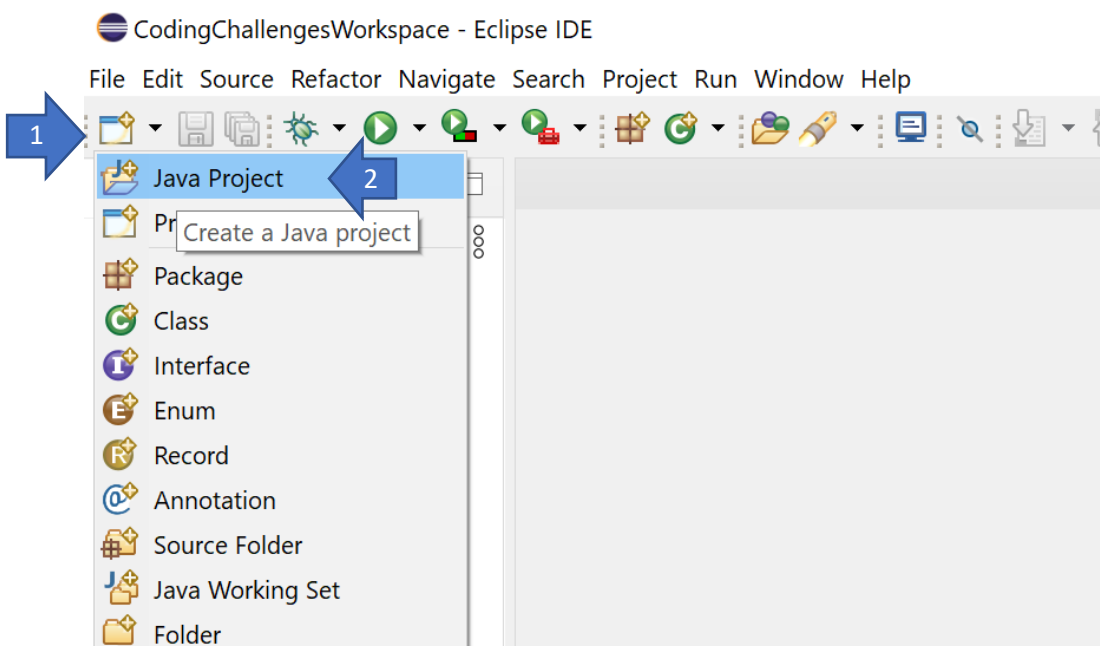


Setting up Eclipse to use JUnit

When coding challenges have a provided JUnit test, it is required that you pass all tests to pass the coding challenge. Make sure to run the JUnit tests before submitting. We don't provide any information in D2L about which tests failed. So run the tests yourself before submitting. If you need feedback on why a test failed, the D2L is not the place to get this feedback. Instead, come to class, make an appointment with someone on the teaching team, or post a question on the discussion board with your questions.

Follow the steps in this document to setup an Eclipse project that uses JUnit tests.

1. Open Eclipse and select to create a new Java Project:



2. In the wizard, give the project a name, un-select *create module-info.java* and press the *next* button.

New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: 1

☒ Use default location

Location: Browse...

JRE

☒ Use an execution environment JRE:

☐ Use a project specific JRE:

☐ Use default JRE 'jdk-16.0.2' and workspace compiler preferences [Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files [Configure default...](#)

Working sets

☐ Add project to working sets

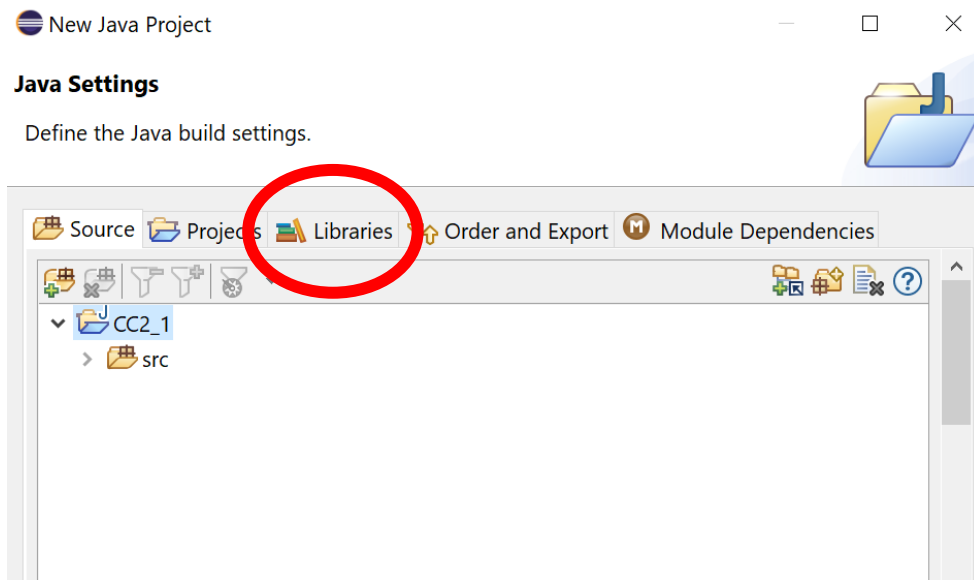
Working sets:

Module

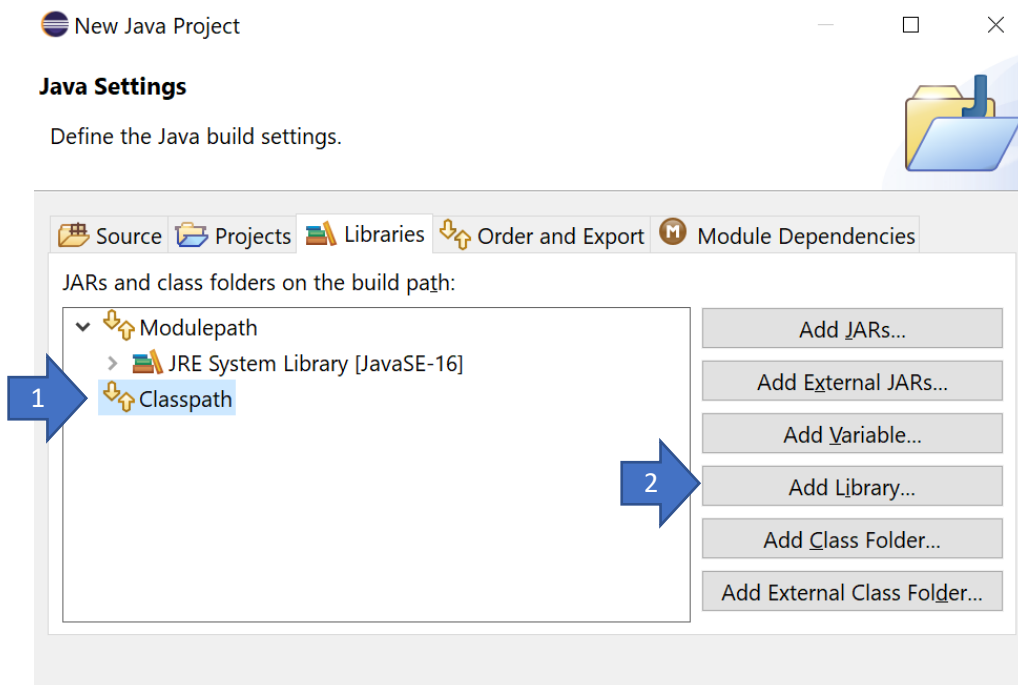
2 ☐ Create module-info.java file

3

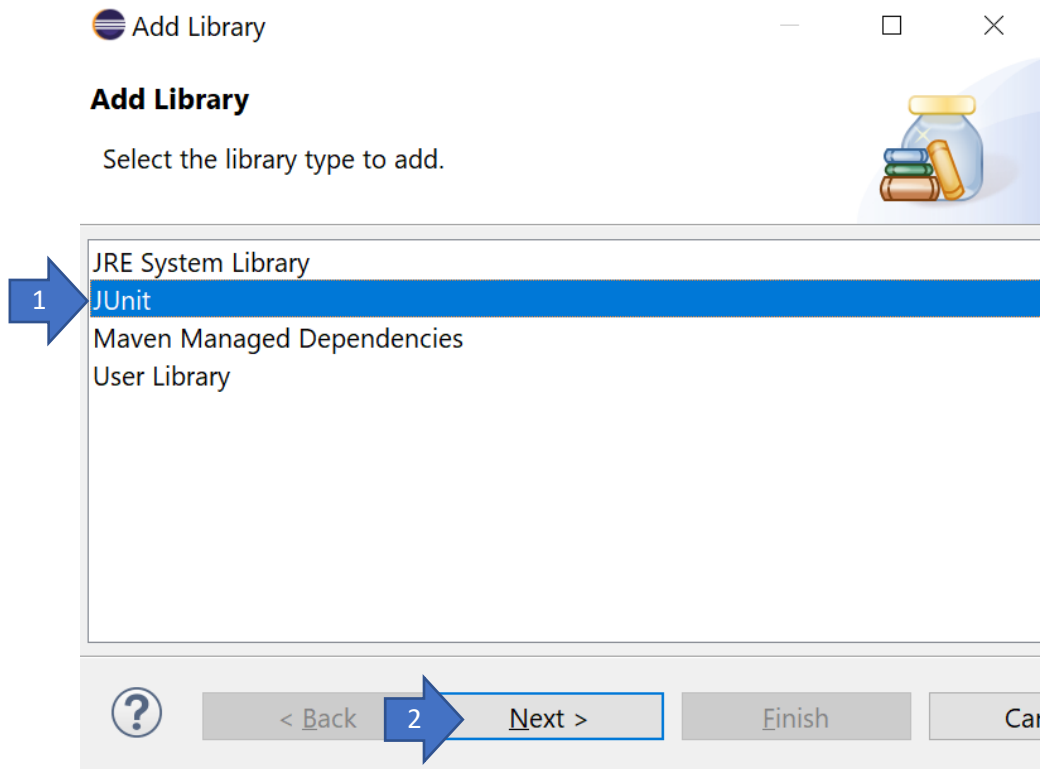
3. Select the *Libraries* tab.



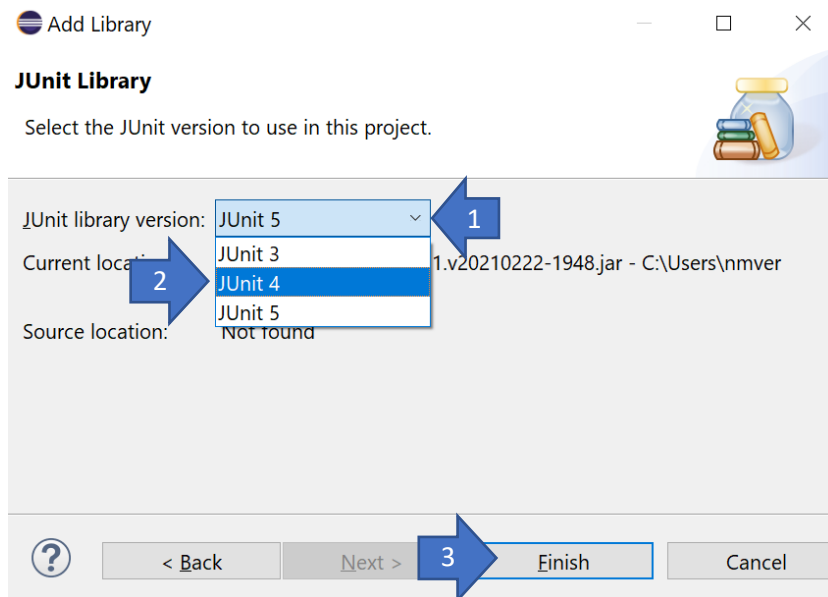
4. Select *Classpath* and then press the *Add Library...* button.



5. Select *JUnit* from the list and press *Next*.

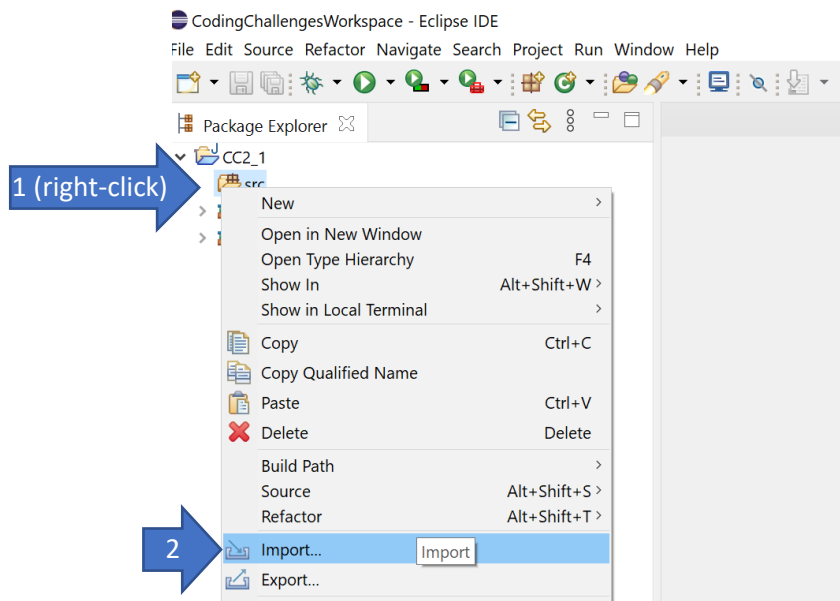


6. Select *JUnit 4* from the dropdown list and press *Finish*.

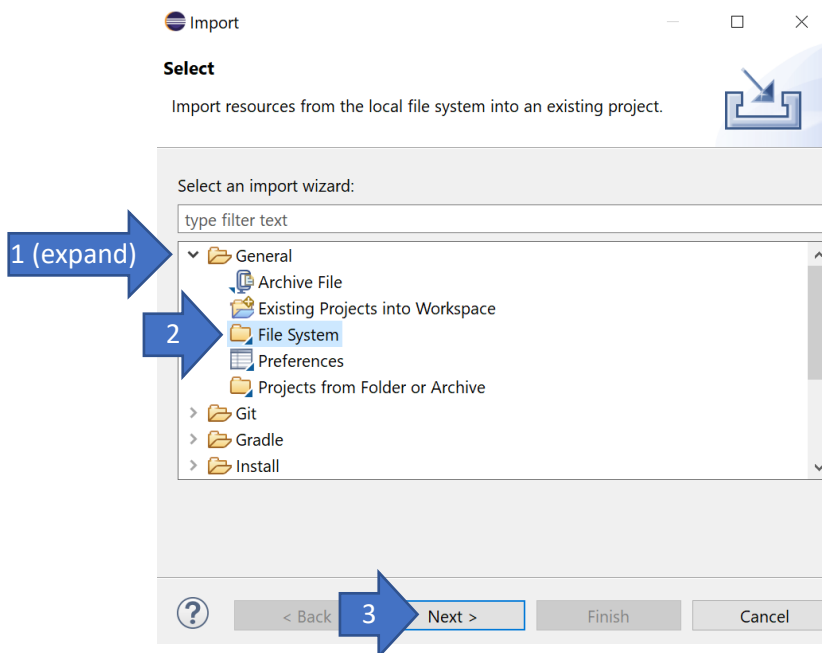


To test your setup, you'll need a JUnit test class. For the first coding challenge that uses JUnit, a skeleton of the Java class you are required to create will also be provided. These steps show how to use run JUnit tests for this first coding challenge.

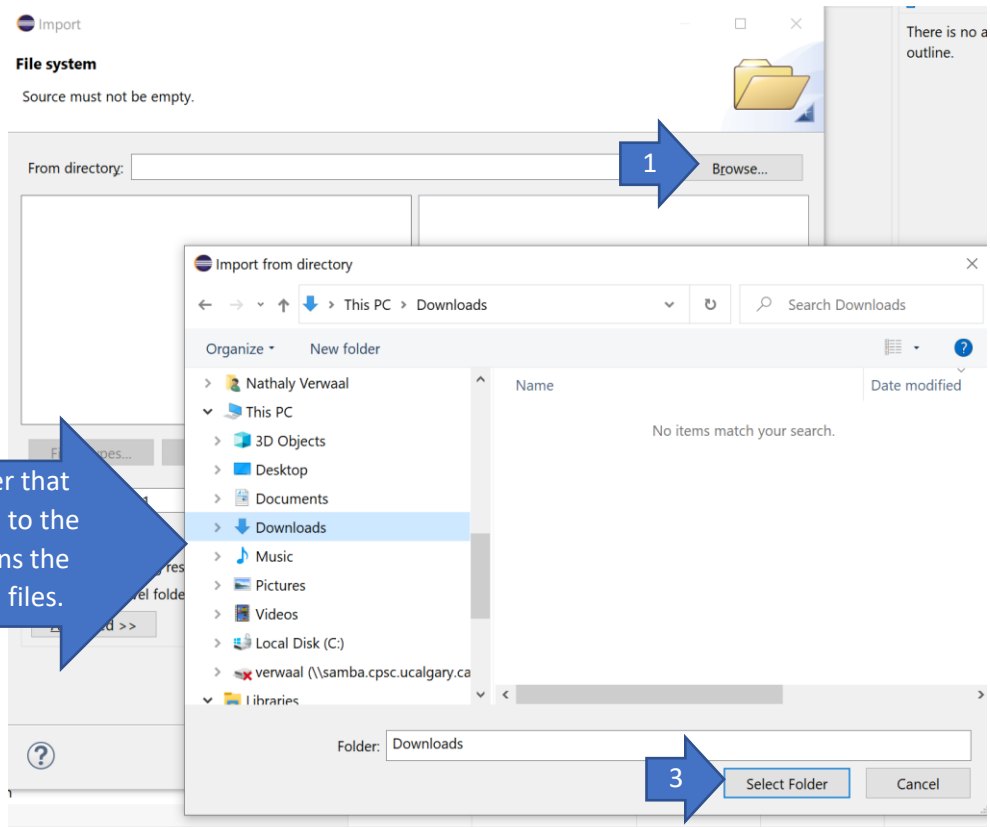
7. Download the JUnit class (CodingChallenge2_1Test.java) and the skeleton class (CodingChallenge2_1.java) from D2L.
8. To import these two files in your new JUnit project, right-click on the *src* folder inside project you just created and select *import...* from the dropdown list.



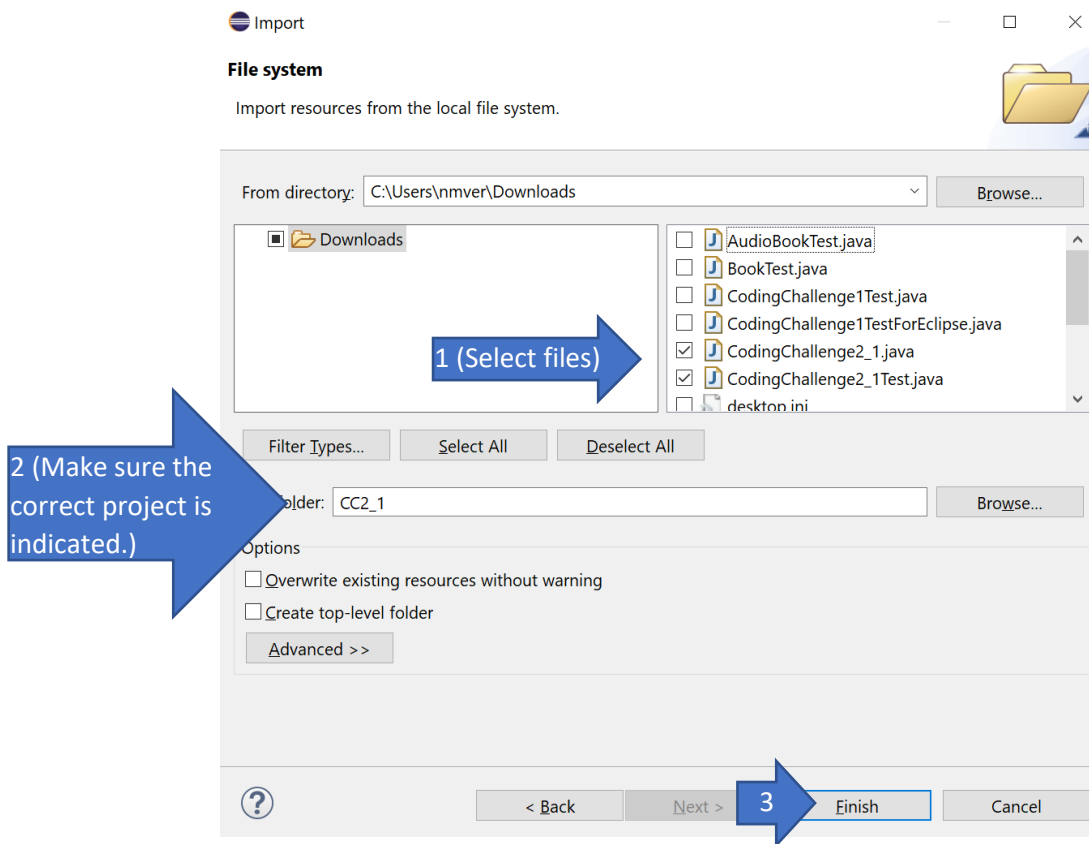
9. In the import wizard, expand *General*, select *File System*, and press *next*.



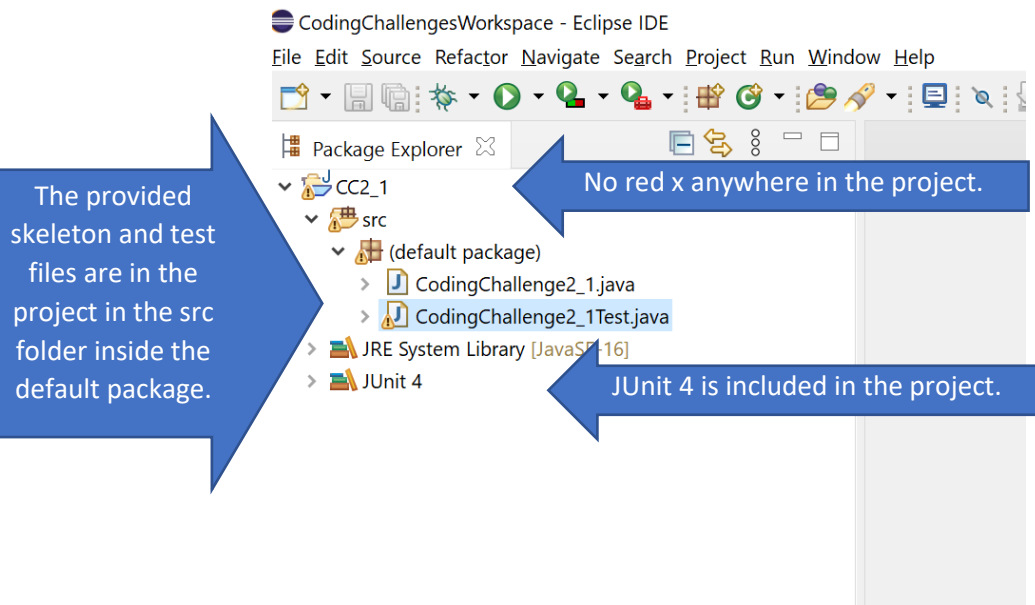
10. Browse to the folder that you downloaded the provided Java files to. (You are expected to browse to the folder. Only folders will be shown in the browser window.)



11. Select the Java files to import, make sure that it will be imported into your new project, and press *Finish*.

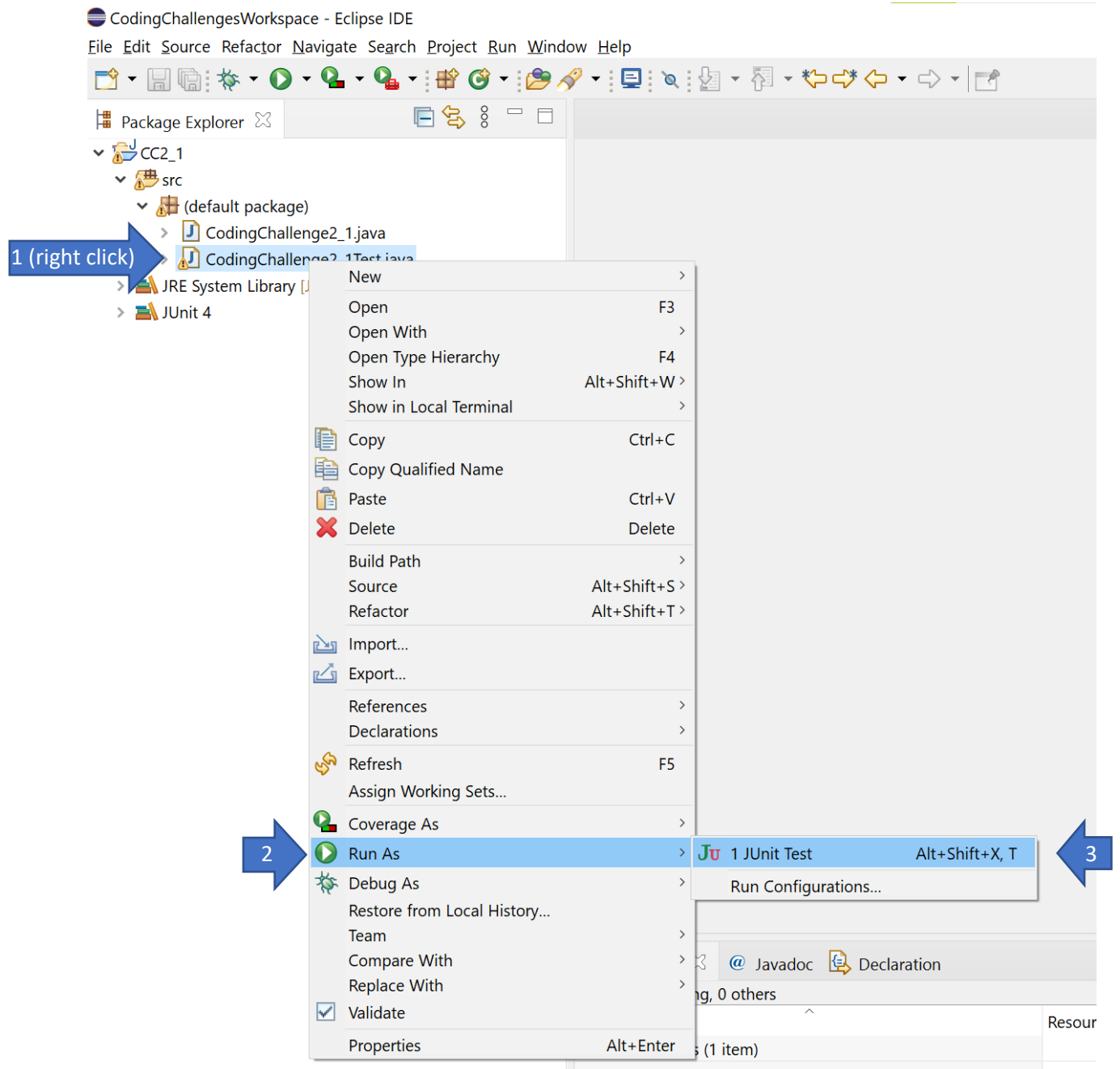


12. Your project should now look as follows:

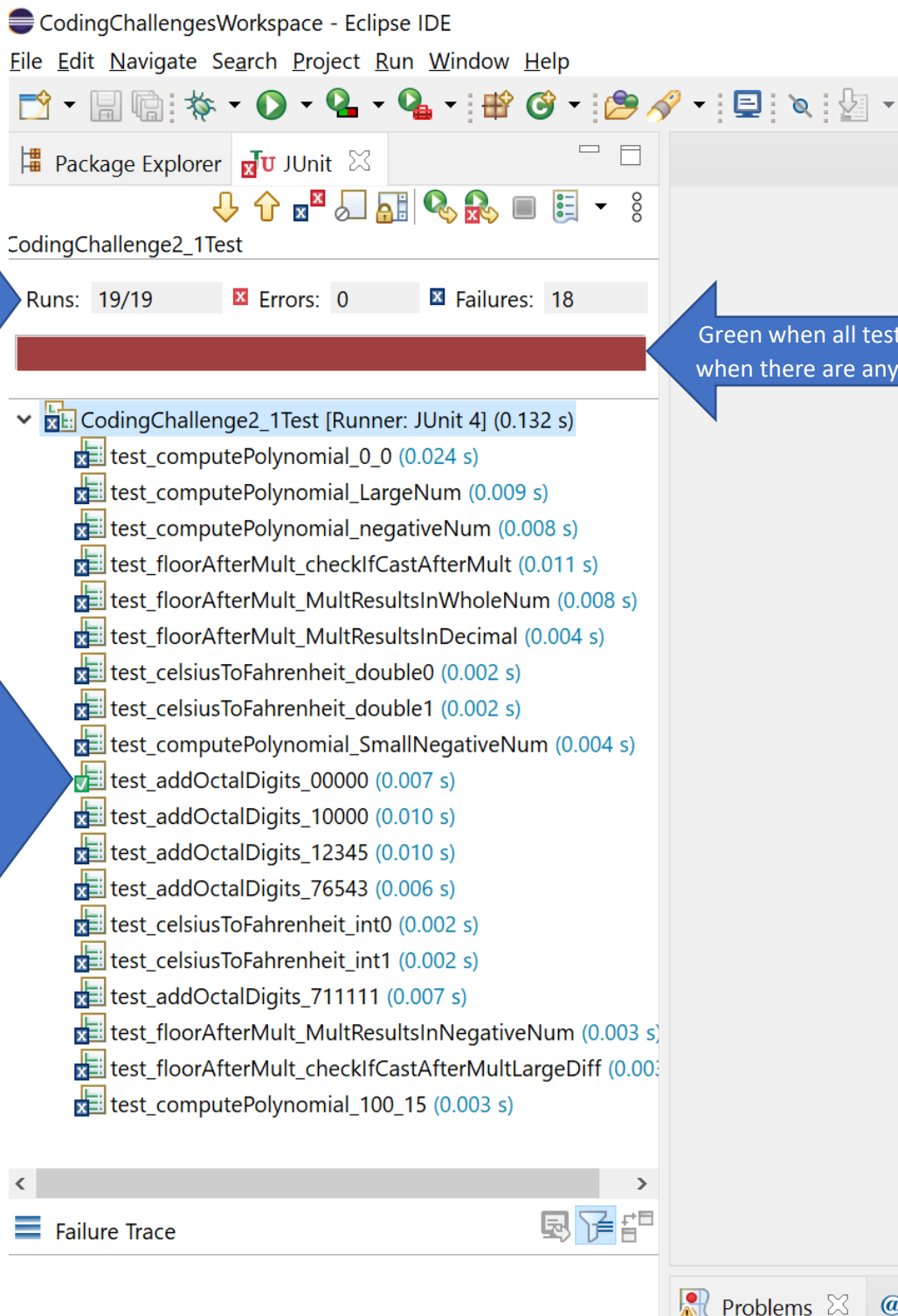


To test that the project is setup correct, you can run the tests provided as follows. (Note that almost all the tests will fail since you haven't written any code yet.)

13. Right-click on the test class, select *Run As* from the dropdown menu, then press *JUnit Test* from the next menu.



14. This should open the JUnit window and show the tests that passed and the tests that failed.



Indicates the number of tests run. Make sure that all tests ran!

Green when all tests pass. Red when there are any failed tests.

Green checkmark indicates that this test passed (and it is the only test that passed).

15. You can find more information about a test in the Failure Trace window about a failed test by selecting the failure.

The screenshot shows the 'Failure Trace' window in an IDE. At the top, a list of failed tests is displayed. A blue arrow labeled 'Select a failed test.' points to the first test in the list: `test_computePolynomial_100_15 (0.003 s)`. Below the list, the details for the selected test are shown. A blue arrow labeled 'We called the method with value 100.15' points to the first line of the failure message: `java.lang.AssertionError: Value of polynomial at 100.15 expected:<9857.7225> but was:<0.0>`. Another blue arrow labeled 'Expected return value.' points to the second line of the failure message: `at CodingChallenge2_1Test.test_computePolynomial_100_15(CodingChallenge2_1Test.java:138)`. A third blue arrow labeled 'Actual return value.' points to the third line of the failure message: `at CodingChallenge2_1Test.test_computePolynomial_100_15(CodingChallenge2_1Test.java:138)`. The 'Failure Trace' window title is visible at the bottom left of the panel.

Select a failed test.

We called the method with value 100.15

Expected return value.

Actual return value.

Failure Trace

java.lang.AssertionError: Value of polynomial at 100.15 expected:<9857.7225> but was:<0.0>
at CodingChallenge2_1Test.test_computePolynomial_100_15(CodingChallenge2_1Test.java:138)