LABORATORIO HILOS

Resolver los siguientes problemas usando Hilos, en cada problema se muestra un código de apoyo ya hecho, no se limite simplemente a copiar y pegar sino que lea detenidamente el código, haga un análisis y realice los ajustes que se piden.

Problema 1

Dibujar en un Panel de JavaFX dos matrices de rectángulos (de nxn). Para la primera matriz debe ir pintando cada rectángulo (cambiar el color del botón a rojo) fila por fila, para la segunda matriz debe ir pintando cada rectángulo (cambiar el color del botón a azul) de forma diagonal. Pero ambas se deben ir pintando al mismo tiempo. Para cada matriz se asigna una velocidad de llenado diferente.

Use el siguiente código para generar las matrices de forma automatizada:

1. Cree la clase `MatrizController` con el siguiente código:

```
package controller;

import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import java.net.URL;
import java.util.ResourceBundle;

public class MatrizController implements Initializable {

   @FXML
   private Pane panel;
   private Rectangle[][] matriz1, matriz2;

   @Override
   public void initialize(URL url, ResourceBundle resourceBundle) {

       //Se inicializan las dos matrices, una al lado de la otra
       matriz1 = crearMatrizBotones(4, 0);
       matriz2 = crearMatrizBotones(4, 250);
   }

   public void iniciarLlenado(){
       //Se llena la primer matriz al dar click en el botón de la ventana
       llenarMatriz1(matriz1);
   }

   private void llenarMatriz1(Rectangle[][] m) {

       for(int i = 0; i < m.length; i++){
           for(int j = 0; j < m.length; j++) {
               //En cada iteración se actualiza el color de fondo a rojo
               m[i][j].setFill( Color.RED );
               try {
                   Thread.sleep(1000);
               } catch (InterruptedException e) {
```

```
                    throw new RuntimeException(e);
                }
            }
        }
    }

    private Rectangle[][] crearMatrizBotones(int n, int xInicial){

        Rectangle[][] botones = new Rectangle[n][n];

        for(int i = 0; i < n; i++){
            for(int j = 0; j < n; j++){

                Rectangle boton = new Rectangle();
                boton.setWidth(50);
                boton.setHeight(50);
                boton.setFill( Color.WHITE );

                //En cada iteración se actualiza la posición del botón en el eje X, X cambia
cuando cambia el valor de j. Ya que x representa las columnas (horizontal).
                boton.setLayoutX( xInicial + (50+3)*j );

                //En cada iteración se actualiza la posición del botón en el eje Y, Y cambia
cuando cambia el valor de i. Ya que y representa las fila (vertical).
                boton.setLayoutY( (50+3)*i );

                botones[i][j] = boton;

                //El botón se añade al panel para poder visualizarlo
                panel.getChildren().add(boton);

            }
        }

        return botones;

    }

}
```

Analice el código anterior y observe qué es lo que se está haciendo allí.

2. Este controlador responde a un archivo `fxml` con la siguiente estructura:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane xmlns="http://javafx.com/javafx"
            xmlns:fx="http://javafx.com/fxml"
            fx:controller="controller.MatrizController"
            prefHeight="400.0" prefWidth="600.0">

   <Button onAction="#iniciarLlenado" text="Iniciar llenado" />
```

```
   <Pane fx:id="panel" layoutY="50.0" />

</AnchorPane>
```

3. Cree la clase `Aplicacion` y haga que se ejecute correctamente usando JavaFX. Ponga a correr el proyecto y observe qué pasa.

4. Haga que cuando se invoque el método `iniciarLlenado()`, se ejecute un hilo para llenar la primera matriz y otro hilo para que se llene la otra matriz. Recuerde seguir el orden de llenado descrito anteriormente.

   Para crear y ejecutar un hilo recuerde que puede usar el siguiente código:

```
new Thread(new Runnable() {
   @Override
   public void run() {
      //Hacer algo
   }
}).start();
```

Problema 2

Escribir un programa que busque un valor dentro de un archivo (txt) con 2 millones de líneas. Suponga que es un archivo con el registro de información de clientes. Haga un método que realice la búsqueda de un valor de forma secuencial (lo normal), luego programe un método que use 4 hilos, cada hilo debe buscar dicho valor en un rango equitativo de registros.

Para este problema se sugiere hacer lo siguiente:

1. Crear la clase `Principal`, esta clase debe estar en paquete `model` y debe tener un `ArrayList` de tipo `String` (que se llamará `clientes`) y un `String` con el valor a buscar dentro del archivo. El constructor de dicha clase debe recibir el valor buscado por parámetro y se lo debe asignar al `String` de la clase. Además, en el constructor se debe inicializar el `Arraylist` de clientes leyendo el archivo txt. Así debe quedar el constructor:

```
public Principal(String valorBuscado){
   this.clientes = new ArrayList<>();
   this.valorBuscado = valorBuscado;
   try {
       //Se leen los datos del archivo txt de clientes
       this.clientes =
ArchivoUtils.leerArchivoBufferedReader("src/main/resources/clientes.txt");
   } catch (IOException e) {
       throw new RuntimeException(e);
   }

}
```

2. En la misma clase, se debe crear el método para buscar el valor que el usuario desee de forma secuencial:

```

public void realizarBusquedaSecuencial(){
```

```
   boolean encontrado = false;
   int i;
   long tiempoInicial = System.currentTimeMillis();

   for(i = 0; i < clientes.size() && !encontrado; i++){
       if(clientes.get(i).contains(valorBuscado)){
           encontrado = true;
       }
   }

   long tiempoFinal = System.currentTimeMillis();
   System.out.println("Tiempo total gastado en la búsqueda: " + (tiempoFinal-tiempoInicial)/1000
+ " segundos. Hice "+i+" iteraciones");
}
```

3. Procure buscar un valor que esté en una posición muy alta. Haga la clase `Aplicacion` y agregue el `main`:

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

   Principal principal = new Principal("Nombre 1900100");
   principal.realizarBusquedaSecuencial();

}
```

Ejecute la aplicación y observe los resultados.

4. En la clase `Principal`, agregue un método que haga la búsqueda por hilos, así:

```
public void realizarBusquedaHilos(){

   BusquedaHilo[] hilos = new BusquedaHilo[4];

   //Se crean los 4 hilos con los rangos que deben usar para buscar el valor que se requiera
   hilos[0] = new BusquedaHilo(clientes, valorBuscado, 0, 500000);
   hilos[1] = new BusquedaHilo(clientes, valorBuscado, 500000, 1000000);
   hilos[2] = new BusquedaHilo(clientes, valorBuscado, 1000000, 1500000);
   hilos[3] = new BusquedaHilo(clientes, valorBuscado, 1500000, clientes.size());

   for(BusquedaHilo h : hilos){
       //Se ejecutan los hilos
       h.start();
   }

}
```

5. Para esto, debe crear la clase `BusquedaHilo` (en paquete `model`) así:

```
public class BusquedaHilo extends Thread {

   private final ArrayList<String> clientes;
   private final String cadenaBuscada;
```

```
    private final int posInicial;
    private final int posFinal;

    @Getter @Setter
    private boolean vivo;

    public BusquedaHilo(ArrayList<String> clientes, String cadenaBuscada, int posInicial, int
posFinal) {
        this.clientes = clientes;
        this.cadenaBuscada = cadenaBuscada;
        this.posInicial = posInicial;
        this.posFinal = posFinal;
        this.vivo = true;
    }

    @Override
    public void run() {

        boolean encontrado = false;
        long tiempoInicial = System.currentTimeMillis();
        int iteraciones = 0;

        for (int i = posInicial; i < posFinal && vivo; i++){
            if(clientes.get(i).contains(cadenaBuscada)){
                encontrado = true;
                //Como ya encontró el valor buscado se finaliza el for haciendo la variable false
                vivo = false;
            }
            iteraciones++;
        }

        long tiempoFinal = System.currentTimeMillis();
        String mensaje = encontrado ? "lo encontré" : "no lo encontré";

        System.out.println("Tiempo total gastado en la búsqueda:
"+(tiempoFinal-tiempoInicial)/1000+" segundos pero "+mensaje+". Hice "+iteraciones+"
iteraciones");

    }

}
```

6. Invoque el método `realizarBusquedaHilos()` en el `main` de la clase `Aplicacion`. No quite la invocación del método `realizarBusquedaSecuencial()`. Analice los resultados y dé una breve explicación de lo que sucede.

7. Lo ideal es que cuando un hilo termine, le notifique a los demás que ya no sigan buscando nada. Para esto usaremos un patrón que se llama **Observer**. Así:

   Creamos el interface `Observador`:

```
package model;

public interface Observador {
```

```
    void notificar();

}
```

Haga que la clase `Principal` implemente dicho interface y modifique el método que le pide implementar para que quede así:

```
@Override
public void notificar() {
    //Cuando se invoca este método es porque uno de los hilos encontró el valor buscado, entonces se detienen todos los hilos para que no sigan buscando
    for(BusquedaHilo h : hilos){
        h.setVivo(false);
    }
}
```

Para que esto funcione, debe sacar el array de `BusquedaHilo` del método `realizarBusquedaHilos()` para que se vuelva una variable de la clase.

Además, modifique el método para que quede así:

```
public void realizarBusquedaHilos(){

    hilos[0] = new BusquedaHilo(clientes, valorBuscado, 0, 500000);
    hilos[1] = new BusquedaHilo(clientes, valorBuscado, 500000, 1000000);
    hilos[2] = new BusquedaHilo(clientes, valorBuscado, 1000000, 1500000);
    hilos[3] = new BusquedaHilo(clientes, valorBuscado, 1500000, clientes.size());

    for(BusquedaHilo h : hilos){
        //Se le asigna el observador actual a cada uno de los hilos creados
        h.setObservador(this);
        h.start();
    }

}
```

8. En la clase `BusquedaHilo` agregue un atributo de tipo `Observador`:

```
@Getter @Setter
private Observador observador;
```

Modifique el `for` que tenemos en el método `run` para que quede así:

```
for (int i = posInicial; i < posFinal && vivo; i++){
    if(clientes.get(i).contains(cadenaBuscada)){
        encontrado = true;
        //Se notifica al observador que ya encontró el valor buscado
        observador.notificar();
        vivo = false;
    }
    iteraciones++;
}
```

9. Ejecute la clase `Aplicacion` y observe los resultados. Haga un análisis y documente lo que observa con este cambio.

10. Modifique el código anterior para que el método `realizarBusquedaHilos()` reciba por parámetro el número de hilos que se desean crear y haga que los rangos sean dinámicos, que los valores iniciales y finales de los hilos no estén quemados.

11. Investigue qué es un Manejador de eventos (**EventManager**) y cómo se puede aplicar en el patrón **Observer** que usamos en el código anterior. Para más información sobre este tema, les recomiendo visitar: https://refactoring.guru/es/design-patterns/observer

Problema 3

Escribir un programa que permita simular una carrera de caballos. Un hipódromo tiene 4 caballos, cada caballo debe tener su propio hilo. Todos los caballos comienzan en la posición 0 en x y deben moverse hasta el otro extremo de la pantalla. Dentro del método run de cada Caballo sólo debe actualizar su posición en x (los caballos corren de manera horizontal) asigne de manera aleatoria la velocidad para cada uno. Verifique el momento en el que llega un caballo a la meta y finalice la competición (todos los demás hilos deben detenerse).

Puede basarse en el siguiente código:

1. El código base del archivo FXML:

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.control.*?>
<?import javafx.scene.layout.*?>

<AnchorPane   prefHeight="400.0"   prefWidth="600.0"   xmlns="http://javafx.com/javafx/17.0.2-ea"
xmlns:fx="http://javafx.com/fxml/1" fx:controller="controller.JuegoController">

   <Button onAction="#ejecutarTarea" text="Iniciar movimientos" />
   <Pane fx:id="panel" layoutY="26.0" prefHeight="374.0" prefWidth="600.0">
       <Separator layoutY="170.0" prefHeight="2.0" prefWidth="600.0" />
   </Pane>

</AnchorPane>
```

2. El código base del controlador de dicho archivo FXML:

```
package controller;

import javafx.fxml.FXML;
import javafx.fxml.Initializable;
import javafx.scene.control.Alert;
import javafx.scene.layout.Pane;
import javafx.scene.paint.Color;
import model.CajaHilo;
import java.net.URL;
import java.util.ArrayList;
import java.util.ResourceBundle;

public class JuegoController implements Initializable{
```

```
    @FXML
    private Pane panel;
    private ArrayList<CajaHilo> cajas;

    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {
        this.cajas = new ArrayList<>();

        //Se crean dos Rectangles (CajaHilo) con los valores iniciales: índice, posición en X,
posición en Y, y color de fondo
        cajas.add( new CajaHilo(1, 0, 50, Color.RED));
        cajas.add( new CajaHilo(2, 0, 200, Color.BLUE));

        for(CajaHilo cajaHilo : cajas){
            //Se agregan los Rectangles al panel de la ventana
            panel.getChildren().add(cajaHilo);
        }

    }

    public void ejecutarTarea(){
        //Al dar click al botón se inician los hilos, y por lo tanto inicia el movimiento de las
cajas
        for(CajaHilo cajaHilo : cajas){
            Thread hilo = new Thread(cajaHilo);
            hilo.setDaemon(true);
            hilo.start();
        }
    }

}
```

Lea detenidamente el código y haga un análisis del mismo. Haga un breve informe de lo que se está haciendo.

3. El código base de la clase `CajaHilo`.

```
package model;

import javafx.application.Platform;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import lombok.Setter;
import java.util.Random;

public class CajaHilo extends Rectangle implements Runnable{

    @Setter
    private boolean animationRunning;
    private final int pos;

    public CajaHilo(int pos, double x, double y, Color color){
        //Se llama al constructor de Rectangle, tiene posición x, posición y, ancho y alto
        super(x, y, 100, 100);
```

```
        //Un simple número que diferencia al hilo de los demás hilos
        this.pos = pos;

        //Se le da un color de fondo al Rectangle
        this.setFill(color);

        //Esta variable controla la ejecución del hilo
        this.animationRunning = true;
    }

    @Override
    public void run() {

        while (animationRunning) {
            Platform.runLater(this::moverCaja);

            try {
                //Ajusta la velocidad del movimiento
                Thread.sleep(new Random().nextInt(10)+3);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }

    }

    private void moverCaja(){

        double movementDelta = 2; // Cantidad de movimiento en píxeles

        if (this.getX() + this.getWidth() > 600) { // Se detiene al llegar al borde derecho
            animationRunning = false;
            return;
        }

        this.setX(this.getX() + movementDelta); // Se actualiza su posición en el eje x
    }

}
```

Lea detenidamente el código y haga un análisis del mismo. Haga un breve informe de lo que se está haciendo.

4. Cree la clase `Aplicacion` y haga que se ejecute correctamente usando JavaFX. Ejecute el proyecto y observe qué pasa.

5. Modifique el código anterior para que no sean 2 cajas sino 4. Que cuando una caja llega al final del recorrido le notifique a las otras para que no se muevan más. Adicionalmente, haga que cada caja tenga una imagen de un caballo y cuando finalice el primero se muestre una alerta que diga qué posición ganó (el primero que llega a la meta).

6. Para crear un poco más de dinamismo al recorrido, haga que cuando un caballo vaya en la mitad del recorrido, se vuelva a calcular un valor aleatorio para cambiar la velocidad de movimiento que tiene.