Taller 1 – Threads

El propósito de este taller es entender la forma como se crean los threads para implementar aplicaciones concurrentes. En este taller implementaremos el concepto en los lenguajes Java y Python. Comencemos por dar un vistazo a la implementación en el lenguaje Java. En Java existen dos formas de realizar la implementación de los threads, bien sea mediante la extensión de la clase **`Thread`** o la implementación de la interface **`Runnable`**.

Ejemplo 1: Creación de threads como extensión de la clase `Thread`

Escriba el código de la clase **`EjThreads01`**.

```
public class EjThreads01 extends Thread {

	public void run() {
		System.out.println("Extendiendo la clase Thread.");
	}

	public static void main(String[] args) {
		EjThreads01 t = new EjThreads01();

		t.start();
	}
}
```

Ejemplo 2: Creación de threads como implementación de la interface `Runnable`

Escriba el código de la clase **`EjThreads02`**.

```
public class EjThreads02 implements Runnable {

	public void run() {
		System.out.println("Implementando la interfaz Runnable.");
	}

	public static void main(String[] args) {

		Thread t = new Thread(new EjThreads02());

		t.start();
	}
}
```

Carlos Eduardo Gómez Montoya
Jairo Emilio Bautista Mora
Geovanny Andrés González Rodríguez

Responda:

1. Complete la siguiente tabla, con respecto a la creación de threads usando la extensión de la clase **`Thread`** y la implementación de la interface **`Runnable`**.

Se parecen	Se diferencian
1. La extensión Thread y la implementación de la interfaz Runnable requiere el método run, en donde tiene acción el subproceso.	1. Después de que una clase extiende clase Thread, no puede extender más clases. Por el contrario, si se implementa la interfaz Runnable se puede extender otra clase (Java no admite herencia múltiple).
2. Tanto la extensión de la clase Thread como la implementación de la interfaz Runnable permite definir hilos de ejecución dentro de un mismo programa.	2. Cada hilo creado al extender la clase Thread crea un objeto único para él y se asocia con ese objeto. Por otro lado, cada subproceso creado al implementar una interfaz ejecutable comparte la misma instancia ejecutable.
3. Tanto la ejecución del método run dentro de una clase que extiende Thread como la clase que implementa la interfaz Runnable permite la concurrencia con otros métodos run de otros hilos de ejecución.	3. Como cada subproceso está asociado con un objeto único cuando se crea al extender la clase Thread, se requiere más memoria. Por otro lado, cada subproceso creado al implementar la interfaz Runnable requiere menos memoria.
4. La extensión Thread y la implementación de la interfaz Runnable empieza a ejecutar un hilo cuando se invoca el método start( ) perteneciente a un proceso.	4. La extensión de la clase Thread introduce un acoplamiento estricto, ya que el código y trabajo de thread están contenidos en la misma clase. Por otro lado, la interfaz de ejecución ejecutable introduce un acoplamiento suelto, ya que el código del subproceso está separado del trabajo asignado al subproceso.

Ejemplo 3: Creación de threads con un valor inicial – como extensión de la clase Thread

Escriba el código de la clase **`EjThreads01a`**.

```
public class EjThreads01a extends Thread {

	private int n;

	public EjThreads01a(int n) {
		System.out.println("Extendiendo la clase Thread.");
		this.n = n;
	}

	public void run() {
		System.out.println("El valor inicial es: " + n);
	}

	public static void main(String[] args) {
		EjThreads01a t = new EjThreads01a(5);

		t.start();
	}
}
```

Ejemplo 4: Creación de threads con un valor inicial – como implementación de la interface Runnable

Escriba el código de la clase **`EjThreads02a`**.

```
public class EjThreads02a implements Runnable {

	private int n;

	public EjThreads02a(int n) {
		System.out.println("Implementando la interfaz Runnable.");
		this.n = n;
	}

	public void run() {
		System.out.println("El valor inicial es: " + n);
	}

	public static void main(String[] args) {

		Thread t = new Thread(new EjThreads02a(5));

		t.start();
	}
}
```

Ejemplo 5: Creación de threads con nombre – como extensión de la clase `Thread`

Escriba el código de la clase **`EjThreads01b`**.

```
public class EjThreads01b extends Thread {

	private String name;

	public EjThreads01b(String name) {
		System.out.println("Extendiendo la clase Thread.");
		this.name = name;
	}

	public void run() {
		System.out.println("El nombre es: " + name);
	}

	public static void main(String[] args) {
		EjThreads01b t0 = new EjThreads01b("Thread"+0);
		EjThreads01b t1 = new EjThreads01b("Thread"+1);
		EjThreads01b t2 = new EjThreads01b("Thread"+2);

		t0.start();
		t1.start();
		t2.start();
	}
}
```

Ejemplo 6: Creación de threads con nombre – como implementación de la interface `Runnable`

Escriba el código de la clase **`EjThreads02b`**.

```
public class EjThreads02b implements Runnable {

	private String name;

	public EjThreads02b(String name) {
		System.out.println("Implementando la interfaz Runnable.");
		this.name = name;
	}

	public void run() {
		System.out.println("El nombre es: " + name);
	}

	public static void main(String[] args) {

		Thread t0 = new Thread(new EjThreads02b("Thread"+0));
		Thread t1 = new Thread(new EjThreads02b("Thread"+1));
		Thread t2 = new Thread(new EjThreads02b("Thread"+2));

		t0.start();
		t1.start();
		t2.start();
	}
}
```

Ejemplo 7: Creación de threads con nombre usando ciclos – como extensión de la clase Thread

Escriba el código de la clase **`EjThreads01c`**.

```
public class EjThreads01c extends Thread {

	private final static int MAX = 3;

	private String name;

	public EjThreads01c(String name) {
		System.out.println("Extendiendo la clase Thread.");
		this.name = name;
	}

	public void run() {
		System.out.println("El nombre es: " + name);
	}

	public static void main(String[] args) {
		EjThreads01c [] ta = new EjThreads01c[MAX];

		for (int i = 0; i < ta.length; i++) {
			ta[i] = new EjThreads01c("Thread"+i);
		}

		for (int i = 0; i < ta.length; i++) {
			ta[i].start();
		}
	}
}
```

Ejemplo 8: Creación de threads con nombre usando ciclos – como implementación de la interface `Runnable`

Escriba el código de la clase **`EjThreads02c`**.

```
public class EjThreads02c implements Runnable {

	private final static int MAX = 3;
	private String name;

	public EjThreads02c(String name) {
		System.out.println("Implementando la interfaz Runnable.");
		this.name = name;
	}


	public void run() {
		System.out.println("El nombre es: " + name);
	}

	public static void main(String[] args) {
		Thread [] ta = new Thread[MAX];

		for (int i = 0; i < ta.length; i++) {
			ta[i] = new Thread(new EjThreads02c("Thread"+i));
		}

		for (int i = 0; i < ta.length; i++) {
			ta[i].start();
		}
	}
}
```

Ejemplo 9: Creación de threads con nombre usando ciclos – como extensión de la clase `Thread`

Escriba el código de la clase **`EjThreads01d`**.

```
public class EjThreads01d extends Thread {

	private String name;

	public EjThreads01d(String name) {
		System.out.println("Extendiendo la clase Thread.");
		this.name = name;
	}

	public void run() {
		try {
			for (int i = 0; i < 5; i++) {
				System.out.println(name + " " + "valor: " + i);
				Thread.sleep(50);
			}
		}
		catch (Exception e) {}

		System.out.println("Saliendo: " + name);
	}

	public static void main(String[] args) {
		EjThreads01d t0 = new EjThreads01d("Thread"+0);
		EjThreads01d t1 = new EjThreads01d("Thread"+1);
		EjThreads01d t2 = new EjThreads01d("Thread"+2);

		t0.start();
		t1.start();
		t2.start();
	}
}
```

Ejemplo 10: Creación de threads con nombre usando ciclos – como implementación de la interface `Runnable`

Escriba el código de la clase **`EjThreads02d`**.

```
public class EjThreads02d implements Runnable {

	private String name;

	public EjThreads02d(String name) {
		System.out.println("Implementando la interfaz Runnable.");
		this.name = name;
	}

	public void run() {
		try {
			for (int i = 0; i < 5; i++) {
				System.out.println(name + " " + "valor: " + i);
				Thread.sleep(50);
			}
		}
		catch (Exception e) {}

		System.out.println("Saliendo: " + name);
	}

	public static void main(String[] args) {

		Thread t0 = new Thread(new EjThreads02d("Thread"+0));
		Thread t1 = new Thread(new EjThreads02d("Thread"+1));
		Thread t2 = new Thread(new EjThreads02d("Thread"+2));

		t0.start();
		t1.start();
		t2.start();
	}
}
```

A continuación, exploremos la implementación de los threads en Python. Existen dos maneras de crear threads en este lenguaje. La primera consiste en crear un objeto thread y enviar como referencia el proceso a invocar, lo anterior se expresa pasando un objeto que sea invocable (por ejemplo, una referencia a una función). La segunda opción consiste en escribir una clase que herede de la clase **Thread,** esta se encuentra disponible en el paquete **threading**. Luego se sobrescribe el método **run()** igual como lo hacemos en Java.

Ejemplo 11: Creación de threads en Python utilizando referencias a objetos invocables

```python
import logging
import threading
import time

# Logger Config
format = "%(asctime)s: %(message)s"
logging.basicConfig(format=format, level=logging.INFO, datefmt="%H:%M:%S")

# Funcion a ejecutar
def process(msg: str):
    logging.info(msg)
    time.sleep(3)


thread = threading.Thread(target=process, args=("Un saludo con un poco de espera",))
thread.start()
```

Como podemos observar la función se referencia en el argumento **target.** Los argumentos de la función se envian en una tupla mediante el parametro **args.**

Ejemplo 12: Creación de threads en Python utilizando clases

```python
import logging
import threading
import time
import random

# Logger Config
format = "%(asctime)s: %(message)s"
logging.basicConfig(format=format, level=logging.INFO, datefmt="%H:%M:%S")

# Clase
class Example_Thread(threading.Thread):
    # Constructor
    def __init__(self, id_thread: int, delay_time: float, msg: str):
        super().__init__()
        self.id_thread = id_thread
        self.delay_time = delay_time
        self.msg = msg

    # Método
    def message(self):
        time.sleep(self.delay_time)
        print(f"ID Thread: {self.id_thread} - Message: {self.msg}")

    # Run()
    def run(self):
        self.message()

threads = []
delay_time = random.uniform(2.5, 4.5)
print(f"Estimado usuario por favor espere {delay_time} segundos")
thread = Example_Thread(
    id_thread=0, delay_time=delay_time, msg="Un saludo con un poco de espera"
)
thread.start()
```

Ejercicio

Desarrolle un programa en Java que tenga dos threads. El primer thread imprime los números pares entre 1 y un límite superior recibido por entrada estándar. El segundo thread imprime los números impares entre el mismo rango. Después de imprimir un número en la consola, el thread debe dormir por una cantidad de milisegundos especificada desde la creación del thread. Haga dos versiones de este programa, una utilizando la extensión de la clase **`Thread`** y otra mediante la implementación de la interface **`Runnable`** o alguna de las dos implementaciones existentes en Python.