



**EAST WEST UNIVERSITY**

**Department of Computer Science and Engineering**

**CSE430 - Software Testing and Quality Assurance**

---

## **Lab Topic: JUnit Unit Testing**

### **1. Objective**

This lab introduces unit testing in Java using the JUnit framework within NetBeans IDE. Students will learn how to create Java classes, write unit tests for class methods, and interpret the results of automated test executions.

### **2. Learning Outcomes**

After completing this lab, students will be able to:

1. Set up and configure a JUnit 5 test environment in NetBeans.
2. Implement a Java class (Calculator1) with several methods.
3. Write and execute corresponding test cases in JUnit.
4. Interpret test outcomes and maintain structured test documentation.
5. Understand good testing practices (naming, structure, exception handling).

### 3. Software Requirements

- NetBeans IDE (latest Java SE version).
- Java JDK 17 or higher.
- JUnit 5 Library.

### 4. Lab Procedure

#### Step 1 – Creating the Project

1. Open NetBeans → *File* → *New Project*.
2. Choose Java with Maven → Java Application → *Next*.
3. Enter project name: CalculatorApp → *Finish*.

#### Step 2 – Creating Package and Class

1. Right-click Source Packages → New → Java Package → name it Calculator.
2. Right-click the new package → *New* → *Java Class* → name it Calculator1.

## 5. Implementation under Test

Source Code – Calculator1.java

```
package Calculator;

public class Calculator1 {

    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new IllegalArgumentException("Cannot divide by zero.");
        }
        return a / b;
    }
}
```

**Explanation:**

- Each method performs a basic arithmetic operation.
- The divide() method includes an exception check to demonstrate how JUnit handles error cases

## 6. Generating the JUnit Test Class

1. Right-click Calculator1.java → *Tools* → *Create Tests*.
2. Choose JUnit → *OK*.
3. NetBeans automatically creates Calculator1Test.java under Test Packages.

## 7. JUnit Test Implementation

Source Code – Project Files → pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.mycompany</groupId>
  <artifactId>CalculatorApp</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>jar</packaging>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.release>25</maven.compiler.release>
    <maven.compiler.plugin.version>3.11.0</maven.compiler.plugin.version>
    <maven.surefire.plugin.version>3.2.5</maven.surefire.plugin.version>
    <junit.jupiter.version>5.10.3</junit.jupiter.version>
    <exec.mainClass>com.mycompany.calculatorapp.CalculatorApp</exec.mainClass>
  </properties>

  <dependencies>
    <!-- JUnit 5 (Jupiter): includes API + Engine + Params -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter</artifactId>
      <version>${junit.jupiter.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

```

<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>${maven.compiler.plugin.version}</version>
      <configuration>
        <release>${maven.compiler.release}</release>
      </configuration>
    </plugin>

    <!-- Required for JUnit 5 tests -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>${maven.surefire.plugin.version}</version>
      <configuration>
        <useModulePath>>false</useModulePath>
      </configuration>
    </plugin>
  </plugins>
</build>
</project>

```

Source Code – Calculator1Test.java

```

package Calculator;

import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class Calculator1Test {

    private final Calculator1 calculator = new Calculator1();

    @Test
    void testAdd() {
        assertEquals(10, calculator.add(7, 3));
        assertEquals(0, calculator.add(-3, 3));
    }

    @Test
    void testSubtract() {

```

```

    assertEquals(4, calculator.subtract(10, 6));
    assertEquals(-6, calculator.subtract(3, 9));
}

@Test
void testMultiply() {
    assertEquals(15, calculator.multiply(3, 5));
    assertEquals(0, calculator.multiply(0, 100));
}

@Test
void testDivide() {
    assertEquals(2, calculator.divide(10, 5));
    assertEquals(3, calculator.divide(9, 3));
}



@Test
void testDivideByZero() {
    IllegalArgumentException exception = assertThrows(
        IllegalArgumentException.class,
        () -> calculator.divide(10, 0)
    );
    assertEquals("Cannot divide by zero.", exception.getMessage());
}
}

```

### Explanation:

- `@Test` annotates each independent test case.
- `assertEquals(expected, actual)` verifies correctness.
- `fail()` ensures exception behavior is explicitly tested.
- The output console lists pass/fail messages, and the Test Results window shows a green bar if all tests pass, or red if any fail.

## 8. Executing the Tests

- Run Single Test File: Right-click Calculator1Test.java → *Test File*.
- Run All Tests: Right-click Project → *Test*.
- Results: View in the *Test Results* panel:
  -  Green Bar → All tests passed.
  -  Red Bar → At least one test failed → inspect assertion error.

## 9. Test Case Documentation

Method	Inputs	Expected Output	Description
add	5, 3	8	Adding positive numbers
add	-2, 2	0	Mixed sign addition
subtract	9, 5	4	Basic subtraction
multiply	4, 0	0	Multiplication by zero
divide	6, 3	2	Simple division
divide	7, 0	Exception	Division by zero case

## 10. Best Practices

- Use clear method names (testAdd, testDivideByZero).
- Test one behavior per method.
- Apply the AAA pattern: *Arrange* → *Act* → *Assert*.
- Test edge and exception cases.
- Keep tests deterministic (no random inputs or I/O).

## 11. Lab Task

### Lab Task 1 — Temperature Converter

**Objective:** Test correctness of temperature conversion formulas between Celsius, Fahrenheit, and Kelvin.

#### Implementation (TemperatureConverter.java)

```
public class TemperatureConverter {  
    public double celsiusToFahrenheit(double c) { return (c * 9/5) + 32; }  
    public double fahrenheitToCelsius(double f) { return (f - 32) * 5/9; }  
    public double celsiusToKelvin(double c) { return c + 273.15; }  
}
```

#### Test ideas (TemperatureConverterTest.java)

- Verify known pairs:  $0\text{ }^{\circ}\text{C} \rightarrow 32\text{ }^{\circ}\text{F}$ ,  $100\text{ }^{\circ}\text{C} \rightarrow 212\text{ }^{\circ}\text{F}$ .
- Round-trip check:  $\text{fahrenheitToCelsius}(\text{celsiusToFahrenheit}(x)) \approx x$ .
- Assert approximate equality using `assertEquals(expected, actual, 0.01)`.



## Lab Task 2 — Bank Account Operations

**Objective:** Test class invariants and exceptional conditions (negative balance, overdraft).

### Implementation (BankAccount.java)

```
public class BankAccount {
    private double balance;
    public void deposit(double amt){ if(amt<0) throw new IllegalArgumentException();
    balance+=amt; }
    public void withdraw(double amt){ if(amt>balance) throw new
    IllegalStateException(); balance-=amt; }
    public double getBalance(){ return balance; }
}
```

### Test ideas

- Deposit positive → balance increases.
- Withdraw valid → balance decreases.
- Withdraw beyond balance → expect IllegalStateException.
- Deposit negative → expect IllegalArgumentException.

## Lab Task 3 — String Utility (Palindrome Checker)

**Objective:** Use JUnit to test logic and case sensitivity in string operations.

### Implementation (StringUtil.java)

```
public class StringUtil {
    public boolean isPalindrome(String s) {
        if(s==null) return false;
        String clean = s.replaceAll("[^A-Za-z]", "").toLowerCase();
        return new StringBuilder(clean).reverse().toString().equals(clean);
    }
}
```

### Test ideas

- “madam” → true
- “RaceCar” (case insensitive) → true
- “hello” → false
- null or empty string → false

## Lab Task 4 — Simple Timer Utility

**Objective:** Test logic that handles elapsed-time computation and boundary conditions.

### Implementation (TimerUtil.java)

```
public class TimerUtil {  
    public int secondsBetween(int start, int end) {  
        if(end < start) throw new IllegalArgumentException("End < start");  
        return end - start;  
    }  
}
```

### Test ideas

- Normal case: start = 10, end = 25 → 15 s.
- Boundary: start = 0, end = 0 → 0 s.
- Invalid input: expect IllegalArgumentException.

## Lab Task 5 — Shopping Cart (Mini Case Study)

**Objective:** Combine multiple test assertions and simulate business rules.

### Implementation (ShoppingCart.java)

```
import java.util.*;
public class ShoppingCart {
    private final List<String> items = new ArrayList<>();
    public void addItem(String item){ items.add(item); }
    public void removeItem(String item){ items.remove(item); }
    public int getItemCount(){ return items.size(); }
    public void clear(){ items.clear(); }
}
```

### Test ideas

- Add 3 items → count = 3.
- Remove 1 item → count = 2.
- Clear cart → count = 0.
- Removing a non-existent item doesn't throw an error.

## 12. Homework

### Homework 1 – Enhanced Calculator

**Objective:** Extend the in-lab Calculator1 class by adding advanced operations and ensuring comprehensive test coverage.

### Tasks

1. Add new methods:
  - power(int base, int exp)

- `modulus(int a, int b)` → throws `IllegalArgumentException` if `b = 0`
- 2. Write JUnit 5 tests covering normal, boundary, and exceptional cases.
- 3. Use `@BeforeEach` and `@AfterEach` to initialize and reset the calculator instance.
- 4. Generate a code-coverage report in NetBeans and record the percentage.
- 5. Submit:
  - `Calculator1.java`, `Calculator1Test.java`
  - coverage screenshot
  - short reflection ( $\approx$  100 words on what improved testing quality)

## Homework 2 – Account Validation

**Objective:** Model a small bank account and verify business rules.

### Tasks

1. Implement `BankAccount` with:

```
public void deposit(double amt);  
public void withdraw(double amt);  
public double getBalance();  
public boolean isActive();
```

- Disallow negative deposits or withdrawals beyond balance.
  - Mark inactive if `balance < 100` after withdrawal.
2. Write JUnit tests using `assertThrows`, `assertTrue`, `assertFalse`.

3. Add at least five distinct test cases: valid deposit, overdraft, inactivity, etc.
4. Submit test-case table + source files

## Homework 3 – String Utility Testing

**Objective:** Apply JUnit to verify correctness of string-processing logic.

### Tasks

1. Implement StringAnalyzer with methods:

```
public boolean isPalindrome(String s);  
public int countVowels(String s);  
public boolean isAnagram(String s1, String s2);
```

2. Write JUnit tests that include:

- Empty or null inputs (should return false).
- Case-insensitive checks for palindromes/anagrams.
- Various vowel-count scenarios.

3. Use @DisplayName and meaningful test names.

4. Document results in a Test Case Summary Table and add one observation about potential edge-case handling.

### **13. Submission Format**

- Source code files (.java, .java test).
- Screenshot of successful JUnit results.
- Test-case table (method, input, expected output, result).
- Reflection paragraph (what you learned or would improve).

### **14. Conclusion**

JUnit testing in NetBeans provides a systematic method to validate Java programs. By writing structured test methods and verifying expected outputs, students gain a clear understanding of unit testing principles and software quality assurance practices in industry and research contexts.