INGENIERIA EN SISTEMAS COMPUTACIONALES

TOPICOS AVANZADOS DE PROGRAMACION

REPORTE – SLEEPING BARBER PROBLEM

ALUMNO:

LEONEL ALEJANDRO AGUIRRE SERRANO

PROFESOR

ING. LUIS EDUARDO GUTIERREZ AYALA

LEÓN, GUANAJUATO A 26 DE MAYO DEL 2020

REDACCION DEL PROBLEMA:

El problema presentado en este reporte consiste en la creación de un programa en el que se ejemplifique el problema del barbero durmiente utilizando sincronización de hilos y el modelo productor/consumidor.

INVESTIGACION:

El **problema del barbero durmiente** es un problema utilizado en ciencias de la computación para ejemplificar la implementación de sincronización de subprocesos.

La idea del problema consiste en una barbería hipotética, en la que se encuentra un barbero y varias sillas de espera para los clientes que van llegando. El comportamiento del barbero y los clientes seguirá las siguientes reglas:

- Si no hay ningún cliente esperando su turno, el barbero ira a dormir.
- Si hay clientes en espera el barbero despertara y comenzara a cortar el cabello de los clientes.
- Cuando un cliente llega evalúa si hay asientos disponibles en las sillas de espera, si hay un asiento disponible, este se sentara y esperara su turno para que el barbero corte su cabello, en cambio, si no hay asientos disponibles, el cliente se ira de la barbería.

CODIGO FUENTE:

Clase SleepingBarberProbleGUI

```
package com.milkyblue;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JTextField;
```

```java
import com.github.tomaslanger.chalk.Chalk;

// Class SleepingBarberProblemGUI. Models the GUI.
public class SleepingBarberProblemGUI {

  private JFrame mainFrame;
  private JPanel mainPanel, topPanel, centerPanel, bottomPanel;
  private JLabel lblAdvice, lblChairAmount;
  private JTextField txtChairAmount;
  private JButton btnExecute;

  // Class constructor.
  public SleepingBarberProblemGUI() {
    mainFrame = new JFrame("Sleeping Barber Problem");
    mainPanel = new JPanel(new BorderLayout());
    topPanel = new JPanel();
    centerPanel = new JPanel();
    bottomPanel = new JPanel();


    lblAdvice = new JLabel("Input the required data");
    lblChairAmount = new JLabel("Chairs:");
    txtChairAmount = new JTextField(10);
    btnExecute = new JButton("Execute");

    addAttributes();
    addListeners();
    build();
    launch();
  }

  // Adds attributes to elements in the class.
  private void addAttributes() {
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    mainFrame.setResizable(false);
  }

  // Adds listeners to elements in GUI.
  private void addListeners() {

    // When btnExecute is pressed a new BarberShop is created with the amount of
    // chairs specified in the GUI. Also a CustomerGenerator and a Barber Threads
    // are started with an ExecutorService.
    btnExecute.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
        Chalk.setColorEnabled(true);

        try {
          BarberShop bShop = new BarberShop(Integer.parseInt(txtChairAmount.getText()));

          CustomerGenerator generator = new CustomerGenerator(bShop);
          Barber barber = new Barber(bShop);

          ExecutorService executor = Executors.newFixedThreadPool(2);

          executor.execute(generator);
          executor.execute(barber);

        } catch (Exception error) {
          JOptionPane.showMessageDialog(null,
              "<html><span style='font-
weight: bold; color: red'>ERROR: </span>Type valid information.<html>", "Error",
              JOptionPane.PLAIN_MESSAGE);
        }

      }
    });
  }

  // Builds the GUI.
  private void build() {
    topPanel.add(lblAdvice);
    centerPanel.add(lblChairAmount);
    centerPanel.add(txtChairAmount);
    bottomPanel.add(btnExecute);

    mainPanel.add(topPanel, BorderLayout.NORTH);
    mainPanel.add(centerPanel, BorderLayout.CENTER);
    mainPanel.add(bottomPanel, BorderLayout.SOUTH);

    mainFrame.add(mainPanel);
  }

  // Launches the GUI by setting the mainFrame's visible value to true.
  private void launch() {
    mainFrame.setVisible(true);
    mainFrame.pack();
    mainFrame.setLocationRelativeTo(null);
  }

}
```

Clase BarberShop

```java
package com.milkyblue;

import java.util.Arrays;
import java.util.concurrent.ArrayBlockingQueue;

import com.github.tomaslanger.chalk.Chalk;

// BarberShop class. Models a buffer based object, keeps track of the amount
// of chairs in the shop.
public class BarberShop {

  private ArrayBlockingQueue<Customer> buffer;
  private int nChairs;

  // Class constructor.
  public BarberShop(int nChairs) {
    buffer = new ArrayBlockingQueue<Customer>(nChairs);
    this.nChairs = nChairs;
  }

  // Removes the customer passed as a parameter from the buffer.
  public void remove(Customer customer) {
    buffer.remove();
  }

  // Gets a copy of the first customer from the buffer.
  public Customer getCustomer() {
    return buffer.element();
  }

  // Adds a customer passed as a parameter to the buffer.
  public void put(Customer customer) throws InterruptedException {
    buffer.put(customer);
  }

  // Returns whether there is an available chair or not.
  public boolean availableChair() {
    return (buffer.remainingCapacity() > 0);
  }

  // Returns whether the buffer is empty or not.
  public boolean isEmpty() {
    return buffer.isEmpty();
  }
```

```
  // Returns a string of the actual state of the chairs.
  public String toString() {

    Chalk[] chairs = new Chalk[nChairs];

    for (int i = 0; i < chairs.length; i++)
      chairs[i] = Chalk.on("Empty").yellow();

    for (int i = 0; i < buffer.toArray().length; i++)
      chairs[i] = Chalk.on(buffer.toArray()[i].toString()).cyan();

    return Arrays.toString(chairs);
  }

}
```

Clase CustomerGenerator

```
package com.milkyblue;

import com.github.tomaslanger.chalk.Chalk;

// CustomerGenerator class. Models a producer based thread,
// generates a new customer.
public class CustomerGenerator implements Runnable {

  BarberShop buffer;

  // Class constructor.
  public CustomerGenerator(BarberShop buffer) {
    this.buffer = buffer;
  }

  // Runs when the thread is started. Generates the new customer in random
  // intervals of time between 0 and 10 seconds.
  public void run() {
    while (true) {
      try {
        int time = (int) Math.floor(Math.random() * 10000);
        Thread.sleep(time);
        Customer newCustomer = new Customer(buffer);

        System.out.println("[" + Chalk.on("C-
" + newCustomer.getId()).green() + "] New customer arrived after: " + time
            + " miliseconds.");
        newCustomer.enter();
```

```
        System.out.println("[" + Chalk.on("Chairs").magenta() + "] => " + buffer.toString(
));
      } catch (Exception e) {
        e.printStackTrace();
      }
    }
  }
}
```

Clase Customer

```
package com.milkyblue;

import com.github.tomaslanger.chalk.Chalk;

// Customer class. Models a customer, each one of them with a unique id and
// a reference to a buffer.
public class Customer {
  private BarberShop buffer;
  private static int idCount = 0;
  private int id;

  // Class constructor.
  public Customer(BarberShop buffer) {
    this.buffer = buffer;
    this.id = ++idCount;
  }

  // Customer enters to the barber shop, if there is an available chair the
  // customer takes that chair and waits for his turn, otherwise, if there are no
  // available chairs the customer leaves the shop.
  public void enter() {
    if (buffer.availableChair()) {
      try {
        buffer.put(this);
      } catch (Exception e) {
        e.printStackTrace();
      }
      System.out.println("[" + Chalk.on("C-
" + id).cyan() + "] Customer is waiting his turn.");
    } else {
      System.out.println("[" + Chalk.on("C-
" + id).red() + "] No available chairs, customer leave.");
    }
  }
```

```
  // Returns the customer's id.
  public int getId() {
    return id;
  }

  // Returns a text representation of the customer.
  public String toString() {
    return "C-" + id;
  }
}
```

Clase Barber

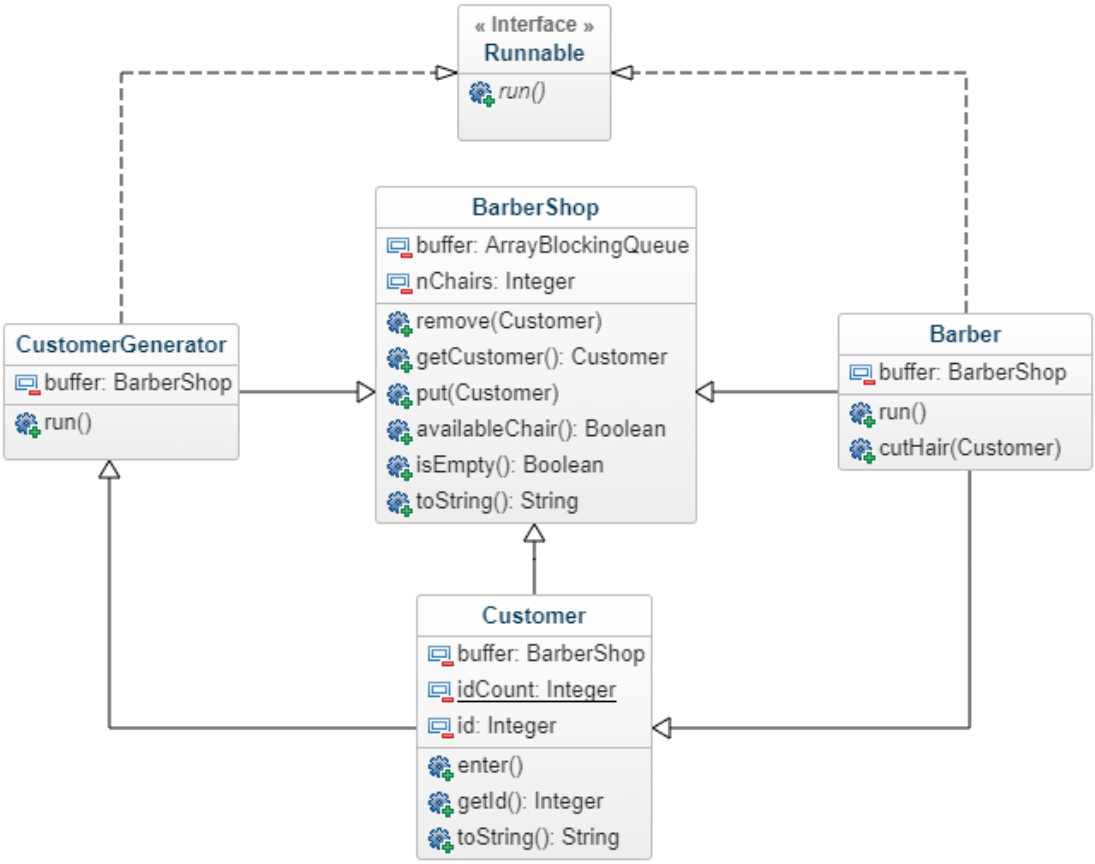
```
package com.milkyblue;

import com.github.tomaslanger.chalk.Chalk;

// Barber class. Models a consumer based Thread, when there are no clients in the
// buffer the Barber is sleeping, otherwise, when a client is in the buffer (sitting
// on a chair) the Barber wakes and cuts his hair.
public class Barber implements Runnable {

  private BarberShop buffer;

  // Class constructor.
  public Barber(BarberShop buffer) {
    this.buffer = buffer;
  }

  // Runs when the Thread is started. Checks if there is a customer in the buffer
  // in intervals of 1 second.
  public void run() {
    while (true) {

      try {
        Thread.sleep(1000);
      } catch (Exception e) {
        e.printStackTrace();
      }

      if (!buffer.isEmpty()) {
        // BARBER CUTS CUSTOMER'S HAIR.
        Customer customer = buffer.getCustomer();
        System.out.println(
            "[" + Chalk.on("Barber").yellow() + "] cutting " + Chalk.on("C-
" + customer.getId()).cyan() + "'s hair.");
```

```
        cutHair(customer);
      } else {
        // BARBER SLEEPS.
        System.out.println("[" + Chalk.on("Barber").yellow() + "] sleeping.");
      }

    }

  }

  // Cuts the hair of a customer passed as a parameter in a random amount of time
  // from 0 to 10 seconds. Then the customer is removed from the buffer.
  private void cutHair(Customer customer) {
    try {
      Thread.sleep((int) Math.floor(Math.random() * 10000));
      System.out.println("[" + Chalk.on("C-
" + customer.getId()).cyan() + "] Customer got its hair cut.");
      buffer.remove(customer);
    } catch (Exception e) {
      e.printStackTrace();
    }

  }

}
```

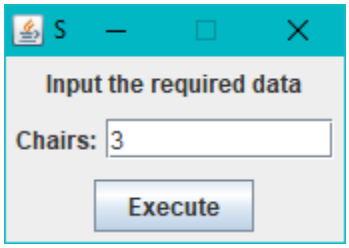
Clase App

```
package com.milkyblue;

// App Class.
public class App {
    // Creates an anonymous instance of SleepingBarberProblemGUI.
    public static void main(String[] args) {
        new SleepingBarberProblemGUI();
    }

}
```

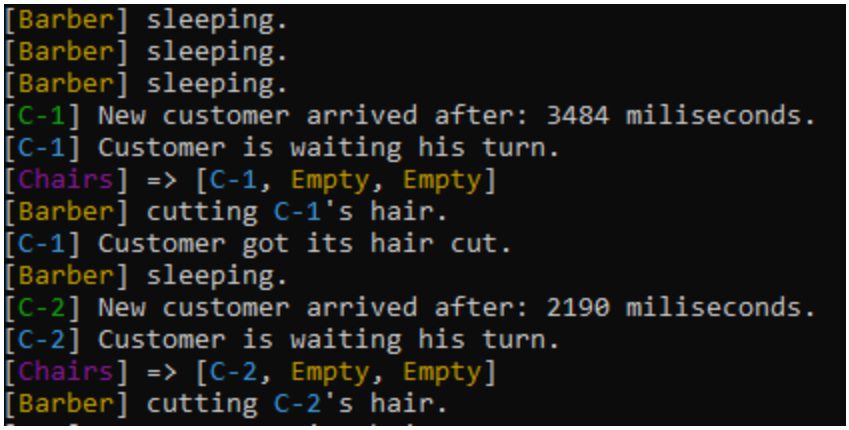
DIAGRAMA UML:



CAPTURAS:

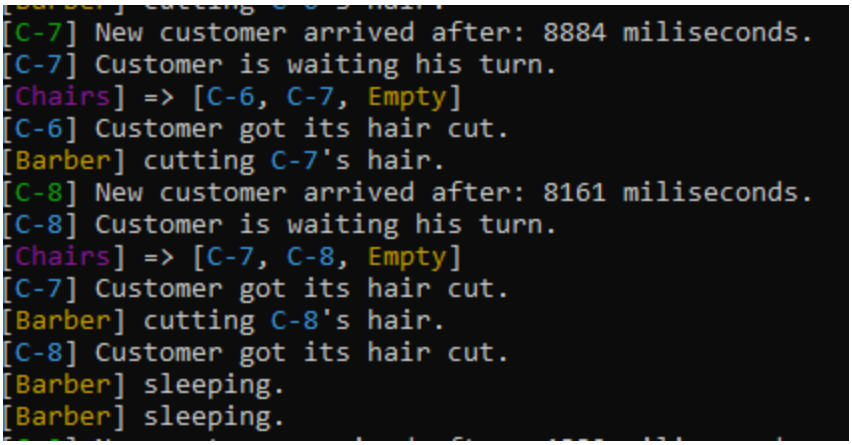


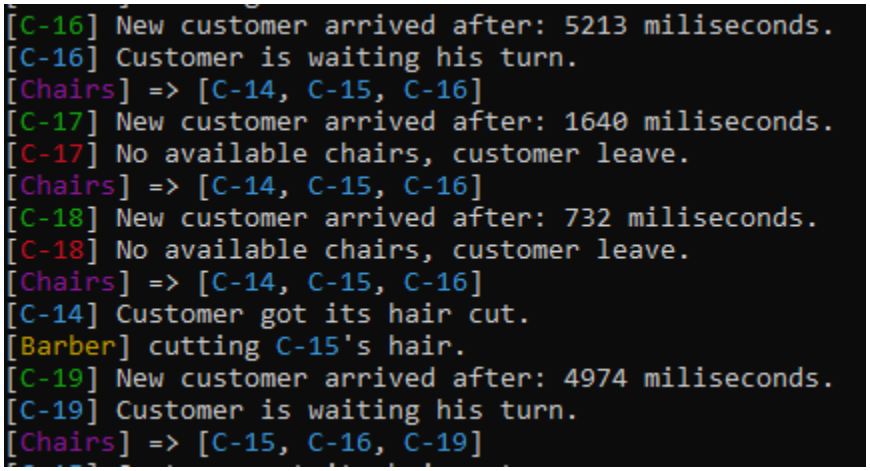
INTERFAZ GRAFICA DE USUARIO DEL PROGRAMA ESTABLECIENDO 3 COMO LA CANTIDAD DE SILLAS EN LA BARBERIA.



IMPRESION EN CONSOLA. EL BARBERO COMIENZA DORMIDO YA QUE NO HAY CLIENTES, A MEDIDA QUE SE GENERAN CLIENTES, ESTOS TOMAN UN LUGAR Y ENSEGUIDA EL BARBERO COMIENZA A ATENDERLOS, SEGUIDO DE ESTO CADA CLIENTE SE RETIRA DE LA BARBERIA.

IMPRESION EN CONSOLA. A MEDIDA QUE EL BARBERO ATIENDE A LOS CLIENTES Y ESTOS SE RETIRAN DE LA BARBERIA, EL LUGAR QUE ESTABAN OCUPANDO SE VACIA NUEVAMENTE, CUANDO LA BARBERIA SE VACIA POR COMPLETO, NUEVAMENTE EL BARBERO REGRESA A DORMIR.





IMPRESION EN CONSOLA. CUANDO TODAS LAS SILLAS DE ESPERA DE LA BARBERIA ESTAN SIENDO OCUPADAS Y LLEGAN NUEVOS CLIENTES, ESTOS SE RETIRARAN DE LA BARBERIA YA QUE NO HAY UN LUGAR PARA ELLOS DONDE ESPERAR.

NOTAS:

- Puede encontrar el repositorio de este proyecto en mi cuenta de github en el siguiente enlace: https://github.com/NoisyApple/AdTopics-18.SleepingBarberProblem