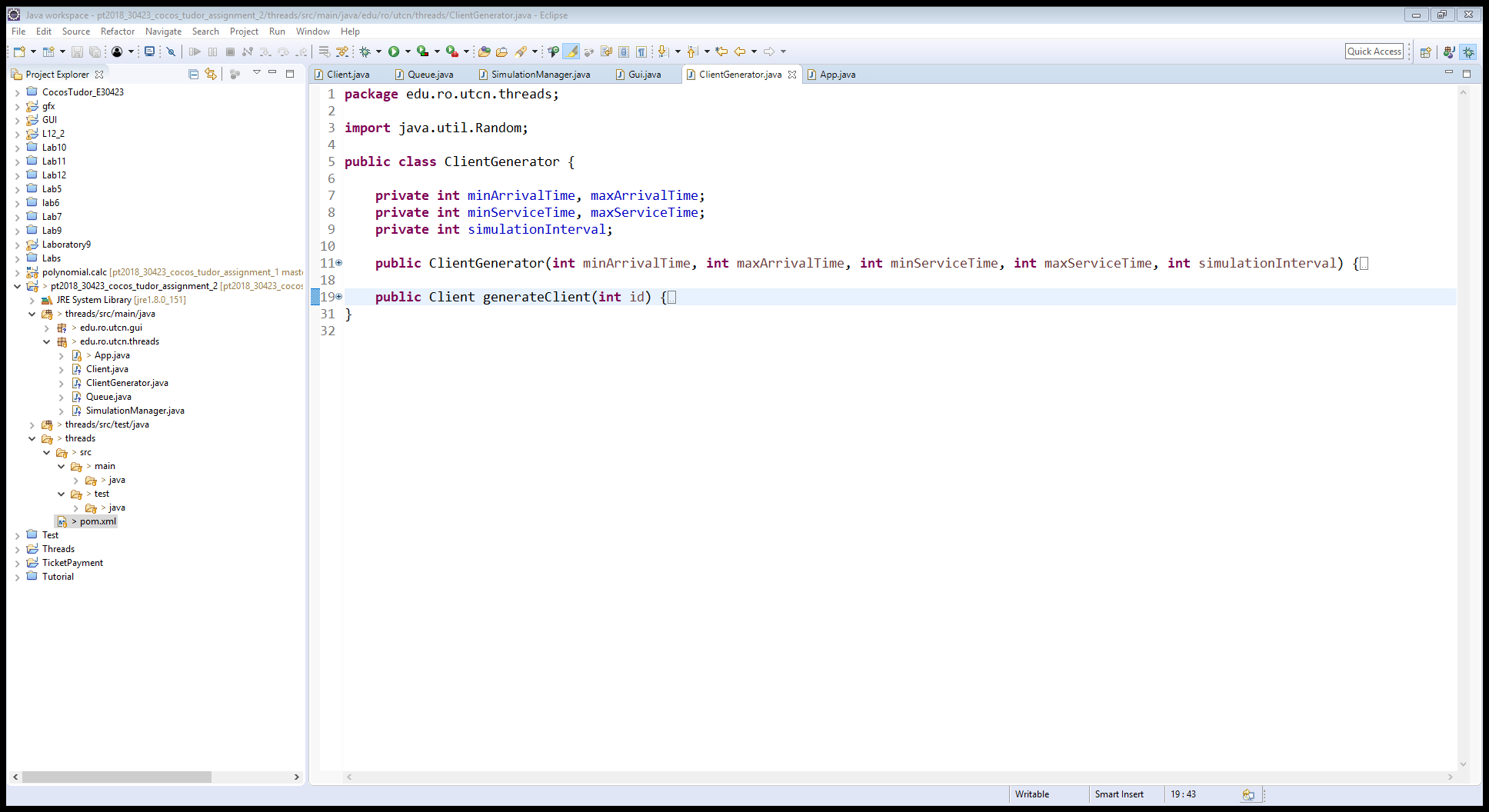
***Queue class***

******

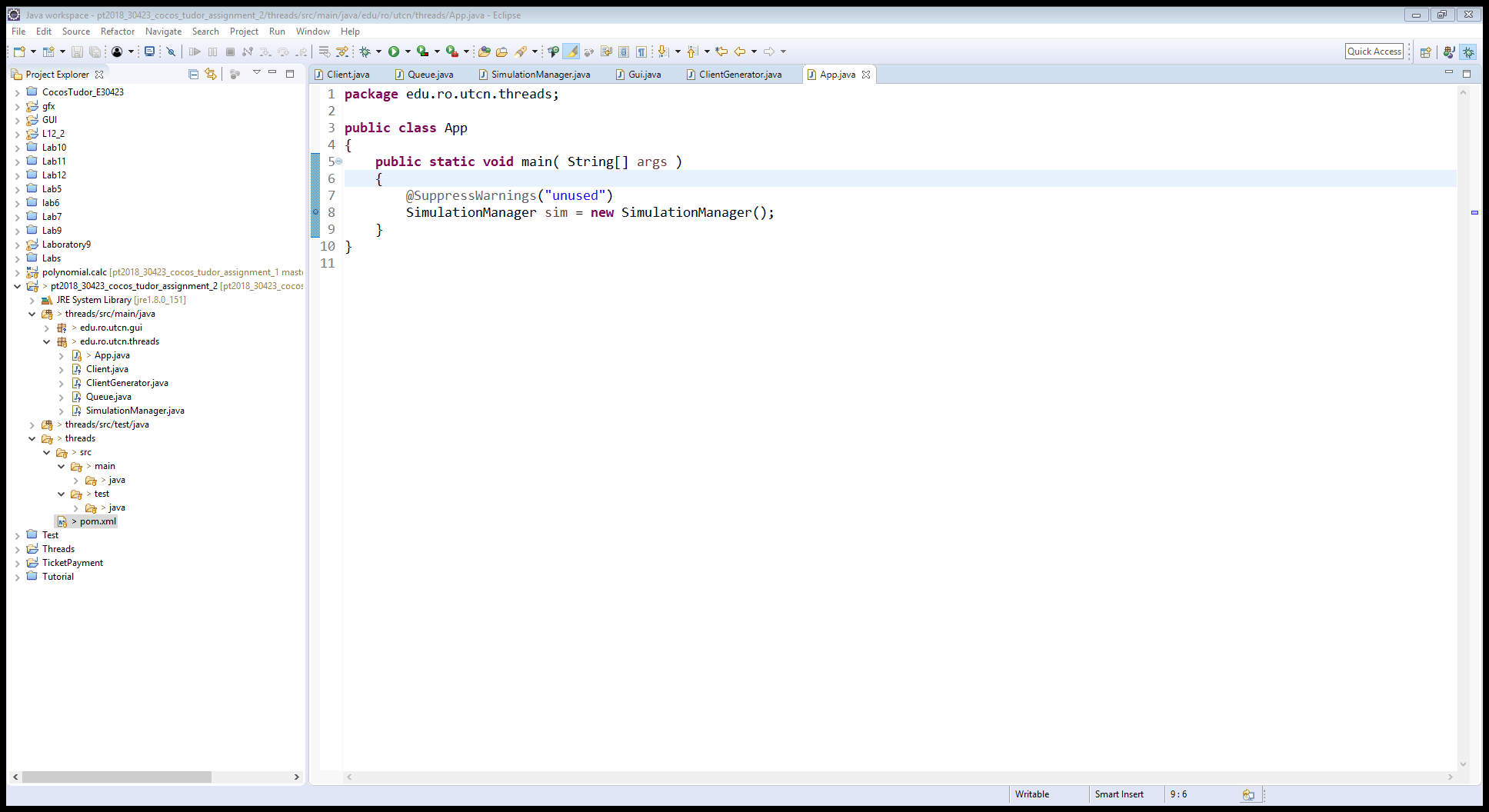
***Simulation Manager class***

******

***Gui class***

******

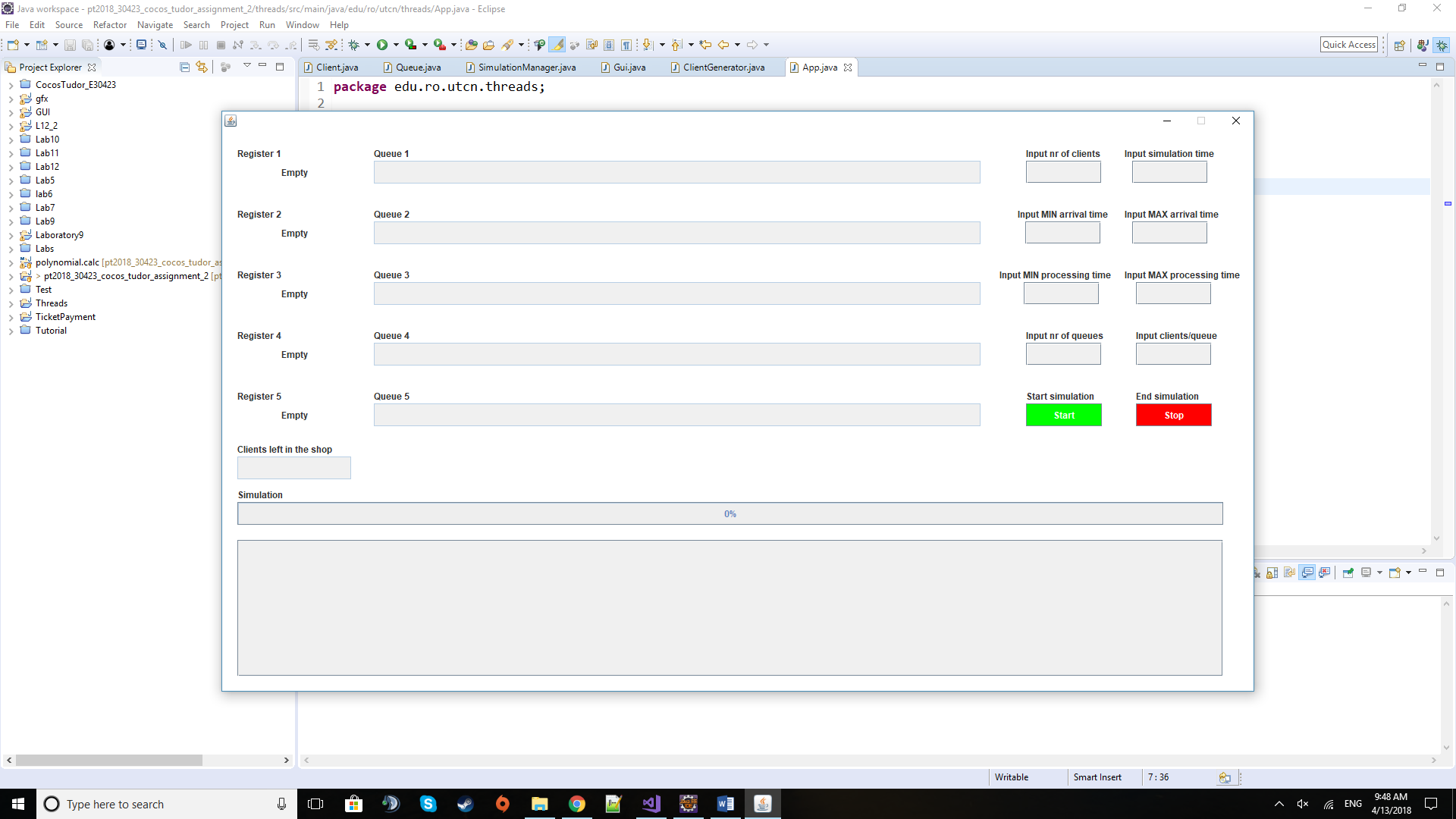
***Client Generator class***

******

***App class***

*d) Algorithms*

As for algorithms there are no special mathematical-based algorithms like in the previous assignment. The usual ones imply iterating through lists of clients or queues and applying some kind of “treatment” to each member of those lists. The operations that are performed vary from popping the first element of a queue and by taking its service time, freezing the thread queue for that period, to adding new clients to queues or generating them in the first place.

*e) User Interface*

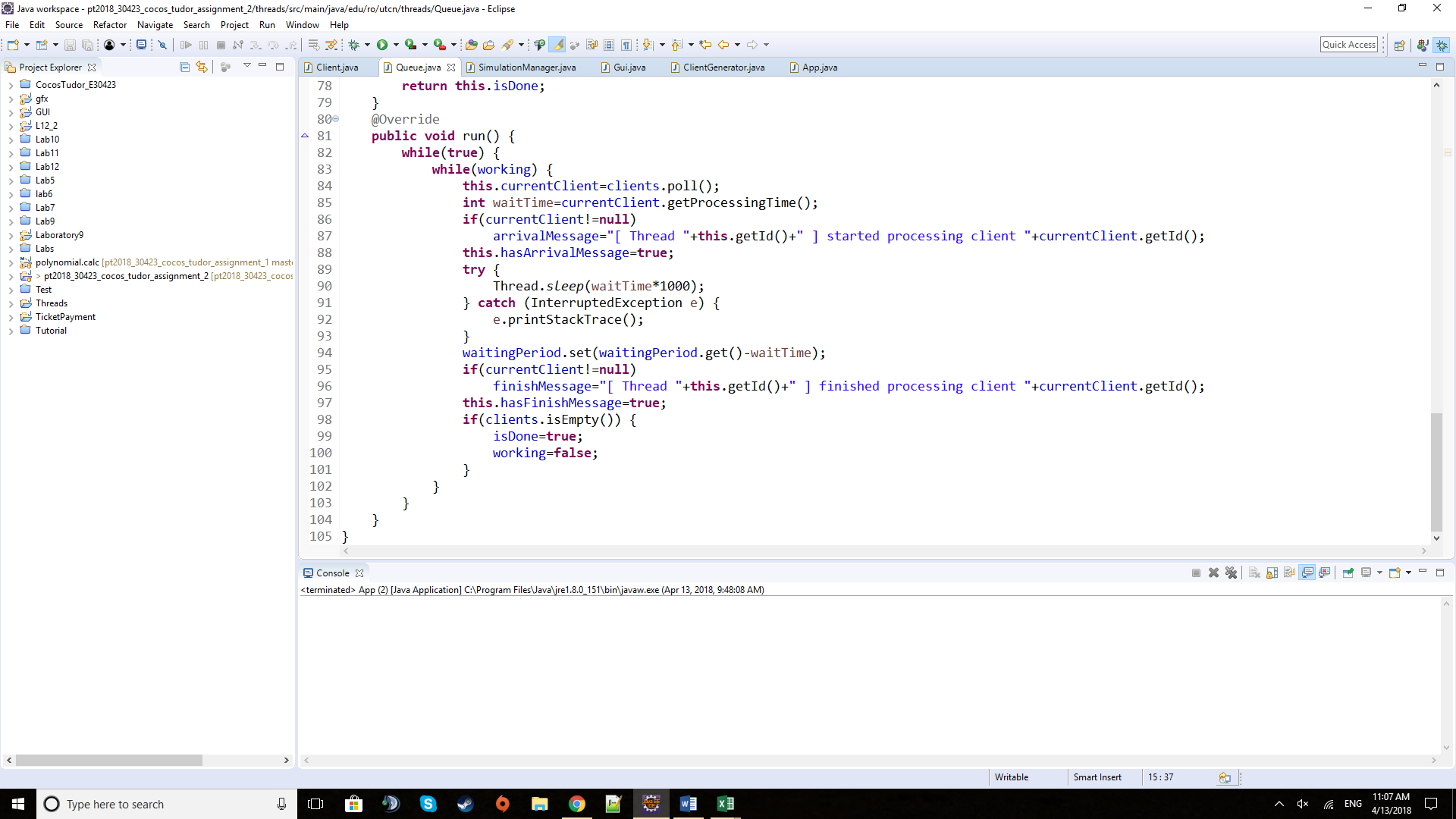
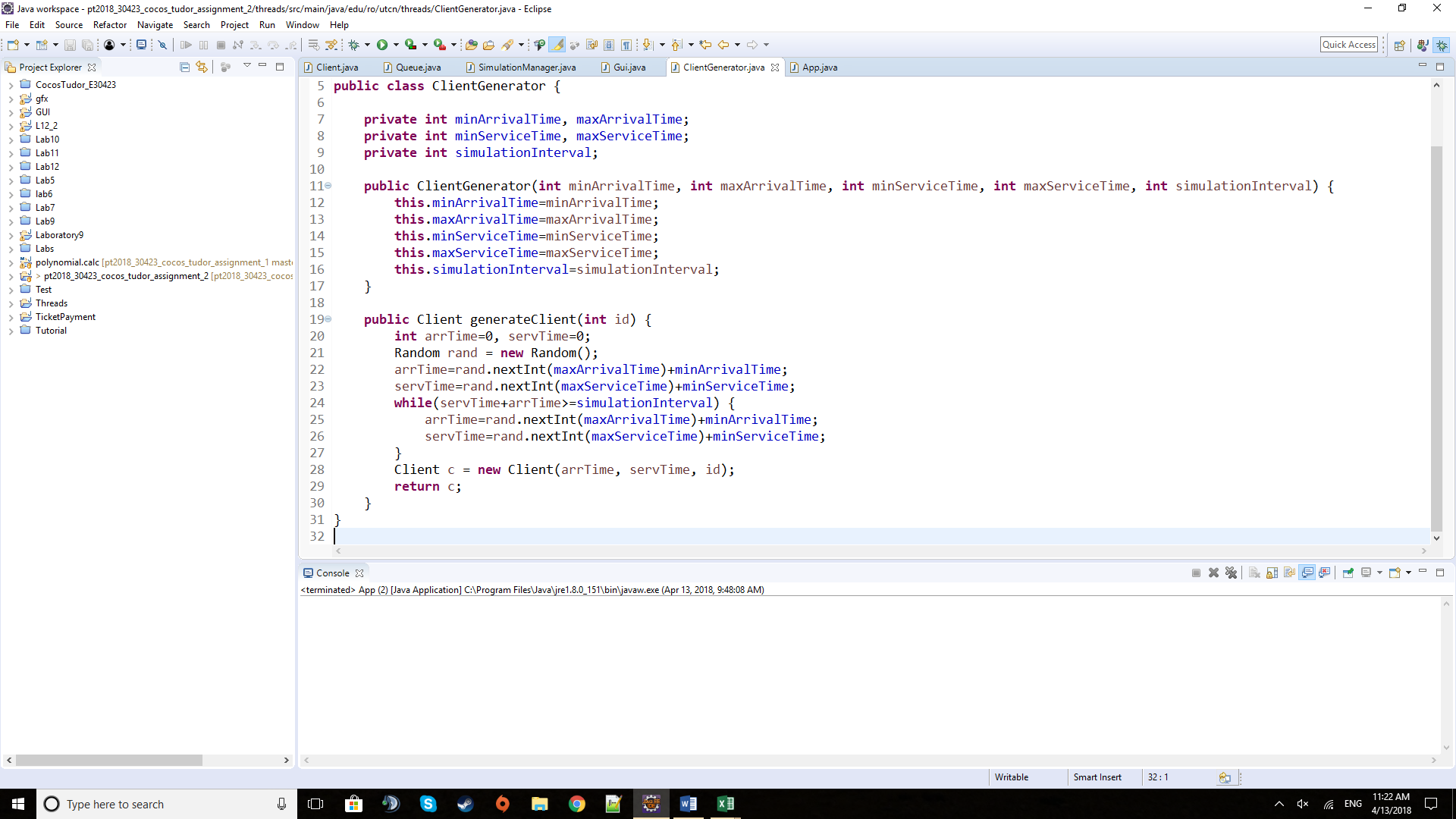
This is the window that the user is prompted to when the application is started. It contains several types of graphical elements, each with its own specific use. Starting from left to right, coming from the top, there are 5 register labels with their own label designated to show the current client being processed at the register. If there is none, then the label will show “Empty”. To the right of them there is the visual representation of the queues with their labels respective labels at the top. Each queue field will be filled from left to right with the clients that await to get to the register to be processed by the cashier. Further to their right there are 8 fields for the input data each with their own label to tell the user what data needs to be inputted. Below these ones there are the two buttons used to start or suspend the simulation with their pairing descriptor labels.

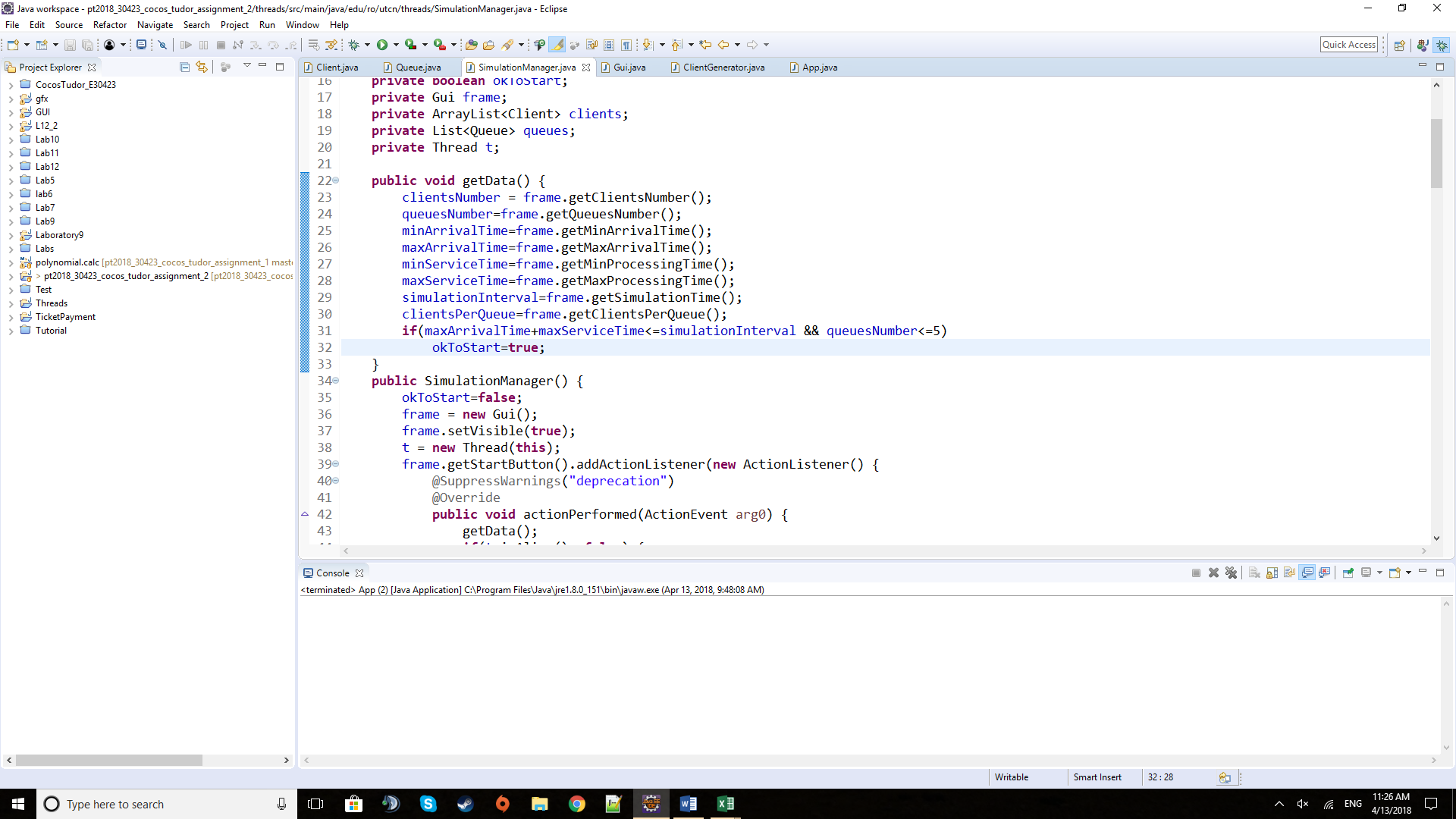
Under the previous section there is a field that displays the clients that were not assigned to a queue yet, meaning the clients that have not done their shopping yet or have no queue to join because they may be filled. Following that, there is a progress bar that runs alongside the simulation displaying the passing of the time while the program is running.

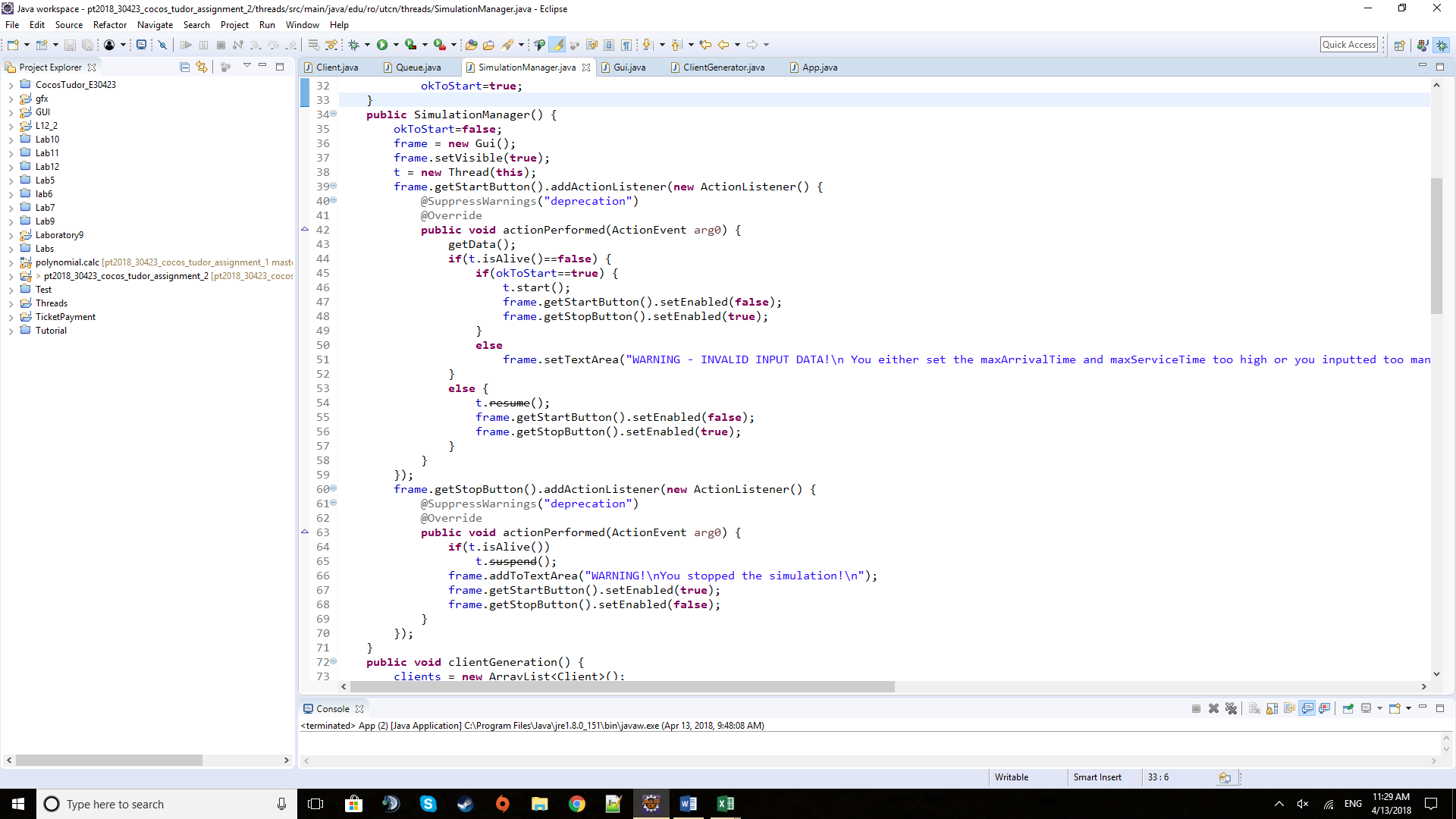
Finally, at the bottom there is the Text Area which display all the events that happen like a client x arriving at the queue y or a queue finishing processing a certain client and also relevant information like warnings if the inputted data is not valid and needs to be inserted again.

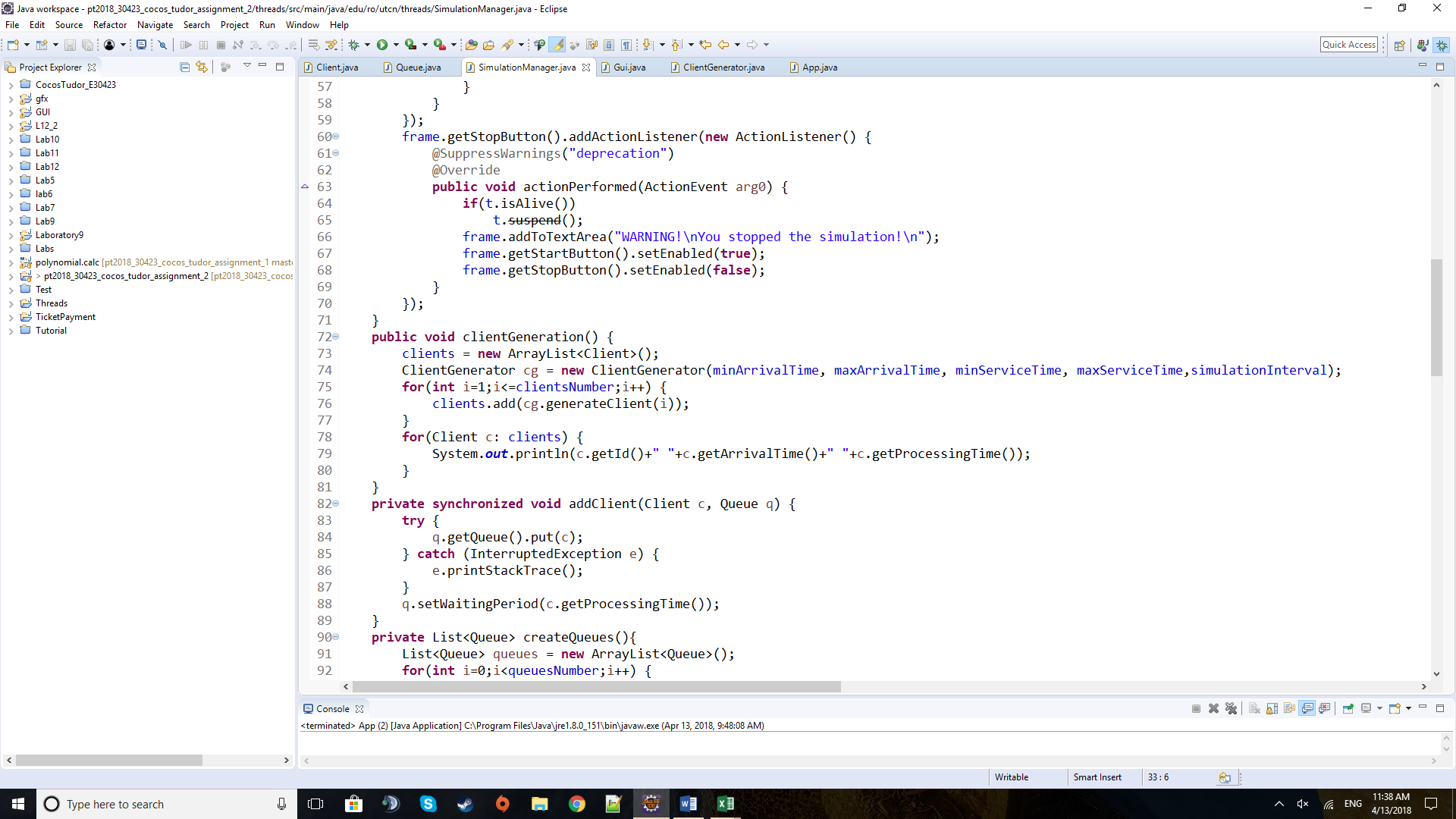
**4) Implementation**

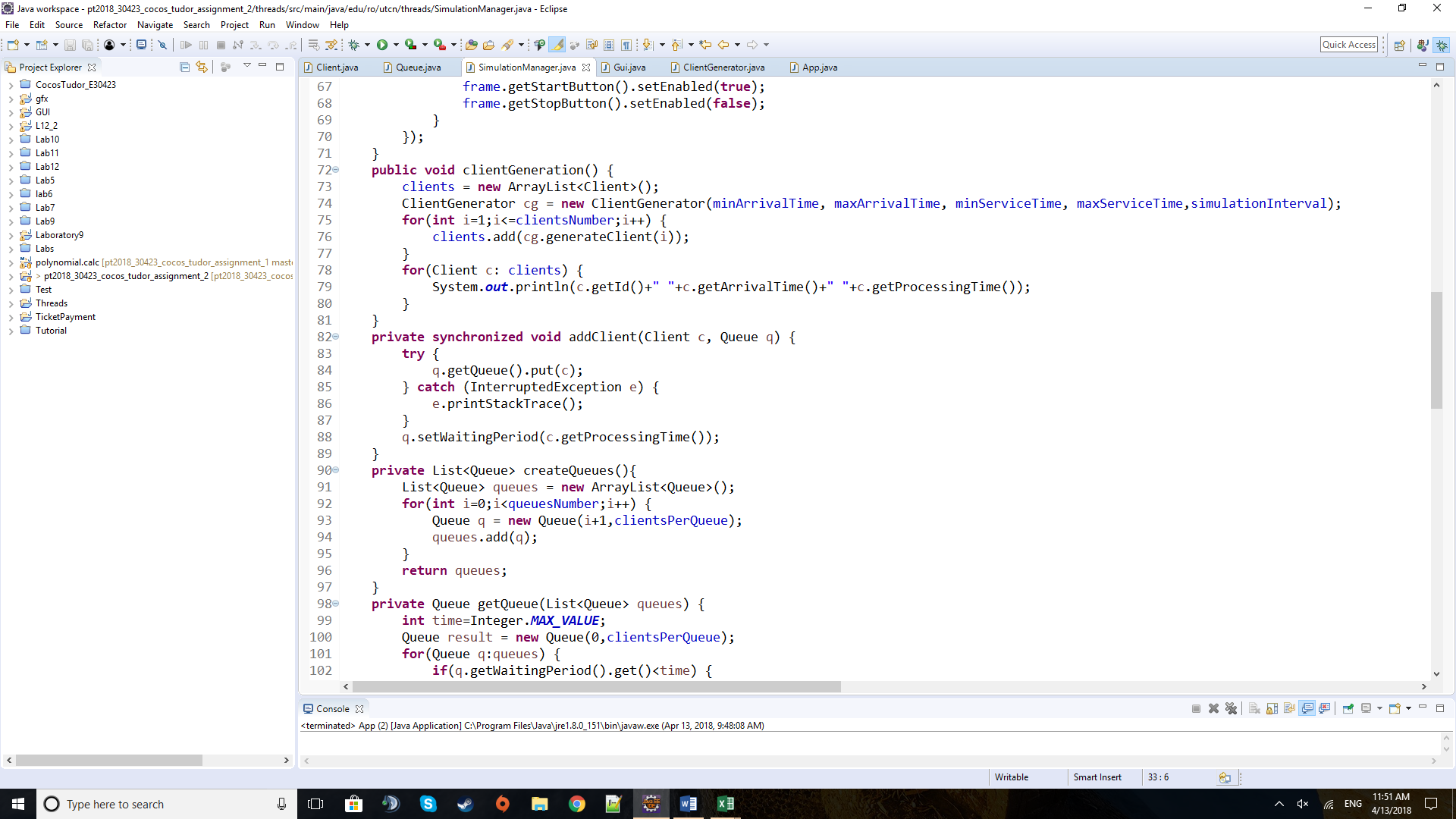
I will describe here the most relevant pieces of code and especially those that are not that common and straight-forward (unlike getters, setters, and some basic constructors).

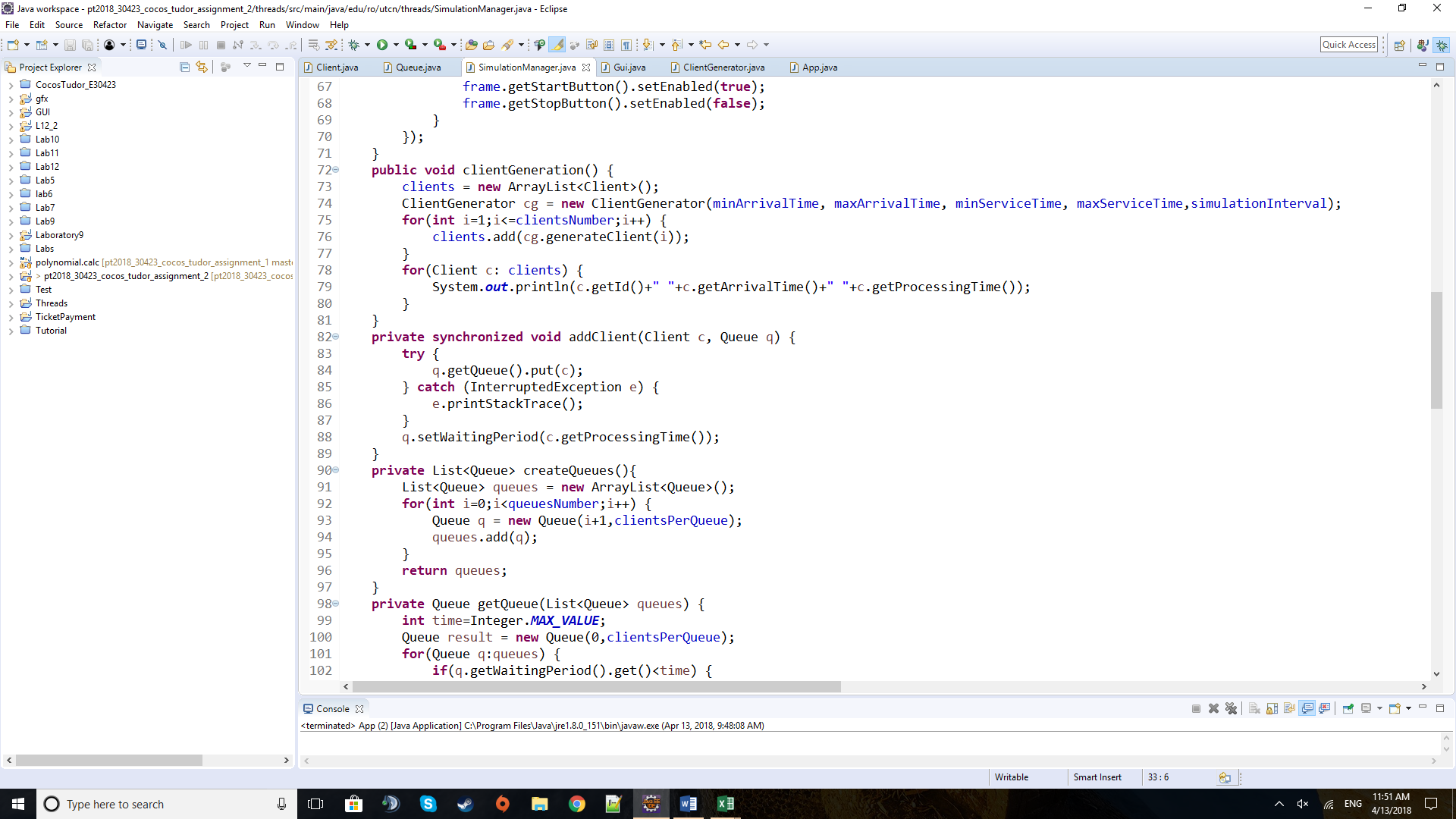
 Above is the run method of the Queue class. It contains the actions that each queue will perform on their own thread. It starts by popping the first element of the list of clients in the queue. If it is a valid client, it will create the message that will be displayed as event. Then the queue will sleep for the duration of its processing time and at the end of this waiting period, it will subtract the servicing time of this client from the total waiting time for the queue. It will also build the message for finishing a job. If the list of clients is empty, meaning that there are no clients in the queue, the work of the method is halted. It is technically still running in order not to lose the thread by killing it but is not doing anything like a cashier that waits for clients at the register because that is his or her job until the store is closing (the simulation is ending). There are also two flags in the form of two boolean values that are used to notify the main thread that one of the two events, of client arriving or finishing, has happened in the queue, leaving up to the Simulation Manager to deal with the situation and decide what and where to display the information.

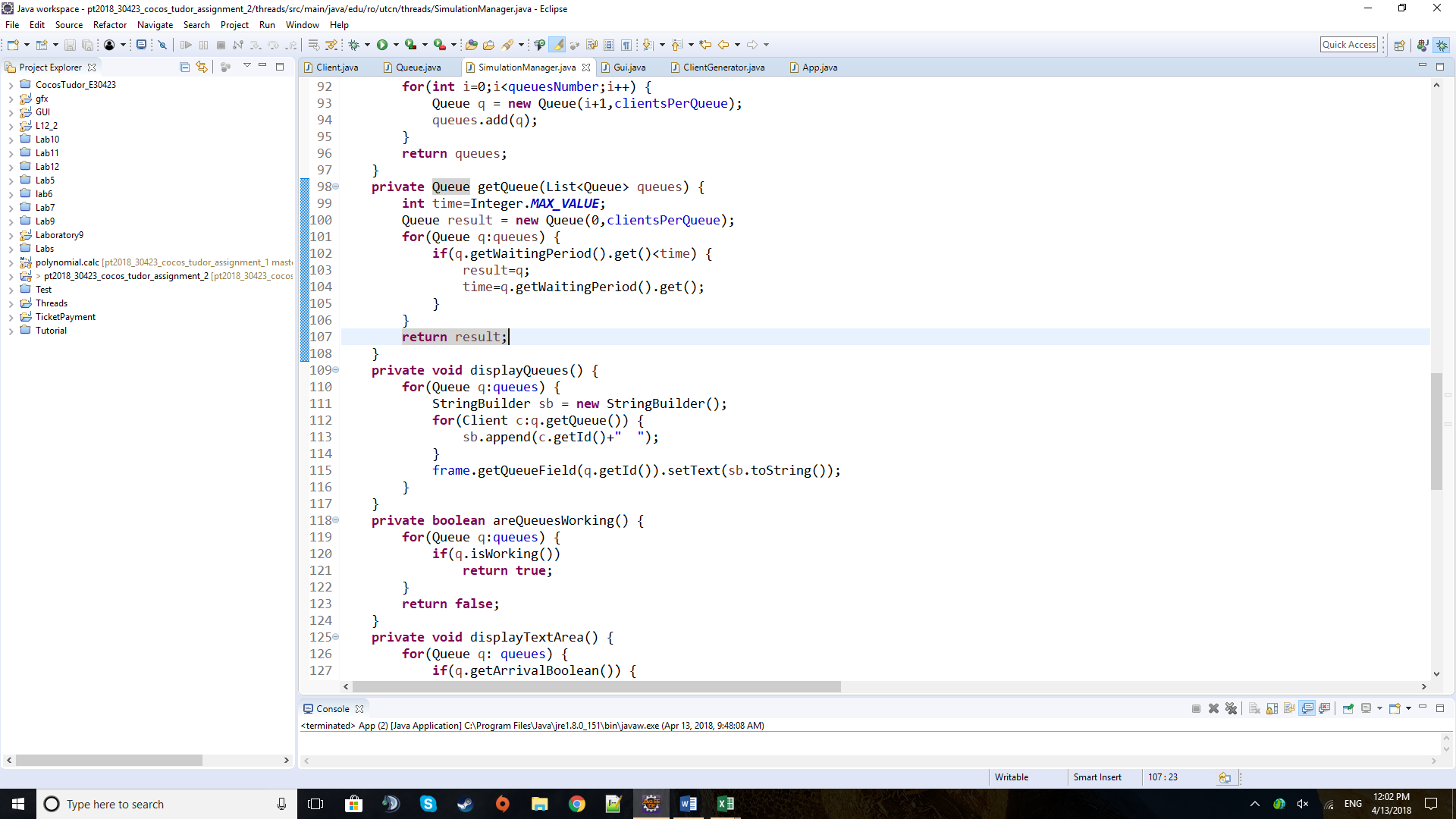
 The generate Client method above is part of the Client Generator class. It is using the time intervals for clients arriving at the queues and for their servicing time. It is, using random generated values into the given intervals, constructing the new client that the method, afterwards is returning. The values are checked to be in the simulation interval.

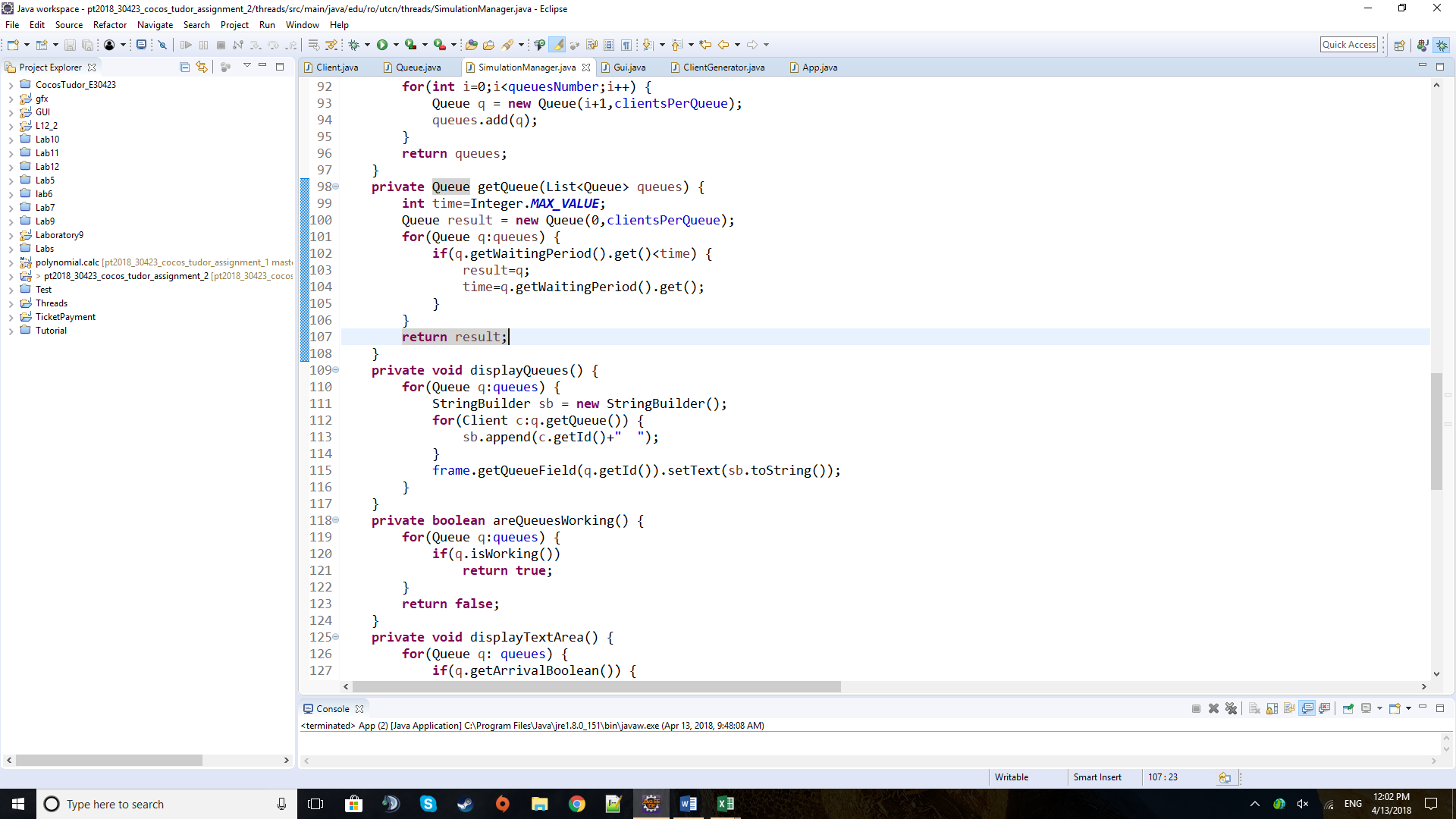
 The get Data method is found in the Simulation Manager class and is responsible with reading the input fields and making a quick check of their values. If everything is alright it will provide the “go ahead” for the main thread to start running.

 The constructor method of the Simulation Manager class is the one that produces the GUI and the main thread that is running in this class. If the get Data method is not providing true to the boolean condition, the thread will not start once the Start button is pressed. Otherwise, by pressing the Start button, it will start the main thread, essentially starting the simulation. Complimentary to the Start button is the Stop button that suspends the thread is the user chooses to stop the simulation at a given time.

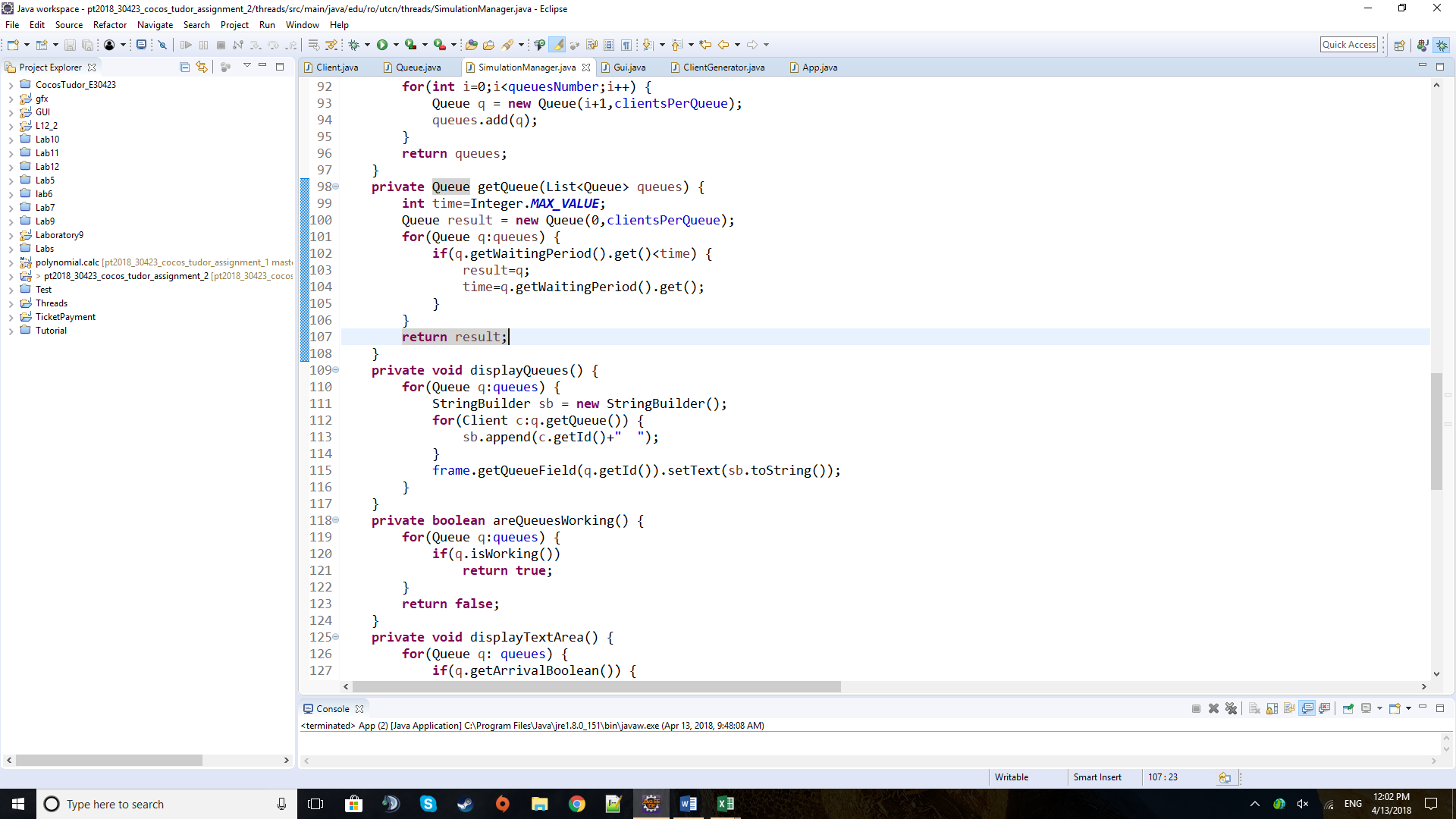
 The method above will create the list of clients that are found in the store, shopping, and from which the main thread will assign the clients to their queues when their arrival time at the queue comes. This list is being generate by using the Client Generator class with its method of generating a client with randomized values in the given intervals. For curiosity there is an iteration through the list to take a look at the clients generated and their values.

 The above method is used by the main thread of the Simulation Manager class to add a client from the list of shopping clients to the chosen queue, at the end of the list of clients of the queue. It will also set a new waiting period for the queue by increasing the previous one with its processing time.

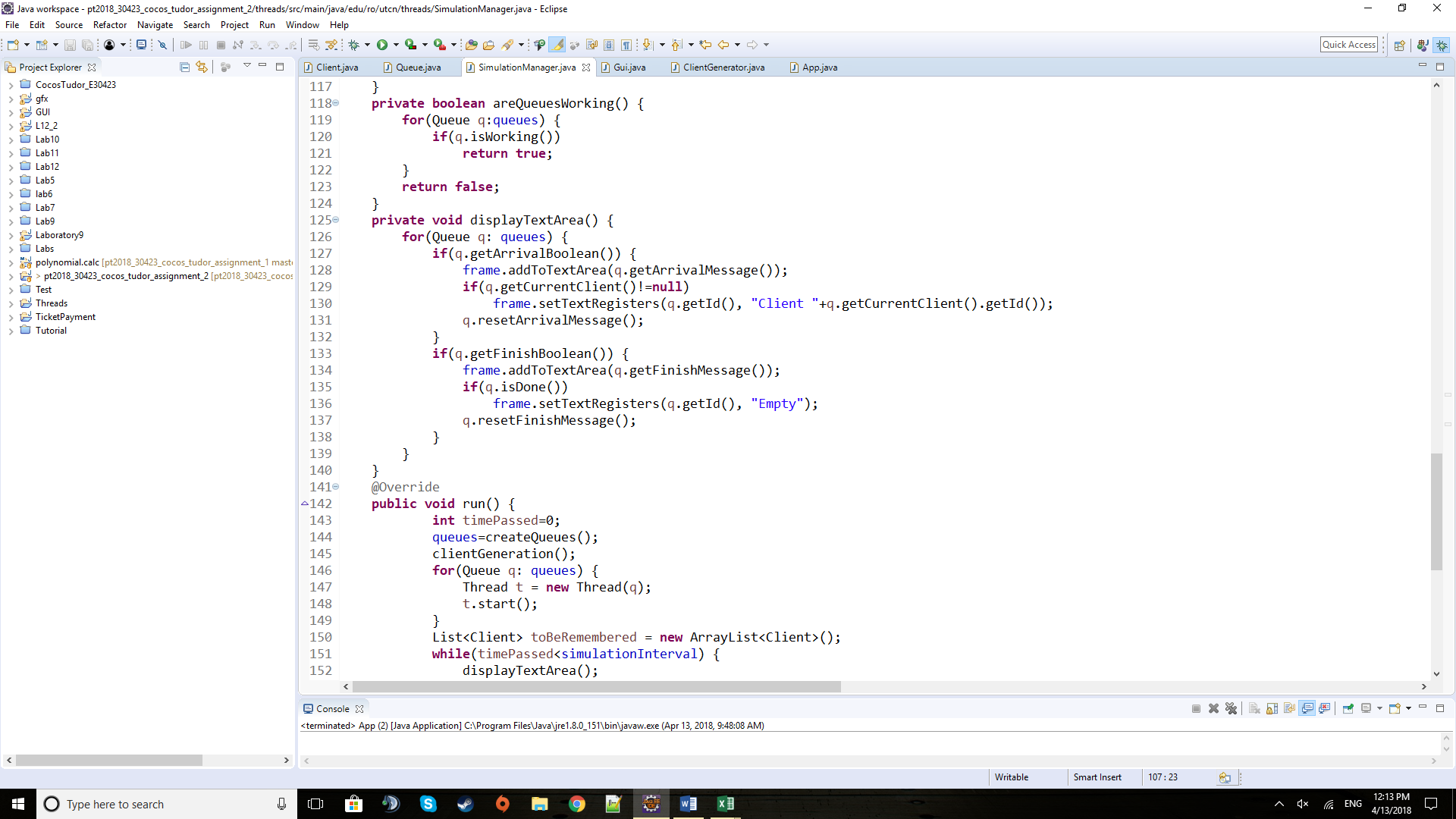
 The method of create Queues is used to return a list of queues for a given number of queues inputted at the beginning. It will construct the queues with an id and a maximum number of clients that the queue can hold.

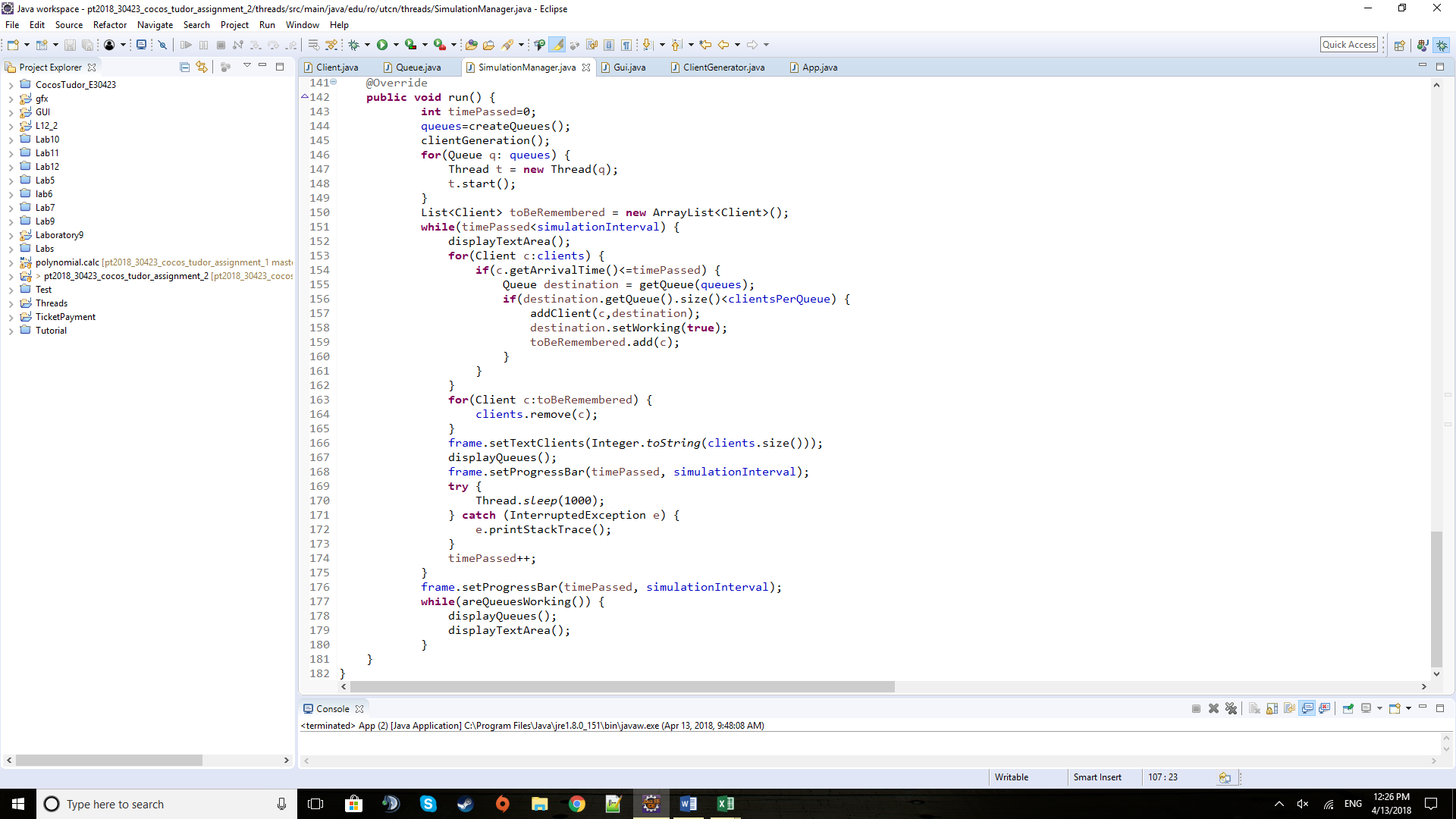
 This method, above, is taking the list of queues in the simulation and is iterating through all of them, searching for the queue with the minimum waiting period and returning the appropriate queue in the end.

The method of display Queues is going through all the queues of the simulation and, taking each client that is found in a queue, it is building the string to be printed. The message to be displayed is replacing the previous one in the appropriate field.

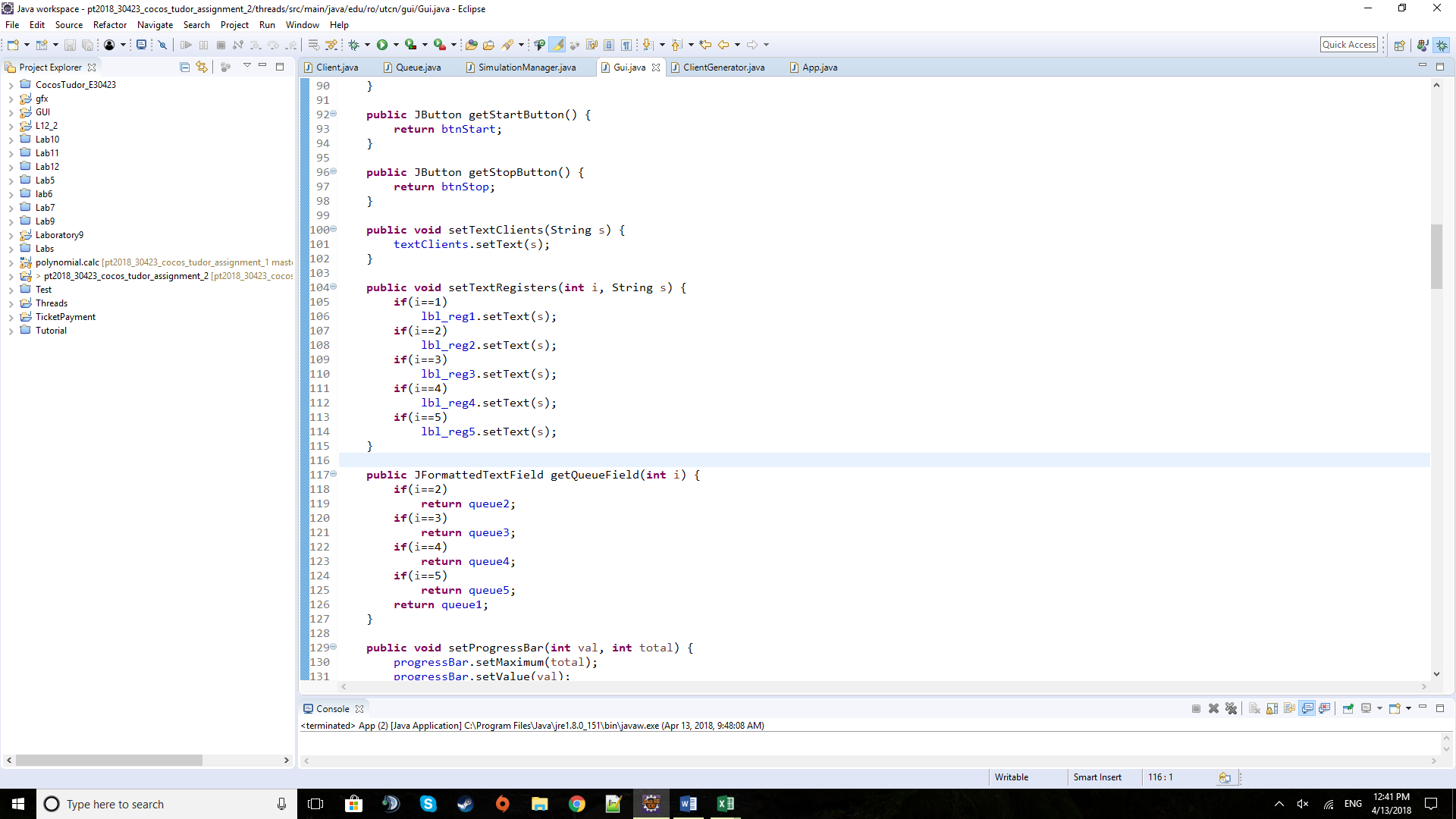


The method to the left is searching to see if one of the queues is still working and if at least one of them still does, then it returns true immediately. Otherwise it will return false at the end.

 The method above will be used to print the events that happen in each queue and the registers of the respective queues. If in a queue one of the flags of arriving / finishing customer service is raised, the method will print the message in the text area at the bottom. It will also display the current client at the cashier or if the queue is emptied and the queue has done its work it means that there is no client at the register and it should display “Empty” for it.



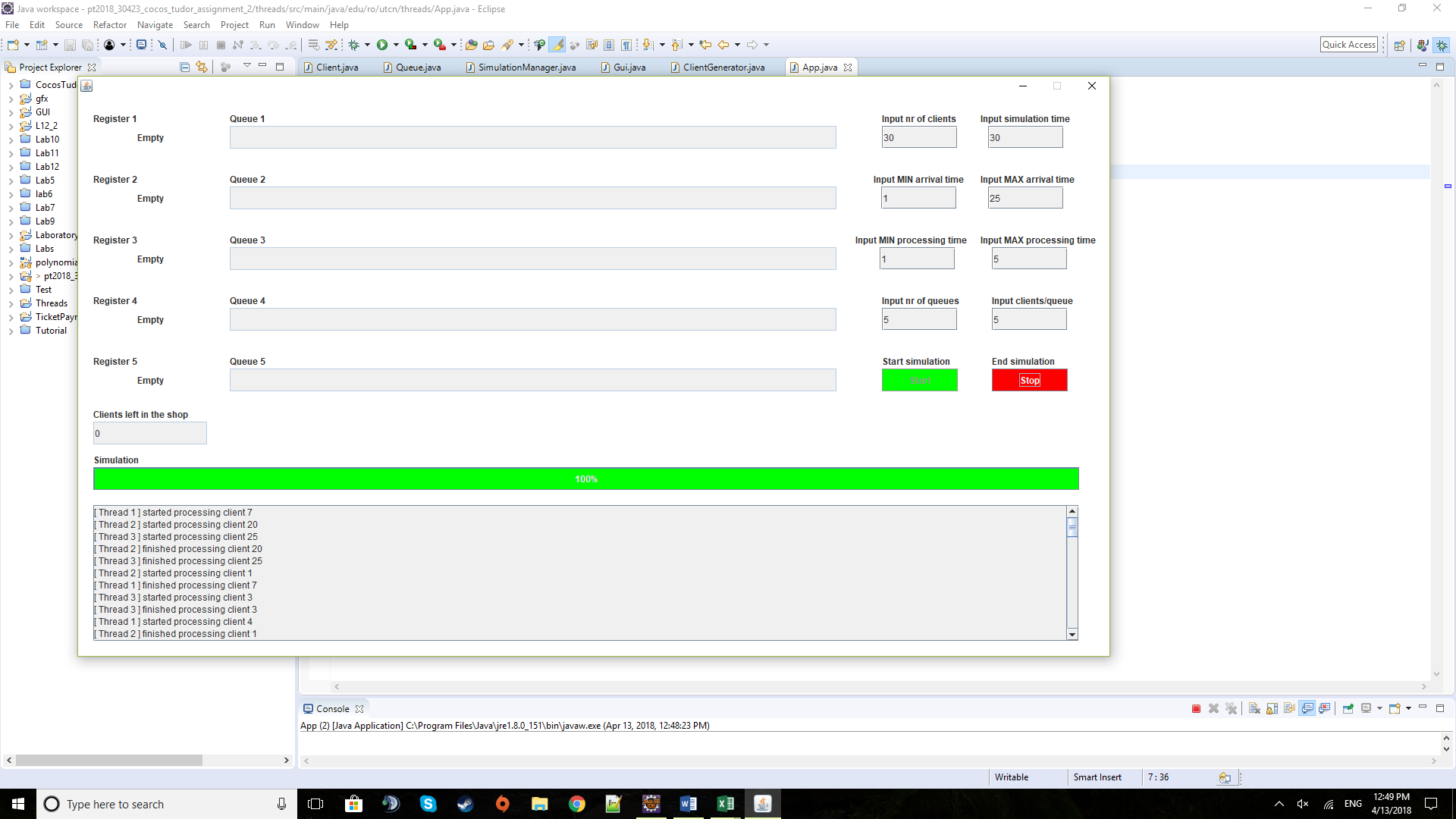
The run method of the Simulation Manager begins by creating the shoppers and by starting the queues, making them ready to receive clients as soon as possible. While running the simulation interval, it will iterate through the list of clients in the shop to check if there are any that need to be placed at the queue. If added, each one of them will be remembered in a separate list to be removed afterwards from the clients that are still shopping. After finishing the simulation interval, it will still run if there are queues that have clients waiting to simulate the scenario when the clients were waiting to finish their shopping, but the store is closing. It will not allow for any other clients to be added to the queues. For all the duration of the run method, it is also responsible to call the methods that display the relevant information.



The two methods above are part of the GUI class and are used to retrieve the appropriate graphical element to display a certain String or just to get the element as a separate entity.

**5) Testing**

The testing for this project was done by running the application and observing the evolution of the queues and the events that happen inside. The generation of the clients is verified by displaying them field by field for each one of them.

**6) Results**

The results of the simulation are available at any time during the running of the application, being displayed in the appropriate fields and also live in the event area. These events will be saved, and they will remain there even after the end of the simulation.

**7) Conclusions**

This assignment proved to be extremely useful to learn how the threads work in a program. It proved quite difficult at first as I had no knowledge of how they operate but as the work progressed so did my knowledge on the subject.

As a whole, the subject given is excellent for a large number of uses and scenarios. It can be used for clients and queues at a shop, tasks and servers in a computing complex, workers on assembly lines, etc.

**8) Bibliography**

Tutorial to understand the issue and start working on the project the following presentation was really useful: <http://coned.utcluj.ro/~marcel99/PT/Tema%202/Java%20Concurrency.pdf>

For any issues regarding exceptions: <https://stackoverflow.com/>

For details about the methods and fields of predefined classes: <https://docs.oracle.com/javase/8/docs/api/overview-summary.html>

For the UML diagram I used an online application: <https://creately.com/blog/diagrams/uml-diagram-types-examples/>