Technical University of Cluj-Napoca

Programming Techniques

Assignment 2

QUEUES MANAGEMENT APPLICATION USING THREADS AND SYNCHRONIZATION MECHANISMS

STUDENT NAME: CRISTIAN CASIAN-CRISTI

GROUP:30422

CONTENTS

1. Assignment Objective

Design and implement a queues management application which assigns clients to queues such that the waiting time is minimized. 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.

2. Dimension of the problem

2.1 Problem analysis

Queue managers are used a lot every day all around the globe. For example,a queue manager can be used in supermarkets to help keep check out time and efficiency maximum.

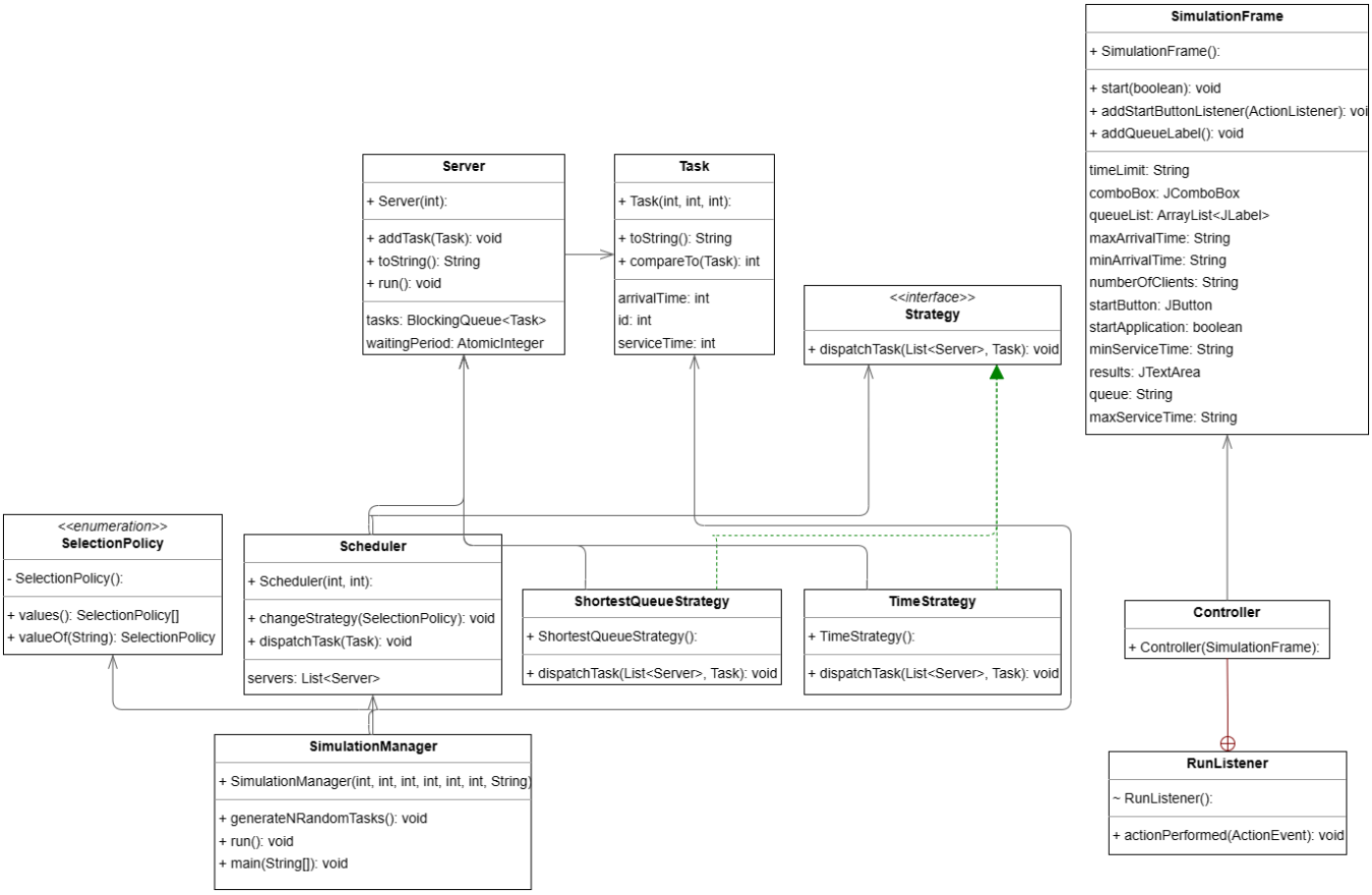
For this project, the queues will be composed of clients, which will a unique id, arrival time and service time.

2.2 Modeling the problem

The user will be able to input the details of how the simulation of the queue manager will run. They will need to input: the number of clients that will be randomly generated, the number of queues in which the clients will be put, the simulation time, minimum arrival time and the maximum arrival time, the minimum service time and the maximum service time, select one of the 2 strategies: time strategy which will put the clients in the queue with the shortest time waiting and the shortest queue strategy which will put the clients in the queue with the shortest least amounts of clients.

The running of the queue manager will be displayed in the logs area, in real time.

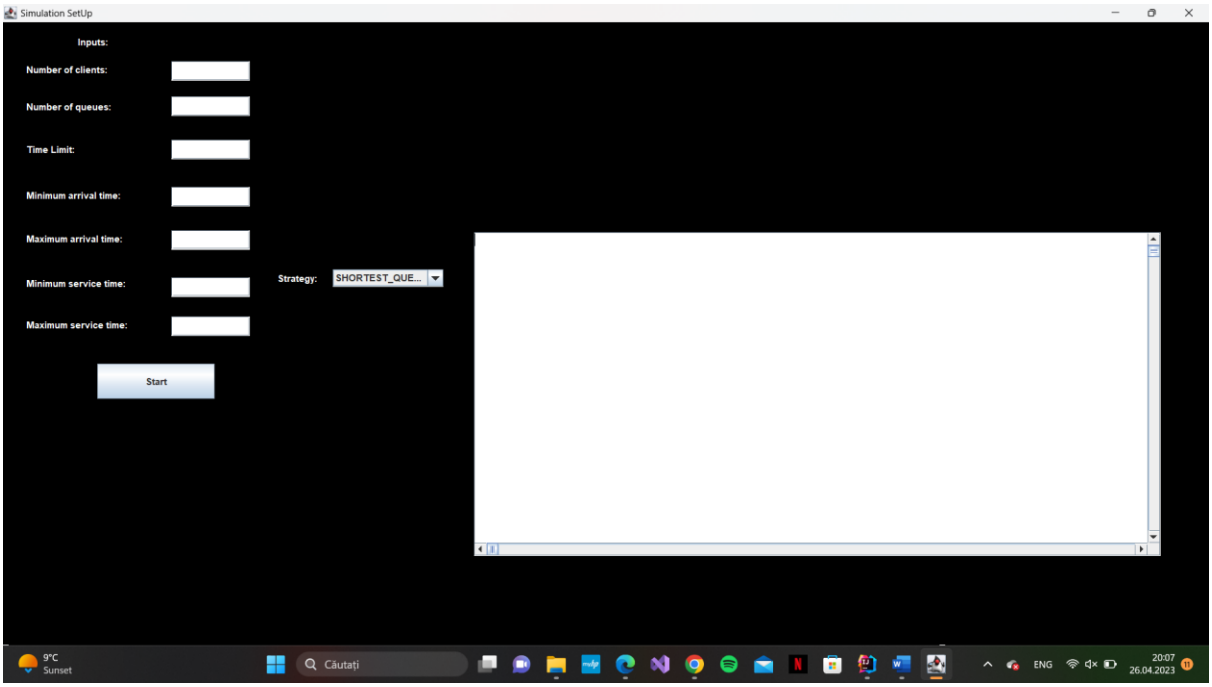
Classes diagram



2.3 Scenarios and use cases

A use case is the methodology used in system analysis to identify, clarify and organize system requirements. The use case is made up of a set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

The use cases area created by the user, which is why the GUI, needs to be a simple and straight-forward.

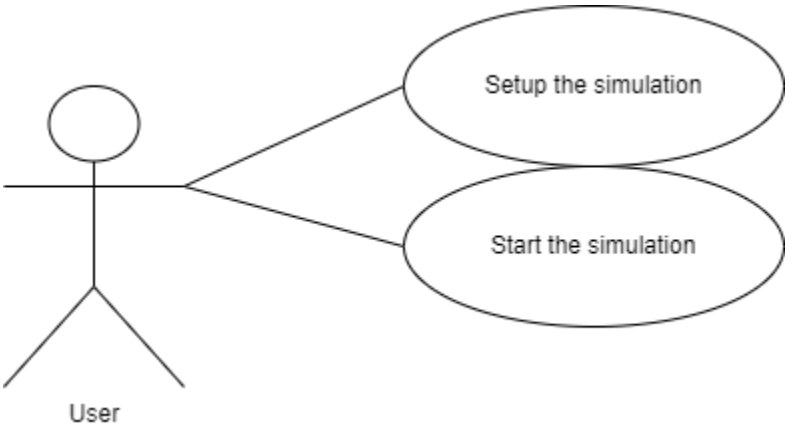


The user will need to input a valid number in each of the 7 text fields, select which strategy they want, and click the Start button to start the Queue Manager. If any of the values entered is wrong, the program will give an error and specify what is wrong. There need to be only numbers in the specified text fields and the number must be match, for example we can not have a minimum or maximum arrival time greater than the simulation time because that would be redundant, and the client will not get to enter a queue.

3. Design

3.1 Diagrams

Use case diagram:



3.2 Data structures

Integer: In Java, the Integer class is used to wrap primitive int values as objects. It provides various methods to perform arithmetic and logical operations on integers.

String: The String class in Java represents a sequence of characters. Strings are immutable in Java, meaning once they are created, their contents cannot be modified. The String class provides various methods for manipulating and accessing strings.

Float: The Float class in Java is used to wrap primitive float values as objects. It provides various methods to perform arithmetic and logical operations on floats.

BlockingQueue: The BlockingQueue interface in Java represents a queue that is thread-safe and allows multiple threads to access it concurrently. It provides methods to add, remove and inspect elements, and blocks threads that try to add or remove elements when the queue is full or empty.

Threads: In Java, a thread is a lightweight process that can be used to execute tasks concurrently. Java provides built-in support for creating and managing threads through the Thread class and related interfaces.

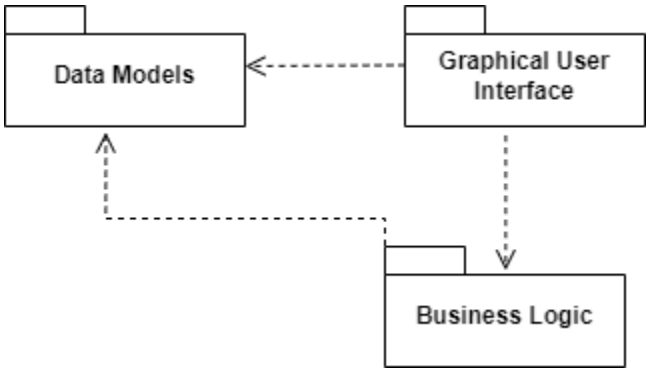
AtomicInteger: The AtomicInteger class in Java provides a thread-safe way to perform atomic operations on an integer value. It provides methods to perform operations like increment and decrement, and ensures that these operations are executed atomically, without interference from other threads.

List: The List interface in Java represents an ordered collection of elements. Lists can contain duplicate elements, and elements can be accessed by their index. Java provides several implementations of the List interface, including ArrayList, LinkedList, and Vector.

3.3 Packages

In Java, a package is a mechanism for organizing related classes and interfaces into a single namespace. A package is simply a folder that contains Java classes and interfaces, along with a special file called a manifest that provides metadata about the package.

Using packages can help to avoid naming conflicts between classes and provide better organization and modularity for large Java applications. By organizing related classes into packages, it becomes easier to locate and maintain classes, and it also helps to create a clear structure for the application.



Data Model package contains the way of storing the clients and the queues.

Graphical User Interface this package contains everything that has to do with the graphical user interface interface, inserting the needed variables, and displaying the results. In this package I have also the controller which is calling the BussinesLogic in the project, and it contains all the implementations of the needed classes and algorithms needed to place the clients in the right queues, such as inserting a client in the queue with the shortest time and putting the client in the queue with the least number of clients.

4. Implementation

1.Task class

This Java class is called Task and represents a single task with its associated arrival time, service time, and ID. It implements the Comparable interface to allow tasks to be compared based on their arrival times.

The class has a constructor that takes in three parameters: arrivalTime, serviceTime, and id. These values are used to set the corresponding instance variables of the class.

The class also has getter methods for retrieving the service time, arrival time, and ID of the task, as well as a setter method for modifying the service time of the task.

The toString() method is overridden to return a string representation of the task in the format "(id, arrivalTime, serviceTime);".

The class also implements the compareTo() method from the Comparable interface, which is used to compare tasks based on their arrival times. The method returns 1 if the arrival time of the other task is less than the current task's arrival time, -1 if the arrival time of the other task is greater than the current task's arrival time, and 0 if the arrival times are equal.

2. Server class

This Java class is called Server and represents a server that can process tasks. It uses a blocking queue to hold incoming tasks and an atomic integer to keep track of the waiting period for the tasks in the queue.

The constructor initializes the blocking queue and waiting period.

The getTasks() and getWaitingPeriod() methods return the blocking queue and waiting period of the server, respectively.

The addTask() method is used to add a new task to the queue and update the waiting period accordingly.

The run() method is the main method of the server and runs indefinitely in a loop. It takes the next task from the queue, sleeps for a time equal to the task's processing time, and decrements the waiting period accordingly. If the queue is empty, the loop ends and the server stops running.

The toString() method returns a string representation of the server's tasks and waiting period.

3. Scheduler class

The **Scheduler** class is responsible for managing the **Server** objects and dispatching incoming **Task** objects to the appropriate server based on the selected scheduling policy.

The class has a list of **Server** objects and a **Strategy** object that determines the scheduling policy. The **setStrategy()** method is used to set the scheduling policy. The **changeStrategy()** method is used to change the scheduling policy to either **SelectionPolicy.SHORTEST_QUEUE** or **SelectionPolicy.SHORTEST_TIME**.

The constructor initializes the **Server** objects and starts the corresponding threads.

The **dispatchTask()** method is used to dispatch incoming **Task** objects to the appropriate server based on the selected scheduling policy. It calls the **dispatchTask()** method of the **Strategy** object and passes in the list of servers and the incoming **Task** object. The **dispatchTask()** method of the **Strategy** object returns the index of the server where the task was dispatched. The **dispatchTask()** method of the **Scheduler** class returns the total processing time of the dispatched **Task**.

4. ShortestQueueStrategy class

The **`ShortestQueueStrategy`** class implements the **`Strategy`** interface and represents a scheduling strategy where a task is assigned to the server with the shortest waiting queue.

The **`dispatchTask`** method takes a list of servers and a task as input, and returns the position of the server in the list where the task has been assigned. The method iterates through all the servers in the list and finds the one with the smallest queue size. The task is then added to the queue of the selected server, and the position of the server in the list is returned.

The **`server`** instance variable is used to keep track of the server with the shortest queue, and it is set in the **`dispatchTask`** method.

5. TimeStrategy class

The **`TimeStrategy`** class implements the **`Strategy`** interface, which specifies a method **`dispatchTask`** for dispatching a given task to a server. This class uses a strategy of selecting the server with the shortest waiting period (i.e., the server that is expected to finish its current tasks soonest).

The **`dispatchTask`** method takes a list of servers and a task as parameters, and returns the position of the selected server in the list. It loops through all the servers in the list and finds the server with the shortest waiting period. Once the server with the shortest waiting period is found, the task is added to its queue and the position of the server in the list is returned.

6. SimulationManager class

The SimulationManager class is the main class that contains the simulation logic. It implements the Runnable interface to allow running it in a separate thread. Its constructor takes several parameters to initialize the simulation, including the number of clients, time limit, arrival and service time ranges, and selection policy.

The generateNRandomTasks() method generates a list of random tasks for the simulation based on the input parameters. It creates a new Task object for each client and adds it to the generatedTasks list. The list is then sorted based on the natural order of the tasks (which is their arrival time).

The run() method is the main simulation loop. It runs until the current time reaches the time limit. In each iteration, it first checks if there are any new clients that arrived at the current time and adds them to the scheduler. It then checks each server in the scheduler and updates the remaining service time for the task at the front of its queue (if any). Finally, it waits for one second before incrementing the current time and repeating the loop.

Inside the loop, there are several other operations, such as printing the current time, the queues of each server, and some statistics. The waiting time is also calculated by summing the service times of all tasks that waited in the queues.

At the end of the loop, some performance indicators are calculated and printed to the GUI. These include the average service time, waiting time, and peak hour.

7. SimulationFrame class

The class is a GUI application that sets up and displays a simulation. It contains various text fields and labels that represent input parameters to the simulation, such as minimum and maximum arrival time, minimum and maximum service time, time limit, number of clients, and number of queues. The user can set these input parameters and select a strategy from a drop-down list. The application also includes a start button to start the simulation and a text area to display the simulation results.

There are methods to retrieve the values of the input parameters and to add labels for each queue. The addQueueLabel() method adds a JLabel for each queue specified by the user. The SimulationFrame class extends the JFrame class and creates a new JFrame with the title "Simulation SetUp". The background color of the content pane is set to black.

8. Controller class

It serves as a controller in a graphical user interface (GUI) application that simulates a queueing system.

The class has a constructor that takes a SimulationFrame object as a parameter. The SimulationFrame object represents the graphical user interface of the queueing system simulation.

The class has an inner class named "RunListener" that implements the ActionListener interface. This inner class is responsible for handling the user's click on the "Start" button in the graphical user interface.

The actionPerformed method in the RunListener class extracts input values from the graphical user interface (such as the number of clients, simulation time, etc.) and creates a new SimulationManager object with these values. The SimulationManager object is responsible for running the queueing system simulation in a separate thread.

If the input values are not valid numbers, an error message is displayed using the JOptionPane class.

5. Results

The application was tested manually, and the results are displayed in the logs are in the User Interface, and in a separate file named test1.txt, test2.txt, test3.txt.

6. Conclusions

In my opinion, this project was a really good work with threads, working with threads, synchronize threads and how to organize my code really well in packages, and also how to draw a really friendly graphical user interface writing code, and now using the drag and drop from the IntelliJ.

7. Bibliography

Programming Techniques – Lecture and Laboratory slides of prof. Cristina Pop

https://www.wikipedia.org/

https://www.geeksforgeeks.org/

https://stackoverflow.com/

https://www.w3schools.com/java/java_threads.asp