

# Tests

## Programming - memo

Writing automated tests **ensures** code **quality** and releases us from the drag of having to do manual testing.

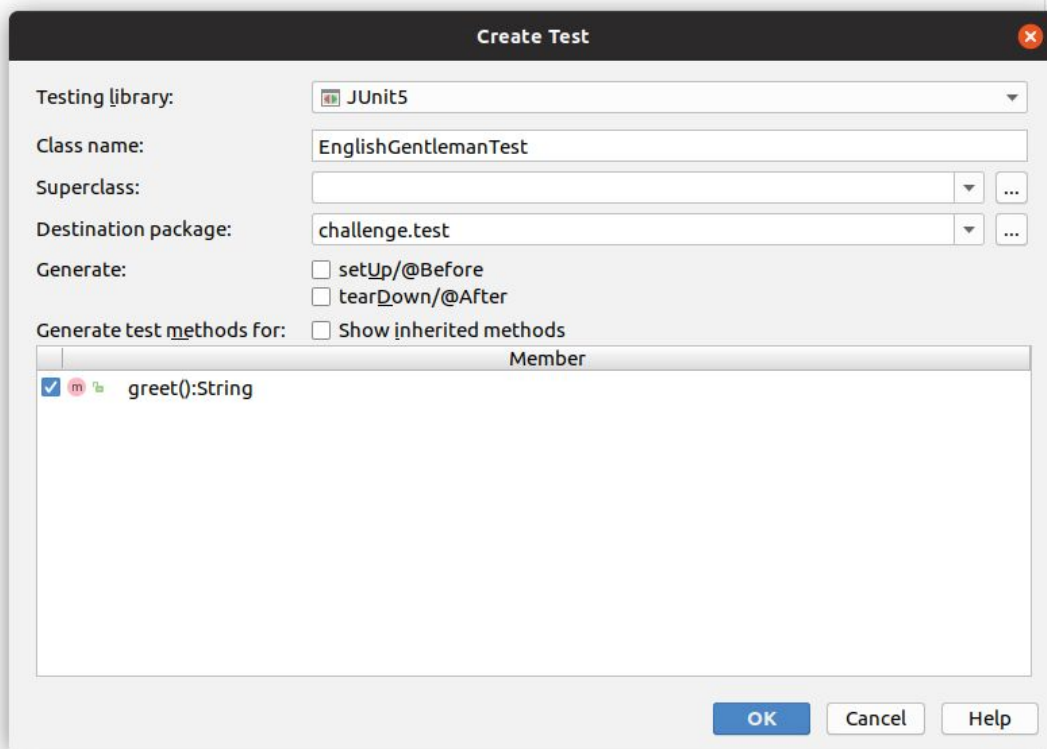
## Enabling tests

Enabling tests with IntelliJ is very easy. We just need to follow a set of steps:

- Click on the name of the class that we want to test
- Use the create test shortcut (Ctrl+Shift+T) and choose create new test



- Select the testing library JUnit5
- If the library is not found, a yellow light bulb will notify us and show a *fix* button
- Click on fix and wait for a while
- Activate the checkbox *download to* so that the test library is downloaded inside your current project
- Make sure the class name of the test class is exactly the same as the class you want to test, followed by the *Test* word
- Optionally, you can check the methods that you want to test so they are generated automatically



- Click on *ok* and create the test
- The test class is created in the same package as the class you want to test. Some dependencies might be broken. If so, just delete those lines with errors.
- In the project window, click on your project and go to *File>Project Structure*

- Click on *Libraries*. If *JUnit5* is not yet present there, click on the *plus* symbol to add a new project library and choose *Java*.
- Find the file in the *lib* folder where you chose to download it and choose *junit-jupiter-api-5.3.1.jar*. The version number may vary.
- Click on ok. The library is now connected to our project and we can create as many tests as we want.
- Write `@Test` over the test method and import the test library from *junit jupiter*.

The screenshot shows an IDE with a Java test class `EnglishGentlemanTest` and its execution results. The code in the editor is as follows:

```

1 package challenge.test;
2
3
4 import org.junit.jupiter.api.Assertions;
5 import org.junit.jupiter.api.Test;
6
7 class EnglishGentlemanTest {
8
9     EnglishGentleman gentleman = new EnglishGentleman();
10
11     @Test
12     void greet() {
13         String greeting = gentleman.greet();
14         Assertions.assertEquals( expected: "Good evening.", greeting, message: "Greetings do not match.");
15     }
16 }

```

Below the code editor, the `Run` tab shows the test results for `EnglishGentlemanTest.greet()`. The test passed successfully, taking 14 ms. The output pane shows the message: "Greetings do not match." and the process finished with exit code 0.

## Writing tests

In order to know which tests are recommended to be done, think about the **desirable cases** (usually only one) and the **undesirable cases** (usually from one to three, but it depends on the problem).

## Available questions

These are the most useful test methods so that we can ensure that the outputs of our classes behave exactly how we wanted:

- `assertTrue(actual, message)`
- `assertFalse(actual, message)`
- `assertEquals(expected, actual, message)`
- `assertNotEquals(expected, actual, message)`
- `assertNull(actual, message)`
- `assertNotNull(actual, message)`