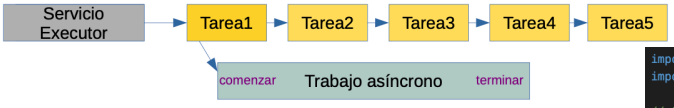
Executors

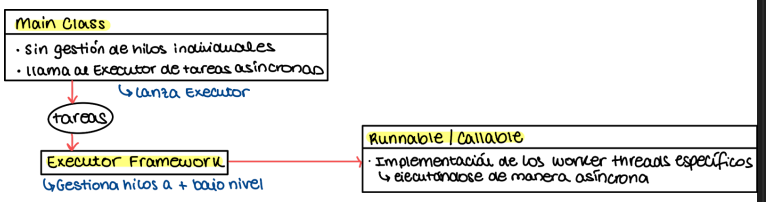
Ejecución asíncrona

- Hilo delga una tarea al ExecutorService, y continua su propia ejecución
- El ExecutorService ejecuta la tarea de forma concurrente, independientemente del ciclo de vida del hilo que la presentó



Executors

Envía tareas para ejecución asíncrona




```java
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

// Clase EXECUTOR de EJEMPLO
public class ExecutorExample {

    Run | Debug
    public static void main(String[] args) {

        // Crea un ExecutorService con un pool fijo de 2 hilos
        ExecutorService executor = Executors.newFixedThreadPool(nThreads:2);

        // ENVÍA una tarea al executor
        executor.submit(() -> {
            System.out.println("Task executed by: " + Thread.currentThread().getName());
        });

        // CIERRA el executor después de ejecutar las tareas enviadas
        executor.shutdown();
    }
}
```

Componentes básicos de los Executors:

Interfaz Executor:

- La interfaz principal para asignar tareas Runnable a un ejecutor

Interfaz ExecutorService:

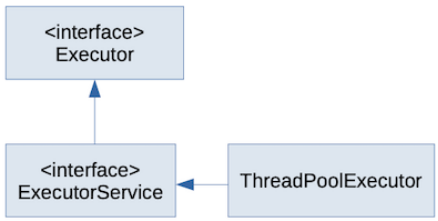
- Una extensión de Executor, que agrega métodos para cancelar tareas y apagar el servicio.

Clase ThreadPoolExecutor:

- Implementa Executor y ExecutorService, ofreciendo una gestión flexible de grupos de hilos.

Clase Executors:

- Proporciona métodos de utilidad para crear fácilmente servicios de ejecutor y grupos de hilos.



- Métodos de Thread Pool:
  - **newFixedThreadPool(n)** → Pool con n hilos fijos, las tareas extra esperan en cola.
  - **newCachedThreadPool()** → Pool dinámico, reutiliza hilos o crea nuevos según necesidad.
  - **newSingleThreadExecutor()** → Un solo hilo, ejecuta tareas en orden
  - **secuencial.newScheduledThreadPool(n)** → Pool de n hilos para tareas programadas o repetitivas.

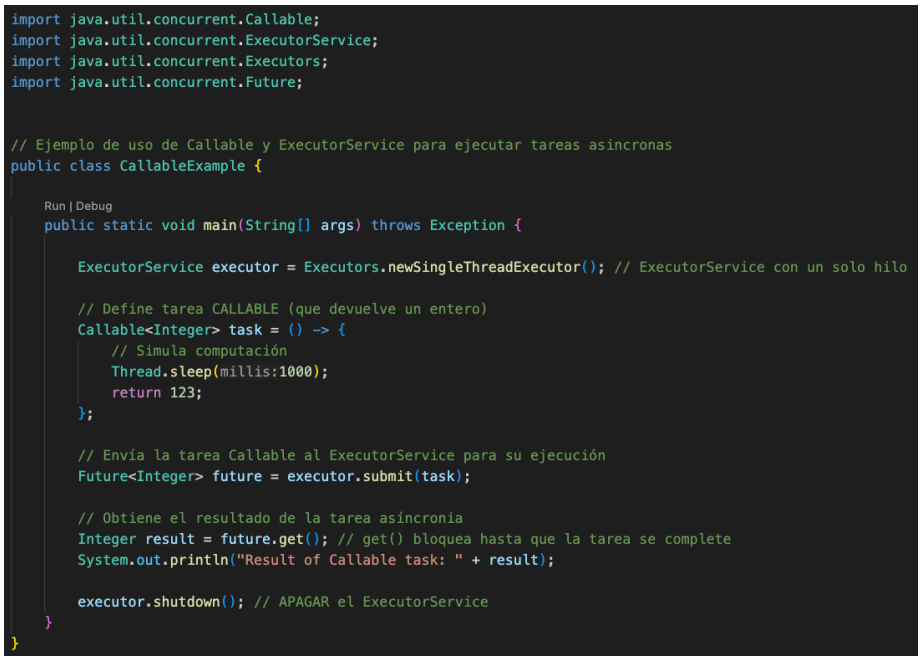
Future y Callables (Executor interfaces)

Future Interface: Future<V>

```java
import java.util.concurrent.*;

// Clase FUTURE EXAMPLE (ejemplo de uso de Future Interface )
public class FutureExample {

    Run | Debug
    public static void main(String[] args) throws ExecutionException, InterruptedException {

        // Crea EXECUTOR SERVICE
        ExecutorService executor = Executors.newCachedThreadPool(); // Thread pool dinámico

        // Envía una tarea asíncrona al executor y obtiene un FUTURE que representa su resultado
        Future<String> future = executor.submit(() -> {
            Thread.sleep(millis:2000); // Simulate a long-running task
            return "Result of the Callable task";
        });

        // Otras acciones mienteas está en progreso la tarea


        // Bloquea la ejecución hasta que el resultado esté disponible
        String result = future.get();

        // Imprime el resultado
        System.out.println(result);

        // APAGA the ExecutorService
        executor.shutdown();
    }
}
```

- Representa el resultado de una computación asíncona; actuando como marcador de posición para un resultado que estará disponible en el futuro
- **get()** obtiene el resultado cuando está completo
- **cancel()** intenta cancelar una tarea
- **isDone()** e **isCancelId()** comprueban el estado de la tarea

Callable Interface: Callable<V>



```java
import java.util.concurrent.Callable;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
import java.util.concurrent.Future;


// Ejemplo de uso de Callable y ExecutorService para ejecutar tareas asincronas
public class CallableExample {

    Run | Debug
    public static void main(String[] args) throws Exception {

        ExecutorService executor = Executors.newSingleThreadExecutor(); // ExecutorService con un solo hilo

        // Define tarea CALLABLE (que devuelve un entero)
        Callable<Integer> task = () -> {
            // Simula computación
            Thread.sleep(millis:1000);
            return 123;
        };

        // Envía la tarea Callable al ExecutorService para su ejecución
        Future<Integer> future = executor.submit(task);

        // Obtiene el resultado de la tarea asíncronia
        Integer result = future.get(); // get() bloquea hasta que la tarea se complete
        System.out.println("Result of Callable task: " + result);

        executor.shutdown(); // APAGAR el ExecutorService
    }
}
```

- Representa una tarea asíncrona que devuelve un resultado, generalmente ejecutada por otro hilo usando ExecutorService
- Diseñada para tareas que deben ser ejeuctadas por otro hilo, como Runnable
- Incluye un método **V call() throws Exception**, que permite devolver un valor y lanzar excepciones (al contrario que Runnable)