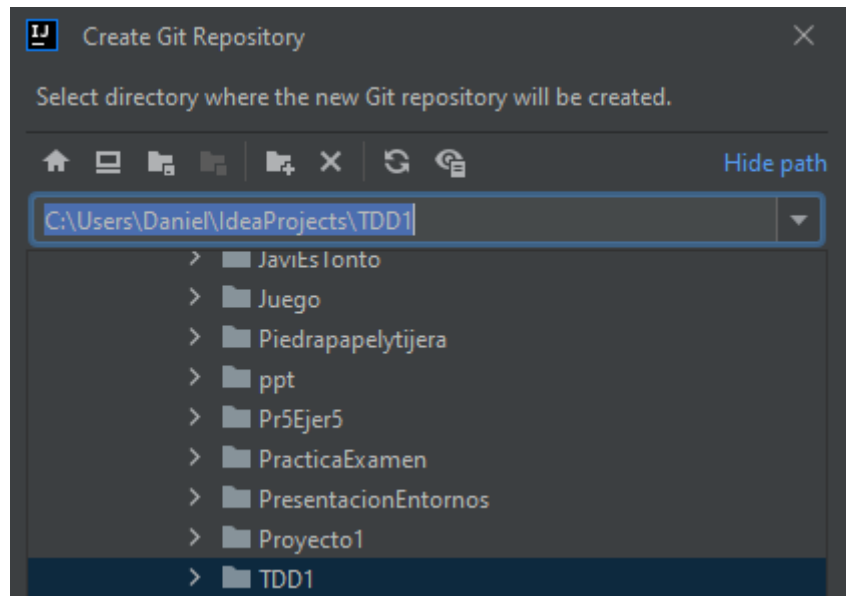


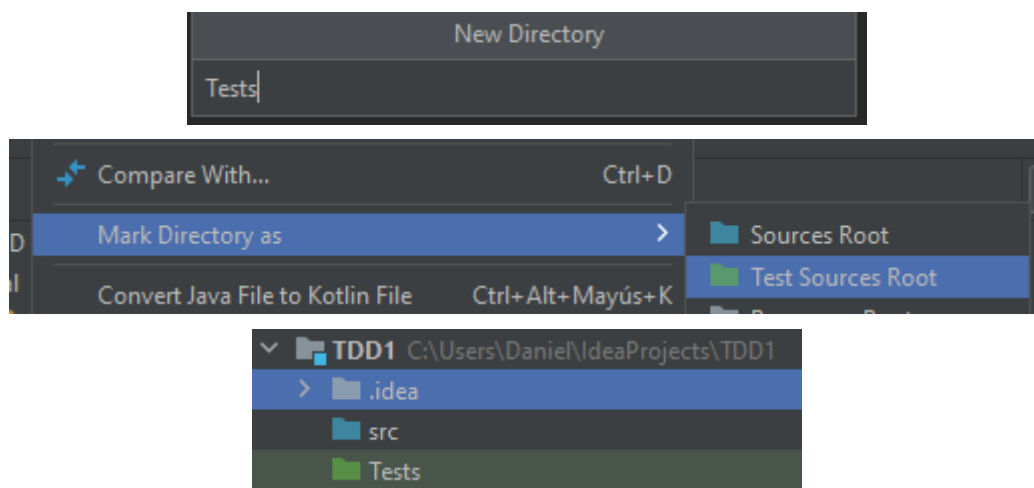
Entornos de desarrollo

Mi primer TDD

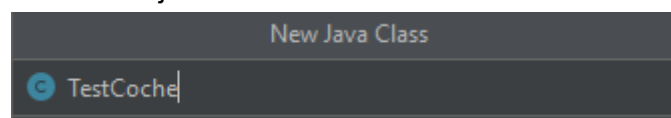
En esta práctica voy a crear un proyecto usando TDD en IntelliJ. Lo primero que vamos a hacer es activar el control de versiones en nuestro directorio de trabajo:



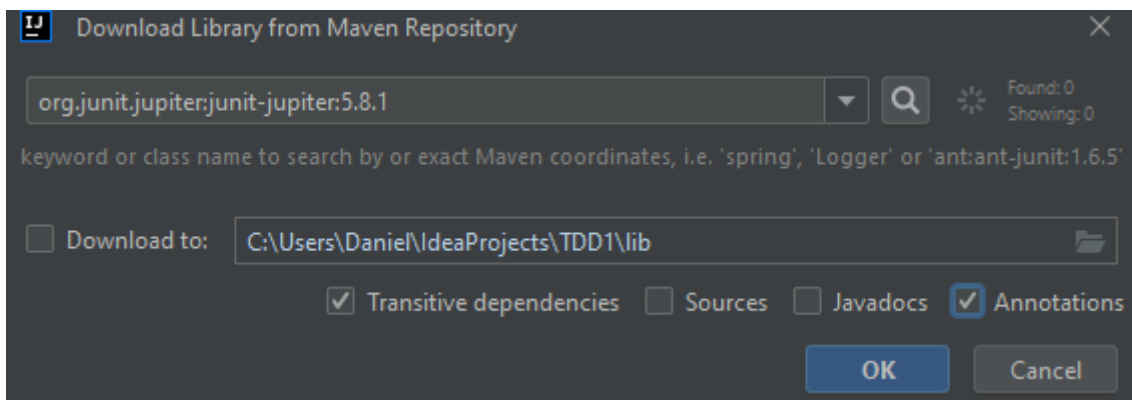
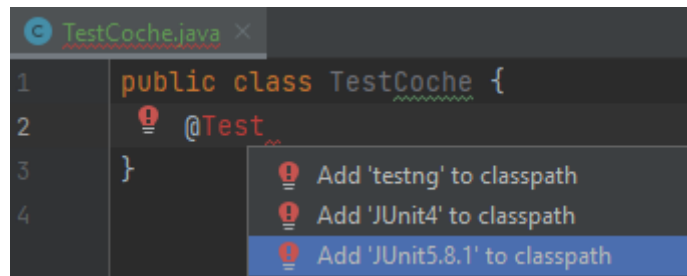
Comenzamos creando un nuevo directorio en el raíz de nuestro proyecto, que se llamará tests. Además lo vamos a marcar como “Test Sources Root” de tal forma que quede de color verde:



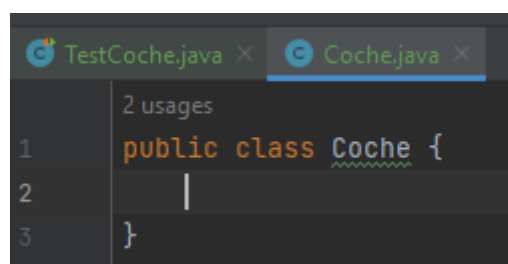
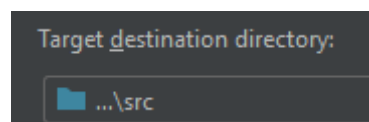
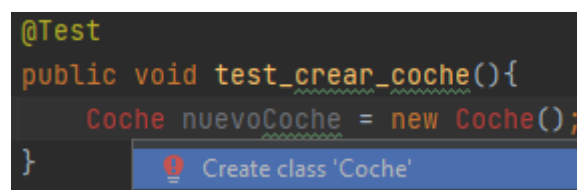
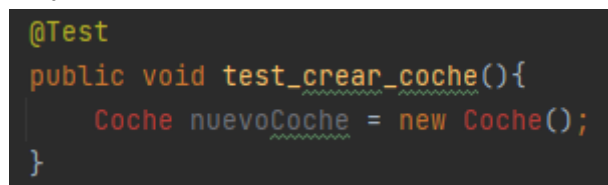
Vamos ahora a crear una clase java llamada TestCoche:



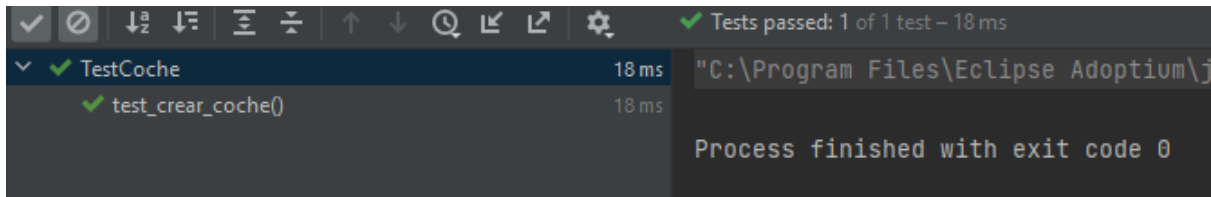
Ahora vamos a hacer un test en esta clase. Para ello, necesitamos junit. Para añadirlo a nuestro proyecto, escribimos `@Test`. Al hacer click derecho sobre el error, nos dejará solucionarlo añadiendo diferentes versiones de JUnit al proyecto. Elegimos la más reciente y la añadimos:



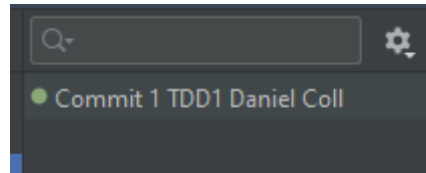
Ahora que tenemos JUnit instalado, vamos a crear nuestro primer test, que será para crear un coche. Como la clase Coche no existe, vamos a crearla una vez más haciendo click derecho en el error y seleccionando la opción adecuada (importante crear la clase en la carpeta del código fuente y no en la de los tests):



Si ejecutamos ahora mismo el test, vemos que lo pasamos (se ha podido crear el objeto):



Vamos a hacer el primer commit del proyecto:



Vamos a modificar el test que tenemos ahora mismo. Hasta ahora era un test sobre si podíamos crear un coche, ahora va a ser un test que pretenderá comprobar si la velocidad de un coche es 0 cuando se acaba de crear:

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

Importamos la clase Assertions y vamos a usar el método assertEquals, al que se le pasa un resultado esperado y luego cualquier expresión o método que queremos que dé como resultado dicho valor. En este caso, el atributo velocidad del nuevo coche que creamos:

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

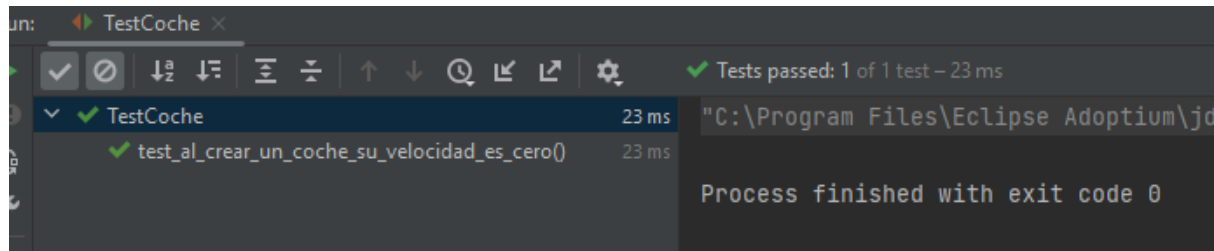
De nuevo, no existe el atributo velocidad en la clase Coche, por tanto hacemos click derecho y podemos crearlo directamente para solucionarlo:

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

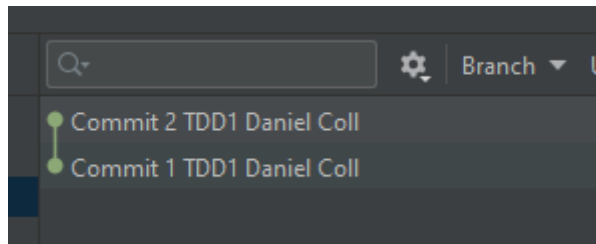
Create field 'velocidad' in 'Coche'



Ahora ejecutamos de nuevo nuestro test y podemos ver que lo pasamos satisfactoriamente:



Ahora vamos a hacer el segundo commit del proyecto:

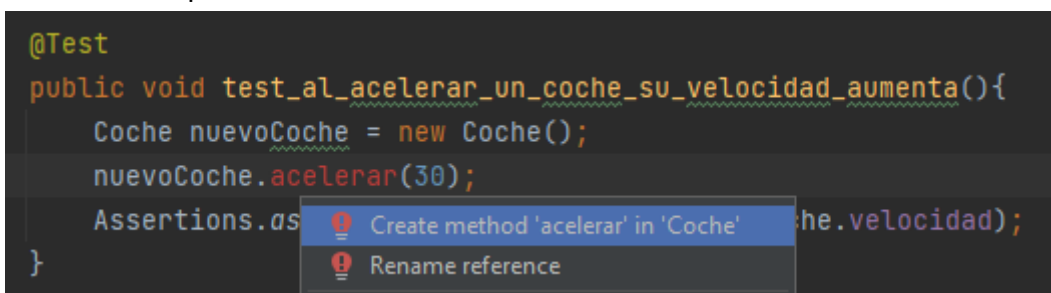


Ya hemos conseguido crear a partir de los tests la clase Coche y su atributo velocidad. Ahora vamos a hacer un test del que sacaremos un método acelerar que nos permita alterar la velocidad del coche. El test va a tener este aspecto:

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

Como ya es habitual, nos da un error porque el método acelerar no existe. Haciendo click derecho en el error podemos crear el método acelerar:

```
@Test
public void test_al_acelerar_un_coche_su_velocidad_aumenta(){
    Coche nuevoCoche = new Coche();
    nuevoCoche.acelerar(30);
    Assertions.as
}
```

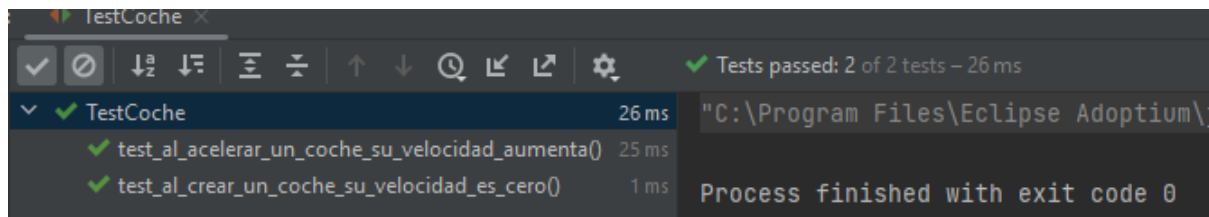


```
1 usage new *
public void acelerar(int i) {
}
```

Vamos a cambiarle un poco el aspecto al método. Para empezar vamos a llamar a la variable “aceleración” y después vamos a acumular el valor de aceleración en el atributo “velocidad”:

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

Y probamos para ver que pasamos el test satisfactoriamente una vez más:



Queremos también un método para decelerar. Para ello vamos a duplicar todo el código del test de acelerar y lo vamos a modificar de la siguiente forma:

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

new *
@Test
public void test_al_decelerar_un_coche_su_velocidad_disminuye(){
    Coche nuevoCoche = new Coche();
    nuevoCoche.acelerar( aceleración: 30);
    nuevoCoche.decelerar(20);
    Assertions.assertEquals( expected: 10, nuevoCoche.velocidad);
}
```

Repetimos el proceso para crear el método decelerar y lo editamos tal que así:

```
@Test
public void test_al_decelerar_un_coche_su_velocidad_disminuye(){
    Coche nuevoCoche = new Coche();
    nuevoCoche.acelerar( aceleración: 30);
    nuevoCoche.decelerar(20);
    Assertions.assertEquals( expected: 10, nuevoCoche.velocidad);
}
```

Create method 'decelerar' in 'Coche'

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

```

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

```

Creamos un nuevo método, esta vez queremos que la velocidad de un coche no pueda ser menor que 0, así que el test va a ser el siguiente:

```

@Test
public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_0(){
    Coche nuevoCoche = new Coche();
    nuevoCoche.acelerar( aceleración: 30);
    nuevoCoche.decelerar( deceleración: 50);
    Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
}

```

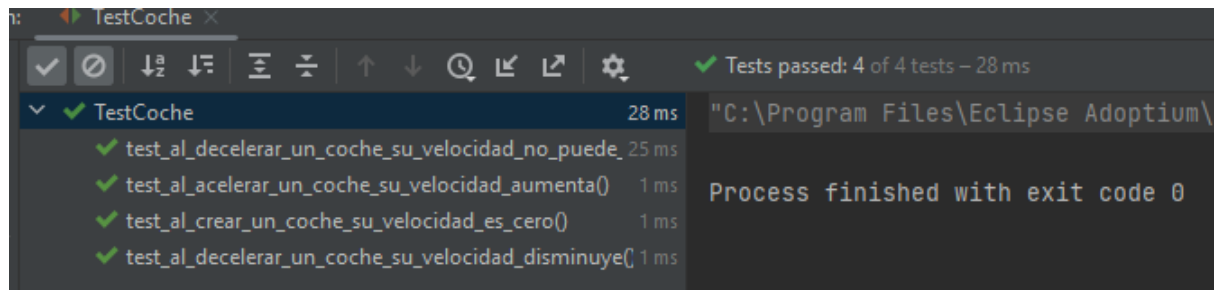
Esta vez tenemos que editar directamente el código. Vamos a buscar hacerlo de la forma en la cual la condición del test se cumpla con el menor código posible:

```

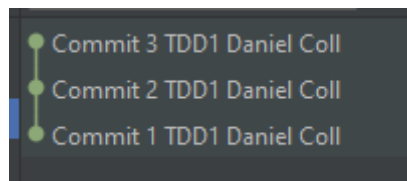
public void decelerar(int deceleración) {
    velocidad = velocidad - deceleración;
    if (velocidad<0){
        velocidad=0;
    }
}

```

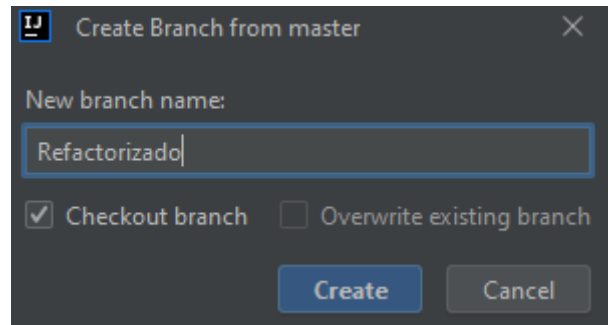
Probamos a ejecutar todos los tests y vemos que pasamos satisfactoriamente:



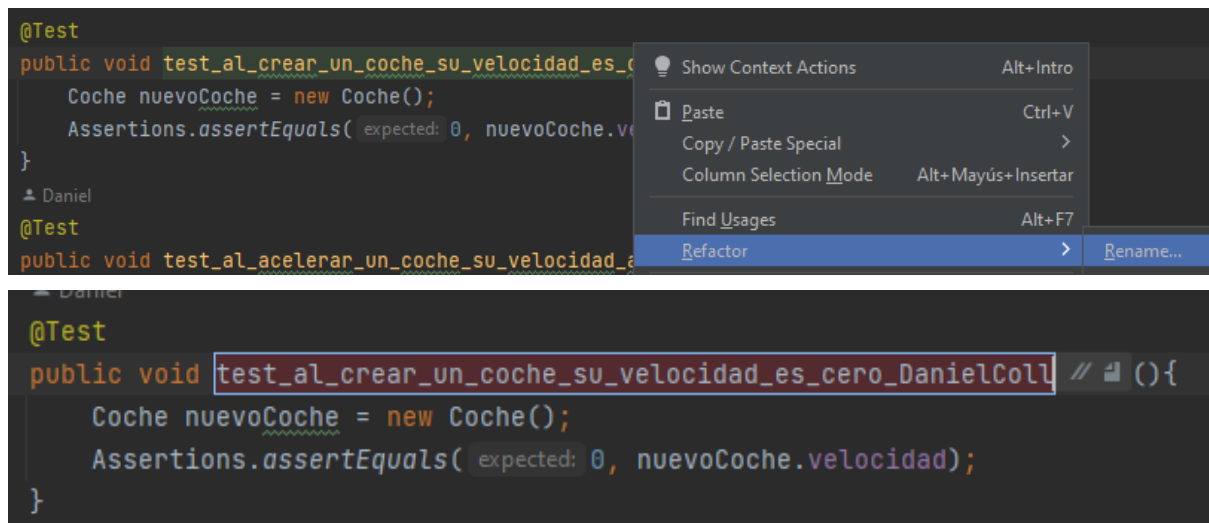
Vamos a hacer un commit de nuevo:



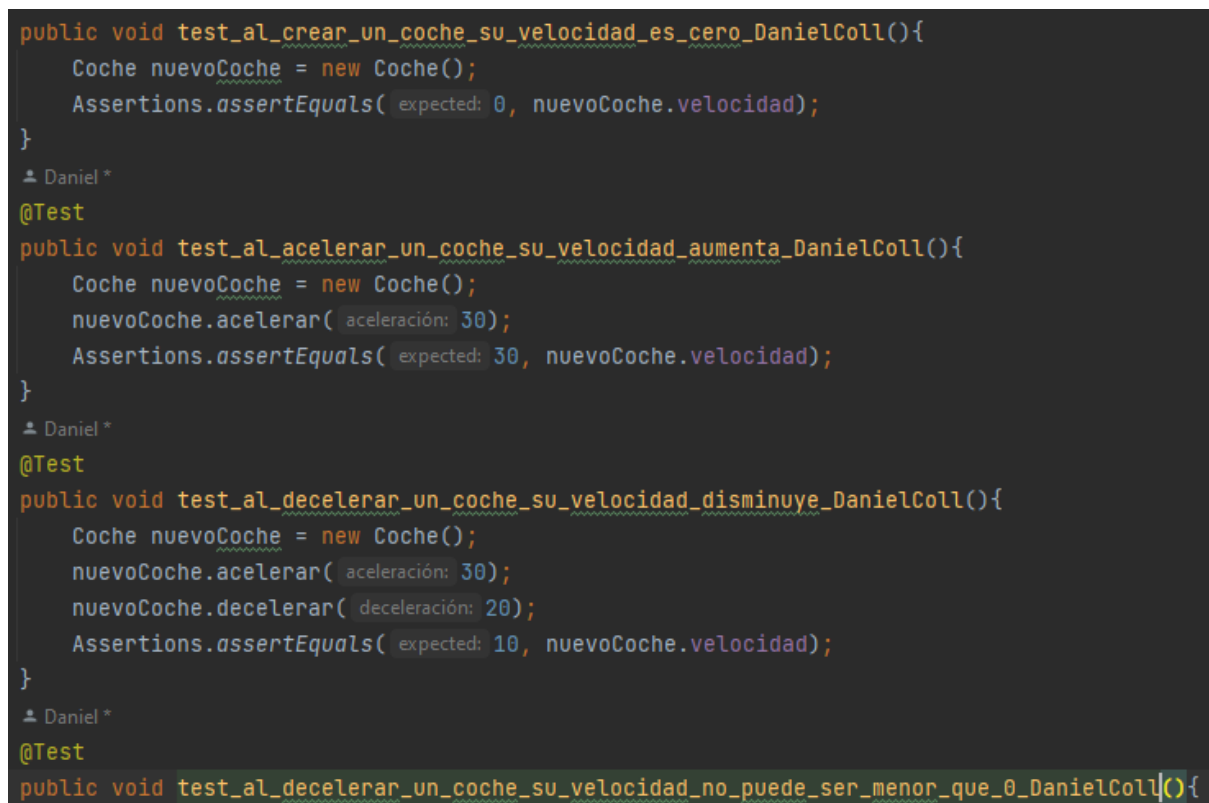
Vamos a crear una nueva rama llamada refactorizado y nos vamos a mover a ella:



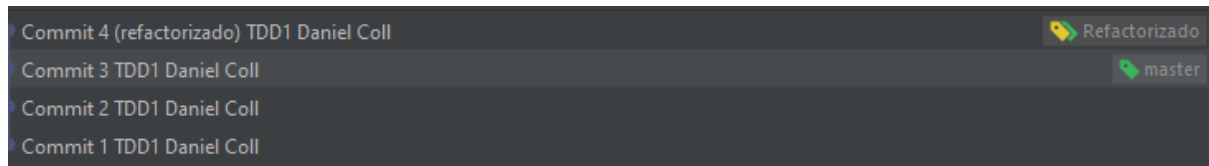
Y lo que voy a hacer en ella es refactorizar el nombre de todos los métodos para añadirle mi nombre:



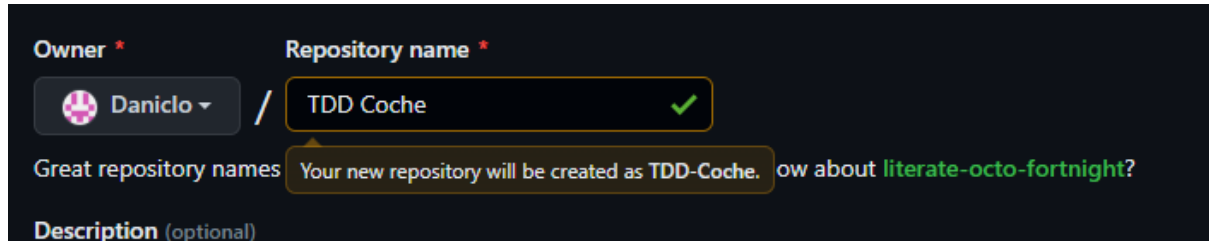
Repetimos este proceso con todos los métodos:



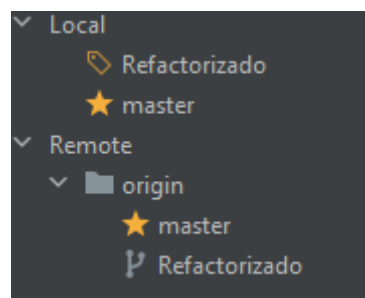
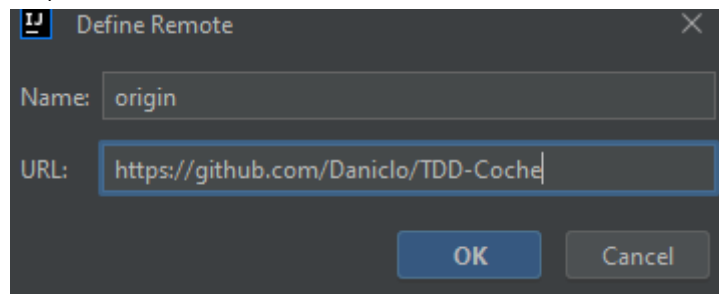
Y hacemos un commit con estos cambios:



Ahora voy a terminar la práctica subiéndola a GitHub. Primero creamos un repositorio:



Y ahora hacemos un push de ambas ramas desde IntelliJ (seleccionamos una rama y push, y luego la otra y push):



Y voy a crear una nueva rama llamada memoria en la que voy a subir este documento:

