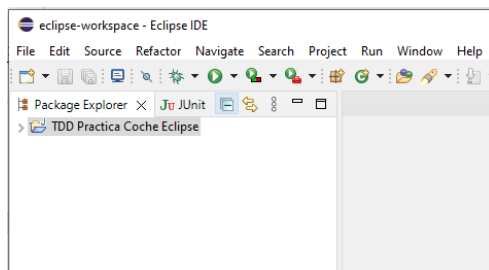
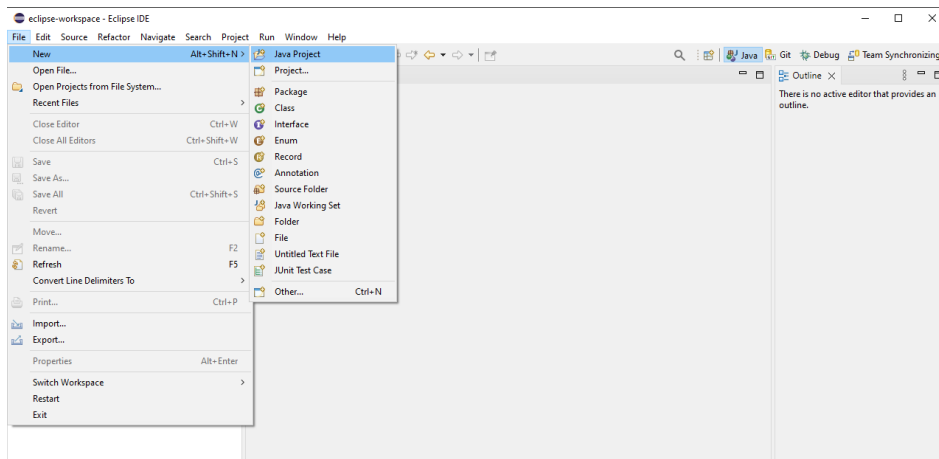


MEMORIA MI PRIMER TDD V 2.0

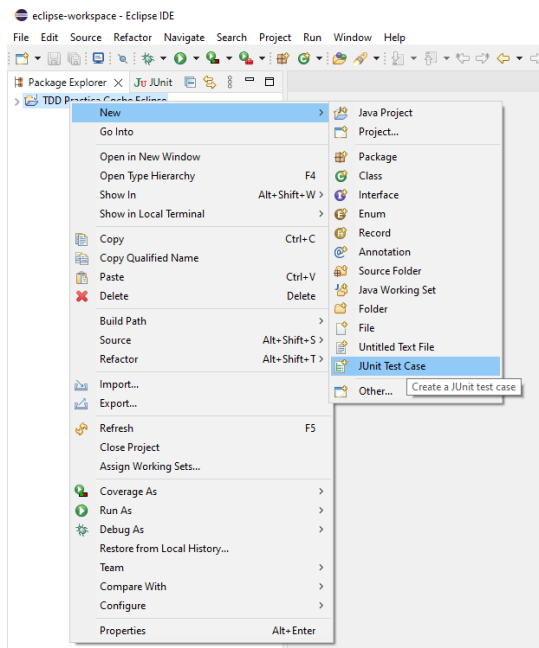
Laura María Pinedo Puertas

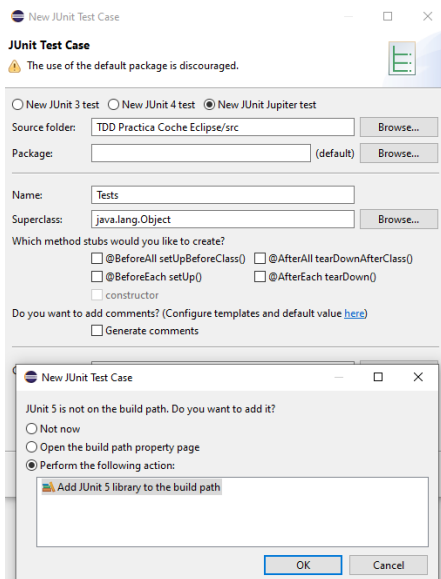
Entornos de Desarrollo, 1ºQ

Creamos un nuevo proyecto Java en Eclipse llamado “TDD Practica Coche Eclipse” pulsando en “File” -> “new” -> “Java Project”.

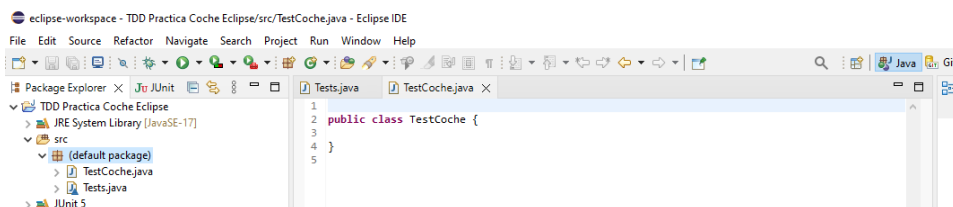


Creamos una nueva clase para realizar nuestros tests, para ello, tendremos que hacer click derecho en la carpeta del proyecto donde queramos generarla y pulsar en “JUnit Test Case” tal y como aparece en la captura y seleccionar “JUnit Jupiter test”.

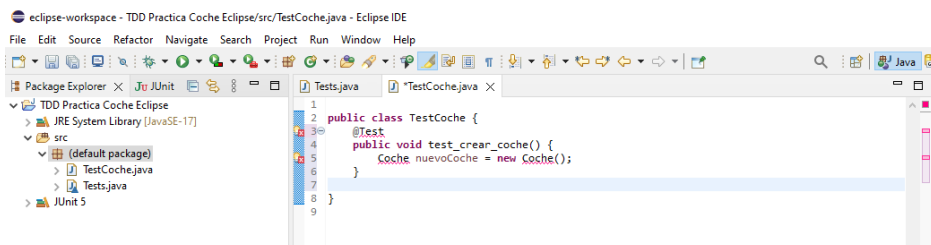




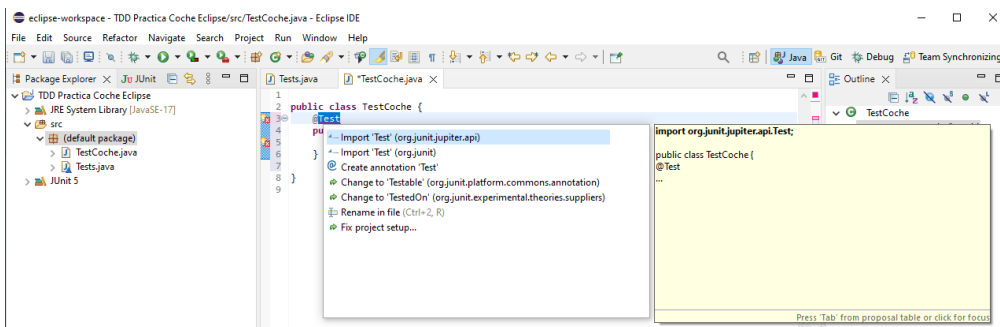
Crearemos una clase de Java dentro del directorio “tests” que se llamará TestCoche haciendo click derecho en la carpeta que acabamos de crear.



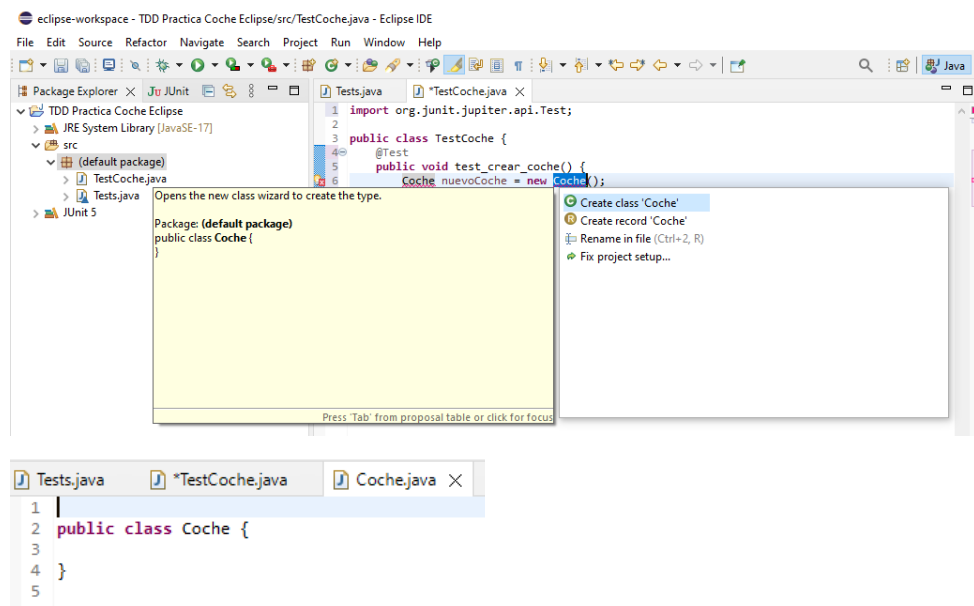
Escribimos nuestro test



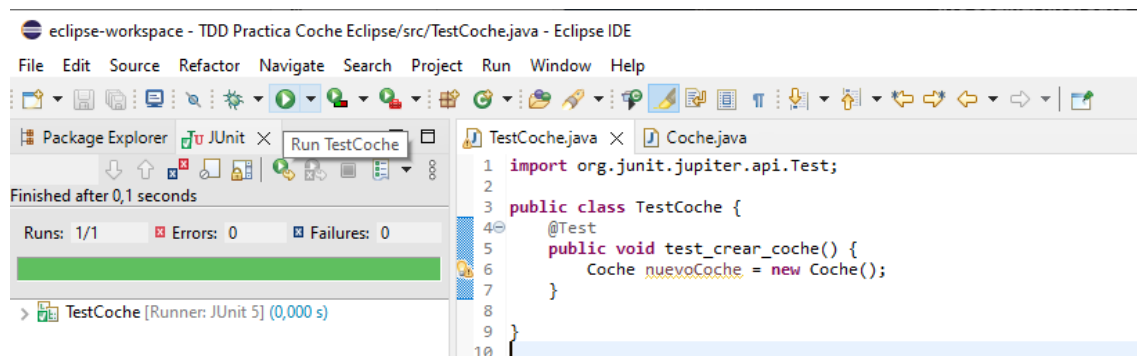
Y como podemos comprobar, nos aparecen errores tanto en @Test como en Coche, solucionaremos primero el error que aparece en @Test haciendo click izquierdo en la X que se muestra a la izquierda de la palabra @Test y especificaremos que queremos que se añada el “import org.junit.jupiter.api.Test”



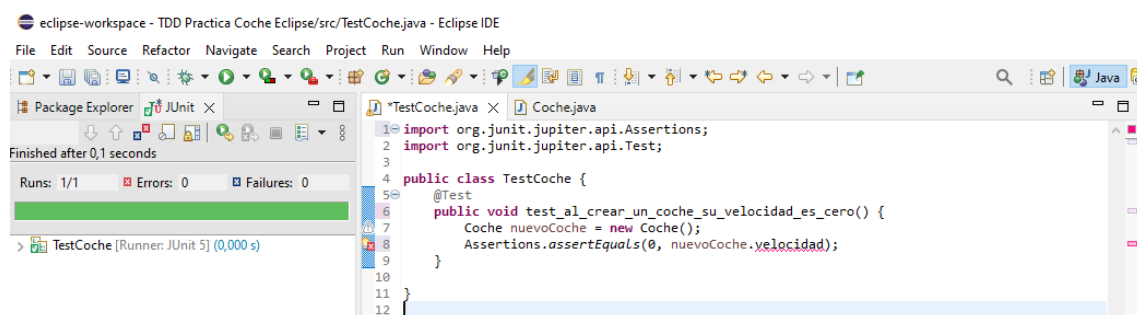
Ahora para solucionar el error que aparece en “Coche” hacemos click izquierdo en la X que aparece al lado de “Coche” y creamos la clase “Coche”



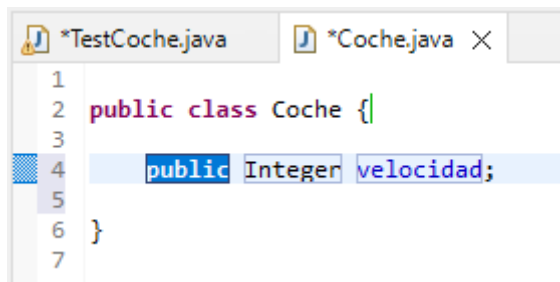
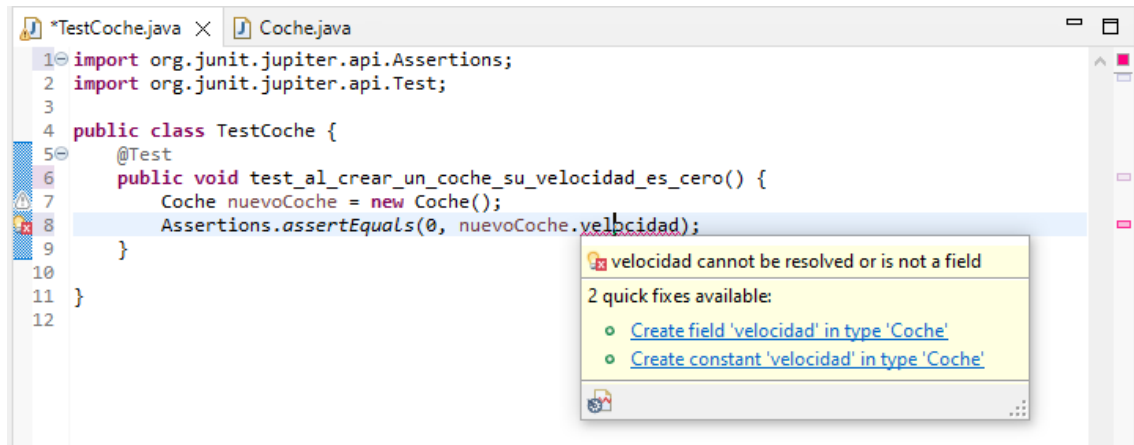
Ahora hacemos click en “Run test” para ejecutarlo y vemos que hemos pasado nuestro primer test.



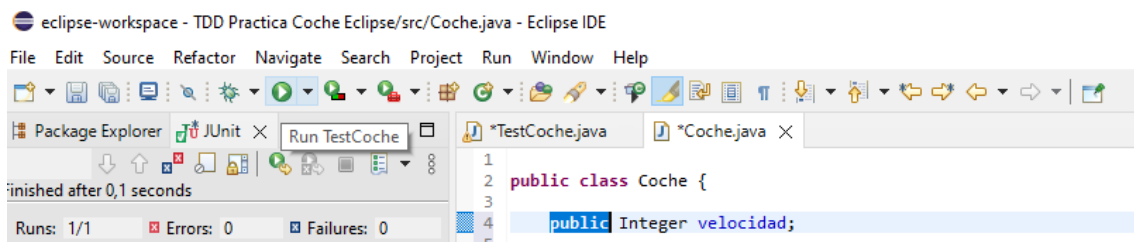
Mejoramos nuestro test, y para ello, cambiamos su nombre y utilizamos un Assertion.



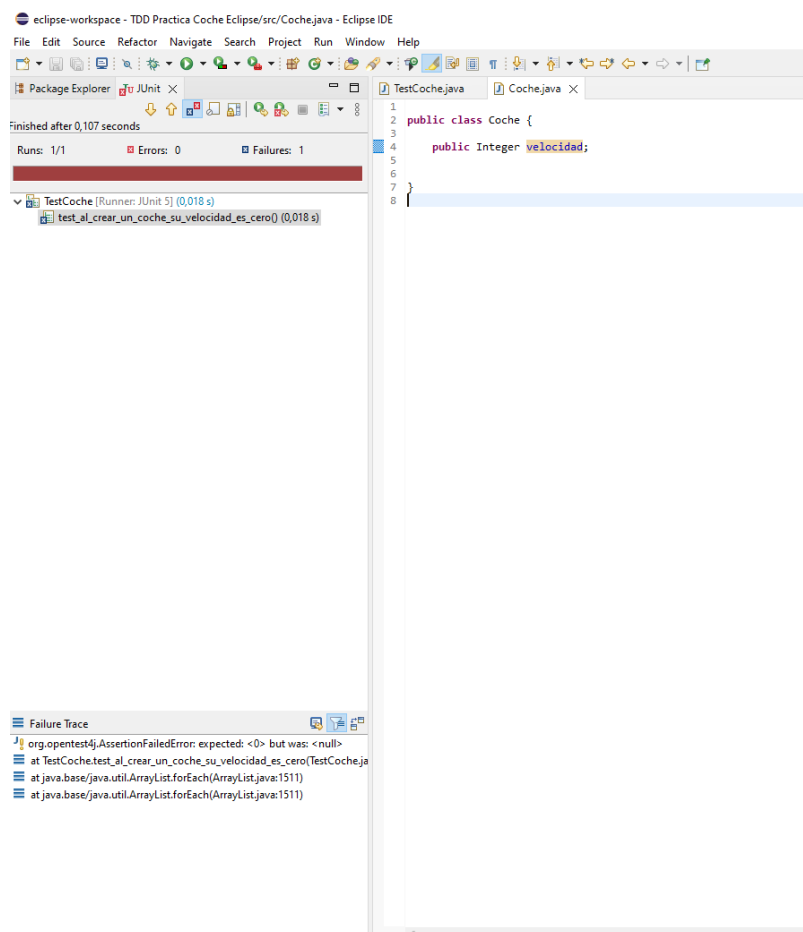
Como se aprecia en el pantallazo “velocidad” da error, debido a que, como antes, velocidad aún no ha sido creada, por lo que dejamos el ratón encima de “velocidad” y hacemos click en “Create field ‘velocidad’ in type ‘Coche’” y la creamos.



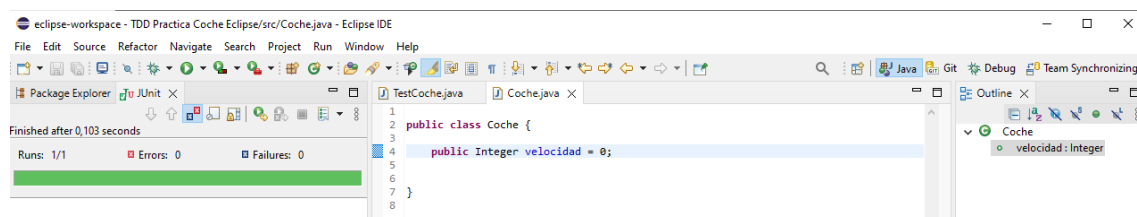
Ahora hacemos click en “Run” para ejecutar nuestro test.



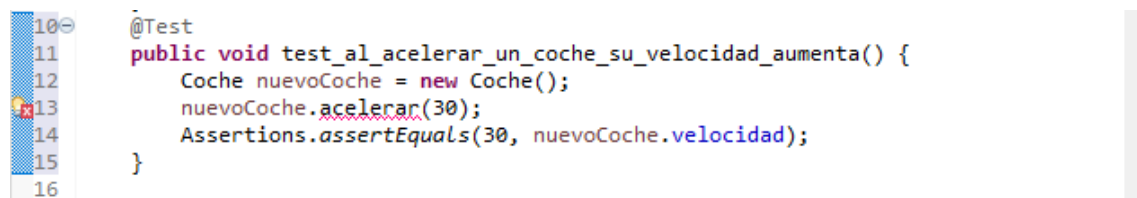
Y vemos que nos encontramos con un fallo



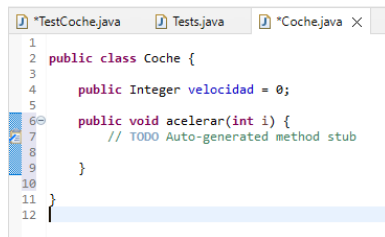
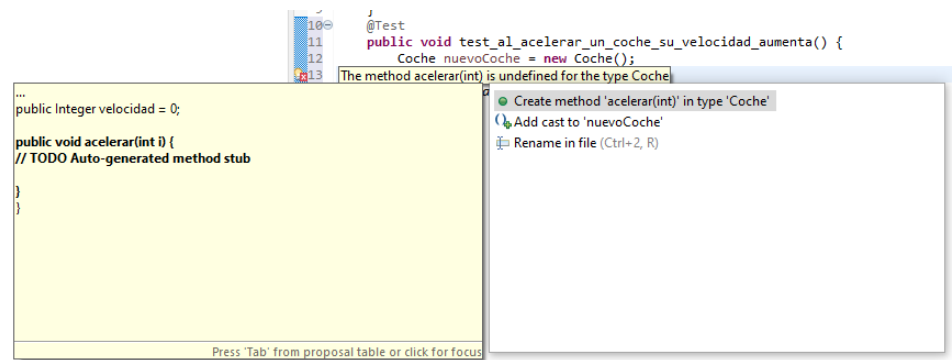
Éste es provocado por que no hemos dado un valor inicial de 0 a velocidad en la clase “Coche”, ya que, a diferencia de IntelliJ, en Eclipse se necesita, lo arreglamos y vemos que ahora sí pasa el test sin problemas.



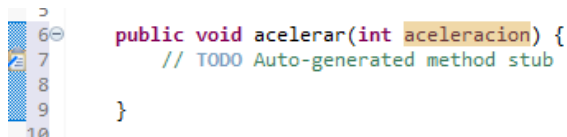
Creamos un nuevo test para “acelerar”.



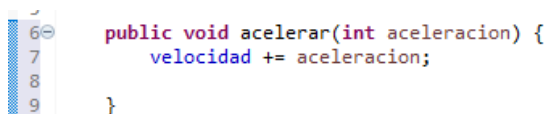
Tal y como pasaba antes, volvemos a tener el problema en “acelerar”, ya que todavía no hemos creado el método, por lo tanto, lo creamos haciendo click derecho en la bombilla con la X que aparece a la izquierda de “acelerar” y pulsamos “Create method ‘acelerar(int)’ in type ‘Coche’”.



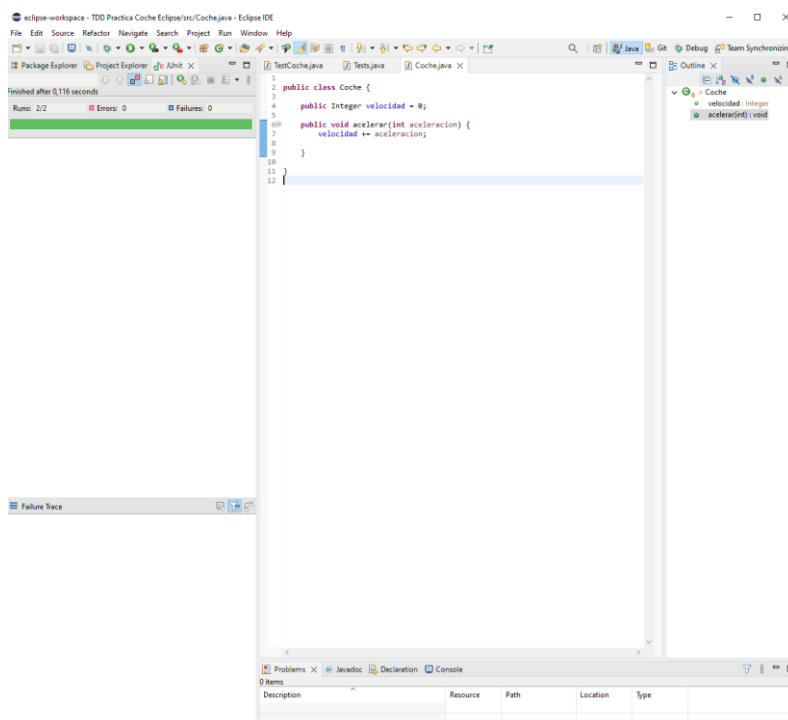
Cambiamos el nombre del argumento de “i” a “aceleracion”.



Le indicamos que su “velocidad” aumente en “aceleracion”.



Hecho esto, hacemos click en “Run” para ejecutar el test y vemos que el resultado no muestra ningún error



Creamos un nuevo test para “decelerar”.

```
16 @Test
17 public void test_al_decelerar_un_coche_su_velocidad_disminuye() {
18     Coche nuevoCoche = new Coche();
19     nuevoCoche.decelerar(30);
20     Assertions.assertEquals(30, nuevoCoche.velocidad);
21 }
```

Así sin mas no funcionaría, debido que al crear el coche sin más la velocidad estaría a 0, por lo que, antes de nada, debemos asignarle una velocidad mayor, por ejemplo 50.

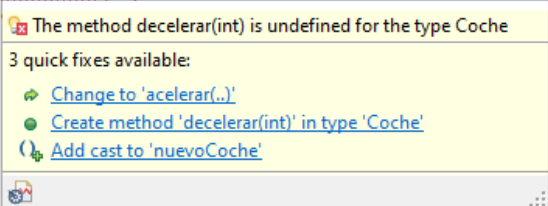
```
Coche nuevoCoche = new Coche();
nuevoCoche.velocidad = 50;
```

Hecho esto, indicaremos que queremos que decelere 20 y expected lo dejaremos en 30.

```
20     nuevoCoche.decelerar(20);
21     Assertions.assertEquals(30, nuevoCoche.velocidad);
22 }
```

Como vemos, otra vez tenemos el problema de que no encuentra el método “decelerar”, así que lo creamos de la misma forma que hemos hecho anteriormente usando “Create method”.

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

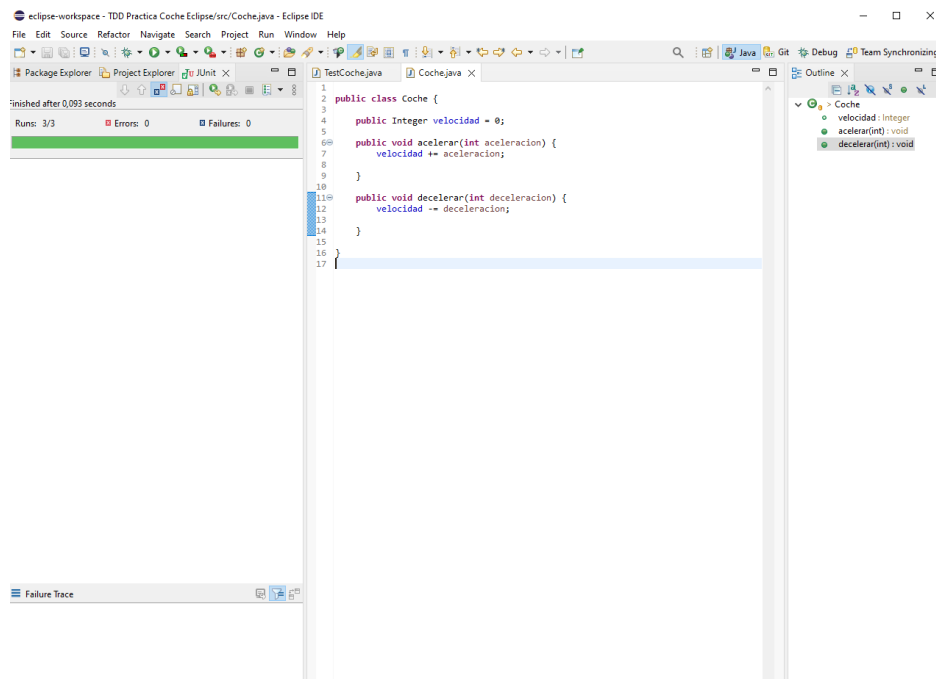


```
11 public void decelerar(int i) {
12     // TODO Auto-generated method stub
13
14 }
```

Y cambiamos el nombre del parámetro de “i” a “deceleración”, además le indicamos que la velocidad disminuya en la deceleración.

```
11 public void decelerar(int deceleracion) {
12     velocidad -= deceleracion;
13
14 }
```

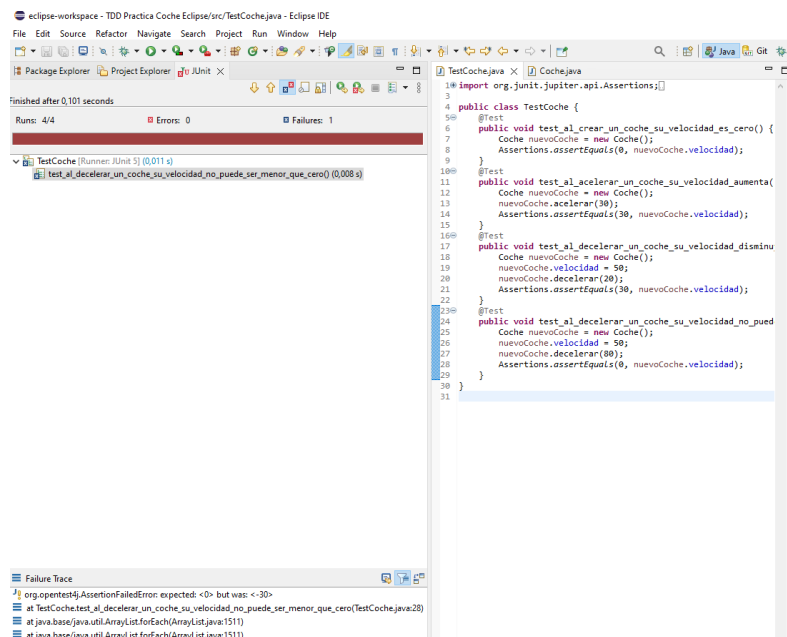

Ahora hacemos click en “Run” y vemos que pasa los tests sin problemas.



Creamos un nuevo test para que al decelerar su velocidad no pueda ser menor de 0.

```
23 @Test
24 public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero() {
25     Coche nuevoCoche = new Coche();
26     nuevoCoche.velocidad = 50;
27     nuevoCoche.decelerar(80);
28     Assertions.assertEquals(0, nuevoCoche.velocidad);
29 }
```

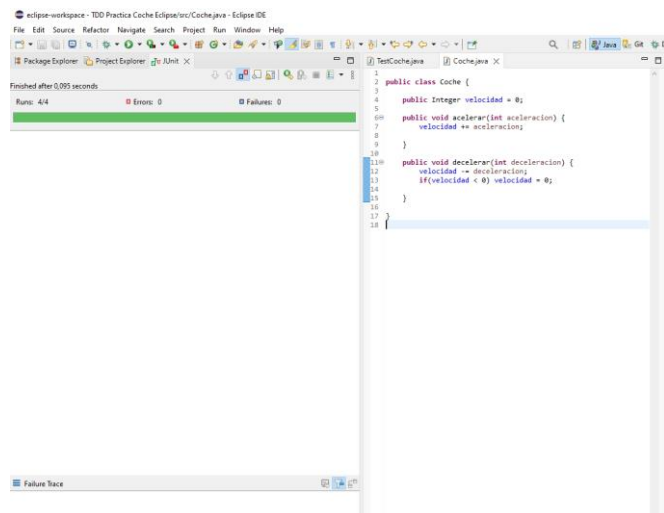
Hacemos click en “Run” para comprobar si pasamos los tests y, como podemos ver, nos encontramos con que este último falla, ya que se esperaba que el resultado fuese 0 pero ha dado -30.



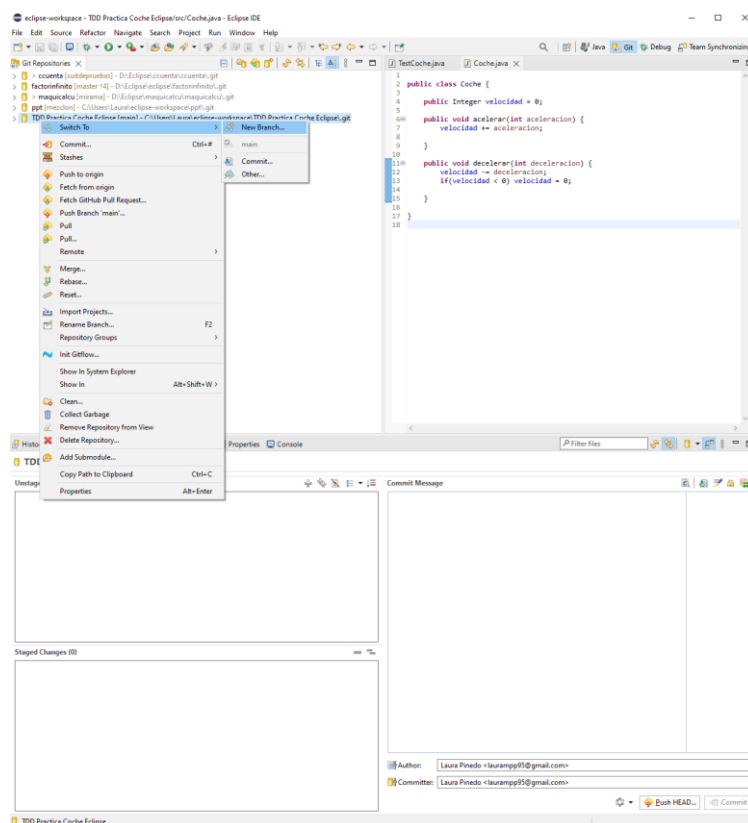
Para evitar este error, nos vamos a la clase “Coche” y especificamos que, si la velocidad es menor que 0, entonces el valor de velocidad será igual a 0.

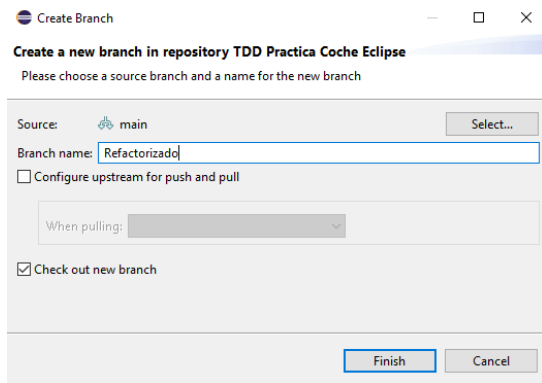
```
11 public void decelerar(int deceleracion) {
12     velocidad -= deceleracion;
13     if(velocidad < 0) velocidad = 0;
14
15 }
```

Hecho esto, volvemos a pulsar “Run” y vemos que todo se ejecuta sin problemas.

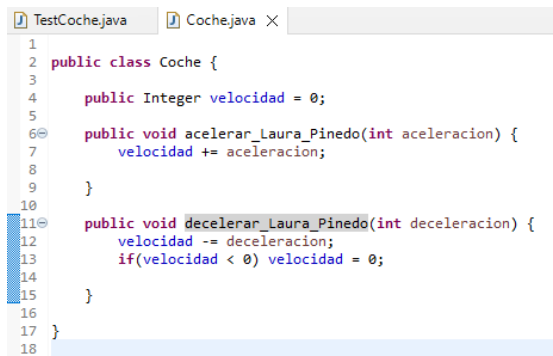
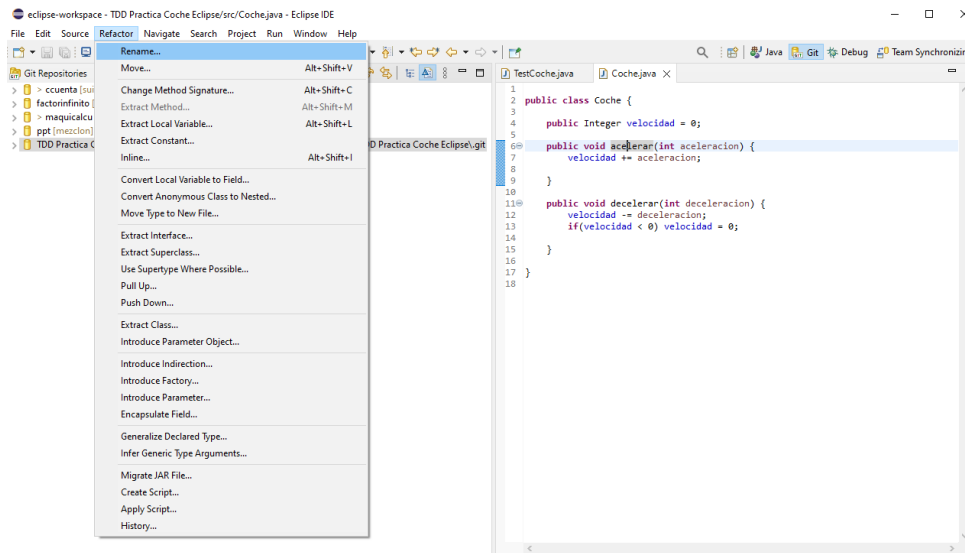


Por último, nos vamos a la pestaña de Git, hacemos click derecho en el repositorio en el que deseamos crear una nueva rama, dejamos el cursor encima de “Switch to” y pulsamos “New Branch” para crear una nueva rama llamada “Refactorizado”





Una vez creada, hacemos click en el nombre del método que queramos refactorizar, pinchamos en “Refactor” -> “Rename”, que se encuentra en el menú de arriba, tal y como muestra la imagen, y cambiamos al nombre deseado, en este caso: nombremetodo_Laura_Pinedo y pulsamos Enter.



```

1* import org.junit.jupiter.api.Assertions;
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.assertEquals(0, nuevoCoche.velocidad);
9     }
10    @Test
11    public void test_al_acelerar_un_coche_su_velocidad_aumenta() {
12        Coche nuevoCoche = new Coche();
13        nuevoCoche.acelerar_Laura_Pinedo(30);
14        Assertions.assertEquals(30, nuevoCoche.velocidad);
15    }
16    @Test
17    public void test_al_decelerar_un_coche_su_velocidad_disminuye() {
18        Coche nuevoCoche = new Coche();
19        nuevoCoche.velocidad = 50;
20        nuevoCoche.decelerar_Laura_Pinedo(20);
21        Assertions.assertEquals(30, nuevoCoche.velocidad);
22    }
23    @Test
24    public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_ce
25        Coche nuevoCoche = new Coche();
26        nuevoCoche.velocidad = 50;
27        nuevoCoche.decelerar_Laura_Pinedo(80);
28        Assertions.assertEquals(0, nuevoCoche.velocidad);
29    }
30 }

```

Y así, habremos acabado la práctica.

Como conclusión, destacar que, en comparación con IntelliJ, Eclipse tiene diferencias muy marcadas que pueden hacer que te confundas a la hora de hacer alguna acción. Por ejemplo, al refactorizar, ya que a diferencia de IntelliJ que aparece al hacer click derecho en el nombre del método que queremos refactorizar, en Eclipse tienes que buscarlo en la barra de tareas o menú. Éste es solo un ejemplo entre otros muchos que se pueden ver al comparar ambas prácticas.