UNIVERSIDAD TECNOLÓGICA DE PANAMÁ

FACULTAD DE INGENIERÍA DE SISTEMAS COMPUTACIONALES

DEPARTAMENTO DE INGENIERÍA DE SOFTWARE

LIC. INGENIERÍA DE SOFTWARE

APLICACIONES PARA AMBIENTES DISTRIBUIDOS

LABORATORIO #1

SEBASTIÁN SÁNCHEZ 8-944-2002

FACILITADOR:
Prof. Regis Rivera

Grupo 1SF251

ABRIL, 2025

Introducción

En este laboratorio estaremos observando el comportamiento de la programación concurrente a través de 3 ejemplos en java. La programación concurrente es un paradigma de la programación que implica realizar varios hilos en un proceso. Un hilo es la secuencia de problemas a resolver para cumplir con el proceso, al tener varios hilos en concurrencia podemos mejorar ciertos aspectos del programa.

Problema 1

```
import java.lang.Math;

// Each instance of this class will be a separate thread
class EjemploThread extends Thread {
    int numero;

    // Constructor: receives and stores a number to print later
    EjemploThread(int n) {
        numero = n;
    }

    // This method is run when the thread starts
    public void run() {
        try {
            while (true) { // Infinite loop
                System.out.println(numero); // Print the thread's assigned number
                // Sleep for a random time between 0 and 1000 milliseconds (1 second)
                sleep((long)(1000 * Math.random()));
            }
        } catch (InterruptedException e) {
            return; // Ends the thread if interrupted
        }
    }
}

// Main class to launch the program
public class Main {
    public static void main(String args[]) {
        // Start 10 threads, each with a different number from 0 to 9
        for (int i = 0; i < 10; i++) {
            new EjemploThread(i).start(); // Each thread begins running its 'run()' method
        }
    }
}
```

- Explique la salida del programa
    - Cada hilo está constantemente imprimiendo su “numero” y reposa por una cantidad al azar de tiempo. Se imprimen números de 0 al 9 pero el orden de los números no aparece en un orden en particular debido a que los hilos se están ejecutando concurrentemente.

- Mencione cuantos hilos se ejecutaron
    - Se ejecuta un hilo por cada instancia de 10 numeros
- Qué haría si se quisiera agregar un segundo hilo al programa. Explique
    - Según la lógica del programa se ejecutaría en la función run, para que cuando se ejecute el main, esta lógica pueda ser aplicada.

Problema 2

```
import java.util.concurrent.*;

public class EjemploConcurrenciaJava {
    public static void main(String[] args) {
        // Crear un ExecutorService para administrar hilos
         // Executors.newFixedThreadPool(3) crea un pool de hilos fijo con 3 hilos disponibles.
        ExecutorService executor = Executors.newFixedThreadPool(3);

        // Crear tareas (Runnable) que se ejecutaran en paralelo
        // Runnable es una interfaz funcional que representa una tarea que puede ser ejecutada por un hilo.
        Runnable tarea1 = () -> {
            // Esta es la implementación de la tarea 1.
            for (int i = 0; i < 5; i++) {
                System.out.println("Tarea 1: " + i);
            }
        };

        Runnable tarea2 = () -> {
            // Esta es la implementación de la tarea 2.
            for (int i = 0; i < 5; i++) {
                System.out.println("Tarea 2: " + i);
            }
        };

        Runnable tarea3 = () -> {
            // Esta es la implementación de la tarea 3.
            for (int i = 0; i < 5; i++) {
                System.out.println("Tarea 3: " + i);
            }
        };

        // Ejecutar las tareas en hilos separados
         // executor.execute() envía una tarea Runnable al ExecutorService para su ejecución.
         // El ExecutorService se encarga de asignar un hilo disponible del pool para ejecutar la tarea.
        executor.execute(tarea1);
        executor.execute(tarea2);
        executor.execute(tarea3);
```

```
        // Cerrar el ExecutorService cuando ya no se necesite
            // Es importante llamar a shutdown() para liberar los recursos del ExecutorService
        // y permitir que la aplicación termine de forma limpia una vez que todas las tareas han finalizado.
        executor.shutdown();
    }
}
```

- explique la salida del programa
    - La salida del programa consistirá en la impresión intercalada de los mensajes de las tres tareas. Cada tarea tiene un bucle que imprime su nombre (Tarea 1, Tarea 2, Tarea 3) seguido de un número del 0 al 4.

      Debido a que las tres tareas se ejecutan de forma concurrente (en paralelo, en la medida en que los hilos del pool lo permitan), el orden exacto de las líneas de salida puede variar en cada ejecución. Sin embargo, siempre verás las 5 impresiones de cada tarea.

- Cuantos hilos se ejecutan y porque
    - En este programa, se ejecutan **3 hilos**. Esto se debe a la línea donde se crea el ExecutorService

      Executors.newFixedThreadPool(3) crea un **pool de hilos fijo** con una capacidad de **3 hilos**. Esto significa que el ExecutorService gestionará un máximo de 3 hilos para ejecutar las tareas que se le envíen.
      Cuando se llaman a executor.execute(tarea1), executor.execute(tarea2) y executor.execute(tarea3), el ExecutorService toma cada tarea y la asigna a uno de los 3 hilos disponibles en el pool. Si hubiera más tareas enviadas al executor mientras los 3 hilos estuvieran ocupados, esas tareas se encolarían y esperarían a que un hilo se liberara para poder ejecutarse.

- que haría si se quisiera agregar un hilo adicional al programa. explique.
    - Si quisiera agregar un hilo adicional para que el programa pudiera potencialmente ejecutar más tareas en paralelo (siempre y cuando el hardware subyacente lo permita), modificaría la creación del ExecutorService para aumentar el tamaño del pool de hilos.

Problema 3

- Explique la salida del programa
    - la salida será una mezcla desordenada de los números que resultan de los incrementos realizados por ambos hilos, debido a la falta de control sobre el acceso concurrente a la variable compartida n. Este ejemplo ilustra la importancia de la sincronización cuando se trabaja con recursos compartidos en entornos multihilo para evitar resultados inesperados e inconsistencias.
- Imagen de la ejecución del programa