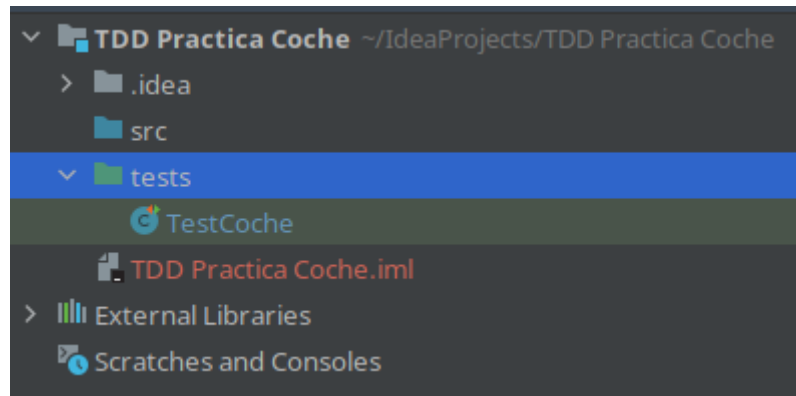


# Creando TDD

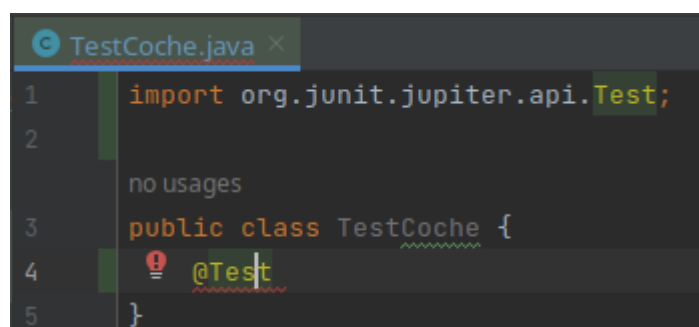
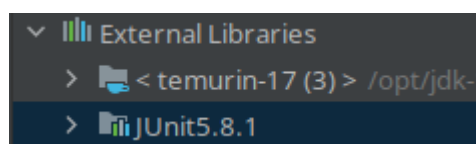
El primer paso para crear el TDD será crear una carpeta llamada tests y marcarla como carpeta de tests, se nos pondrá el directorio de color verde.



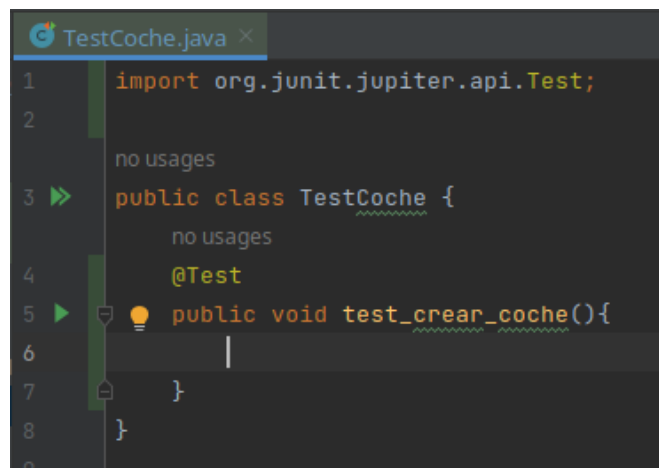
Ahora, dentro de esta carpeta, como podemos ver hemos creado la clase TestCoche, la cual utilizaremos para realizar los tests. Una vez la tenemos creada, la marcaremos como test poniendo @Test dentro de la clase.



Como vemos, no nos reconoce la palabra Test, para solucionar esto pulsamos Alt+Enter y nos aparecerán varias opciones, seleccionaremos la de añadir las librerías de JUnit 5.8.1. Una vez hecho esto nos aparecerá la carpeta JUnit 5.8.1 en External Libraries, y se nos habrá importado la clase para que funcione el @Test.

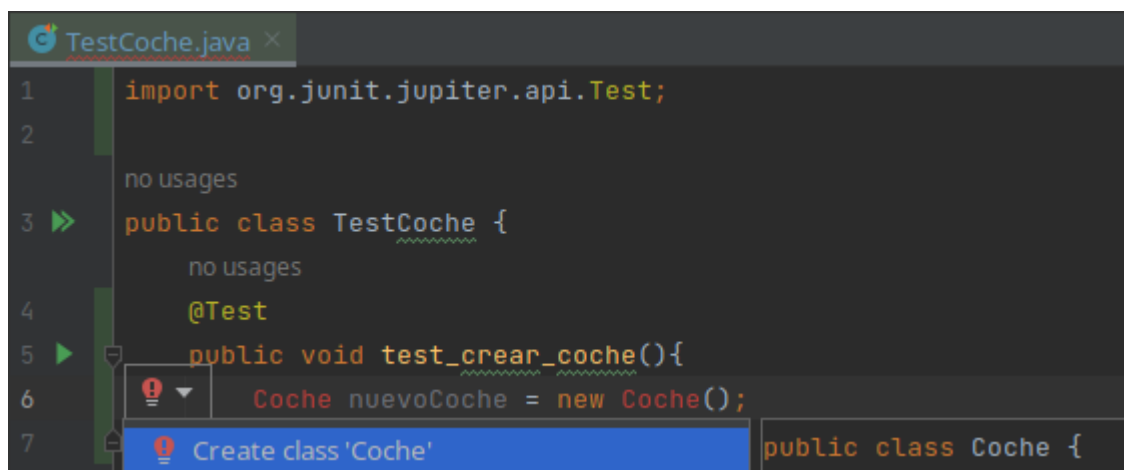


Ahora crearemos un método llamado “test\_crear\_coche”.



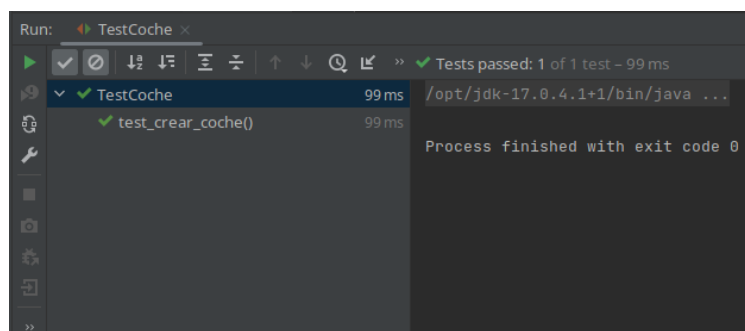
```
TestCoche.java x
1  import org.junit.jupiter.api.Test;
2
3  public class TestCoche {
4      @Test
5      public void test_crear_coche(){
6
7      }
8  }
```

Ahora dentro de este método crearemos un objeto de la clase Coche que aún no ha sido creado. Para crear esta clase automáticamente haremos click en la bombilla roja que nos aparecerá y le damos a “Create class Coche”.



```
TestCoche.java x
1  import org.junit.jupiter.api.Test;
2
3  public class TestCoche {
4      @Test
5      public void test_crear_coche(){
6          Coche nuevoCoche = new Coche();
7      }
8  }
```

Una vez hecho esto vamos a ejecutar la clase TestCoche.



```
Run: TestCoche x
✓ Tests passed: 1 of 1 test - 99 ms
✓ TestCoche 99 ms /opt/jdk-17.0.4.1+1/bin/java ...
  ✓ test_crear_coche() 99 ms
Process finished with exit code 0
```

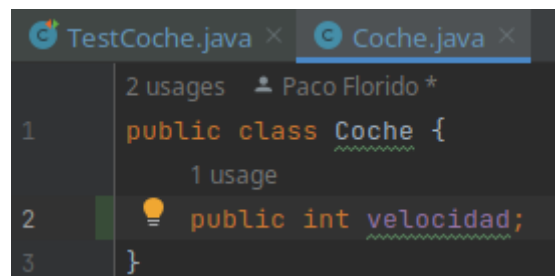
Y así habremos pasado nuestro primer test correctamente. Ahora vamos a complicarlo un poco más ya que este ejemplo es demasiado simple. Cambiaremos el nombre al método “test\_crear\_coche” por “test\_al\_crear\_coche\_su\_velocidad\_es\_cero”.

```
public void test_al_crear_coche_su_velocidad_es_cero(){  
    Coche nuevoCoche = new Coche();  
}
```

Ahora vamos a poner la siguiente línea de código dentro del método:

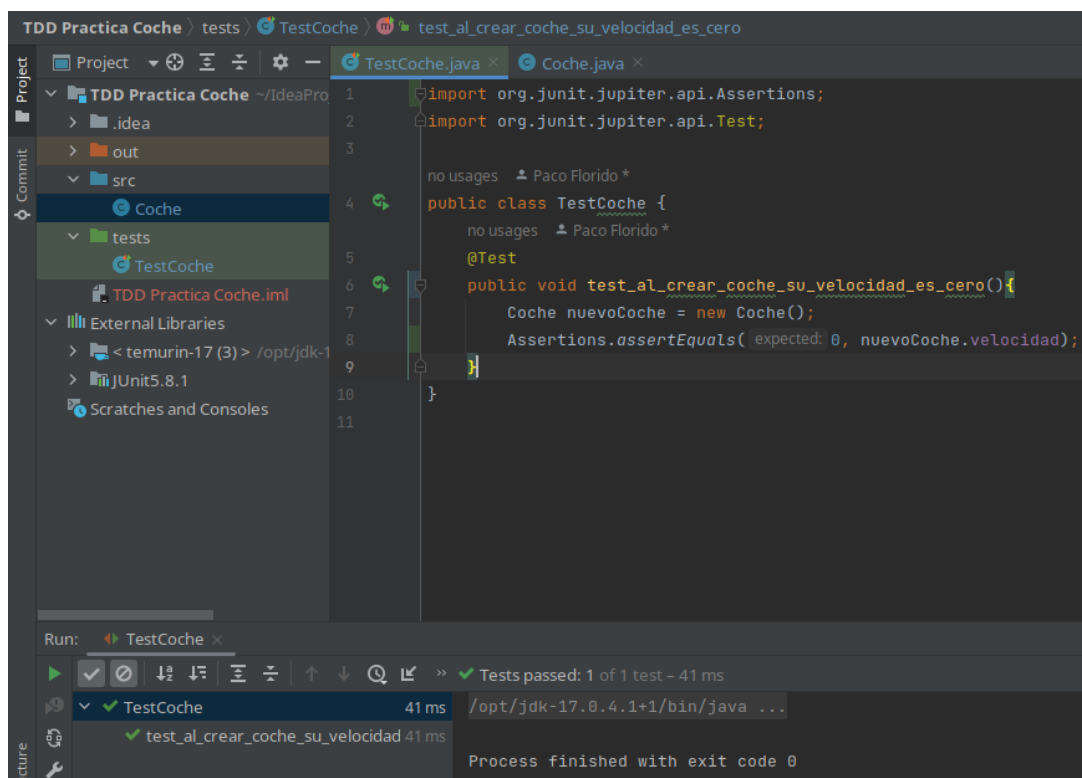
```
Assertions.assertEquals(0, nuevoCoche.velocidad);
```

Esta línea lo que hará será comprobar que al crear el objeto nuevoCoche la velocidad inicial sea 0. Para ello deberemos crear el atributo velocidad en la clase Coche.



```
public class Coche {  
    public int velocidad;  
}
```

Una vez hecho esto ejecutamos de nuevo la clase TestCoche y vemos que hemos pasado el test correctamente.



```
import org.junit.jupiter.api.Assertions;  
import org.junit.jupiter.api.Test;  
  
public class TestCoche {  
    @Test  
    public void test_al_crear_coche_su_velocidad_es_cero(){  
        Coche nuevoCoche = new Coche();  
        Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);  
    }  
}
```

Run: TestCoche

Tests passed: 1 of 1 test - 41 ms

TestCoche 41 ms

test\_al\_crear\_coche\_su\_velocidad 41 ms

Process finished with exit code 0

Ahora crearemos un nuevo método llamado:

“test\_al\_acelerar\_un\_coche\_su\_velocidad\_aumenta”.

En este método lo que haremos será testear que la velocidad del coche aumenta cuando utilicemos el método acelerar que vamos a crear en la clase Coche.

```
@Test
public void test_al_acelerar_un_coche_su_velocidad_aumenta(){
    Coche nuevoCoche = new Coche();
    nuevoCoche.acelerar(30);
    Assertions.assertEquals( expected: 30, nuevoCoche.velocidad);
}
```

```
TestCoche.java x Coche.java x
4 usages Paco Florido *
1 public class Coche {
2     3 usages
3     public int velocidad;
4     1 usage new *
5     public void acelerar(int aceleracion) {
6         velocidad += aceleracion;
7     }
}
```

Ahora ejecutamos, y si hemos seguido los pasos correctamente deberíamos pasar el test sin ningún problema.

```
Run: TestCoche x
[Icons] [TestCoche] 50 ms /opt/jdk-17.0.4.1+1/bin/java ...
  ✓ test_al_crear_coche_su_velocidad_es_cero() 49 ms
  ✓ test_al_acelerar_un_coche_su_velocidad_aumenta() 1 ms
Process finished with exit code 0
```

El siguiente paso será crear un nuevo método para comprobar la deceleración del coche, este método se llamará:

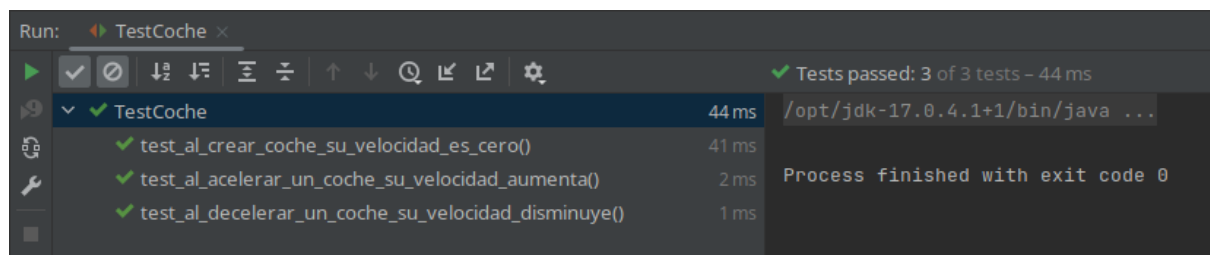
“test\_al\_decelerar\_un\_coche\_su\_velocidad\_disminuye”.

Este método comprobará que el método que vamos a crear en la clase Coche llamado decelerar funciona correctamente. El código de estos dos métodos es el siguiente.

```
1 usage new*
public void decelerar(int deceleracion) {
    velocidad -= deceleracion;
}

@Test
public void test_al_decelerar_un_coche_su_velocidad_disminuye(){
    Coche nuevoCoche = new Coche();
    nuevoCoche.velocidad = 50;
    nuevoCoche.decelerar( deceleracion: 20);
    Assertions.assertEquals( expected: 30, nuevoCoche.velocidad);
}
```

Si ejecutamos el test ahora vemos que hemos pasado el test correctamente.



Ahora crearemos otro nuevo método de test que se llamará:

“test\_al\_decelerar\_un\_coche\_su\_velocidad\_no\_puede\_ser\_menor\_que\_cero”

Con este método comprobaremos que la velocidad no puede ser menor que cero al decelerar un coche. El código de este método de test sería el siguiente:

```
@Test
public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero() {
    Coche nuevoCoche = new Coche();
    nuevoCoche.velocidad = 50;
    nuevoCoche.decelerar(80);
    Assertions.assertEquals(0, nuevoCoche.velocidad);
}
```

Si ejecutamos ahora vemos que no habremos pasado el test en este método, esto es porque no hemos modificado el código del coche para que al decelerar el coche, su velocidad no pueda ser menor que 0. Para solucionar esto hay que modificar el código del método decelerar creado anteriormente.

```
org.opentest4j.AssertionFailedError:  
Expected :0  
Actual   :-30  
<Click to see difference>
```

```
public void decelerar(int deceleracion) {  
    velocidad -= deceleracion;  
    if (velocidad < 0) velocidad = 0;  
}
```

Con esta modificación realizada ejecutamos de nuevo y ya nos funcionará correctamente.

