

Crearemos nuestro proyecto con el IDE Eclipse y veremos las diferencias respecto a IntelliJ.

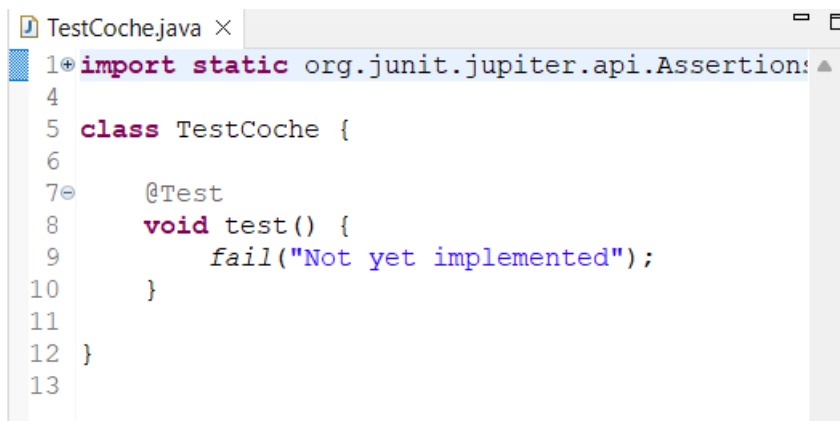
Lo primero, abrimos el Eclipse y creamos un nuevo proyecto llamado “TDD_Practica_Coche”. Para ello y cada vez que vayamos a crear algo nuevo seguimos el mismo esquema:

File → new → Java Project → *nombre* (TDD_Practica_Coche)

Ya tenemos el proyecto creado, ahora en la ventana de la izquierda nos aparece la carpeta ‘src’. Aquí vamos a crear la clase de los tests con ‘JUnit Test Case’. Nos pedirá que añadamos la librería ‘JUnit 5’ y aceptamos.

File → new → JUnit Test Case.

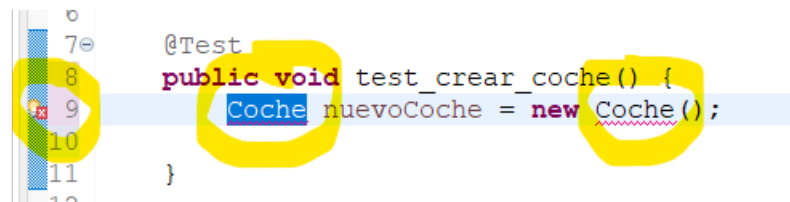
Como podemos ver, la primera diferencia con IntelliJ es que ya el propio Eclipse nos da el cuerpo para los test.



```
1 import static org.junit.jupiter.api.Assertions.*;
4
5 class TestCoche {
6
7     @Test
8     void test() {
9         fail("Not yet implemented");
10    }
11
12 }
13
```

Una vez tenemos el cuerpo del método, ya podemos crear nuestro test.

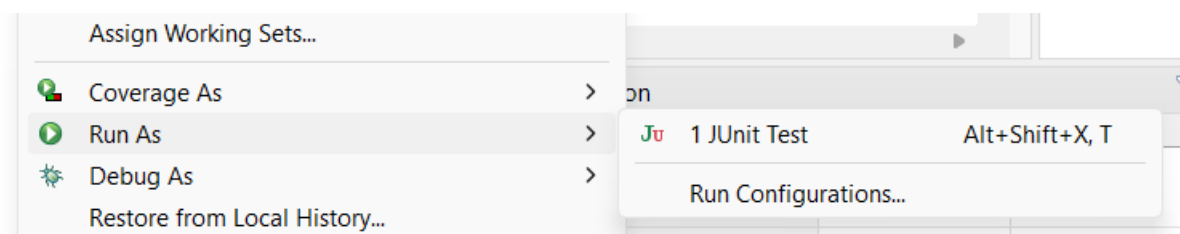
Vamos a indicar una clase *Coche* que todavía no existe y la instanciamos. El propio IDE nos avisa de que no existe y nos da la opción de crear esa clase.



```
7     @Test
8     public void test_crear_coche() {
9         Coche nuevoCoche = new Coche();
10    }
11
12 }
```

Como vemos a la izquierda nos aparece una ‘x’ en rojo. Clicamos y creamos la clase. Ahora ya está creada dicha clase.

Ahora, en el panel de la izquierda, hacemos click con el botón derecho sobre ‘TestCoche.java’ y abajo nos sale la opción de “Run As → JUnit Test”.



Lo ejecutamos y comprobamos que el test corre perfectamente.

```
Finished after 0,139 seconds
Runs: 1/1   Errors: 0   Failures: 0
> TestCoche [Runner: JUnit 5] (0,001 s)
```

El paso siguiente será modificar algo el test. Vamos a comprobar que al crear un coche, su velocidad es cero. Para ello, utilizamos la expresión 'assertEquals' que es una expresión booleana que compara el resultado esperado(primer valor) y el resultado dado (segundo valor) y si son el mismo devolverá true y sino false.

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

Al igual que sucedía antes, no detecta el campo velocidad en la clase "Coche" por lo que haremos el mismo paso de antes con la 'x' roja.

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

Cambiamos el tipo de dato a INT y volvemos a ejecutar el 'JUnit 5' para comprobar que lo ejecuta bien.

Ahora, iremos con la segunda parte del proyecto.

Vamos a crear un método que al acelerar, la velocidad aumente.

El esqueleto es el mismo pero con alguna modificación. Nos dará el error de que no existe el método 'acelerar' por lo que haremos lo mismo que hicimos con la 'velocidad', cambiando el nombre de la variable.

```
1
2 public class Coche {
3
4     public int velocidad;
5
6     public void acelerar(int aceleracion) {
7         velocidad +=aceleracion;
8     }
9
10 }
11
```

Volvemos a ejecutar el test y comprobamos que funciona.

Ahora, siguiendo los mismos pasos, vamos a crear el método decelerar. Como al crear Coche, la velocidad inicial es 0, no puede decelerar, por lo que le tenemos que pasar la velocidad que queramos. Por ejemplo, 50 y que el coche decelere 20. Entonces, el parámetro de resultado esperado que le tenemos que pasar a `Assertions.assertEquals` es de 30.

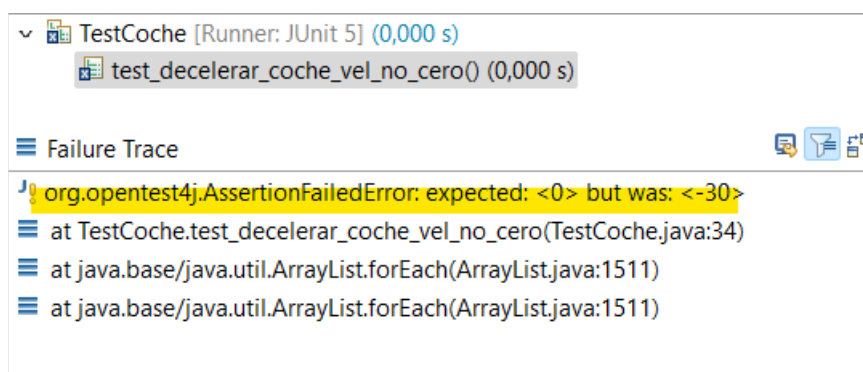
```
}
@Test
public void test_decelerar_coche_vel_disminuye() {
    Coche nuevoCoche = new Coche();
    nuevoCoche.velocidad=50;
    nuevoCoche.decelerar(20);
    Assertions.assertEquals(30, nuevoCoche.velocidad);
}
```

Ejecutamos y comprobamos que está bien.

Por último, vamos a crear un test que si el coche decelera, la velocidad no puede ser menor a cero.

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

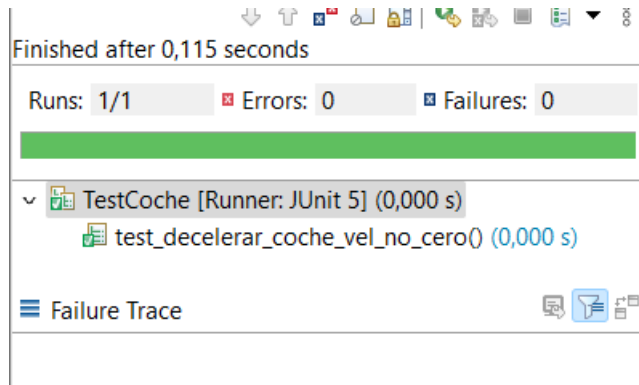
En este ejemplo, llevamos una velocidad de 50 y una deceleración de 80, por lo que el coche debería de frenar. Pero cuando comprobamos el test nos da error.



Como podemos ver, el error que nos da es que el resultado esperado es de 0 pero el resultado real es de -30 y nuestro test lo hemos hecho con la idea de que no puede ser 0, por lo que hay que arreglarlo. Para ello, nos vamos a nuestra clase Coche y modificamos el método decelerar con una condición que si la velocidad es menor de 0 la velocidad será 0.

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

Comprobamos el test y vemos como, ahora, sí funciona.



The screenshot shows a JUnit test runner interface. At the top, there is a toolbar with various icons. Below the toolbar, it says "Finished after 0,115 seconds". A summary bar shows "Runs: 1/1", "Errors: 0", and "Failures: 0". Below this is a green progress bar. The test results are listed below, showing a tree structure with "TestCoche [Runner: JUnit 5] (0,000 s)" expanded, revealing the test "test_decelerar_coche_vel_no_cero() (0,000 s)". At the bottom, there is a "Failure Trace" section with icons for viewing the trace.

Finished after 0,115 seconds

Runs: 1/1 Errors: 0 Failures: 0

TestCoche [Runner: JUnit 5] (0,000 s)

test_decelerar_coche_vel_no_cero() (0,000 s)

Failure Trace