/\* ***Hilos en Java:***

La programación multihilo ejecuta diferentes hilos de ejecución de forma concurrente. Esta da más agilidad a la ejecución, ya que cada hilo se encargará de una tarea específica en el programa.

La programación tradicional trabaja secuencialmente de forma que cuando termina un proceso entra otro. ***Los hilos / threads permiten ejecutar paralelamente diferentes partes de un mismo programa.***

El ***hilo*** es la ***unidad de procesamiento*** más pequeña que admiten los sistemas operativos; estos son ***entidades ligeras*** que hacen un gasto mínimo de recursos, ya que los de un mismo proceso ***comparten el espacio en memoria*** y su creación consume poco tiempo de procesador.

En Java el método main() crea el hilo de ejecución principal y se encarga de crear el resto de hilos del programa. Este debe programarse para que sea el último en finalizar, mediante una espera al resto de hilos.

Un hilo creado por el main() puede crear otros hilos, que a su vez pueden crear otros. Estos dependen directamente de sus creadores, mantienen una jerarquía de padres e hijos.

Java usa la clase Thread para la gestión de hilos, gracias a esta podemos crear y manipular hilos. La funcionalidad de esta clase se complementa con la funcionalidad de la clase Object destinada a resolver la sincronización entre hilos. La herencia permite incorporar la clase Thread… Debido a que Java no permite herencia múltiple se ofrece la alternativa de implementar interfaces Runnable que serán instanciadas en objetos Thread.

\*/

/\* ***Clase Object:***

Los métodos de la clase Object permiten la comunicación y manipulación de hilos. Todas las clases de Java los tienen también ya que esta es la clase padre de Java. Los métodos más destacables son:

***wait():*** Hace que un hilo en ejecución se ponga en espera hasta que se le notifique para continuar la ejecución.

***notify():*** Notifica a un hilo en espera.

***notifyAll():*** Notifica a todos los hilos que estén en espera.

Estos se usan exclusivamente en métodos sincronizados / synchronized.

\*/

/\* ***Creación y ejecución de un hilo:***

Cuando se produce la ejecución de un hilo ***se debe controlar*** su comportamiento para comprobar que hace lo que queremos y en el órden que queremos.

Existen dos formas de crear hilos en Java: Heredando la clase Thread o implementando la interfaz Runnable y pasarle el objeto de la clase a un objeto Thread. Ambas formas implementan un método run() que equivale al hilo de ejecución principal main().

Para ***iniciar la ejecución de un hilo*** se hace uso del ***método start(),*** este iniciará su ejecución en la primera instrucción del método ***run().***

Es posible que la ejecución del ***programa termine antes que los hilos, para evitarlo*** se determina un tiempo de espera al hilo mediante el ***método sleep() o join()*** que permite que el hilo padre espere a los hilos hijos.

\*/

***/\**** ***Estados y cambios de estado de un hilo:***

El ciclo de vida de un hilo tiene varios estados que adopta a lo largo de su ejecución:

1. Nuevo: Cuando se crea el hilo con el método new().
2. Preparado: Cuando se ejecuta el método start().
3. Ejecución: Cuando el organizador de hilos avisa que ya puede hacer uso del procesador.
4. Espera: Cuando se ejecuta el método wait(), no volverá a ejecutarse hasta que llegue a llamar a notify() / notifyAll().
5. Bloqueado: Un hilo en ejecución puede bloquearse si está dormido. Cuando un evento lo desbloquee o se despierte pasará de nuevo a preparado para volver a ejecutarse.

1. Acabado / Muerto: Cuando este finaliza su ejecución al llegar al final del método run(); aunque un hilo esté en este estado no desaparecerá ni liberará memoria, esto se puede forzar asignando a la variable que usa el hilo como null.

Un hilo estará vivo mientras se encuentre en los estados preparado, en ejecución o bloqueado.

Mientras este se encuentre ***en ejecución podrá salir por*** varios motivos***:***

1. Ha expirado el tiempo de uso del procesador o se ha ejecutado el método yield().
2. Inicia una operación de entrada/salida y pasa a bloqueado.
3. Pasa a bloqueado si se encuentra el método join() y tiene que esperar que los demás hilos terminen también.
4. Cuando intente ejecutar un método synchronized y no pueda porque otro hilo está ejecutándolo; este se bloqueará hasta que el otro termine.
5. Cuando llama al método ***wait()*** y pasa al ***estado de espera,*** hasta que se invoque el método ***notify() / notifyAll().***
6. Cuando se invoca el método sleep() el hilo pasa a dormido durante un tiempo, minimizando los recursos que usa.

El método isAlive() indica si un hilo ha sido inicializado por el método start(); si este devuelve false, el estado del hilo será nuevo o acabado; si devuelve true indica que el hilo ha sido inicializado con el método start() y está en otro estado que no sean nuevo y acabado.

***\*/***