Trabajo Práctico n° 2

Introducción al trabajo con hilos: Thread / Runnable

1. En Java, ejecutar el siguiente código **repetidas** veces:

```
public class Cliente extends Thread {
	public void run(){
		System.out.println("soy"+Thread.currentThread().getName());
		Recurso.uso();
		try {
			Thread.sleep(2000);
		}catch (InterruptedException e) {
		};
	};
}
```

```
 public class Recurso {
	static void uso(){
		Thread t=Thread.currentThread();
		System.out.println("en Recurso:  Soy" + t.getName());
	}
}
public class testeoRecurso {
	public  static void main (String[] args){
		Cliente juan=new Cliente();
		juan.setName("Juan Lopez");
		Cliente ines=new Cliente ();
		ines.setName ("Ines Garcia");
		juan.start();
		ines.start();
	}
}
```

   a. Analice el funcionamiento del siguiente código. ¿Cuántos hilos de control participan en la ejecución?
   b. ¿Cúal es la funcionalidad del método “uso” de Recurso?
   c. Indique una salida posible.
   d. ¿Qué sucede si agrega la línea “Recurso.uso()” al final del main?
   e. ¿Qué sucede si cambia el orden de las instrucciones “start()”?

2. Ejecutar el siguiente código en **repetidas veces**:

```
public class MiEjecucion extends Thread{
     public void run(){
               ir();
     }
     public void ir(){
               hacerMas();
     }
     public void hacerMas(){
               System.out.println("En la pila");
     }
}

class TesteoHilos{
     public static void main (String[] args){
         Thread miHilo=  new MiEjecucion();
         miHilo.start();
         System.out.println("En el main");
     }
}
```

   a. ¿Cómo es el comportamiento de las diferentes ejecuciones?.
   b. ¿Se podría forzar las ejecuciones para que se comporte de una manera determinada? Realice las modificaciones pertinentes.

3. Ejecute el siguiente programa:

```
public class ThreadEjemplo extends Thread {
 public ThreadEjemplo(String str) {
  super(str);
 }
 public void run() {
  for (int i = 0; i < 10 ; i++)
   System.out.println(i + " " + getName());
  System.out.println("Termina thread " + getName());
 }
 public static void main (String [] args) {
  new ThreadEjemplo("Maria Jose").start();
  new ThreadEjemplo("Jose Maria").start();
  System.out.println("Termina thread main");
 }
}
```

   a. Si lo ejecuto varias veces, ¿qué sucede?

4. La interfaz **Runnable** proporciona un método alternativo a la utilización de la clase

Thread, implemente el ejercicio anterior utilizando la interfaz **Runnable.**

a. ¿Qué sucede? ¿Es necesario el constructor?

5. Analizar el siguiente código:

```
class MiHilo implements Runnable {
    String nombreHilo;

    unHilo(String nombre){
        nombreHilo=nombre;
    }
    //Punto de entrada del hilo
    //Los hilos comienzan a ejecutarse aquí
    public void run(){
        System.out.println("Comenzando "+nombreHilo);
        try {
            for (int contar=0; contar<10; contar++){
                Thread.sleep(400);
                System.out.println("En "+nombreHilo+", el recuento "+contar);
            }
        }catch (InterruptedException exc){
            System.out.println(nombreHilo + "Interrumpido.");
        }
        System.out.println("Terminando "+nombreHilo);
    }
}
```

```
class UsoHilos{
    public static void main(String[] args) {
        System.out.println("Hilo principal iniciando.");

        //Primero, construye un objeto unHilo.
        unHilo mh=new unHilo("#1");

        //Luego, construye un hilo de ese objeto.
        Thread nuevoHilo=new Thread(mh);

        //Finalmente, comienza la ejecución del hilo.
        nuevoHilo.start();

        for (int i=0; i<50;i++){
            System.out.print(" .");
        }try{
            Thread.sleep(100);
        }catch (InterruptedException exc){
            System.out.println("Hilo principal interrumpido.");
        }
        System.out.println("Hilo principal finalizado.");
```

```
}}
```

a. ¿Qué pasaría si quitamos el sleep()?¿Cuál sería la salida del programa?
b. El **main()** ¿siempre termina al final, o puede suceder que termine antes que el **run()**? ¿Por qué puede suceder esto?
c. Realizar los cambios necesarios para implementar extendiendo la clase Thread. NO usar la interfaz Runnable.
d. Realice el ejercicio para que en vez de ejecutar un solo hilo, ejecute 3. Realice el procedimiento varias veces. ¿Qué ocurre? ¿Se ejecutan en orden?

6. **Simulación de Carrera Multithread**:
Imagina que estás organizando una carrera de atletismo con varios corredores. Quieres simular esta carrera utilizando múltiples hilos en Java. Cada corredor será representado por un hilo independiente que avanzará en la pista de carreras.
Crea una clase llamada **Corredor** cuyas instancias serán utilizadas por los hilos (utilizar interfaz Runnable). Dentro de esta clase, define los atributos necesarios como el nombre del corredor y la distancia recorrida. Cada corredor sabe la distancia que recorrió, por lo que deberá imprimir su nombre y el avance (aleatorio entre 1 y 10), por cada paso que realiza. Entre cada paso realizado descansa. Una vez que haga 100 pasos, el corredor habrá terminado.
En la clase principal, crea un arreglo de " Corredores" y un hilo para cada corredor en el arreglo.
Inicia todos los hilos creados usando el método start(). Utiliza Thread.sleep() dentro del método run() de cada corredor para simular el tiempo entre pasos.
Al finalizar la carrera se desea saber qué corredor hizo la mayor distancia y cual fue esa distancia. ¿Quién será el encargado de mostrar este mensaje? ¿Cómo hará para esperar que todo los corredores terminen la carrera?

7. Supongamos que debemos simular el proceso de cobro de un supermercado; es decir, unos clientes van con un carro lleno de productos y una cajera les cobra los productos, pasando uno a uno por el escáner de la caja registradora. En este caso el cajero debe procesar la compra cliente a cliente, es decir que primero le cobra al cliente 1, luego al cliente 2 y así sucesivamente.
Para ello se debe definir una clase "Cajera" y una clase "Cliente" el cual tendrá un "array de enteros" que representarán los productos que ha comprado y el tiempo que El cajero tardará en pasar el producto por el escáner; es decir, que si tenemos un array con [1,3,5] significa que el cliente ha comprado 3 productos y que El cajero tardará en procesar el producto 1 '1 segundo', el producto 2 '3 segundos' y el producto 3 '5 segundos', con lo cual el tiempo total empleado por El cajero será de 9 segundos.

a. El siguiente código simula la operación de cobro con dos Clientes con un solo proceso (que es lo que se suele hacer normalmente), teniendo en

cuenta que se procesa primero la compra del Cliente 1 y después la del Cliente 2, con lo cual se tardará el tiempo del Cliente 1 + Cliente 2. Completar y ubicar en la clase que corresponda la implementación del método: *esperarXsegundos*.

```
public class Cajero {
	private String nombre;
	// Agregar Constructor, y métodos de acceso

	public void procesarCompra(Cliente cliente, long timeStamp) {
		System.out.println ("El cajero " + this.nombre +" COMIENZA A
		PROCESAR LA COMPRA DEL CLIENTE " + cliente.getNombre() + " EN EL
		TIEMPO: " + (System.currentTimeMillis() - timeStamp) / 1000 +
		"seg");

		for (int i = 0;  i < cliente.getCarroCompra().length;  i++) {
			this.esperarXsegundos(cliente.getCarroCompra()[i]);
			System.out.println("Procesado el producto " + (i + 1) +
			"->Tiempo: " + (System.currentTimeMillis() - timeStamp) /
			1000 + "seg");
		}
		System.out.println("El cajero " + this.nombre +" HA TERMINADO DE
		PROCESAR " + cliente.getNombre() + " EN EL TIEMPO: " +
		(System.currentTimeMillis() - timeStamp) / 1000 + "seg");
	}
}
```

```
public class Cliente {
	private String nombre;
	private int[] carroCompra;
	// Constructor y métodos de acceso
	}
```

```
public class Main {p
	ublic static void main(String[] args) {
		Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5,
		2, 3 });
		Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1,
		1 });
		Cajero cajero1 = new Cajera("Cajero 1");
		// Tiempo inicial de referencia
		long initialTime = System.currentTimeMillis();
		cajero1.procesarCompra(cliente1, initialTime);
		cajero1.procesarCompra(cliente2, initialTime);
	}
}
```

Salida:

```
El cajero Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL
TIEMPO:
0seg
Procesado el producto 1 ->Tiempo: 2seg
Procesado el producto 2 ->Tiempo: 4seg
Procesado el producto 3 ->Tiempo: 5seg
Procesado el producto 4 ->Tiempo: 10seg
Procesado el producto 5 ->Tiempo: 12seg
Procesado el producto 6 ->Tiempo: 15seg
El cajero Cajero 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
El cajero Cajero 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL
TIEMPO:
15seg
Procesado el producto 1 ->Tiempo: 16seg
Procesado el producto 2 ->Tiempo: 19seg
Procesado el producto 3 ->Tiempo: 24seg


Procesado el producto 4 ->Tiempo: 25seg
Procesado el producto 5 ->Tiempo: 26seg
El cajero Cajera 1 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 26seg
GENERACIÓN CORRECTA (total time: 26 seconds)
```

b. ¿Y si en vez de procesar primero un cliente y después otro, procesamos los dos a la vez?, ¿Cuánto tardaría el programa en ejecutarse?. Si en vez de haber solo una Cajera (es decir un solo hilo), hubiese dos Cajeras (es decir dos hilos o threads) podríamos procesar los dos clientes a la vez y tardar menos tiempo en ejecutar el programa. Complete el siguiente código a fin de representar el escenario descrito.

```
public class CajeroThread extends Thread {
	private String nombre;
	private Cliente cliente;
	private long initialTime;
	// Constructor, y métodos de acceso

	public void run() {
		System.out.println("El cajero " + this.nombre +
		" COMIENZA A PROCESAR LA COMPRA DEL CLIENTE "
		+ this.cliente.getNombre() + " EN EL TIEMPO: "
		+ (System.currentTimeMillis() - this.initialTime) / 1000 + "seg");

		for (int i = 0;  i < this.cliente.getCarroCompra().length;  i++) {
			this.esperarXsegundos(cliente.getCarroCompra()[i]);
			System.out.println("Procesado el producto " + (i + 1) + "
```

```
                    del cliente " + this.cliente.getNombre() + "->Tiempo: " +
                          (System.currentTimeMillis() - this.initialTime) / 1000
                    + "seg");
            }

            System.out.println("El cajero" + this.nombre + "HA TERMINADO DE
            PROCESAR"+ this.cliente.getNombre() + " EN EL TIEMPO: " +
                 (System.currentTimeMillis() - this.initialTime) / 1000 +
            "seg");
      }
}

public class MainThread {
      public static void main(String[] args) {
            Cliente cliente1 = new Cliente("Cliente 1", new int[] { 2, 2, 1, 5,
            2, 3 });
            Cliente cliente2 = new Cliente("Cliente 2", new int[] { 1, 3, 5, 1,
            1 });
            ........
      }
}
```

Posible salida: run:

```
El cajero Cajera 1 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 1 EN EL
TIEMPO: 0seg
El cajero Cajera 2 COMIENZA A PROCESAR LA COMPRA DEL CLIENTE Cliente 2 EN EL
TIEMPO: 0seg
Procesado el producto 1 del cliente Cliente 2->Tiempo: 1seg
Procesado el producto 1 del cliente Cliente 1->Tiempo: 2seg
Procesado el producto 2 del cliente Cliente 1->Tiempo: 4seg
Procesado el producto 2 del cliente Cliente 2->Tiempo: 4seg
Procesado el producto 3 del cliente Cliente 1->Tiempo: 5seg
Procesado el producto 3 del cliente Cliente 2->Tiempo: 9seg
Procesado el producto 4 del cliente Cliente 1->Tiempo: 10seg
Procesado el producto 4 del cliente Cliente 2->Tiempo: 10seg
Procesado el producto 5 del cliente Cliente 2->Tiempo: 11seg
El cajero Cajera 2 HA TERMINADO DE PROCESAR Cliente 2 EN EL TIEMPO: 11seg
Procesado el producto 5 del cliente Cliente 1->Tiempo: 12seg
Procesado el producto 6 del cliente Cliente 1->Tiempo: 15seg
El cajero Cajera 1 HA TERMINADO DE PROCESAR Cliente 1 EN EL TIEMPO: 15seg
GENERACIÓN CORRECTA (total time: 15 seconds)
```

8. Realice el ejercicio 7b utilizando la *interfaz Runnable*