

Lab 1: Basic Unit Testing

Software Testing 2021

2021/03/04

Overview

Lab overview

1. 10 ~ 11 Lab assignments
 - a. *Unit Testing*
 - b. *Continuous Integration*
 - c. *Web Applications Testing*
 - d. *Control Flow Graph*
 - e. *Behavior-Driven Development*
 - f. *Fuzz Testing*
 - g. *Symbolic Execution*
2. Homework: TBD

Github

- <https://github.com/iasthc/NYCU-Software-Testing-2021>

JAVA IDE

IntelliJ IDEA Community

→ <https://www.jetbrains.com/idea/download>

The screenshot shows the IntelliJ IDEA download page on the JetBrains website. The top navigation bar includes links for Developer Tools, Team Tools, Learning Tools, Solutions, Store, and a search bar. A prominent 'Download' button is located in the top right corner.

The main content area features the IntelliJ logo and version information: Version: 2020.3.2, Build: 203.7148.57, 26 January 2021, with links to Release notes and System requirements.

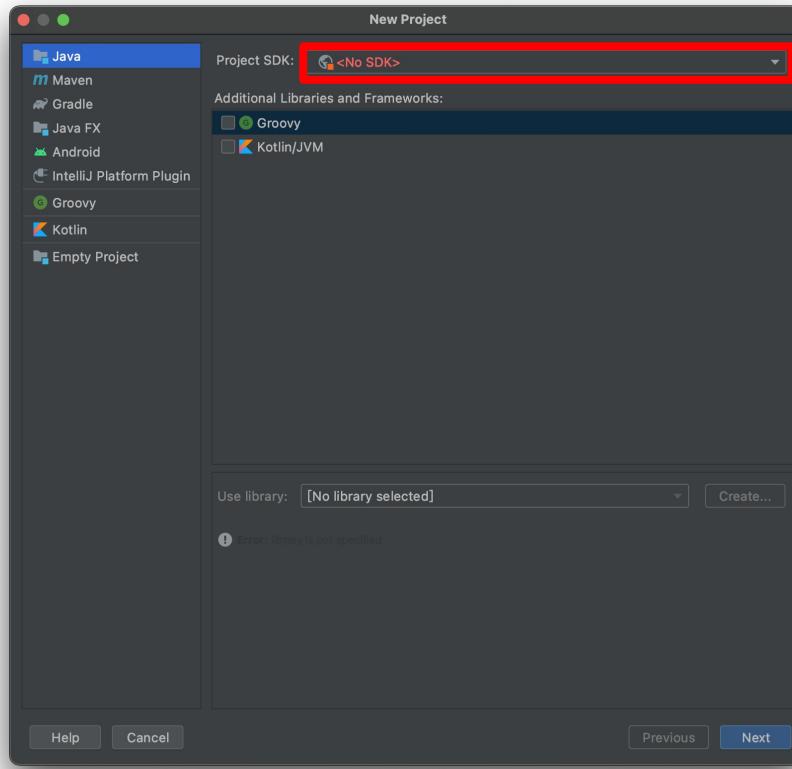
Two download options are presented:

- Ultimate**: For web and enterprise development. Available for Windows, Mac, and Linux. Includes a 'Download' button and a '.dmg (Intel)' dropdown.
- Community**: For JVM and Android development. Free, open-source. Available for Intel and Apple Silicon. This option is highlighted with a red rectangular box. It also has a 'Download' button and a '.dmg (Intel)' dropdown.

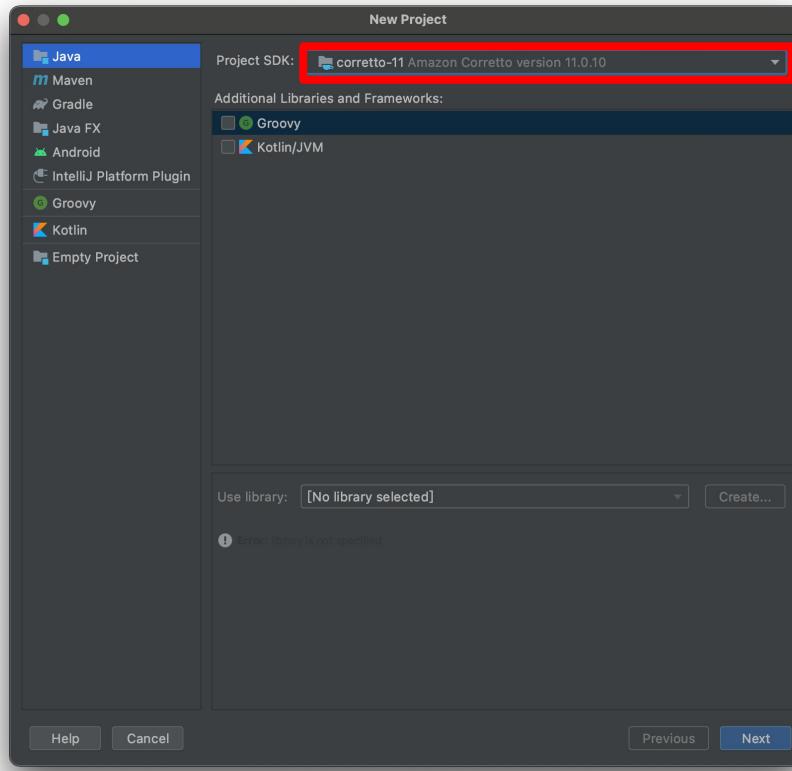
Below the download sections, there's a comparison table:

	IntelliJ IDEA Ultimate	IntelliJ IDEA Community Edition
Java, Kotlin, Groovy, Scala	✓	✓
Android ⓘ	✓	✓
Maven, Gradle, sbt	✓	✓

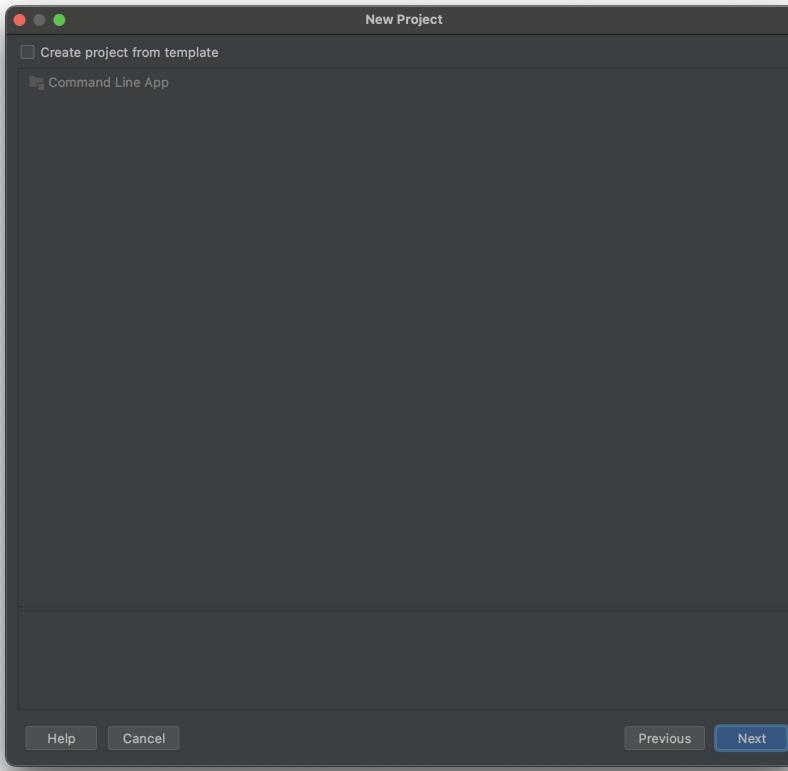
Guide - SDK



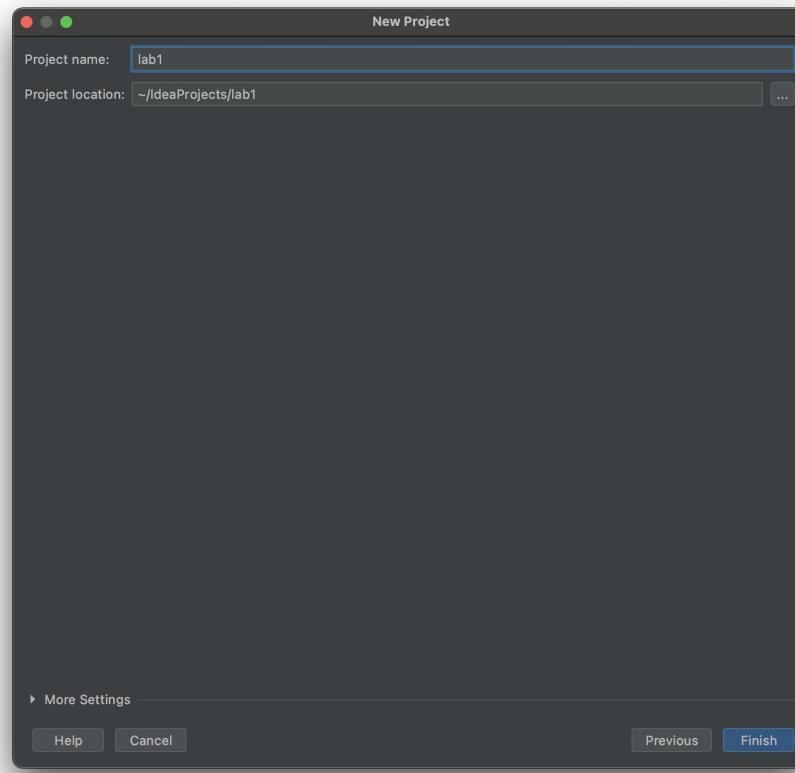
Guide - SDK



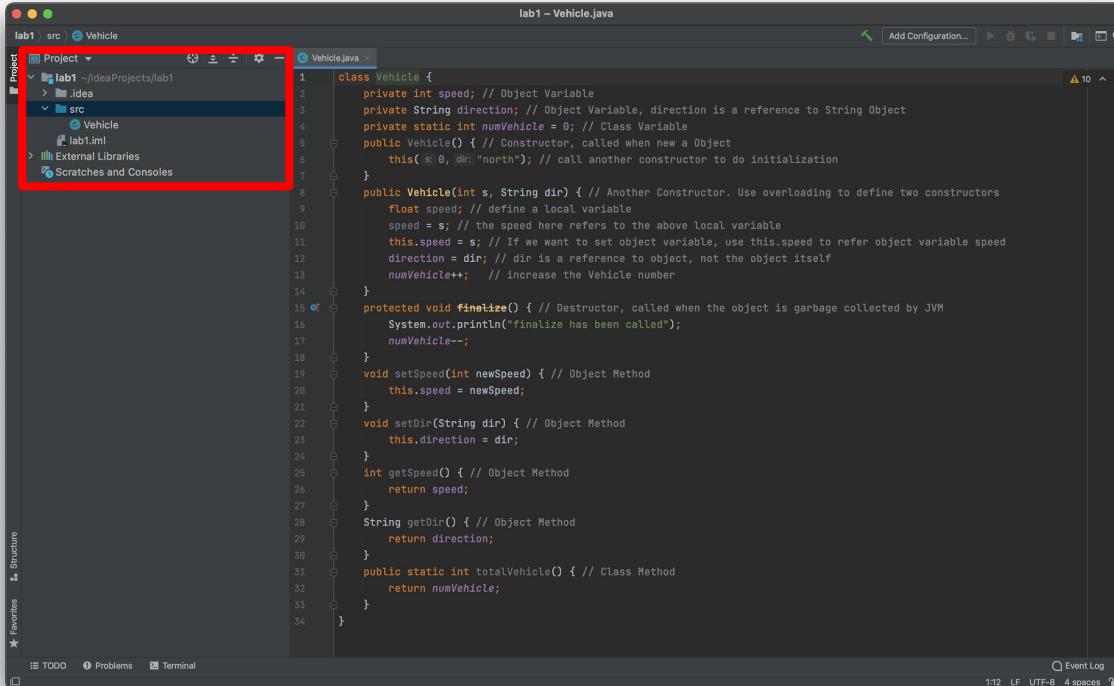
Guide - Template



Guide - Name



Guide - SRC



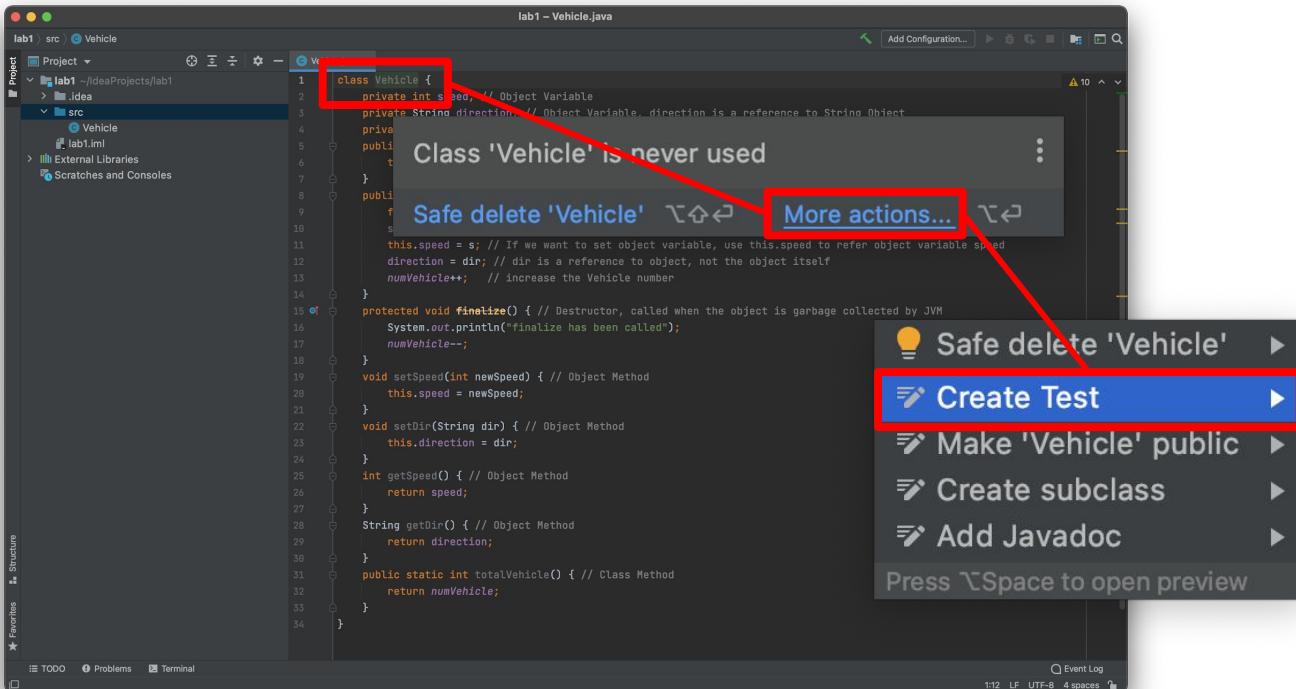
The screenshot shows the IntelliJ IDEA interface with the following details:

- Title Bar:** lab1 – Vehicle.java
- Toolbars:** Add Configuration..., Run, Stop, Build, Find, Search.
- Project View (Left):** Shows the project structure with a red box highlighting the `src` folder containing a `Vehicle` class.
- Code Editor (Center):** Displays the `Vehicle.java` file content. The code defines a `Vehicle` class with private attributes `speed` and `direction`, a static attribute `numVehicle`, and several methods including constructors, `finalize`, `setSpeed`, `setDir`, `getSpeed`, `getDir`, and a class method `totalVehicle`.
- Sidebar (Bottom Left):** Includes tabs for `Structure` and `Favorites`.
- Bottom Status Bar:** Event Log, 1:12 LF, UTF-8, 4 spaces, and a terminal icon.

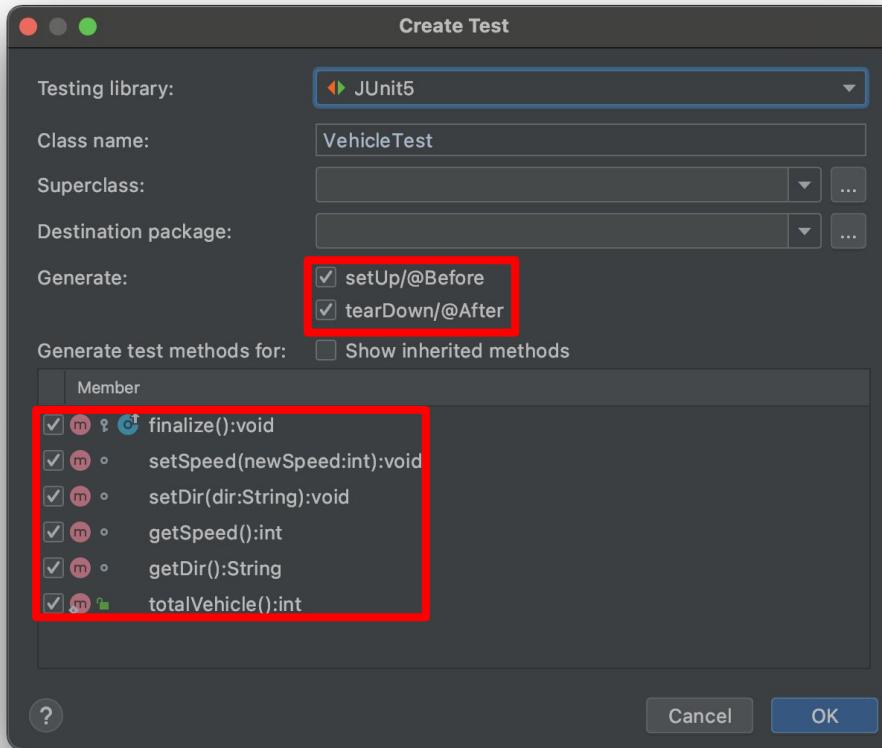
```
class Vehicle {  
    private int speed; // Object Variable  
    private String direction; // Object Variable, direction is a reference to String Object  
    private static int numVehicle = 0; // Class Variable  
    public Vehicle() { // Constructor, called when new a Object  
        this(0, "north"); // call another constructor to do initialization  
    }  
    public Vehicle(int s, String dir) { // Another Constructor. Use overloading to define two constructors  
        float speed; // define a local variable  
        speed = s; // the speed here refers to the above local variable  
        this.speed = s; // If we want to set object variable, use this.speed to refer object variable speed  
        direction = dir; // dir is a reference to object, not the object itself  
        numVehicle++; // increase the Vehicle number  
    }  
    protected void finalize() { // Destructor, called when the object is garbage collected by JVM  
        System.out.println("finalize has been called");  
        numVehicle--;  
    }  
    void setSpeed(int newSpeed) { // Object Method  
        this.speed = newSpeed;  
    }  
    void setDir(String dir) { // Object Method  
        this.direction = dir;  
    }  
    int getSpeed() { // Object Method  
        return speed;  
    }  
    String getDir() { // Object Method  
        return direction;  
    }  
    public static int totalVehicle() { // Class Method  
        return numVehicle;  
    }  
}
```

Create Test

Create Test

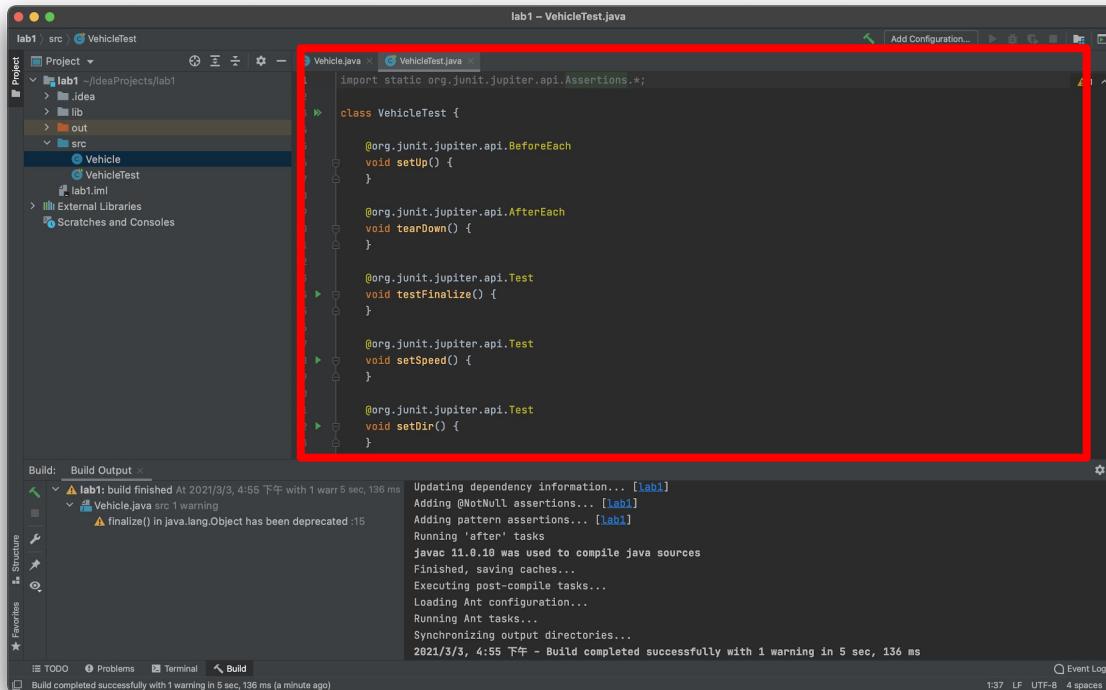


Create Test - JUnit 5



Create Test - New File

1.

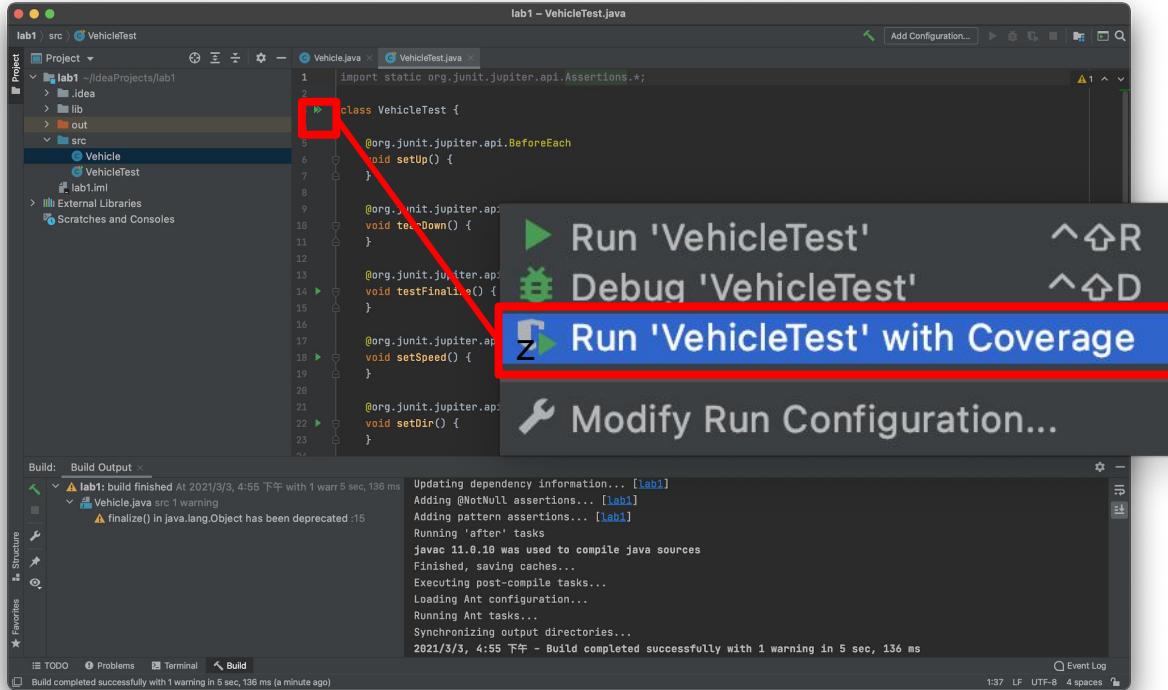


Code Coverage

Code Coverage

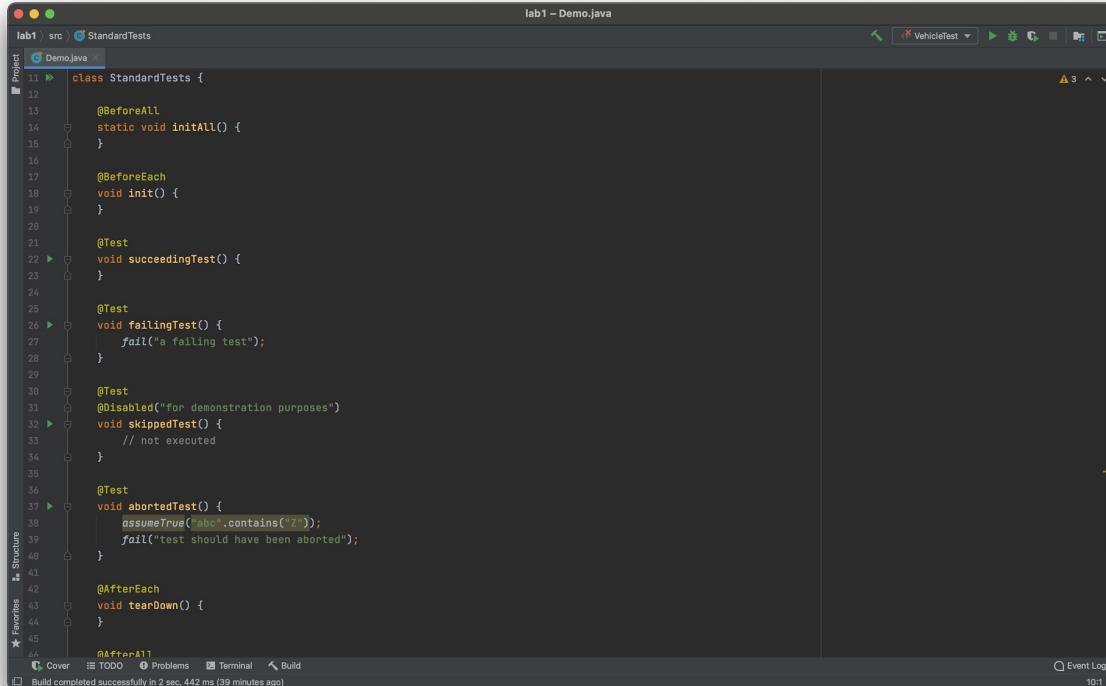
- In computer science, test coverage is a **measure used to describe the degree to which the source code of a program is executed when a particular test suite runs.**
- A program with high test coverage, measured as a percentage, has had more of its source code executed during testing, which suggests it has a lower chance of containing undetected software bugs compared to a program with low test coverage.

Get Code Coverage



JUnit 5

Test Classes and Methods



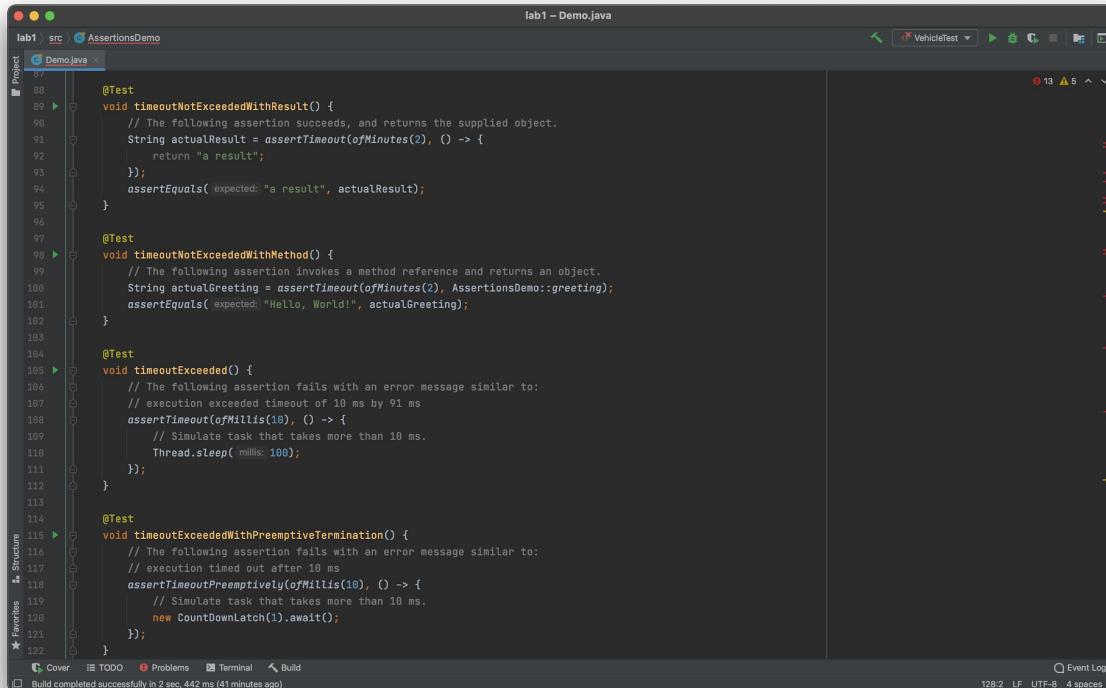
The screenshot shows a Java code editor with the file `Demo.java` open. The code defines a class `StandardTests` containing various test methods and annotations:

```
11  class StandardTests {
12
13      @BeforeAll
14      static void initAll() {
15      }
16
17      @BeforeEach
18      void init() {
19      }
20
21      @Test
22      void succeedingTest() {
23      }
24
25      @Test
26      void failingTest() {
27          fail("a failing test");
28      }
29
30      @Test
31      @Disabled("for demonstration purposes")
32      void skippedTest() {
33          // not executed
34      }
35
36      @Test
37      void abortedTest() {
38          assumeTrue("abc".contains("z"));
39          fail("test should have been aborted");
40      }
41
42      @AfterEach
43      void tearDown() {
44      }
45
46      @AfterAll
```

The IDE interface includes a project tree on the left, a code editor with syntax highlighting, and various toolbars and status bars at the bottom.

→ <https://junit.org/junit5/docs/current/user-guide/#writing-tests-classes-and-methods>

Assertions



The screenshot shows a Java IDE interface with a dark theme. The left sidebar displays a project structure for 'lab1' with a single file 'Demo.java' selected. The main editor area contains the following code:

```
lab1 – Demo.java
lab1 src AssertionsDemo
Project Demo.java
87
88     @Test
89     void timeoutNotExceededWithResult() {
90         // The following assertion succeeds, and returns the supplied object.
91         String actualResult = assertTimeout(ofMinutes(2), () -> {
92             return "a result";
93         });
94         assertEquals(expected: "a result", actualResult);
95     }
96
97     @Test
98     void timeoutNotExceededWithMethod() {
99         // The following assertion invokes a method reference and returns an object.
100        String actualGreeting = assertTimeout(ofMinutes(2), AssertionsDemo::greeting);
101        assertEquals(expected: "Hello, World!", actualGreeting);
102    }
103
104    @Test
105    void timeoutExceeded() {
106        // The following assertion fails with an error message similar to:
107        // execution exceeded timeout of 10 ms by 91 ms
108        assertTimeout(ofMillis(10), () -> {
109            // Simulate task that takes more than 10 ms.
110            Thread.sleep(millis: 100);
111        });
112    }
113
114    @Test
115    void timeoutExceededWithPreemptiveTermination() {
116        // The following assertion fails with an error message similar to:
117        // execution timed out after 10 ms
118        assertTimeoutPreemptively(ofMillis(10), () -> {
119            // Simulate task that takes more than 10 ms.
120            new CountDownLatch(1).await();
121        });
122    }

```

The code uses JUnit 5's `assertTimeout` and `assertTimeoutPreemptively` methods to test various timeout behaviors.

→ <https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions>



Lab

Lab

1. Download **Vehicle.java** from Github.
 - a. <https://github.com/iasthc/NYCU-Software-Testing-2021>
2. Cover all code with test cases.
3. Deliverables shall include the following:
 - a. Test Code: *VehicleTest.java*
 - b. Screenshot of test coverage: *STUDENT_ID.png*

Hint: Assertion

Do not compress the files and plagiarism.

Lab

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** lab1
- File:** VehicleTest.java
- Coverage:** VehicleTest
- Code Coverage Data:**

Element	Class, %	Method, %	Line, %
apple			
com			
images			
java			
jax			
idk			
META-INF			
netscape			
org			
sun			
Vehicle	100% (1/1)	100% (9/9)	100% (19/19)
VehicleTest	100% (1/1)	100% (7/7)	100% (21/21)
- Test Results:** Tests passed: 5 of 5 tests - 21ms
- Bottom Status Bar:** Tests passed: 5 (moments ago) | Event Log | 7:20