Mohammed V University in Rabat

National Higher School of Computer Science and Systems Analysis

Lab Report No. 2

Unit Testing a Class

Prepared by :

BAKKALI Douae

Instructor :

Pr. Hamlaoui

Academic Year 2024-2025

1 Introduction

Unit testing is a fundamental practice in software development where individual components of a program are verified in isolation. This lab focuses on unit testing a `Point` class in Java, demonstrating key testing concepts through practical implementation.

Unit tests provide several critical benefits :

— Early bug detection during development

— Documentation of expected behavior

— Safety net for code refactoring

— Improved code design through testability

In this lab, we specifically examine :

— Test case design for a simple class

— Proper implementation of equality comparison

— Test organization using setup/teardown methods

— JUnit framework features

The following sections provides the complete correction for the Software Testing lab on unit testing a Point class. Each question is addressed in order with properly formatted Java code examples.

2 Project Configuration

JUnit Dependencies

The project requires proper JUnit dependencies in the `pom.xml` file :

```
<dependencies>
    <!-- JUnit 5 -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>5.9.2</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

3 Lab Corrections

3.1 Question 1 : Add testTranslater0_0()

Let add the following test methods in `PointTest` class in `src_test_java` :

```java
@Test
public final void testTranslater0_0() {
    Point a = new Point(1, 2);
    Point expected = new Point(1, 2);
    Point obtained = a.translater(0, 0);
    assertEquals(expected, obtained);
}
```

3.2 Question 2 : Why the test fails

The test fails because the Point class doesn't override the `equals()` method. By default, Java's `Object.equals()` compares object references rather than their content.

3.3 Question 3 : Add equals() method

After the first test failed, we need to override the equals() method in the Point class and test it again.

```java
@Override
public boolean equals(Object obj) {
    if (this == obj) return true;
    if (obj == null || getClass() != obj.getClass()) return false;
    Point point = (Point) obj;
    return Double.compare(point.x, x) == 0 &&
           Double.compare(point.y, y) == 0;
}
```

3.4 Question 4 : Test for equals() method

```java
@Test
public final void testEquals() {
    Point p1 = new Point(1, 2);
    Point p2 = new Point(1, 2);
    Point p3 = new Point(3, 4);

    assertTrue(p1.equals(p2));
    assertTrue(p2.equals(p1));
    assertFalse(p1.equals(p3));
    assertFalse(p1.equals(null));
    assertFalse(p1.equals(new Object()));
}
```

The tests for the equals() method pass after the implementation of the equals() method in the Point class.

3.5 Question 5 : Add testTranslater1_3()

```java
@Test
public final void testTranslater1_3() {
    Point a = new Point(1, 2);
    Point expected = new Point(2, 5);
    Point obtained = a.translater(1, 3);
    assertEquals(expected, obtained);
}
```

3.6 Question 6 : Why test fails and fix

The test should pass if `equals()` is implemented correctly. The `translater()` method already returns a new Point with correct coordinates.

3.7 Question 7 : Verify all tests pass

After implementing `equals()`, all tests should pass when executed.

3.8 Question 8 : Common initialization

```java
private Point testPoint;

@BeforeEach
public void setUp() {
    testPoint = new Point(1, 2);
}

@Test
public final void testTranslater0_0() {
    Point expected = new Point(1, 2);
    Point obtained = testPoint.translater(0, 0);
    assertEquals(expected, obtained);
}

@Test
public final void testTranslater1_3() {
    Point expected = new Point(2, 5);
    Point obtained = testPoint.translater(1, 3);
    assertEquals(expected, obtained);
}
```

3.9 Question 9 : Common cleanup

```java
@AfterEach
public void tearDown() {
    // Cleanup code
    testPoint = null;
}
```

3.10 Question 10 : @BeforeAll version

```java
private static Point testPoint;

@BeforeAll
public static void setUp() {
    testPoint = new Point(1, 2);
}

@Test
public final void testTranslater0_0() {
    Point expected = new Point(1, 2);
    Point obtained = testPoint.translater(0, 0);
    assertEquals(expected, obtained);
}

@Test
public final void testTranslater1_3() {
    Point expected = new Point(2, 5);
    Point obtained = testPoint.translater(1, 3);
    assertEquals(expected, obtained);
}
```

Here When using `@BeforeAll`, avoid combining it with `@AfterEach` that modifies the shared state. Instead, we can use `@AfterAll`.

4 Conclusion

This lab demonstrated essential unit testing practices through a `Point` class implementation, covering :

- Proper `equals()` implementation for value comparison
- Test initialization strategies (`@BeforeEach` vs `@BeforeAll`)
- Resource cleanup with `@AfterEach` and `@AfterAll`
- JUnit 5 test organization best practices

These techniques form the foundation for writing maintainable, isolated unit tests that reliably verify class behavior.