1.- Introducción.

Seguro que en más de una ocasión mientras te descargabas una imagen desde tu navegador web, seguías navegando por Internet e incluso iniciabas la descarga de un nuevo archivo, y todo esto ejecutándose el navegador como un único proceso, es decir, teniendo un único ejemplar del programa en ejecución.

Pues bien, ¿Cómo es capaz de hacer el navegador web varias tareas a la vez? Seguro que estarás pensando en la programación concurrente, y así es; pero un nuevo enfoque de la concurrencia, denominado "programación multihilo". Justo lo que vamos a estudiar en esta unidad.

Los programas realizan actividades o tareas, y para ello pueden seguir uno o más flujos de ejecución. Dependiendo del número de flujos de ejecución, podemos hablar de dos tipos de programas:

Programa de flujo único. Es aquel que realiza las actividades o tareas que lleva a cabo una a continuación de la otra, de manera secuencial, lo que significa que cada una de ellas debe concluir por completo, antes de que pueda iniciarse la siguiente.

Programa de flujo múltiple. Es aquel que coloca las actividades a realizar en diferentes flujos de ejecución, de manera que cada uno de ellos se inicia y termina por separado, pudiéndose ejecutar éstos de manera simultánea o concurrente.

La programación multihilo o multithrearing consiste en desarrollar programas o aplicaciones de flujo múltiple. Cada uno de esos flujos de ejecución es un thread o hilo.

En el ejemplo anterior sobre el navegador web, un hilo se encargaría de la descarga de la imagen, otro de continuar navegando y otro de iniciar una nueva descarga. La utilidad de la programación multihilo resulta evidente en este tipo de aplicaciones. El navegador puede realizar "a la vez" estas tareas, por lo que no habrá que esperar a que finalice una descarga para comenzar otra o seguir navegando.

Cuando decimos "a la vez" recuerda que nos referimos a que las tareas se realizan concurrentemente, pues el que las tareas se ejecuten realmente en paralelo dependerá del Sistema Operativo y del número de procesadores del sistema donde se ejecute la aplicación. En realidad, esto es transparente para el programador y usuario, lo importante es la sensación real de que el programa realiza de forma simultánea diferentes tareas.

2.- Conceptos sobre hilos.

Pero ¿qué es realmente un hilo? Un hilo, denominado también subproceso, es un flujo de control secuencial independiente dentro de un proceso y está asociado con una secuencia de instrucciones, un conjunto de registros y una pila.

Cuando se ejecuta un programa, el Sistema Operativo crea un proceso y también crea su primer hilo, hilo primario, el cual puede a su vez crear hilos adicionales.

Desde este punto de vista, un proceso no se ejecuta, sino que solo es el espacio de direcciones donde reside el código que es ejecutado mediante uno o más hilos.

Por lo tanto podemos hacer las siguientes observaciones:

Un hilo no puede existir independientemente de un proceso.

Un hilo no puede ejecutarse por si solo.

Dentro de cada proceso puede haber varios hilos ejecutándose.

Un único hilo es similar a un programa secuencial; por si mismo no nos ofrece nada nuevo.

Es la habilidad de ejecutar varios hilos dentro de un proceso lo que ofrece algo nuevo y útil; ya que cada uno de estos hilos puede ejecutar actividades diferentes al mismo tiempo. Así en un programa un hilo puede encargarse de la comunicación con el usuario, mientras que
otro hilo transmite un fichero, otro puede acceder a recursos del sistema (cargar sonidos, leer ficheros,), etc.

2.1.- Recursos compartidos por los hilos.

Un hilo lleva asociados los siguientes elementos:

Un identificador único.

Un contador de programa propio.

Un conjunto de registros.

Una pila (variables locales).

Por otra parte, un hilo puede compartir con otros hilos del mismo proceso los siguientes recursos:

Código.

Datos (como variables globales).

Otros recursos del sistema operativo, como los ficheros abiertos y las señales.

Seguro que te estarás preguntando "si los hilos de un proceso comparten el mismo espacio de memoria, ¿qué pasa si uno de ellos la corrompe?" La respuesta es, que los otros hilos también sufrirán las consecuencias. Recuerda que en el caso de procesos, el sistema operativo normalmente protege a un proceso de otro y si un proceso corrompe su espacio de memoria los demás no se verán afectados

El hecho de que los hilos compartan recursos (por ejemplo, pudiendo acceder a las mismas variables) implica que sea necesario utilizar esquemas de bloqueo y sincronización, lo que puede hacer más difícil el desarrollo de los programas y así como su depuración.

Realmente, es en la sincronización de hilos, donde reside el arte de programar con hilos; ya que de no hacerlo bien, podemos crear una

aplicación totalmente ineficiente o inútil, como por ejemplo, programas que tardan horas en procesar servicios, o que se bloquean con facilidad y que intercambian datos de manera equivocada.

Profundizaremos más adelante en la sincronización, comunicación y compartición de recursos entre hilos dentro del contexto de Java.

2.2.- Ventajas y uso de hilos.

Como consecuencia de compartir el espacio de memoria, los hilos aportan las siguientes ventajas sobre los procesos:

Se consumen menos recursos en el lanzamiento y la ejecución de un hilo que en el lanzamiento y ejecución de un proceso.

Se tarda menos tiempo en crear y terminar un hilo que un proceso.

La conmutación entre hilos del mismo proceso o cambio de contexto es bastante más rápida que entre procesos.

Es por esas razones, por lo que a los hilos se les denomina también procesos ligeros.

Y ¿cuándo se aconseja utilizar hilos? Se aconseja utilizar hilos en una aplicación cuando:

La aplicación maneja entradas de varios dispositivos de comunicación.

La aplicación debe poder realizar diferentes tareas a la vez.

Interesa diferenciar tareas con una prioridad variada. Por ejemplo, una prioridad alta para manejar tareas de tiempo crítico y una prioridad baja para otras tareas.

La aplicación se va a ejecutar en un entorno multiprocesador.

Por ejemplo, imagina la siguiente situación:

Debes crear una aplicación que se ejecutará en un servidor para atender peticiones de clientes. Esta aplicación podría ser un servidor de bases de datos, o un servidor web.

Cuando se ejecuta el programa éste abre su puerto y queda a la escucha, esperando recibir peticiones.

Si cuando recibe una petición de un cliente se pone a procesarla para obtener una respuesta y devolverla, cualquier petición que reciba mientras tanto no podrá atenderla, puesto que está ocupado.

La solución será construir la aplicación con múltiples hilos de ejecución.

En este caso, al ejecutar la aplicación se pone en marcha el hilo principal, que queda a la escucha.

Cuando el hilo principal recibe una petición, creará un nuevo hilo que se encarga de procesarla y generar la consulta, mientras tanto el hilo principal sigue a la escucha recibiendo peticiones y creando hilos.

De esta manera un gestor de bases de datos puede atender consultas de varios clientes, o un servidor web puede atender a miles de clientes.

Si el número de peticiones simultáneas es elevado, la creación de un hilo para cada una de ellas puede comprometer los recursos del sistema. En

este caso, como veremos al final de la unidad lo resolveremos mejor con un pool de hilos.

Resumiendo, los hilos son idóneos para programar aplicaciones de entornos interactivos y en red, así como simuladores y animaciones.
Los hilos son más frecuentes de lo que parece. De hecho, todos los programas con interfaz gráfico son multihilo porque los eventos y las rutinas de dibujado de las ventanas corren en un hilo distinto al principal. Por ejemplo en Java, AWT o la biblioteca gráfica Swing usan hilos.

3.- Multihilo en Java. Librerías y clases.

3.1.- Utilidades de concurrencia del paquete java.util.concurrent.

El paquete java.util.concurrent incluye una serie de clases que facilitan enormemente el desarrollo de aplicaciones multihilo y aplicaciones complejas, ya que están concebidas para utilizarse como bloques de diseño. Concretamente estas utilidades están dentro de los siguientes paquetes: java.util.concurrent. En este paquete están definidos los siguientes elementos:

Clases de sincronización. Semaphore, CountDownLatch, CyclicBarrier y Exchanger.

Interfaces para separar la lógica de la ejecución, como por ejemplo Executor, ExecutorService, Callable y Future.

Interfaces para gestionar colas de hilos. BlockingQueque, LinkedBlokingQueque, nArrayBlockingQueque, SynchronousQueque, PriorityBlockingQueque y DelayQueque.

java.util.concurrent.atomic. Incluye un conjunto de clases para ser usadas como variables atómicas en aplicaciones multihilo y con diferentes tipos de dato, por ejemplo AtomicInteger y AtomicLong.java.util.concurrent.locks. Define una serie de clases como uso alternativo a la cláusula sinchronized. En este paquete se encuentran algunas interfaces como por ejemplo Lock, ReadWriteLock.

A lo largo de esta unidad estudiaremos las clases e interfaces más importantes de este paquete.

4.- Creación de hilos.

En Java, un hilo se representa mediante una instancia de la clase java.lang.thread. Este objeto thread se emplea para iniciar, detener o cancelar la ejecución del hilo de ejecución. Los hilos o threads se pueden implementar o definir de dos formas:

Extendiendo la clase thread.

Mediante la interfaz Runnable .

En ambos casos, se debe proporcionar una definición del método run(), ya que este método es el que contiene el código que ejecutará el hilo, es decir, su comportamiento.

El procedimiento de construcción de un hilo es independiente de su uso, pues una vez creado se emplea de la misma forma. Entonces, ¿cuando utilizar uno u otro procedimiento?

Extender la clase thread es el procedimiento más sencillo, pero no siempre es posible. Si la clase ya hereda de alguna otra clase padre, no será posible heredar también de la clase thread (recuerda que Java no permite la herencia múltiple), por lo que habrá que recurrir al otro procedimiento.

Implementar Runnable siempre es posible, es el procedimiento más general y también el más flexible.

Por ejemplo, piensa en la programación de applets, cualquiera de ellos tiene que heredar de la clase java.applet.Applet; y en consecuencia ya no puede heredar de thread si se quiere utilizar hilos. En este caso, no queda más remedio que crear los hilos implementando Runnable.

Cuando la Máquina Virtual Java (JVM) arranca la ejecución de un programa, ya hay un hilo ejecutándose, denominado hilo principal del programa, controlado por el método main(), que se ejecuta cuando comienza el programa y es el último hilo que termina su ejecución, ya que cuando este hilo finaliza, el programa termina.

Siempre hay un hilo que ejecuta el método main(), y por defecto, este hilo se llama "main".

Para saber qué hilo se está ejecutando en un momento dado, el hilo en curso, utilizamos el método currentThread() y que obtenemos su nombre invocando al método getName(), ambos de la clase thread.

4.1.- Creación de hilos extendiendo la clase Thread.

Para definir y crear un hilo extendiendo la clase thread, haremos lo siguiente:

Crear una nueva clase que herede de la clase thread.

Redefinir en la nueva clase el método run() con el código asociado al hilo. Las sentencias que ejecutará el hilo.

Crear un objeto de la nueva clase thread. Éste será realmente el hilo.

Una vez creado el hilo, para ponerlo en marcha o iniciarlo:

Invocar al método start() del objeto thread (el hilo que hemos creado).

4.2.- Creación de hilos mediante la interfaz Runnable.

Para definir y crear hilos implementando la interfaz Runnable seguiremos los siguientes pasos:

Declarar una nueva clase que implemente a Runnable.

Redefinir (o sombrear) en la nueva clase el método run() con el código asociado al hilo. Lo que queremos que haga el hilo.

Crear un objeto de la nueva clase.

Crear un objeto de la clase thread pasando como argumento al constructor, el objeto cuya clase tiene el método run(). Este será realmente el hilo.

Una vez creado el hilo, para ponerlo en marcha o iniciarlo:
Invocar al método start() del objeto thread (el hilo que hemos creado).