****



**TECNOLÓGICO NACIONAL DE MÉXICO**

**Instituto Tecnológico de León**

**INGENIERIA EN SISTEMAS COMPUTACIONALES**

**TOPICOS AVANZADOS DE PROGRAMACION**

**REPORTE – THREAD SYNCHRONIZATION**

**ALUMNO:**

LEONEL ALEJANDRO AGUIRRE SERRANO

**PROFESOR**

ING. LUIS EDUARDO GUTIERREZ AYALA

**LEÓN, GUANAJUATO A 19 DE MAYO DEL 2020**

**REDACCION DEL PROBLEMA:**

El problema presentado en este reporte consiste en la creación de un programa que haga uso de la sincronización de subprocesos.

**CODIGO FUENTE:**

**Clase ThreadSyncGUI**

package com.milkyblue;

import java.awt.BorderLayout;

import java.awt.Color;

import java.awt.Dimension;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import java.util.concurrent.TimeUnit;

import javax.swing.BorderFactory;

import javax.swing.JButton;

import javax.swing.JCheckBox;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JPanel;

import com.github.tomaslanger.chalk.Chalk;

// ThreadSyncGUI class. Models the GUI of the program.

public class ThreadSyncGUI {

  // Constants to control color between threads.

  public static final *int* BLUE = 0, YELLOW = 1;

  private *JFrame* mainFrame;

  private *JPanel* mainPanel, topPanel, centerPanel, bottomPanel;

  private *JCheckBox* chkSync;

  private *JLabel*[][] arrayLookups;

  private *JButton* btnExecute;

  // Class constructor.

  public ThreadSyncGUI() {

    // Enables colored output on terminal.

    Chalk.setColorEnabled(true);

    mainFrame = **new** JFrame("Thread Synchronization");

    mainPanel = **new** JPanel();

    topPanel = **new** JPanel();

    centerPanel = **new** JPanel();

    bottomPanel = **new** JPanel();

    chkSync = **new** JCheckBox("Thread Synchronization");

    // arrayLookups is initialized as a two dimensional array of labels, one row for

    // array positions and the other row for array actual values.

    arrayLookups = **new** *JLabel*[2][6];

    for (*int* i = 0; i < arrayLookups.length; i++)

      for (*int* j = 0; j < arrayLookups[i].length; j++) {

        if (i == 0)

          arrayLookups[i][j] = **new** JLabel(Integer.toString(j), JLabel.CENTER);

        else

          arrayLookups[i][j] = **new** JLabel("0", JLabel.CENTER);

      }

    btnExecute = **new** JButton("Execute");

    // Main methods are called.

    addAttributes();

    addListeners();

    build();

    launch();

  }

  // Adds attributes to elements in the program.

  private *void* addAttributes() {

    mainPanel.setLayout(**new** BorderLayout());

    centerPanel.setLayout(**new** GridBagLayout());

    centerPanel.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));

    // Sets attributes to arrayLookups labels based on the correspondent row.

    for (*int* i = 0; i < arrayLookups.length; i++)

      for (*int* j = 0; j < arrayLookups[i].length; j++) {

        arrayLookups[i][j].setOpaque(true);

        if (i == 0)

          arrayLookups[i][j].setBackground(Color.decode("#333333"));

        else

          arrayLookups[i][j].setBackground(Color.decode("#888888"));

        arrayLookups[i][j].setForeground(Color.decode("#DDDDDD"));

        arrayLookups[i][j].setPreferredSize(**new** Dimension(30, 20));

      }

    mainFrame.setResizable(false);

    mainFrame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

  }

  // Adds listeners to elements in the GUI.

  private *void* addListeners() {

    // Creates a SimpleArray instance and uses two ArrayWriter instances to fill it.

    btnExecute.addActionListener(**new** ActionListener() {

      public *void* actionPerformed(*ActionEvent* *e*) {

*boolean* useSync = chkSync.isSelected();

        // Resets values row background and text.

        for (*JLabel* lookup : arrayLookups[1]) {

          lookup.setBackground(Color.decode("#888888"));

          lookup.setText("0");

        }

*SimpleArray* sharedArray = **new** SimpleArray(6);

*ArrayWriter* writer1 = **new** ArrayWriter(1, sharedArray, arrayLookups[1], BLUE, useSync);

*ArrayWriter* writer2 = **new** ArrayWriter(11, sharedArray, arrayLookups[1], YELLOW, useSync);

        // Executes both instances of ArrayWriter.

*ExecutorService* executor = Executors.newCachedThreadPool();

        executor.execute(writer1);

        executor.execute(writer2);

        executor.shutdown();

        // Waits for executor threads termination.

        try {

*boolean* tasksStopped = executor.awaitTermination(1, TimeUnit.MINUTES);

          if (tasksStopped)

            System.out.println(sharedArray);

          else

            System.out

                .println("[" + Chalk.on("FAILURE").red() + "] Timeout expired while awaiting for tasks termination");

        } catch (*Exception* *ex*) {

          System.out

              .println("[" + Chalk.on("ERROR").red() + "] Interruption ocurred while awaiting for tasks termination");

        }

      }

    });

  }

  // Builds the GUI.

  private *void* build() {

    topPanel.add(chkSync);

*GridBagConstraints* gbc = **new** GridBagConstraints();

    // Adds arrayLookups labels.

    for (*int* i = 0; i < arrayLookups.length; i++)

      for (*int* j = 0; j < arrayLookups[i].length; j++) {

        gbc.gridy = i;

        gbc.gridx = j;

        gbc.ipadx = 0;

        gbc.ipady = 0;

        gbc.insets = **new** Insets(1, 2, 1, 2);

        centerPanel.add(arrayLookups[i][j], gbc);

      }

    bottomPanel.add(btnExecute);

    mainPanel.add(topPanel, BorderLayout.NORTH);

    mainPanel.add(centerPanel, BorderLayout.CENTER);

    mainPanel.add(bottomPanel, BorderLayout.SOUTH);

    mainFrame.add(mainPanel);

  }

  // Launches mainFrame by setting its visible value to true, then centers and

  // resizes the frame.

  private *void* launch() {

    mainFrame.setVisible(true);

    mainFrame.pack();

    mainFrame.setLocationRelativeTo(null);

  }

}

**Clase ArrayWriter**

package com.milkyblue;

import javax.swing.JLabel;

// ArrayWriter class. fills a SimpleArray instance.

public class ArrayWriter implements *Runnable* {

  private final *SimpleArray* sharedArray;

  private final *int* initValue;

  private *JLabel*[] lookups;

  private *int* color;

  private *boolean* useSync;

  // Class constructor. Keeps track of the lookup labels from the GUI, also the

  // selected color and if a Synchronized method will be used or not.

  public ArrayWriter(*int* *value*, *SimpleArray* *array*, *JLabel*[] *lookups*, *int* *color*, *boolean* *useSync*) {

    initValue = value;

    sharedArray = array;

*this*.lookups = lookups;

*this*.color = color;

*this*.useSync = useSync;

  }

  // Method executed when thread runs. Calls wether a synchronized or

  // non-synchronized method to add a value to the SimpleArray element depending

  // on the constructor passed arguments.

  public *void* run() {

    for (*int* i = initValue; i < initValue + 3; i++)

      if (useSync)

        sharedArray.addSync(i, lookups, color);

      else

        sharedArray.addNonSync(i, lookups, color);

  }

}

**Clase PrintableTask**

package com.milkyblue;

import java.awt.Color;

import java.util.Random;

import javax.swing.JLabel;

import com.github.tomaslanger.chalk.Chalk;

// SimpleArray class. Models an array that keeps track of the last updated index. Also updates the GUI when its own array is updated.

public class SimpleArray {

  private final *int*[] array;

  private *int* index;

  private final static *Random* generator = **new** Random();

  // Class constructor.

  public SimpleArray(*int* *arrLength*) {

    array = **new** *int*[arrLength];

    index = 0;

  }

  // Synchronized method.

  public synchronized *void* addSync(*int* *value*, *JLabel*[] *lookups*, *int* *color*) {

    add(value, lookups, color);

  }

  // Non-synchronized method.

  public *void* addNonSync(*int* *value*, *JLabel*[] *lookups*, *int* *color*) {

    add(value, lookups, color);

  }

  // Functionality method to update the array.

  private *void* add(*int* *value*, *JLabel*[] *lookups*, *int* *color*) {

*int* postition = index;

    // Waits between 0 and 5 seconds before updating the array.

    try {

      Thread.sleep(generator.nextInt(5000));

    } catch (*Exception* *e*) {

      e.printStackTrace();

    }

    // Updates both the array and the correspondent GUI label.

    array[postition] = value;

    lookups[postition].setText(Integer.toString(value));

    // Prints the updated data depending on the tracked color.

    if (color == ThreadSyncGUI.BLUE) {

      lookups[postition].setBackground(Color.decode("#1D70A2"));

      System.out.println("[" + Chalk.on(Thread.currentThread().getName().toUpperCase()).cyan() + "] wrote "

          + Chalk.on(Integer.toString(value)).green() + " in the element number "

          + Chalk.on(Integer.toString(postition)).green());

    } else {

      lookups[postition].setBackground(Color.decode("#F7B626"));

      System.out.println("[" + Chalk.on(Thread.currentThread().getName().toUpperCase()).yellow() + "] wrote "

          + Chalk.on(Integer.toString(value)).green() + " in the element number "

          + Chalk.on(Integer.toString(postition)).green());

    }

    // Increases the index.

    ++index;

    System.out.println("[" + Chalk.on("NEXT").magenta() + "] " + index);

  }

  // Overrides the toString method. Prints the actual elements in the array.

  public *String* toString() {

*String* arrayString = "\n[" + Chalk.on("SIMPLE ARRAY CONTENT").magenta() + "]\n[";

    for (*int* i = 0; i < array.length; i++)

      arrayString += (i != array.length - 1) ? array[i] + ", " : array[i];

    arrayString += "]\n";

    return arrayString;

  }

}

**Clase App**

package com.milkyblue;

// App class.

public class App {

    // Creates an anonymous instance of ThreadSyncGUI.

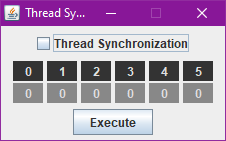
    public static *void* main(*String*[] *args*) {

**new** ThreadSyncGUI();

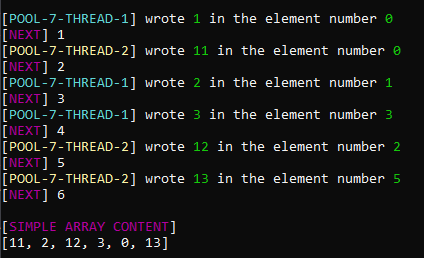
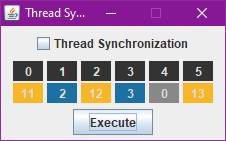
    }

}

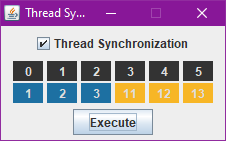
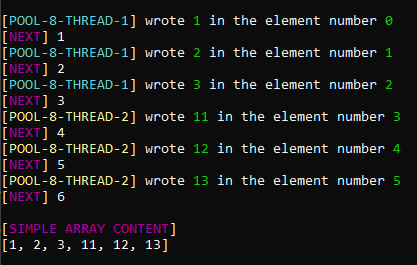
**CAPTURAS:**



**ESTADO INICIAL DE LA INTERFAZ.**



**LLENADO DEL ARREGLO CON EL METODO NO SINCRONIZADO.**



**LLENADO DEL ARREGLO CON EL METODO SINCRONIZADO.**

**PREGUNTAS:**

1. **¿Crees que es importante la sincronización?**

Si, porque nos permite evitar resultados inesperados al momento de acceder a un mismo recurso a través de subprocesos ejecutados en paralelo, permitiendo que exista un mejor orden en la ejecución de dichos subprocesos.

1. **Da un ejemplo real donde crees que esto podría funcionar.**

Cuando deseemos actualizar una base de datos desde distintos clientes, por ejemplo, un programa que lleve el control del inventario de alguna tienda pero que se use en varias terminales por distintos usuarios a la vez. Puede suceder que varios usuarios hagan uso de un mismo recurso al mismo tiempo, por lo que si se accede de manera sincronizada se evitaran resultados inesperados.

1. **¿Cuándo no sería necesaria la sincronización de subprocesos?**

Cuando no dependemos de recursos que hagan seguimiento a algún valor en base a las veces que este se modifica, por ejemplo, si quisiéramos acceder a un método que simplemente nos ayuda a obtener un cálculo, de esta manera no hay problema si varios subprocesos hacen uso de el al mismo tiempo, ya que el resultado obtenido no dependerá de si esta siendo utilizado por otros subprocesos.

**CONCLUSION:**

La implementación de subprocesos sincronizados en nuestras aplicaciones es una herramienta más que hace más flexible y útil la funcionalidad de estas mismas, es algo que siempre se debe de tener en cuenta para mejorar procesos y que estos se realicen de una manera óptima, en este caso en especifico cuando sabemos que vamos a utilizar un mismo recurso por distintos subprocesos.

**NOTAS:**

* Puede encontrar el repositorio de este proyecto en mi cuenta de github en el siguiente enlace: <https://github.com/NoisyApple/AdTopics-15.ThreadSynchronization>