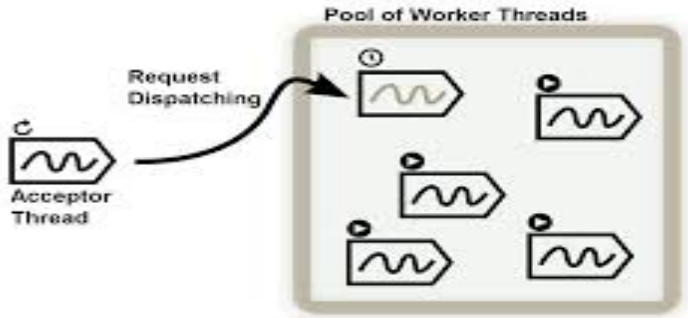
Herramientas de java.util.concurrent

- La biblioteca incluye varias herramientas útiles:
  - Locks
  - Variables condición
  - Colecciones concurrentes
  - Variables atómicas
  - Sincronización: Semáforos y Barreras
  - Entorno para la gestión de hilos (Executor)

Entorno para la ejecución de hilos

- La **creación de hilos** es una operación costosa, que requiere muchos recursos y puede resultar lenta.
- **Interfaz *Executor***: ofrece un entorno para la creación y gestión de hilos en Java
  - Permite su invocación, planificación, ejecución y control de políticas de ejecución.



- Ventajas:
  - Previene el consumo desmedido de recursos
  - Se dispone de una biblioteca ya implementada (Executor) para crear tareas de una forma muy flexible
- Este tipo de entorno está ampliamente utilizado

Entorno para la ejecución de hilos

- La interfaz Executor ofrece la subinterfaz ExecutorService, que permite crear diferentes tipos de **Executors** :
  - Un hilo único (*SingleThreadExecutor*): utiliza un solo hilo de trabajo que opera en una cola sin límites.
    - Se garantiza que las tareas se ejecutan de forma secuencial, y que no habrá más de una tarea activa en un momento dado.

```
ExecutorService executorService1 = Executors.newSingleThreadExecutor();
```

Crea un solo hilo en el que las tareas quedan en cola y se ejecutan de forma secuencial

  - Un *thread-pool* (ej.- para servidor)
    - Permite mantener un conjunto de hilos ya creados, reciclándolos para que ejecuten nuevas tareas.

```
ExecutorService executorService2 = Executors.newFixedThreadPool(10);
```

Crea un pool con 10 hilos

Entorno para la ejecución de hilos

- Ejemplo sencillo de uso de un ExecutorService:

```
ExecutorService executorService =
                    Executors.newFixedThreadPool(10);

executorService.execute(new Runnable() {
    public void run() {
        System.out.println("Asynchronous task");
    }
});

executorService.shutdown();
```

Entorno para la ejecución de hilos

- Existen diferentes formas para delegar tareas para la ejecución de un ExecutorService:
  - execute(Runnable), submit(Runnable), submit(Callable), invokeAny(…), invokeAll(…)

- Nos centraremos aquí en el uso de Runnable

```
Runnable command = new Runnable() {
    public void run() {
        doLongWork();
    }
};

ExecutorServe executor = Executors.newSingleThreadExecutor();
executor.execute(command);
```

Entorno para la ejecución de hilos

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

public class ExecutorServiceExample {

   public static void main(String[] args) {

      ExecutorService executor = Executors.newFixedThreadPool(10);
      executor.execute(() -> doLongWork("hi 1"));
      executor.execute(() -> doLongWork("hi 2"));
      executor.execute(() -> doLongWork("hi 3"));
   }

   private static void doLongWork(String hola) {
      System.out.println("Running " + hola);
      try {
         Thread.sleep(1000l);
      } catch (InterruptedException e) {
         e.printStackTrace();
      }
   }
}
```

Nota: se ha utilizado notación Lambda (no entraremos en el detalle)

Entorno para la ejecución de hilos

- Por defecto, ExecutorService se queda "escuchando" nuevas tareas
  - Queda activo para recibir nuevas tareas y asignarlas a su pool
  - Aunque acabe el hilo main, ExecutorService continúa activo

- Existen dos formas para detener el **ExecutorService**:
  - shutdown() → permite detener los hilos del Executor.
    - No se detiene inmediatamente, pero ya no aceptará nuevas tareas
    - Cuando todos los hilos hayan acabado con sus tareas actuales, entonces se detendrá
  - shutdownNow() → trata de detener todas las tareas en ejecución
    - Las tareas presentadas pero no procesadas las ignora
    - Intenta parar las tareas en ejecución (aunque algunas podrían ejecutarse hasta su finalización)
    - Devuelve la lista de tareas que no finalizaron

Entorno para la ejecución de hilos

- Importante: aunque el hilo principal (main) de la aplicación termine, el ExecutorService continuará ejecutándose (y, por tanto, la aplicación), mientras esté activo.

- Podemos utilizar awaitTermination() para esperar a que ExecutorService se detenga.
  - Ejemplo:

```
executorService.shutdown();
executorService.awaitTermination();
```

Resultados de aprendizaje de la Unidad Didáctica

- Al finalizar esta unidad, el alumno deberá ser capaz de:
  - Identificar los inconvenientes de las primitivas básicas de Java.
  - Describir las herramientas del package *java.util.concurrent*, que facilitan el desarrollo de aplicaciones concurrentes:
    - Ilustrar la utilización de los *locks* y las *condiciones*. Compararlos con los monitores.
    - Interpretar uso de colecciones concurrentes *thread-safe* (e.g. *BlockingQueue*)
    - Describir el funcionamiento de las clases atómicas. Comparar *AtomicInteger* con los monitores.
    - Ilustrar el uso de semáforos (*Semaphore*) para sincronización.
    - Ilustrar el funcionamiento de las barreras, distinguiendo entre *CyclicBarrier* y *CountDownLatch*.
    - Conocer el entorno de gestión de hilos *Executor*