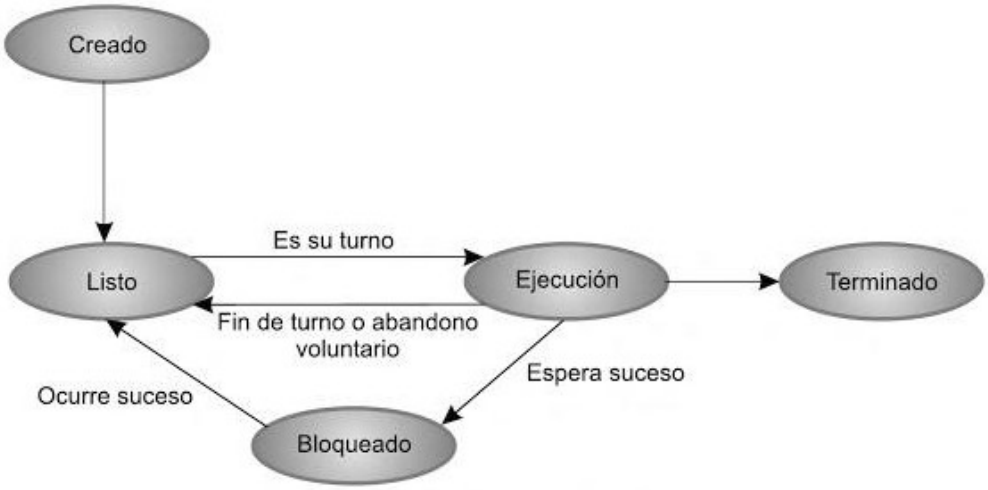
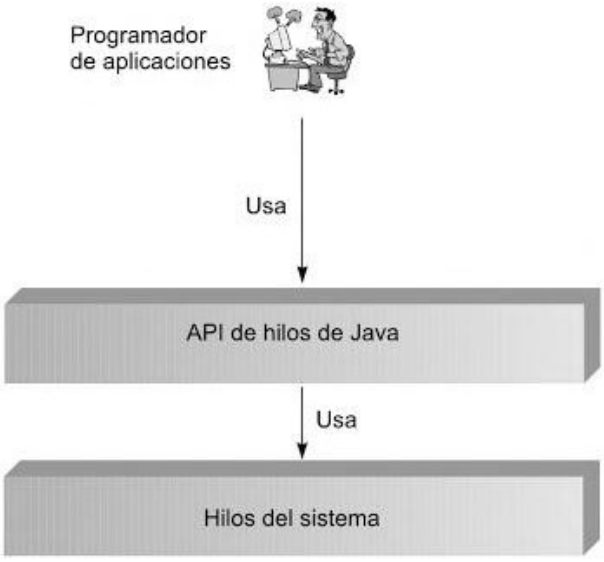
Capítulo 2
Procesos versus hilos

A continuación vemos el diagrama del ciclo de vida de un proceso, es prácticamente estándar en todos los sistemas operativos.



En un sistema operativo la memoria suele estar dividida en dos partes, un espacio de usuario donde se encuentra la mayoría de la información relativa a los procesos de usuario y un espacio del núcleo donde reside código del SO y estructuras de datos propios.
En un SO multitarea la información de un proceso está dividida en los dos espacios, la información que está en el núcleo (indispensable para hacer un cambio de contexto **context switch**) se la conoce como **PCB** (bloque control de proceso), cuando se realiza un cambio de contexto hay que recuperar toda la estructura de un proceso y esto es muy costoso desde el punto de vista de tiempo de ejecución. Cada proceso suele tener un hilo de ejecución (monohilo).

Hilos (thread / hebra)
A fines de los 90 el uso de hilo comenzó a utilizarse con fuerza, pero ¿qué es un hilo?, de la misma manera que un SO puede ejecutar varios procesos al mismo tiempo por concurrencia o paralelismo, dentro de un proceso puede haber varios hilos de ejecución, por lo tanto se puede definir hilo como: "cada secuencia de control dentro de un proceso que ejecuta sus instrucciones de forma independiente".
Los procesos son entidades pesadas, parte de la estructura del proceso está en el núcleo y cada vez que un proceso quiere acceder a ella se debe hace un "system call" consumiendo tiempo extra del procesador, por lo tanto los cambios de contexto "context switch" son muy costosos. Por el contrario la estructura de los hilos reside en el espacio del usuario con lo que es una entidad ligera; los hilos comparten información del proceso (código, datos, etc.), si un hilo modifica una variable del proceso los demás hilos verán esa modificación, un detalle no menos importante es que los cambios de contexto entre hilos consumen poco tiempo de procesador.
Podemos hablar de dos niveles de hilos, los que nosotros usaremos para programar y los hilos propios del SO que sirven para dar soporte a los hilos de usuario; cuando nosotros programamos con hilos hacemos uso de una API (application program interface) proporcionada por el lenguaje, SO o una librería.



Hilos en Java
Al ejecutar un programa en Java, existe un hilo en ejecución, el indicado por el método main (hilo principal).

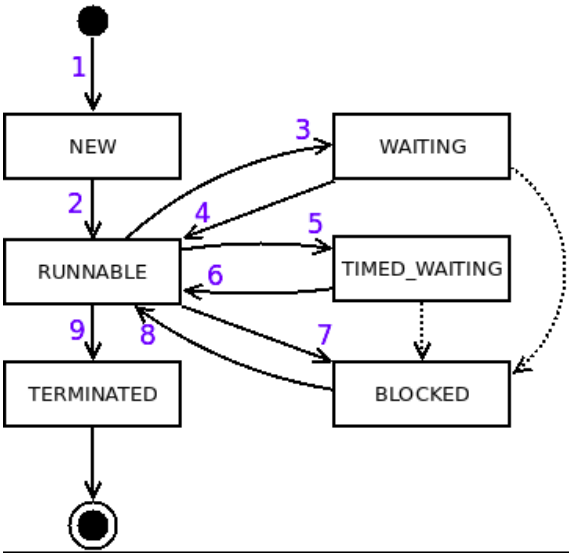
Existen dos formas de crear hilos:

- Heredando de la clase Trhead
- Implementando la interface Runnable

Cuando heredamos de Thread hay que redefinir el método run(), en este método se escribe el código que queremos que se ejecute cuando se ejecute el hilo.

Utilizaremos la forma de implementar Runnable debido a que Java no tiene herencia múltiple y con esta podemos definir un thread sin desperdiciar la única herencia.

Estados
java.lang.Thread.State <Enum>



Prioridades:
Los hilos en java tienen prioridades, sin embargo no se garantiza que se este ejecutando el hilo de mayor prioridad, esto depende del SO, por lo tanto siempre hay que esperar el peor escenario, es decir, los hilos pueden intercalarse en cualquier momento y nuestros programas no pueden estar basados en que vaya a haber un intercalado entre hilos.
Prioridades de hilos en Java: MIN_PRIORITY, MAX_PRIORITY y NORM_PRIORITY.

Hilos daemon:
Antes de ejecutar el hilo se puede definir como Daemon, este tipo de hilo realiza una ejecución continua durante toda la aplicación como tarea de fondo (baja prioridad), se define con el método setDaemon(true). La propia JVM pone en ejecución hilos daemon, entre ellos destacamos el garbage collector (libera memoria ocupada por objetos que ya no están referenciados).

Algo de la clase Thread:

CONSTRUCTORES	
public	**Thread** () Crea un nuevo objeto Thread. Este constructor tiene el mismo efecto que `Thread (null, null, gname)`, donde ***gname*** es un nombre generado automáticamente y que tiene la forma `"Thread-"+n`, donde n es un entero asignado consecutivamente.
public	**Thread** (String name) Crea un nuevo objeto Thread, asignándole el nombre ***name***.
public	**Thread** (Runnable target) Crea un nuevo objeto Thread. ***target*** es el objeto que contiene el método `run()` que será invocado al lanzar el hilo con `start()`.
public	**Thread** (Runnable target, String name) Crea un nuevo objeto Thread, asignándole el nombre ***name***. ***target*** es el objeto que contiene el método `run()` que será invocado al lanzar el hilo con `start()`.

MÉTODOS	
public static Thread	**currentThread** () Retorna una referencia al hilo que se está ejecutando actualmente.

MÉTODOS	
public String	**getName** () Retorna el nombre del hilo.
int	**getPriority** () Retorna la prioridad del hilo.
public final boolean	**isAlive** () Chequea si el hilo está vivo. Un hilo está vivo si ha sido lanzado con start y no ha muerto todavía.
public final void	**isDaemon** () Devuelve verdadero si el hilo es daemon.
public void	**run** () Si este hilo fue construido usando un objeto que implementaba Runnable, entonces el método `run` de ese objeto es llamado. En cualquier otro caso este método no hace nada y retorna.
public final void	**setDaemon** (boolean on) Marca este hilo como daemon si el parámetro ***on*** es verdadero o como hilo de usuario si es falso. El método debe ser llamado antes de que el hilo sea lanzado.
public final void	**setName** (String name) Cambia el nombre del hilo por ***name***.
public final void	**setPriority** (int newPriority) Asigna la prioridad ***newPriority*** a este hilo.
public static void	**sleep** (long millis) throws InterruptedException Hace que el hilo que se está ejecutando actualmente cese su ejecución por los milisegundos especificados en ***millis***. Pasa al estado dormido. El hilo no pierde la propiedad de ningún cerrojo que tuviera adquirido con *synchronized*.
public static void	**sleep** (long millis, int nanos) throws InterruptedException Permite afinar con los nanosegundos ***nanos*** el tiempo a estar dormido.
public void	**start** () Hace que este hilo comience su ejecución. La MVJ llamará al método `run` de este hilo.
public String	**toString** () Devuelve una representación en forma de cadena de este hilo, incluyendo su nombre, su prioridad y su grupo.
public static void	**yield** () Hace que el hilo que se está ejecutando actualmente pase al estado listo, permitiendo a otro hilo ganar el procesador.