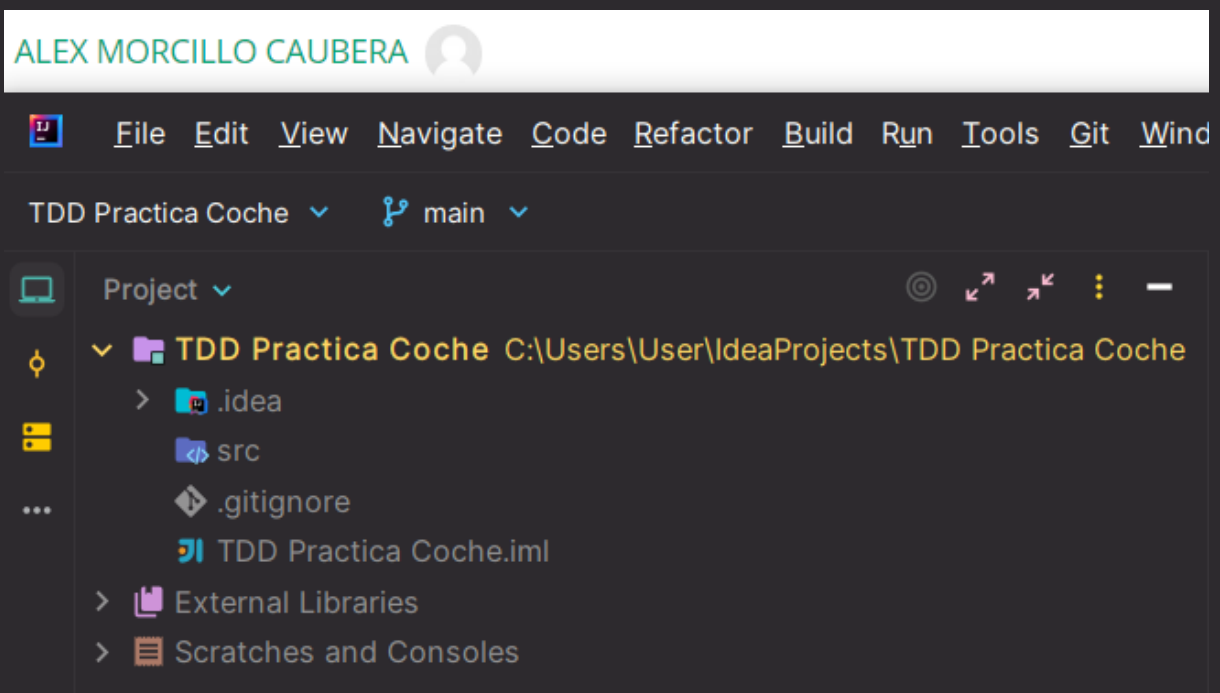


# Mi primer TDD

Alex Morcillo Caubera

Lo primero es crear el proyecto en el IntelliJ y crear nuestro repositorio sobre él.



Create a new repository

Name

Mi primer TDD -Alex Morcillo Caubera

⚠ Will be created as Mi-primer-TDD--Alex-Morcillo-Caubera

Description

Tarea Entornos de Desarrollo - Alex Morcillo Caubera

Local path

C:\Users\User\IdeaProjects\TDD Practica Coche

Choose...

☐ Initialize this repository with a README

Git ignore

None

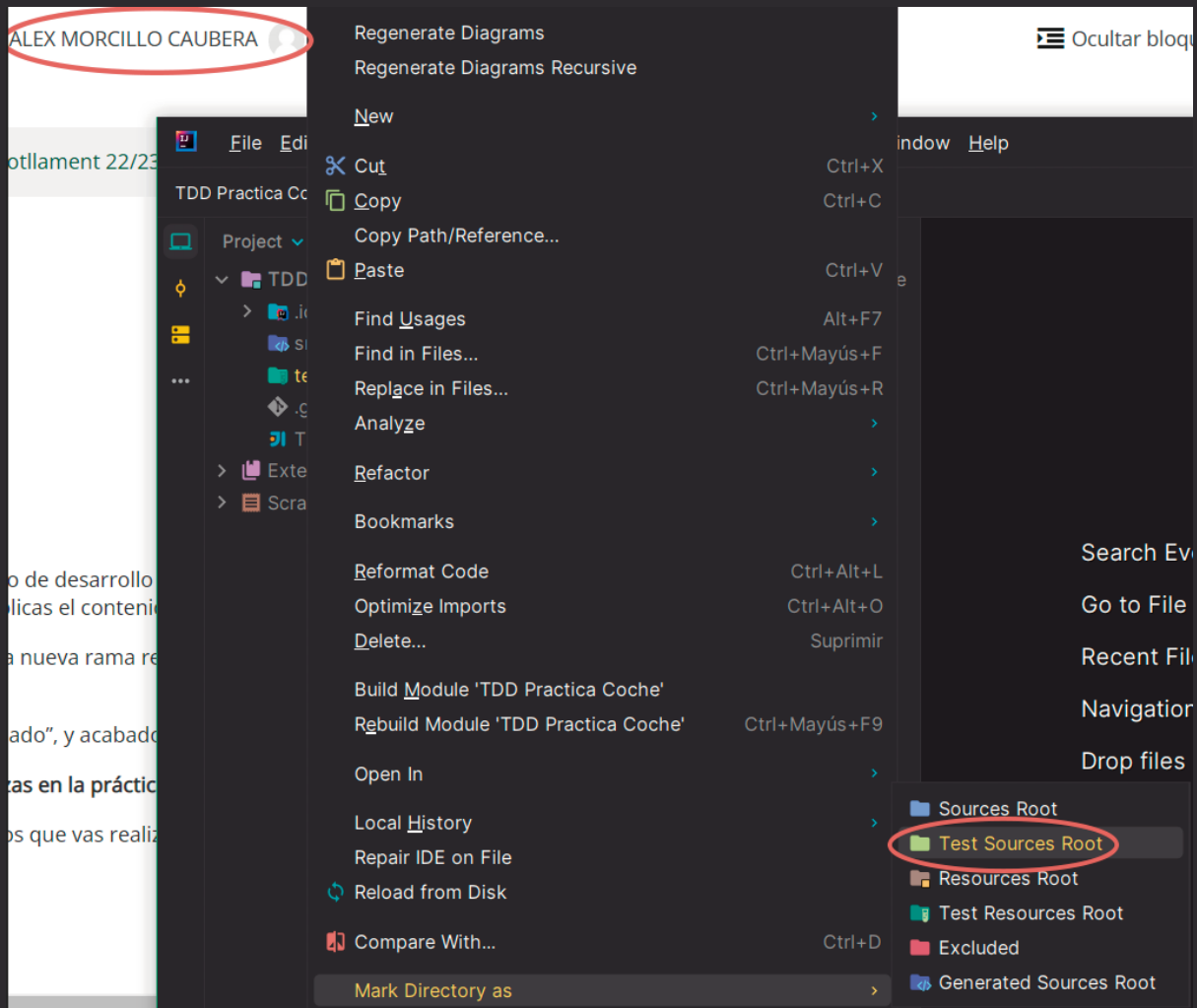
License

None

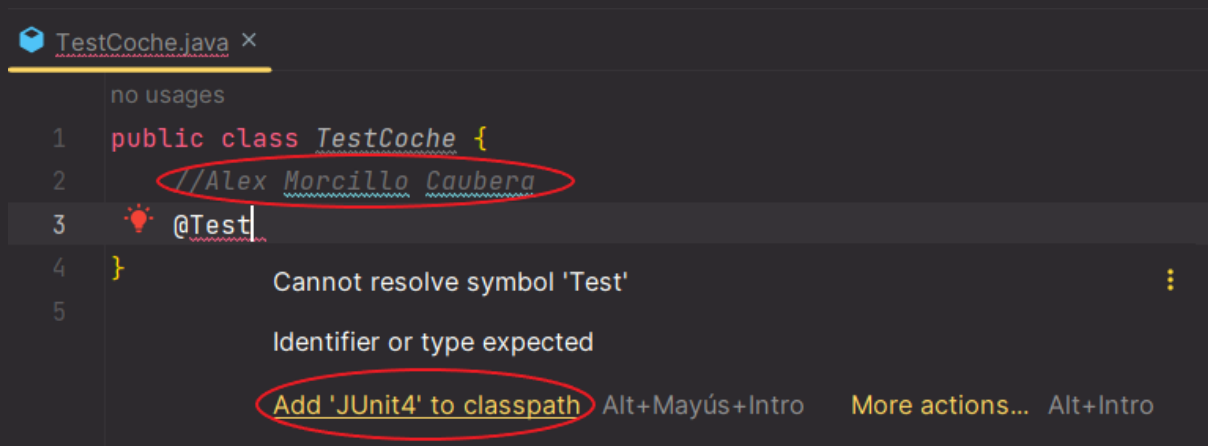
Create repository

Cancel

Luego, creamos una nueva carpeta y la marcamos como “Test Sources Root”.



Ahora, creamos una clase nueva dentro de este directorio, llamada "TestCoche". Dentro de ésta, intentamos inicializar un método "@test". Podemos ver que no nos deja, porque Junit no está agregado al classpath.



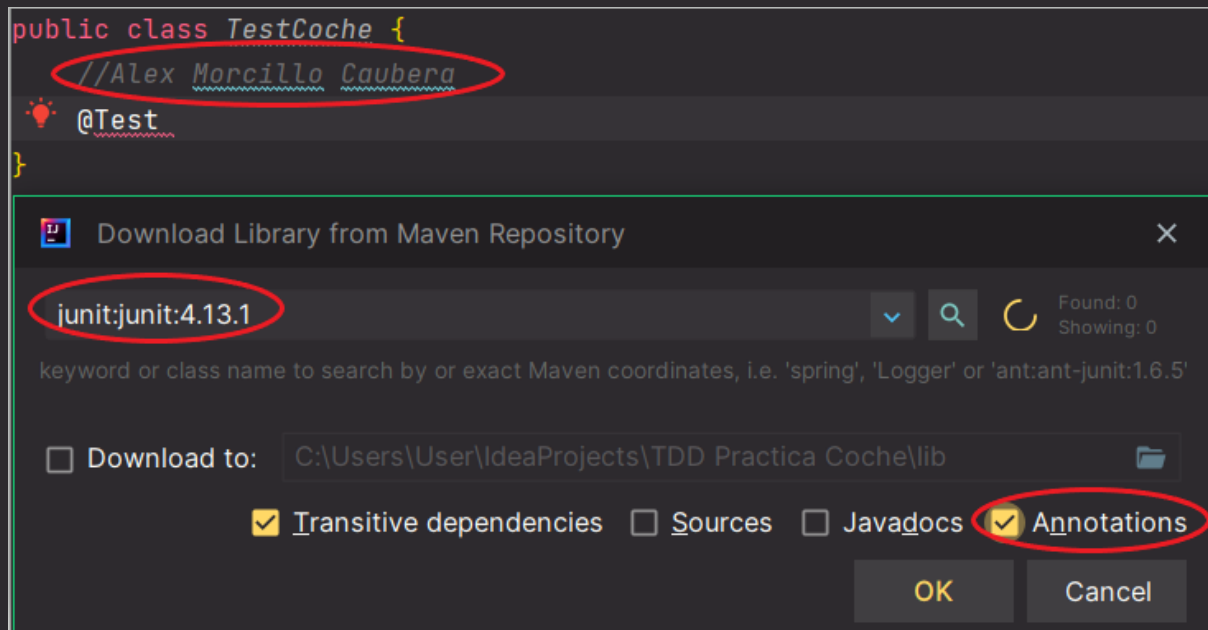
```
TestCoche.java x
no usages
1 public class TestCoche {
2     //Alex Morcillo Caubera
3     @Test
4 }
5
```

Cannot resolve symbol 'Test'

Identifier or type expected

Add 'JUnit4' to classpath Alt+Mayús+Intro More actions... Alt+Intro

Lo agregamos al classpath de la siguiente forma



Ahora, vamos a crear el método “test\_crear\_coche” en el que instanciaremos un nuevo objeto de la clase “Coche”. Veremos que nos da error ya que la clase coche no existe.

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

Cannot resolve symbol 'Coche'

Create class 'Coche' Alt+Mayús+Intro More actions... Alt+Intro

Clicamos en donde dice “Create class ‘Coche’”. Se nos abrirá una ventanita en la que solo tendremos que clicar en “OK”

```
public class TestCoche {  
    //Alex Morcillo Caubera  
    @Test
```

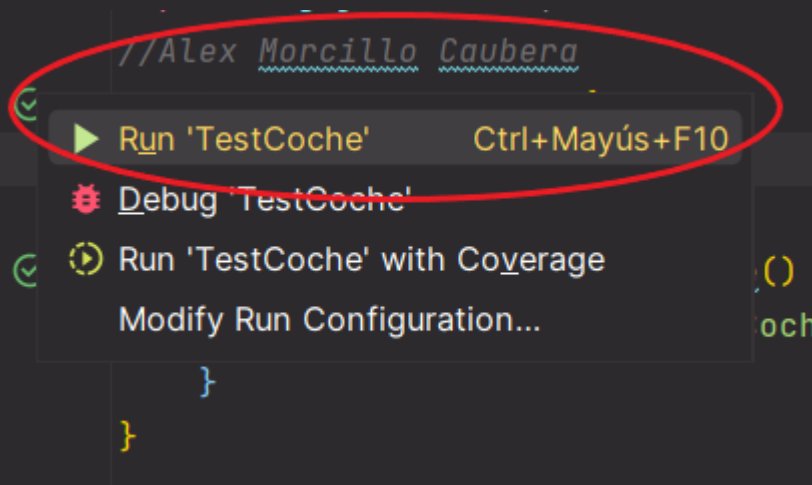
Create Class Coche

Destination package:

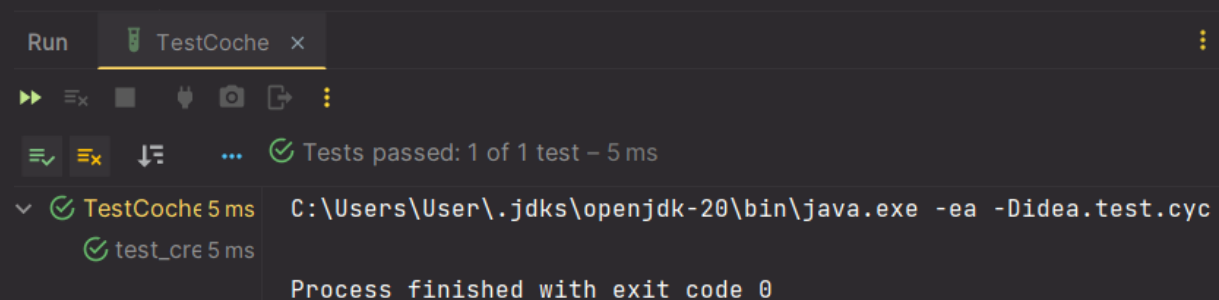
Target destination directory:

OK Cancel

Una vez hayamos creado la clase, ejecutamos la clase TestCoche para ver si supera la prueba.



Podemos ver que la supera sin problemas



Ahora vamos a crear un método mejorado a partir del mismo de antes. Lo renombraremos a “test\_al\_crear\_un\_coche\_su\_velocidad\_es\_cero” y le añadiremos una nueva línea de código para comprobar si su velocidad es cero. Como se puede ver, al crear esta nueva línea salta un error en “velocidad”, ya que la clase coche no tiene esa variable. Simplemente la añadiremos pulsando en “Create field ‘velocidad’ in ‘Coche’”

```

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

```

Cannot resolve symbol 'velocidad'

Create field 'velocidad' in 'Coche' Alt+Mayús+Intro More actions.

Una vez hemos añadido la variable “velocidad”, simplemente, ejecutamos la clase una vez más para ver si supera la prueba. En este caso, supera la prueba sin complicaciones.

```

public class TestCoche {
    //Alex Morcillo Caubera
    @Alex_Morcillo_Caubera *
    @Test
    public void test_al_crear_un_c...() {
        Coche nuevoCoche = new Coche();
        Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
    }
}

```

Run 'test\_al\_crear\_un\_c...()' Ctrl+Mayús+F10

Debug 'test\_al\_crear\_un\_c...()'

Run 'test\_al\_crear\_un\_c...()' with Coverage

Modify Run Configuration...

Tests passed: 1 of 1 test – 11 ms

TestCoch 11 ms

test\_al\_11 ms

C:\Users\User\.jdk\openjdk-20\bin\java.exe ...

Process finished with exit code 0

Ahora vamos a crear un nuevo método para acelerar, vamos a copiar y pegar el método anterior y a modificarlo. Se llamará “test\_al\_acelerar\_un\_coche\_su\_velocidad\_aumenta” y será de la siguiente forma:

```
//Alex Morcillo Caubera
@Test
public void test_al_acelerar_un_coche_su_velocidad_aumenta() {
    Coche nuevoCoche = new Coche();
    nuevoCoche.acelerar();
    Assertions.assertEquals(0, nuevoCoche.velocidad);
}
```

Como podemos ver, tenemos un error en la segunda línea del método, esto es debido a que la clase “Coche” no tiene ningún método llamado “acelerar”, así que tendremos que crearlo.

```
//Alex Morcillo Caubera
1 usage
public void acelerar(int aceleracion) {
    velocidad += aceleracion;
}
```



Ahora probamos el método y vemos que supera el test sin problemas.

```
4      //Alex Morcillo Caubera
      1 usage
5      public void acelerar(int aceleracion) {
6          |      velocidad += aceleracion;
7      }
8  }
```

Run TestCoche.test\_al\_crear\_un\_coche\_su\_velocidad\_es\_cero x

▶ ≡x ■ 🔌 📷 ➡ ⋮

≡✓ ≡x ⚡ ... ✓ Tests passed: 1 of 1 test – 12 ms

✓ TestCoch 12 ms

✓ test\_al, 12 ms

C:\Users\User\.jdk\openjdk-20\bin\

Process finished with exit code 0

Vamos a hacer lo mismo pero esta vez con un método decelerar, que en general, quedaría así:

```
//Alex Morcillo Caubera
no usages

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

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

Si ejecutamos esto, supera la prueba sin problemas.

Ahora crearemos un método que nos compruebe si la velocidad es menor que 0:

```
//Alex Morcillo Caubera
no usages
public void test_al_decelerar_un_coche_su_velocidad_no_puede_ser_menor_que_0() {
    Coche nuevoCoche = new Coche();
    nuevoCoche.velocidad = 50;
    nuevoCoche.decelerar( deceleracion: 80);
    Assertions.assertEquals( expected: 0, nuevoCoche.velocidad);
}
```

Si ejecutamos esto, no superará la prueba, porque la velocidad es negativa, por lo tanto, hay que modificar el método “decelerar”, para que la velocidad no pueda bajar de 0.

```
//Alex Morcillo Caubera
2 usages
public void decelerar(int deceleracion) {
    velocidad -= deceleracion;
    if (velocidad < 0) velocidad = 0;
}
```

Una vez agregado esto, podremos volver a probar y superaremos la prueba exitosamente.