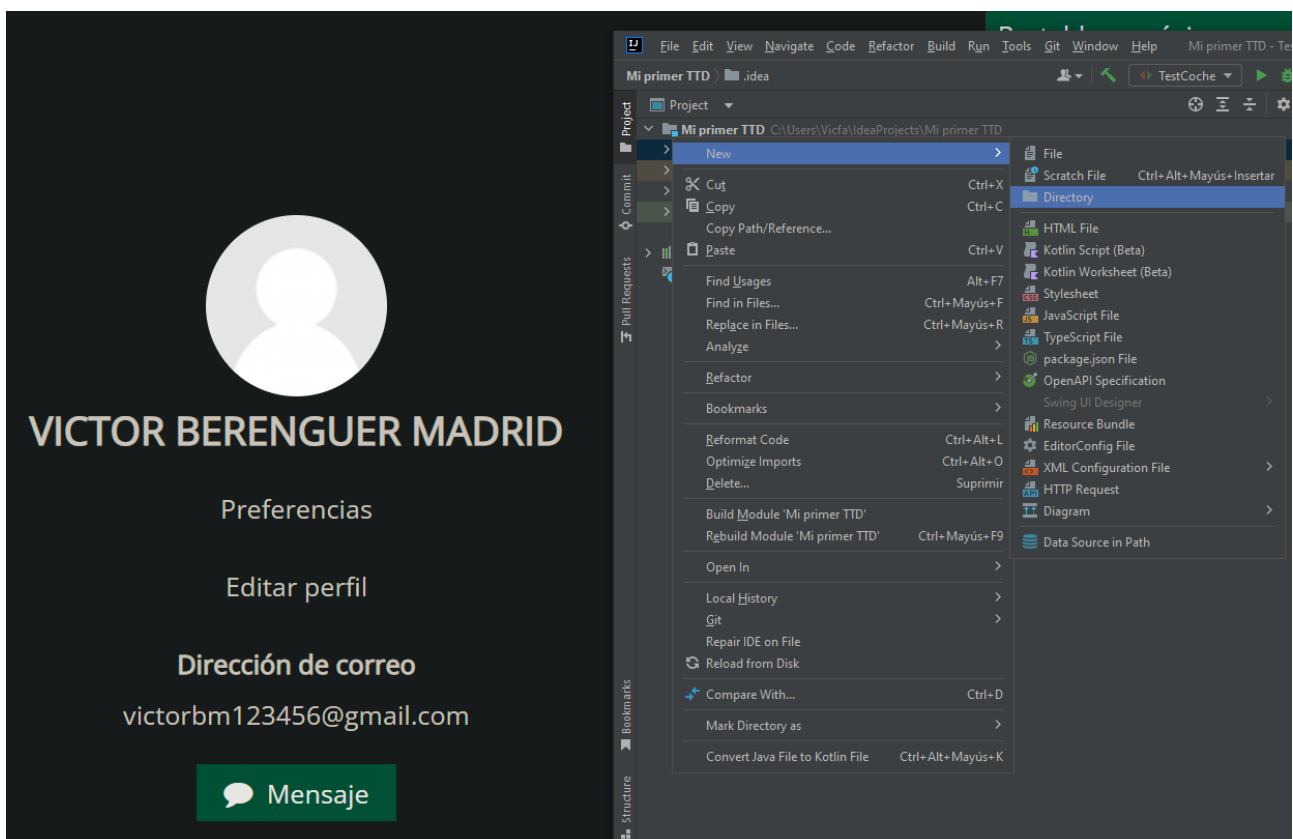
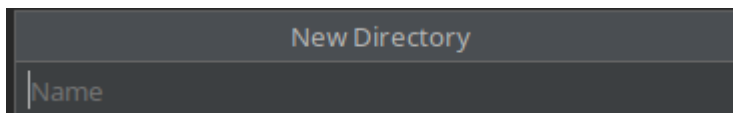


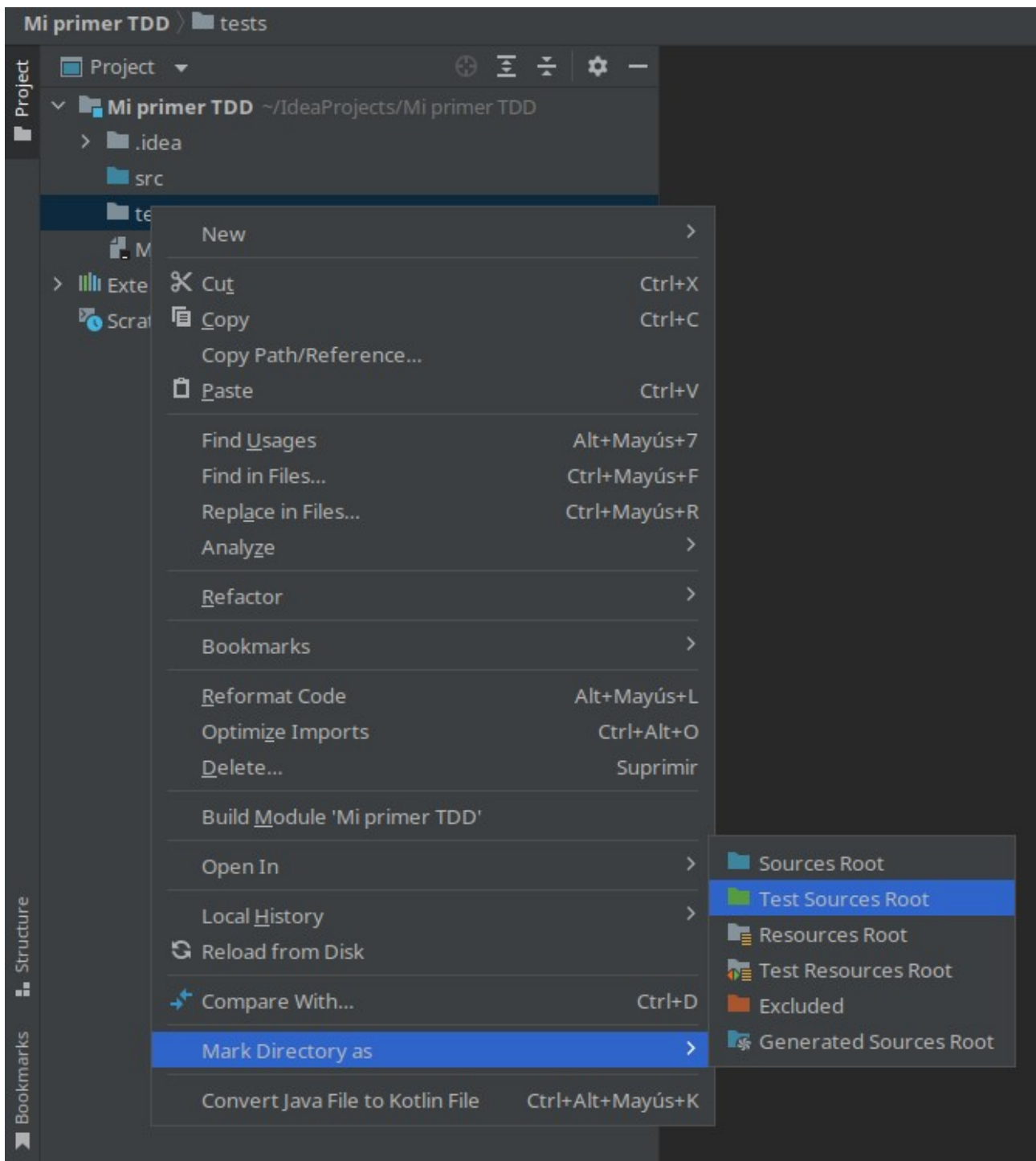
Comenzamos con la creación de un nuevo proyecto de Java, el cual, obviamente requerirá de un nombre.



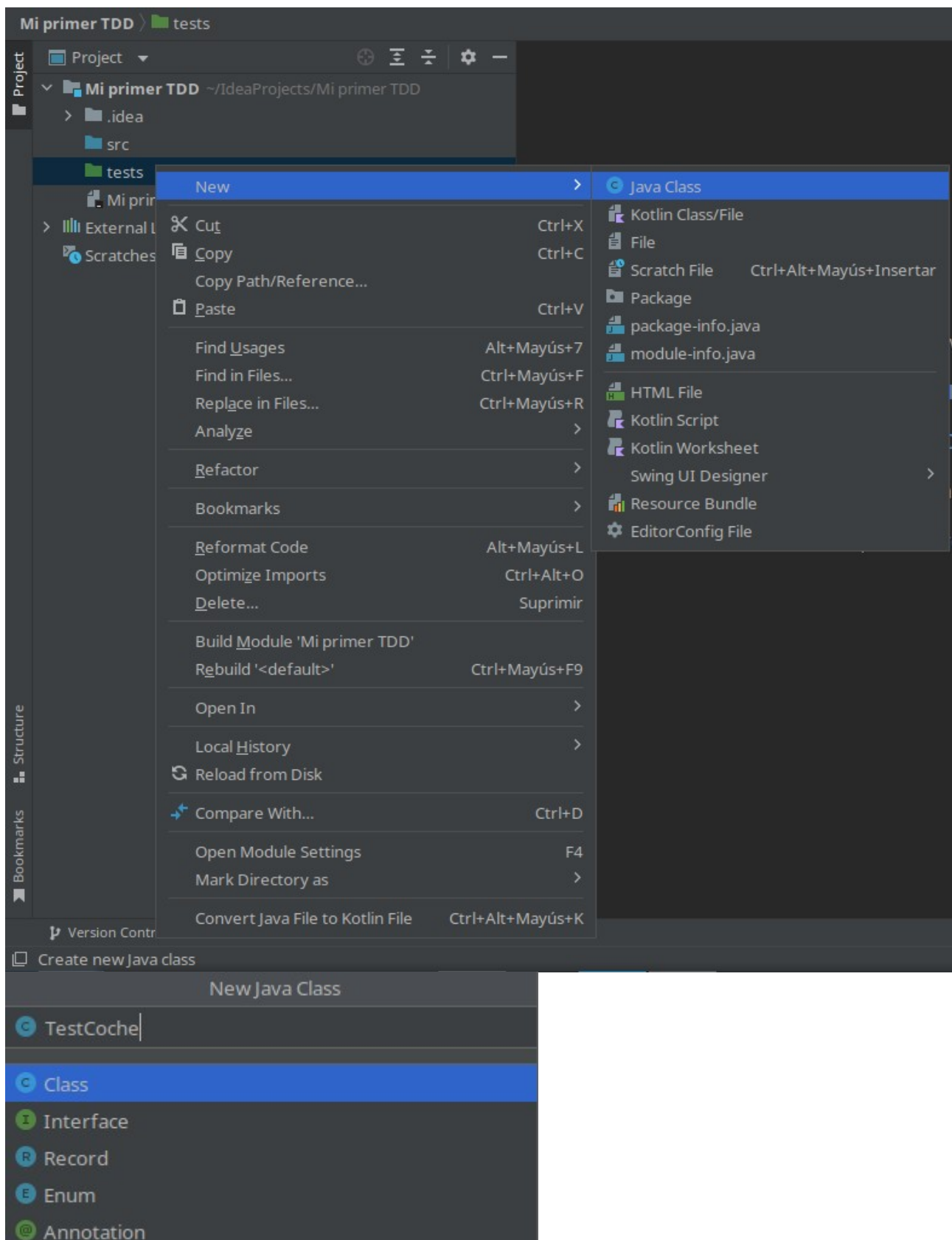
Una vez creado, en el menú del proyecto, crearemos un nuevo directorio en la carpeta raíz de dicho proyecto, haciendo click derecho > New > Directory sobre el mismo.



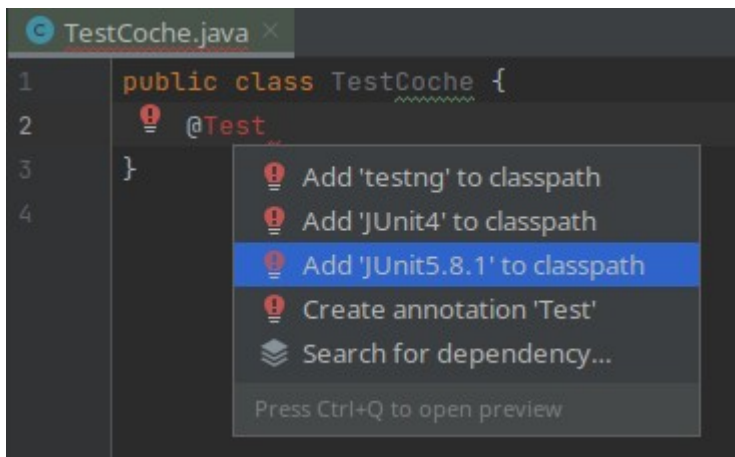
Aparecerá un pequeño recuadro en el que rellenar el nombre que tendrá el nuevo directorio, por ejemplo, tests.



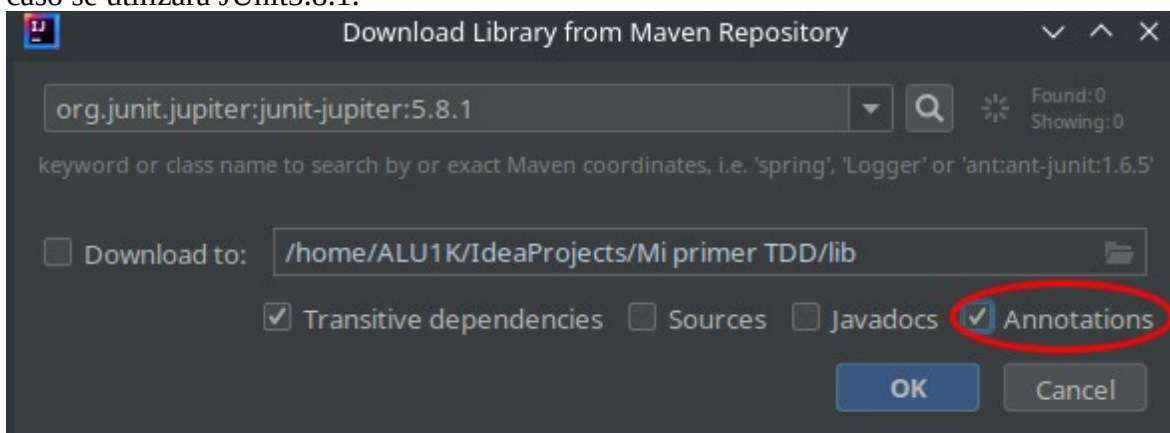
A continuación, haciendo click derecho en el nuevo directorio, lo marcaremos como directorio de tests con Mark Directory as > Test Sources Root.



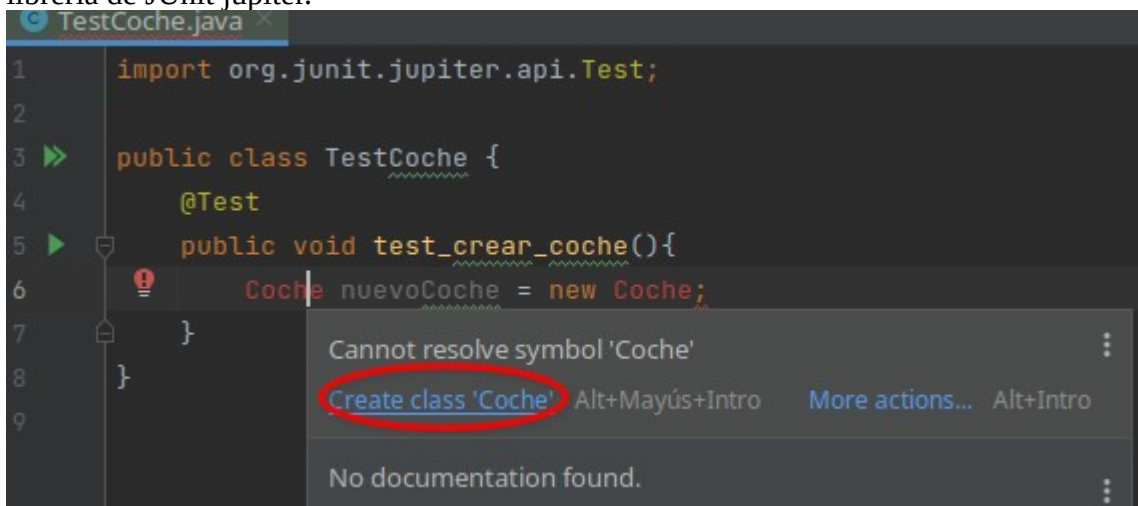
Dentro del directorio, crearemos una clase de Java y le llamaremos, en este caso, TestCoche. En esta clase, se crearán las clases para, posteriormente, añadir las clases.



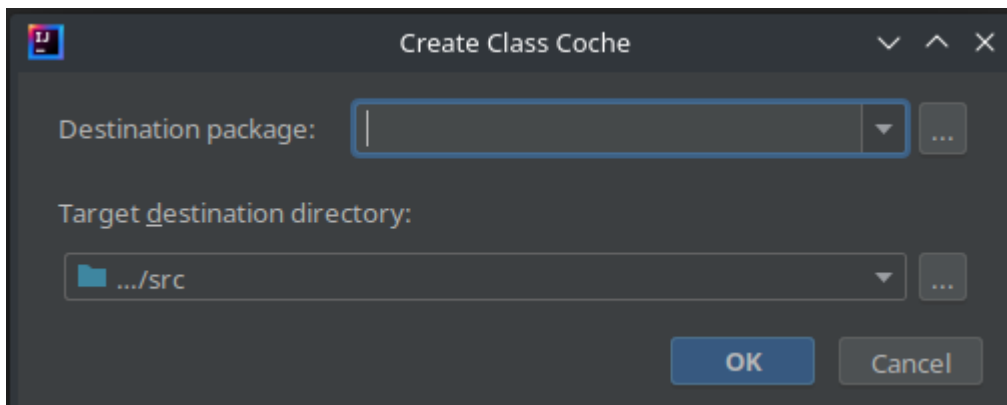
Una vez creada, dentro de la clase escribiremos @Test para indicar que la siguiente instrucción será un test. Haciendo click derecho, se desplegarán las distintas versiones de JUnit disponibles, en este caso se utilizará JUnit5.8.1.



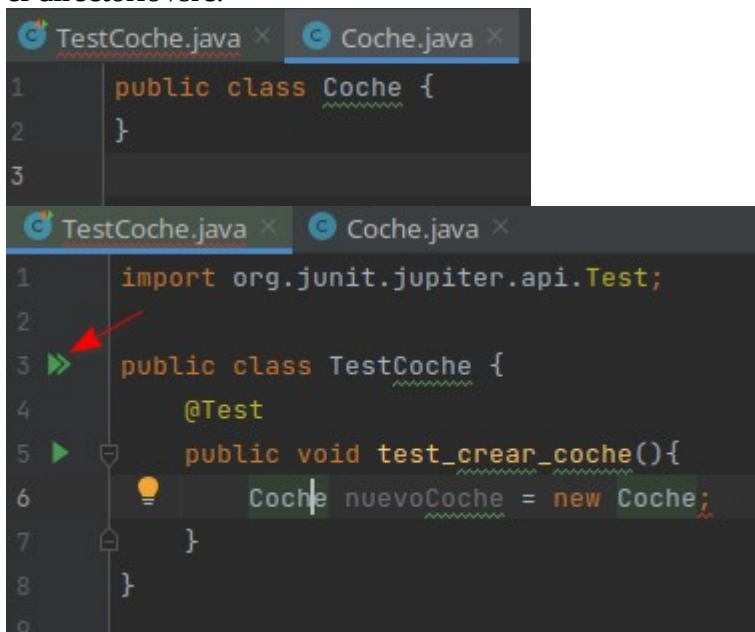
Aparecerá un recuadro en el que marcaremos las anotaciones y confirmaremos que descargamos la librería de JUnit jupiter.



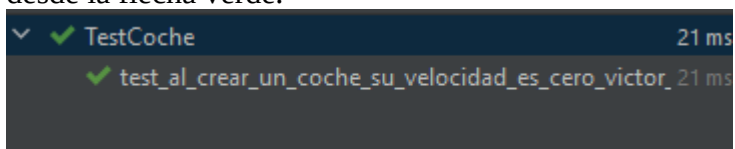
A continuación, creamos un primer método que sirva como test para crear un objeto de la clase Coche, la cual crearemos pasando el cursor por encima y haciendo click en el recuadro que aparecerá.



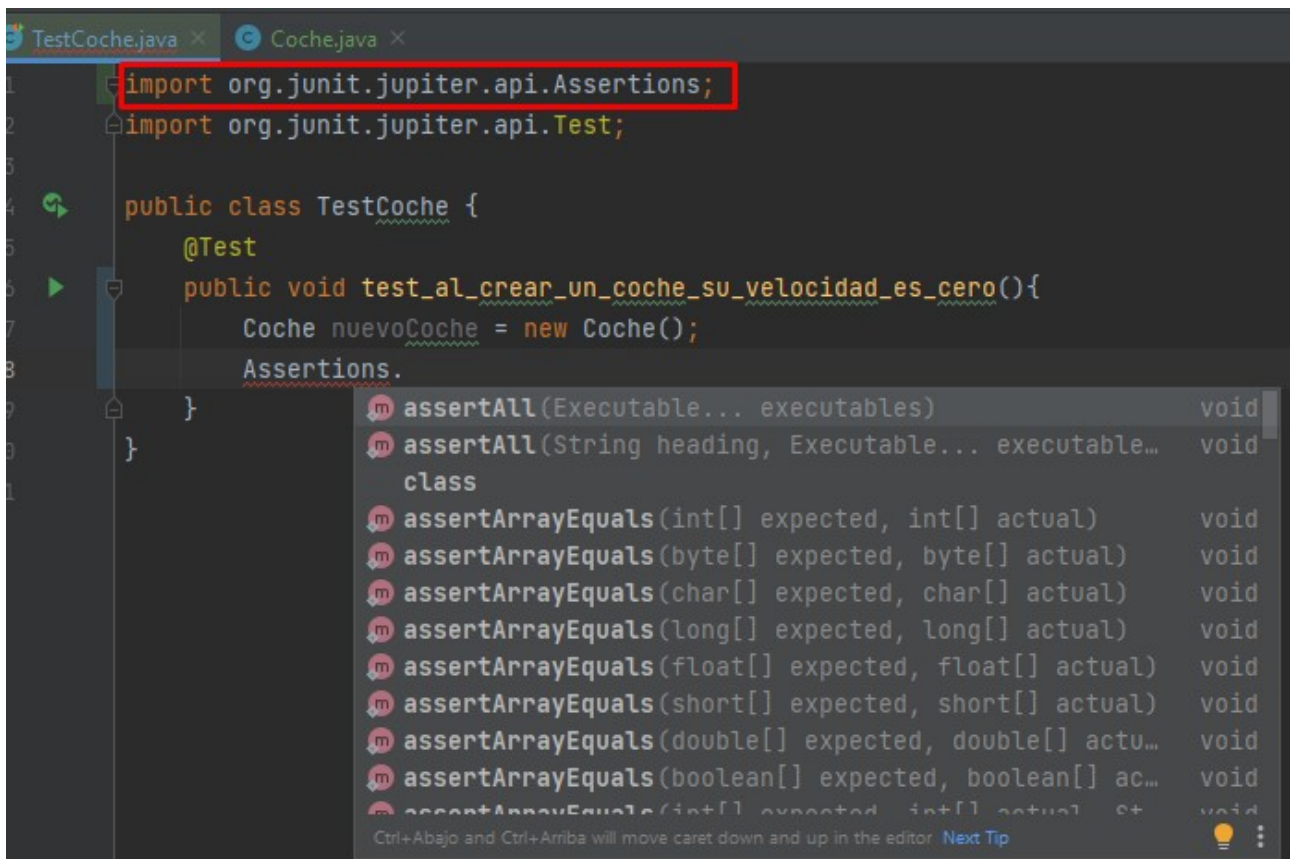
Nos preguntará cual será el destino que contendrá dicha clase, en este caso, indicaremos que será en el directorio /src.



Una vez creada, se abrirá otra pestaña en el IDE. Haremos un primer test iniciando el programa, desde la flecha verde.

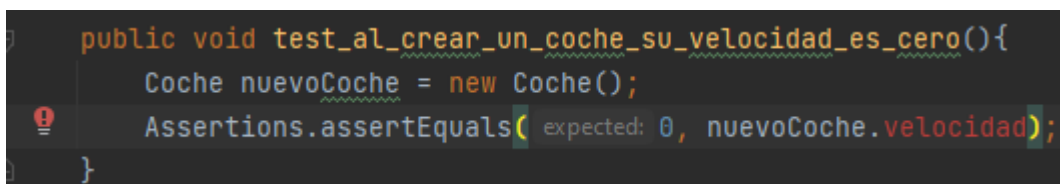


Vemos que, sin problema el programa pasa el test.



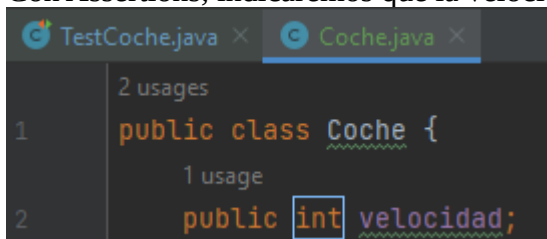
```
1 import org.junit.jupiter.api.Assertions;
2 import org.junit.jupiter.api.Test;
3
4 public class TestCoche {
5     @Test
6     public void test_al_crear_un_coche_su_velocidad_es_cero(){
7         Coche nuevoCoche = new Coche();
8         Assertions.
9     }
10 }
```

Cambiaremos este primer método de prueba y construiremos uno en el que, al crear un coche, la velocidad inicial sea 0. Por lo que, utilizaremos Assertions.



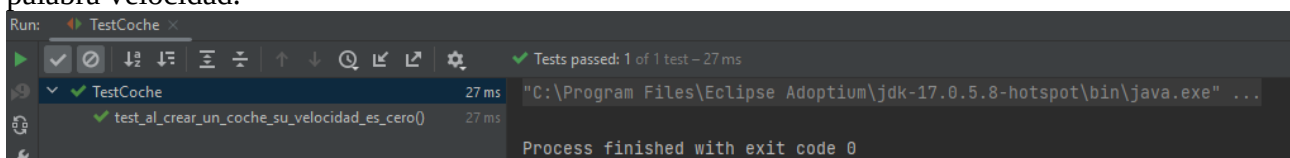
```
public void test_al_crear_un_coche_su_velocidad_es_cero(){
    Coche nuevoCoche = new Coche();
    Assertions.assertEquals(0, nuevoCoche.velocidad);
}
```

Con Assertions, indicaremos que la velocidad esperada al crear un coche, sea 0.



```
1 public class Coche {
2     public int velocidad;
```

Dado que el atributo velocidad no está creado, podemos hacerlo haciendo click derecho sobre la palabra velocidad.



```
Run: TestCoche
Tests passed: 1 of 1 test - 27 ms
TestCoche
test_al_crear_un_coche_su_velocidad_es_cero() 27 ms
Process finished with exit code 0
```

Si volvemos a realizar un test, comprobaremos que lo pasa sin problemas.



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

Crearemos un nuevo método con el que aumente la velocidad del coche al acelerar. Dicho método contendrá las mismas instrucciones que el anterior y además, se añadirá el valor de aceleración y, también con Assertions, haremos que la velocidad esperada tras accionar el método sea 30.

```
public class Coche {
    3 usages
    public int velocidad;

    1 usage
    public void acelerar(int aceleracion) {
        velocidad += aceleracion;
    }
}
```

Creamos el método acelerar en la clase Coche, en el que se indicará que el valor de velocidad aumentará en aceleración.

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

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

Run: TestCoche x

Test Case	Duration	Status
TestCoche	24 ms	✓
test_al_crear_un_coche_su_velocidad	22 ms	✓
test_al_acelerar_un_coche_su_velocidad	1 ms	✓
test_al_decelerar_un_coche_su_velocidad	1 ms	✓

También crearemos el método de decelerar. Si probamos de nuevo el test, vemos que lo vuelve a pasar sin problemas.

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

TestCoche x

Tests failed: 1, passed: 3 of 4 tests - 34 ms

TestCoche 34 ms

- test\_al\_decelerar\_un\_coche\_su\_velocidad\_no\_puede\_ser\_menor\_que\_cero 28 ms
- test\_al\_acelerar\_un\_coche\_su\_velocidad\_aumenta() 4 ms
- test\_al\_crear\_un\_coche\_su\_velocidad\_es\_cero() 1 ms
- test\_al\_decelerar\_un\_coche\_su\_velocidad\_disminuye() 1 ms

org.opentest4j.AssertionFailedError:

Expected :0

Actual :-30

[Click to see difference](#)

<5 internal lines>

at TestCoche.test\_al\_decelerar\_un\_coche\_su\_velocidad\_no\_puede\_ser\_menor\_que\_cero(TestCoche.java:28) <31 internal lines>

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <9 internal lines>

at java.base/java.util.ArrayList.forEach(ArrayList.java:1511) <27 internal lines>

Para finalizar, crearemos un método para que el programa no permita una velocidad negativa. Si indicamos una velocidad inicial de 50, al decelerar le restamos un valor de 80 y que el valor de velocidad esperada sea 0, en caso de probar el test, nos indicará que ha habido un fallo en este último método, ya que la velocidad que posee es -30 cuando habíamos indicado que la velocidad esperada sería 0.

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

Arreglaremos esto simplemente añadiendo al método decelerar de la clase Coche un condicional para que cuando la velocidad sea menor que 0, el valor sea automáticamente 0.

TestCoche x

Tests passed: 4 of 4 tests - 22 ms

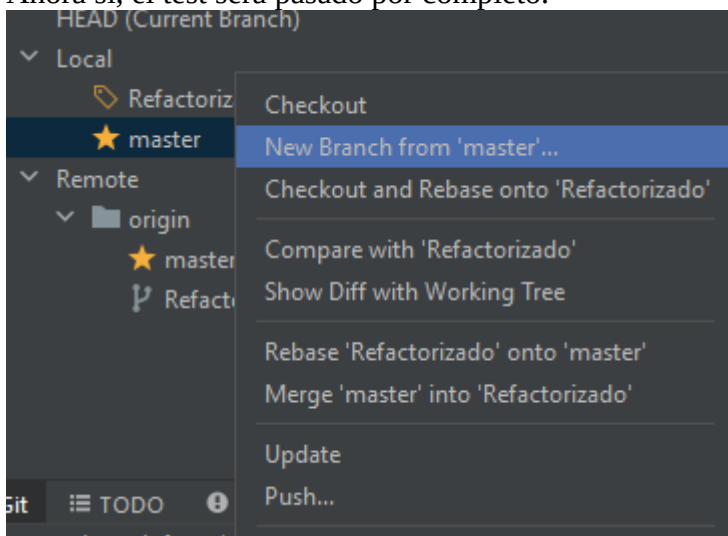
TestCoche 22 ms

- test\_al\_decelerar\_un\_coche\_su\_velocidad\_no\_puede\_ser\_menor\_que\_cero 20 ms
- test\_al\_acelerar\_un\_coche\_su\_velocidad\_aumenta() 4 ms
- test\_al\_crear\_un\_coche\_su\_velocidad\_es\_cero() 1 ms
- test\_al\_decelerar\_un\_coche\_su\_velocidad\_disminuye() 1 ms

"C:\Program Files\Eclipse Adoptium\jdk-17.0.5.8-hotspot\bin\java.exe" ...

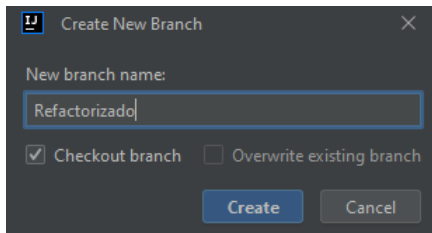
Process finished with exit code 0

Ahora sí, el test será pasado por completo.

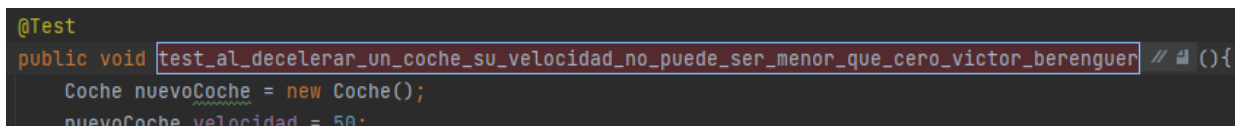
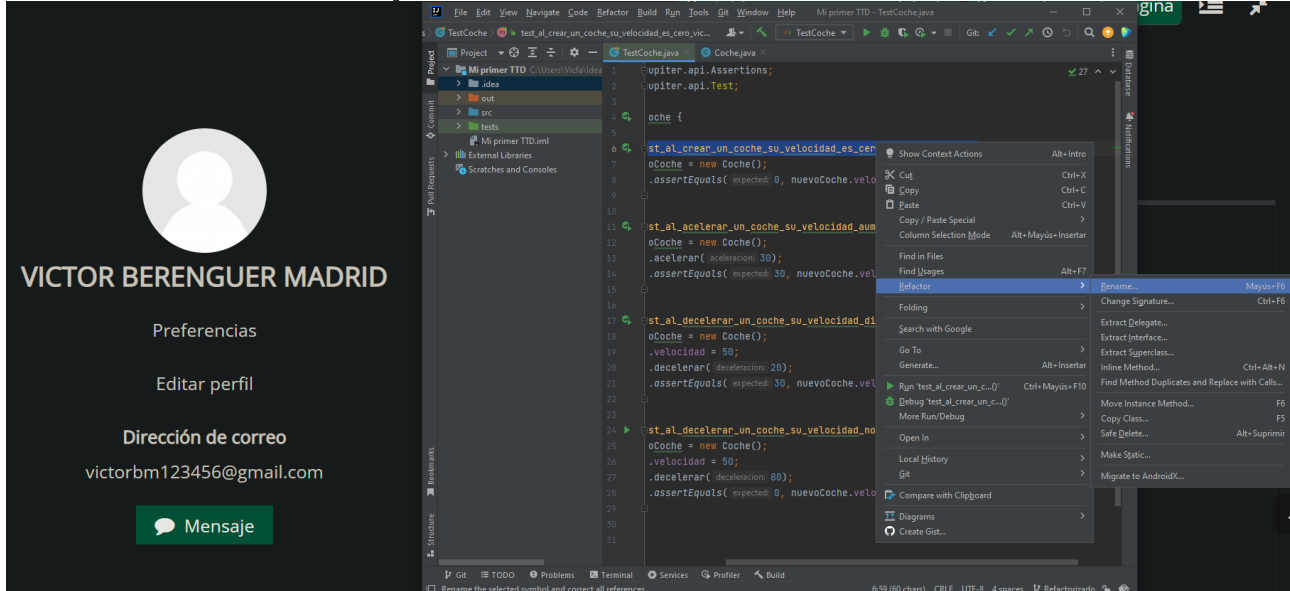


En la interfaz de Git en IntelliJ, crearemos una nueva rama con click derecho sobre la master.





La llamaremos Refactorizado y nos cambiaremos a ella.



A continuación, refactorizaremos todos los métodos renombrándolos con click derecho sobre el método > Refactor > Rename.