Ejercicios de clase

Entorno de desarrollo Eclipse

Proyecto Hilos

Ejercicio 1

Juego de animación de hilos: Diseñar un programa en modo gráfico que con dos botones Pulsa y Salir. Cuando se pulse sobre el botón Pulsa aparecerá en pantalla una pelota que se irá moviendo por la pantalla un número determinado de veces. Al pulsar Salir termina la aplicación.

En este ejercicio se estudiará el concepto de programa monotarea.

Un programa monotarea ejecuta una tarea a la vez. O lo que es lo mismo un hilo o flujo de ejecución a la vez.

En el juego (cuando se pulse en Pulsa) la siguiente pelota saldrá cuando termine la animación de la primera pelota. Si quiero cerrar la aplicación (que es otro hilo de ejecución) no lo hará hasta que no termine la animación de la última pelota.

ejercicio01_un_hilo

Ejercicio 2

Seguimos con la animación anterior. Pero ahora interesa que: El programa sea multitarea, es decir ejecute varias tareas de forma simultánea. Va a ejecutar varios hilos de forma simultánea. Cada vez que sale una pelota es un hilo y al pulsar el botón Salir se inicia también otro hilo.

Thread

Interface Runnable.

Queremos que el juego sea multitarea pq puede interesar salir de la aplicación en cualquier momento o lanzar varias pelotas a la vez.

ejercicio02_varios_hilos_simultaneos

Ejercicio 9 banco sin sincronizar

Aplicación de transferencias entre cuentas. Siempre se debe transferir una cantidad entre dos cuentas y nadie extrae ni ingresa ningún capital nuevo. Cada transferencia será un hilo de ejecución. El saldo total debería mantenerse, pero no lo hace porque no se han sincronizado las transferencias. Esto ocurre porque los hilos no están sincronizados.

Para ello crearemos una clase Banco:

Banco.java

```
/*clase que crea las cuentas*/
public class Banco {
	private final Double[] cuentas;
	public void transferencia(int cuentaOrigen, int cuentaDestino, Double cantidad) {
		if (cuentas[cuentaOrigen]<cantidad) {
			return; // no hace nada se sale
		}
		System.out.println(Thread.currentThread());
		cuentas[cuentaOrigen]-=cantidad;
		System.out.printf("%10.2f de %d para %d", cantidad,cuentaOrigen,cuentaDestino);
		cuentas[cuentaDestino]+=cantidad;
		System.out.printf("Saldo total: %10.2f%n",this.getSaldoTotal());
	}
	public Double getSaldoTotal() {
		Double suma_cuentas=0.0;
		for(Double a:cuentas) {
			suma_cuentas+=a;
		}
		return suma_cuentas;
	}
	public Banco() {
		cuentas=new Double[100];
		for(int i=0;i<cuentas.length;i++) {
			cuentas[i]=2000.0;
		}
	}
	public Double[] getCuentas() {
		return cuentas;
	}
}
```

EjecucionTransferencias.java

```
public class EjecucionTransferencias implements Runnable {

    private Banco banco;

    private Integer cuentaOrigen;

    private Double cantidadMaxima;

        public EjecucionTransferencias(Banco banco, Integer cuentaOrigen, Double cantidadMaxima) {

                super();

                this.banco = banco;

                this.cuentaOrigen = cuentaOrigen;

                this.cantidadMaxima = cantidadMaxima;

    //método que ejecuta cada hilo

    //la cuenta destino va a ser aleatoria (entre 0 y 100)

    //La cantidad a mover será aleatoria entre la cantidadMaxima

    //el retardo se retarda en 0 y 10 milisegundos

        @Override

        public void run() {

        }


}
```

Ejercicio 10 banco sincronizado

Puede haber varios hilos de ejecución en paralelo, a la vez (porque estamos con programación concurrente). Puede haber varios hilos que ejecuten el método transferencia a la vez.

La clase ReentrantLock implementa la interface serializable y la interface Lock

Interface Lock tiene los métodos:

- look() bloquea un trozo de código del programa para que sólo pueda ser ejecutado por un único hilo simultáneamente.
- `Unlock() desbloquea el código.`

`Problema: Hay hilos que mueren y no realizan su trabajo.`

Ejercicio 11 bloqueo con condición

Solución: establecer condiciones a los bloqueos de código. Vamos a establecer una condición en un cierre. En la clase ReentrantLock, usamos el método newCondiction() que permite crear un bloqueo con una condición.

Este método devuelve un objeto de tipo interface Condition que tiene dos métodos importantes:

- await(): pone un hilo a la espera **y desbloquea el código para que pueda entrar otro hilo**.
- SignalAll(): informa a todos los hilos para que despierten los hilos que están a la espera.

Con la solución 11 todos los hilos van a realizar su trabajo. Cuando otros hilos realicen las transferencias que incrementen el saldo de las cuentas y los otros hilos saldrán de la espera y podrán hacer su trabajo.

Ejercicio 12 establecer una condición de bloqueo de código con synchronized y los metodos wait() notifyAll() de object.

Con esta forma solo se puede establecer un condición de bloqueo.