Cheatsheet

Thread

```
public Thread(Runnable target)
```

Instancia un objeto de tipo *Thread*. Recibe por parámetro un objeto que implemente la interfaz *Runnable*.

```
public static void sleep(long millis)
```

Detiene la ejecución del hilo durante *millis* milisegundos.

```
public void start()
```

Ejecuta el hilo. La Máquina Virtual de Java ejecuta el método *run* de este objeto.

```
public long threadId()
```

Devuelve el identificador único del hilo. Se trata de un número positivo generado en el momento que se crea el hilo.

```
public final String getName()
public final void setName(String name)
```

Permiten obtener y asignar un nombre al hilo.

```
public final int getPriority()
public final void setPriority(int newPriority)
```

Permiten obtener y fijar la prioridad del hilo. La prioridad es un valor entero entre 1 y 10, siendo 1 la prioridad mínima y 10 la máxima. Estos valores los encontramos en las constantes *Thread.MIN_PRIORITY* y *Thread.MAX_PRIORITY*.

```
public Thread.State getState()
```

Obtiene el estado del hilo, que puede ser:

- NEW: Hilo nuevo que aún no se ha lanzado con el método *start()*.
- RUNNABLE: Hilo que se encuentra en ejecución. Esto es, que sus instrucciones se están ejecutando en el procesador.
- BLOCKED: Hilo que permanece bloqueado al tratar de acceder a un recurso ya ocupado.
- WAITING: Hilo que permanece a la espera indefinidamente hasta que se produzca un evento. El hilo permanecerá en este estado al llamar a los métodos *wait* y *join* en las versiones sin *timeout*.
- TIMED_WAITING: Hilo que permanece a la espera una cantidad determinada de tiempo. Cuando un hilo ejecuta el método *sleep* pasa a estar en este estado. También al llamar a los métodos *wait* y *join* cuando se utiliza un *timeout*.
- TERMINATED: Hilo finalizado.

El estado devuelto por *getState()* es un enumerado de tipo *Thread.State*.

```
public final boolean isAlive()
```

Devuelve si el hilo aún está vivo o no, es decir, si ya ha terminado su ejecución.

```
public static Thread currentThread()
```

Devuelve una referencia al hilo que actualmente se está ejecutando.

```
public final void join() throws InterruptedException
```

Espera a que el hilo finalice. Se trata de una llamada bloqueante.

```
void interrupt()
```

Interrumpe el hilo actual.

```
boolean isInterrupted()
```

Devuelve si el hilo actual ha sido interrumpido.

Object

```
void  wait()
void  wait(long timeout)
```

Hace que el hilo actual se suspenda a la espera de que otro hilo invoque al método *notify* o *notifyAll* de este objeto.

```
void notify()
```

Despierta a uno de los hilos suspendidos a la espera del monitor de este objeto. Este método solo debe ser llamado por un hilo que sea el propietario del monitor del objeto.

```
void notifyAll()
```

Despierta a todos los hilos suspendidos a la espera del monitor de este objeto. Este método solo debe ser llamado por un hilo que sea el propietario del monitor del objeto.

ExecutorService (interfaz)

```
void execute (Runnable command)
```

Ejecuta en un hilo la tarea que recibe por parámetro.

```
void shutdown()
```

Hace que el *executor* no acepte más llamadas. El *executor* esperará a que terminen de ejecutarse las tareas que se encuentren en ejecución y las que estén a la espera de ejecutarse. Una vez finalizadas se cierra el *executor*.

```
List<Runnable> shutdownNow()
```

Finaliza las tareas en ejecución de forma abrupta. Devuelve una lista con las tareas que estaban a la espera de ejecutarse.

```
boolean awaitTermination(long timeout, TimeUnit unit)
```

Llamada bloqueante que espera hasta que todas las tareas finalicen su ejecución o hasta que finalice la espera indicada en *timeout*. Este método se suele llamar tras llamar al método *shutdown* par esperar la finalización de las tareas.

ThreadPoolExecutor

Implements *ExecutorService*

```
public int getPoolSize()
```

Devuelve el número de hilos que existen en el *pool*.

```
public int getPoolSize()
```

Devuelve el número de hilos que existen en el *pool*.

```
public int getActiveCount()
```

Devuelve el número de hilos que están ejecutando alguna tarea.

ScheduledExecutorService

Hereda de *ThreadPoolExecutor*

```
ScheduledFuture<?> schedule(Runnable command, long delay,
                            TimeUnit unit)
```

Lanza una tarea cuando transcurra el tiempo indicado en *delay*. El parámetro *unit* indica la unidad de tiempo utilizada.

```
ScheduledFuture<?> scheduleAtFixedRate(Runnable command,
                       long initialDelay, long period, TimeUnit unit)
```

Lanza una tarea una vez pasado el tiempo indicado en el parámetro *initialDelay* y la ejecutará repetidamente con la frecuencia indicada en el parámetro *period*.

Executors

```
static ExecutorService newFixedThreadPool(int nThreads)
```

Crea un *thread pool* que reutiliza un número fijo de hilos *nThreads* con una cola de espera ilimitada.

```
static ExecutorService newSingleThreadExecutor()
```

Crea un *thread pool* de un único hilo con una cola de espera ilimitada.

```
static ExecutorService newCachedThreadPool()
```

Crea un *thread pool* que permite crear tantos hilos nuevos como resulten necesarios, reutilizando los que se hubiesen construido previamente si están disponibles.

```
static ScheduledExecutorService
            newScheduledThreadPool(int corePoolSize)
```

Crea un *thread pool* de un número fijo de hilos *corePoolSize* hilos que se ejecutarán tras un determinado tiempo de espera o periódicamente.

```
static ScheduledExecutorService newSingleThreadScheduledExecutor()
```

Crea un *thread pool* que permite programar un único hilo que se ejecutará tras un determinado tiempo de espera o periódicamente.