ENTORNOS DE DESARROLLO

UD 10: Mi primer TDD V2.0

Francisco Salar Pérez 1º S

Mi primer TDD V2.0

Después de realizar la práctica anterior ya debes haber entendido y practicado el concepto de desarrollo TDD, por lo tanto vamos a seguir ampliando nuestras habilidades y conocimientos.

Vuelve a realizar la práctica anterior pero esta vez con el entorno de desarrollo ECLIPSE.

Si no lo tienes instalado puedes descargar la versión que necesites desde aquí:

https://www.eclipse.org/downloads/packages/release/kepler/sr1/eclipse-ide-java-developers

Crea un nuevo repositorio, sus tres ramas y ve realizando commits cuando lo estimes necesario en cada una de las ramas de trabajo. Sobre todo ponle mucho esmero a la redacción de la memoria teniendo en cuenta que los botones y opciones no están situados en las mismas posiciones que en IntelliJ.

Realiza una memoria de los pasos que vas realizando conforme avanzas en la práctica.

Es parte de la práctica (la más importante) que navegues por este nuevo entorno y reportes en la memoria las diferencias respecto al que hemos estado utilizando de forma más habitual. Esta memoria debe incluir pantallazos apoyados con texto de los pasos que vas realizando, botones que pulsas, opciones de menús, etc. etc. etc.

Entrega de la práctica:

Copia un enlace al repositorio gitHub donde has subido la práctica.

Asegúrate de que cada rama contiene lo que se pide.

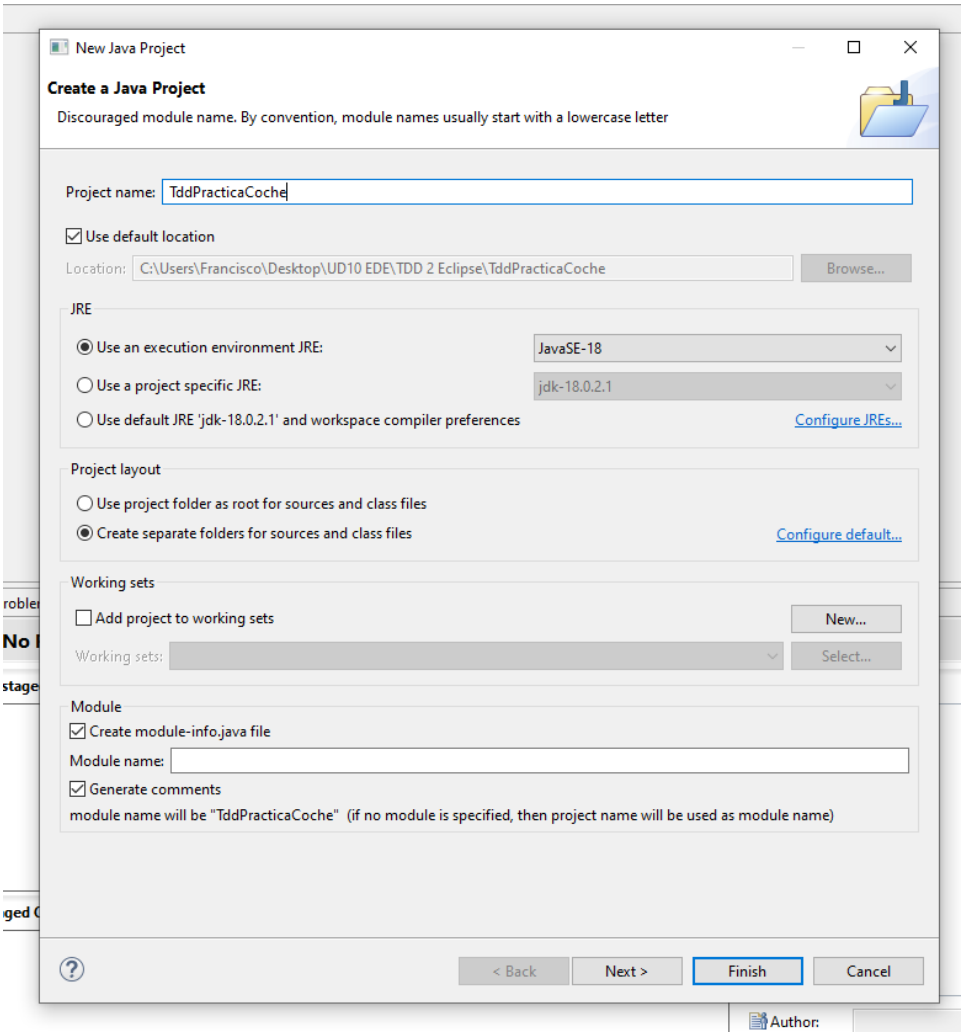
Crea una rama que se llame "Memoria" y sube a esta rama la memoria realizada. *** FORMATO PDF O WORD ***

Tiempo estimado para realizar la práctica: 2h:15'. Los 30 minutos extra son por si tienes que instalar ECLIPSE y puesto que hay que bichear un poquito por los menús.

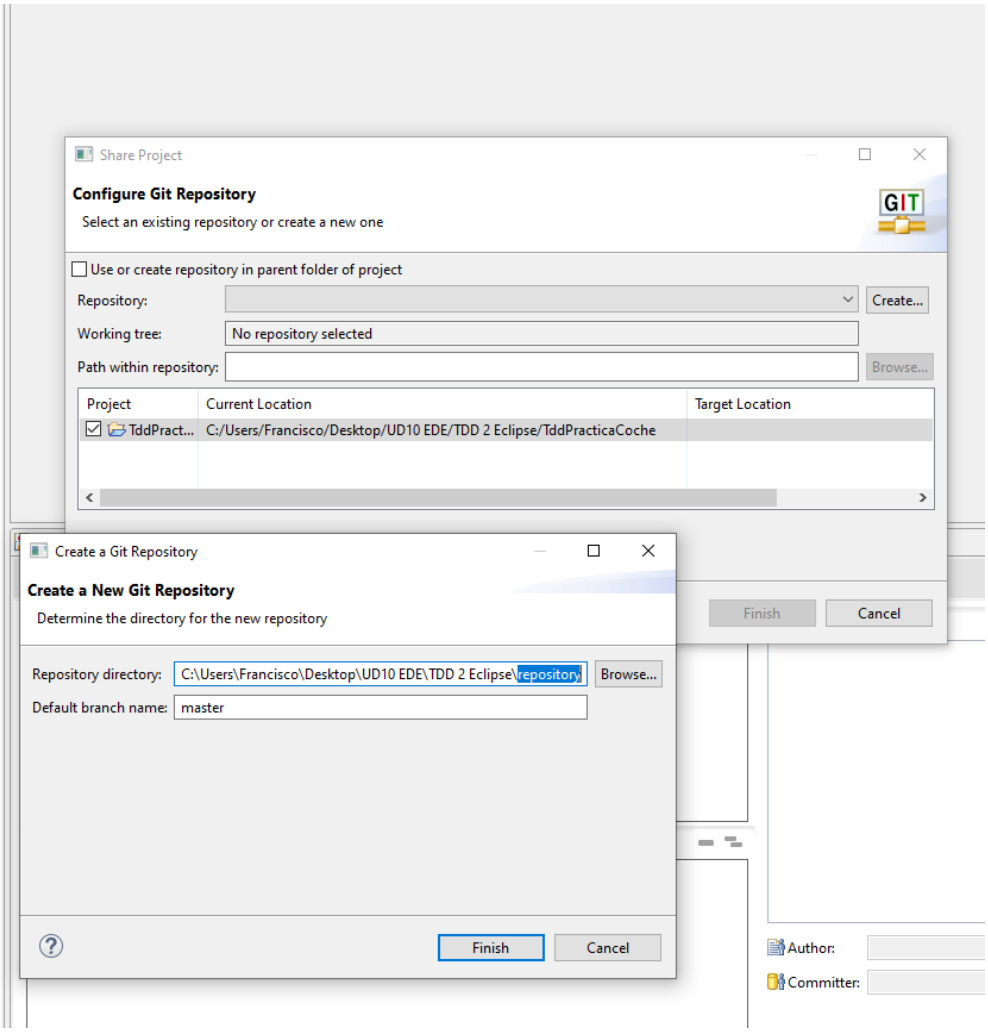
Solución:

Me hubiera gustado poder hacer la memoria un poco más elaborada, pero la falta de tiempo se ha hecho muy patente, sobre todo teniendo que utilizar un IDE nuevo como Eclipse y los problemas que he tenido para poder trabajar con GitHub en él.

Primero creamos el proyecto de Java, clicando en la pestaña “New Proyect” seleccionando un nuevo proyecto en java nombrándolo: TDD Practica Coche. Obteniendo una vista en Eclipse como se ve en la siguiente captura:

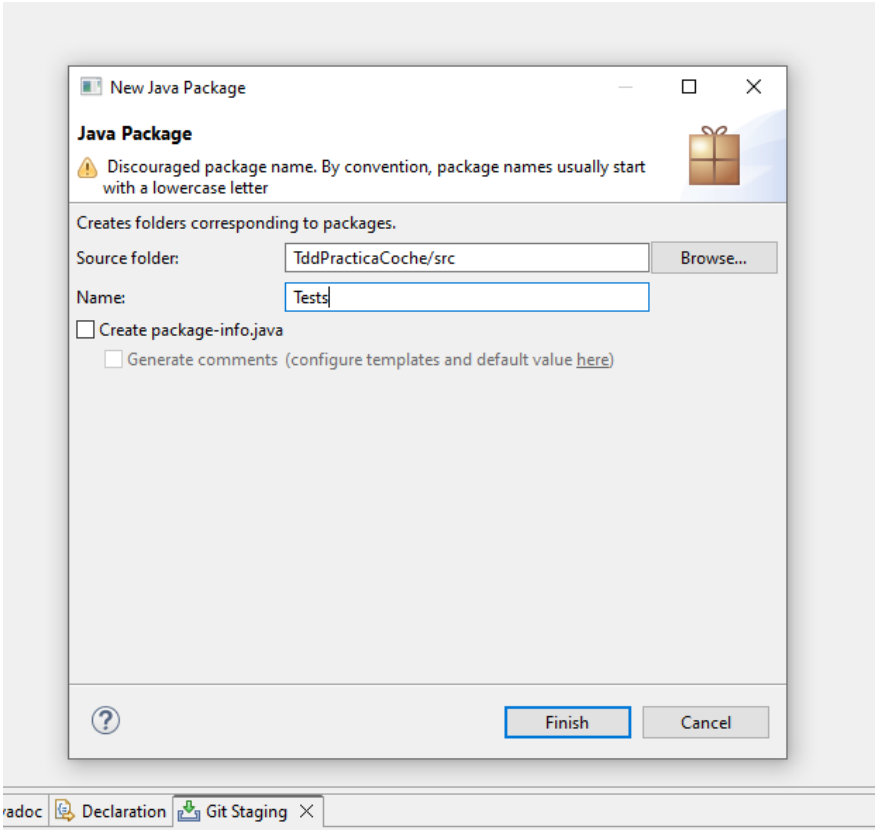


Primero debemos crear nuestro repositorio local de Git de la siguiente manera:

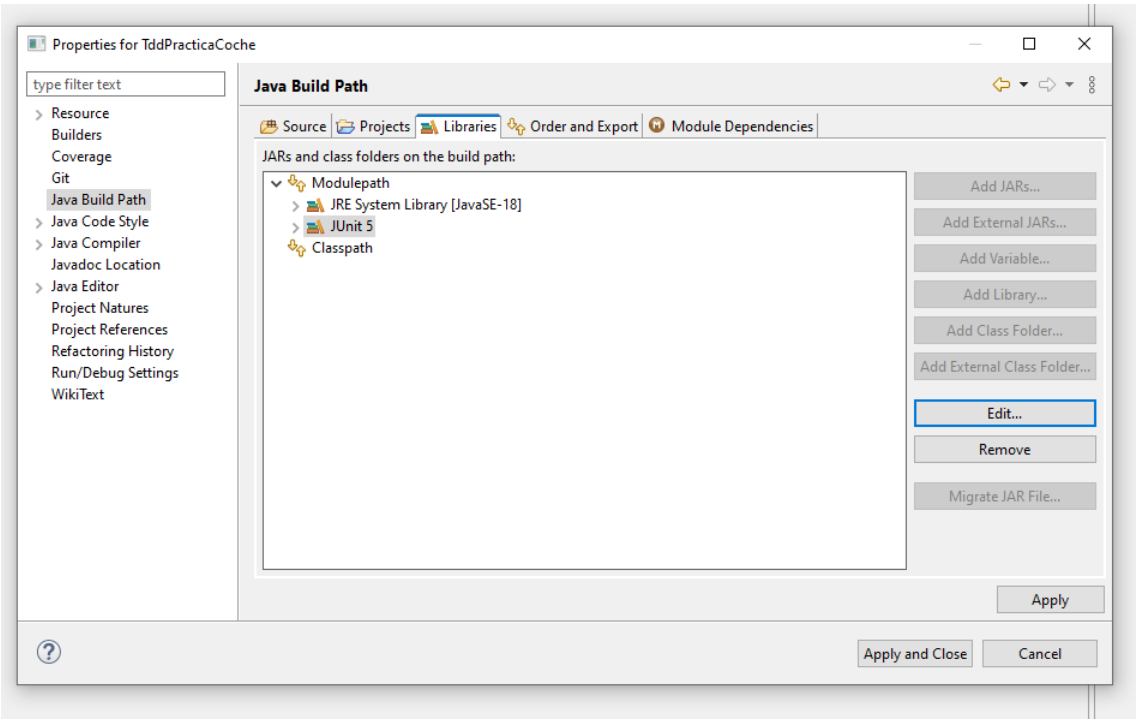


En este punto, realizamos nuestro primer “commit”, justo después de crear nuestro proyecto.

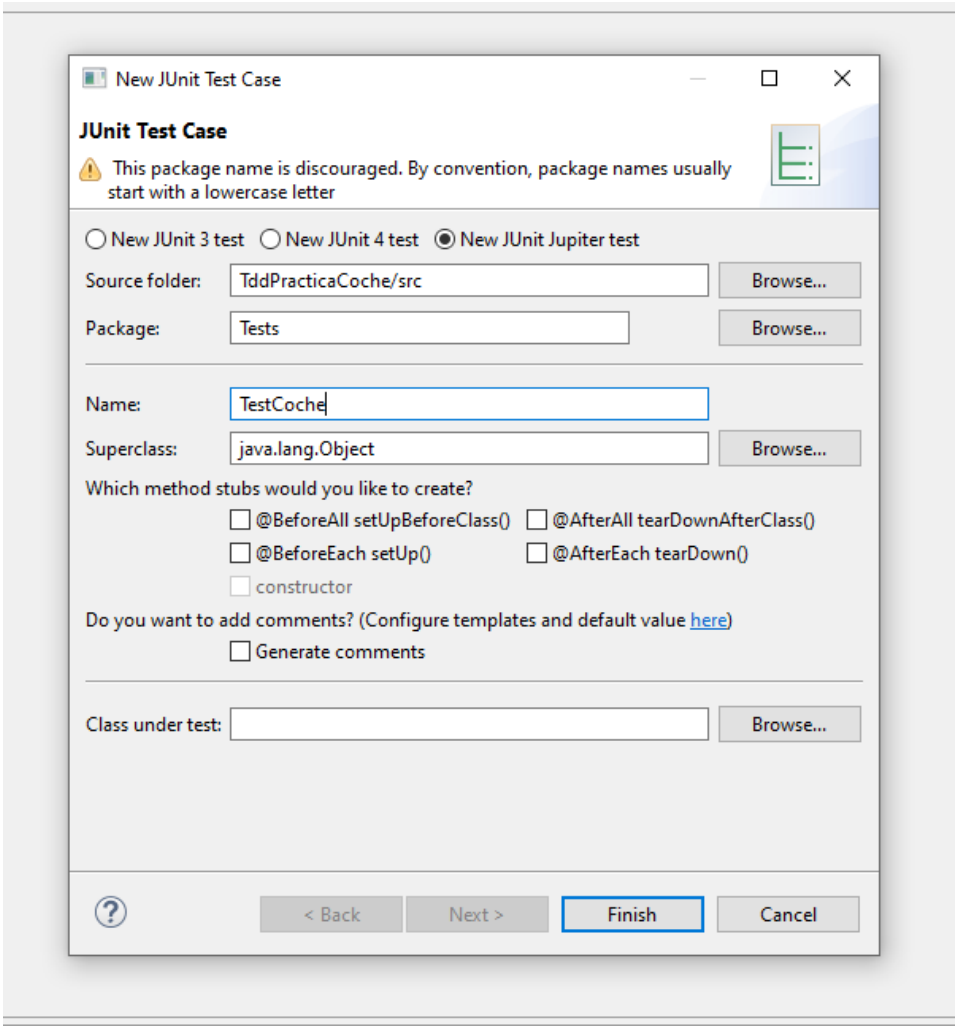
Creamos entonces un “package” con el nombre Tests para poder crear dentro de él las diferentes “java class” que vamos a necesitar.



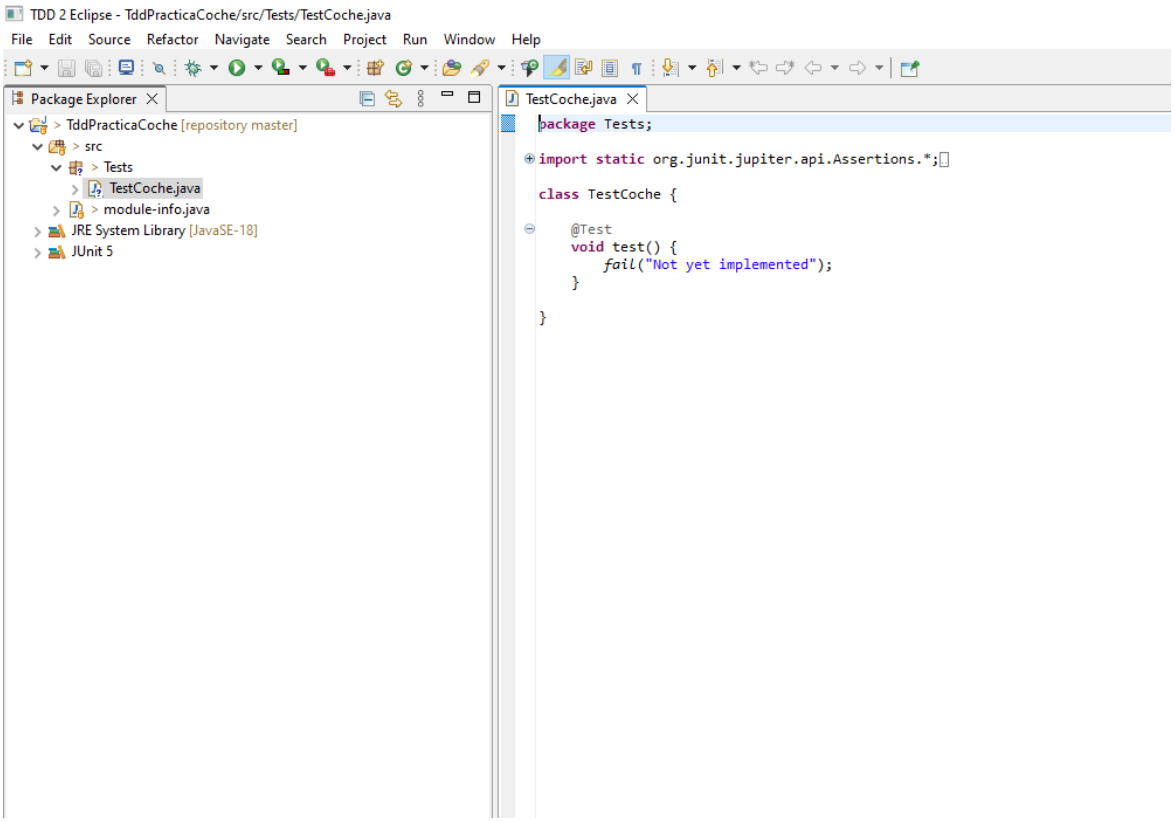
Una vez hecho lo anterior, tenemos que importar a nuestra librería del proyecto Junit para poder utilizar @test y Assertions, obtendríamos algo así:



Pasamos ahora a crear un nuevo Junit de nombre “TestCoche” donde podremos empezar a desarrollar los test.

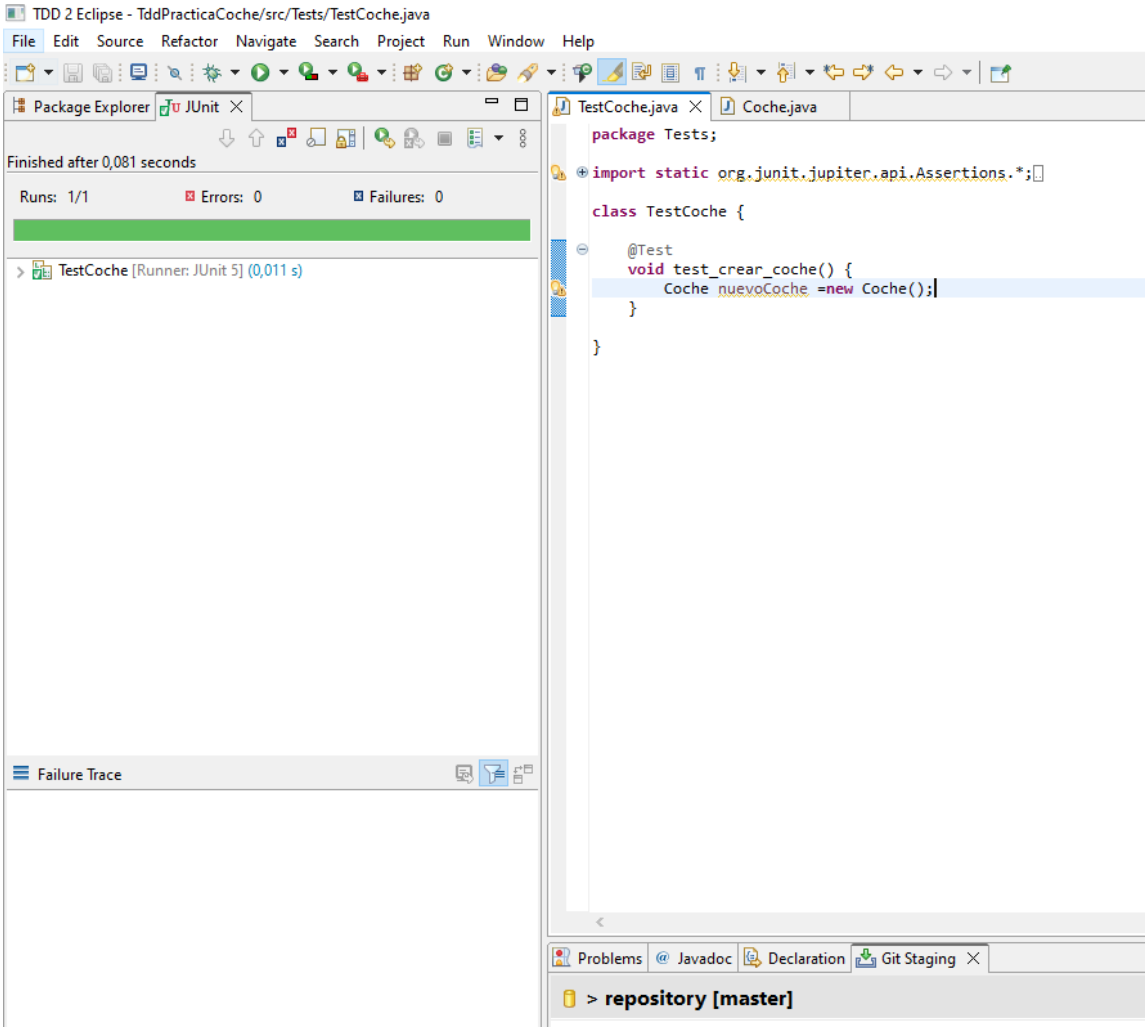


Una vez hecho lo anterior obtenemos lo siguiente:



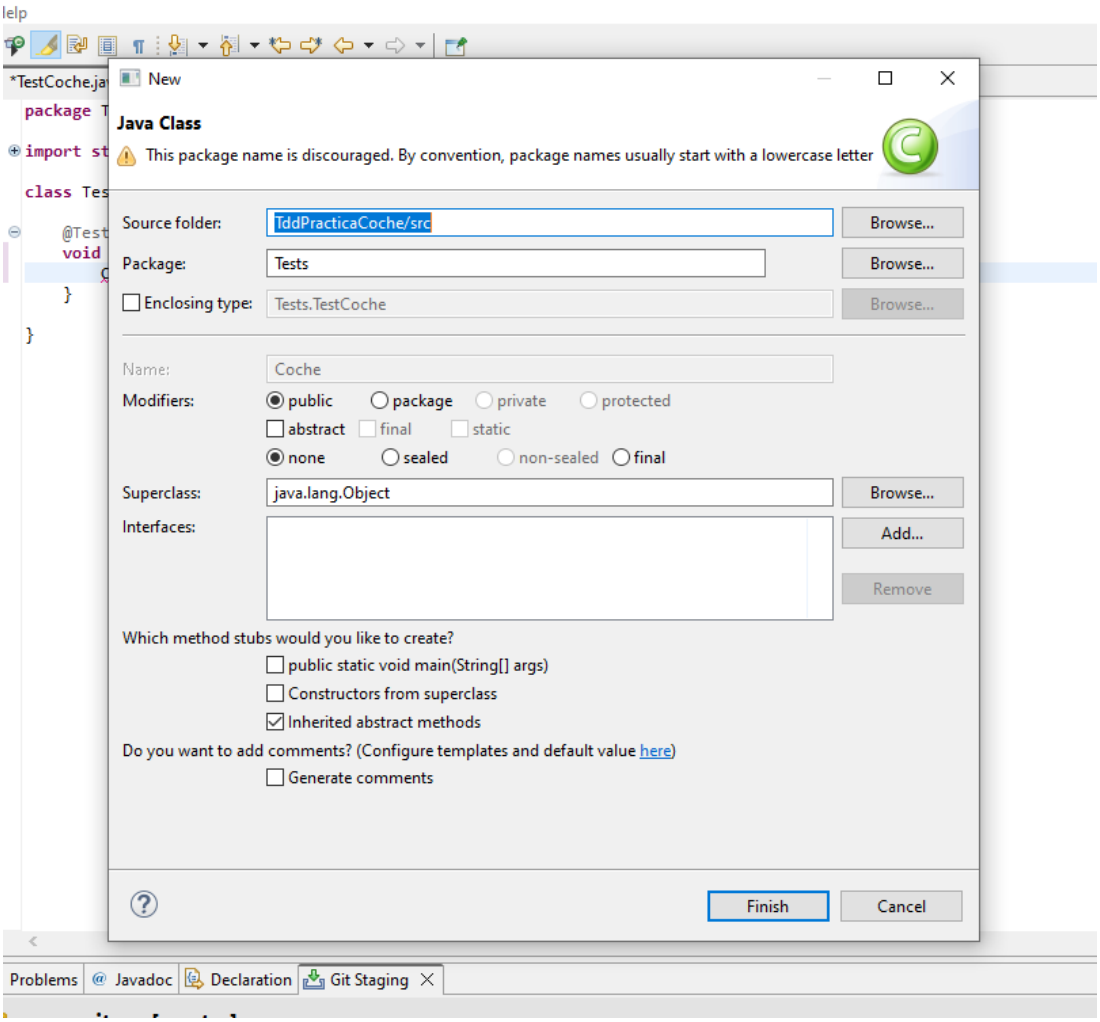
En todo momento y con cada cambio estamos realizando los “commits” necesarios.

Ahora procedemos a escribir nuestro primer test: “crear coche”:

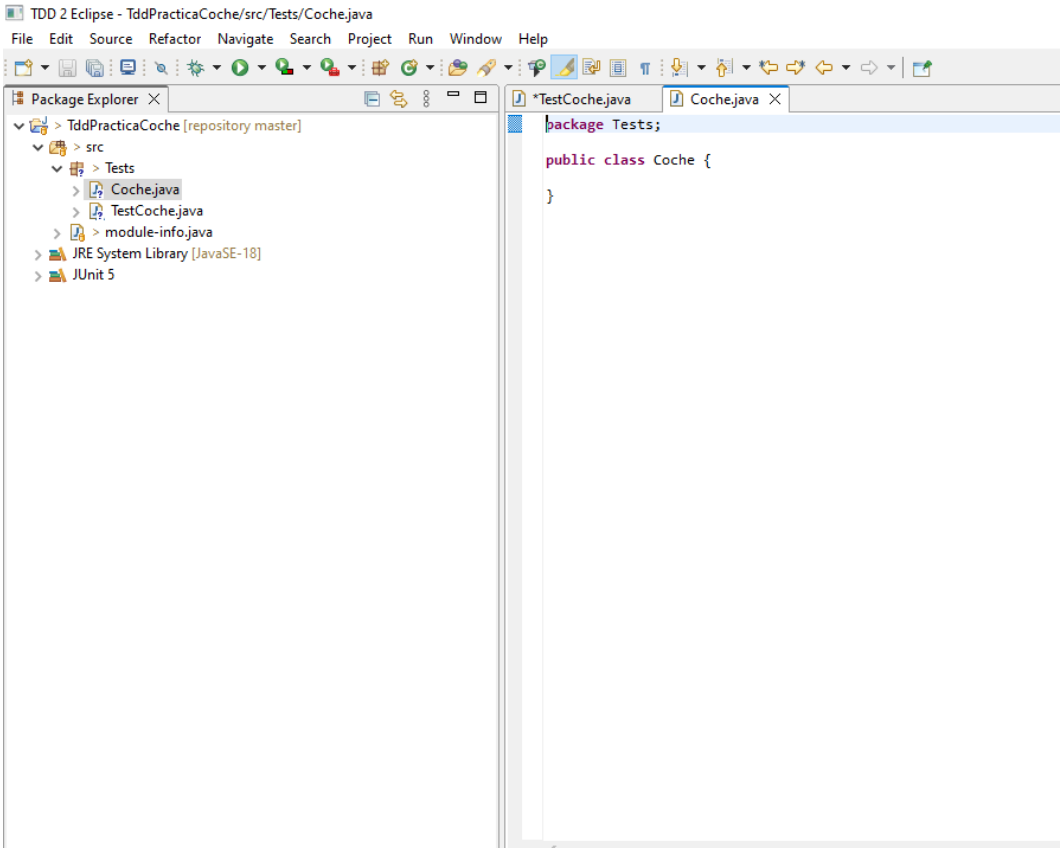


Podemos ver como se ha superado perfectamente sin arrojar ningún error. También es cierto que este primer test es muy simple.

Como nos pasaba con el IDE anterior, como estamos intentando crear objetos de la clase Coche antes de crear dicha clase, el IDE nos reporta un error que subsanaremos creando la clase coche tal y como nos indica:

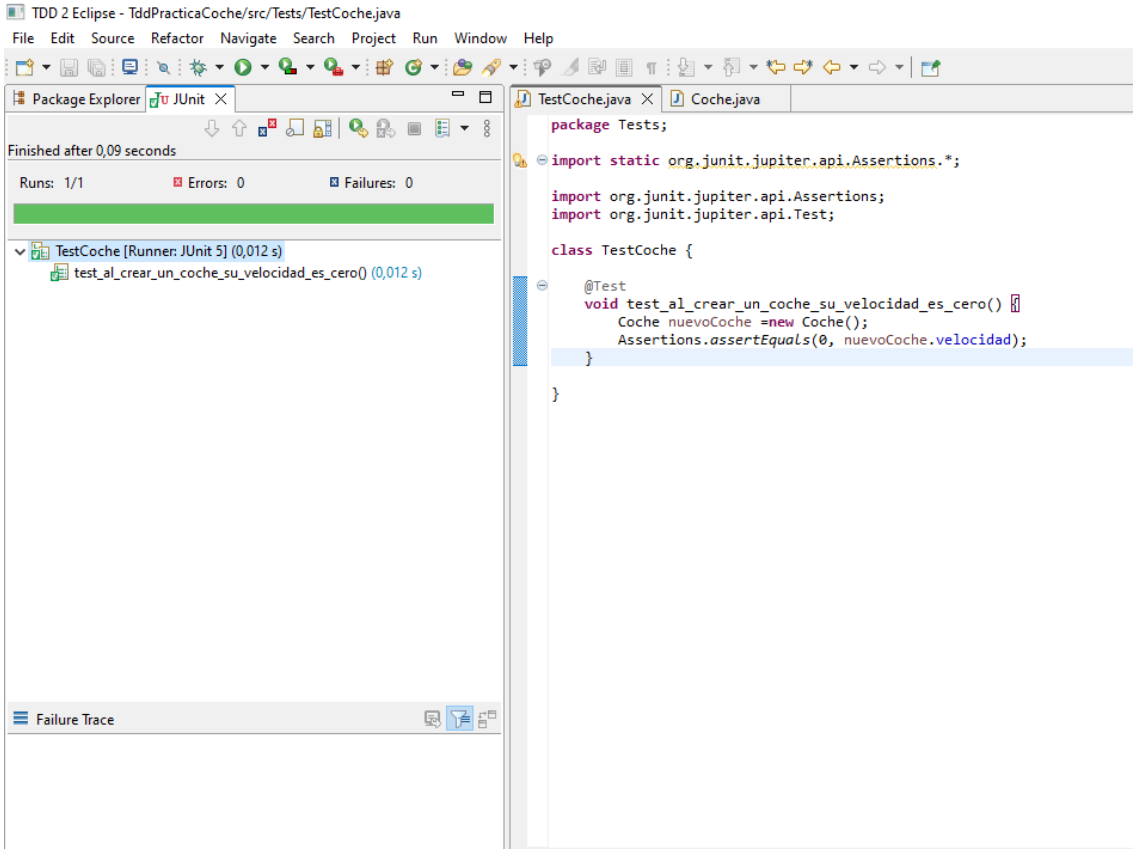


Una vez hecho lo anterior, vemos como se ha creado la clase Coche:

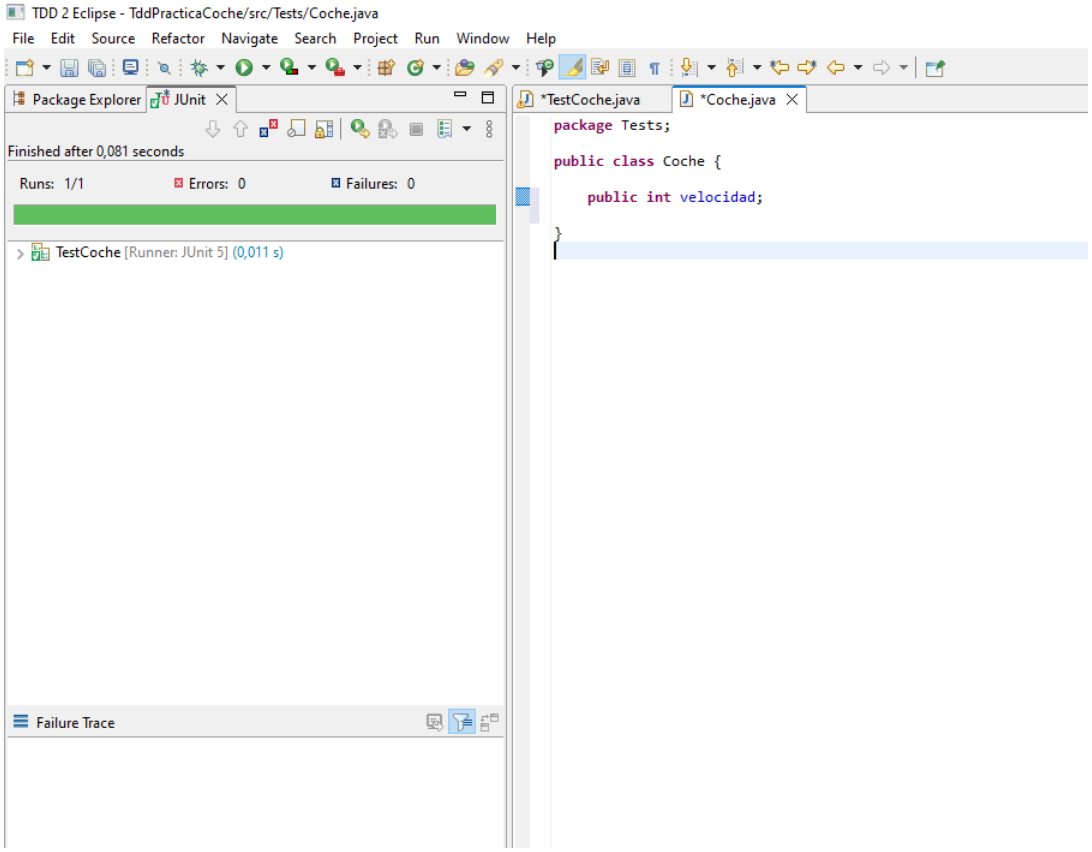


Hacemos el “commit” correspondiente con los cambios realizados.

Para hacerlo más completo, procedemos a cambiar el primer test añadiendo una “Assertions” para poder crear un coche, en este caso, y que su velocidad sea cero una vez se cree.



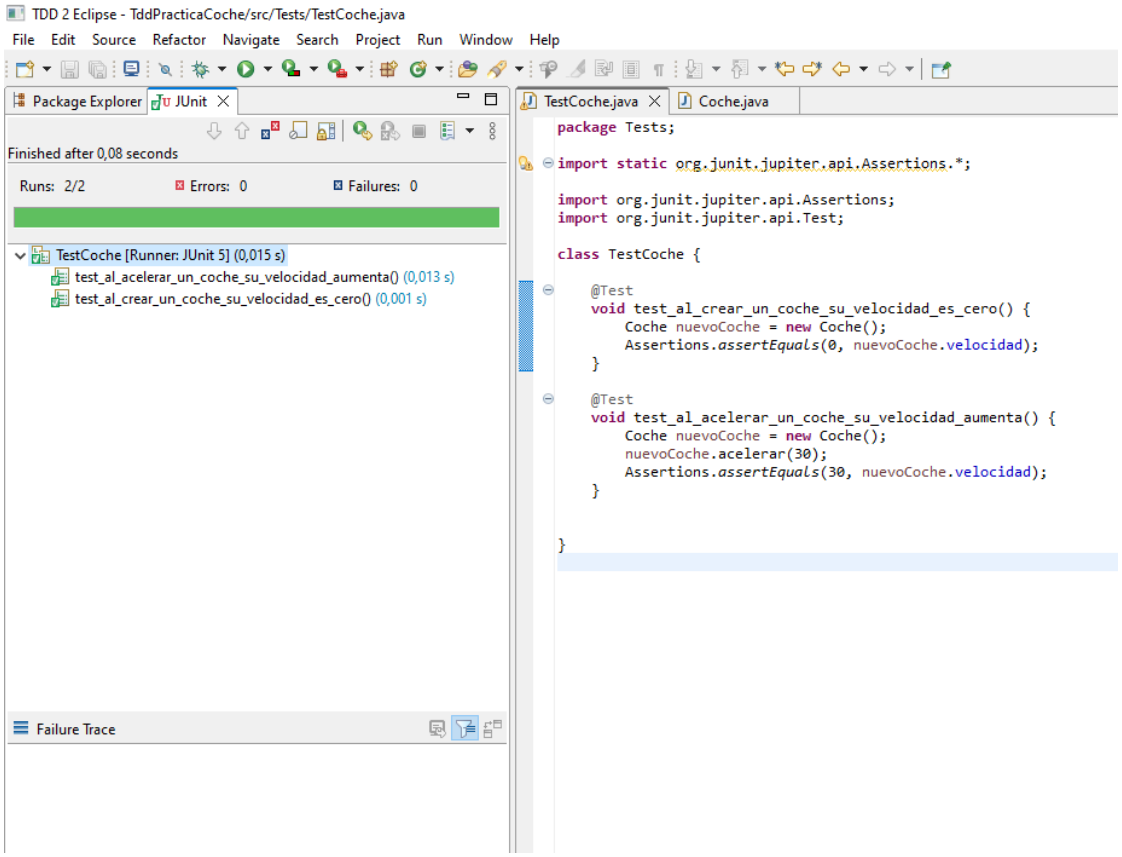
Para ello, utilizamos el método velocidad que, al no estar creado, nos da un error que subsanaremos creándolo en la clase Coche automáticamente como vemos a continuación.



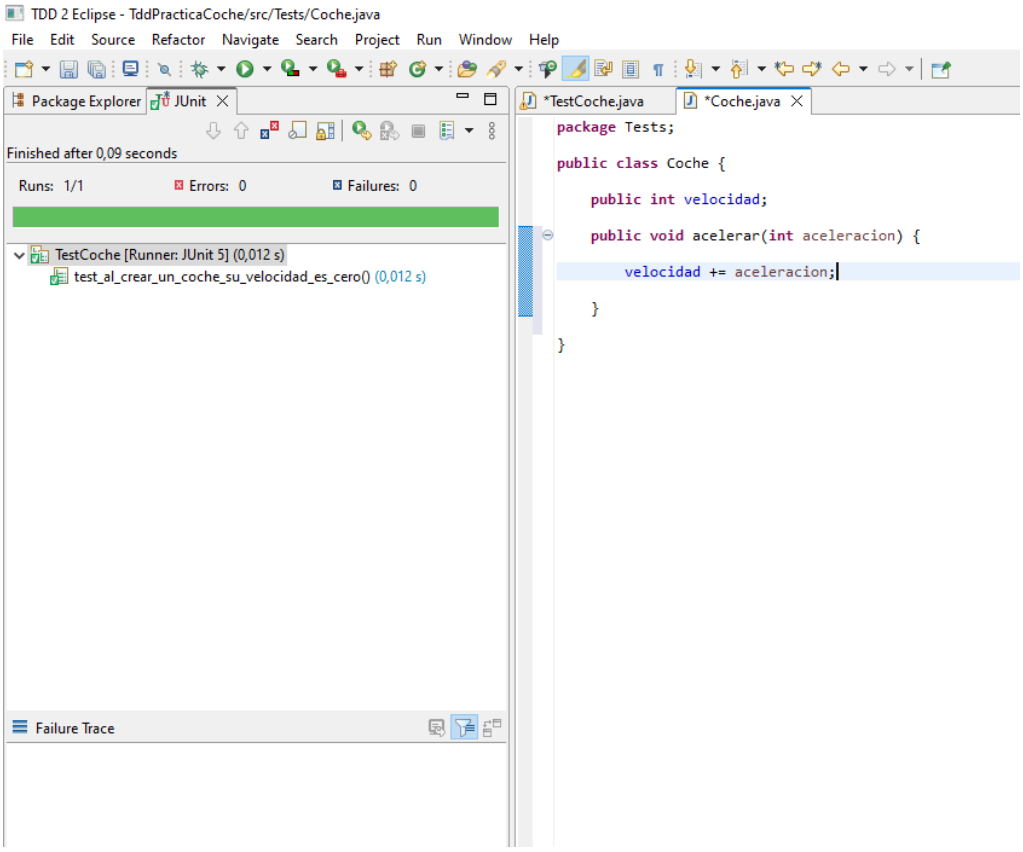
Vemos que el test se ejecuta correctamente una vez corregido el error.

Una vez realizado, hacemos otro “commit” con los cambios hechos: modificación test y creación método velocidad.

Ahora procedemos a crear nuestro segundo test, el de acelerar aumenta la velocidad. Para ello, hacemos un copia y pega del test anterior y lo modificamos añadiendo un nuevo método acelerar, pasándole por parámetro 30.

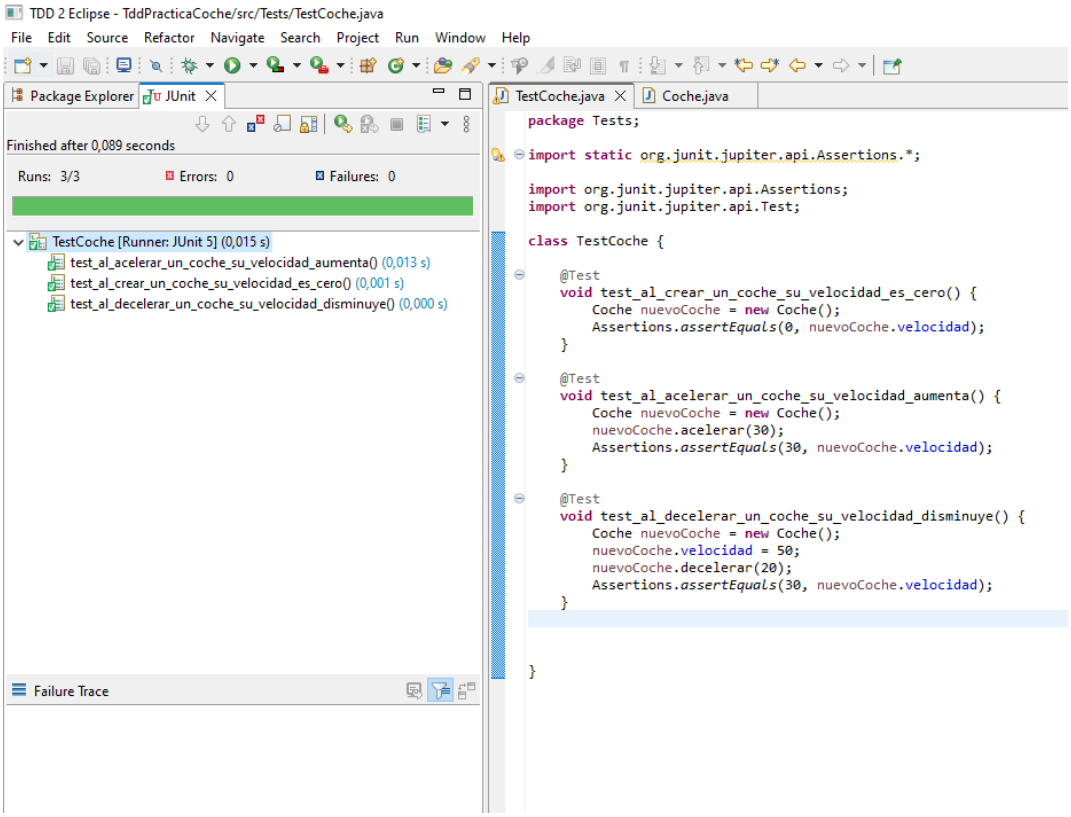


También en este caso nos da un error y es poque no hemos creado el método acelerar en la clase Coche, pero el IDE nos da la opción de crearlo automáticamente. Una vez creado le decimo que la velocidad aumente con la instrucción que se ve dentro del método.

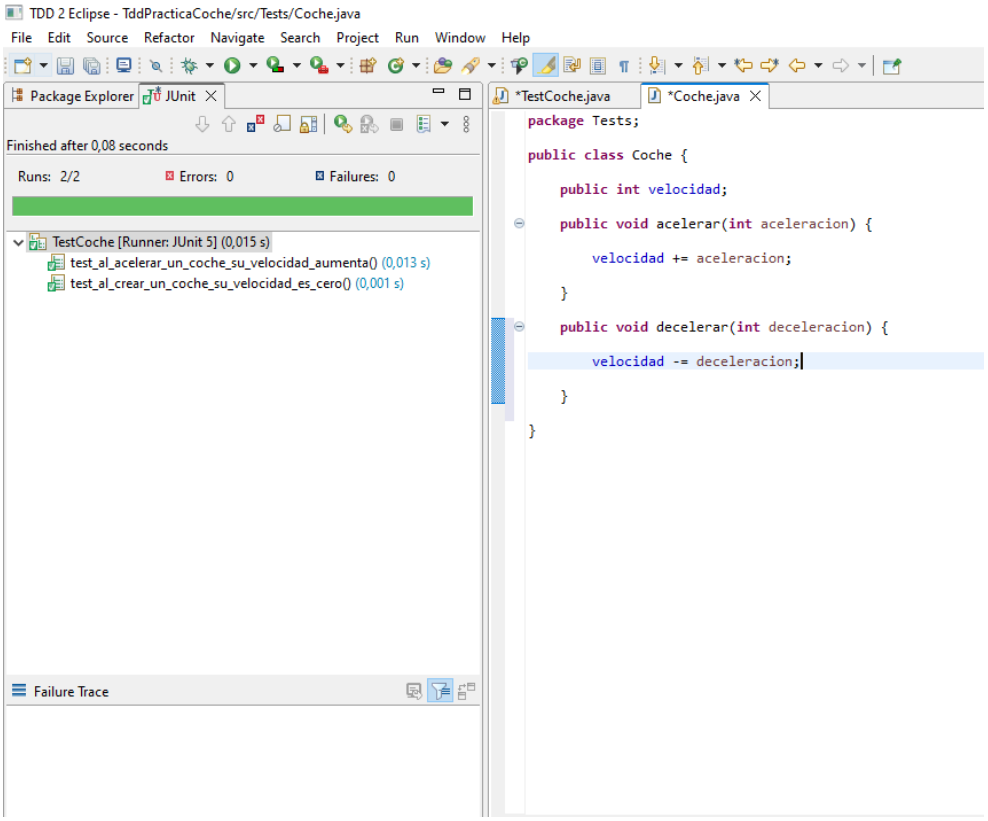


Una vez comprobado que el test se ejecuta correctamente, procedemos a realizar un nuevo “commit” con los cambios realizados: creación de un nuevo método y un nuevo test.

Todo seguido, creamos un nuevo test, esta vez decelerando al coche y disminuyendo su velocidad, para ello debemos hacer que el coche ya tenga una velocidad, en este caso de 50, y posteriormente, decelerarlo mediante el método decelerar unos 20.



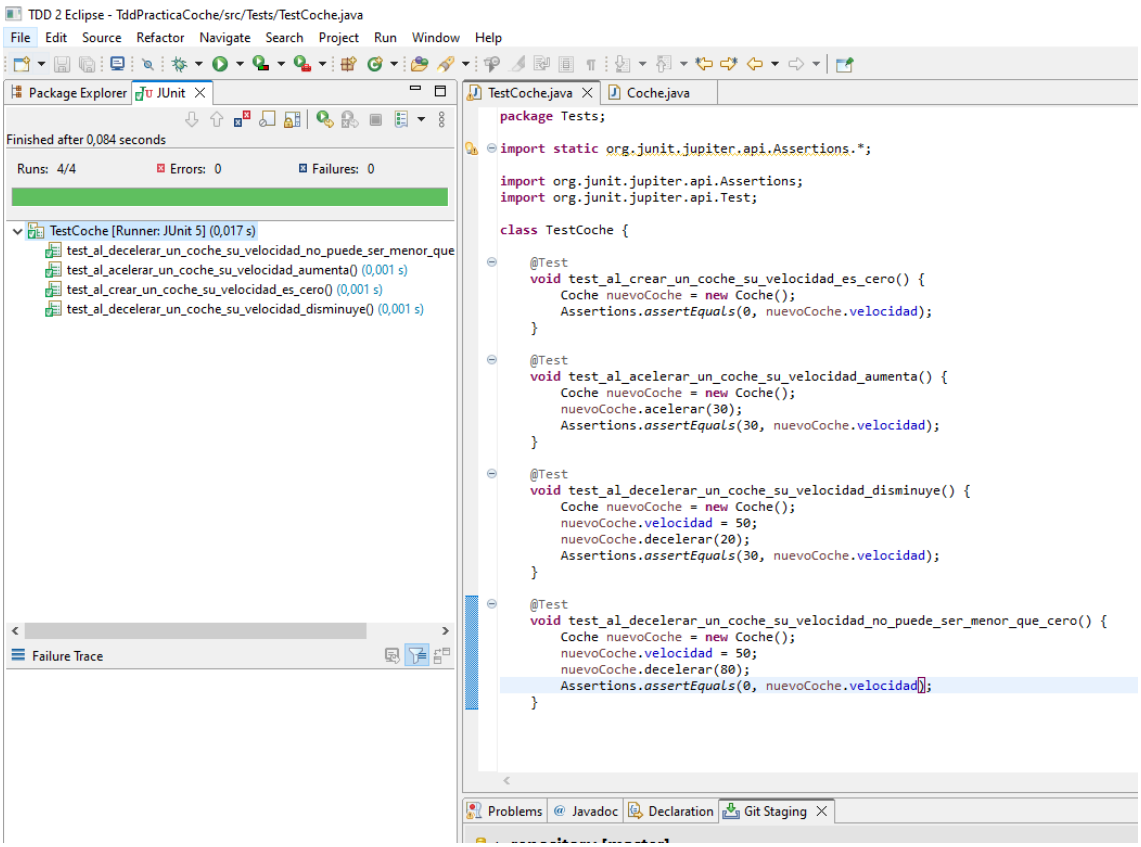
Como viene ya siendo costumbre, como no hemos creado el método antes de utilizarlo, nos da un error, que como anteriormente subsanamos creándolo automáticamente con su correspondiente opción. Dentro del método, incluimos la instrucción que nos permitirá que el coche pueda disminuir la velocidad.



Posteriormente, realizamos el consiguiente “commit” con los cambios realizados. Creación nuevo test decelerar y método decelerar.

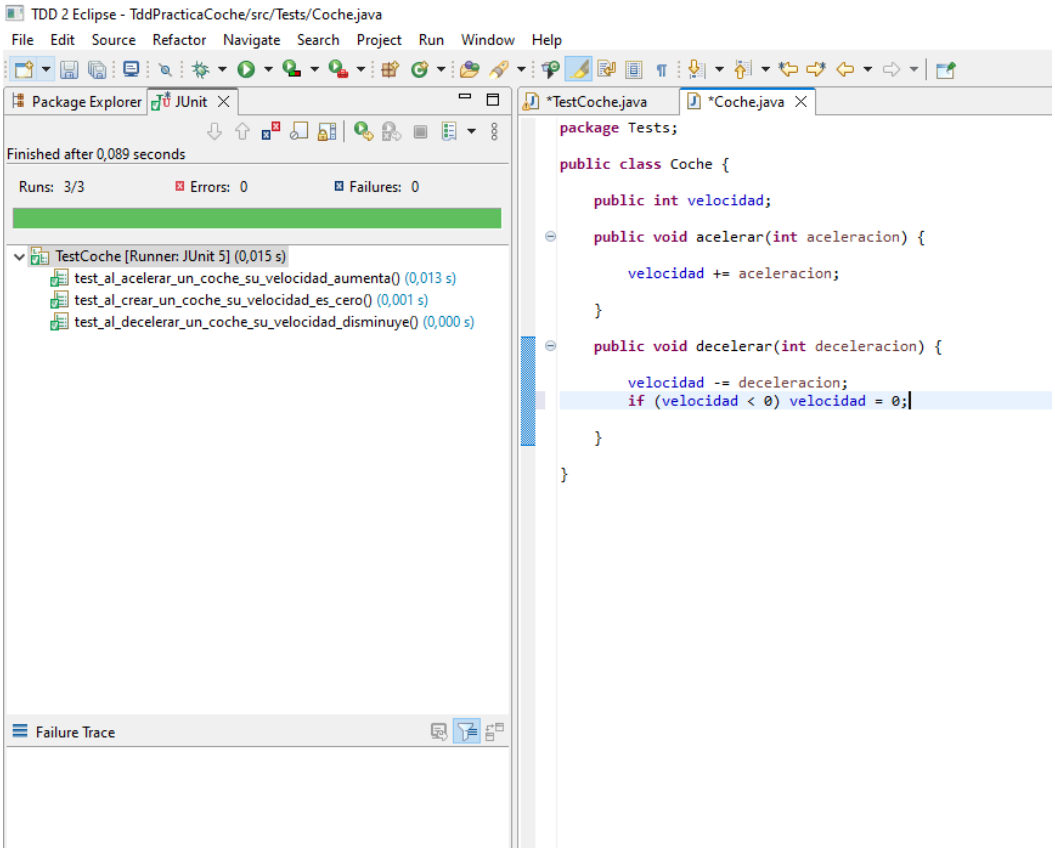
Vemos como hasta ahora se van ejecutando con éxito todos los test.

Pasamos ahora a crear otro test, esta vez para que cuando el coche decelere su velocidad no pueda ser inferior a cero. Para ello modificamos el “Assertions” con la velocidad “0” esperada y procedemos a su ejecución.



En este caso nos da un error, y este se debe a que no hemos modificado el método decelerar para que en caso de obtener valores negativos, como es el caso, el resultado igualmente sea “0”.

Para ello, modificamos el código dentro de nuestro método como se ve en la siguiente imagen:



Y como se puede ver, el test ya no arroja ningún error.

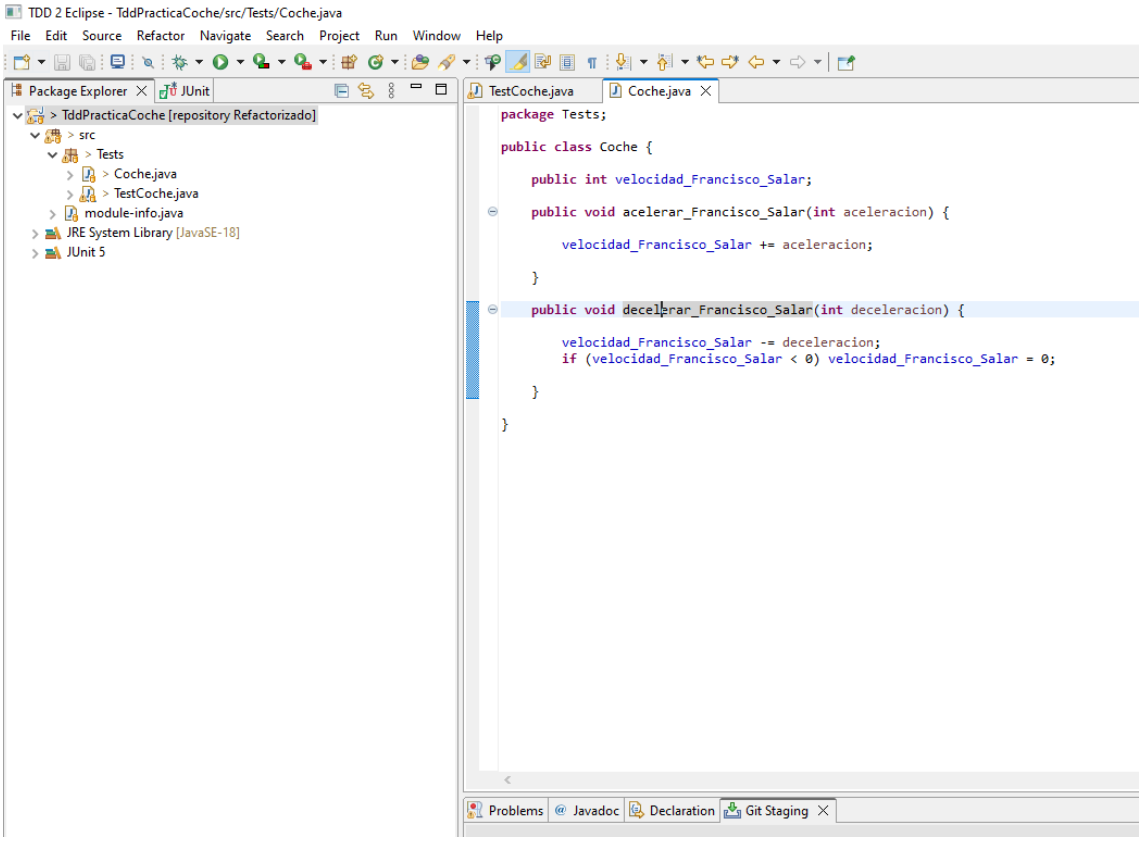
Una vez hecho lo anterior, procedemos a subir a nuestro repositorio remoto el proyecto con los comandos pertinentes, una vez habiendo creado en GitHub nuestro repositorio y habiendo copiado la url para poder hacer un “push a origen”.

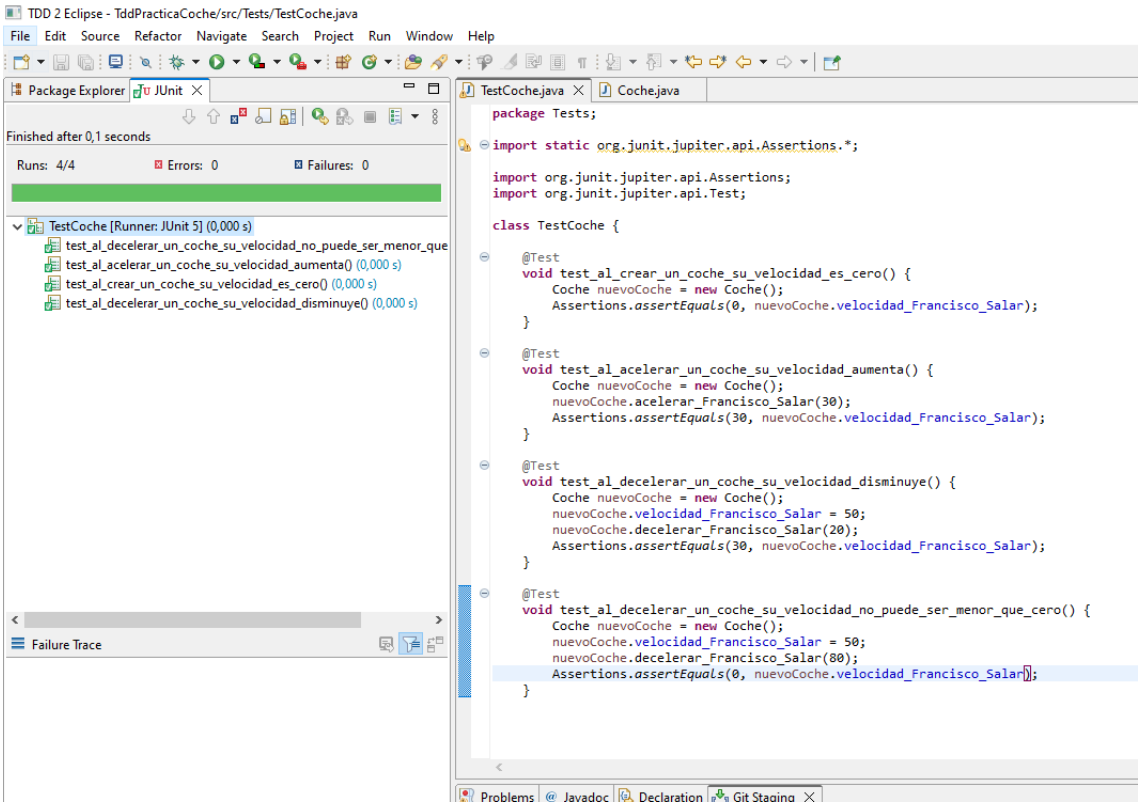
Ahora creamos una nueva rama donde refactorizaremos los nombres de los métodos añadiendo nuestro nombre a cada uno de ellos.

Para ello mediante “git Branch Refactorizado”, creamos la nueva rama, y con “git checkout Refactorizado” nos posicionamos en ella para realizar el refactorizado de los métodos.

Hacemos entonces un “commit” con los cambios hechos: creación de la nueva rama “Refactorizado”.

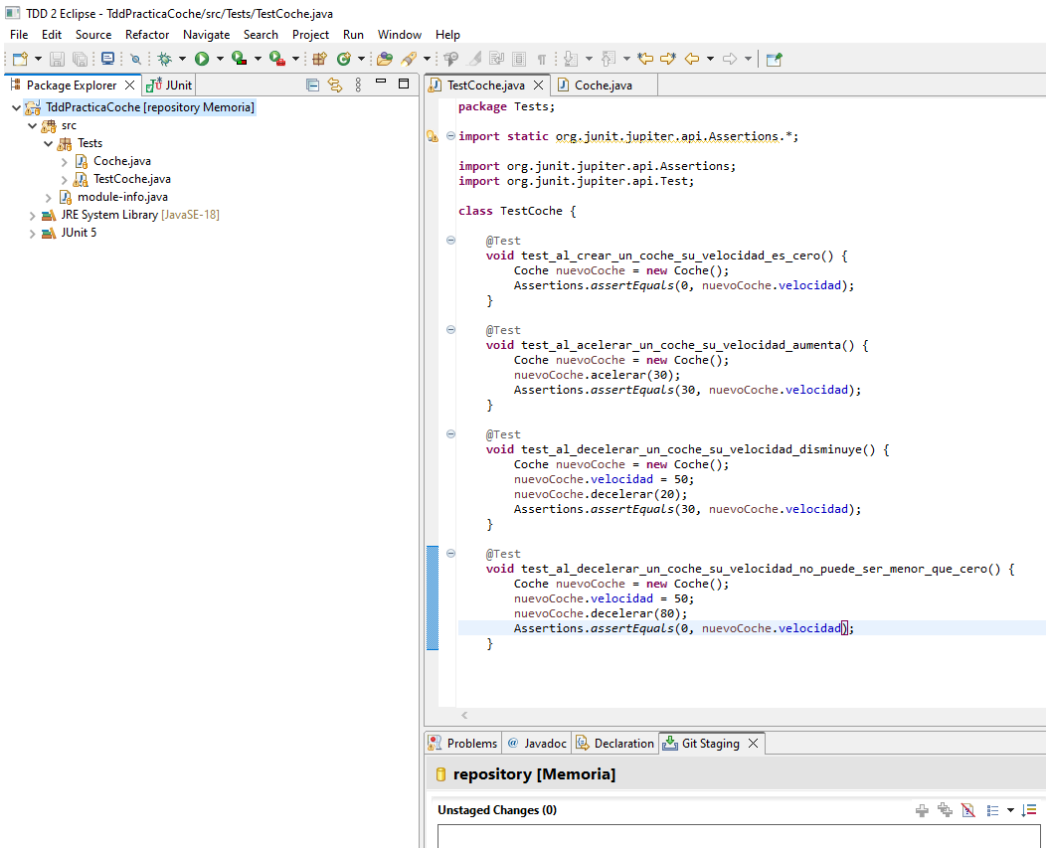
En las dos imágenes siguientes podemos ver como se han modificado los métodos con la adición de mi nombre al final de los mismos.





A continuación, hacemos un “commit” a nuestro repositorio local con los cambios realizados en la rama “Refactorizado” y posteriormente con los comandos correspondientes subimos a nuestro repositorio remoto la nueva rama con sus cambios “git push origin Refactorizado”.

Por último, creamos una rama llamada “Memoria” donde subiremos el pdf con la memoria realizada de esta práctica.



Haciendo el último “commit” con los cambios realizados: creación nueva rama ”Memoria” y adición pdf de la memoria. Posterior mente con los comandos que hemos visto anteriormente hacemos el push correspondiente a nuestro repositorio remoto.