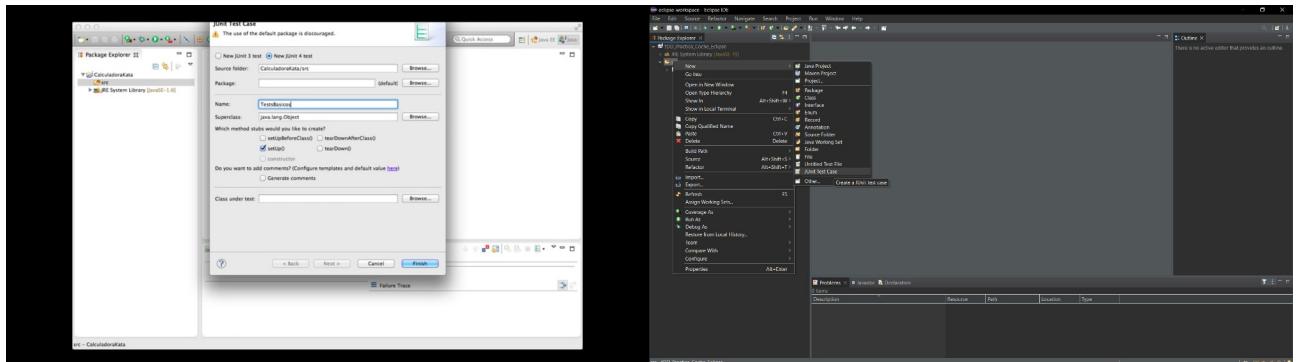
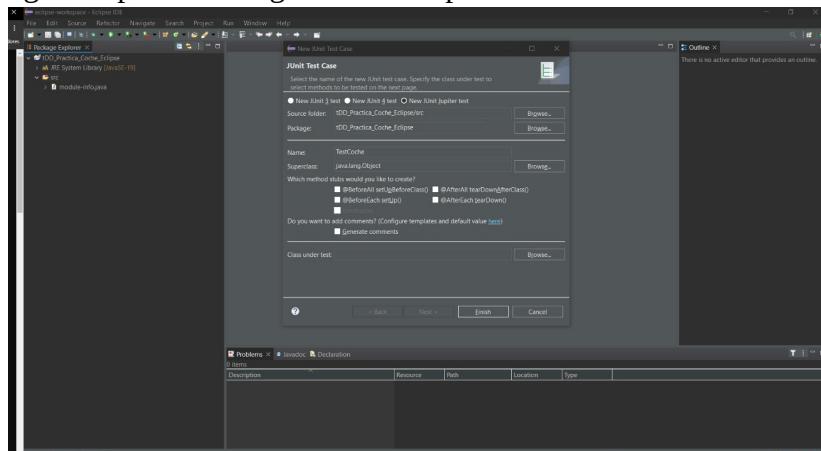


En primer lugar crearemos el proyecto y lo llamaremos tDD\_Practica\_Coche\_Eclipse

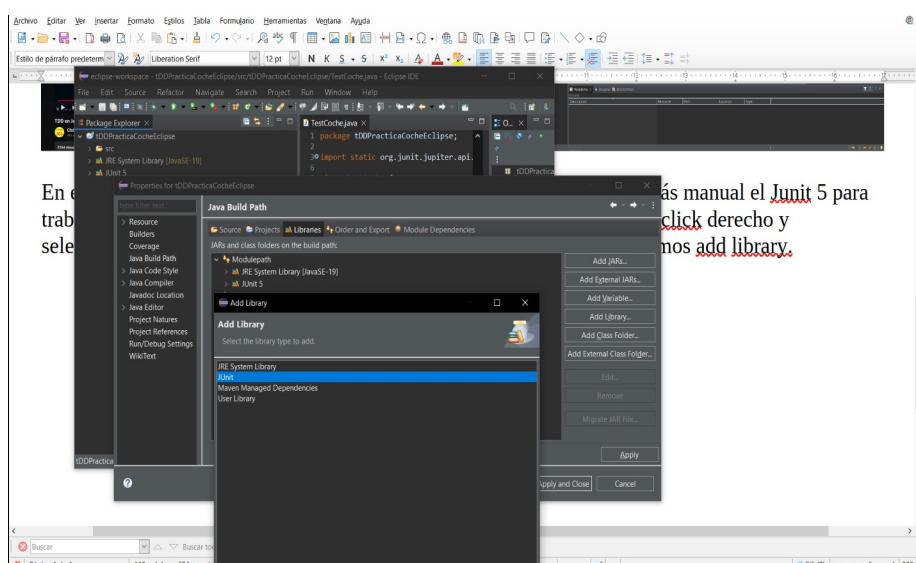
En esta ocasión el procedimiento va a ser un poco distinto, en primer lugar iremos a new y clickearemos en Junit Test Case y lo llamaremos TestCoche.



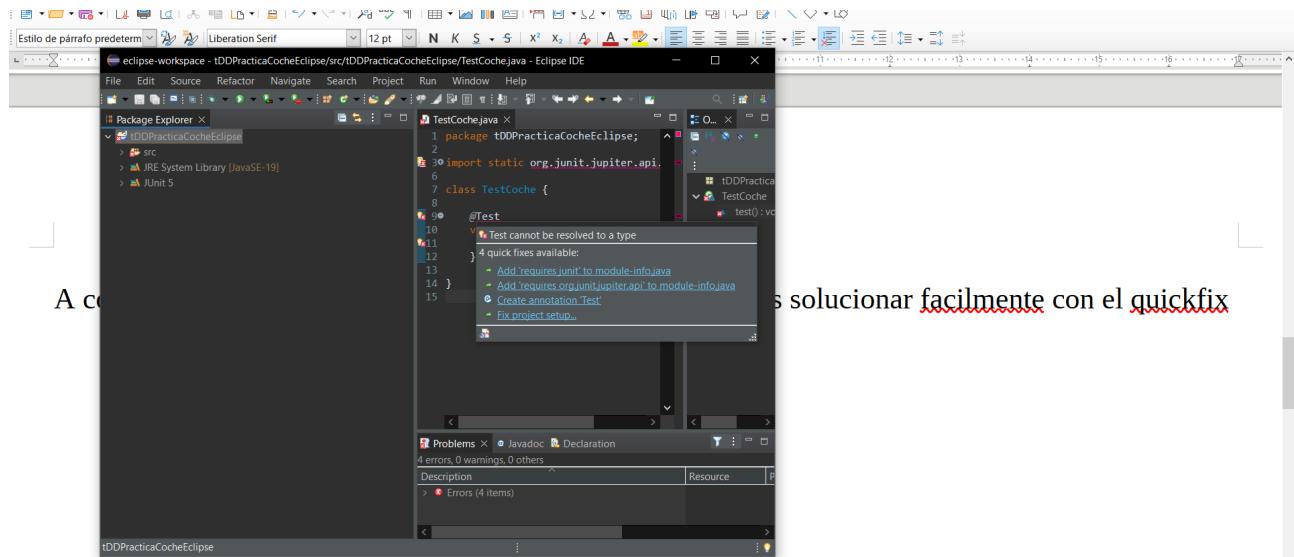
Como en la práctica anterior nos decían de usar el Junit 5 en esta ocasión utilizaremos el Junit Jupiter que según he podido indagar es lo más parecido a la versión de Junit5 de intelij.



En eclipse y a diferencia de IntelIJ necesitaremos instalar de una forma más manual el Junit 5 para trabajar con los test, para esto vamos a la carpeta del proyecto y hacemos click derecho y seleccionamos propiedades, acto seguido vamos a la librería y seleccionamos add library y vamos a seleccionar Junit5 le damos a next lo seleccionamos y acto seguido pulsamos Apply and close.



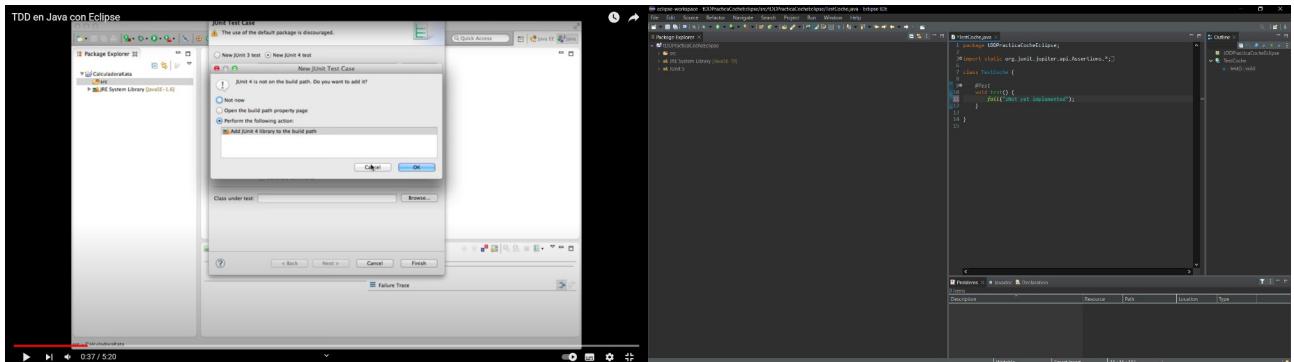
A continuación se generará un error por defecto que podemos solucionar fácilmente con el quickfix, seleccionamos la segunda opción que aparece en la foto y el error habrá desaparecido



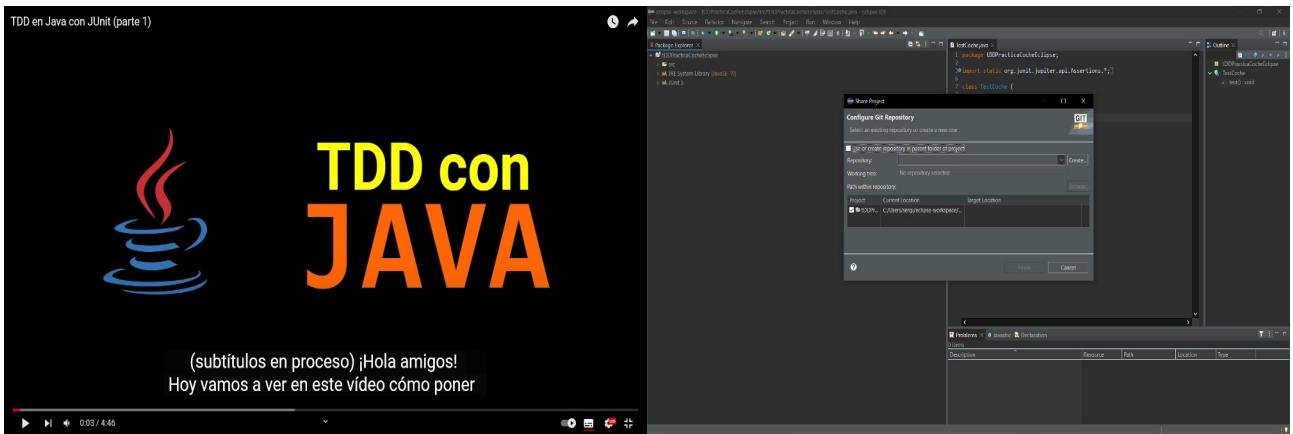
A continuación se muestra como solucionar fácilmente con el quickfix



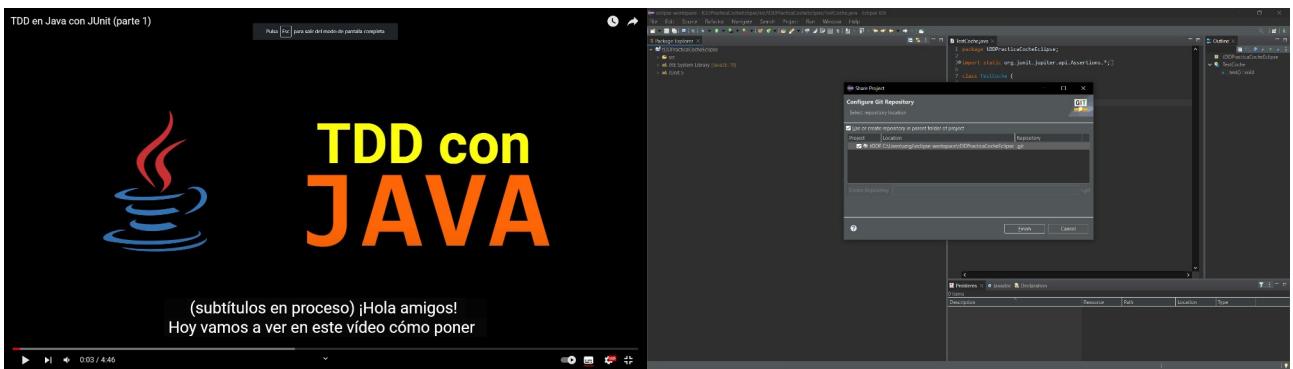
Como vemos en la siguiente imagen el error ha desaparecido.



A continuación vamos a proceder a crear el repositorio de git para poder realizar los commits conforme vayamos realizando cambios en nuestro código, para ello hacemos click derecho sobre la carpeta del proyecto y seleccionamos Team, share project en el menú desplegable.

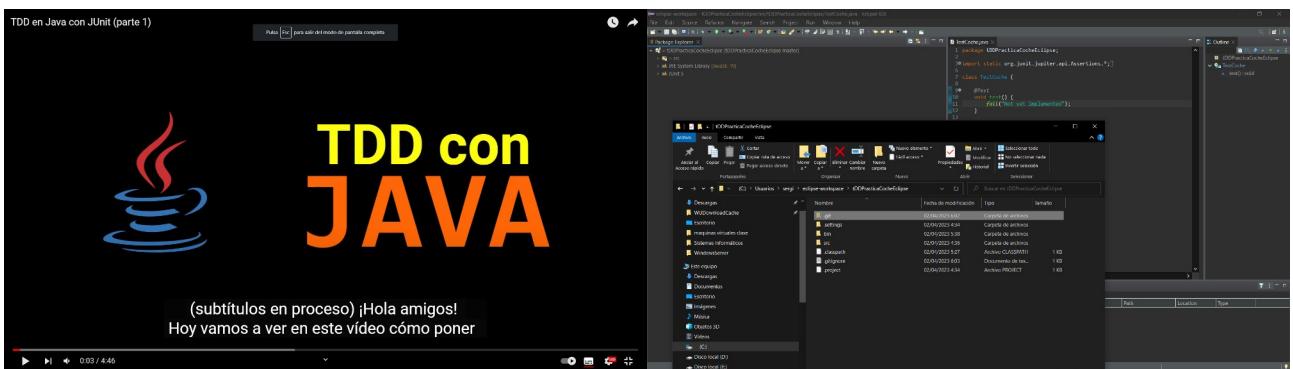


A continuación hacemos click y marcamos use or create repository in parent folder of project y clickeamos en la carpeta del proyecto, acto seguido pulsaremos create Repository.



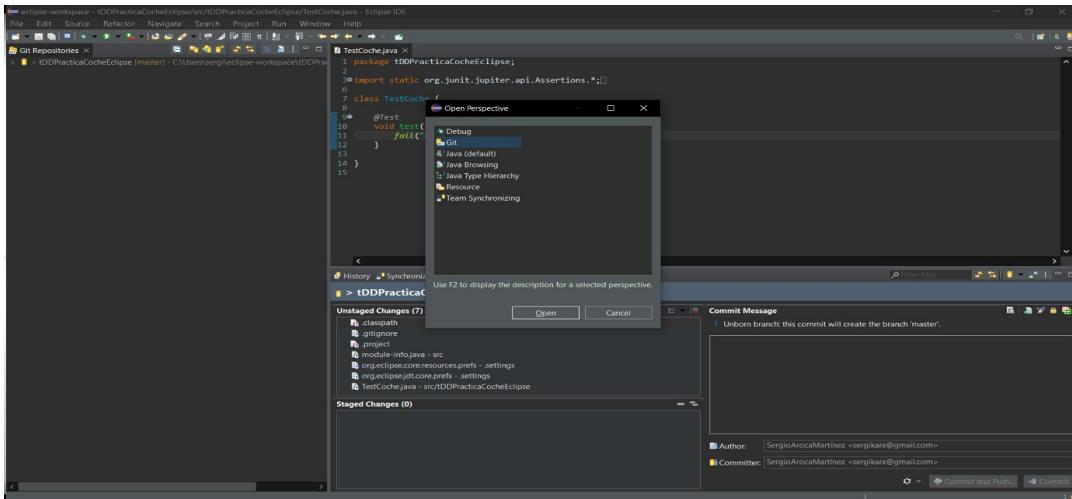
Como vemos ahora se ha marcado la casilla de la carpeta la cual no será seleccionable hasta que creamos el Repositorio, a continuación pulsaremos en Finish.

Como vemos el repositorio se ha creado correctamente.

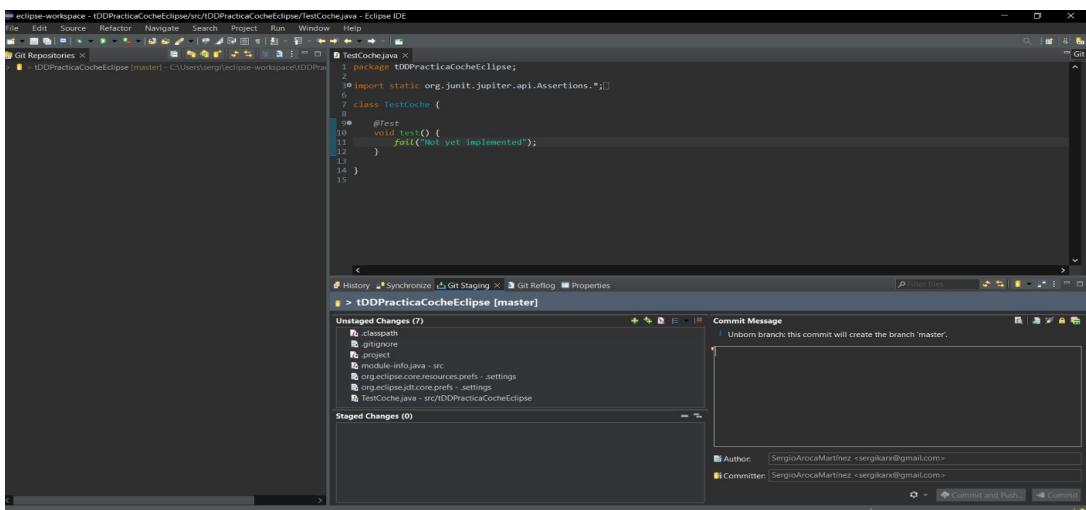


Ahora para utilizar la interfaz de git en Eclipse utilizaremos el open perspective(arriba a la derecha, los cuadraditos donde está posicionado el ratón en la imagen) y añadiremos el de Git seleccionándolo y pulsando open.

Como vemos ahora arriba a la derecha junto al icono de java ahora nos sale el de Git, si



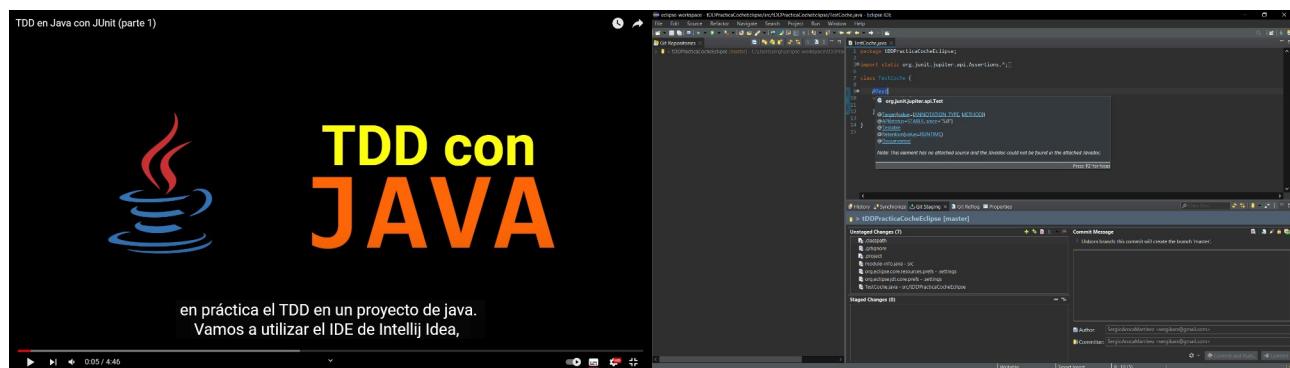
cliqueamos en este icono accederemos a la interfaz grafica de git para poder hacer los commit, push y todo lo relacionado con nuestro futuro repositorio.



Ahora que he

conseguido instalar JUnit5 y configurar Git procederé a replicar la práctica anterior en eclipse.

Una vez creada la clase TestCoche indicaremos que es un test mediante el @Test,



A continuación dentro de TestCoche crearemos test\_crear\_coche y crearemos una clase Coche que como vemos todavía no existe.

Procedemos a crear la clase coche, en lugar de picar código la crearemos directamente con el quickfix de Eclipse.

Una vez con nuestra clase Coche creada y con todo el código correctamente escrito procedemos a ejecutar nuestro primer test.

Como vemos al compilar y ejecutar el código a la derecha nos sale una ventana de Junit que nos indica que el test se ha pasado correctamente sin errores, los warning de abajo los ignoraremos porque son indicaciones de variables no utilizadas.

A continuación procedemos a realizar nuestro primer commit, después de realizar el test correctamente sin errores procederemos a mover los archivos de unstaged a staged clickeando en la interfaz de git anteriormente configurada y hacemos un commit.

The screenshot shows the Eclipse IDE interface. On the left, the Project Explorer displays a Java project named 'TDD Practice Coche'. The 'src' folder contains a package 'tests' which has a class 'TestCoche'. The code in 'TestCoche.java' is:

```
import org.junit.jupiter.api.Test;
public class TestCoche {
    @Test
    public void test_crear_coche(){
        Coche nuevoCoche = new Coche();
    }
}
```

The 'Run' view at the bottom shows a successful test run with 1 test passed in 23 ms. The terminal output shows the command used: "C:\Program Files\Java\jdk-13.8.2\bin\java.exe" ...

In the top right, the Git reflog window shows a single commit:

```
commit 7776030 (HEAD) Commit message: 2023-04-07 20:09:00 +0000
```

This screenshot is identical to the one above, showing the same code, test results, and commit in the Git reflog.

Como vemos en la ventana de Git reflog el commit se ha realizado correctamente.

A continuación vamos a mejorar el test y modificaremos el método crear coche.

Para el nuevo método importaremos la clase Assertions porque necesitaremos una aserción y crearemos nuestro nuevo método.

The screenshot shows the Eclipse IDE interface. The 'src' folder in the Project Explorer now includes an 'assertions' folder. The 'tests' package contains the 'TestCoche' class. The code in 'TestCoche.java' is:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
public class TestCoche {
    @Test
    public void test_al_crear_un_coche_su_velocidad_es_cero(){
        Coche nuevoCoche = new Coche();
        Assertions.assertEquals(expected 0, nuevoCoche.velocidad);
    }
}
```

The Java editor highlights the word 'velocidad' with a red underline, indicating it cannot be resolved. A tooltip suggests creating a field. The 'Run' view shows the test passed. The terminal output is: "C:\Program Files\Java\jdk-13.8.2\bin\java.exe" ...

In the top right, the Git reflog shows the same commit as before.

Como vemos el símbolo velocidad del nuevo método no existe por lo tanto lo crearemos.

The screenshot shows the Eclipse IDE interface. The 'src' folder now includes an 'assertions' folder and a 'models' folder. The 'tests' package contains the 'TestCoche' class. The code in 'TestCoche.java' is:

```
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.Test;
public class TestCoche {
    @Test
    public void test_al_crear_un_coche_su_velocidad_es_cero(){
        Coche nuevoCoche = new Coche();
        Assertions.assertEquals(expected 0, nuevoCoche.velocidad);
    }
}
```

The Java editor highlights the word 'velocidad' with a red underline, indicating it cannot be resolved. A tooltip suggests creating a field. The 'Run' view shows the test passed. The terminal output is: "C:\Program Files\Java\jdk-13.8.2\bin\java.exe" ...

In the top right, the Git reflog shows the same commit as before.

Ahora procederemos a pasar un segundo test.

The screenshot shows the Eclipse IDE interface. In the top center, there's a code editor with the file `Coche.java` open, containing a simple class definition:

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

Below the code editor is the `Run` view, which shows the results of a test run named `TestCoche`. It indicates that 1 test passed in 23 ms. The command used was `"C:\Program Files\Java\jdk-13.0.2\bin\java.exe" ...`. The message in the console output is:

propone crearlo vamos a crear y ya nos lo propone que sea un entero lo cual me parece correcto

Como vemos el test se ha pasado correctamente por lo que ahora procederemos a realizar un segundo commit.

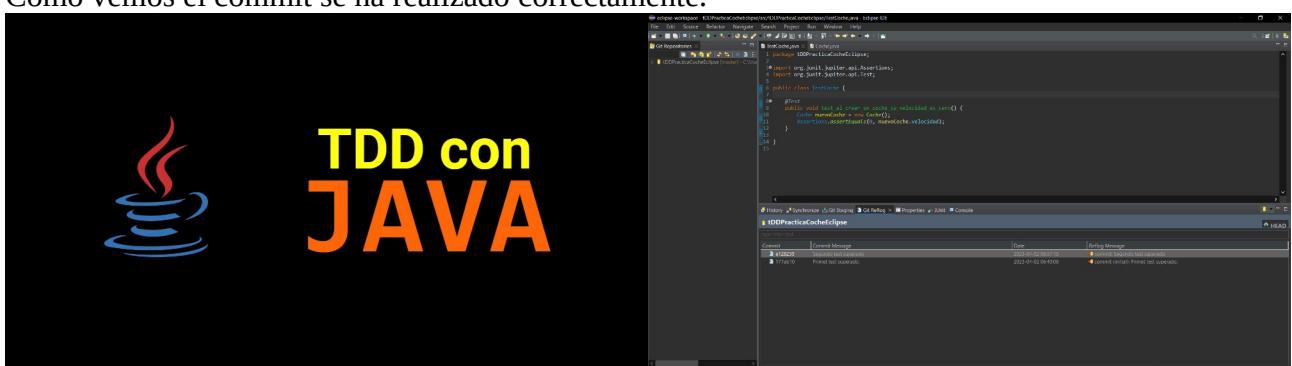
The screenshot shows the Eclipse IDE interface with the `Git Repositories` view open. A commit message is being typed into the `Commit Message` field:

Segundo commit

The commit message history shows two previous commits:

- Segundo commit - 2020-01-02 09:57:05
- Primer commit - 2020-01-02 09:49:05

Como vemos el commit se ha realizado correctamente.



A continuación crearemos el método acelerar para cambiar la velocidad del coche.

```

public class Coche {
    public int velocidad;
}

public void acelerar(int velocidad) {
}

```

Como vemos acelerar no existe por lo que lo crearemos como hemos hecho anteriormente con velocidad.

```

public class Coche {
    public int velocidad;
}

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

```

A continuación pasaremos otro test y realizaremos un tercer commit.

Como vemos ha pasado el de nuevo el test correctamente por lo que ahora es momento de realizar el tercer commit.

```

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

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

```

Commit Message: commit Segundo commit

Build Status: Success 0 / 0 Failures 0

A continuación crearemos un tercer método para decelerar.

Al igual que anteriormente nos falta decelerar en la clase Coche, por lo que habrá que crearlo como anteriormente hemos hecho.

Como vemos se ha pasado el test correctamente por lo que es momento de realizar de nuevo un commit.

Por último crearemos un nuevo test que va a servir para que al decelerar un coche su velocidad no pueda ser menor que 0.

```

public class Coche {
    public int velocidad;
    public void acelerar(int aceleracion) {
        velocidad += aceleracion;
    }
    public void decelerar(int deceleracion) {
        velocidad -= deceleracion;
    }
}

@Test
public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero() {
    Coche nuevoCoche = new Coche();
    nuevoCoche.decelerar(28);
    Assertions.assertEquals(expected: 0, nuevoCoche.velocidad);
}

```

The screenshot shows the Eclipse IDE interface with two tabs open: 'Coche.java' and 'TestCoche.java'. The 'TestCoche.java' tab contains a single test method. The code for 'Coche.java' is also visible. The status bar at the bottom indicates a failure: 'Tests failed: 1, passed: 3 of 4 tests'.

**Nota:** Aquí en teoría debería de darme un fallo el test por no tener un condicional que indique que la velocidad no pueda ser menor que 0 en la clase Coche, pero como he copiado el método de la práctica anterior (la realizada en IntelliJ) lo tengo ya y no me ha dado error, como es exactamente lo mismo no le daré importancia ya que la práctica se base en el funcionamiento del TDD y no en el código en si.

```

public class Coche {
    public int velocidad;
    public void acelerar(int aceleracion) {
        velocidad += aceleracion;
    }
    public void decelerar(int deceleracion) {
        velocidad -= deceleracion;
    }
}

@Test
public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_cero() {
    Coche nuevoCoche = new Coche();
    nuevoCoche.decelerar(28);
    Assertions.assertEquals(expected: 0, nuevoCoche.velocidad);
}

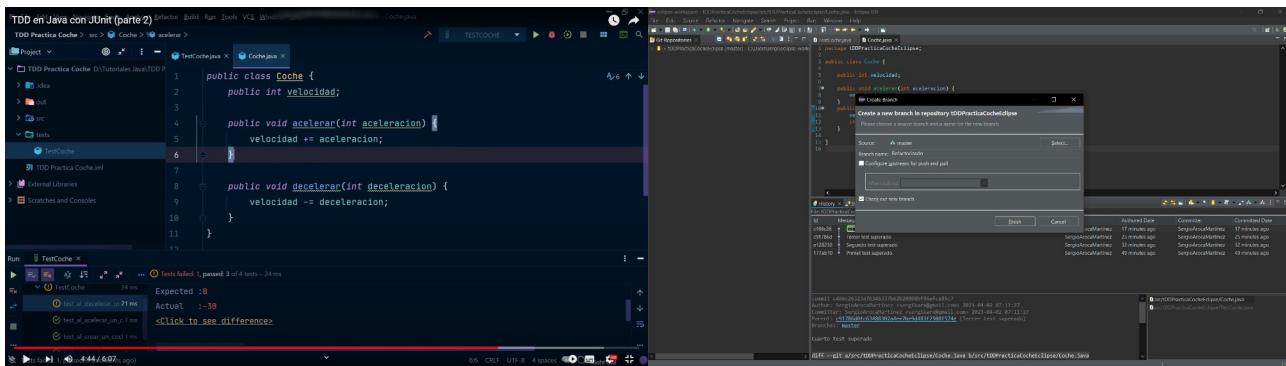
```

The screenshot shows the Eclipse IDE interface with the same setup as before. The test failure details are displayed in the bottom right pane:

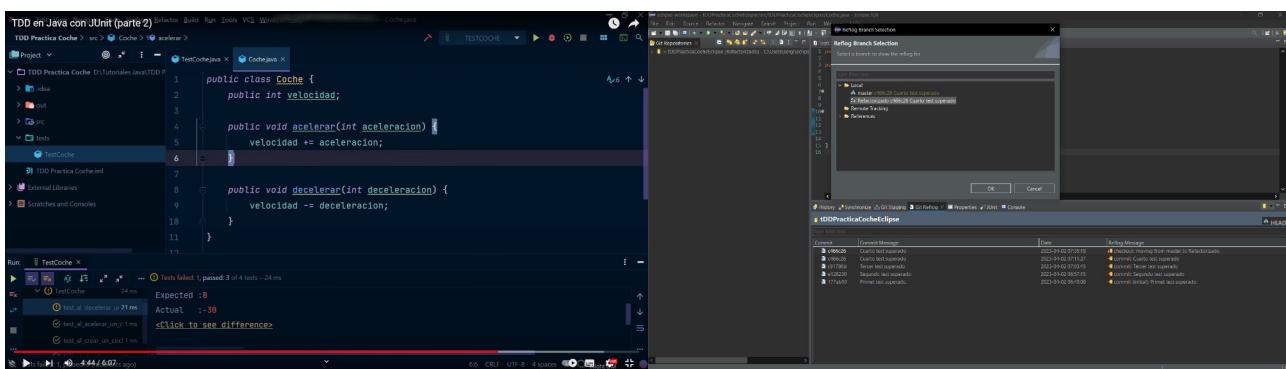
- Actual: -38
- Expected: 0
- Diff: -38

A continuación como indica el ejercicio añadiremos una nueva rama haciendo click derecho sobre la rama master y seleccionando create branch, esta nueva rama la llamaremos Refactorizado y nos posicionaremos en ella, después añadiré el nombre a los Test método creados y subiremos las dos ramas.

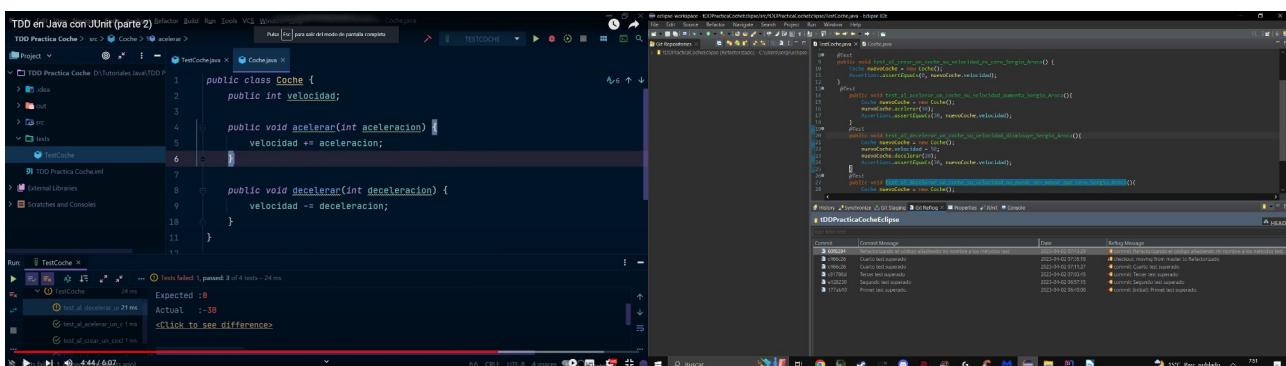
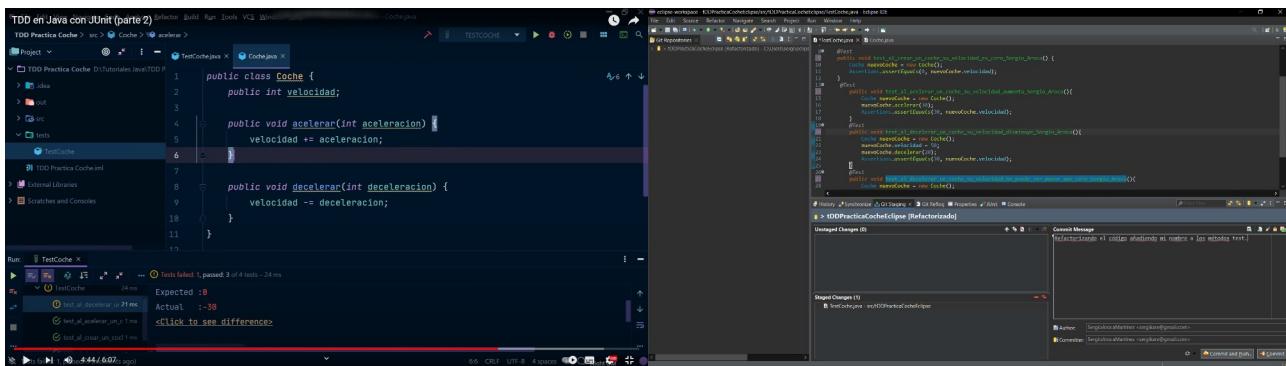
The screenshot shows the Eclipse IDE interface with the same Java code and failing test case. A context menu is open over the 'master' branch in the Git Repositories view, with 'Create Branch...' selected. The commit history and file differences are visible in the bottom right pane.



Como vemos a continuación estamos posicionados sobre la nueva rama Refactorizado que se ha creado correctamente a partir de la rama máster después del cuarto test.

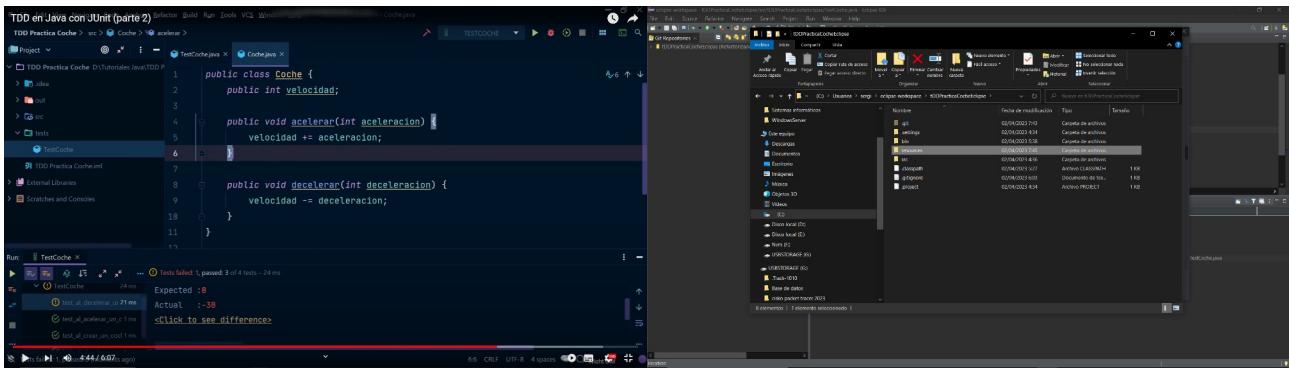


Como pide el ejercicio a continuación procederé a refactorizar el código añadiendo mi nombre a los métodos y realizaré un commit, acto seguido subiré las dos ramas.



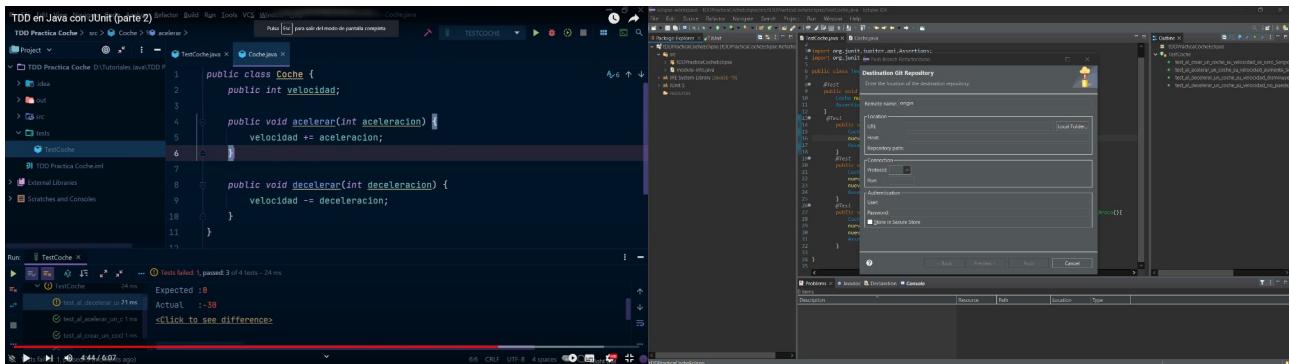
Después de crear la rama Memoria subiré esta última rama, no sin antes crear una carpeta resources

donde alojaré el pdf, también subiré la rama Refactorizando y la rama master a mi repositorio de github.

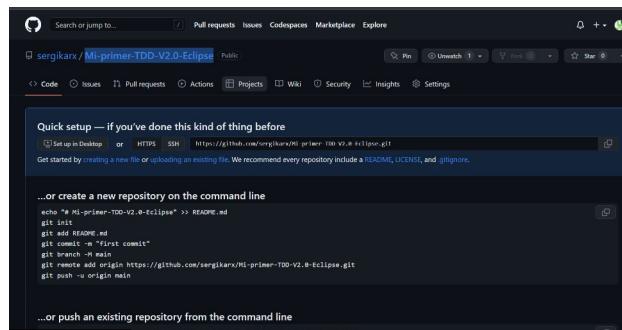


Para hacer un pull a nuestro repositorio de Github en eclipse pulsaremos sobre la vista Java (arriba a la izquierda) y en el explorador de paquetes seleccionaremos nuestro proyecto y haremos click derecho sobre el.

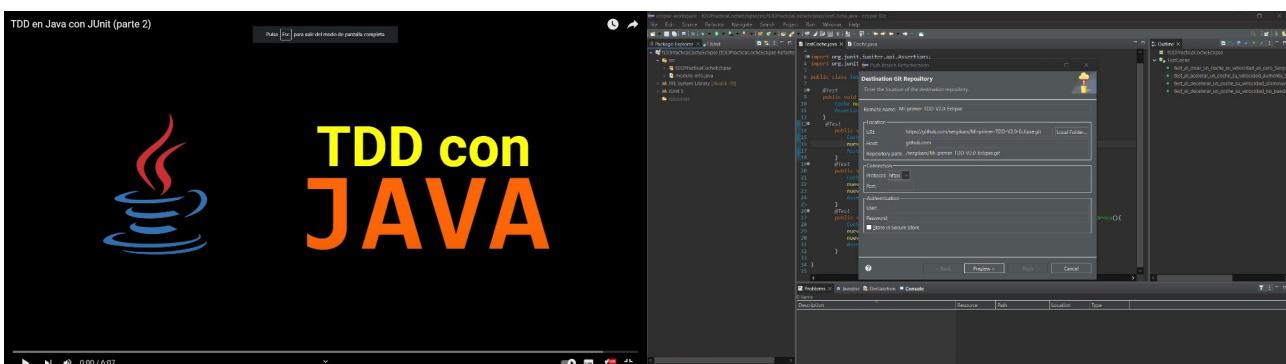
Se nos abrirá un menú desplegable, vamos a team y seleccionamos Push branch



Aquí procederemos a poner nuestro remote name que es el nombre del repositorio, en mi caso será el siguiente:



En URI pondremos el link a nuestro repositorio.



Le damos a preview y en branch elegiremos la rama en este caso Refactorizando.

