MEDAC
Instituto Oficial de Formación Profesional

PROGRAMACIÓN DE SERVICIOS Y PROCESOS

TÉCNICO EN DESARROLLO DE APLICACIONES MULTIPLATAFORMA

Gestión de hilos

05

ÍNDICE

OBJETIVOS

Profundizar en el concepto de hilo.

Aprender cómo crear hilos.

Conocer cómo ejecutar y finalizar hilos.

Configurar la prioridad de los hilos.

/ 1. Introducción y contextualización práctica

En esta unidad, profundizaremos en el concepto de hilo.

Veremos de qué formas podemos crear hilos en Java, analizando las diferencias de cada una de ellas, así como de sus resultados.

También, aprenderemos cómo inicializar y ejecutar un hilo en Java, atendiendo a cualquiera de las formas en las que los podemos crear, que ya hemos estudiado previamente.

Seguidamente, aprenderemos cómo podemos finalizar un hilo dependiendo de distintas necesidades que se pudieran presentar, además de hacer que los hilos esperen que los demás finalicen.

Por último, vamos a configurar prioridades sobre los hilos.

Como puedes observar, muchos de los conceptos ya los hemos estudiado en temas anteriores, por lo que, en este, profundizaremos sobre ellos.

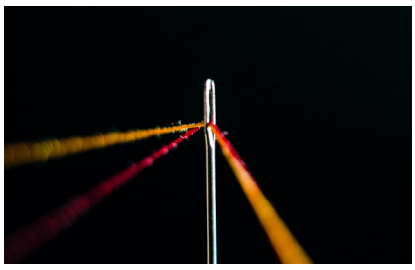

Fig. 1. Hilos.

Escucha el siguiente audio, en el que planteamos el caso práctico que iremos resolviendo a lo largo de esta unidad:

Audio Intro. "Los hilos y sus estados"

https://bit.ly/2WL8WxD

/ 2. Creación de hilos

2.1. Thread

En el lenguaje de programación Java, un hilo será creado mediante una instancia de la clase Thread, que nos permitirá gestionar completamente nuestros hilos.

Para crear una clase con la funcionalidad de hilo en Java, deberemos seguir los siguientes pasos:

- Crear una clase que herede de la clase Thread.
- Reescribir el método run, en el que deberemos poner todo el código que queramos que ejecute nuestro hilo.
- Crear un objeto de la nueva clase, que será el hilo con el que trabajaremos.

```
public class Hilo1 extends Thread
{

    @Override
    public void run()
    {

    }
}
```

Fig. 2. Clase que hereda de Thread.

Dentro de la clase, podremos crear todas las variables y métodos que necesitemos, incluidos constructores, métodos get, set y toString, ya que lo que estamos creando es una clase sin ningún tipo de peculiaridad.

Mediante los constructores, podremos crear los hilos con los datos que necesitemos, y con los métodos get y set, podremos obtener o modificar sus variables sin ningún tipo de problema.

Normalmente, dentro del método run, deberemos crear un bucle infinito, ya que la hebra estará ejecutándose de forma constante hasta que decidamos finalizarla o hasta que finalice el propio programa.

2.2. Runnable

Además de la forma anterior, el lenguaje de programación Java nos permitirá crear hilos implementando la interfaz Runnable, la cual nos ofrecerá, igualmente, gestionar completamente los hilos de nuestros programas.

Para crear una clase con la funcionalidad de hilo de esta forma en Java, deberemos seguir los siguientes pasos:

- Crear una clase que implemente la interfaz Runnable.
- Reescribir el método run, en el que deberemos poner todo el código que queramos que ejecute nuestro hilo.
- Crear un objeto de la nueva clase, que será el hilo con el que trabajaremos.

```java
public class Hilo2 implements Runnable
{

    @Override
    public void run()
    {

    }
}
```

Fig. 3. Clase que implementa la interfaz Runnable.

Al implementar una interfaz, tendremos que reescribir el método run de forma obligatoria, cosa que no ocurría al heredar de Thread. Si, en este caso, no reescribimos el método, no tendrá funcionalidad.

Al igual que ocurría con el método anterior, en esta clase podremos crear todos los atributos y métodos que vayamos a necesitar, incluyendo constructores, métodos get, set y toString.

También, en el método run, deberemos crear un bucle infinito para que la hebra se quede en segundo plano, ejecutándose constantemente hasta que decidamos finalizarla o hasta que finalice el propio programa.

```java
public class Hilo2 implements Runnable
{

    @Override
    public void run()
    {
        while(true)
        {
            /*
                Código a ejecutar
                por el hilo.
            */
        }
    }
```

Fig. 4. Método run con bucle infinito II.

/ 3. Caso práctico 1: "¿Thread o Runnable?"

Planteamiento: Pilar y José están empezando a crear sus primeros programas con hilos y, mientras que Pilar está utilizando la implementación de la interfaz Runnable, a José le gusta más utilizar la clase Thread.

Fig. 5. Muchos hilos.

«Debe haber alguna diferencia significativa para utilizar una u otra forma de crear los hilos», se pregunta Pilar. José, en cambio, no tiene ninguna duda y prefiere utilizar la clase Thread: «a mí me gusta más así», dice, convencido.

Nudo: ¿Crees que es indiferente crear los hilos de cualquiera de estas dos formas? ¿Tiene razón José utilizando la clase Thread sin pensar en la utilidad de la interfaz Runnable?

Desenlace: El lenguaje de programación Java tiene la peculiaridad de ofrecer dos mecanismos para crear hilos, cosa que no ocurre en otros lenguajes de programación como, por ejemplo, el C++, que solo ofrece una forma de crearlos.

Pilar piensa que debe haber alguna explicación para que existan estos dos mecanismos de creación de hilos y la verdad es que sí existe una muy buena razón.

En la asignatura de Programación vimos que Java no nos permitía utilizar la herencia múltiple, ya que solo puede heredar de una única clase. Pues bien, ¿qué ocurre entonces si queremos que una clase que ya hereda de otra sea una hebra? La respuesta es sencilla: no podríamos hacerlo. Por este motivo, tenemos la posibilidad de implementar la interfaz Runnable para que cualquier clase que necesitemos tenga la funcionalidad de hebra, ya que, en Java, podremos implementar todas las interfaces que necesitemos.

/ 4. Inicialización y ejecución de hilos

Cuando creamos un objeto de una clase que tiene la funcionalidad de hilo mediante la herencia de Thread, no bastará con crear el objeto con new().

Para que este hilo sea ejecutado y pase a segundo plano, debemos hacer que pase al estado Ejecutándose y, para ello, deberemos iniciarlo mediante el método start(). Hay que tener mucho cuidado de no utilizar el método run(), ya que no hará que se ejecute el hilo de forma paralela. A fin y al cabo, el método start() hará que se ejecute el método run() de forma paralela.

El método start() va a realizar las siguientes tareas:

- Reservar todos los recursos necesarios para la correcta ejecución del hilo.
- Llamar al método run() y hacer que se ejecute de forma paralela.

```
public static void main(String[] args) {
    // Creamos el objeto
    Hilo1 hilo = new Hilo1();
    // Lanzamos el hilo
    hilo.start();
}
```

Fig. 6. Creación y ejecución de hilo.

Si, por el contrario, hemos creado un hilo implementando la interfaz Runnable, debemos seguir los siguientes pasos:

- Crear un objeto de la clase hilo.
- Crear un objeto de la clase Thread mediante el objeto anterior.
- Llamar al método run() y hacer que se ejecute de forma paralela.

Hay que tener en cuenta que, una vez hemos llamado al método start() de un hilo, no podremos hacerlo de nuevo, ya que el hilo se está ejecutando. En caso de hacerlo, tendremos una excepción del tipo IllegalThreadStateException.

Vídeo 1. "Creación y ejecución de hilos – Thread"
https://bit.ly/3eRzgw6

/ 5. Suspensión y finalización de hilos

Cuando estamos trabajando con hilos, como hemos comentado anteriormente, necesitaremos de un bucle infinito para que la hebra se ejecute constantemente en segundo plano.

Puede ocurrir que necesitemos que nuestros hilos ejecuten una acción cada cierto tiempo, lo cual es imposible, a no ser que consigamos detener el hilo de alguna forma.

El lenguaje de programación Java nos proporciona los siguientes métodos para suspender hilos de forma segura:

- **Método sleep():** Mediante este método, conseguiremos que nuestro hilo «se duerma» una cierta cantidad de milisegundos, que tendremos que pasarle como parámetro. En realidad, el hilo pasará al estado bloqueado durante esa cantidad tiempo, volviendo a ejecutarse después. En caso de que necesitemos reactivar el hilo antes de que se cumpla el tiempo de bloqueo, podemos hacerlo mediante el método interrupt().

- **Método wait():** Mediante este método, conseguiremos que el hilo se quede bloqueado hasta nuevo aviso. Para desbloquear el hilo, debemos utilizar los métodos notify() o notifyAll(). La diferencia entre estos dos métodos es que notifiyAll() notifica a todos y cada uno de los hilos que estén bloqueados. Al método notify() también podremos pasarle como parámetro una cantidad de milisegundos para que el hilo esté bloqueado, desbloqueándose cuando se cumpla esa cantidad de tiempo o cuando le llegue una señal notify, lo que antes ocurra.

Otra operación que podemos realizar es la de finalizar un hilo (que en condiciones normales finalizaría al terminar su tarea) directamente por nosotros mismos.

Para esto, podemos utilizar los siguientes métodos:

- **Método isAlive():** Este método nos indicará si un hilo está vivo o no, es decir, nos devolverá 'verdadero' en caso de que el hilo esté en un estado diferente al de finalizado y 'falso' si está finalizado.

- **Método stop():** Este método permite finalizar un hilo, pero es extremadamente peligroso utilizarlo, ya que puede dar situaciones de interbloqueo de hilos y hacer que nuestra aplicación se bloquee sin posibilidad de arreglo.

Vídeo 2. "Creación y ejecución de hilos – Runnable"
https://bit.ly/2OMlNLt

/ 6. Planificación de hilos

Cuando tenemos varios hilos ejecutándose en segundo plano, no vamos a poder controlar cuál de ellos se ejecutará, ya que será el propio planificador del sistema operativo que estemos usando quien decida qué hilo se ejecutará en cada momento.

Por este motivo, cuando ejecutamos varios hilos a la vez en un programa, puede que tengamos diferentes salidas de datos por pantalla en las diferentes ejecuciones del programa.

Puede ocurrir que necesitemos esperar a que un hilo o, incluso, un grupo de ellos finalice para poder seguir adelante con las demás tareas, pero, como tenemos «el problema» de que los hilos se **ejecutan concurrentemente** y es el planificador del sistema operativo el que los va eligiendo, puede ser que tengamos salidas que no deseemos y que estén relacionadas con el orden de ejecución de los hilos.

En el caso de que queramos hacer que una acción se realice después de finalizar un hilo, podemos hacer que se espere a que el hilo acabe utilizando el método join(). A este método podremos incluirle como parámetro la cantidad de tiempo en milisegundos que queremos esperar para que acabe el hilo o, si queremos esperar a que acabe el hilo de forma «natural», sin ninguna restricción de tiempo, no le pasaremos ningún parámetro.

Debemos tener mucho cuidado con el método join(), ya que, si se produce un error de bloqueo con los hilos anteriores, no habrá forma de que ese código se ejecute.

```
public static void main(String[] args)
{
    try
    {
        // Creación del hilo, estado: nuevo o new
        Thread t1 = new Thread(new Hilo2("Hilo1", 1000));
        Thread t2 = new Thread(new Hilo2("Hilo2", 1000));
        Thread t3 = new Thread(new Hilo2("Hilo3", 1000));
        t1.start(); // Iniciamos el hilo, estado: listo o runnable
        t1.join();  // Hacemos que se espere
        t2.start(); // start llama a run(), y el hilo pasa a en ejecución
        t2.join();
        t3.start();
        t3.join();
    }
    catch (InterruptedException ex)
    {
        System.out.println("Error -> " + ex.toString());
    }
}
```

Fig. 7. Ejecución de hilos en orden.

/ 7. Caso práctico 2: "¿Qué usos pueden tener los hilos?"

Planteamiento: Una vez Pilar y José comienzan a tener claro cómo se crean y ejecutan los hilos en Java, empiezan a tener ganas de poner en práctica sus conocimientos creando programas multihilo.

«Al principio ha sido complicado, pero ya le estoy pillando el truco a los hilos», le dice bastante contento José a Pilar. Sin embargo, Pilar no lo tiene tan claro: «¿para qué nos puede servir esto a nosotros en nuestras aplicaciones?», se pregunta.

Fig. 8. La hora del sistema.

Nudo: ¿Qué piensas al respecto? ¿Qué usos crees que pueden tener los hilos en una aplicación «normal y corriente»?

Desenlace: Es muy común que, cuando se empiezan a estudiar los hilos, no se tenga claro qué utilidades le podemos dar en una aplicación simple como las que hemos creado hasta ahora.

En cierto modo, es muy comprensible que dudemos de ello, pero, en cuanto comprendemos la funcionalidad de los hilos, veremos muy claro un uso que tienen, que siempre hemos tenido muy cerca y que no hemos tenido en cuenta.

El uso más común que le vamos a dar a los hilos está relacionado con el factor «tiempo». Concretamente, consiste en mostrar el propio tiempo en una aplicación, ya sea un cronómetro hacia adelante o hacia atrás. O, también, mostrar la fecha y hora, por ejemplo.

Centrándonos en el caso de que una aplicación pueda mostrar la hora actual del sistema, esta se tendrá que ir actualizando segundo a segundo o minuto a minuto, según necesitemos. Esto no es más que un hilo ejecutándose de forma paralela, que consulta la hora del sistema cada intervalo de tiempo configurado para actualizarla en la interfaz de la aplicación.

/ 8. Prioridad en los hilos

En el lenguaje de programación Java, los hilos van a tener una prioridad que irá de 1 al 10, siendo 1 la menor prioridad y 10 la máxima.

Si no indicamos nada, tendremos, por defecto, una prioridad media, o de 5, en nuestros hilos.

La clase Thread tiene definidas tres constantes que van a representar los niveles de prioridad relativos a los hilos:

- MIN_PRIORITY tiene un valor de 1.
- MAX_PRIORITY tiene un valor de 10.
- NORM_PRORITY tiene un valor de 5, que es el valor por defecto.

Para la gestión de las prioridades de los hilos, podremos utilizar los siguientes métodos de la clase Thread:

- Método getPriority(): Mediante este método, podremos obtener la prioridad que tiene un hilo.
- Método setPriority(): Con este, podremos cambiar en cualquier momento la prioridad de un hilo de nuestra aplicación. Esta nueva prioridad, deberemos pasársela al método como parámetro, que puede valer:
  - Thread.NORM_PRORITY
  - Thread.MIN_PRIORITY
  - Thread.MAX_PRIORITY

Cuando todos los hilos tengan la misma prioridad, el comportamiento será exactamente el mismo al que estamos acostumbrados. En este caso, si queremos que el planificador del sistema operativo equilibre la ejecución de los hilos, podremos usar el método yield() de la clase Thread.

```
public static void main(String[] args)
{
    Thread t1 = new Thread(new Hilo2("Hilo1", 1000));
    Thread t2 = new Thread(new Hilo2("Hilo2", 1000));
    Thread t3 = new Thread(new Hilo2("Hilo3", 1000));

    t1.setPriority(Thread.MIN_PRIORITY);
    t2.setPriority(Thread.MAX_PRIORITY);
    t3.setPriority(Thread.NORM_PRIORITY);

    t1.start();
    t2.start();
    t3.start();
}
```

Fig. 9. Cambiando la prioridad de los hilos.

Audio 1. "Peligros de cambiar la prioridad"
https://bit.ly/30z4Tpe

/ 9. Resumen y resolución del caso práctico de la unidad

A lo largo de esta unidad, hemos profundizado un poco más en el concepto de hilo.

Hemos visto qué mecanismos nos ofrece el lenguaje de programación Java para crear hilos y hemos visto las diferencias entre cada uno de ellos.

También hemos aprendido a inicializar y ejecutar un hilo en cualquiera de las formas de creación estudiadas.

Otro de los aspectos importantes que hemos expuesto es la posibilidad de finalización de un hilo de forma correcta, así como hacer que un hilo espere de forma adecuada la ejecución de los demás para iniciar su ejecución.

Por último, hemos visto qué opciones nos ofrece Java para dar una prioridad de ejecución a los hilos, haciendo que se ejecuten antes o después que los demás.

Resolución del Caso práctico de la unidad

Uno de los mayores problemas que tendremos cuando empecemos a programar aplicaciones multihilo es no tener claro el esquema de estados de los mismos.

Esto se debe a que, si conocemos los estados por los que pasa un proceso, podremos intuir de una forma mucho más sencilla si va a haber un fallo o, en el caso de que se produjese, saber el porqué.

Imaginemos que estamos realizando una aplicación que usa hilos y que, en cierto momento, necesita que el usuario le introduzca un número. Si no tenemos en cuenta que durante ese período en el que está esperando esa información se encontrará en estado 'Bloqueado', no podremos realizar de forma correcta la entrada del siguiente hilo, por lo que podríamos provocar una inanición.

/ 10. Bibliografía

Gómez, O. (2016). Programación de Servicios y Procesos Versión 2.1. Recuperado de: https://github.com/OscarMaestre/ServiciosProcesos/blob/master/_build/latex/Servicios Procesos.pdf

Java multithreading - thread priority. (2 de diciembre de 2013). [Entrada de Blog]. Stack Overflow. Recuperado de: https://stackoverflow.com/questions/20333150/java-multithreading-thread-priority

Sánchez, J. M. y Campos, A. S. (2014). Programación de servicios y procesos. Madrid: Alianza Editorial.