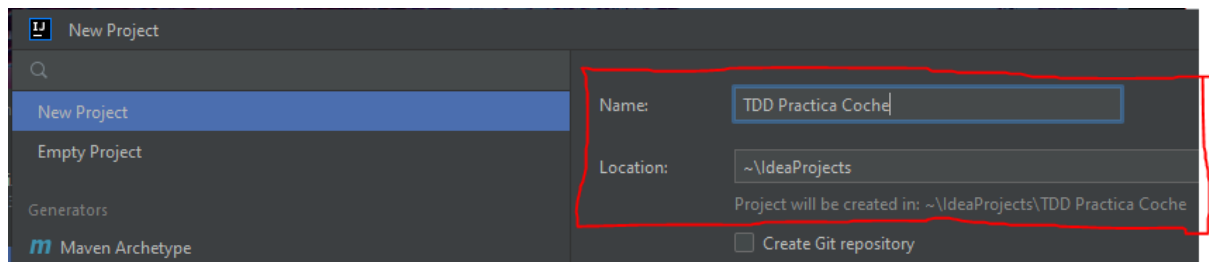
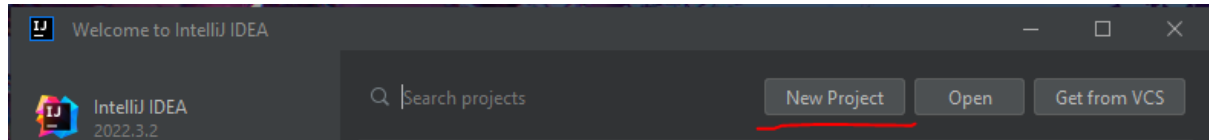
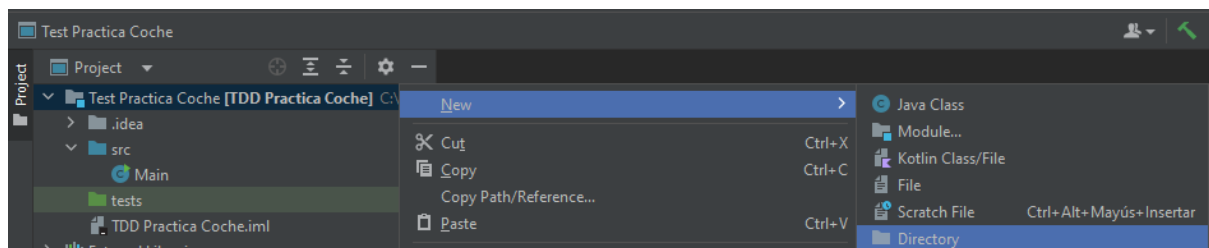


Documentación TDD IntelliJ

Lo primero que tenemos que hacer es crear un nuevo proyecto. En este caso lo llamaremos “TDD Practica Coche”

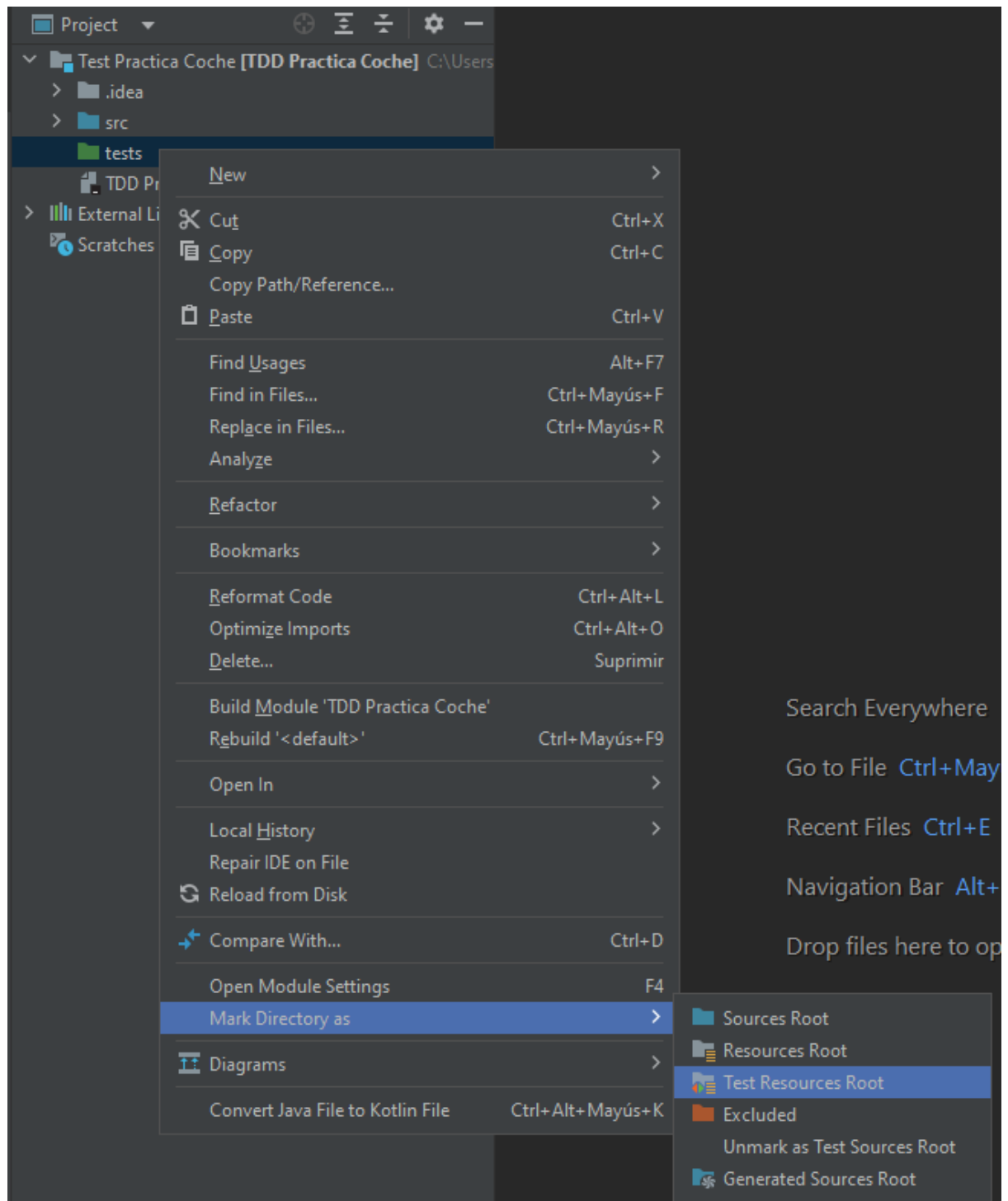


Una vez estamos ya dentro, hacemos click derecho en el propio proyecto y creamos un nuevo directorio

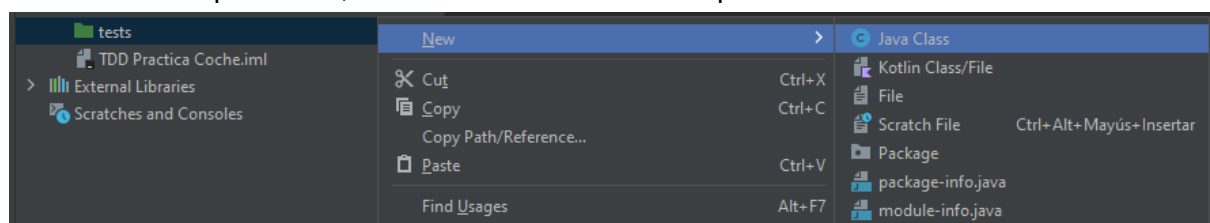


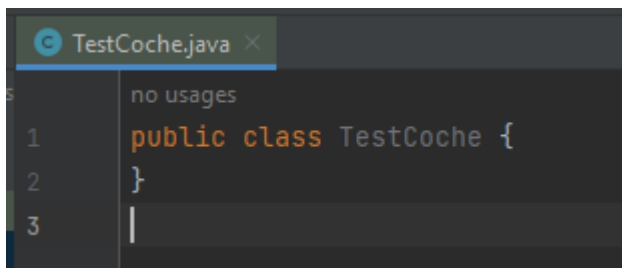
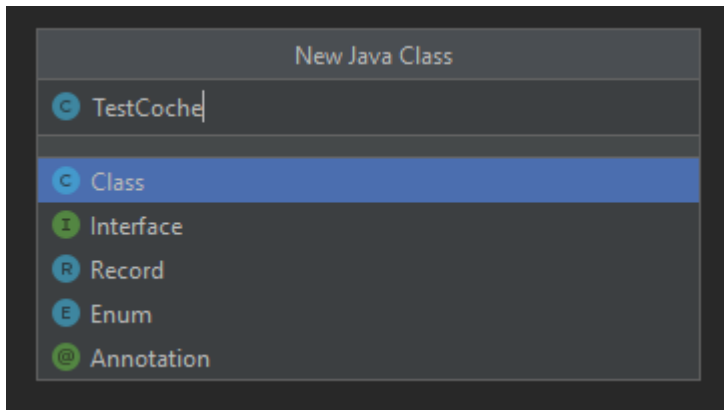
Lo llamaremos “tests”

Y, cuando lo creamos, tenemos que hacer click derecho sobre el y buscar la opción “Mark Directory as” y tendremos que clicar en “Test Sources Root”

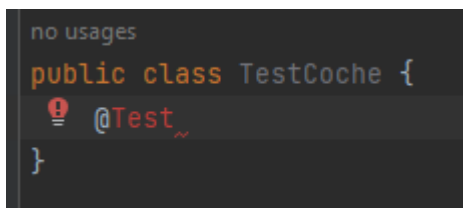


Dentro de la carpeta tests, crearemos una clase Java que se llame “TestCoche”

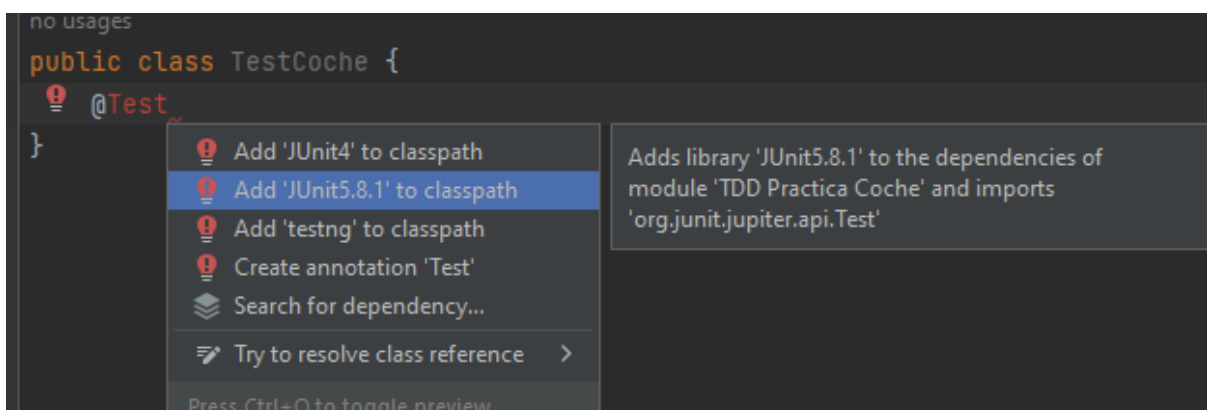




Una vez aquí, vamos a escribir “@Test”, esto aparecerá en rojo.



Así que haremos Alt + Enter y seleccionaremos “Add JUnit 5 to classpath” o lo que se le parezca.



Aparecerá una ventanita para descargar la librería. Pulsamos Ok y esperamos que se descargue.

```
import org.junit.jupiter.api.Test;

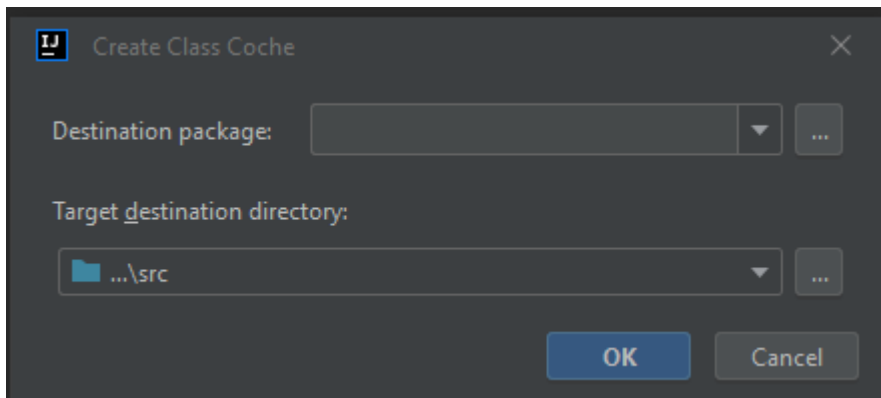
no usages
public class TestCoche {
    @Test
}
```

Ahora empezaremos con el primer test.

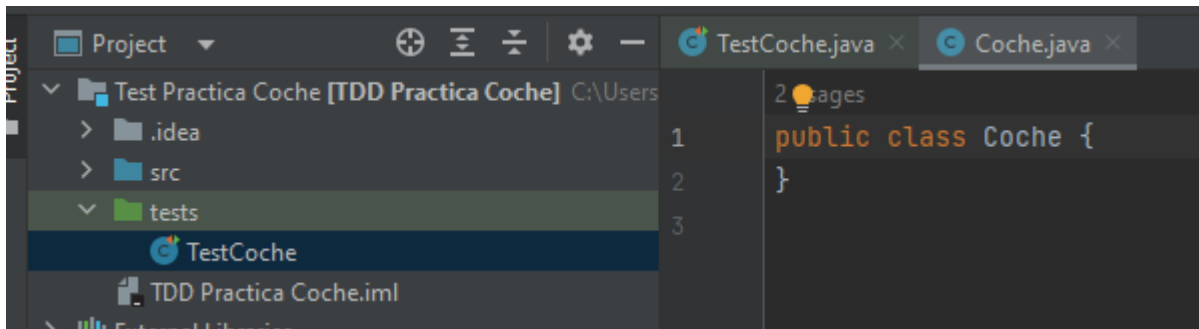
```
public class TestCoche {
    @Test
    public void test_crear_coche(){
        Coche nuevoCoche = new Coche();
    }
}
```

Como vemos, Coche sale en rojo porque es una clase que aún no hemos creado. No existe. Por ello, le vamos a crear la clase. Hacemos click en Coche y hacemos Alt + Mayus + Intro y pulsamos en “Create class ‘Coche’”

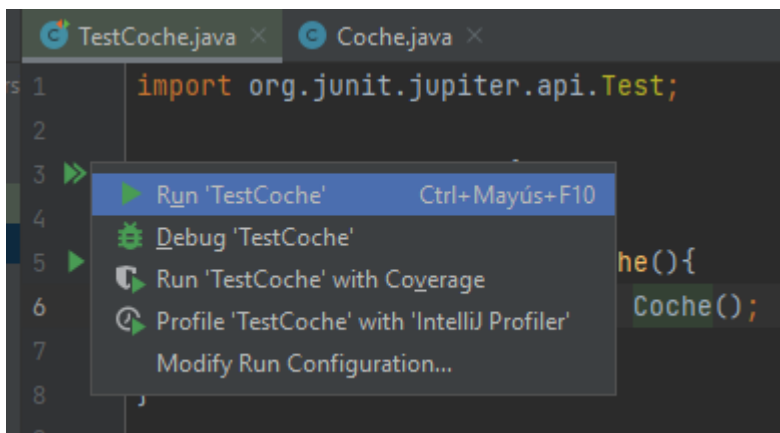
Es importante que creamos la clase en la carpeta src y no en tests.



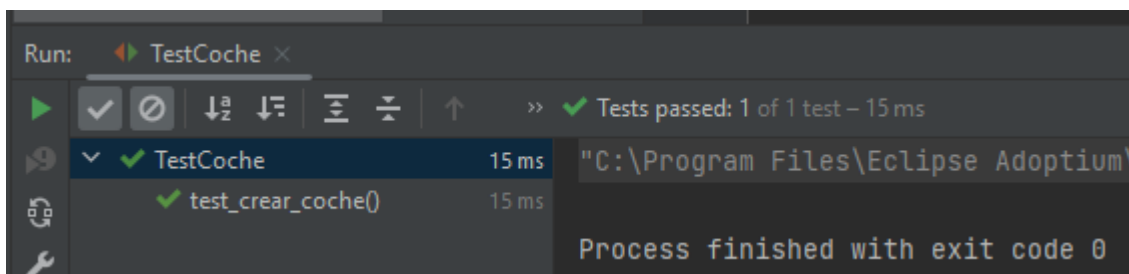
Pulsamos Ok y se nos creará la clase.



Una vez que tengamos creada la clase Coche, volvemos a TestCoche.java y ejecutamos el test de la siguiente forma:



Y pulsamos en Run 'TestCoche'

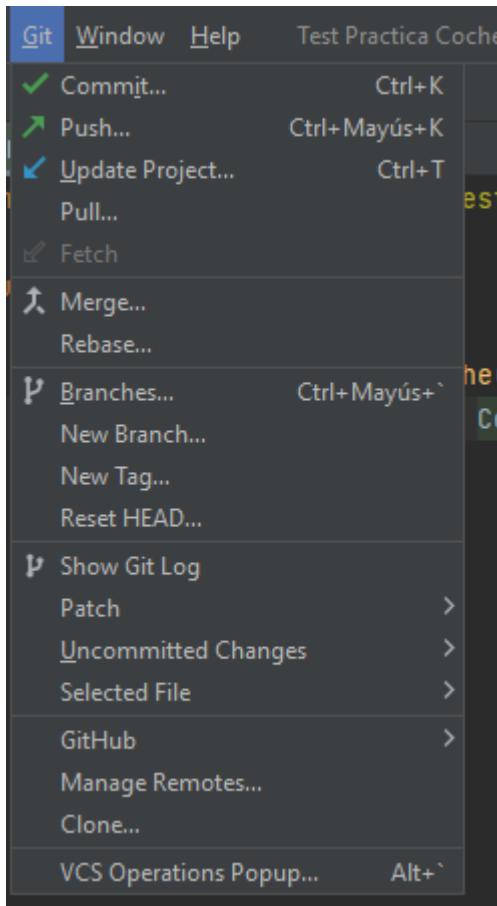


Hemos pasado el primer test

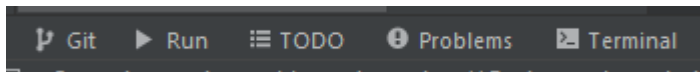
Vamos ahora a hacer un commit de todo lo que hemos hecho hasta ahora en el proyecto.

En la parte de arriba, debería haber una pestaña de Git, si en su lugar, hay una pestaña llamada VCS, presionamos ahí y pulsamos en "Enable Version Control" y seleccionamos Git.

Entonces debería aparecer Git en lugar de VCS y debería tener este aspecto:



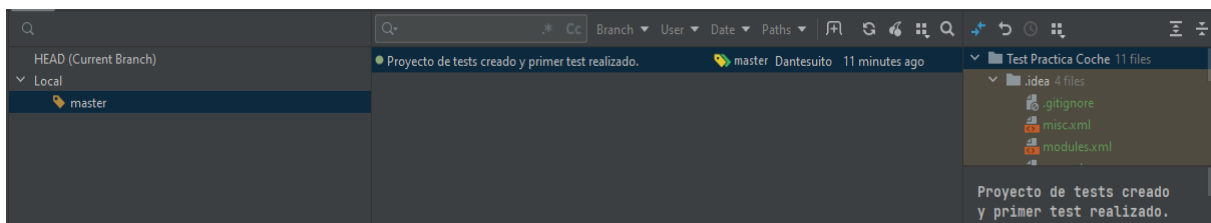
Cuando esto esté hecho, abajo a la izquierda habrá un botón de Git, junto con la pestaña de ejecución del programa, los problemas etc...

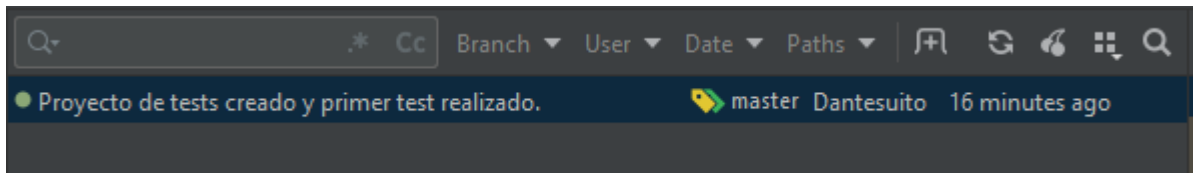


Pulsamos en el.

Hacemos Ctrl + K y arriba a la izquierda marcamos todos los archivos modificados de los que queremos hacer commit. Escribimos un mensaje de commit y pulsamos en commit.

En este caso he escrito "Proyecto de tests creado y primer test realizado".
Debería aparecer esto:





Ahora con el commit hecho, podemos seguir con el proyecto.

Vamos a mejorar el test y vamos a darle más funcionalidad.

Lo primero que haremos será modificar el nombre del método.

```
@Test
public void test_al_crear_un_coche_su_velocidad_es_cero(){
    Coche nuevoCoche = new Coche();
}
```

Nada muy complejo.

Ahora pondremos un Assertion

Podemos importar las Assertions escribiendo "import org.junit.jupiter.api.Assertions"

O al poner en el código Assertions, debería importarse directamente.

Escribiremos lo siguiente:

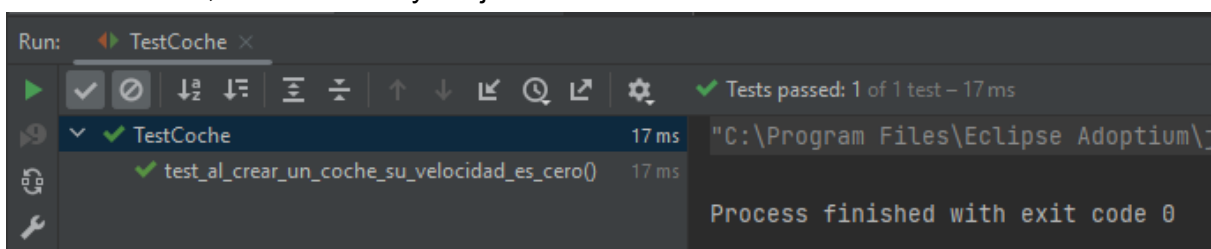
```
Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
```

Vemos que velocidad sale en rojo, lo mismo que la clase de antes, no está, no existe.

Pulsamos en velocidad, hacemos Alt + Mayus + Intro y se nos crea la variable velocidad en la clase coche.

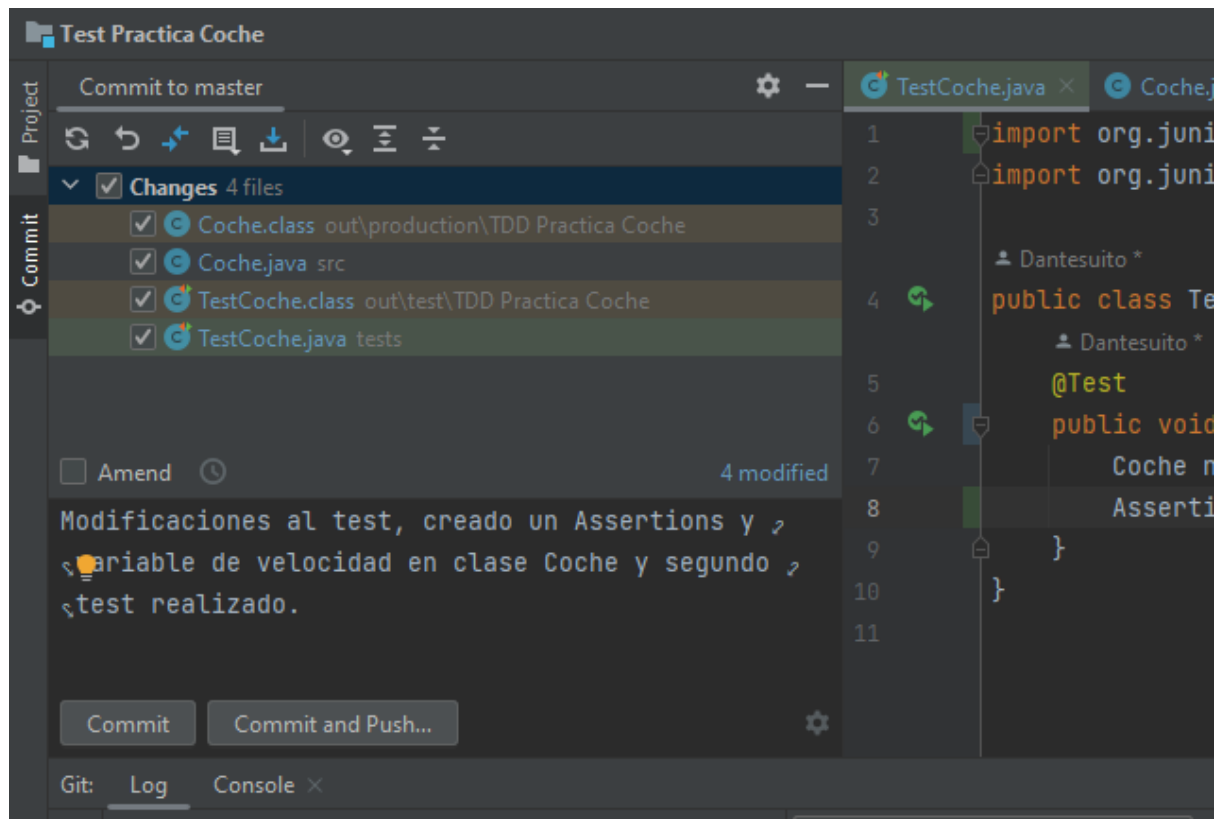


Con esto hecho, vamos al Test y lo ejecutamos como hicimos antes.

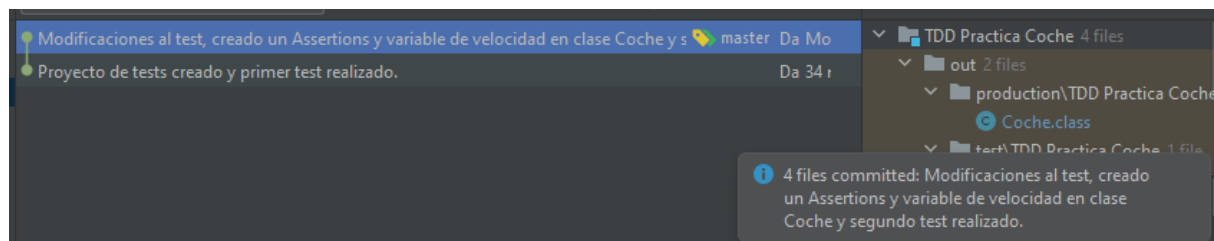


También lo pasamos.

Ahora vamos a hacer otro commit de los cambios que hemos hecho respecto al último.



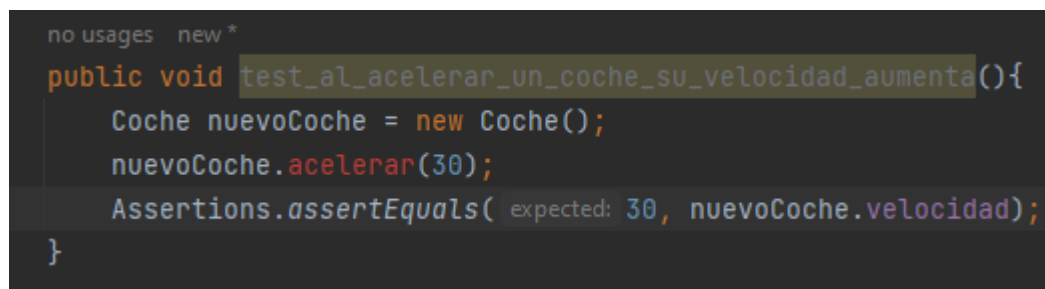
Marcamos lo que queremos hacer commit, escribimos el mensaje y pulsamos en commit.



Vemos que tenemos una pestaña con los cambios en cada commit.

Seguimos con el proyecto.

Vamos a modificar la clase TestCoche añadiendo otro método.



El nombre del método es bastante auto explicativo.

El Assertions espera que al ejecutarse el método acelerar la velocidad sea de 30.

Como antes, el método acelerar no existe.

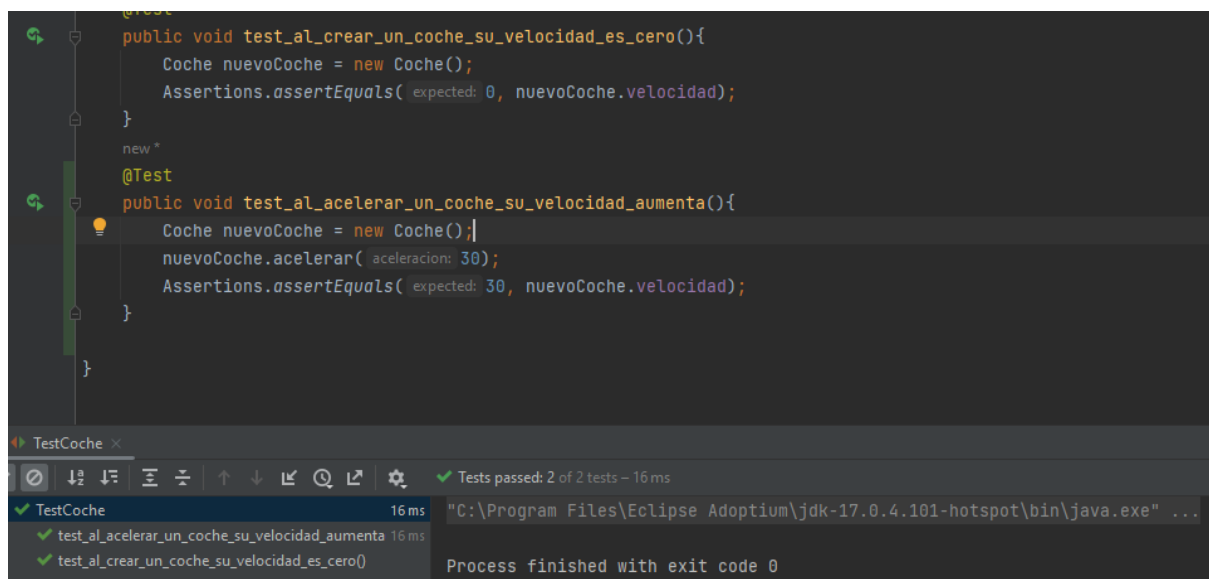
Nos pondremos encima y haremos Alt + Mayus + Intro para crearlo.

```
1 usage new *  
public void acelerar(int i) {  
}
```

También vamos a modificar este método nuevo, le vamos a cambiar el nombre de la variable por “aceleracion” y vamos a indicarle que velocidad aumente en aceleración.

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

Volvemos a ejecutar el test.



```
public void test_al_crear_un_coche_su_velocidad_es_cero(){  
    Coche nuevoCoche = new Coche();  
    Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);  
}  
  
new *  
@Test  
public void test_al_acelerar_un_coche_su_velocidad_aumenta(){  
    Coche nuevoCoche = new Coche();  
    nuevoCoche.acelerar( aceleracion: 30);  
    Assertions.assertEquals( expected: 30, nuevoCoche.velocidad);  
}
```

TestCoche x

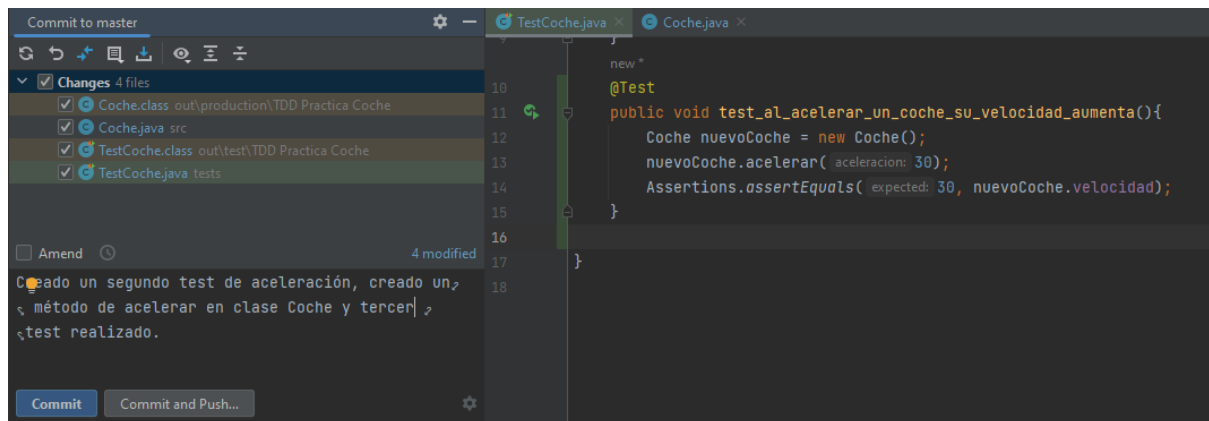
Tests passed: 2 of 2 tests - 16 ms

Test	Duration	Exit Code
TestCoche	16 ms	0
test_al_acelerar_un_coche_su_velocidad_aumenta	16 ms	0
test_al_crear_un_coche_su_velocidad_es_cero()		0

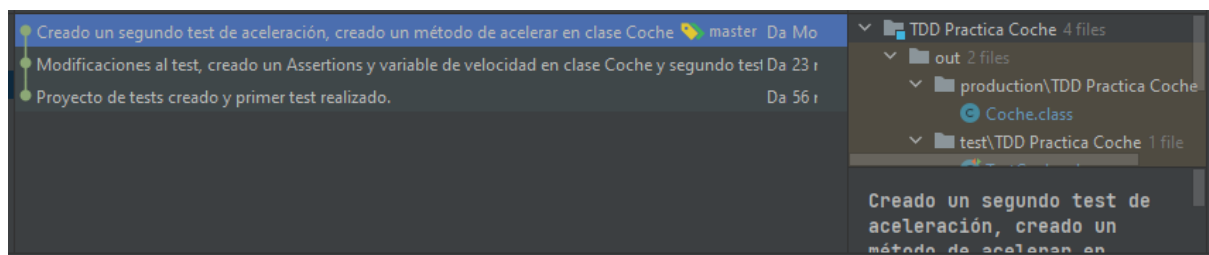
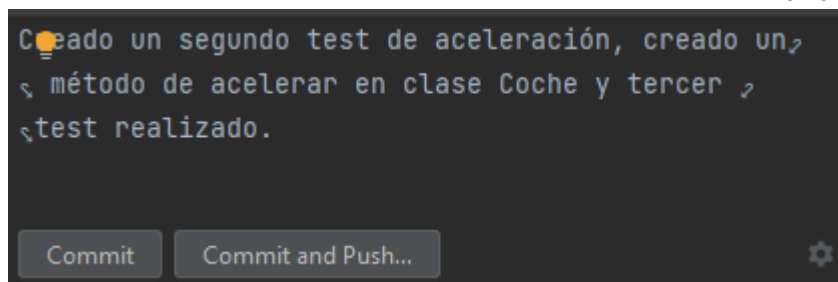
Process finished with exit code 0

Y volvemos a superarlo.

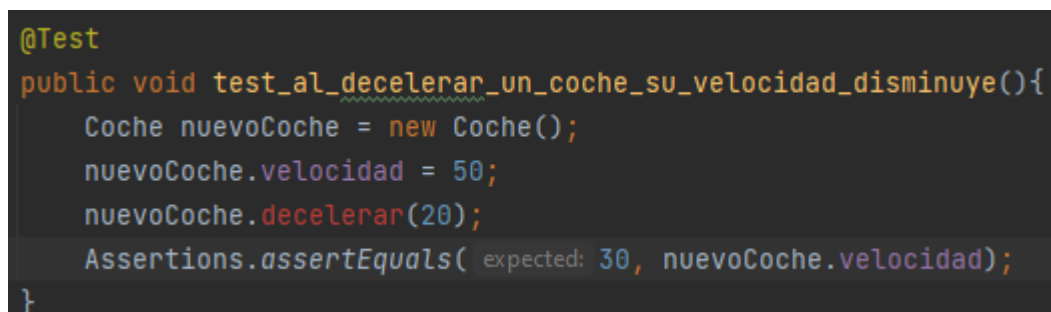
Volvemos a hacer commit de los cambios respecto al último test.



Marcamos los ficheros a hacer commit, escribimos el mensaje y pulsamos en commit.



Ahora haremos otro método en el test pero en vez de acelerar será decelerar. Podemos copiar lo que tenemos escrito y modificarlo después sin ningún problema.



De nuevo tenemos el problema del método que no existe.

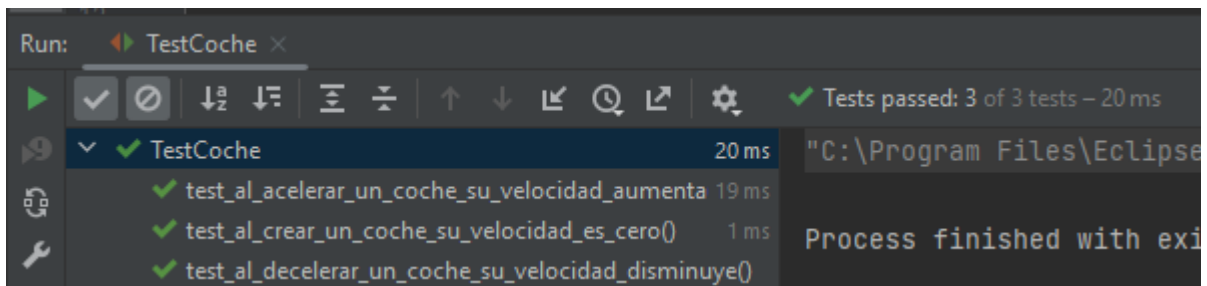
Nos ponemos en el método y hacemos Alt + Mayus + Intro

```
1 usage new *
public void decelerar(int i) {
}
```

Lo vamos a modificar de la siguiente manera:

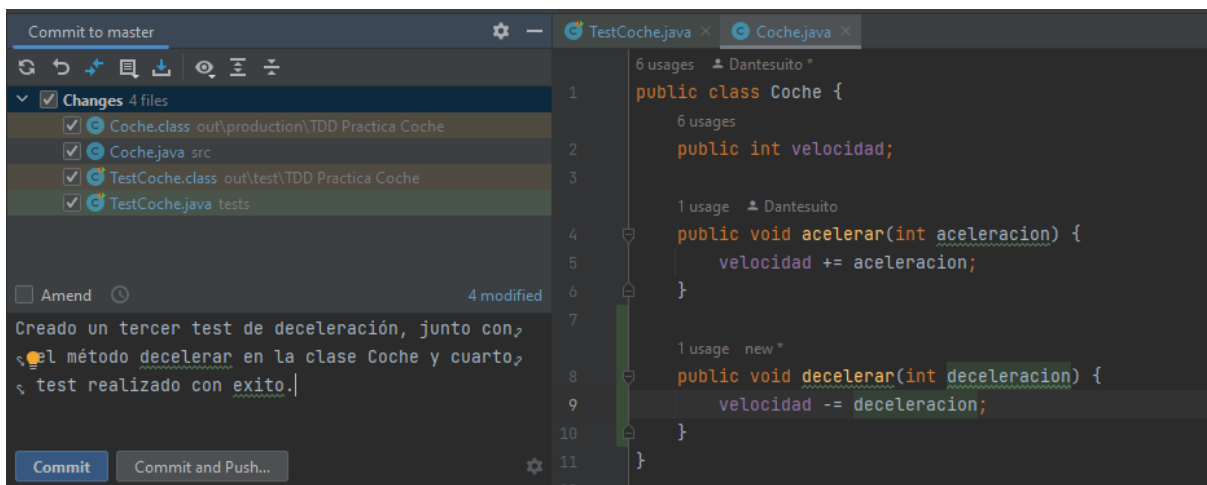
```
1 usage new *
public void decelerar(int deceleracion) {
    velocidad -= deceleracion;
}
```

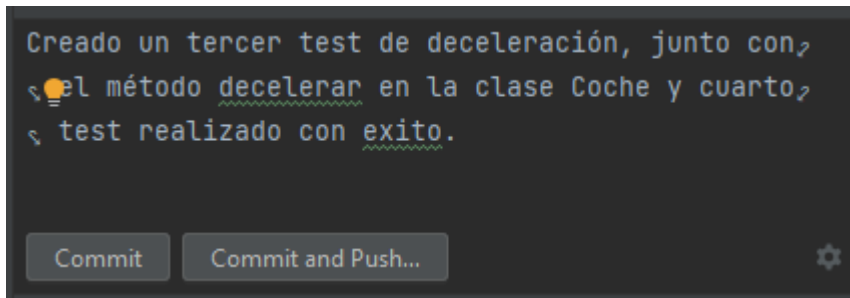
Volvemos a ejecutar el test



Pasamos el test satisfactoriamente.

Hacemos commit de los cambios nuevos.

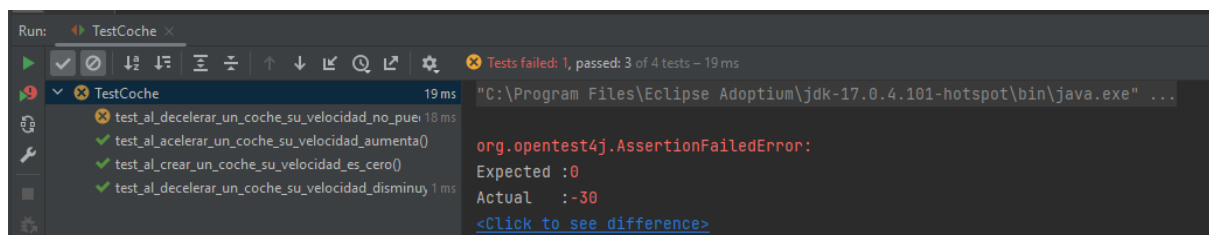




Por último vamos a crear un último test que compruebe que al frenar, no se conseguirá una velocidad negativa.

```
@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( deceleracion: 80);
    Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
}
```

Tenemos el test, y vemos que compilar bien. Vamos a ejecutarlo.

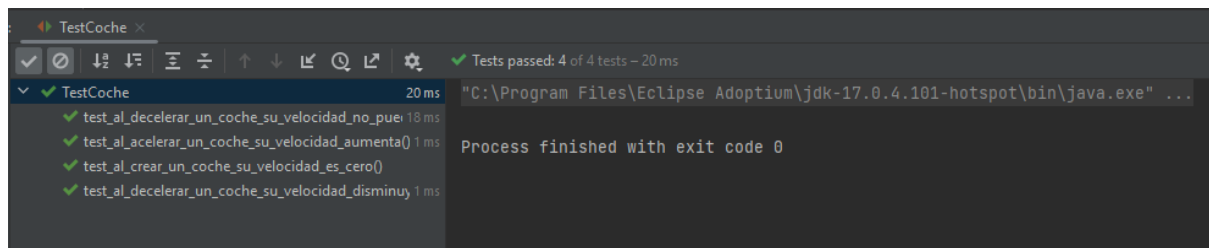


Vemos que el último nos da error, ya que se esperaba una velocidad de 0 pero realmente se ha pasado a -30.

Lo que haremos para solucionarlo será ir a la clase Coche y, en el método de decelerar, pondremos un condicionante de que si la velocidad es menor que 0, se convierta en 0.

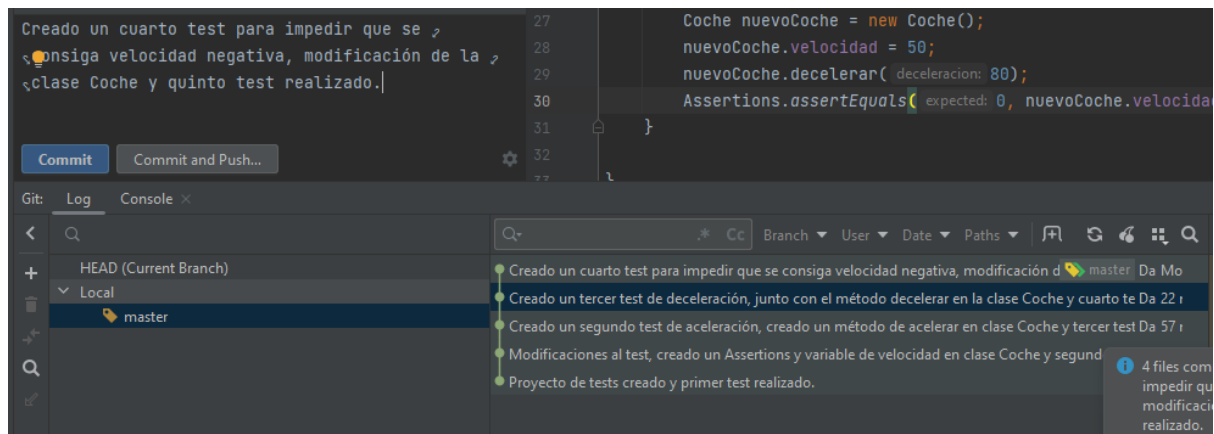
```
2 usages  Dantesuito *
public void decelerar(int deceleracion) {
    velocidad -= deceleracion;
    if (velocidad < 0){
        velocidad = 0;
    }
}
```

Ahora ejecutamos el test y deberíamos pasarlo correctamente.



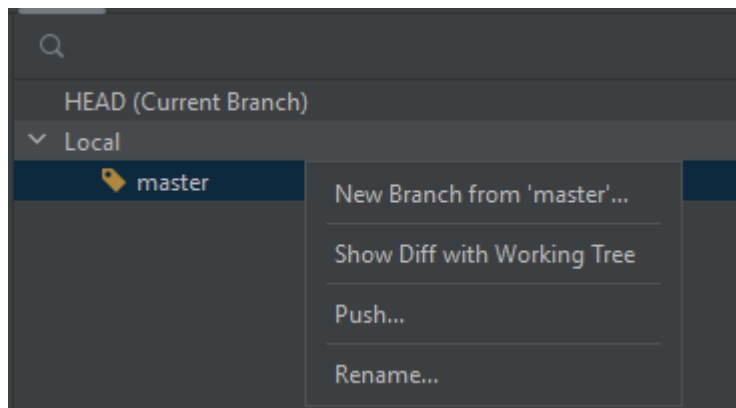
En efecto el test se realiza bien.

Vamos a hacer un commit de los cambios

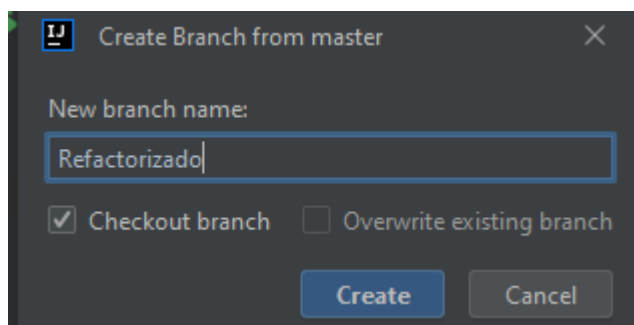


Ahora, a partir de la rama en la que estamos, tenemos que crear una nueva que se llame “Refactorizado”.

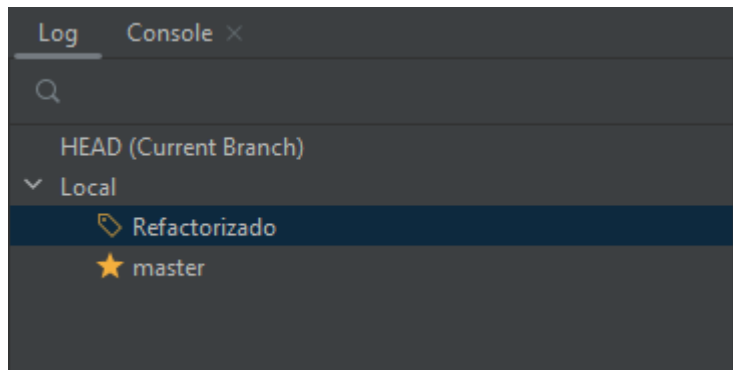
Para ello, tenemos que hacer click derecho donde pone “master” a la izquierda.



Y pulsar en la primera opción. “Nueva rama a partir de ‘master’ ”



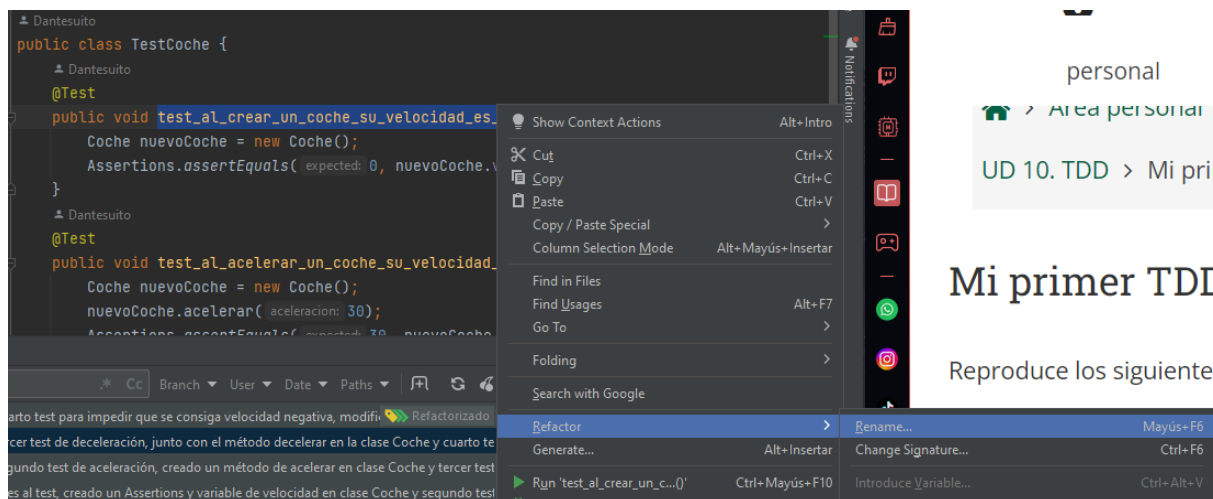
Le escribimos el nombre de la rama y, opcionalmente, podemos marcar la opción de checkout, que, al crear la rama, nos colocará en esa rama automáticamente. No es necesario, nos ahorrará unos segundos simplemente.



Vemos que ya tenemos las 2 ramas.

Ahora tenemos que, mediante refactorización, modificar el nombre de los métodos del proyecto por “nombreOriginal_tu_nombre”
Por ejemplo, si un método se llama “desatornillar”, tendremos que modificarlo para que ponga “desatornillar_danielvital” (En mi caso es Daniel Vital, en tu caso será tu nombre)

Para esto, tenemos que seleccionar el método que queremos cambiar, hacer click derecho -> Refactor -> Rename



Y hacemos que pase de esto:

```

public class TestCoche {
    Dantesuito *
    @Test
    public void test_al_crear_un_coche_su_velocidad_es_cero // 1 () {
        Coche nuevoCoche = new Coche();
        Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
    }
}

```

A esto:

```

public class TestCoche {
    Dantesuito *
    @Test
    public void test_al_crear_un_coche_su_velocidad_es_cero_Daniel_Vital_Torres // 1 () {
        Coche nuevoCoche = new Coche();
        Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
    }
}

```

Pulsamos Enter y repetimos el proceso con el resto de métodos del proyecto.

Hacemos un commit de estos últimos cambios.

The screenshot shows an IDE interface. At the top, a commit message is being entered: "Nombres de métodos refactorizados para incluir nuestro nombre en ellos." Below the message are buttons for "Commit" and "Commit and Push...". To the right, a snippet of Java code is visible, showing a method named `decelerar_Daniel_Vital_Torres`. Below the code editor, a Git log is displayed, showing a list of commits. The most recent commit is highlighted, showing the message "Nombres de métodos refactorizados para incluir nuestro nombre en ellos." and the author "Dantesuito".