

Benemérita Universidad Autónoma de Puebla.

Facultad de Ciencias de la Computación.

Programación Paralela y Concurrente.

- 201938092, Javier Hernández Martínez.
- 201908877, Oliver Axel Juárez Oropeza.
- 201909400, Victor Hugo Ramírez López.
- 201923940, Martinez Flores Dan.

Docente: Carmen Cerón Garnica

Práctica 1

Introducción

La práctica consiste en la generación de hilos mediante el uso de Thread en Java mediante los conceptos de programación concurrente y paralela que permitan manejar interrupciones y mecanismos de comunicación entre operaciones (siendo esencial la organización de los procesos) generando de esta forma una serie de aplicaciones eficientes, además de robustas.

Objetivo

Comprender el funcionamiento y uso básico de hilos, desde la creación de hilos hasta el empleo de funciones nativas mediante el lenguaje de programación Java.

Desarrollo

1. Ejercicios de ejemplo

- a. **Ejercicio 1:** Creación de hilos por herencia donde tres hilos imprimen 10 números de manera concurrente

Escribimos el siguiente código en la clase HiloHerencia:

```
public class HiloHerencia extends Thread{
    // Nombre del hilo
    private final String nombre;

    //Constructor del hilo
    public HiloHerencia(String nombre) {
        this.nombre = nombre;
    }

    //Se define la accion que hara el hilo en el metodo run
    @Override
    public void run() {
        for(int i=0; i<10; i++) {
            System.out.println("Este es el hilo "+nombre+ " con el numero "+i);
        }
    }
}
```

Después en la clase Practica1 escribimos el siguiente código:

```
public static void main(String[] args) {
    System.out.println("Inicio de los hilos heredados");

    //se crean los hilos y se les da un nombre
    HiloHerencia a = new HiloHerencia("Uno");
    HiloHerencia b = new HiloHerencia("Dos");
    HiloHerencia c = new HiloHerencia("Tres");

    //se inician los hilos
    a.start();
    b.start();
    c.start();

    System.out.println("Fin de los hilos por herencia");
}
```

La ejecución del programa queda de la siguiente manera:

```
Inicio de los hilos heredados
Fin de los hilos por herencia
Este es el hilo Dos con el numero 0
Este es el hilo Uno con el numero 0
Este es el hilo Tres con el numero 0
Este es el hilo Dos con el numero 1
Este es el hilo Tres con el numero 1
Este es el hilo Uno con el numero 1
Este es el hilo Tres con el numero 2
Este es el hilo Dos con el numero 2
Este es el hilo Tres con el numero 3
Este es el hilo Uno con el numero 2
Este es el hilo Tres con el numero 4
Este es el hilo Dos con el numero 3
Este es el hilo Tres con el numero 5
Este es el hilo Uno con el numero 3
Este es el hilo Tres con el numero 6
Este es el hilo Dos con el numero 4
Este es el hilo Tres con el numero 7
Este es el hilo Uno con el numero 4
Este es el hilo Tres con el numero 8
Este es el hilo Uno con el numero 5
Este es el hilo Dos con el numero 5
Este es el hilo Uno con el numero 6
Este es el hilo Tres con el numero 9
Este es el hilo Uno con el numero 7
Este es el hilo Dos con el numero 6
Este es el hilo Uno con el numero 8
Este es el hilo Dos con el numero 7
Este es el hilo Uno con el numero 9
Este es el hilo Dos con el numero 8
Este es el hilo Dos con el numero 9
```

- b. **Ejercicio 2:** Creación de hilos por Interfaz runnable donde tres hilos imprimen 10 números de manera concurrente

Escribimos el siguiente código en la clase HiloInterfaz:

```
public class HiloInterfaz implements Runnable {
    private String nombre;

    public HiloInterfaz(String nombre) {
        this.nombre = nombre;
    }

    @Override
    public void run() {
        for(int i=0; i<10; i++) {
            System.out.println("Este es el hilo "+nombre+ " con el numero "+i);
        }
    }
}
```

Después en la clase Practica1 escribimos el siguiente código:

```
public static void main(String[] args) throws InterruptedException {
    System.out.println("Inicio de los hilos con interfaz");

    //se crean los hilos con la interfaz runneable
    HiloHerencia d = new HiloHerencia("Uno");
    HiloHerencia e = new HiloHerencia("Dos");
    HiloHerencia f = new HiloHerencia("Tres");

    //se crean hilos que utilizan objetos
    Thread hilod = new Thread(d);
    Thread hiloef = new Thread(e);
    Thread hilof = new Thread(f);

    //se inician los hilos
    hilod.start();
    hiloef.start();
    hilof.start();

    //se esperan hilos
    hilod.join();
    hiloef.join();
    hilof.join();

    System.out.println("Fin de los hilos por interfaz");
}
```

La ejecución del programa queda de la siguiente manera:

```
Inicio de los hilos con interfaz
Este es el hilo Uno con el numero 0
Este es el hilo Uno con el numero 1
Este es el hilo Uno con el numero 2
Este es el hilo Uno con el numero 3
Este es el hilo Uno con el numero 4
Este es el hilo Uno con el numero 5
Este es el hilo Uno con el numero 6
Este es el hilo Uno con el numero 7
Este es el hilo Uno con el numero 8
Este es el hilo Uno con el numero 9
Este es el hilo Dos con el numero 0
Este es el hilo Dos con el numero 1
Este es el hilo Tres con el numero 0
Este es el hilo Dos con el numero 2
Este es el hilo Dos con el numero 3
Este es el hilo Tres con el numero 1
Este es el hilo Dos con el numero 4
Este es el hilo Tres con el numero 2
Este es el hilo Dos con el numero 5
Este es el hilo Tres con el numero 3
Este es el hilo Dos con el numero 6
Este es el hilo Tres con el numero 4
Este es el hilo Dos con el numero 7
Este es el hilo Tres con el numero 5
Este es el hilo Dos con el numero 8
Este es el hilo Tres con el numero 6
Este es el hilo Dos con el numero 9
Este es el hilo Tres con el numero 7
Este es el hilo Dos con el numero 0
Este es el hilo Tres con el numero 8
Este es el hilo Dos con el numero 1
Este es el hilo Tres con el numero 9
Fin de los hilos por interfaz
```

2. Ejercicios planteados

- Realizar un programa que permita visualizar los estados de tres hilos, utiliza los métodos de la clase Thread.

Escribimos el siguiente código en la clase HiloHerencia:

```
public class HiloHerencia extends Thread{

    @Override
    public void run() {
        for(int i=0; i<=5; i++) {
            System.out.println(i + " " +getName());
        }
    }
}
```

Después en la clase Practica1 escribimos el siguiente código:

```
public class Practica1 {

    public static void main(String[] args) {

        // Estados en los hilos

        // Creacion de los hilos
        HiloHerencia a = new HiloHerencia();
        HiloHerencia b = new HiloHerencia();
        HiloHerencia c = new HiloHerencia();

        // Consultamos el estado del hilo
        // Como no hemos invocado el metodo start el estado será New
        System.out.println("Estado hilo a: "+a.getState());
        System.out.println("Estado hilo b: "+b.getState());
        System.out.println("Estado hilo c: "+c.getState()+"\n");

        // Consultamos si los hilos estan vivos o muertos
        a.start();
        System.out.println("Estado isAlive hilo a: "+a.isAlive());

        b.start();

        // Al terminar el proceso del hilo b, dormimos por 1 segundo
        try {
            b.sleep(1000);
            System.out.println("Estado isAlive hilo b: "+b.isAlive());
            System.out.println("Estado hilo b: "+b.getState());
        } catch (InterruptedException e) {
            System.out.println(e);
        }

        c.start();
        System.out.println("Estado isAlive hilo c: "+c.isAlive());

    }
}
```

La ejecución del programa queda de la siguiente manera:

```
Estado hilo a: NEW
Estado hilo b: NEW
Estado hilo c: NEW

Estado isAlive hilo a: true
0 Thread-0
1 Thread-0
2 Thread-0
0 Thread-1
1 Thread-1
2 Thread-1
3 Thread-0
3 Thread-1
4 Thread-1
5 Thread-1
4 Thread-0
5 Thread-0
Estado isAlive hilo b: false
Estado hilo b: TERMINATED
Estado isAlive hilo c: true
0 Thread-2
1 Thread-2
2 Thread-2
3 Thread-2
4 Thread-2
5 Thread-2
```

- b. Realiza un programa usando la interfaz Runnable y que permita el manejo de hilos y cambios del estado de un hilo

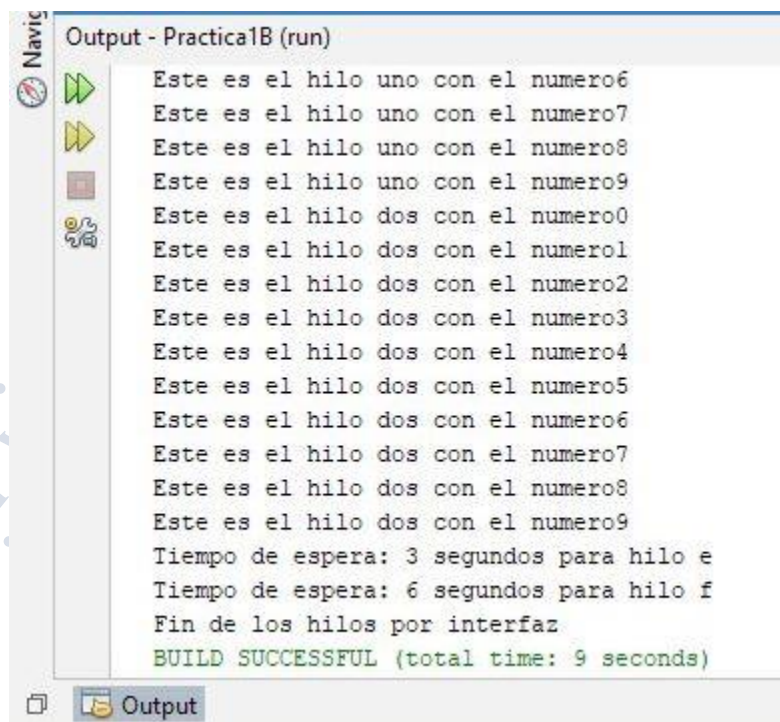
Escribimos el siguiente código

```
18 Hilol d = new Hilol( nombre: " uno");
19 Hilol e = new Hilol( nombre: " dos");
20 Hilol f = new Hilol( nombre: " tres");
21
22 //se crean hilos que utilizan los objetos
23
24 Thread hilod = new Thread( task: d);
25 Thread hiloee = new Thread( task: e);
26 Thread hilof = new Thread( task: f);
27
28 //se inician los hilos
29
30 hilod.start();
31 hiloee.start();
32 hilof.start();
33
34
35
36 //se esperan los hilos
37
38 hiloee.sleep( millis: 6000);
39 hiloee.join();
40 System.out.println( "Tiempo de espera: 3 segundos para hilo e");
41 hiloee.sleep( millis: 3000);
42 hiloee.join();
43 //hiloee.sleep(3000);
44 System.out.println( "Tiempo de espera: 6 segundos para hilo f");
45 hilod.join();
46 //hilof.sleep(6000);
47
```

Prac1B > main >

Output

La ejecución del programa queda de la siguiente manera



```
Output - Practica1B (run)
Este es el hilo uno con el numero6
Este es el hilo uno con el numero7
Este es el hilo uno con el numero8
Este es el hilo uno con el numero9
Este es el hilo dos con el numero0
Este es el hilo dos con el numero1
Este es el hilo dos con el numero2
Este es el hilo dos con el numero3
Este es el hilo dos con el numero4
Este es el hilo dos con el numero5
Este es el hilo dos con el numero6
Este es el hilo dos con el numero7
Este es el hilo dos con el numero8
Este es el hilo dos con el numero9
Tiempo de espera: 3 segundos para hilo e
Tiempo de espera: 6 segundos para hilo f
Fin de los hilos por interfaz
BUILD SUCCESSFUL (total time: 9 seconds)
```

- c. Realiza un programa de alternancia de dos hilos para imprimir los números pares e impares usando los cambios de estado de los hilos, el hilo A imprime impares y el hilo B los pares.

Escribimos el siguiente código en la clase HiloInterfazA:

```
public class HiloInterfazA implements Runnable{
    private String nombre;
    public HiloInterfazA(String nombre){
        this.nombre=nombre;
    }
    @Override
    public void run() {
        for (int i = 1; i < 10; i++) {
            //Se hara impresion unicamente de los numeros impares
            if(i % 2 !=0){
                System.out.println("Este es el hilo"+nombre+" con el numero "+i);
            }
            /*En caso de ser un numero par, se hara una espera para la sincronizacion
            de la impresion con el hilo B*/
            else{
                try {
                    Thread.sleep(100);
                } catch (InterruptedException ex) {
                    Logger.getLogger(HiloInterfazA.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}
```

Escribimos el siguiente código en la clase HiloInterfazB:

```
public class HiloInterfazB implements Runnable{
    private String nombre;
    public HiloInterfazB(String nombre){
        this.nombre=nombre;
    }
    @Override
    public void run() {
        for (int i = 2; i <11; i++) {
            //Se hara impresion unicamente de los numeros pares
            if(i % 2 ==0){
                System.out.println("Este es el hilo"+nombre+" con el numero "+i);
            }
            /*En caso de ser un numero impar, se hara una espera para la sincronizacion
            de la impresion con el hilo A*/
            else{
                try {
                    Thread.sleep(100);
                } catch (InterruptedException ex) {
                    Logger.getLogger(HiloInterfazB.class.getName()).log(Level.SEVERE, null, ex);
                }
            }
        }
    }
}
```

Escribimos el siguiente código en la clase Practica1_3

```
public class Practica1_3 {
    public static void main(String[] args) throws InterruptedException {
        System.out.println("Inicio de los hilos con interfaz\n");
        //se crean hilos con la interfaz runnable
        HiloInterfazA A= new HiloInterfazA(" A: Impares ");
        HiloInterfazB B= new HiloInterfazB(" B: Pares ");
        //se crean hilos que utilizan los objetos
        Thread hiloA= new Thread(A);
        Thread hiloB =new Thread(B);
        //se inicia el hilo A
        hiloA.start();
        //Sincronizando los hilos a partir de sleep (No ejecucion durante X tiempo) (Aplicado a su vez en los run)
        try {
            hiloA.sleep(100);
        }
        catch (InterruptedException ex) {
            System.out.println("Error en hilo A");
        }
        //Se inicia el Hilo B
        hiloB.start();
        //Sincronizando los hilos a partir de sleep (No ejecucion durante X tiempo) (Aplicado a su vez en los run)
        try {
            hiloB.sleep(100);
        }
        catch (InterruptedException ex) {
            System.out.println("Error en hilo B");
        }
    }
}
```


La ejecución del programa queda de la siguiente manera:

```
ant -f C:\Users\Javier\Desktop\Practical -Dnb.internal.action.name=run run
init:
Deleting: C:\Users\Javier\Desktop\Practical\build\build-jar.properties
deps-jar:
Updating property file: C:\Users\Javier\Desktop\Practical\build\build-jar.properties
compile:
run:
Inicio de los hilos con interfaz

Este es el hilo A: Impares con el numero 1
Este es el hilo B: Pares con el numero 2
Este es el hilo A: Impares con el numero 3
Este es el hilo B: Pares con el numero 4
Este es el hilo A: Impares con el numero 5
Este es el hilo B: Pares con el numero 6
Este es el hilo A: Impares con el numero 7
Este es el hilo B: Pares con el numero 8
Este es el hilo A: Impares con el numero 9
Este es el hilo B: Pares con el numero 10
BUILD SUCCESSFUL (total time: 2 seconds)
```

3. Preguntas

a. ¿Qué es un hilo en java?

Un hilo en Java es un mecanismo que permite la ejecución de diferentes tareas al mismo tiempo en un programa. Cada hilo tiene su propia memoria y ejecución independiente, aunque puede compartir recursos con otros hilos. La clase Thread es utilizada para crear, controlar y administrar hilos en Java. Los hilos pueden ser creados mediante herencia de la clase Thread o mediante implementación de la interfaz Runnable.

b. ¿Cuál es la diferencia entre crear los hilos con herencia y crear con la interfaz Runnable?

La diferencia principal entre crear hilos mediante herencia y mediante interfaz es que la primera limita la capacidad de heredar de otras clases mientras que la segunda permite crear hilos sin limitar la capacidad de heredar de otras clases.

c. ¿Cuáles son los estados de un hilo en Java y sus métodos para que el hilo pueda cambiar de estado?

Existen los estados New, Vivo y muerto.

El estado New resulta de cuando el hilo se ha creado y no se ha ejecutado, se obtiene del método getState().

El estado Vivo resulta de cuando invocamos el método start().

El estado muerto (Terminated) resulta de cuando el hilo finalizó su tarea.

El método isAlive() retorna un booleano, el cual indica si el hilo está vivo o no.

Existen otros métodos para manipular el funcionamiento del hilo, como lo es sleep() que detiene el proceso por cierta cantidad de tiempo; o el método wait() que espera a que se invoque el método notify() o notifyAll() para poder ejecutarse.

Conclusión

Con el desarrollo de esta primera práctica, pudimos observar el compartimiento de los hilos a la hora de ejecutarse, a la vez de que observamos las dos formas en las que pueden crearse e identificar las formas de trabajo que hay en cada una.

Bibliografía

- Ernesto. [La Geekipedia de Ernesto]. (12 de marzo de 2018). Curso Java Intermedio #24 | Estados de un hilo en Java. [Archivo de Video]. Youtube. <https://www.youtube.com/watch?v=0Ez-QPXsKTo&t=1087s>
- Material de apoyo de Manual de Prácticas de Laboratorio.

