INGENIERIA EN SISTEMAS COMPUTACIONALES

TOPICOS AVANZADOS DE PROGRAMACION

REPORTE - MULTITHREADING

ALUMNO:

LEONEL ALEJANDRO AGUIRRE SERRANO

PROFESOR

ING. LUIS EDUARDO GUTIERREZ AYALA

LEÓN, GUANAJUATO A 12 DE MAYO DEL 2020

REDACCION DEL PROBLEMA:

El problema presentado en este reporte consiste en la creación de varios programas que ejemplifiquen el uso de los hilos en java. Para mayor comodidad se añadió un programa principal que ejecuta cada uno de los ejemplos sin la necesidad de modificar directamente el código.

CODIGO FUENTE:

Clase CounterSelector

```
package com.milkyblue;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.BorderFactory;
import javax.swing.BoxLayout;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JRadioButton;

// CounterSelector class.
public class CounterSelector {
  private JFrame mainFrame;
  private JPanel mainPanel, topPanel, centerPanel, bottomPanel;
  private JLabel lblSelect;
  private String[] options;
  private JRadioButton[] rButtons;
  private ButtonGroup group;
  private JButton btnSelect;

  // Class constructor.
  public CounterSelector() {
    mainFrame = new JFrame("Counter Selector");
    mainPanel = new JPanel(new BorderLayout());
    topPanel = new JPanel();
    centerPanel = new JPanel();
    bottomPanel = new JPanel();
    lblSelect = new JLabel("Select a counter:");
```

```
    options = new String[] { "No thread counter", "Inner thread counter", "Runnable counte
r", "Multi thread counter",
        "Countdown counter" };
    rButtons = new JRadioButton[options.length];

    // Initializes each element in radio button array based on option array.
    for (int i = 0; i < rButtons.length; i++) {
      rButtons[i] = new JRadioButton(options[i]);
      rButtons[i].setActionCommand(Integer.toString(i));
    }

    group = new ButtonGroup();
    btnSelect = new JButton("Select");

    // Main methods are called.
    addAttributes();
    addListeners();
    build();
    launch();
  }

  // Adds attributes to elements in the class.
  private void addAttributes() {
    topPanel.setPreferredSize(new Dimension(200, 25));

    // Adds radio buttons to button group.
    for (JRadioButton rBtn : rButtons)
      group.add(rBtn);

    rButtons[0].setSelected(true);
    centerPanel.setBorder(BorderFactory.createEmptyBorder(0, 10, 0, 10));
    centerPanel.setLayout(new BoxLayout(centerPanel, BoxLayout.Y_AXIS));
    mainFrame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
    mainFrame.setResizable(false);
  }

  // Adds listeners to GUI events.
  private void addListeners() {
    // Creates a new instance based on the selected radio button label, then
    // disposes this window.
    btnSelect.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        switch (Integer.parseInt(group.getSelection().getActionCommand())) {
          case 0:
            new NoThreadCounter();
            break;
```

```
          case 1:
            new InnerThreadCounter();
            break;
          case 2:
            new RunnableCounter();
            break;
          case 3:
            new MultiThreadCounter(1);
            break;
          case 4:
            new CountDownCounter();
            break;
        }
        mainFrame.dispose();
      }
    });
  }

  // Builds the GUI.
  private void build() {
    topPanel.add(lblSelect);

    // Adds the radio buttons to the centerPanel.
    for (JRadioButton rBtn : rButtons)
      centerPanel.add(rBtn);

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

  // Launches the window by setting its visible value to true. Then its resized
  // and centered.
  private void launch() {
    mainFrame.setVisible(true);
    mainFrame.pack();
    mainFrame.setLocationRelativeTo(null);
  }
}
```

Clase NoThreadCounter

```
package com.milkyblue;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

// Class NoThreadCounter.
public class NoThreadCounter {
  private int count;
  private JFrame mainFrame;
  private JPanel mainPanel;
  private JButton btnStart, btnStop;
  private JTextField txtCount;
  private boolean isRunning;

  // Class constructor.
  public NoThreadCounter() {
    count = 0;
    mainFrame = new JFrame("No Thread Counter");
    mainPanel = new JPanel();
    btnStart = new JButton("Start");
    btnStop = new JButton("Stop");
    txtCount = new JTextField(10);
    isRunning = true;

    // Main methods are called.
    addAttributes();
    addListeners();
    build();
    launch();
  }

  // Adds attributes to elements in the class.
  private void addAttributes() {
    txtCount.setText(Integer.toString(count));
    mainFrame.setResizable(false);
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
```

```java
// Adds functionality to events.
private void addListeners() {
  // Simply calls the run method when pressed.
  btnStart.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      run();
    }
  });

  // Changes the state of isRunning, therefore the run method stops.
  btnStop.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      isRunning = false;
    }
  });
}

// Builds the program's GUI.
private void build() {
  mainPanel.add(txtCount);
  mainPanel.add(btnStart);
  mainPanel.add(btnStop);
  mainFrame.add(mainPanel);
}

// Launches the program by setting the frame's visible value to true, then
// resizes and centers the window.
private void launch() {
  mainFrame.setVisible(true);
  mainFrame.pack();
  mainFrame.setLocationRelativeTo(null);
}

// Runs the counter and sets isRunning to true, if it is changed again to
// false, the while loop stops and so the counter does too.
public void run() {
  btnStart.setEnabled(false);
  btnStop.setEnabled(true);
  isRunning = true;
  while (true) {
    try {
      Thread.sleep(100);
    } catch (Exception e) {
      System.out.println("Interrupted");
    }
```

```
      if (isRunning) {
        txtCount.setText(Integer.toString(count++));
      } else {
        break;
      }
    }
    btnStart.setEnabled(true);
  }

}
```

Clase InnerThreadCounter

```
package com.milkyblue;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;

// Class InnerThreadCounter.
public class InnerThreadCounter {
  private int count;
  private JFrame mainFrame;
  private JPanel mainPanel;
  private JButton btnStart, btnStop;
  private JTextField txtCount;
  private boolean isRunning;

  // Class constructor.
  public InnerThreadCounter() {
    count = 0;
    mainFrame = new JFrame("Inner Thread Counter");
    mainPanel = new JPanel();
    btnStart = new JButton("Start");
    btnStop = new JButton("Stop");
    txtCount = new JTextField(10);
    isRunning = true;

    // Main methods are called.
    addAttributes();
```

```
  addListeners();
  build();
  launch();
}

// Attributes are added to the elements in the class.
private void addAttributes() {
  txtCount.setText(Integer.toString(count));
  btnStop.setEnabled(false);
  mainFrame.setResizable(false);
  mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

// Adds listeners to GUI events.
private void addListeners() {
  // Creates a new instance of CountThread, which starts the counter.
  btnStart.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      new CountThread();
    }
  });

  // Stops the counter by setting isRunning to false.
  btnStop.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      isRunning = false;
      btnStop.setEnabled(false);
    }
  });
}

// Builds the GUI.
private void build() {
  mainPanel.add(txtCount);
  mainPanel.add(btnStart);
  mainPanel.add(btnStop);
  mainFrame.add(mainPanel);
}

// Launches the window by setting its visible value to true, the it is centered
// and resized.
private void launch() {
  mainFrame.setVisible(true);
  mainFrame.pack();
  mainFrame.setLocationRelativeTo(null);
}
```

```
  // CountThread class.
  class CountThread extends Thread {

    // Class constructor. Just starst the thread, which calls the run method.
    public CountThread() {
      start();
    }

    // Triggers the main functionality of the counter. Is stopped when isRunning is
    // set to false.
    public void run() {
      btnStart.setEnabled(false);
      btnStop.setEnabled(true);
      isRunning = true;
      while (true) {
        try {
          Thread.sleep(100);
        } catch (Exception e) {
          System.out.println("Interrupted");
        }
        if (isRunning) {
          txtCount.setText(Integer.toString(count++));
        } else {
          break;
        }
      }
      btnStart.setEnabled(true);
    }
  }
}
```

Clase RunnableCounter

```
package com.milkyblue;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

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

```
// Class RunnableCounter.
public class RunnableCounter implements Runnable {
  private int count;
  private JFrame mainFrame;
  private JPanel mainPanel;
  private JButton btnStart, btnStop;
  private JTextField txtCount;
  private boolean isRunning;
  private Thread selfThread;

  // Class constructor.
  public RunnableCounter() {
    count = 0;
    mainFrame = new JFrame("Runnable Counter");
    mainPanel = new JPanel();
    btnStart = new JButton("Start");
    btnStop = new JButton("Stop");
    txtCount = new JTextField(10);
    isRunning = true;
    selfThread = null;

    // Main methods are called.
    addAttributes();
    addListeners();
    build();
    launch();
  }

  // Adds attributes to element in the class.
  private void addAttributes() {
    txtCount.setText(Integer.toString(count));
    btnStop.setEnabled(false);
    mainFrame.setResizable(false);
    mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }

  // Adds listeners to GUI events.
  private void addListeners() {
    // Defines the selfThread based on the actual instance of RunnableCounter, then
    // the Thread is started which also starts the counter.
    btnStart.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        selfThread = new Thread(RunnableCounter.this);
        selfThread.start();
      }
    });
```

```
    // Stops the counter by setting isRunning to false.
    btnStop.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        isRunning = false;
        btnStop.setEnabled(false);
      }
    });
  }

  // Builds the GUI.
  private void build() {
    mainPanel.add(txtCount);
    mainPanel.add(btnStart);
    mainPanel.add(btnStop);
    mainFrame.add(mainPanel);
  }

  // Launches the window by setting its visible value to true, then it is resized
  // and centered.
  private void launch() {
    mainFrame.setVisible(true);
    mainFrame.pack();
    mainFrame.setLocationRelativeTo(null);
  }

  // Triggers the main functionality of the counter. Its stopped when isRunning is
  // set to false.
  public void run() {
    btnStart.setEnabled(false);
    btnStop.setEnabled(true);
    isRunning = true;
    while (true) {
      try {
        Thread.sleep(100);
      } catch (Exception e) {
        System.out.println("Interrupted");
      }
      if (isRunning) {
        txtCount.setText(Integer.toString(count++));
      } else {
        break;
      }
    }
    btnStart.setEnabled(true);
  }
}
```

Clase MultiThreadCounter

```
package com.milkyblue;

import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Stack;

import javax.swing.BoxLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTextField;

// MultiThreadCounter class.
public class MultiThreadCounter {
  private JFrame mainFrame;
  private JPanel mainPanel, topPanel, centerPanel;
  private JLabel lblAdd;
  private JButton btnAdd;
  private Stack<Counter> counters;

  // Class constructor.
  public MultiThreadCounter(int initialCounters) {
    mainFrame = new JFrame("Multithread Counter");
    mainPanel = new JPanel(new BorderLayout());
    topPanel = new JPanel();
    centerPanel = new JPanel();
    lblAdd = new JLabel("Add a new counter: ");
    btnAdd = new JButton("ADD");
    counters = new Stack<Counter>();

    // Push as many Counter objects as specified in initialCounters to the counters
    // stack.
    for (int i = 0; i < initialCounters; i++)
      counters.push(new Counter());

    // Main methods are called.
    addAttributes();
    addListeners();
    build();
    launch();
  }
```

```java
// Adds attributes to elements in class.
private void addAttributes() {
  centerPanel.setLayout(new BoxLayout(centerPanel, BoxLayout.Y_AXIS));
  mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  mainFrame.setResizable(false);
}

// Adds listeners to events in the GUI.
private void addListeners() {
  // Adds a new Counter object to the GUI and pushes it to the array, then resizes
  // the window.
  btnAdd.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
      Counter newCounter = new Counter();
      counters.push(newCounter);
      centerPanel.add(newCounter);
      launch();
    }
  });
}

// Builds the GUI.
private void build() {
  topPanel.add(lblAdd);
  topPanel.add(btnAdd);

  // Adds every element in the stack to the centerPanel.
  for (Counter c : counters)
    centerPanel.add(c);

  mainPanel.add(topPanel, BorderLayout.NORTH);
  mainPanel.add(centerPanel, BorderLayout.CENTER);
  mainFrame.add(mainPanel);
}

// Launches the window by setting its visible value to true. Then the window is
// centered and resized.
private void launch() {
  mainFrame.setVisible(true);
  mainFrame.pack();
  mainFrame.setLocationRelativeTo(null);
}

// Counter inner Class, inherits JPanel to model a Counter GUI and runs on a
// dedicated Thread.
@SuppressWarnings("serial")
class Counter extends JPanel implements Runnable {
```

```
  private int count;
  private JTextField txtCount;
  private JButton btnStart, btnStop;
  private boolean isRunning;
  private Thread selfThread;

  // Class constructor.
  public Counter() {
    count = 0;
    txtCount = new JTextField(10);
    btnStart = new JButton("Start");
    btnStop = new JButton("Stop");
    isRunning = false;
    selfThread = null;

    // Main methods are called.
    this.addAttributes();
    this.addListeners();
    this.build();
  }

  // Adds attributes to the elements in the class.
  private void addAttributes() {
    txtCount.setText(Integer.toString(count));
    btnStop.setEnabled(false);
  }

  // Adds listeners to the elements in the class.
  private void addListeners() {
    // Declares a Thread based on the instance on this class and starts it.
    // Therefore the Counter is started.
    btnStart.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        selfThread = new Thread(Counter.this);
        selfThread.start();
      }
    });

    // Stops the Counter by setting isRunning to false.
    btnStop.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        isRunning = false;
        btnStop.setEnabled(false);
      }
    });
  }
```

```
    // Builds the inner Class GUI.
    private void build() {
      add(txtCount);
      add(btnStart);
      add(btnStop);
    }

    // Triggers the main functionality of the Counter, when isRunning is set to
    // false, the while loop is stopped and therefore the Counter too.
    public void run() {
      btnStart.setEnabled(false);
      btnStop.setEnabled(true);
      isRunning = true;
      while (true) {
        try {
          Thread.sleep(100);
        } catch (Exception e) {
          System.out.println("Interrupted");
        }
        if (isRunning) {
          txtCount.setText(Integer.toString(count++));
        } else {
          break;
        }
      }
      btnStart.setEnabled(true);
    }
  }
}
```

Clase CountDownCounter

```
package com.milkyblue;

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

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

```java
// CountDownCounter class.
public class CountDownCounter {
  private JFrame mainFrame;
  private JPanel mainPanel;
  private JLabel lblCDown;
  private JButton btnCDown;

  // Class constructor.
  public CountDownCounter() {
    Chalk.setColorEnabled(true);

    mainFrame = new JFrame("Countdown counter");
    mainPanel = new JPanel();
    lblCDown = new JLabel("Add a new countdown");
    btnCDown = new JButton("Add");

    // Main methods are called.
    addAttributes();
    addListeners();
    build();
    launch();
  }

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

  // Adds listeners to events from the GUI.
  private void addListeners() {
    // Creates a new instance of CountDown class when pressed.
    btnCDown.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        new CountDown();
      }
    });
  }

  // Builds the GUI.
  private void build() {
    mainPanel.add(lblCDown);
    mainPanel.add(btnCDown);
    mainFrame.add(mainPanel);
  }
```

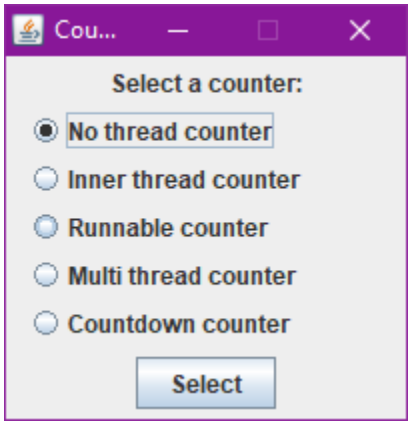
```
  // Launches the window by setting it visible value to true. Then the window is
  // resized and centered.
  private void launch() {
    mainFrame.setVisible(true);
    mainFrame.pack();
    mainFrame.setLocationRelativeTo(null);
  }
}

// CountDown Class.
class CountDown extends Thread {
  private int countDown;
  private static int idCount = 0;
  private int id;

  // Class constructor, assign an unique id to the instance and starts the Thread.
  public CountDown() {
    countDown = (int) Math.floor(Math.random() * 10) + 10;
    id = ++idCount;
    // Countdown starts.
    coloredPrint(id, "CREATED");
    start();
  }

  // Triggers the main functionality of the Countdown,
  public void run() {
    while (true) {
      try {
        Thread.sleep(500);
      } catch (Exception e) {
        e.printStackTrace();
      }
      // Prints the countdown.
      coloredPrint(id, Integer.toString(countDown));
      if (--countDown <= 0) {
        // The countdown is completed.
        coloredPrint(id, "DISPOSED");
        break;
      }
    }
  }

  // Utility method. Prints a custom format message on console with a color
  // determined by the id passed.
  private void coloredPrint(int id, String message) {
    Chalk coloredMsg = null;
```

```
    switch (id % 3) {
      case 0:
        coloredMsg = Chalk.on("Thread-" + id).cyan();
        break;
      case 1:
        coloredMsg = Chalk.on("Thread-" + id).yellow();
        break;
      case 2:
        coloredMsg = Chalk.on("Thread-" + id).magenta();
        break;
    }
    System.out.println("[" + coloredMsg + "] " + message);
  }
}
```

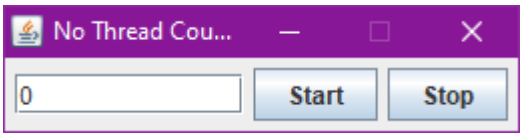
Clase App

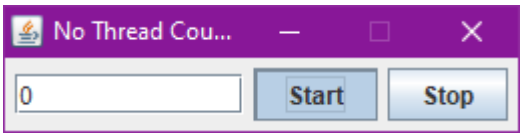
```
package com.milkyblue;

// App class.
public class App {
    // Runs a new instance of CounterSelector class.
    public static void main(String[] args) {
        new CounterSelector();
    }
}
```

CAPTURAS:

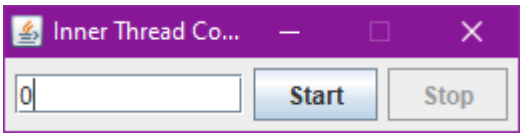


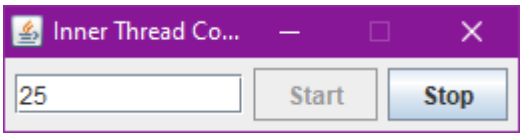
VENTANA PRINCIPAL DE SELECCION DE LOS PROGRAMAS EJEMPLO.



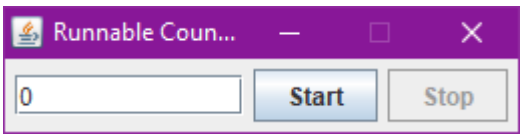


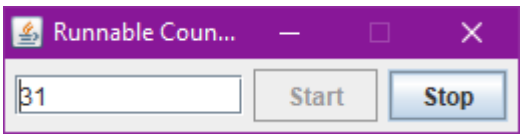
PRIMER EJEMPLO – EL CONTADOR CORRE EN EL HILO PRINCIPAL Y POR LO TANTO LA INTERFAZ QUEDA CONGELADA.



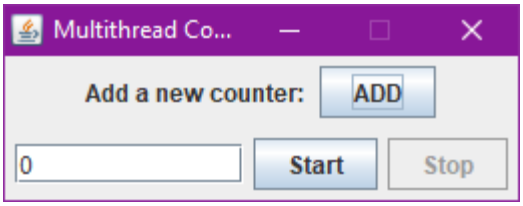


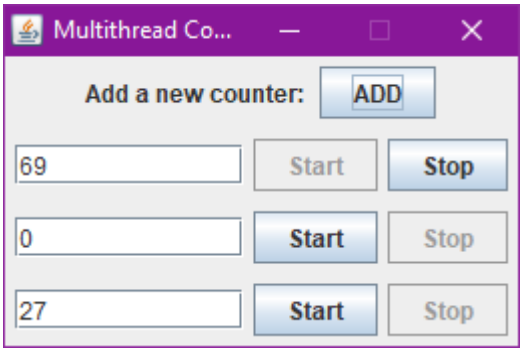
SEGUNDO EJEMPLO – USANDO UNA CLASE INTERNA QUE HEREDA DE "THREAD" PARA EJECUTAR EL CONTADOR EN UN HILO DEDICADO.



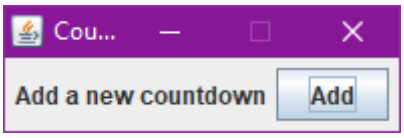


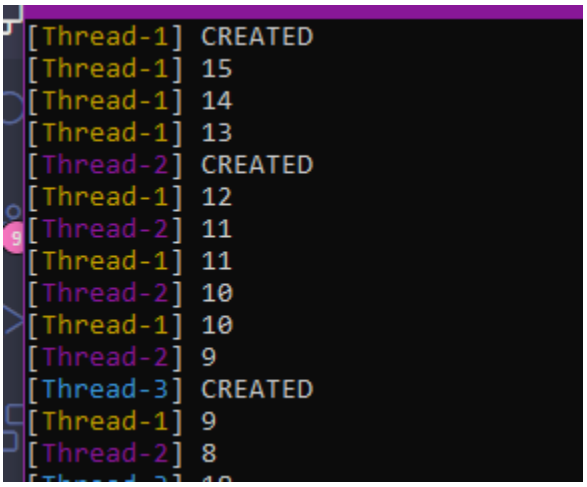
TERCER EJEMPLO – USANDO LA INTERFAZ "RUNNABLE" PARA EJECUTAR EL CONTADOR EN UN HILO DEDICADO.

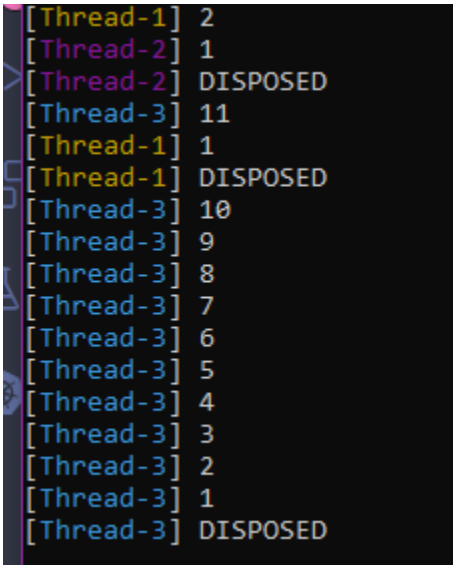




CUARTO EJEMPLO – IMPLEMENTANDO MULTITHREADING PARA CREAR VARIAS INSTANCIAS DE UN CONTADOR, CADA UNA CON UN HILO DEDICADO (UTILIZANDO INTERFAZ "RUNNABLE").







QUINTO EJEMPLO - IMPLEMENTANDO MULTITHREADING PARA CREAR VARIAS INSTANCIAS DE UN CONTADOR DE CUENTA REGRESIVA, CADA UNA CON UN HILO DEDICADO (EXTENDIENDO DE "THREAD").

CONCLUSION:

El uso de hilos múltiples en un programa es una herramienta imprescindible, ya que, si bien en algunos casos no los llegamos a necesitar, conforme nuestra aplicación va creciendo eventualmente estos se tendrán que hacer presentes para ayudarnos a solucionar problemas y optimizar nuestro código. Además, este concepto resulta necesario al momento de modelar una Interfaz Grafica de Usuario, debido a que lo optimo es que el usuario pueda realizar varias tareas al mismo tiempo, sin tener que esperar a terminar una antes de comenzar otra.

NOTAS:

- Puede encontrar el repositorio de este proyecto en mi cuenta de github en el siguiente enlace: https://github.com/NoisyApple/AdTopics-13.Multithreading/