



Queue Simulator manager using
threads
Documentație

Nume: Fărcaș Tudor Ioan
Grupa: 30225
AN: 2

Cuprins

1. Obiectiv
2. Studiul Problemei
3. Implementare
4. Concluzii
5. Bibliografie

1.Obiectiv

Proiectarea și implementarea unei aplicații de gestionare a cozilor care atribuie clienților cozi, astfel încât timpul de așteptare este minimizat.

Cozile sunt utilizate în mod obișnuit pentru a modela domenii din lumea reală. Obiectivul principal al unei cozi este de a oferi un loc pentru un "client" să aștepte înainte de a primi un "serviciu". Gestionarea pe bază de coadă sistemele sunt interesate de minimizarea sumei de timp pe care "clientii" lor o asteapta la cozi inainte acestea sunt servite

2. Studiul Problemei

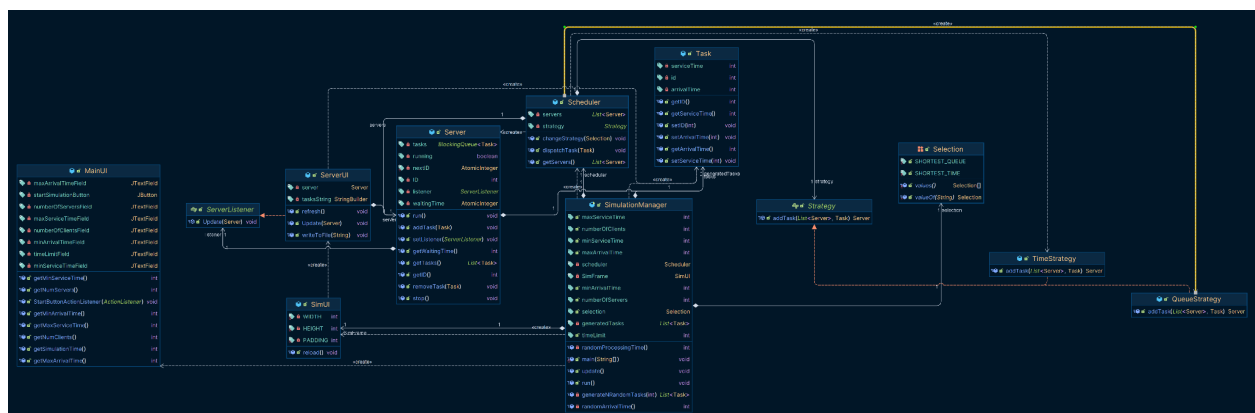
Modelarea problemei a fost facuta in mare parte dupa exemplul alaturat prezentarii temei, astfel incat cozile sunt modelate ca servere, care primesc task-uri (clienti) pe care trebuie sa le proceseze. Serverele sunt monitorizate si primesc task-uri de la un scheduler, care consulta timpul de asteptare la fiecare server in parte si ia o decizie cu privire la coada careia sa ii asigneze urmatorul client pentru a facilita eficienta.

3. Implementare

3.1 Diagrama UML

UML, sau Limbajul de Modelare Unificat, este o limbaj grafic care permite dezvoltatorilor de software să vizualizeze, să specifice, să construiască și să documenteze sisteme complexe. Este larg utilizat în industria software pentru modelarea sistemelor de software.

Diagramelor UML sunt utilizate pentru a reprezenta diferite aspecte ale unui sistem software. Acestea sunt compuse din elemente grafice, cum ar fi cutii, săgeți, linii și simboluri, care sunt utilizate pentru a reprezenta entități și relații între entități.



Pachetul UI

In acest pachet avem elementele de interfata grafica si functia de scriere a rezultatului intr-un fisier cu extensia .txt.

MainUI este interfata de baza de unde preluam datele pe care le vom introduce in aplicatie.

ServerUI este interfata unde se petrece actiunea si SimUI este clasa care leaga cele 2 interfete si le face sa mearga

Pachetul Business_Logic

In pachetul acesta avem toate clasele care ne ajuta sa ruleze aplicatia conform cerintelor(TimeStrategy, QueueStrategy, Selection, etc.)

Pachetul Model

In acest pachet avem elementele de baza pentru a rula aceasta aplicatie, precum Server si Task.

3.2 Code Parts

Simulation Manager:

```
package Business_Logic;

import Model.Task;
import UI.MainUI;
import UI.SimUI;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.Random;
import java.util.concurrent.TimeUnit;

public class SimulationManager implements Runnable {

    private final Scheduler scheduler;
    private final SimUI SimFrame;
    private final List<Task> generatedTasks;
    public int timeLimit;
    public int minArrivalTime;
    public int maxArrivalTime;
```

```

public int minServiceTime;
public int maxServiceTime;
public int numberOfServers;
public int numberOfClients;
public Selection selection = Selection.SHORTEST_TIME;

public SimulationManager(MainUI setupFrame) {
    timeLimit = setupFrame.getSimulationTime();
    minArrivalTime = setupFrame.getMinArrivalTime();
    maxArrivalTime = setupFrame.getMaxArrivalTime();
    minServiceTime = setupFrame.getMinServiceTime();
    maxServiceTime = setupFrame.getMaxServiceTime();
    numberOfServers = setupFrame.getNumServers();
    numberOfClients = setupFrame.getNumClients();

    scheduler = new Scheduler(numberOfServers, numberOfClients);
    scheduler.changeStrategy(selection);

    SimFrame = new SimUI(scheduler.getServers());
    SimFrame.setVisible(true);
    generatedTasks = generateNRandomTasks(numberOfClients);
}

public static void main(String[] args) {
    MainUI setupFrame = new MainUI();
    setupFrame.setVisible(true);
    setupFrame.StartButtonActionListener(e -> {
        SimulationManager simulationManager = new SimulationManager(setupFrame);
        Thread t = new Thread(simulationManager);
        t.start();
        setupFrame.setVisible(false);
    });
}

public void update() {
    SimFrame.reload();
}

private List<Task> generateNRandomTasks(int n) {
    // Random random = new Random();
    List<Task> tasks = new ArrayList<>();

    for (int i = 0; i < n; i++) {
        int arrivalTime = randomArrivalTime();
        int processingTime = randomProcessingTime();
        tasks.add(new Task(i, arrivalTime, processingTime));
    }

    tasks.sort(Comparator.comparing(Task::getArrivalTime));

    return tasks;
}

private int randomArrivalTime() {
    Random random = new Random();

```

```

        return minArrivalTime + random.nextInt(maxArrivalTime - minArrivalTime + 1);
    }

    private int randomProcessingTime() {
        Random random = new Random();
        return minServiceTime + random.nextInt(maxServiceTime - minServiceTime + 1);
    }

    @Override
    public void run() {
        int currentTime = 0;
        while (currentTime < timeLimit) {
            while (!generatedTasks.isEmpty() && generatedTasks.get(0).getArrivalTime()
== currentTime) {
                Task currentTask = generatedTasks.remove(0);
                scheduler.dispatchTask(currentTask);
            }
            update();

            currentTime++;

            try {
                TimeUnit.SECONDS.sleep(1);
            } catch (InterruptedException e) {
                System.err.println("Thread was interrupted: " + e.getMessage());
            }
        }
    }
}

```

Main UI

```

package UI;

import javax.swing.*.*;
import java.awt.*.*;
import java.awt.event.ActionListener;

public class MainUI extends JFrame {

    private final JTextField numberOfClientsField, numberOfServersField,
timeLimitField;
    private final JTextField minArrivalTimeField, maxArrivalTimeField,
minServiceTimeField, maxServiceTimeField;
    private final JButton startSimulationButton;

    public MainUI() {
        setTitle("Queues management");
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new GridBagLayout());
        setSize(500, 400);
    }
}

```

```
setLocationRelativeTo(null);
getContentPane().setBackground(new Color(0, 0, 0));

GridBagConstraints main = new GridBagConstraints();
main.insets = new Insets(10, 10, 10, 10);

Font labelFont = new Font("Arial", Font.PLAIN, 16);
Color labelColor = new Color(255, 255, 255);

JLabel numberOfClientsLabel = new JLabel("Numarul de clienti");
numberOfClientsLabel.setFont(labelFont);
numberOfClientsLabel.setForeground(labelColor);

JLabel numberOfServersLabel = new JLabel("Numarul de cozi");
numberOfServersLabel.setFont(labelFont);
numberOfServersLabel.setForeground(labelColor);

JLabel timeLimitLabel = new JLabel("Timpul maxim de simulare");
timeLimitLabel.setFont(labelFont);
timeLimitLabel.setForeground(labelColor);

JLabel minArrivalTimeLabel = new JLabel("Timpul minim de sosire");
minArrivalTimeLabel.setFont(labelFont);
minArrivalTimeLabel.setForeground(labelColor);

JLabel maxArrivalTimeLabel = new JLabel("Timpul maxim de sosire");
maxArrivalTimeLabel.setFont(labelFont);
maxArrivalTimeLabel.setForeground(labelColor);

JLabel minServiceTimeLabel = new JLabel("Timpul minim de servire");
minServiceTimeLabel.setFont(labelFont);
minServiceTimeLabel.setForeground(labelColor);

JLabel maxServiceTimeLabel = new JLabel("Timpul maxim de servire");
maxServiceTimeLabel.setFont(labelFont);
maxServiceTimeLabel.setForeground(labelColor);

timeLimitField = new JTextField(10);
numberOfServersField = new JTextField(10);
minArrivalTimeField = new JTextField(10);
numberOfClientsField = new JTextField(10);
maxArrivalTimeField = new JTextField(10);
minServiceTimeField = new JTextField(10);
maxServiceTimeField = new JTextField(10);

main.gridx = 0;
main.gridy = 0;
add(numberOfClientsLabel, main);

main.gridx = 1;
add(numberOfClientsField, main);

main.gridx = 0;
main.gridy++;
add(numberOfServersLabel, main);
```



```

        main.gridx = 1;
        add(numberOfServersField, main);

        main.gridx = 0;
        main.gridy++;
        add(timeLimitLabel, main);

        main.gridx = 1;
        add(timeLimitField, main);

        main.gridx = 0;
        main.gridy++;
        add(minArrivalTimeLabel, main);

        main.gridx = 1;
        add(minArrivalTimeField, main);

        main.gridx = 0;
        main.gridy++;
        add(maxArrivalTimeLabel, main);

        main.gridx = 1;
        add(maxArrivalTimeField, main);

        main.gridx = 0;
        main.gridy++;
        add(minServiceTimeLabel, main);

        main.gridx = 1;
        add(minServiceTimeField, main);

        main.gridx = 0;
        main.gridy++;
        add(maxServiceTimeLabel, main);

        main.gridx = 1;
        add(maxServiceTimeField, main);

        main.gridx = 0;
        main.gridy++;
        main.gridwidth = 2;
        main.fill = GridBagConstraints.HORIZONTAL;

        startSimulationButton = new JButton("START");
        startSimulationButton.setBackground(new Color(43, 223, 114, 255));
        startSimulationButton.setForeground(new Color(255, 255, 255));
        startSimulationButton.setBorder(BorderFactory.createEmptyBorder(10, 20, 10,
20));

        add(startSimulationButton, main);
    }

    public void StartButtonActionListener(ActionListener listener) {
        startSimulationButton.addActionListener(listener);
    }

```

```

    }

    public int getNumClients() {
        return Integer.parseInt(numberOfClientsField.getText());
    }

    public int getNumServers() {
        return Integer.parseInt(numberOfServersField.getText());
    }

    public int getSimulationTime() {
        return Integer.parseInt(timeLimitField.getText());
    }

    public int getMinArrivalTime() {
        return Integer.parseInt(minArrivalTimeField.getText());
    }

    public int getMaxArrivalTime() {
        return Integer.parseInt(maxArrivalTimeField.getText());
    }

    public int getMinServiceTime() {
        return Integer.parseInt(minServiceTimeField.getText());
    }

    public int getMaxServiceTime() {
        return Integer.parseInt(maxServiceTimeField.getText());
    }
}

```

3.3 Pics of UI



Queues management



Numarul de clienti

Numarul de cozi

Timpul maxim de simulare

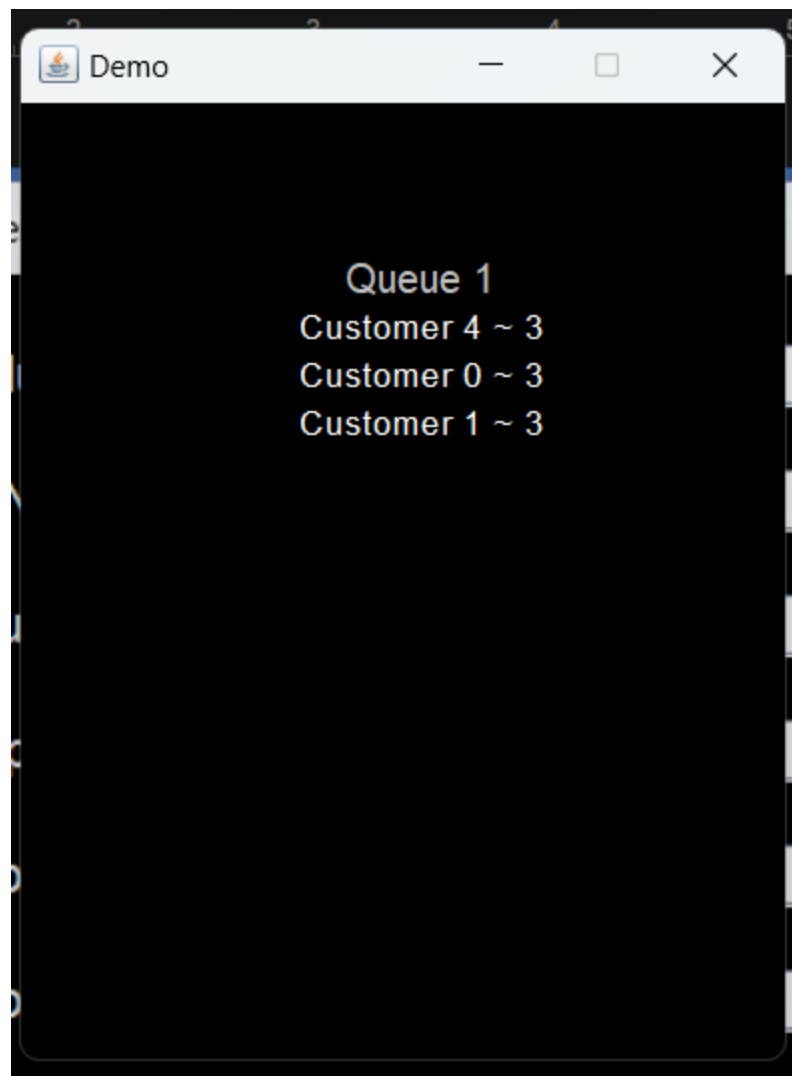
Timpul minim de sosire

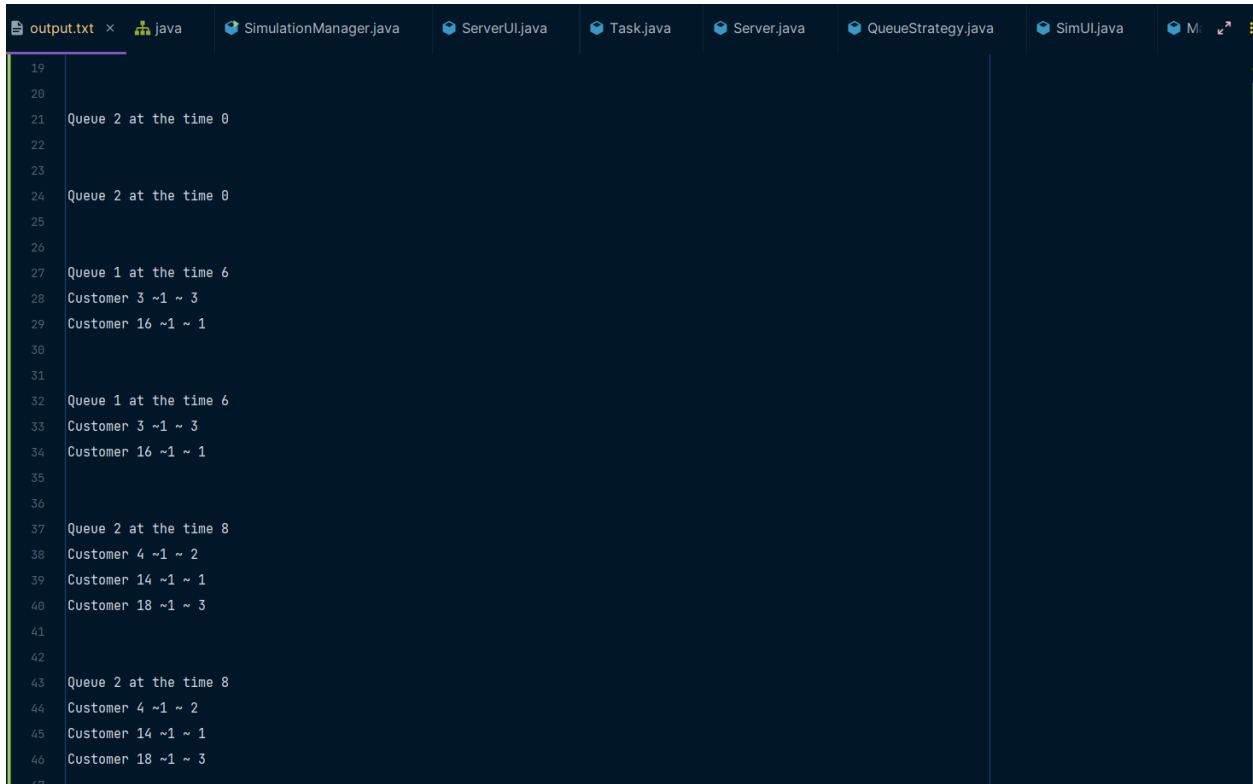
Timpul maxim de sosire

Timpul minim de servire

Timpul maxim de servire

START





The screenshot shows an IDE with several Java files open: `output.txt`, `SimulationManager.java`, `ServerUI.java`, `Task.java`, `Server.java`, `QueueStrategy.java`, `SimUI.java`, and `M...`. The `output.txt` window displays the following log output:

```
19  
20  
21 Queue 2 at the time 0  
22  
23  
24 Queue 2 at the time 0  
25  
26  
27 Queue 1 at the time 6  
28 Customer 3 ~1 ~ 3  
29 Customer 16 ~1 ~ 1  
30  
31  
32 Queue 1 at the time 6  
33 Customer 3 ~1 ~ 3  
34 Customer 16 ~1 ~ 1  
35  
36  
37 Queue 2 at the time 8  
38 Customer 4 ~1 ~ 2  
39 Customer 14 ~1 ~ 1  
40 Customer 18 ~1 ~ 3  
41  
42  
43 Queue 2 at the time 8  
44 Customer 4 ~1 ~ 2  
45 Customer 14 ~1 ~ 1  
46 Customer 18 ~1 ~ 3  
47
```

4. Concluzii

Tind sa mentionez ca tema a fost facuta partial treaz, iar in unele clase le-am facut sub influenta alcoolului.

În concluzie, consider că acest proiect mi-a consolidat cunoștințele în domeniul limbajului Java, implementarea paradigmei POO și crearea unei aplicații cu interfață grafică.

5. Bibliografie

<https://docs.oracle.com/javase/tutorial/essential/concurrency/>

<https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/atomic/AtomicInteger.html>

<https://www.ibm.com/docs/en/i/7.1?topic=techniques-synchronization-among-threads>

<https://www.youtube.com/watch?v=y8liDp5jgTc>

<https://www.geeksforgeeks.org/java-program-to-write-into-a-file/>

<https://stackoverflow.com/questions/13874270/java-gui-writing-to-output-file>