

Nueva multitarea

La interfaz ExecutorService

➤ Proporciona métodos para la ejecución de tareas de forma concurrente, utilizando un pool de threads. Entre estos métodos:

- `submit(Runnable tarea)`. Lanza la tarea y la pone en ejecución concurrente con el resto
- `submit(Callable tarea)`. Lo mismo que el anterior, pero para objetos Callable
- `shutdown()`. Inicia el final del pool de hilos, por lo que no se aceptarán nuevas tareas

Creación de un ExecutorService

➤ Se pueden crear implementaciones de ExecutorService a partir de los siguientes métodos estáticos de Executors:

- `newCachedThreadPool()`. Crea un ExecutorService con un pool de Thread variable que se crean a demanda
- `newFixedThreadPool(int hilos)`. Crea un pool con un número fijo de threads
- `newSingleThreadExecutor()`. Crea un ExecutorService que utiliza un único Thread

Interfaz Callable

- Implementa una tarea que va a ser ejecutada concurrentemente con otras.
- Similar a Runnable, aunque su método puede devolver un resultado

```
public interface Callable<T>{  
    T call() throws Exception;  
}
```

Intefaz Future

- El método `submit(Callable tarea)` de `ExecutorService` devuelve un objeto `Future` que puede ser utilizado para acceder al resultado de la tarea y controlar su ejecución.
- Entre sus métodos están:
 - `isDone()`. Permite conocer si la tarea ha finalizado
 - `get()`. Devuelve el valor generado por `Callable`. Si aún no ha terminado la tarea, queda a la espera del resultado

```
Future<Tipo_resultado> t1 = exec.submit(objetoCallable);  
while(!t1.isDone()){  
    System.out.println("Esperando fin tarea");  
}  
System.out.println("El resultado de la tarea es "+t1.get());
```

Sincronización

➤ En las nuevas clases de multitarea la sincronización se lleva a cabo con la interfaz Lock que proporciona los siguientes métodos:

- `lock()`. Bloquea acceso al código a otros hilos
- `unlock()`. Desbloquea el acceso al código

➤ Se puede obtener una implementación de Lock instanciando `ReentrantLock`

```
Lock lc=new ReentrantLock()  
lc.lock(); //bloquea el acceso  
:  
lc.unlock(); //desbloquea acceso
```

Condiciones

- Se utiliza en contextos de espera y notificaciones, como alternativa a *wait()* y *notify()*
- Un objeto *Condition* se crea mediante el método *newCondition()* de *Lock*
- Utilizando los métodos *await()* y *signal()* se puede realizar la espera y notificación entre procesos:

```
Condition cond=lc.newCondition();  
lc.lock();  
:  
cond.await(); //el hilo se mantiene en espera  
:  
cond.signal();//otro hilo manda una señal para que salga de espera  
:  
lc.unlock();
```