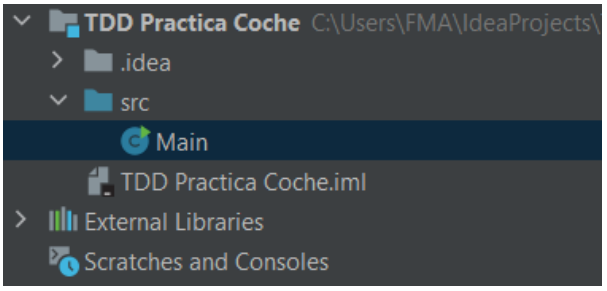
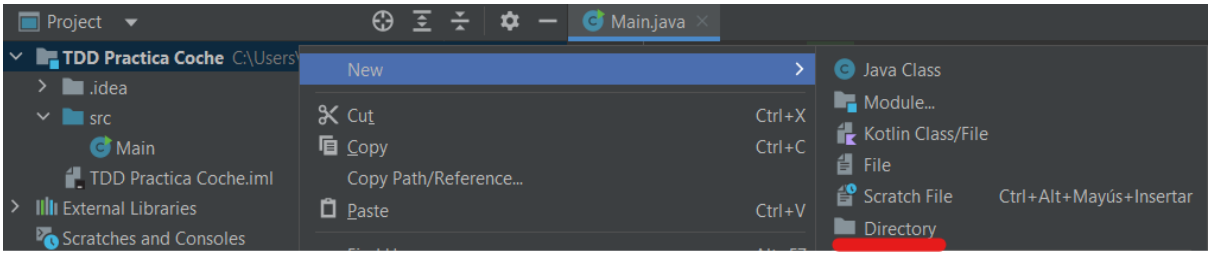
Creación de TDD

Francisco Molina Adan

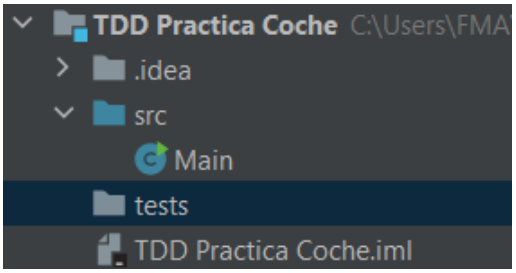
Lo primero que debemos hacer es abrir nuestro IDE y crear un nuevo proyecto, por ejemplo, en IntelliJ con el nombre TDD Practica Coche.



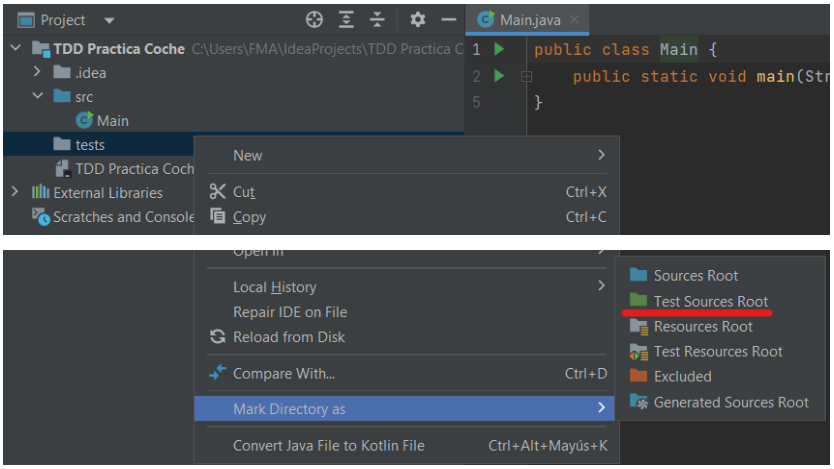
Ahora vamos a crear una carpeta nueva en nuestro proyecto, haciendo click derecho sobre el proyecto.

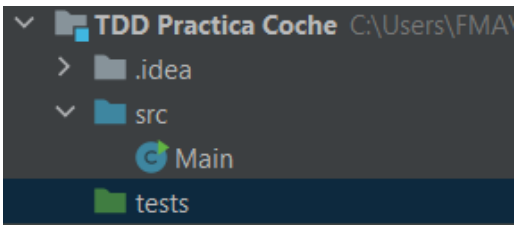


Vamos a nombrar a la carpeta “tests” ya que como su nombre indica es donde vamos a realizar los tests.

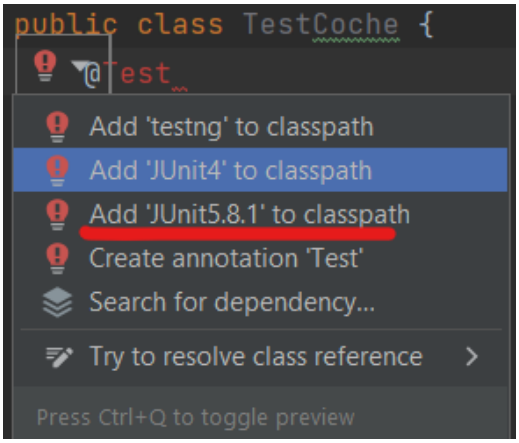


Ahora vamos a marcar este directorio como un directorio de test, para esto click derecho sobre el directorio.

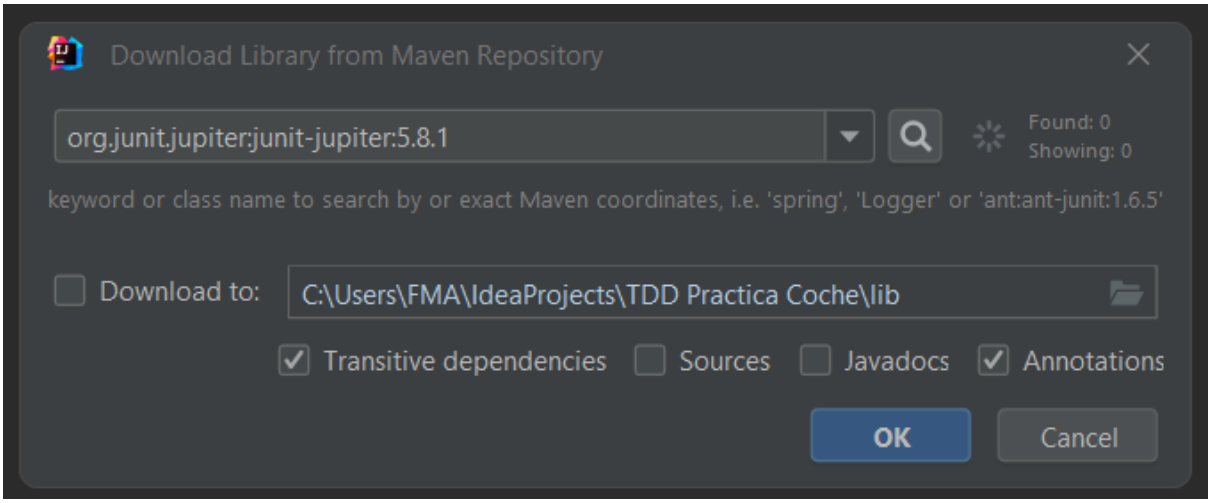




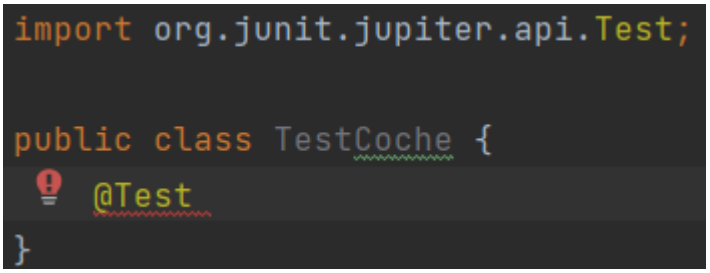
Ahora vamos a crear dentro de esta carpeta una clase con el nombre “TestCoche”.
Dentro de la clase escribimos @Test y el IDE nos marcará en rojo esta línea si hacemos click sobre ella nos mostrará posibles soluciones, donde seleccionaremos JUnit5.



Nos aparecerá un cuadro donde marcaremos la opción de “Annotations” y daremos Ok.



Ya sera capaz de reconocer @Test.

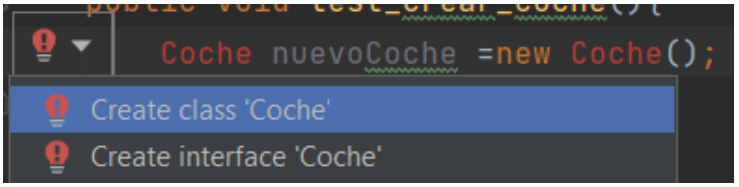


Además vemos que ha importado la librería de JUnit.

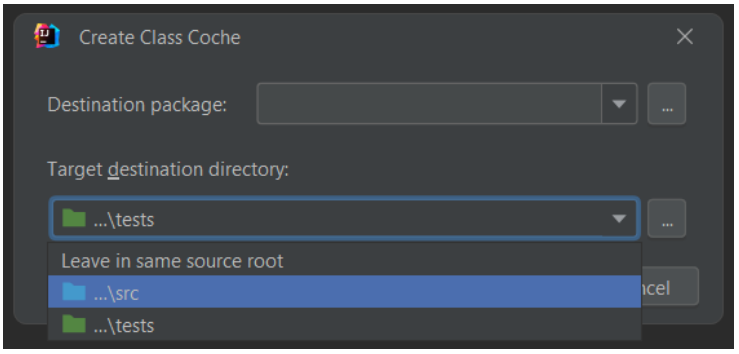
Creamos el siguiente método dentro de @Test.

```
public class TestCoche {
    @Test
    public void test_crear_coche(){
        Coche nuevoCoche =new Coche();
    }
}
```

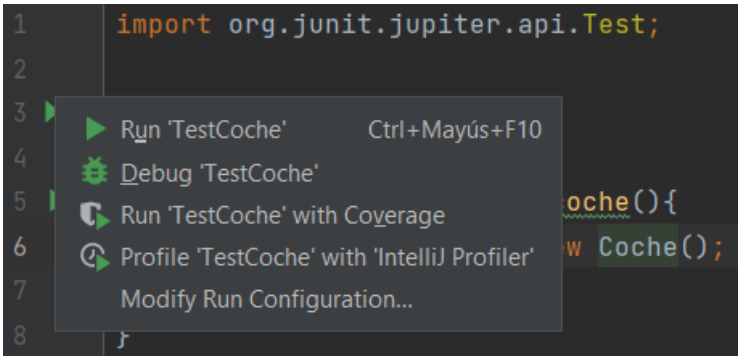
Creamos un objeto Coche aunque no tengamos aún la clase creada, haciendo clic sobre “Coche” el IDE nos recomienda crear la clase Coche.



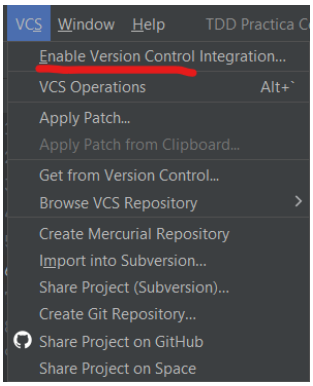
Se nos abrirá un cuadro donde nos aseguraremos de que el archivo se cree en “src” y daremos OK.



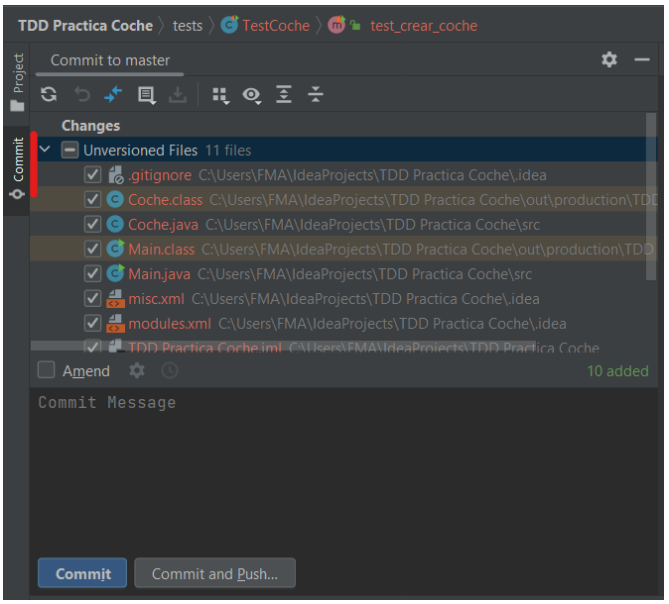
Para comprobar que todo va bien, dentro de TestCoche damos clic en RunTest.



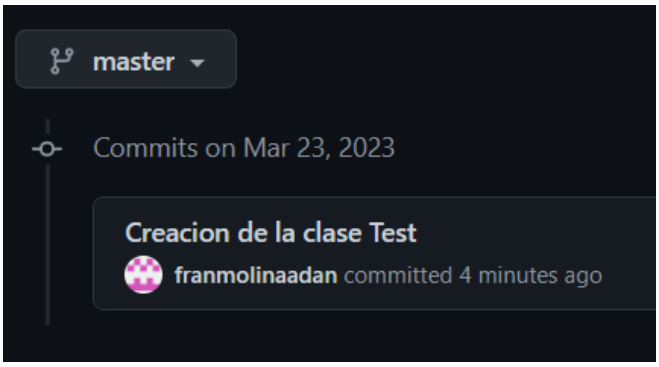
Ahora vamos a activar el control de versiones y hacer nuestro primer commit.



Ahora nos vamos al apartado de commit y seleccionamos todos los archivos.



Añadimos un mensaje en el commit y lo realizamos.
Comprobamos que todo ha salido bien y nuestro proyecto ya está en github.



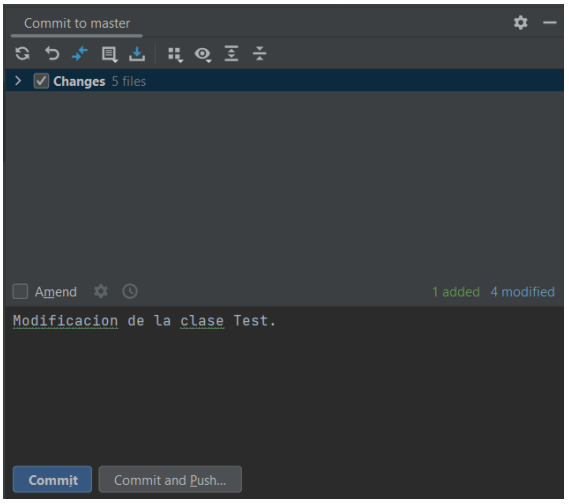
Ahora vamos a modificar el nombre del método a “test_al_crear_un_coche_su_velocidad_es_cero” y vamos a agregar la siguiente linea, automáticamente el IDE te añadirá la librería de Assertions.

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

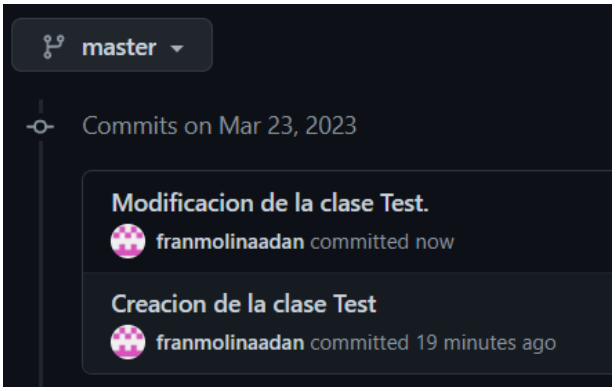
Damos click sobre velocidad y creamos la variable.

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

Nos creará la variable en coche.
Ejecutamos el test para comprobar que todo ha ido bien y realizamos un commit comentando los cambios.



Deberíamos tener dos commits en nuestro github.



A continuación vamos a crear otro método Test llamado como en la imagen.

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

Ahora vamos a acelerar nuevo coche, por ejemplo, 30.

```
Coche nuevoCoche =new Coche();
nuevoCoche.acelerar(30);
```

Como hemos hecho anteriormente hacemos clic en el método acelerar y el IDE nos lo crea automáticamente.
Una vez creado, le ponemos al método como entrada un entero que será la aceleración y dentro del método que la velocidad sea igual a la velocidad más la aceleración.

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

Por último cambia el valor esperado de assert a 30.

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

De la misma forma vamos a crear otro método para decelerar, debe quedar asi.

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

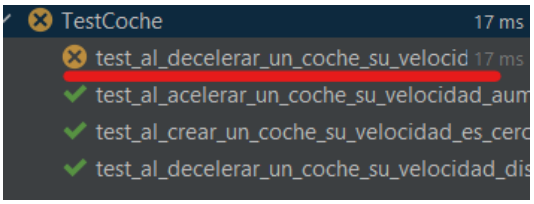
Creamos el método decelerar con la ayuda del IDE y al contrario que en el caso anterior disminuimos su velocidad.

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

Ahora vamos a crear otro método más para comprobar que cuando frenemos un coche su velocidad no sea nunca negativa.

```
@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( expected: 0, nuevoCoche.velocidad);
}
```

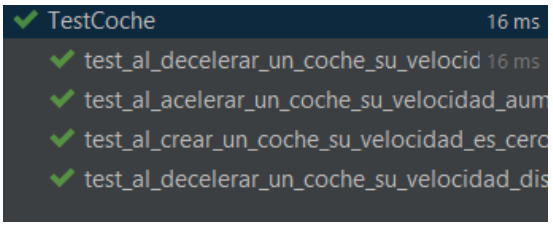
Si intentamos frenar más de la velocidad que tiene el coche y ejecutamos test debe fallar.



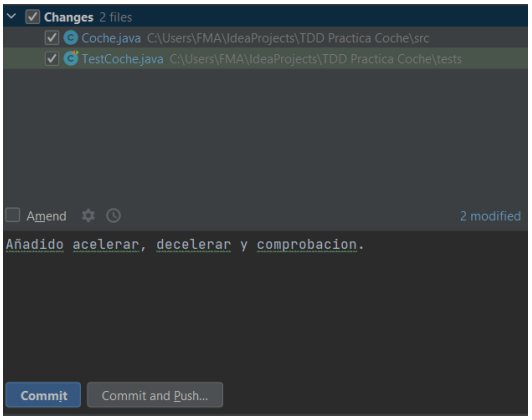
Para arreglar esto vamos a añadir una condición al método decelerar para que cuando su resta sea menor que 0 la velocidad sea 0.

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

Y asi todo debería funcionar correctamente.

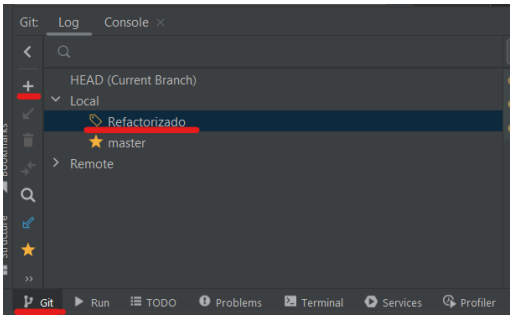


No olvides hacer un commit con tus cambios.



Comprobamos.

Vamos a crear una nueva rama llamada Refactorizado. Para esto clic en “Git” y al botón + para crear la rama.



Ahora vamos a añadir a todos los métodos mi nombre.

```java
@Test
public void test_al_acelerar_un_coche_su_velocidad_aumenta_FRAN_MOLINA(){
    Coche nuevoCoche =new Coche();
    nuevoCoche.acelerar( aceleracion: 30);
    Assertions.assertEquals( expected: 30, nuevoCoche.velocidad);
}
franmolinaadan *
@Test
public void test_al_decelerar_un_coche_su_velocidad_disminuye_FRAN_MOLINA(){
    Coche nuevoCoche =new Coche();
    nuevoCoche.velocidad=50;
    nuevoCoche.decelerar(20);
    Assertions.assertEquals( expected: 30, nuevoCoche.velocidad);
}
franmolinaadan *
@Test
public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero_FRAN_MOLINA(){
    Coche nuevoCoche =new Coche();
```

Una vez refactorizado commit de la rama.