School of Computer Science Engineering and Technology

Course- BTech	Type- Core
Course Code- CSET205	Course Name-Software Engineering
Year-2022	Semester- Odd
Date-11/11/2022	Batch- 2021-2025

Lab Assignment No. 10 – Unit Testing Using JUnit Framework

CO Mapping

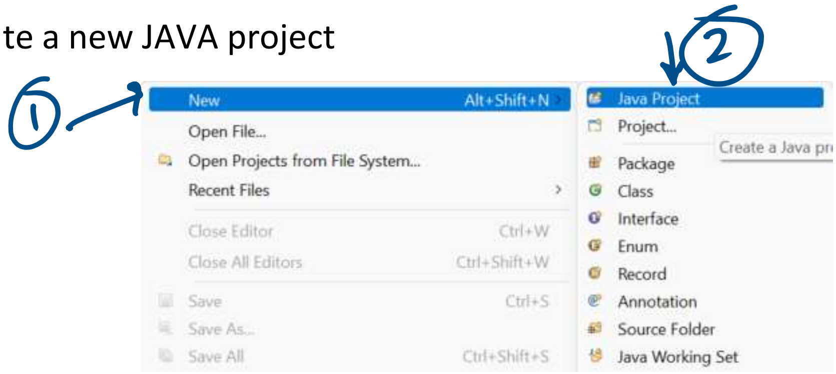
Exp. No.	Name	CO1	CO2	CO3
10	Unit Testing Using JUnit Framework	√	√	√

This exercise enables the understanding of how to write automated test cases and test suite (that runs multiple test cases in one go). JUnit, as the term suggests, is *used for unit testing by individual developers* in the industry. The test case(s)/test suite(s) is needed to run whenever you make any changes to your code. It ensures that your code is error-free and can provide the required service(s).

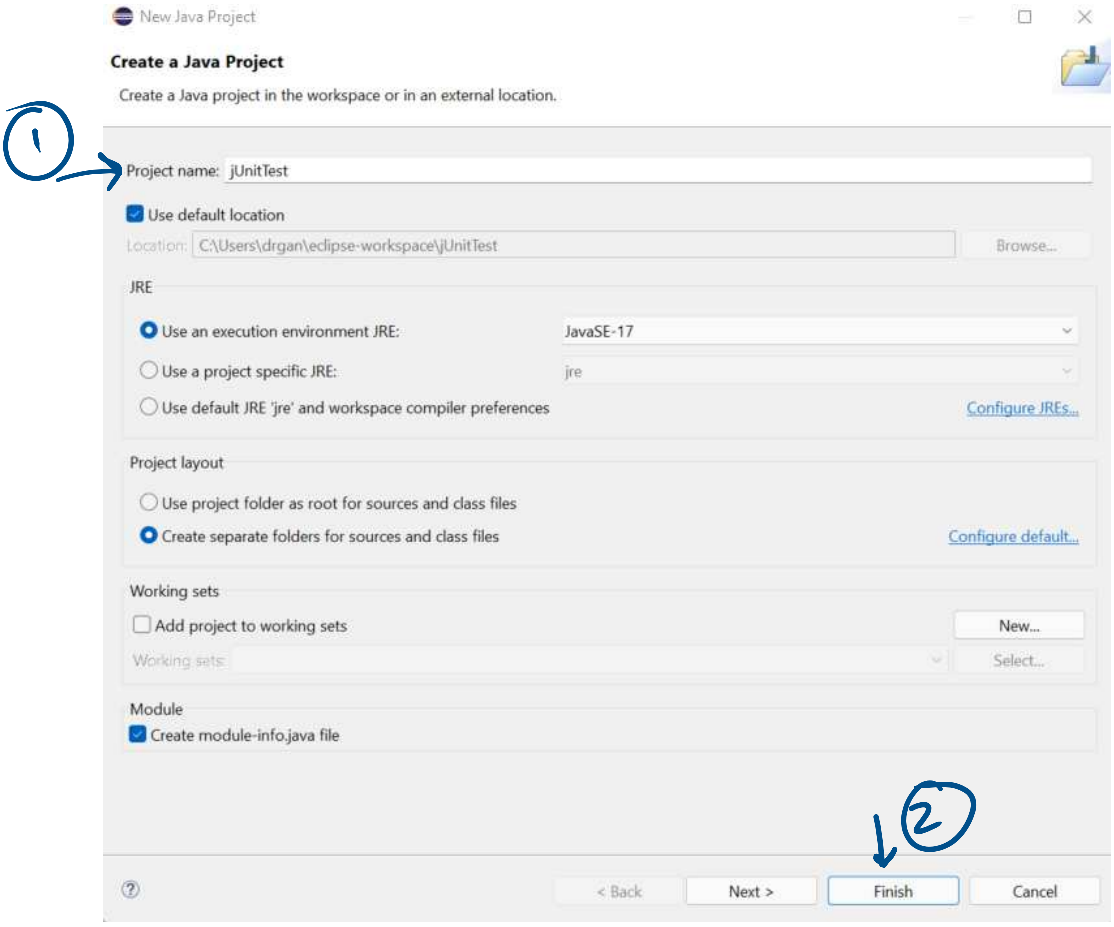
Let's assume that our team leader gave us the task to develop two functions where we have to add two numbers and two words. As a developer, it's our responsibility to produce quality software code (on time, free from errors). Hence, the following tasks are required to achieve that via a JUnit testing framework:

1. Create a new Java project (e.g., ***JUnitTestExample***) and share it on your GitHub repository.
2. Create a class (e.g., ***JUnitTesting***)
3. Create two Java methods (e.g., ***AddNumbers*** and ***AddStrings***)
4. Define the ***AddNumbers*** method where it adds two numbers.
5. Similarly, define ***AddStrings*** method where it adds two strings
6. Create a JUnit test case for each method (***TestAddNumbers*** and ***TestAddStrings***). Check snapshots below to follow:

1. Create a new JAVA project



2. Give your project a name and click the **finish** button



Create a Java Project

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

Project name: jUnitTest

☑ Use default location

Location: C:\Users\drgan\eclipse-workspace\jUnitTest

JRE

● Use an execution environment JRE: JavaSE-17

○ Use a project specific JRE: jre

○ Use default JRE 'jre' and workspace compiler preferences

Project layout

○ Use project folder as root for sources and class files

● Create separate folders for sources and class files

Working sets

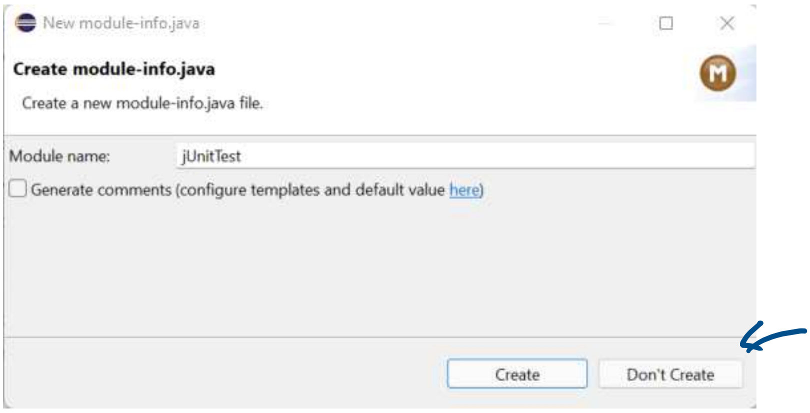
□ Add project to working sets

Working sets:

Module

☑ Create module-info.java file

3. If it asks for new module creation, click on the **Don't create** button.



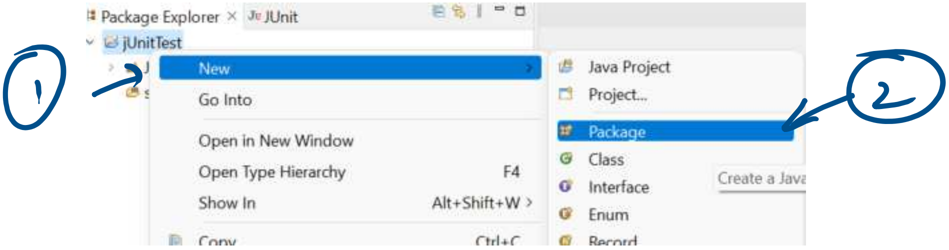
Create module-info.java

Create a new module-info.java file.

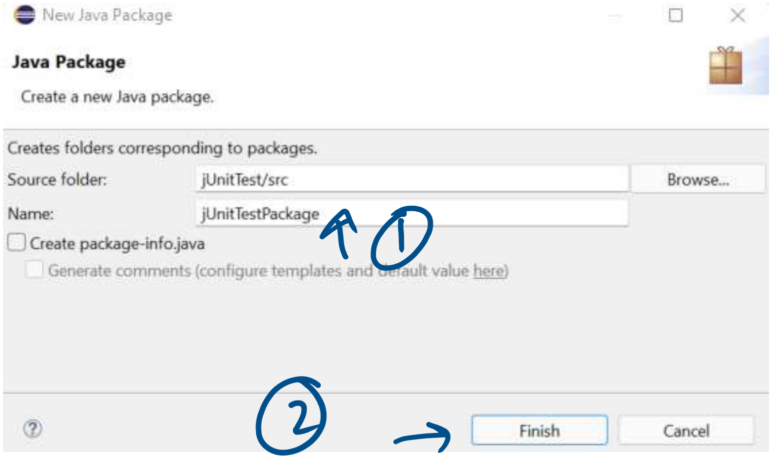
Module name: jUnitTest

□ Generate comments (configure templates and default value here)

4. Right-click on your project name, go to **new** → click on the **Package** option



5. Give the package a name and click the **finish** button.


New Java Package

Java Package

Create a new Java package.

Creates folders corresponding to packages.

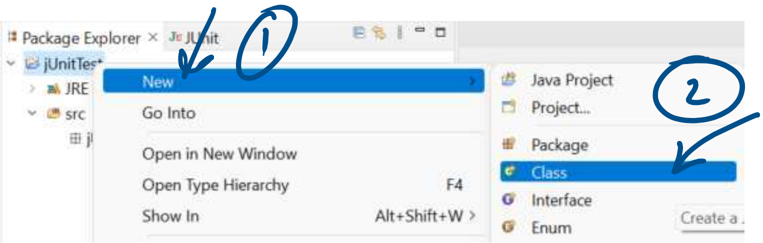
Source folder: jUnitTest/src

Name: jUnitTestPackage

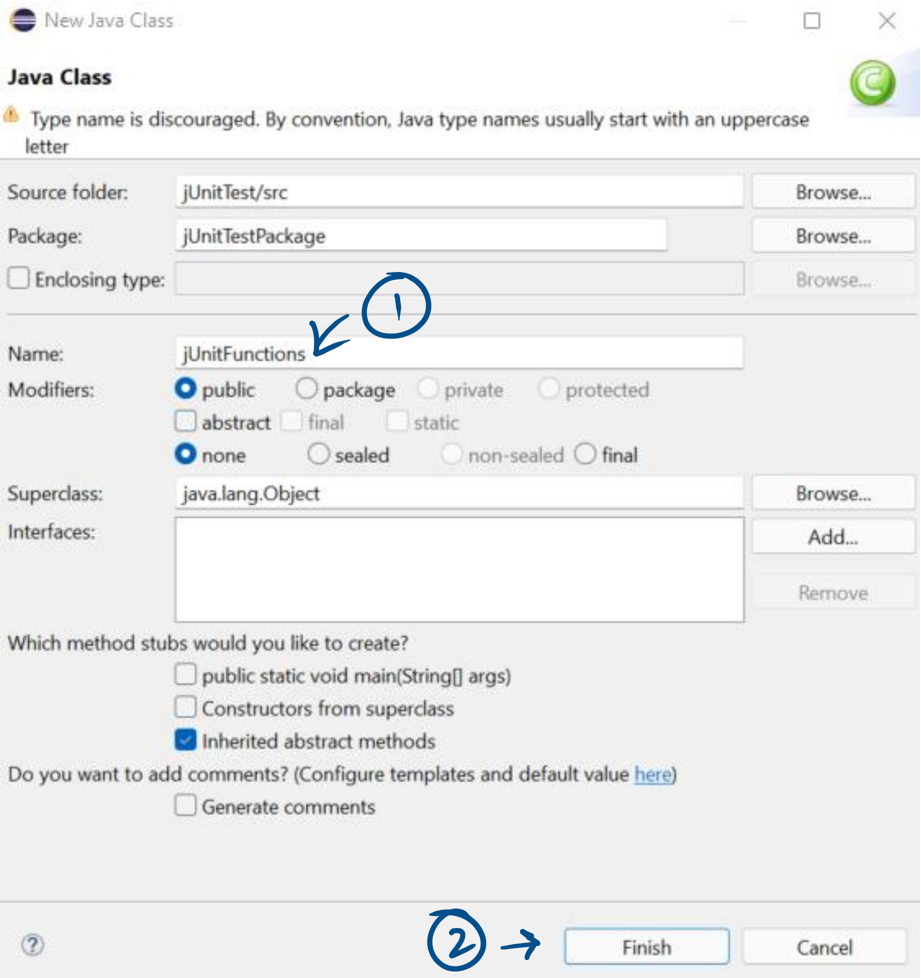
□ Create package-info.java

□ Generate comments (configure templates and default value here)

6. Create a **class** file under your project.



7. Give it a name and click on the **finish** button.



New Java Class

Java Class

⚠ Type name is discouraged. By convention, Java type names usually start with an uppercase letter

Source folder: jUnitTest/src

Package: jUnitTestPackage

□ Enclosing type:

Name: jUnitFunctions

Modifiers: ● public ○ package ○ private ○ protected
□ abstract □ final □ static
● none ○ sealed ○ non-sealed ○ final

Superclass: java.lang.Object

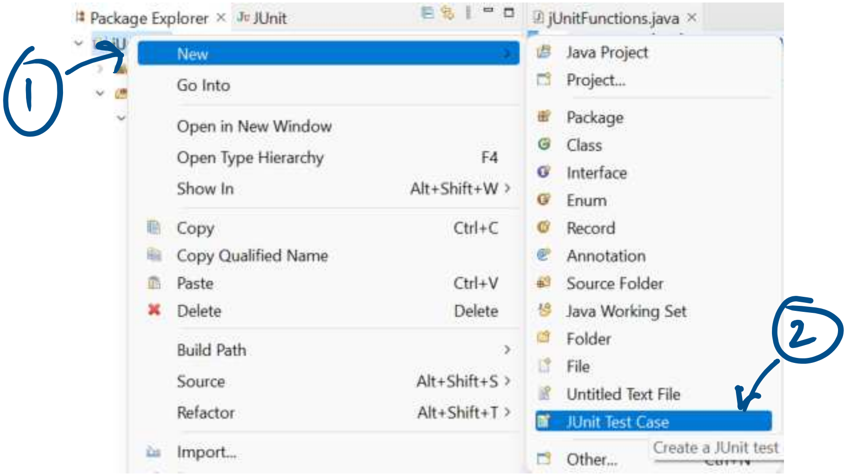
Interfaces:

Which method stubs would you like to create?
□ public static void main(String[] args)
□ Constructors from superclass
☑ Inherited abstract methods

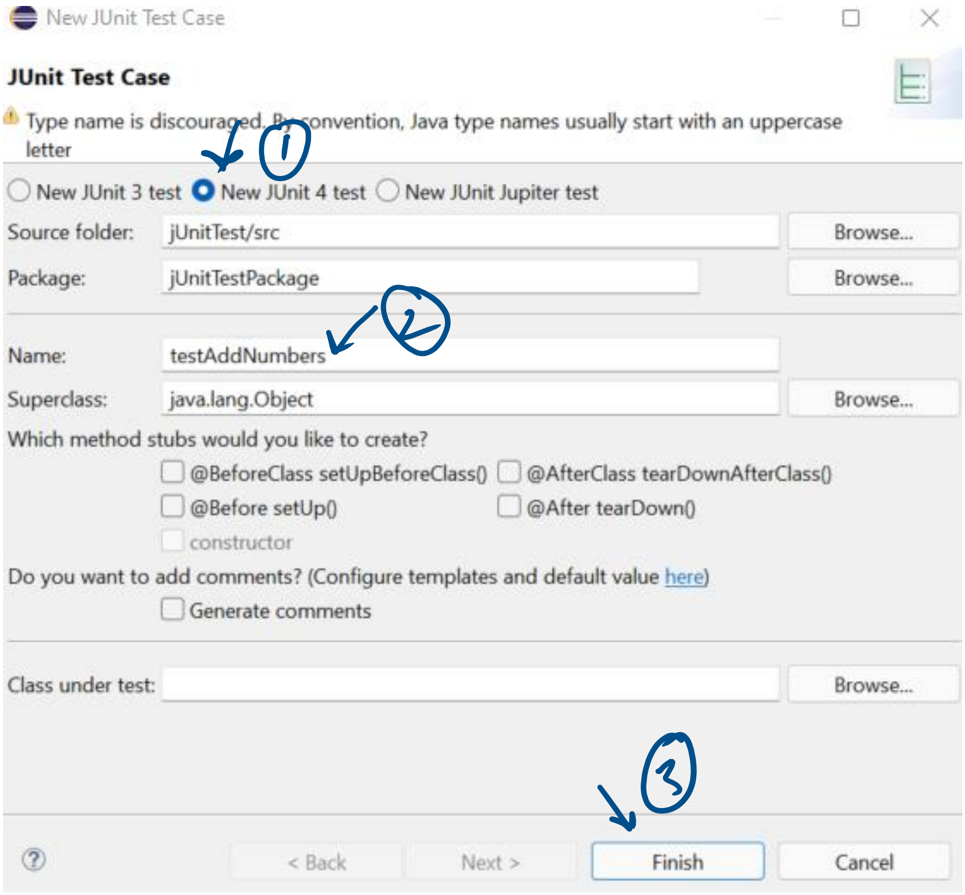
Do you want to add comments? (Configure templates and default value here)
□ Generate comments

8. Create your JAVA program that will be tested in later steps. **Share the project in your Git repository as you did for Lab 04.**

9. Right-click on your project, go to **new →** select **Junit Test Case**



10.Select the **New Junit 4 test** option, give your test case a name, and click the **finish** button.



New JUnit Test Case

JUnit Test Case

Type name is discouraged. By convention, Java type names usually start with an uppercase letter

○ New JUnit 3 test ● New JUnit 4 test ○ New JUnit Jupiter test

Source folder: jUnitTest/src

Package: jUnitTestPackage

Name: testAddNumbers

Superclass: java.lang.Object

Which method stubs would you like to create?

□ @BeforeClass setUpBeforeClass() □ @AfterClass tearDownAfterClass()

□ @Before setUp() □ @After tearDown()

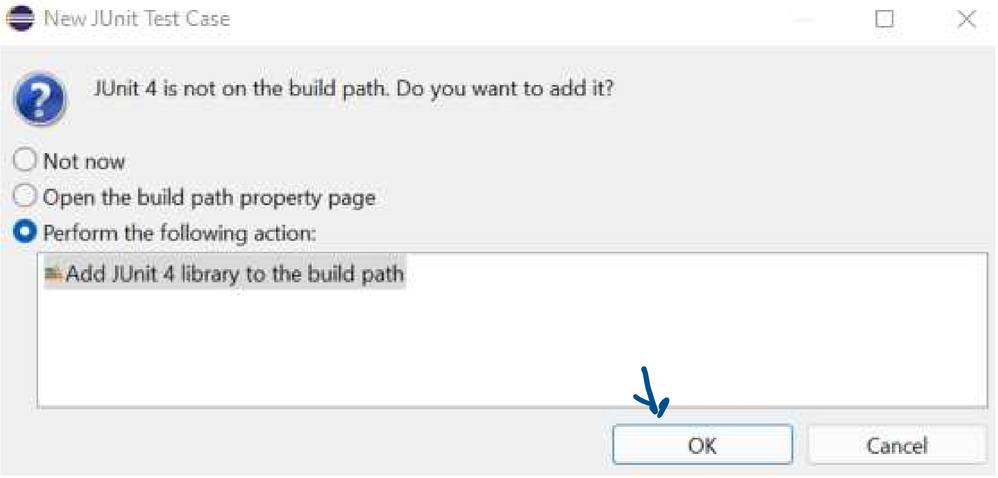
□ constructor

Do you want to add comments? (Configure templates and default value here)

□ Generate comments

Class under test:

11. If **Junit 4** is not on the build path, then add it by clicking the **OK** button.



New JUnit Test Case

JUnit 4 is not on the build path. Do you want to add it?

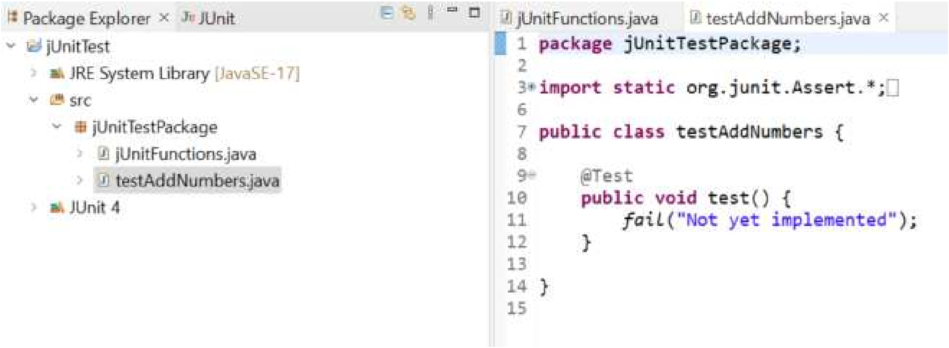
○ Not now

○ Open the build path property page

● Perform the following action:

Add JUnit 4 library to the build path

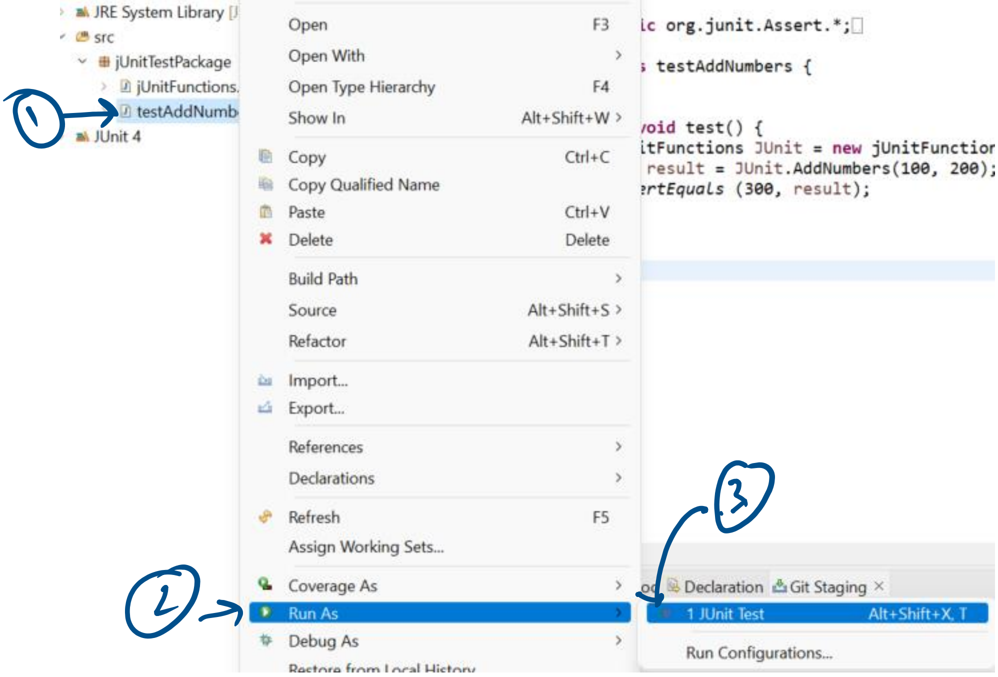
12. You will see that a new JAVA class has been created. Now you have to add the test case to check your program.



```
package jUnitTestPackage;

import static org.junit.Assert.*;

public class testAddNumbers {

    @Test
    public void test() {
        fail("Not yet implemented");
    }

}
```

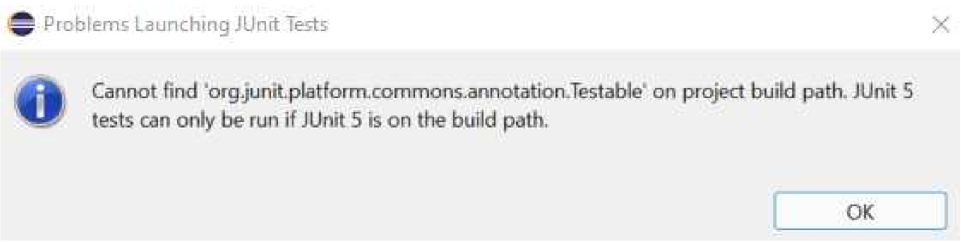
13. Repeat steps 9 – 12 to create all the test cases. In this case, we have two methods to test. Hence, two test cases will be created.

14. To run each test case, right-click on the test case file, and go to **Run As → Junit Test**.



```
ic org.junit.Assert.*;

 testAddNumbers {

void test() {
itFunctions JUnit = new jUnitFunctior
 result = JUnit.AddNumbers(100, 200);
ertEquals (300, result);
```

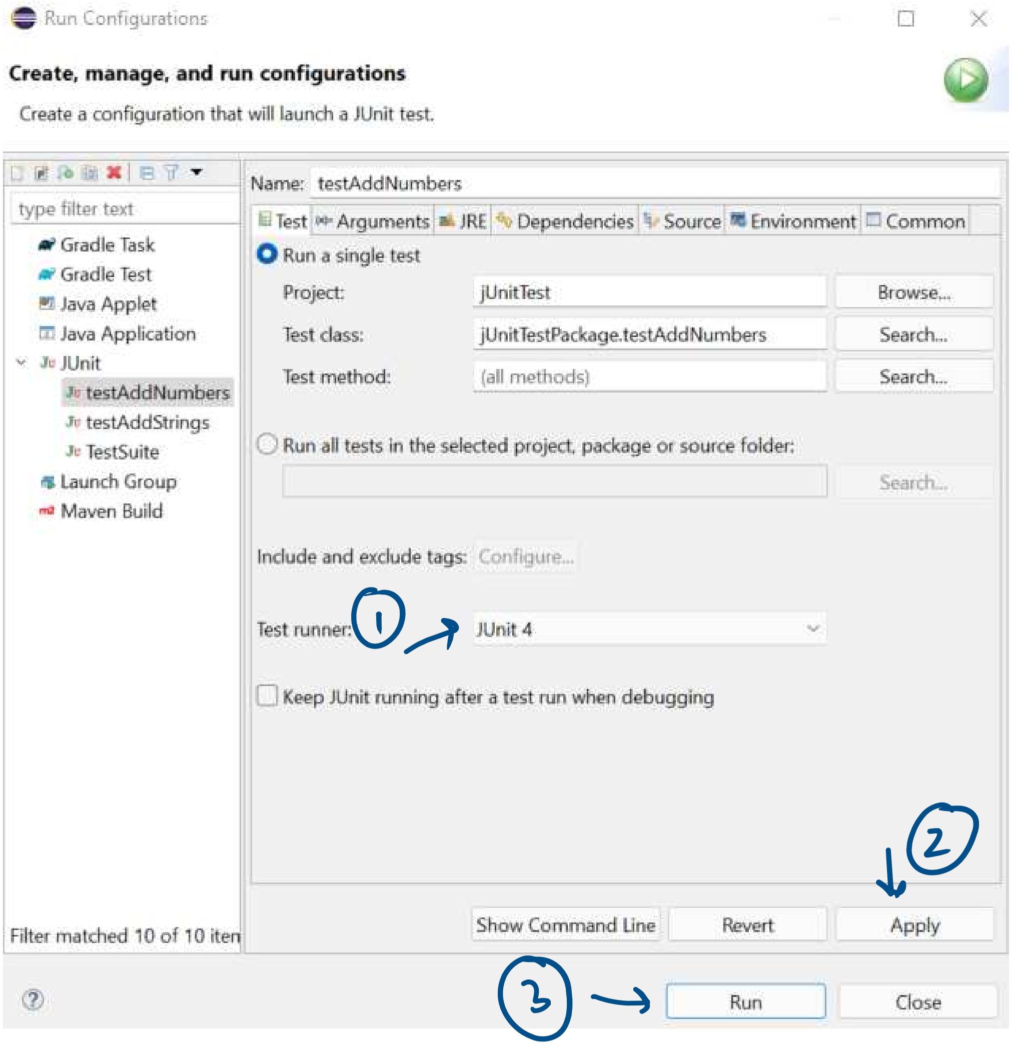
15. If there’s a message for Junit 5, click on the **OK** button.



Cannot find 'org.junit.platform.commons.annotation.Testable' on project build path. JUnit 5 tests can only be run if JUnit 5 is on the build path.

16. In the run configuration, select Test runner as Junit 4 from the dropdown.

    Click on **Apply** button and then the **Run** button.



Run Configurations

Create, manage, and run configurations

Create a configuration that will launch a JUnit test.

Name: testAddNumbers

● Run a single test

Project: jUnitTest

Test class: jUnitTestPackage.testAddNumbers

Test method: (all methods)

○ Run all tests in the selected project, package or source folder:

Include and exclude tags: Configure...

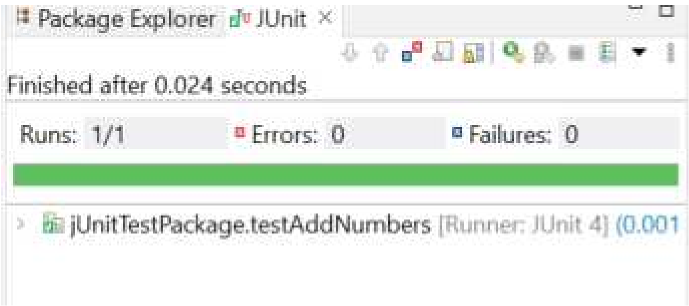
Test runner: JUnit 4

□ Keep JUnit running after a test run when debugging

Filter matched 10 of 10 iten

17. On the left pane of the project, you will see the results of the execution.

    Green means PASS, and red means FAIL.



Finished after 0.024 seconds

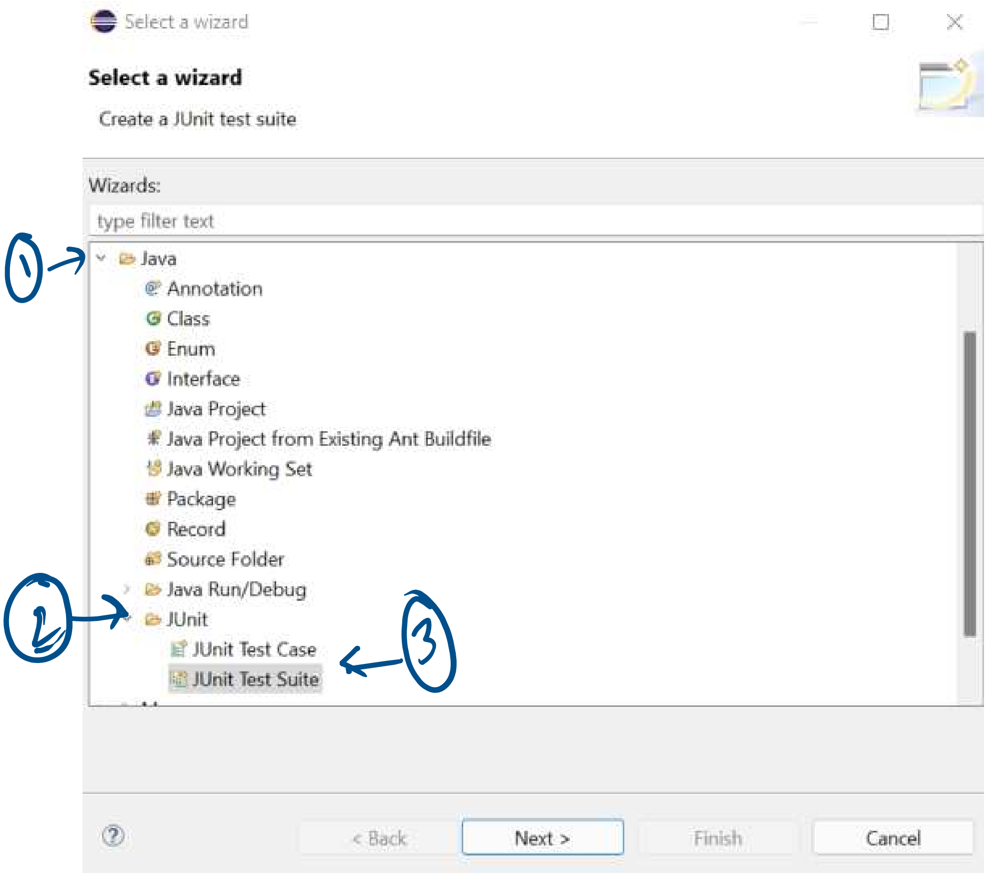
Runs: 1/1 Errors: 0 Failures: 0

jUnitTestPackage.testAddNumbers [Runner: JUnit 4] (0.001

18. Now, let’s add both the test cases to a **test suite.**

19.Right-click on the project, **New → Other…**

20.Select **Junit Test Suite** from the wizard and click on the **next** button.



Select a wizard

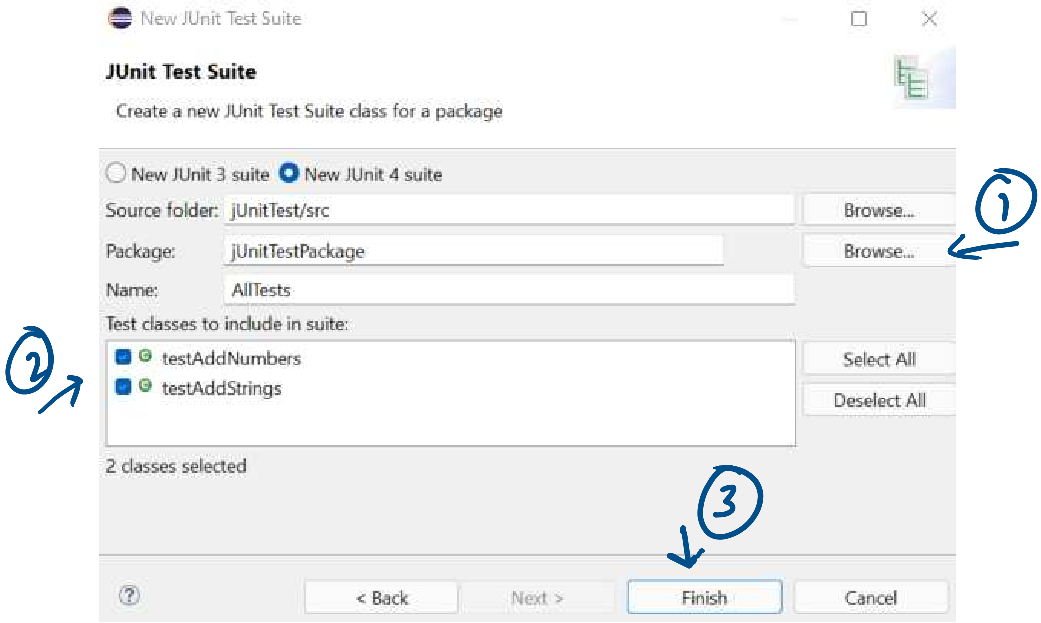
Select a wizard

Create a JUnit test suite

Wizards:

type filter text

21.In the Junit Test Suite window, select the package of your project by clicking the **Browse** button. You will see both the test cases are now listed. Select them and click on the **finish** button.



New JUnit Test Suite

JUnit Test Suite

Create a new JUnit Test Suite class for a package

○ New JUnit 3 suite ● New JUnit 4 suite

Source folder: jUnitTest/src

Package: jUnitTestPackage

Name: AllTests

Test classes to include in suite:

☑ testAddNumbers

☑ testAddStrings

2 classes selected

22. Right-click on the Junit Test Suite file that you have created just now and follow the same steps (as you did for each test case) to execute the Junit Test Suite.

7. Run JUnit test suite with both correct and incorrect code
8. Push your project to the repository (like you did in lab 4).

You have to submit the following **five** files:

1. Java file (that consists of both the methods)
2. JUnit test case for AddNumbers
3. JUnit test case for AddStrings
4. JUnit test suite
5. Link to your repository (in a text file)

Reference: https://youtu.be/YsdGvxt1IqI