****



**TECNOLÓGICO NACIONAL DE MÉXICO**

**Instituto Tecnológico de León**

**INGENIERIA EN SISTEMAS COMPUTACIONALES**

**TOPICOS AVANZADOS DE PROGRAMACION**

**REPORTE - EXECUTORSERVICE**

**ALUMNO:**

LEONEL ALEJANDRO AGUIRRE SERRANO

**PROFESOR**

ING. LUIS EDUARDO GUTIERREZ AYALA

**LEÓN, GUANAJUATO A 16 DE MAYO DEL 2020**

**REDACCION DEL PROBLEMA:**

El problema presentado en este reporte consiste en la creación de un programa que implemente un objeto **ExecutorService**, y por medio de este ejecutar varias tareas y observar su comportamiento.

**CODIGO FUENTE:**

**Clase ExecutorLauncherGUI**

package com.milkyblue;

import java.awt.BorderLayout;

import java.awt.event.ActionListener;

import java.awt.event.ActionEvent;

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;

import com.github.tomaslanger.chalk.Chalk;

// ExecutorLauncherGUI class. Models a GUI that launches a CustomExecutorService

// instance based on the selected mode.

public class ExecutorLauncherGUI {

  private *JFrame* mainFrame;

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

  private *JLabel* lblAdvice;

  private *ButtonGroup* buttonGroup;

  private *JRadioButton* radioButtons[];

  private *JButton* btnLaunch;

  // Class constructor.

  public ExecutorLauncherGUI() {

    // Enables color in the console.

    Chalk.setColorEnabled(true);

    mainFrame = **new** JFrame("Executor Service");

    mainPanel = **new** JPanel(**new** BorderLayout());

    topPanel = **new** JPanel();

    centerPanel = **new** JPanel();

    bottomPanel = **new** JPanel();

    lblAdvice = **new** JLabel("Select an executor service mode:");

    buttonGroup = **new** ButtonGroup();

    // Stores the labels for the radio buttons.

*String*[] radioButtonLabels = **new** *String*[] { "Cached Thread Pool", "Single Thread", "Fixed Thread Pool" };

    // Sets radioButtons to an array with the same length of radioButtonLabels.

    radioButtons = **new** *JRadioButton*[radioButtonLabels.length];

    // Initializes each radio button in the array and assings an action command

    // based on the actual index.

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

      radioButtons[i] = **new** JRadioButton(radioButtonLabels[i]);

      radioButtons[i].setActionCommand(Integer.toString(i + 1));

    }

    btnLaunch = **new** JButton("Launch");

    // Main methods are called.

    addAttributes();

    addListeners();

    build();

    launch();

  }

  // Adds attributes to elements in the class.

  private *void* addAttributes() {

    centerPanel.setLayout(**new** BoxLayout(centerPanel, BoxLayout.Y\_AXIS));

    centerPanel.setBorder(BorderFactory.createEmptyBorder(5, 10, 5, 10));

    // Selects by default the first element in the radio buttons array.

    radioButtons[0].setSelected(true);

    // Adds all the radio buttons to the ButtonGroup.

    for (*JRadioButton* radioBtn : radioButtons)

      buttonGroup.add(radioBtn);

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

    mainFrame.setResizable(false);

  }

// Adds listeners to the elements in the class.

  private *void* addListeners() {

    // Creates an anonymous instance of CustomExecutorService with 6 tasks and the

    // selected mode.

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

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

**new** CustomExecutorService(6, Integer.parseInt(buttonGroup.getSelection().getActionCommand()));

      }

    });

  }

  // Builds the GUI.

  private *void* build() {

    topPanel.add(lblAdvice);

    for (*JRadioButton* radioBtn : radioButtons)

      centerPanel.add(radioBtn);

    bottomPanel.add(btnLaunch);

    mainPanel.add(topPanel, BorderLayout.NORTH);

    mainPanel.add(centerPanel, BorderLayout.CENTER);

    mainPanel.add(bottomPanel, BorderLayout.SOUTH);

    mainFrame.add(mainPanel);

  }

  // Launches the frame by setting its visible value to true. Resizes and

  // relocates the window.

  private *void* launch() {

    mainFrame.setVisible(true);

    mainFrame.pack();

    mainFrame.setLocationRelativeTo(null);

  }

}

**Clase CustomExecutorService**

package com.milkyblue;

import java.util.concurrent.ExecutorService;

import java.util.concurrent.Executors;

import com.github.tomaslanger.chalk.Chalk;

// CustomExecutorService class. Models an ExecutorService with custom behaviour,

// being capable of define the amount of tasks managed by the Executor and also the mode.

public class CustomExecutorService {

  // Static constants that refers to each ExecutorService mode.

  public static final *int* CACHED\_THREAD\_POOL = 1, SINGLE\_THREAD = 2, DOUBLE\_THREAD = 3;

  private *PrintableTask*[] tasks;

  private *ExecutorService* threadExecutor;

  // Class constructor. Takes an amount of tasks to be executed and the

  // ExecutorService mode.

  public CustomExecutorService(*int* *taskAmount*, *int* *execMode*) {

    tasks = **new** *PrintableTask*[taskAmount];

    // Defines all the tasks in the array with a new instance of PrintableTask.

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

      tasks[i] = **new** PrintableTask("Task " + (i + 1));

    // Defines the ExecutorService object based on the selected mode.

*String* mode = "";

    switch (execMode) {

      case CACHED\_THREAD\_POOL:

        threadExecutor = Executors.newCachedThreadPool();

        mode = "CACHED THREAD POOL";

        break;

      case SINGLE\_THREAD:

        threadExecutor = Executors.newSingleThreadExecutor();

        mode = "SINGLE THREAD";

        break;

      case DOUBLE\_THREAD:

        // Amount passed to the fixed thread ExecutorService.

*int* amount = 2;

        threadExecutor = Executors.newFixedThreadPool(amount);

        mode = "FIXED THREAD POOL (SET TO " + amount + ")";

        break;

    }

    System.out.println("[" + Chalk.on("EXECUTOR SERVICE").magenta() + "] " + mode);

    start();

  }

  // Executes all tasks in the array with the ExecutorService, when all tasks are

  // done the ExecutorService its shutted down.

  private *void* start() {

    for (*PrintableTask* task : tasks)

      threadExecutor.execute(task);

    threadExecutor.shutdown();

  }

}

**Clase PrintableTask**

package com.milkyblue;

import java.util.Random;

import com.github.tomaslanger.chalk.Chalk;

// PrintableTask Class. Models a taks that waits a random amount of time between 0 and 5 seconds.

public class PrintableTask implements *Runnable* {

  private final *int* inactivityTime;

  private final *String* taskName;

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

  // Class constructor.

  public PrintableTask(*String* *taskName*) {

    // The random amount of time its generated.

    inactivityTime = generator.nextInt(5000);

*this*.taskName = taskName;

  }

  // Method called when a Thread based on class instance gets executed. Prints

  // when it is created and when the inactivity time has passed.

  public *void* run() {

    try {

      System.out.println(

          "[" + Chalk.on(taskName).cyan() + "] " + " will be inactive for " + inactivityTime + " milliseconds");

      Thread.sleep(inactivityTime);

    } catch (*InterruptedException* *e*) {

      System.out.println("[" + Chalk.on(taskName).red() + "] " + " stopped prematurely due to an interruption");

    }

    System.out.println("[" + Chalk.on(taskName).green() + "] " + " finished its inactivity");

  }

}

**Clase RunnableCounter**

package com.milkyblue;

// App class. Launches the GUI.

public class App {

    // Creates an anonymous instance of ExecutorLauncherGUI.

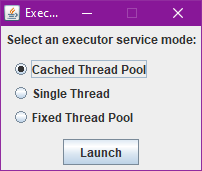
    public static *void* main(*String*[] *args*) {

**new** ExecutorLauncherGUI();

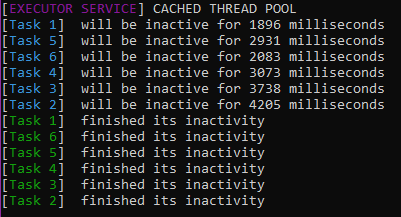
    }

}

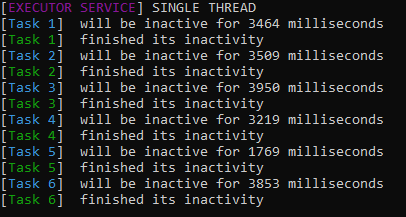
**CAPTURAS:**



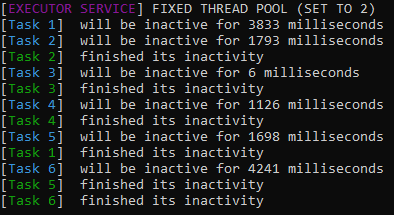
**VENTANA PRINCIPAL DE SELECCION DE LOS MODOS DEL EXECUTORSERVICE.**



**EXECUTORSERVIDE CON 6 TAREAS EN MODO “CACHED THREAD POOL”.**



**EXECUTORSERVIDE CON 6 TAREAS EN MODO “SINGLE THREAD”.**



**EXECUTORSERVIDE CON 6 TAREAS EN MODO “FIXED THREAD POOL” (CON UNA CANTIDAD DE 2 SUBPROCESOS).**

**PREGUNTAS:**

1. **¿Qué cambios notas en la ejecución del programa?**

Cuando el ExecutorService se define con:

* **newCachedThreadPool():** todas las tareas asignadas se ejecutan a la vez, sin tener un límite.
* **newSingleThreadExecutor():** el ejecutor se limita a ejecutar una tarea a la vez, cuando termina una entonces comienza con la siguiente.
* **newFixedThreadPool():** de manera similar al anterior, el ejecutor se limitara a una cantidad de tareas a la vez, dicha cantidad se pasa como argumento, por lo tanto es controlable.

1. **Cambia el parámetro en ...newFixedThreadPool(#); a tu gusto y observa el funcionamiento, ¿Que conclusión sacas del valor colocado como argumento?**

Es evidente que el valor que se pasa como argumento define el limite de tareas que el Ejecutor manejara a la vez.

1. **¿Qué ocurre cuando colocamos uno en el inciso 2)?**

El ejecutor trabajara de la misma manera que con el método “newSingleThreadExecutor” ya que se limita a una sola tarea a la vez.

1. **¿Podemos poner un valor negativo en el inciso 2)?**

Se genera una excepción ya que un ejecutor ejecute “-n” tareas a la vez.

1. **Mueve la línea 26 del programa original en la posición de la linea 23, ¿Qué ocurre? ¿Cuál será la razón?**

Debido a que el ejecutor se “apaga” previo a que otras tareas sean ejecutadas, cuando estas intentan ser ejecutadas se genera un error, ya que el ejecutor ya no se encuentra disponible.

**CONCLUSION:**

El uso de un ExecutorService nos da una mayor flexibilidad como desarrollador de controlar los recursos de los que dispone el programa, ya que el ejecutor nos permite ejecutar una serie de subprocesos limitada, evitando una sobrecarga de procesos ejecutados simultáneamente en la memoria de la computadora, lo que además de ser mas eficiente, mejora considerablemente el rendimiento de nuestro software.

**NOTAS:**

* Puede encontrar el repositorio de este proyecto en mi cuenta de github en el siguiente enlace: <https://github.com/NoisyApple/AdTopics-14.ExecutorService/>