Guidebook: Writing Java Code from Test Code

OBJECT ORIENTED PROGRAMMING COURSE

Table of Contents

1. Introduction

This guidebook will help you learn how to write source code based on test code. You will practice analyzing test cases, writing the required implementation (source code), and verifying your solution using Visual Studio Code.

2. Simple Example

This section gives you a complete walk-through of how to read and understand a JUnit test, and how to write the matching source code.

We use a simple calculator class to demonstrate this process clearly.

When we build a program (like a sorting algorithm, searching algorithms, etc), we should make sure:

- It works correctly
- It doesn't break when something changes
- It handles errors properly

2.1 Problem Sample 1

That's where **test codes** come in—they help you automatically check whether our **main program behaves as expected**.

Test code for Calculator class
CalculatorTest.java

```
 1  package oop;
 2
 3  import static org.junit.Assert.*;
 4  import org.junit.Test;
 5
 6  public class CalculatorTest {
 7
 8      @Test
 9      public void testAdd() {
10          Calculator calc = new Calculator();
11          int codeInput1 = 2;
12          int codeInput2 = 3;
13          int codeOutput = calc.add(codeInput1, codeInput2);
14          int expectedOutput = 5;
15
16          try {
17              assertEquals("Addition test failed:", expectedOutput, codeOutput);
18          } catch (AssertionError ae) {
19              System.out.println(ae);
20          }
21      }
22  }
```

From the test case we create a class with the exact class and method names so that the test passes successfully.

The following is an explanation of each part of the test case:

1. `package oop;`

   The statement in line 1 places the file in a package named `oop`. Packages are used to organize Java classes into namespaces or modules.
   Test cases and source code must be in the same package as illustrated below.

```
project-folder/
└─── src/
   └─── oop/
      ├─── Calculator.java
      ├─── CalculatorTest.java
```

2. `import static org.junit.Assert.*;`

   The import in line 3 imports all static methods from the Assert class in the JUnit framework — including:
   - assertEquals()
   - assertTrue(),
   - assertFalse()

3. `import org.junit.Test;`

   The import in line 4 needed when we use @Test annotation.

4. `public class CalculatorTest {`

   Statement in line 6 defines a test class named CalculatorTest.
   It will contain unit tests to verify the functionality of the Calculator class.

5. `@Test`

   Annotation in line 8 tells JUnit that the method below is a test method, which should be executed when tests run.

6. `public void testAdd() {`

   Statement in line 9 declares the test method called `testAdd`.

7. `Calculator calc = new Calculator();`

Statement in line 10 creates an object from the `Calculator` class so that you can call its methods.

8. `int codeInput1 = 2;`
9. `int codeInput2 = 3;`

Statements in line 11 and 12 are the input values for the `add` method. Using separate variables for inputs is helpful for clarity and debugging.

10. `int codeOutput = calc.add(codeInput1, codeInput2);`

Statement in line 13 calls the `add` method with the given inputs and stores the result in `codeOutput`.

11. `int expectedOutput = 5;`

Statement in line 14 is the expected result of the `add(2, 3)` operation.

12. `try {`

Statement in line 16 starts a try-catch block — useful for **gracefully handling assertion failures** and printing messages.

13. `assertEquals("Addition test failed:", expectedOutput, codeOutput);`

Statement in line 17, compares `expectedOutput` with `codeOutput`. If they are not equal, the assertion will fail and throw an `AssertionError` with the message `"Addition test failed:"`.

14. `} catch (AssertionError ae) {`

Statement in line 18, catches the `AssertionError` if the test fails. This prevents the program from crashing and lets you log the error.

2.2 Answer Sample 1

Based on the test code CalculatorTest.java, we can create the source code with the following steps:

1. `public class Calculator {`

In the test case, an object is created using new Calculator(). Therefore, we need to define a class named Calculator to match the test case.

2. `public int add(int a, int b) {`

   The test calls a method named `add`, which takes **two integer parameters**. So, the class must include a method named `add` with this signature: `int add(int, int)`.

3. `return a + b;`

   The test case expects the result of add(2, 3) to be 5. That means the method should return the sum of the two inputs a and b.

4. Provide comments in "**BAHASA INDONESIA**" in the method section, such as the code section **highlighted** in yellow below.

Source code which is made based on test code
Calculator.java

```
1  package oop;
2
3  public class Calculator {
4
5      // Method untuk menjumlahkan dua bilangan bulat
6      public int add(int a, int b) {
7          // Mengembalikan hasil penjumlahan a dan b
8          return a + b;
9      }
10 }
```

2.3 Problem Sample 2

That's where **test codes** come in—they help you automatically check whether our **main program behaves as expected**.

Test code for ArrayMinMax class
ArrayMinMaxTest.java

```
1  package oop;
2
3  import static org.junit.Assert.*;
4  import org.junit.Test;
5
6  public class ArrayMinMaxTest {
7
8      @Test
9      public void testFindMinMax() {
10         int[] codeInput = { 80, 2, -3, 14, 95 };
11         int[] expectedOutput = { -3, 95 };
12
13         ArrayMinMax arraySum = new ArrayMinMax();
14         int[] codeOutput = arraySum.findMinMax(codeInput);
15
16         try {
17             assertArrayEquals("Test Find MinMax:", expectedOutput, codeOutput);
18         } catch (AssertionError ae) {
19             System.out.println(ae);
20         }
21     }
```

```
22
23      @Test
24      public void testFindMinMaxEmpty() {
25          int[] codeInput = {};
26          // Empty Array, default value
27          int[] expectedOutput = { Integer.MAX_VALUE, Integer.MIN_VALUE };
28
29          ArrayMinMax arraySum = new ArrayMinMax();
30          int[] codeOutput = arraySum.findMinMax(codeInput);
31
32          // Memverifikasi output yang dihasilkan
33          try {
34              assertArrayEquals("Test Find MinMax Empty:", expectedOutput,
    codeOutput);
35          } catch (AssertionError ae) {
36              System.out.println(ae);
37          }
38      }
39  }
```

From the test case we create a class with the exact class and method names so that the test passes successfully.

2.4 Answer Sample 2

Based on the test code ArrayMinMaxTest.java, we can create the source code with the following steps:

1. `public class ArrayMinMax {`

   In the test case, an object is created using `new ArrayMinMax()`. Therefore, we need to define a class named `Calculator` to match that.

2. `public int[] findMinMax(int[] array) {`

   The test calls a method named `findMinMax`, which accepts one parameter of type `int[]` (an integer array) and returns an `int[]` array.
   Thus, we must create a method with the signature `int[] findMinMax(int[] array)` inside the class.

3. `return new int[] { min, max };`

   After completing the search, return an array containing the found minimum value at index 0 and maximum value at index 1.

Source code which is made based on test code
Calculator.java

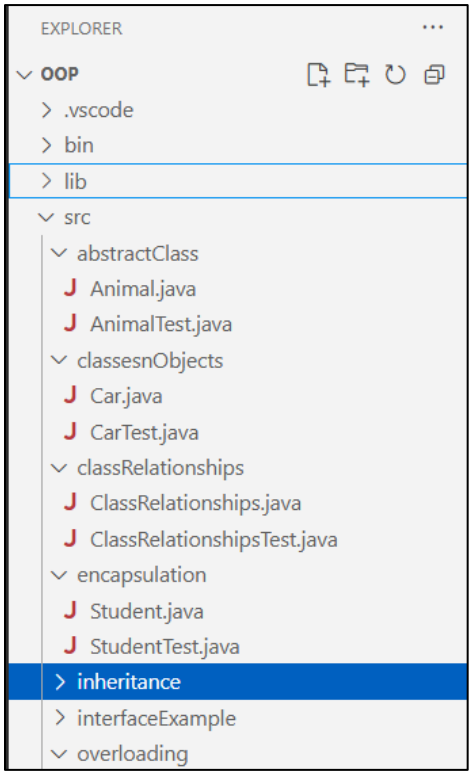
```
1   package oop;
2
3   public class ArrayMinMax {
4
5       public int[] findMinMax(int[] array) {
6           if (array.length == 0) {
```

```
7               // Jika array kosong, kembalikan nilai default (misalnya, min =
                // Integer.MAX_VALUE dan max = Integer.MIN_VALUE)
8               return new int[] { Integer.MAX_VALUE, Integer.MIN_VALUE };
9           }
10
11          // Menganggap elemen pertama sebagai nilai min dan max
12          int min = array[0];
13          int max = array[0];
14
15          // Mengiterasi array untuk mencari nilai minimum dan maksimum
16          for (int num : array) {
17              if (num < min)
18                  min = num; // Menemukan nilai terkecil
19              if (num > max)
20                  max = num; // Menemukan nilai terbesar
21          }
22
23          // Mengembalikan nilai min dan max dalam array
24          return new int[] { min, max };
25      }
26  }
```

3. Problems

There are 9 problems to be solved with details explained in this section. All test cases are also available at the following link https://github.com/mustmentari/CWP-OOP.

Please arrange "oop" project like the image snippet below. **The package name represents the folder name shown in the image**, such as `abstractClass`, `encapsulation`, `classesAndObjects`, etc.



3.1 Class and Object

Please create source code based on the test code below.

Test code for Class and Object Topic

CarTest.java

```
1   package classesnObjects;
2   import static org.junit.Assert.assertEquals;
3   import org.junit.Test;
4   public class CarTest {
5       @Test
6       public void testCar1() {
7           // Code input
8           String codeInputBrand = "Toyota";
9           String codeInputModel = "Corolla";
10          int codeInputYear = 2020;
11          // Execution / code output
12          Car car = new Car(codeInputBrand, codeInputModel, codeInputYear);
13          String codeOutput = car.getCarInfo();
14          // Expected output
15          String expectedOutput = "Toyota Corolla (2020)";
16          // Assertion
17          try {
18              assertEquals("Car creation test:", expectedOutput, codeOutput);
19          } catch (AssertionError ae) {
20              System.out.println(ae);
21          }
21      }
22      @Test
23      public void testCar2() {
24          // Code input
25          String codeInputBrand = "Honda";
26          String codeInputModel = "Civic";
27          int codeInputYear = 2021;
28          // Execution / code output
29          Car car = new Car(codeInputBrand, codeInputModel, codeInputYear);
30          String codeOutput = car.getCarInfo();
31          // Expected output
32          String expectedOutput = "Honda Civic (2021)";
33          // Assertion
34          try {
35              assertEquals("Another car creation test:", expectedOutput, codeOutput);
36          } catch (AssertionError ae) {
37              System.out.println(ae);
38          }
39      }
40  }
```

3.2 Encapsulation

Please create source code based on the test code below.

Test code for Encapsulation Topic

StudentTest.java

```
1   package Encapsulation;
2   import static org.junit.Assert.assertEquals;
3   import org.junit.Test;
4   public class StudentTest {
5       @Test
6       public void testStudentObjectCreation() {
7           // Code input
8           String codeInputName = "Alice";
9           int codeInputAge = 20;
10          // Execution / code output
11          Student codeOutputStudent = new Student(codeInputName, codeInputAge);
12          String codeOutput = codeOutputStudent.getName() + " " + codeOutputStudent.getAge();
```

```
        // Expected output
        String expectedOutput = "Alice 20";
        // Assertion
        try {
            assertEquals("Student object creation test:", expectedOutput, codeOutput);
        } catch (AssertionError ae) {
            System.out.println(ae);
        }
    }
    @Test
    public void testGetterAge() {
        // Code input
        String codeInputName = "Bob";
        int codeInputAge = 22;
        Student codeInputStudent = new Student(codeInputName, codeInputAge);
        // Execution / code output
        int codeOutput = codeInputStudent.getAge();
        // Expected output
        int expectedOutput = 22;
        // Assertion
        try {
            assertEquals("Getter age test:", expectedOutput, codeOutput);
        } catch (AssertionError ae) {
            System.out.println(ae);
        }
    }
    @Test
    public void testSetterAge() {
        // Code input
        String codeInputName = "Charlie";
        int codeInputAge = 25;
        int codeInputNewAge = 30;
        Student codeInputStudent = new Student(codeInputName, codeInputAge);
        // Execution / code output
        codeInputStudent.setAge(codeInputNewAge);
        int codeOutput = codeInputStudent.getAge();
        // Expected output
        int expectedOutput = 30;
        // Assertion
        try {
            assertEquals("Setter age test:", expectedOutput, codeOutput);
        } catch (AssertionError ae) {
            System.out.println(ae);
        }
    }
}
```

3.3 Inheritance

Please create source code based on the test code below.

Test code for Inheritance Topic

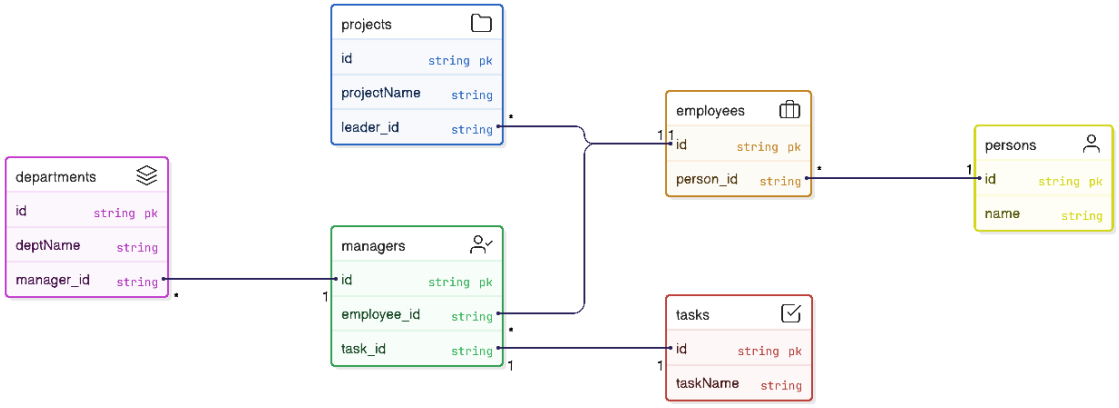
PersonTest.java

```
package Inheritance;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class PersonTest {
    @Test
    public void testSuperclass() {
        // Focus on creating superclass objects and their methods
        String codeInputName = "Alice";
        int codeInputAge = 20;
        Person codeOutputPerson = new Person(codeInputName, codeInputAge);
        String codeOutput = codeOutputPerson.getInfo();
        String expOutput = "Alice (20 years old)";
        try {
            assertEquals("Superclass info test:", expOutput, codeOutput);
        } catch (AssertionError ae) {
            System.out.println(ae);
```

```
17          }
18      }
19      @Test
20      public void testSubclass() {
21          // Focus on creating subclass objects and accessing superclass methods+additional ones
21          String codeInputName = "Bob";
22          int codeInputAge = 21;
23          String codeInputId = "S123";
24          Student codeOutputStudent = new Student(codeInputName, codeInputAge, codeInputId);
25          String codeOutputInfo = codeOutputStudent.getInfo(); // superclass methods
26          String codeOutputId = codeOutputStudent.getStudentId(); // additional method
27          String expOutputInfo = "Bob (21 years old)";
28          String expOutputId = "S123";
29          try {
30              assertEquals("Subclass info test (superclass method):", expOutputInfo,
31  codeOutputInfo);
32              assertEquals("Subclass info test (subclass method):", expOutputId, codeOutputId);
33          } catch (AssertionError ae) {
34              System.out.println(ae);
35          }
36      }
37  }
```

3.4 Class Relationships

Please create the Java source code for this part according to the following diagram.



Please create source code based on the test code below.

Test code for Class Relationships Topic

ClassRelationshipsTest .java

```
1   package classRelationships;
2   import static org.junit.Assert.assertEquals;
3   import org.junit.Test;
4   public class ClassRelationshipsTest {
5       @Test
6       public void testInheritance() {
7           // Code input
8           String codeInputName = "Alice";
9           // Execution / codeOutput
10          Employee employee = new Employee(codeInputName);
11          String codeOutputName = employee.getName(); // method inherited
12          boolean codeOutputIsPerson = employee instanceof Person; // inheritance check
13          // Expected output
14          String expectedOutputName = "Alice";
15          boolean expectedOutputIsPerson = true;
16          // Assertion
```

```
17          try {
18              assertEquals("Method inherited test:", expectedOutputName, codeOutputName);
19              assertEquals("Inheritance check:", expectedOutputIsPerson, codeOutputIsPerson);
20          } catch (AssertionError ae) {
21              System.out.println(ae);
21          }
22      }
23      @Test
24      public void testAssociation() {
25          // codeInput
26          Employee leader = new Employee("Bob");
27          Project project = new Project("Apollo", leader);
28          // Execution / codeOutput
29          String codeOutput = project.getLeaderName();
30          // Expected output
31          String expectedOutput = "Bob";
32          // Assertion
33          try {
34              assertEquals("Association test:", expectedOutput, codeOutput);
35          } catch (AssertionError ae) {
36              System.out.println(ae);
37          }
38      }
39      @Test
40      public void testAggregation() {
41          // codeInput
42          Manager mgr = new Manager("Charlie");
43          Department department = new Department("IT");
44          department.setManager(mgr);
45          // Execution / codeOutput
46          String codeOutput = department.getManagerName();
47          // Expected output
48          String expectedOutput = "Charlie";
49          // Assertion
50          try {
51              assertEquals("Aggregation test:", expectedOutput, codeOutput);
52          } catch (AssertionError ae) {
53              System.out.println(ae);
54          }
55      }
56      @Test
57      public void testComposition() {
58          // codeInput
59          Manager mgr = new Manager("Diana");
60          Task task = new Task("Prepare report");
61          mgr.setTask(task);
62          // Execution / codeOutput
63          String codeOutput = mgr.getTaskName();
64          // Expected output
65          String expectedOutput = "Prepare report";
66          // Assertion
67          try {
68              assertEquals("Composition test:", expectedOutput, codeOutput);
69          } catch (AssertionError ae) {
70              System.out.println(ae);
71          }
72      }
73  }
```

3.5 Overloading

Please create source code based on the test code below.

Test code for Overloading Topic

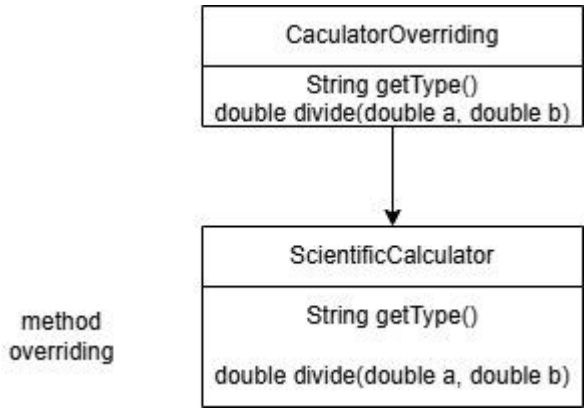
CalculatorTest.java

```
1   package overloading;
2   import static org.junit.Assert.assertEquals;
3   import org.junit.Test;
4   public class CalculatorTest {
5       @Test
```

```
 6      public void testAddIntInt() {
 7          // --- Test: add(int, int) ---
 8          // Code input
 9          int codeInputA = 2;
10          int codeInputB = 3;
11          // Execution / code output
12          Calculator calc = new Calculator();
13          int codeOutput = calc.add(codeInputA, codeInputB);
14          // Expected output
15          int expectedOutput = 5;
16          // Assertion
17          try {
18              assertEquals("add(int, int) test:", expectedOutput, codeOutput);
19          } catch (AssertionError ae) {
20              System.out.println(ae);
21          }
21      }
22      @Test
23      public void testAddDoubleDouble() {
24          // --- Test: add(double, double) ---
25          // Code input
26          double codeInputA = 2.3;
27          double codeInputB = 3.4;
28          // Execution / code output
29          Calculator calc = new Calculator();
30          double codeOutput = calc.add(codeInputA, codeInputB);
31          // Expected output
32          double expectedOutput = 5.7;
33          // Assertion
34          try {
35              assertEquals("add(double, double) test:", expectedOutput, codeOutput, 0.0001);
36          } catch (AssertionError ae) {
37              System.out.println(ae);
38          }
39      }
40      @Test
41      public void testAddIntIntInt() {
42          // --- Test: add(int, int, int) ---
43          // Code input
44          int codeInputA = 3;
45          int codeInputB = 4;
46          int codeInputC = 5;
47          // Execution / code output
48          Calculator calc = new Calculator();
49          int codeOutput = calc.add(codeInputA, codeInputB, codeInputC);
50          // Expected output
51          int expectedOutput = 12;
52          // Assertion
53          try {
54              assertEquals("add(int, int, int) test:", expectedOutput, codeOutput);
55          } catch (AssertionError ae) {
56              System.out.println(ae);
57          }
58      }
59  }
```

3.6 Overriding

Please create the Java source code for this part according to the following diagram.



Please create source code based on the test code below.

Test code for Overriding Topic
CalculatorOverridingTest.java

```
1   package overriding;
2   import static org.junit.Assert.assertEquals;
3   import org.junit.Test;
4   public class CalculatorOverridingTest {
5       @Test
6       public void testOverriding() {
7           // --- Code input ---
8           CalculatorOverriding calculatorOverriding = new CalculatorOverriding();
9           CalculatorOverriding calculatorOverridingScientific = new
    CalculatorOverriding.ScientificCalculator();
10          double codeInputA = 10;
11          double codeInputB = 0;
12          // --- Execution / code output ---
13          String codeOutputTypeCalc = calculatorOverriding.getType();
14          String codeOutputTypeSci = calculatorOverridingScientific.getType();
15          double codeOutputDivideCalc = calculatorOverriding.divide(codeInputA, codeInputB);
16          double codeOutputDivideSci = calculatorOverridingScientific.divide(codeInputA,
    codeInputB);
17          // --- Expected output ---
18          String expectedOutputTypeCalc = "Generic Calculator";
19          String expectedOutputTypeSci = "Scientific Calculator";
20          double expectedOutputDivideCalc = Double.NaN;
21          double expectedOutputDivideSci = Double.POSITIVE_INFINITY;
21          // --- Assertion ---
22          try {
23              // Test overridden getType()
24              assertEquals("getType() for Calculator:", expectedOutputTypeCalc,
    codeOutputTypeCalc);
25              assertEquals("getType() for ScientificCalculator:", expectedOutputTypeSci,
    codeOutputTypeSci);
26              // Test overridden divide()
27              assertEquals("divide(10,0) for Calculator:", expectedOutputDivideCalc,
    codeOutputDivideCalc, 0.0001);
28              assertEquals("divide(10,0) for ScientificCalculator:", expectedOutputDivideSci,
    codeOutputDivideSci, 0.0001);
            } catch (AssertionError ae) {
29              System.out.println(ae);
30          }
31      }
32  }
```

3.7 Abstract Class

Please create source code based on the test code below.

Test code for Abstract Class Topic
AnimalTest.java

```
1  package abstractClass;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.Test;
4  public class AnimalTest {
5      @Test
6      public void testDogMakeSound() {
7          // Code input
8          String codeInput = "Buddy";
9          // Execution / code output
10         Animal animal = new Dog(); // reference superclass
11         String codeOutput = animal.makeSound(codeInput);
12         // Expected output
13         String expectedOutput = "Buddy says Woof!";
14         // Assertion
15         try {
16             assertEquals("Dog makeSound test:", expectedOutput, codeOutput);
17         } catch (AssertionError ae) {
18             System.out.println(ae);
19         }
20     }
21     @Test
21     public void testCatMakeSound() {
22         // Code input
23         String codeInput = "Kitty";
24         // Execution / code output
25         Animal animal = new Cat(); // reference superclass
26         String codeOutput = animal.makeSound(codeInput);
27         // Expected output
28         String expectedOutput = "Kitty says Meow!";
29         // Assertion
30         try {
31             assertEquals("Cat makeSound test:", expectedOutput, codeOutput);
32         } catch (AssertionError ae) {
33             System.out.println(ae);
34         }
35     }
36 }
```

3.8 Interface

Please create source code based on the test code below.

Test code for Interface Topic

PayableTest.java

```
1  package interfaceExample;
2  import static org.junit.Assert.assertEquals;
3  import org.junit.Test;
4  public class PayableTest {
5      @Test
6      public void testGetPaymentAmountCase1() {
7          // Code input
8          double codeInputSalary = 4500.0;
9          Employee employee = new Employee(codeInputSalary);
10         // Execution / code output
11         double codeOutput = employee.getPaymentAmount();
12         // Expected output
13         double expectedOutput = 4500.0;
14         // Assertion
15         try {
16             assertEquals("getPaymentAmount() test case 1:", expectedOutput, codeOutput,
   0.0001);
17         } catch (AssertionError ae) {
18             System.out.println(ae);
19         }
20     }
21     @Test
21     public void testGetPaymentAmountCase2() {
22         // Code input
23         double codeInputSalary = 3000.0;
```

```
24          Employee employee = new Employee(codeInputSalary);
25          // Execution / code output
26          double codeOutput = employee.getPaymentAmount();
27          // Expected output
28          double expectedOutput = 3000.0;
29          // Assertion
30          try {
31              assertEquals("getPaymentAmount() test case 2:", expectedOutput, codeOutput,
    0.0001);
32          } catch (AssertionError ae) {
33              System.out.println(ae);
34          }
35      }
36  }
```

3.9 Polymorphism

Please create source code based on the test code below.

Test code for Polymorphism Topic

ShapeTest.java

```
1   package polymorphism;
2   import static org.junit.Assert.assertEquals;
3   import org.junit.Test;
4   public class ShapeTest {
5       @Test
6       public void testCircleArea() {
7           // Code input
8           double codeInputRadius = 1.0;
9           // Execution / code output
10          Shape shape = new Circle(codeInputRadius); // create Circle object
11          double codeOutput = shape.getArea(); // call getArea() method
12          // Expected output
13          double expectedOutput = Math.PI * codeInputRadius * codeInputRadius;
14          // Assertion
15          try {
16              assertEquals("Circle area test:", expectedOutput, codeOutput, 0.0001);
17          } catch (AssertionError ae) {
18              System.out.println(ae);
19          }
20      }
21      @Test
21      public void testRectangleArea() {
22          // Code input
23          double codeInputWidth = 2.0;
24          double codeInputHeight = 3.0;
25          // Execution / code output
26          Shape shape = new Rectangle(codeInputWidth, codeInputHeight); //create Rectangle object
27          double codeOutput = shape.getArea(); // call getArea() method
28          // Expected output
29          double expectedOutput = codeInputWidth * codeInputHeight;
30          // Assertion
31          try {
32              assertEquals("Rectangle area test:", expectedOutput, codeOutput, 0.0001);
33          } catch (AssertionError ae) {
34              System.out.println(ae);
35          }
36      }
37  }
```

4. How to Run Java Tests in Visual Studio Code

1. Install Visual Studio Code from https://code.visualstudio.com

2. Install the Java Extension Pack from the Extensions Marketplace

3. Create a new folder for your Java project and open it in VS Code

4. Add your `.java` files inside the folder `oop/packageName` (e.g., Animal.java and AnimalTest.java)

5. Use the integrated terminal:

javac -cp .:junit-4.13.2.jar:hamcrest-core-1.3.jar *.java

java -cp .:junit-4.13.2.jar:hamcrest-core-1.3.jar org.junit.runner.JUnitCore CalculatorTest

or

put .jar files inside the folder "lib" like the pictures below. You can directly download .jar files from this link https://github.com/mustmentari/CWP-OOP.

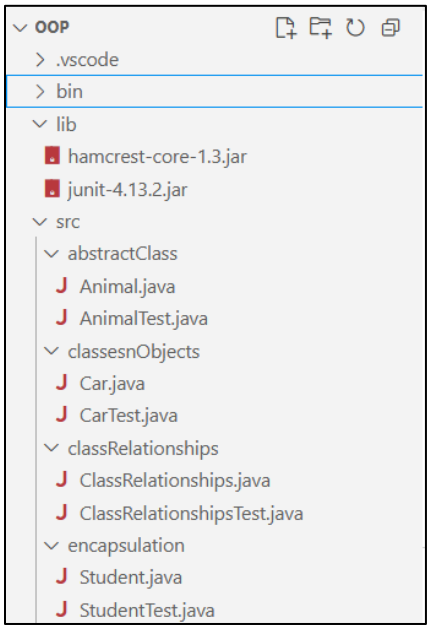
6. Create a java testing file in the package you have created. Copy and paste the test code provided in this guidebook.

7. Create another new java file to store your source. Then, create source code according to the test code in each question.

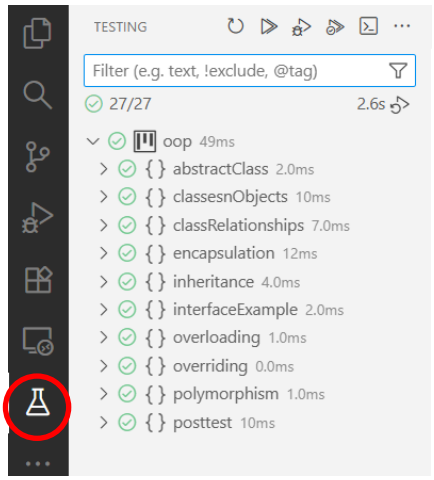
8. Analyze the output to debug and refine your implementation

📷 Example Screenshot:

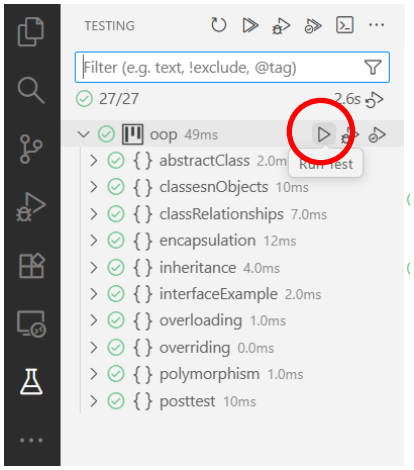
The image below shows the file structure within the project.



Then, click the testing icon according to the part circled in red in the image below. If this icon does not appear, make sure you have run the project.



Then, click run test according to the section circled in red in the image below to ensure that your source code matches the existing test code. If there is a green checklist mark shown according to the image, then your source code is correct, if a red cross appears, it means that the source code you created is not in accordance with the existing test code.



Even if the green check mark appears, please double-check in the **Debug Console** menu in VS Code. If something like the example below shows up, it means there is still part of your source code that is incorrect and needs to be fixed.

code that is incorrect and needs to be fixed.

5. Assignment Submission Guidelines

Submission Deadline:

Deadline for problems 1-9

- **Date:** October 19, 2025
- **Time:** 11:59 PM (WIB)

Submission Steps:

1. **Prepare Your Assignment File**
   Ensure your assignment file is in the correct format and is complete. The file name should follow this format:
   `OOP_FullName_Date`

   Example: `OOP_MustikaMentari_19Oct2025`

2. **Create a Google Drive Folder**
   Create a new folder in Google Drive with the following name:
   `Assignment [Course/Topic] - [Full Name]`

   Example: `OOP - Mustika Mentari`

3. **Upload Your File**
   After creating the folder (Your project folder), upload your assignment file into that folder.
   Make sure the uploaded file is the correct one and follow the instructions given.
4. **Share the Folder**
   Once the file is uploaded, make sure that the folder/file can be accessed by the instructor or teaching assistants. To do so:
   - Right-click on the folder or file, select **"Get link"**.
   - Set the access to **"Anyone with the link"** and make sure the permission is set to **"Viewer"**.
5. **Send the Google Drive Link**
   Send the link to your Google Drive folder to google formulir on this link https://forms.gle/zcawimNsC5ebhDzv8.