RUNNABLE, CALLABLE Y EXECUTORSERVICE

1.

RUNNABLE

RUNNABLE

- Si nuestra clase ya hereda de una, no puede heredar de *Thread*.
- Runnable es un interfaz que nos permite crear tareas para ser ejecutadas en hilos secundarios.
- *Thread* tiene un constructor que permite pasar como argumento un *Runnable*.

```
public interface Runnable {
   public void run();
}
```

RUNNABLE

```
public class HelloRunnable implements Runnable {
    public void run() {
        System.out.println("Hello from a thread!");
    }

    public static void main(String args[]) {
        (new Thread(new HelloRunnable())).start();
    }
}
```

2.

CALLABLE

CALLABLE<V>

- Runnable (o Thread) no permiten devolver valores.
- Complejos mecanismos de sincronización para hacerlo.
- Callable es, básicamente, un Runnable que puede devolver un valor.

```
public interface Callable<V> {
   public V call();
}
```

CALLABLE<V>

```
public class PrimoCallable implements Callable<Long> {

   private long minimo;

   public PrimoCallable(long minimo) {
      this.minimo = minimo;
   }

   @Override
   public Long call() throws Exception {
      long n = minimo;
      System.out.println("Comenzamos a buscar un número primo");
      while(!testPrimalidad(n)) {
            System.out.printf("%d no es primo %n", n);
      ++n;
      }
      return n;
   }
}
```

FUTURE<V>

- Interfaz que representa el resultado de una *computación* asíncrona.
- Nos permite algunas operaciones: comprobar el resultado, saber si la computación ha terminado, esperar a que termine, …
- Método **get** para obtener el valor de la ejecución de un **Callable<V>**.
- Nos *invita* a usar **Executor**

3.

EXECUTORs

MANEJO DE HILOS A ALTO NIVEL

- Hasta ahora, el programador definía y lanzaba los hilos de ejecución según su necesidad.
- Válido para aplicaciones pequeñas.
- Para grandes aplicaciones, hay que separar la administración de los hilos del resto de la aplicación.
- Esto lo podemos realizar mediante *ejecutores* (Executors).

EXECUTORs

3 interfaces

- *Executor*: soporta el lanzamiento de nuevas tareas, bajo demanda.
- ***ExecutorService*: añade a la anterior características que permiten administrar el ciclo de vida.**
- *ScheduledExecutorService*: añade a la anterior la posibilidad de ejecutar tareas periódicas.

EXECUTORSERVICE

- ***submit(…)*** acepta ***Runnable*** o ***Callable***.
- Métodos para la finalización del propio ejecutor.
- Creación a partir de un pool de hilos que haga uso de *worker threads*: hilos que son reutilizables, minimizando la sobrecarga de la creación de hilos nuevos.
- Podemos finalizar el ejecutor con el método ***shutdown***.

POOLs DE HILOS

Single

- Con un solo hilo de ejecución disponible.
- Si le pedimos (*submit*) más de una tarea a la vez, las pone en cola.

Fixed

- Indicamos, en el momento de su creación, el número de hilos.
- Si dispone de n hilos, y enviamos n+1 tareas, las pone en cola.

POOLs DE HILOS

Cached

- Crea hilos conforme enviamos tareas
- Reutiliza los hilos cuyas tareas han finalizado, para ejecutar tareas nuevas.

CREACIÓN DE POOLs DE HILOS

La clase Executors tiene métodos estáticos para construir cada tipo. Entre ellos

- *newSingleThreadExecutor()*: crea un ejecutor de tipo *single*.
- *newFixedThreadPool(int n)*: crea un ejecutor de tipo *fixed* con n hilos disponibles.
- *newCachedThreadPool()*: crea un ejecutor de tipo *cached*.