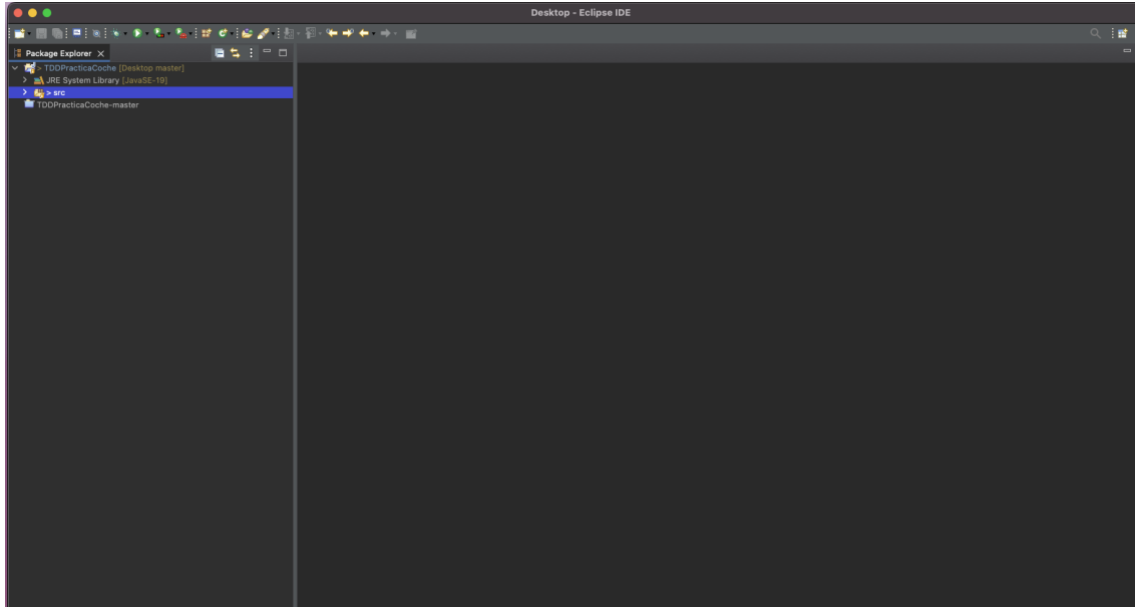
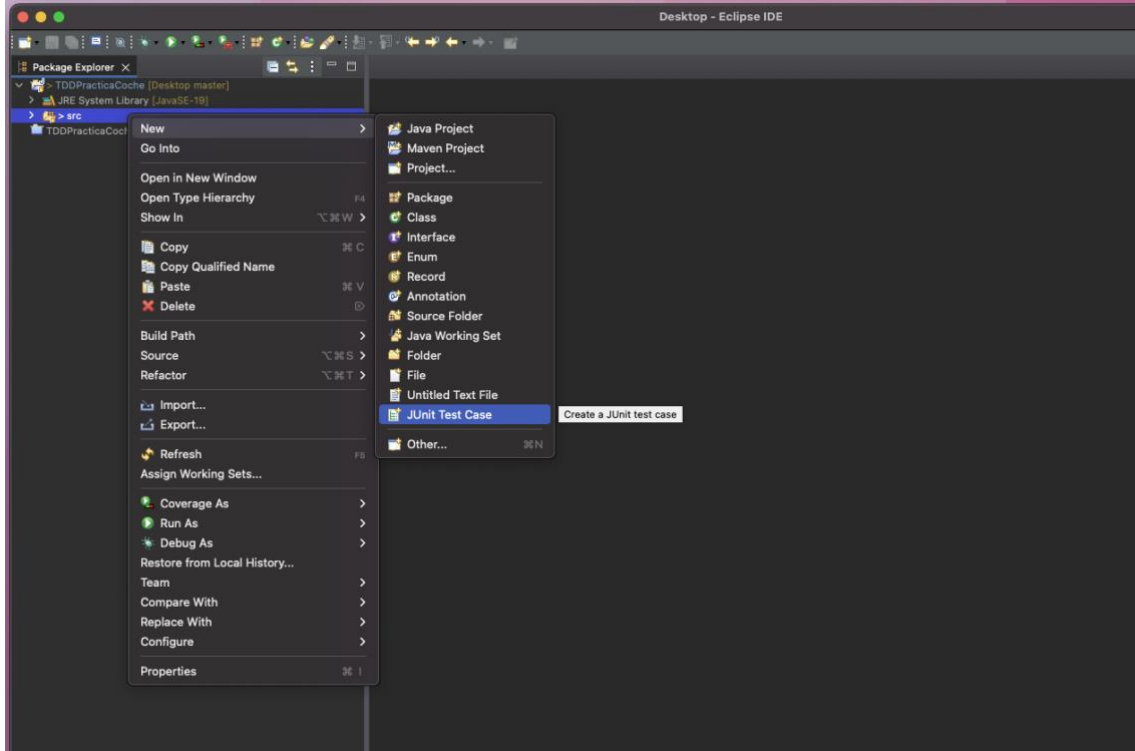


Mi primer TDD 2

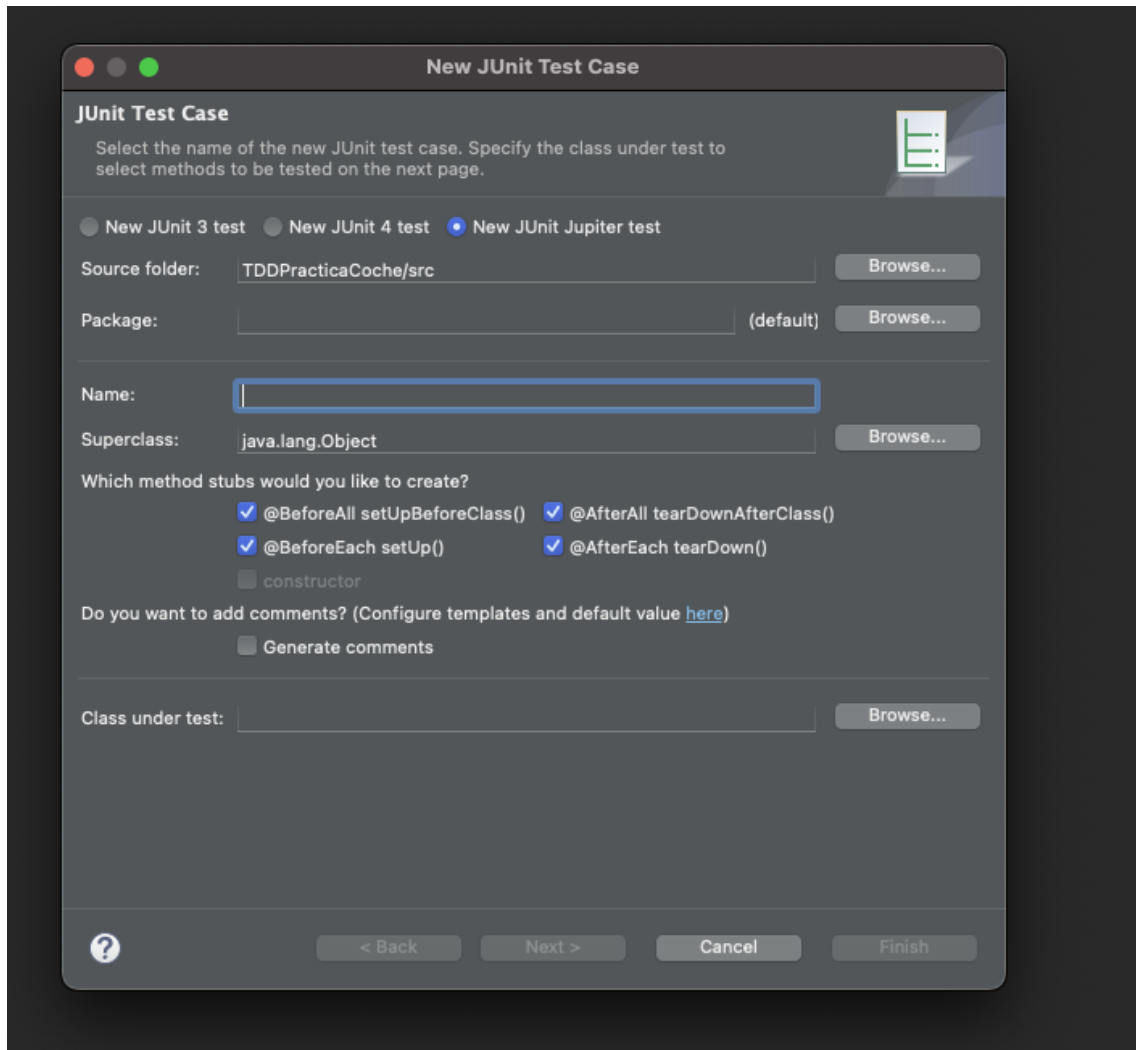
En primer lugar, nos dirigimos a Eclipse y creamos un nuevo proyecto llamado TDDPracticaCoche.



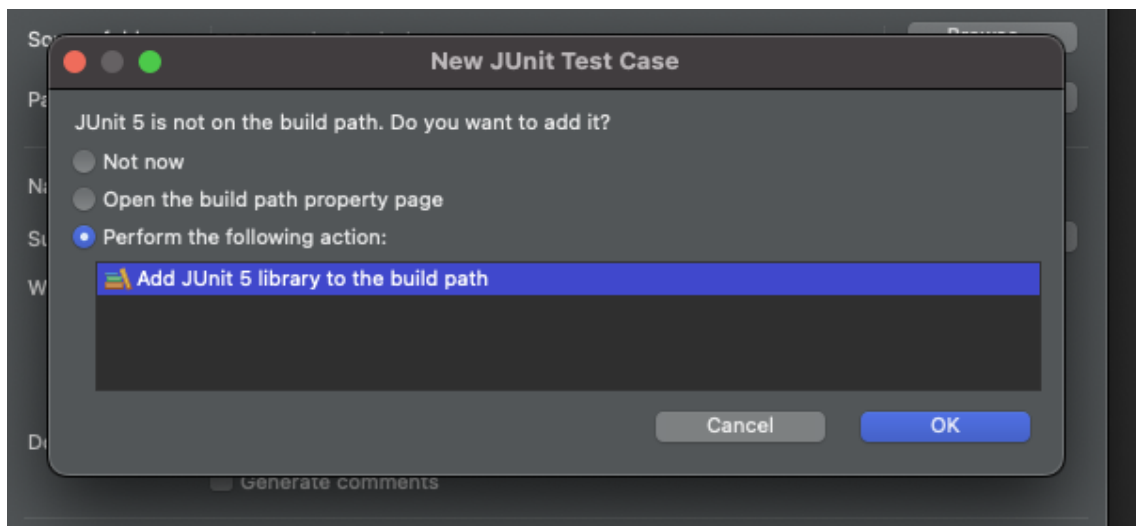
Ahora, hacemos click derecho, "new" y seleccionamos la opción JUnit Test Case.



Una vez le demos nos aparecerá esta ventana en la que debemos darle un nombre al test y seleccionar la versión de Junit que tengamos instalada.

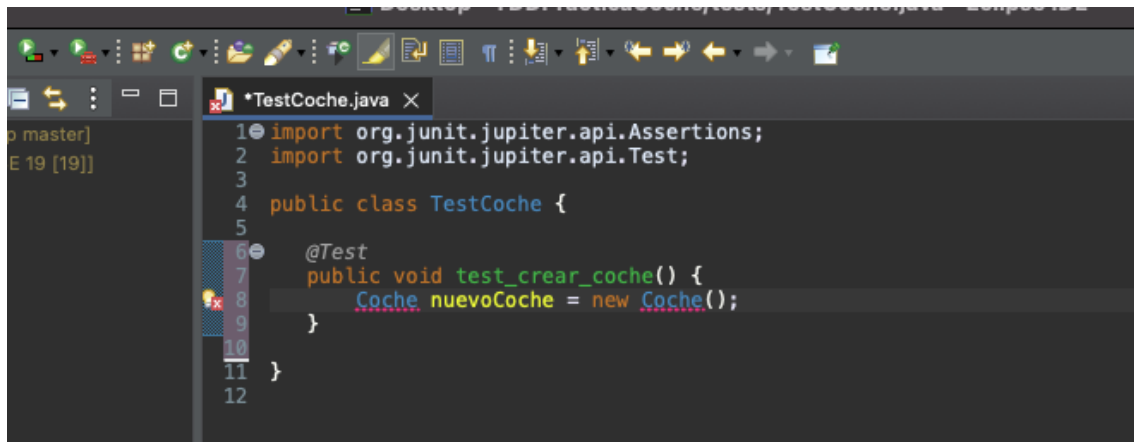


Hecho esto, no aparece una nueva ventana que nos pedirá añadir la versión de Junit seleccionada a la librería. Pulsamos “ok” y ya tendremos la clase test creada.

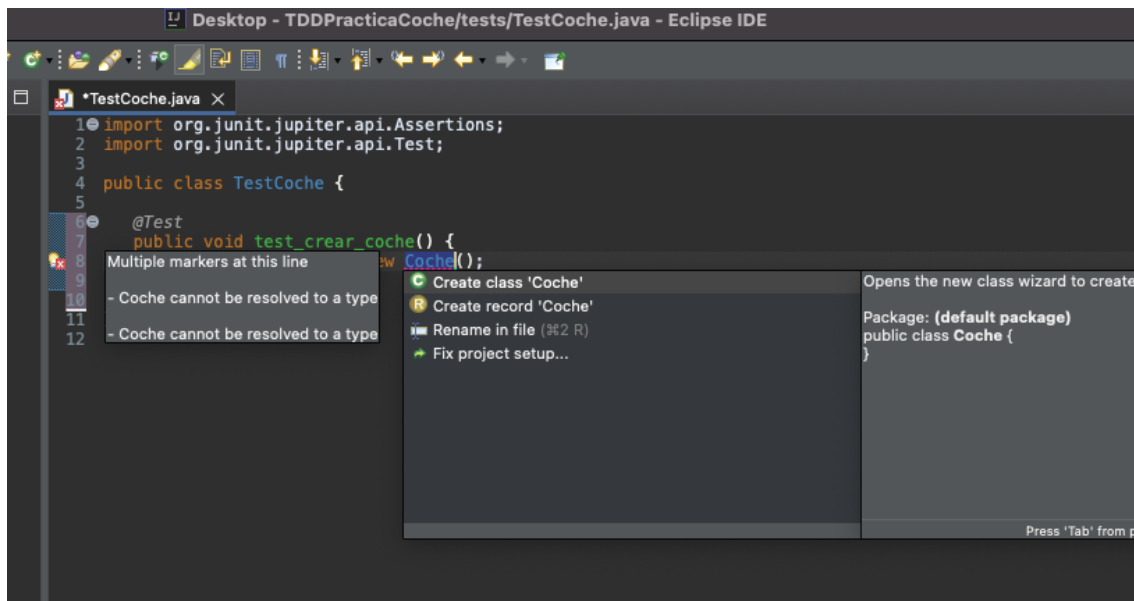


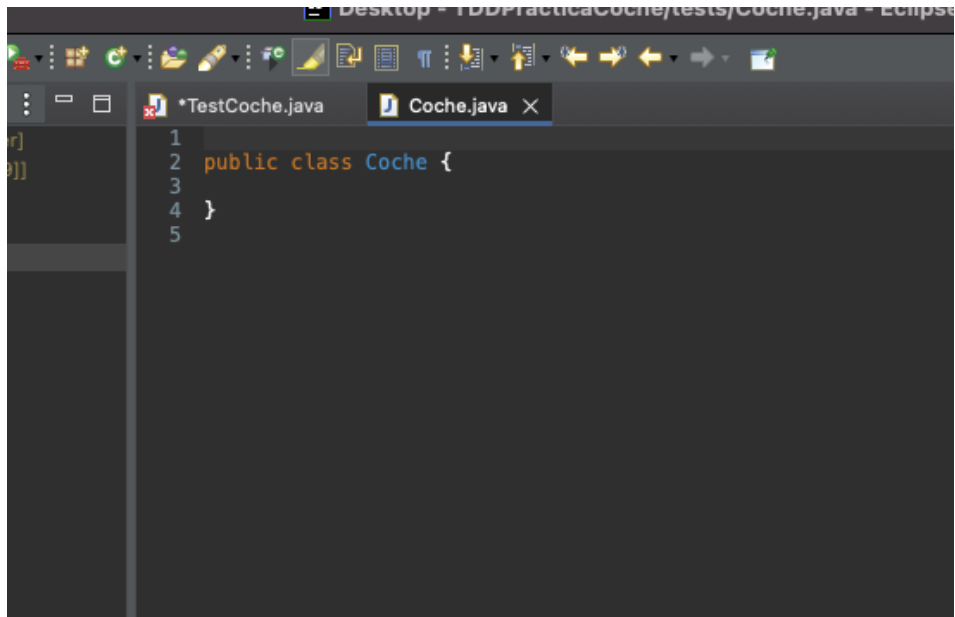
Realizado esto, comenzaremos a implementar los métodos del test.

En primer lugar, implementaremos un método para crear un coche.

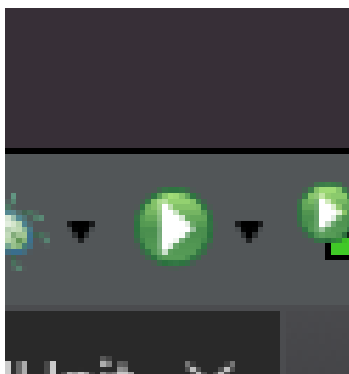


Como vemos, la clase `Coche` aparece subrayada en rojo. Eso es debido a que todavía no existe y debemos crearla. Para ello, nos posicionamos sobre la bombilla que aparece al lado del número de línea y el programa nos mostrará diferentes opciones, entre ellas crear la clase.

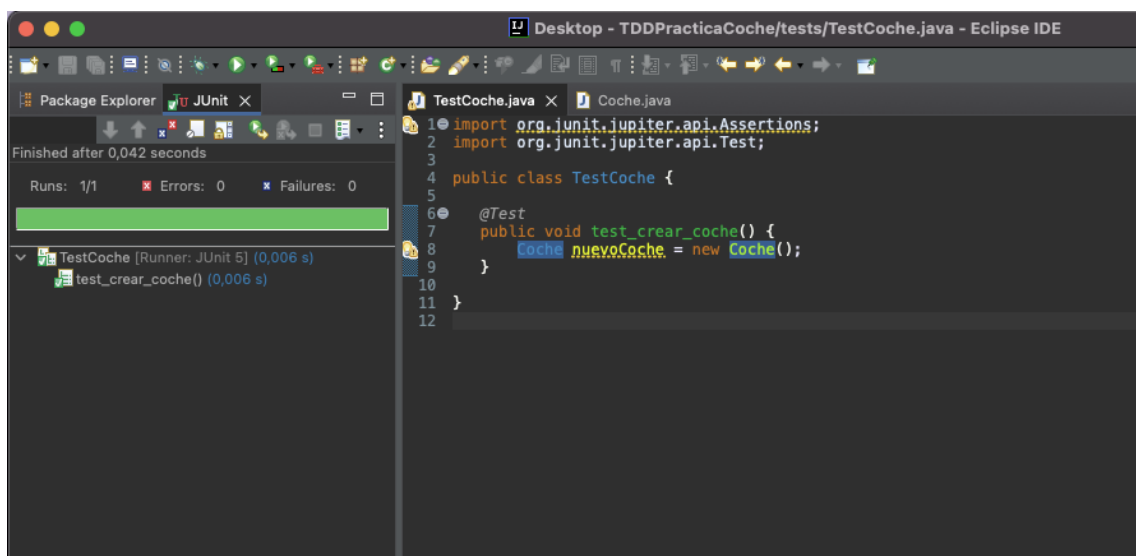




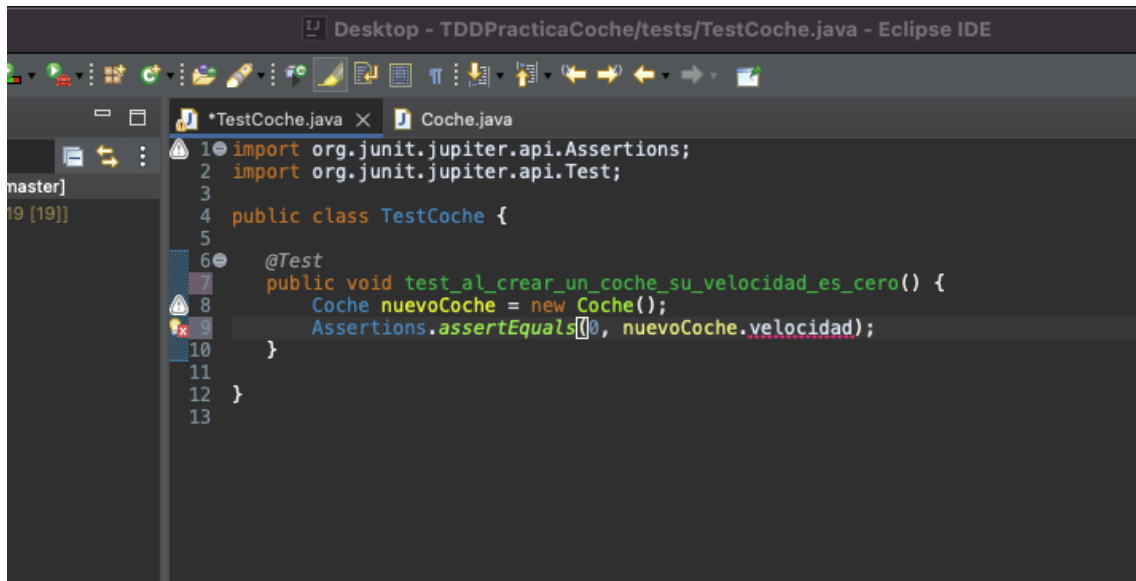
Creada la clase ejecutamos el test. Para ejecutarlo, debemos hacer click en la siguiente opción:



Ejecutamos el test y vemos como se completa satisfactoriamente.



Ahora vamos a mejorar un poco el test haciendo que no sea tan simple utilizando una aserción.

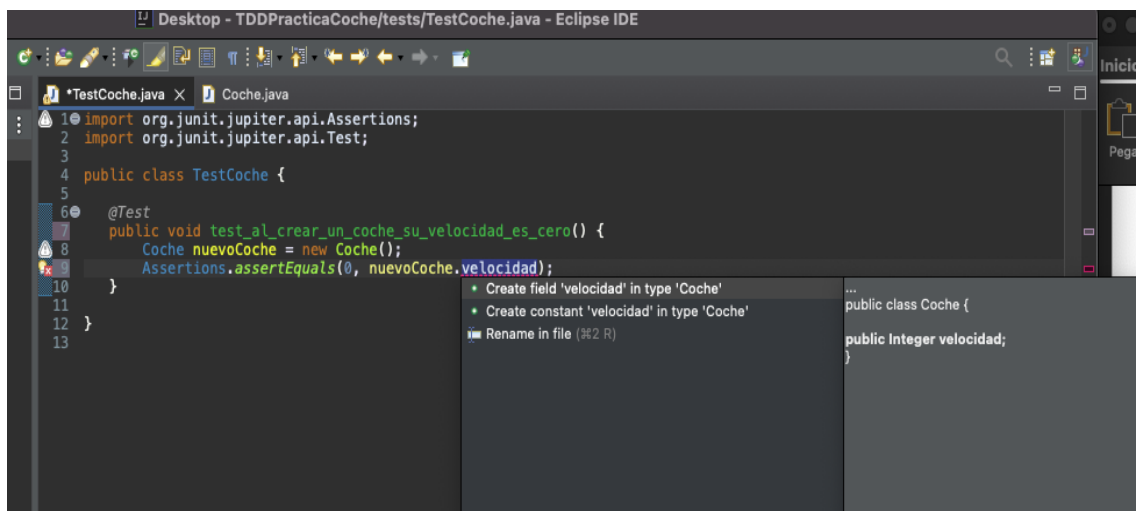


The screenshot shows the Eclipse IDE with two tabs: `*TestCoche.java` and `Coche.java`. The `TestCoche.java` file contains the following code:

```
1 import org.junit.jupiter.api.Assertions;
2 import org.junit.jupiter.api.Test;
3
4 public class TestCoche {
5
6     @Test
7     public void test_al_crear_un_coche_su_velocidad_es_cero() {
8         Coche nuevoCoche = new Coche();
9         Assertions.assertEquals(0, nuevoCoche.velocidad);
10    }
11
12 }
13
```

Hemos modificado el método estableciendo que al crear un coche su velocidad sea cero y hemos utilizado la aserción de manera que el valor que esperamos sea cero y el valor real sea el que se le pasa. Además, como vemos, velocidad aparece en rojo y es debido a que no hemos creado dicho atributo en la clase Coche.

De igual manera que anteriormente con la clase Coche, creamos el atributo velocidad.

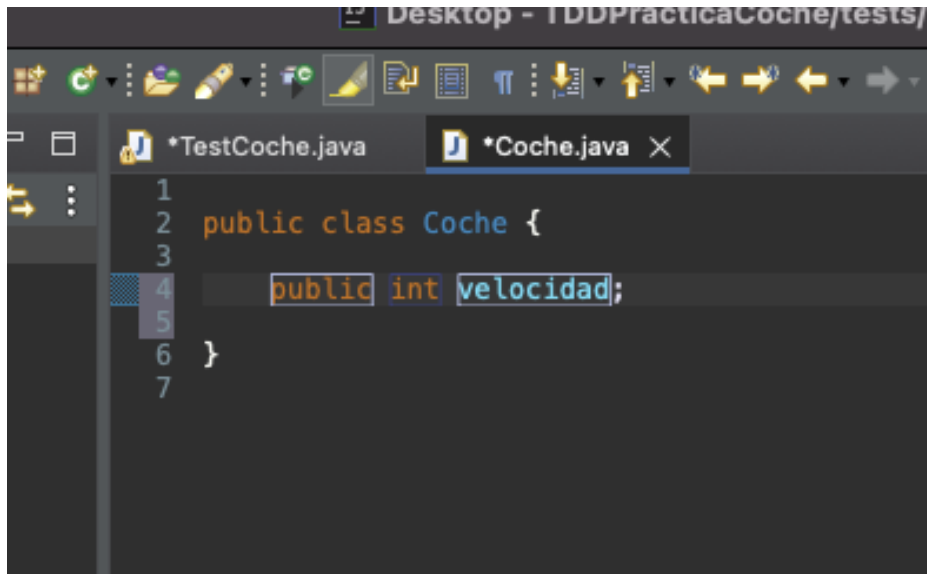


The screenshot shows the Eclipse IDE with two tabs: `*TestCoche.java` and `Coche.java`. The `TestCoche.java` file contains the same code as in the previous screenshot. The `Coche.java` file is currently empty. A context menu is open over the `Coche.java` tab, showing options to create a field or constant named 'velocidad' in the 'Coche' type. The menu options are:

- Create field 'velocidad' in type 'Coche'
- Create constant 'velocidad' in type 'Coche'
- Rename in file (⌘2 R)

The `Coche.java` file content is:

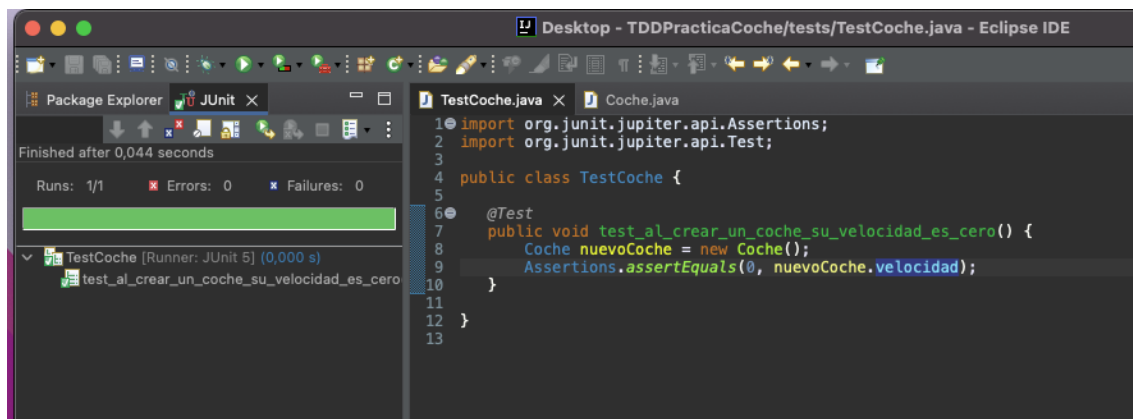
```
public class Coche {
    public Integer velocidad;
}
```



The screenshot shows the Eclipse IDE with the file `Coche.java` open. The code defines a public class `Coche` with a public integer attribute `velocidad`.

```
1  
2 public class Coche {  
3  
4     public int velocidad;  
5  
6 }  
7
```

Hecho esto, ejecutamos el test y observamos como lo hemos pasado correctamente.

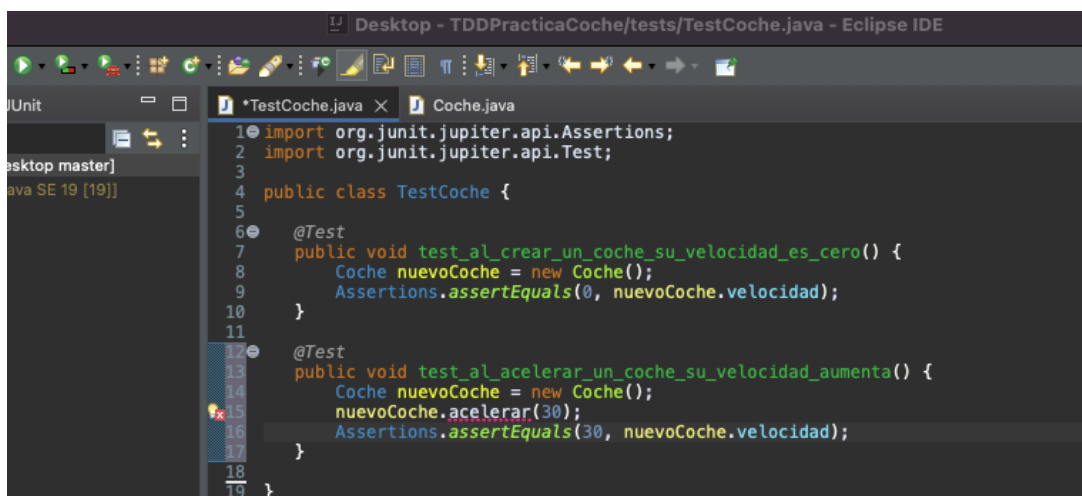


The screenshot shows the Eclipse IDE with the file `TestCoche.java` open. The code defines a public class `TestCoche` with a JUnit test method `test_al_crear_un_coche_su_velocidad_es_cero()` that creates a new `Coche` object and asserts that its `velocidad` attribute is equal to 0.

```
1 import org.junit.jupiter.api.Assertions;  
2 import org.junit.jupiter.api.Test;  
3  
4 public class TestCoche {  
5  
6     @Test  
7     public void test_al_crear_un_coche_su_velocidad_es_cero() {  
8         Coche nuevoCoche = new Coche();  
9         Assertions.assertEquals(0, nuevoCoche.velocidad);  
10    }  
11 }  
12  
13
```

The JUnit test results are shown in the Package Explorer, indicating that the test passed successfully.

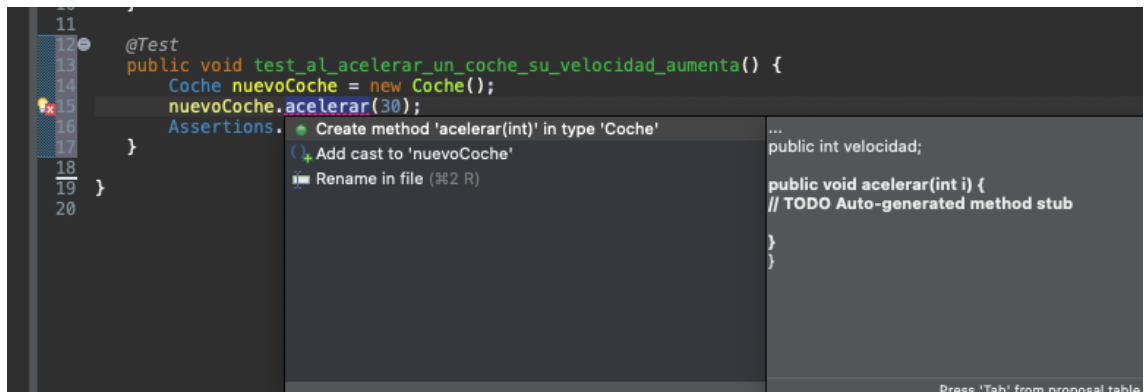
Una vez hecho esto, vamos a implementar un método para acelerar la velocidad del coche.



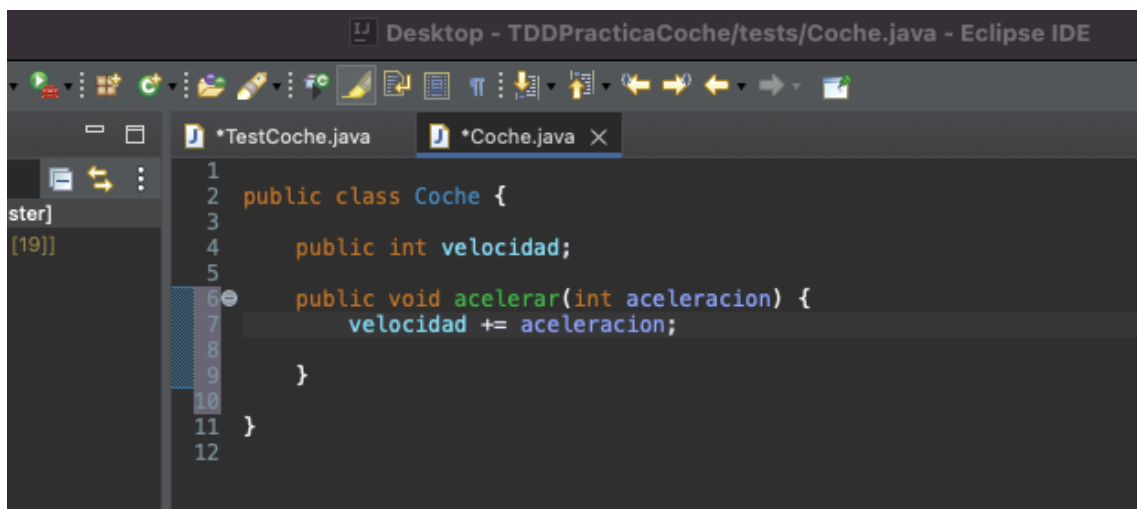
The screenshot shows the Eclipse IDE with the file `TestCoche.java` open. The code defines a public class `TestCoche` with two JUnit test methods. The first method, `test_al_crear_un_coche_su_velocidad_es_cero()`, creates a new `Coche` object and asserts that its `velocidad` attribute is equal to 0. The second method, `test_al_acelerar_un_coche_su_velocidad_aumenta()`, creates a new `Coche` object, calls the `acelerar(30)` method, and asserts that its `velocidad` attribute is equal to 30.

```
1 import org.junit.jupiter.api.Assertions;  
2 import org.junit.jupiter.api.Test;  
3  
4 public class TestCoche {  
5  
6     @Test  
7     public void test_al_crear_un_coche_su_velocidad_es_cero() {  
8         Coche nuevoCoche = new Coche();  
9         Assertions.assertEquals(0, nuevoCoche.velocidad);  
10    }  
11  
12     @Test  
13     public void test_al_acelerar_un_coche_su_velocidad_aumenta() {  
14         Coche nuevoCoche = new Coche();  
15         nuevoCoche.acelerar(30);  
16         Assertions.assertEquals(30, nuevoCoche.velocidad);  
17    }  
18 }  
19
```

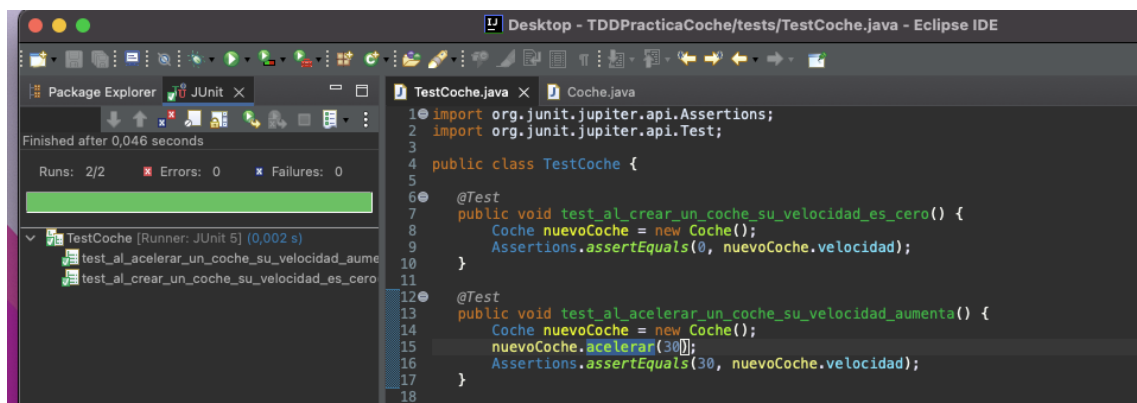
Hemos creado el método para acelerar la velocidad del coche, pero como vemos acelerar aparece en rojo lo que significa que no existe y debemos crear el método en la clase Coche, además, le hemos dicho que su velocidad es de 30.



Creamos el método y le decimos que velocidad aumente en aceleración.



Ejecutamos y vemos como hemos pasado el test correctamente.



Después del método acelerar procedemos a implementar otro método para decelerar.

```
18
19 @Test
20 public void test_al_decelerar_un_coche_su_velocidad_disminuye() {
21     Coche nuevoCoche = new Coche();
22     nuevoCoche.velocidad = 50;
23     nuevoCoche.decelerar(20);
24     Assertions.assertEquals(30, nuevoCoche.velocidad);
25 }
26
27 }
28
```

Hemos creado el método, pero si lo dejamos así no funcionaría puesto que cuando creamos el coche este parte de una velocidad de 0 y aquí le hemos establecido 30, por lo que vamos a establecerle una velocidad mayor para que vaya decelerando. Además, debemos crear el método decelerar.

```
19 @Test
20 public void test_al_decelerar_un_coche_su_velocidad_disminuye() {
21     Coche nuevoCoche = new Coche();
22     nuevoCoche.velocidad = 50;
23     nuevoCoche.decelerar(20);
24     Assertions.
25 }
26
27 }
28
```

Change to 'acelerar(..)'
Create method 'decelerar(int)' in type 'Coche'
Add cast to 'nuevoCoche'
Rename in file (⌘2 R)

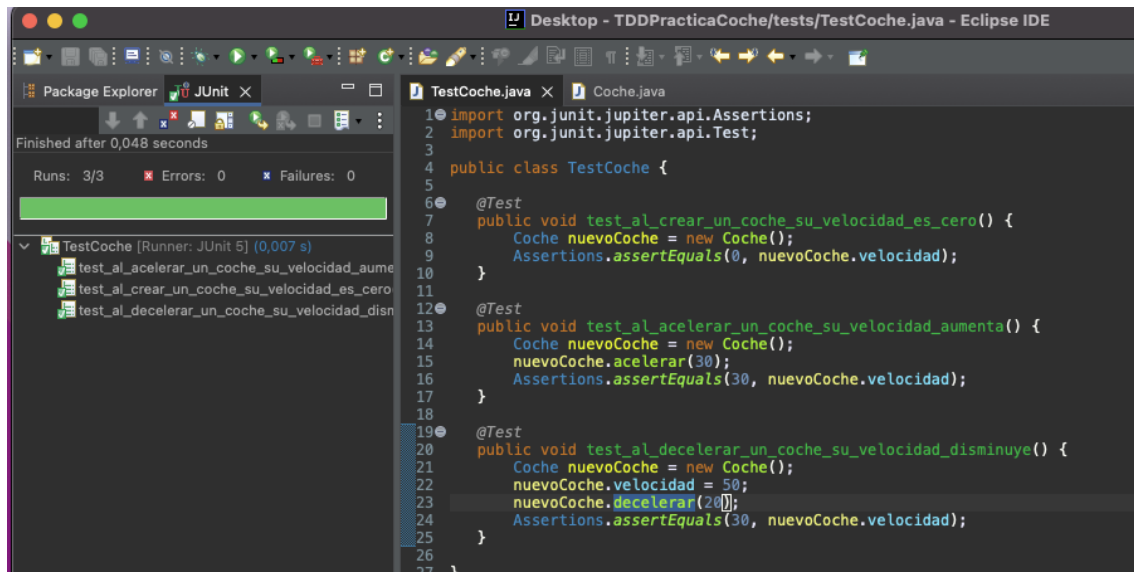
```
public void decelerar(int i) {
    // TODO Auto-generated method
}
}
```

Creamos el método y le decimos que la velocidad disminuya en la deceleración.

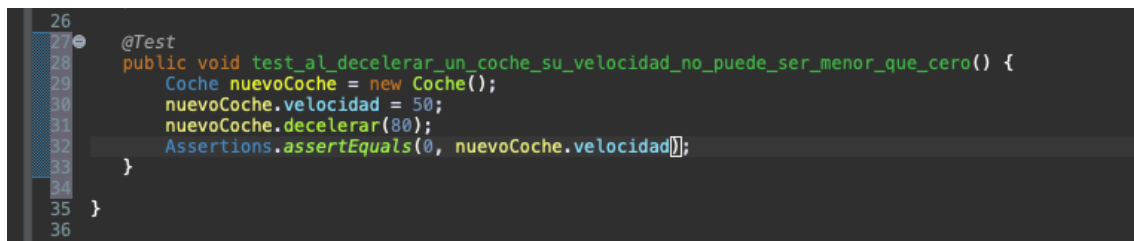
```
9 }
10
11 public void decelerar(int deceleracion) {
12     velocidad -= deceleracion;
13 }
14
15 }
16 }
17
```

Hecho esto, tenemos el método para disminuir la velocidad partiendo en 50 y teniendo que decelerar hasta 20 por lo que, el valor esperado es de 30.

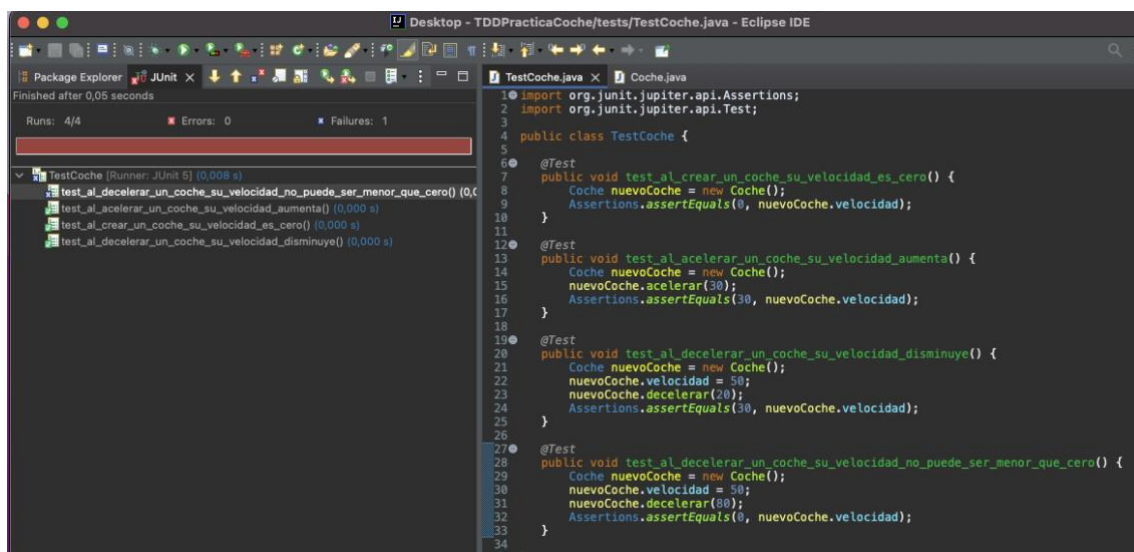
Ahora, ejecutamos el test y observamos como lo hemos pasado correctamente.

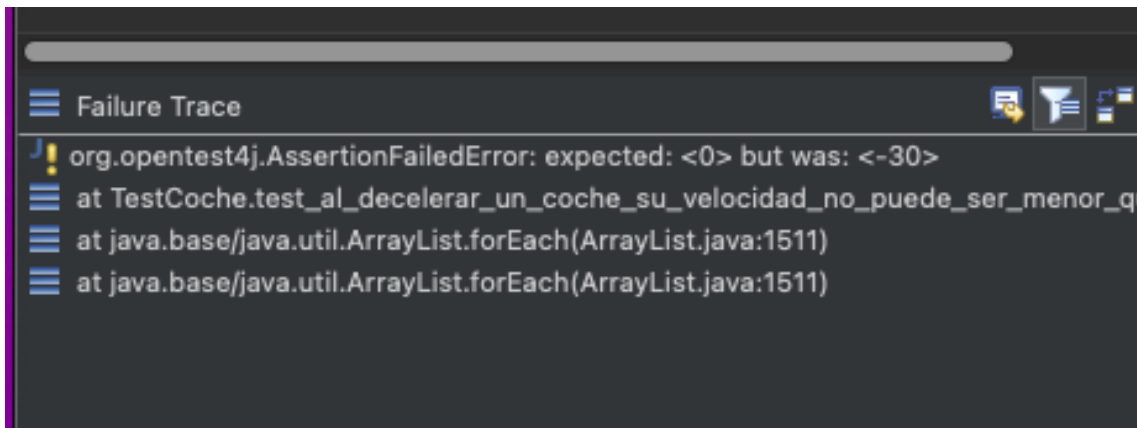


Por último, vamos a implementar un método para que la velocidad del coche no pueda ser menor que cero y así no obtener una velocidad negativa.

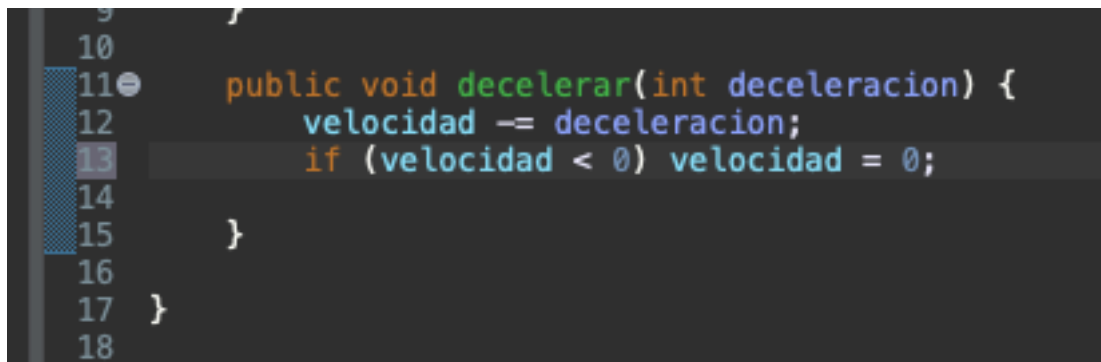


Hemos creado el método estableciendo que el coche lleva una velocidad de 50 y su deceleración en de 80 indicando que el valor esperado sea de 0 porque como hemos dicho el valor no puede ser menor que cero. Observamos que el método se puede compilar, pero falla debido a que esperaba una velocidad de 0 y su actual es de -30.

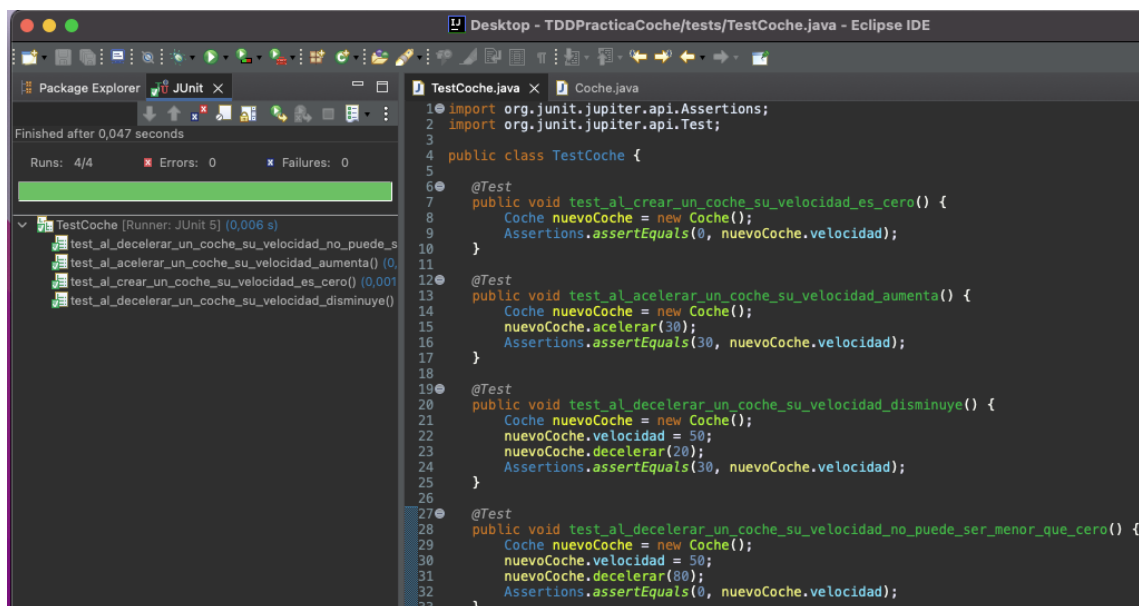




Para resolver esto, debemos irnos a la clase coche y en el método decelerar tendremos que indicar que la velocidad no pueda ser menor que cero.

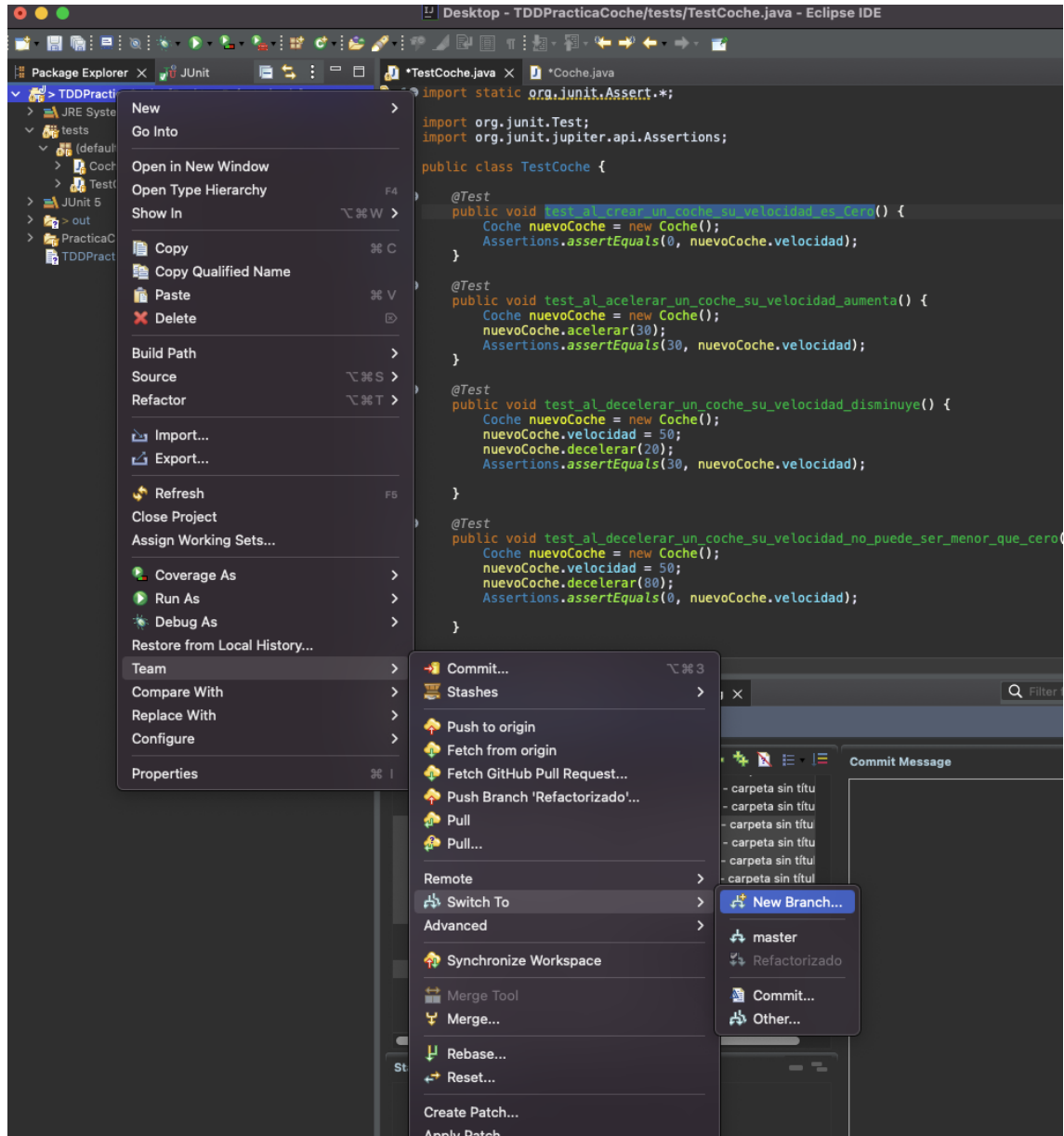


Hecho esto, ejecutamos de nuevo y vemos como ahora si hemos pasado el test satisfactoriamente.



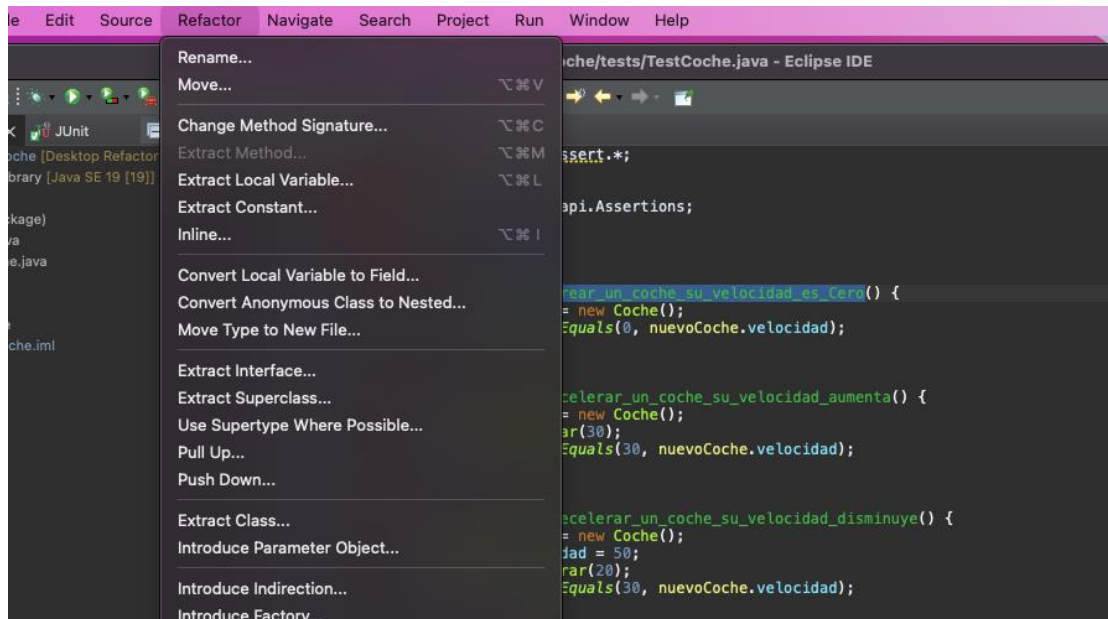
Realizados todos los test, procedemos a crear una nueva rama llamada Refactorizado en la que haremos una refactorización de todos los métodos añadiendo el nombre al final del método.

Para crear una nueva rama debemos hacer click derecho sobre el proyecto y dirigirnos a la opción “team”. Una vez ahí, vamos a “switch to – new branch” y creamos la nueva rama.

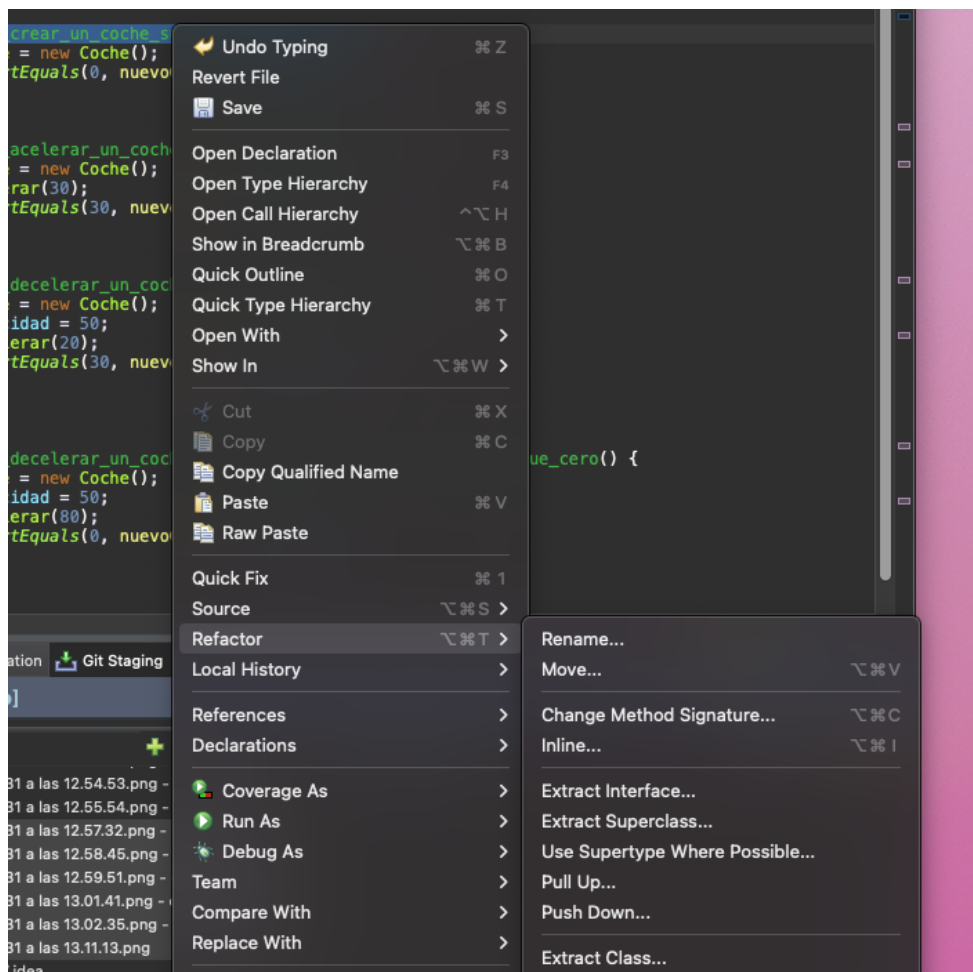


Ya en la nueva rama, para realizar la refactorización en el caso de macOS contamos con dos opciones. Una posibilidad es situándonos encima del método a refactorizar y

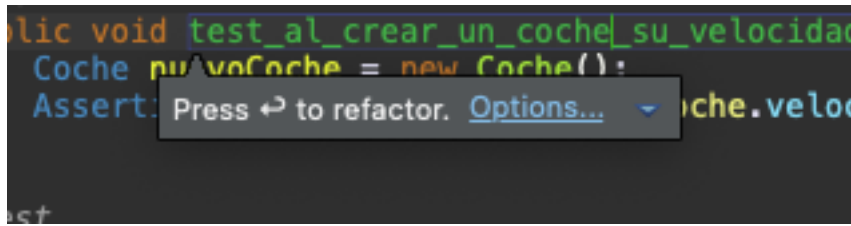
hacer click en la opción de refactor que nos aparece en la barra superior y elegir la opción “rename”.



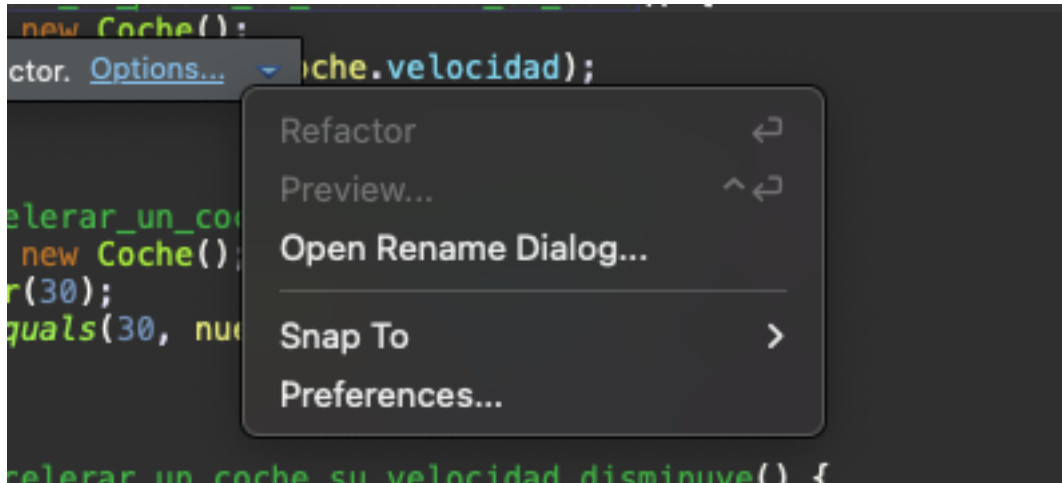
Por otro lado, también situándonos en el método en cuestión, hacemos click derecho y nos encontramos con una opción llamada “refactor – rename”.



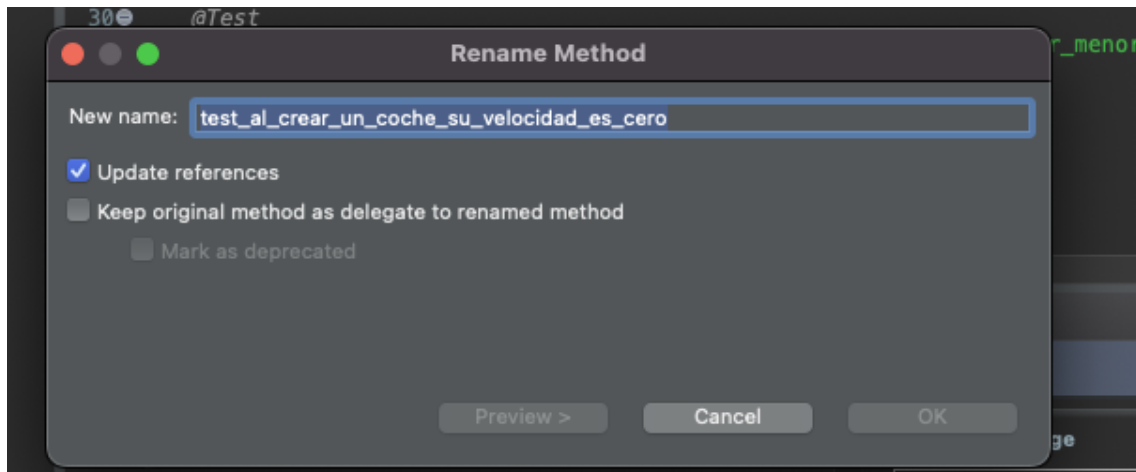
Cuando seleccionamos la opción “rename” nos aparece lo siguiente:



Pulsamos en el desplegable de “options” y obtenemos las siguientes opciones:



Le damos a “open rename dialog” y se nos abre una ventana para modificar el nombre del método.



```

5
6 public class TestCoche {
7
8     @Test
9     public void test_al_crear_un_coche_su_velocidad_es_cero_Jesus() {
10         Coche nuevoCoche = new Coche();
11         Assertions.assertEquals(0, nuevoCoche.velocidad);
12     }
13
14     @Test
15     public void test_al_acelerar_un_coche_su_velocidad_aumenta_Jesus() {
16         Coche nuevoCoche = new Coche();
17         nuevoCoche.acelerar(30);
18         Assertions.assertEquals(30, nuevoCoche.velocidad);
19     }
20
21     @Test
22     public void test_al_decelerar_un_coche_su_velocidad_disminuye_Jesus() {
23         Coche nuevoCoche = new Coche();
24         nuevoCoche.velocidad = 50;
25         nuevoCoche.decelerar(20);
26         Assertions.assertEquals(30, nuevoCoche.velocidad);
27     }
28
29     @Test
30     public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero_Jesus() {
31         Coche nuevoCoche = new Coche();
32         nuevoCoche.velocidad = 50;
33         nuevoCoche.decelerar(80);
34         Assertions.assertEquals(0, nuevoCoche.velocidad);
35     }
36
37 }

```

```

3
4     public int velocidad;
5
6     public void acelerar_Jesus(int aceleracion) {
7         velocidad += aceleracion;
8     }
9
10
11     public void decelerar_Jesus(int deceleracion) {
12         velocidad -= deceleracion;
13         if(velocidad < 0) velocidad = 0;
14     }
15
16
17

```

Observamos como en los métodos de test el nombre de los métodos acelerar y decelerar ha cambiado.

```

8     @Test
9     public void test_al_crear_un_coche_su_velocidad_es_cero_Jesus() {
10         Coche nuevoCoche = new Coche();
11         Assertions.assertEquals(0, nuevoCoche.velocidad);
12     }
13
14     @Test
15     public void test_al_acelerar_un_coche_su_velocidad_aumenta_Jesus() {
16         Coche nuevoCoche = new Coche();
17         nuevoCoche.acelerar_Jesus(30);
18         Assertions.assertEquals(30, nuevoCoche.velocidad);
19     }
20
21     @Test
22     public void test_al_decelerar_un_coche_su_velocidad_disminuye_Jesus() {
23         Coche nuevoCoche = new Coche();
24         nuevoCoche.velocidad = 50;
25         nuevoCoche.decelerar_Jesus(20);
26         Assertions.assertEquals(30, nuevoCoche.velocidad);
27     }
28
29     @Test
30     public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero_Jesus() {
31         Coche nuevoCoche = new Coche();
32         nuevoCoche.velocidad = 50;
33         nuevoCoche.decelerar_Jesus(80);
34         Assertions.assertEquals(0, nuevoCoche.velocidad);
35     }
36
37 }

```