JUNIT FRAMEWORK

Lesson Objectives

After the course, attendees will be able to:

- Understand Unit testing and Junit
- Know how to write a JUnit test class
- Know how to install JUnit and run a JUnit test case
- Know about some tips of Unit testing

Table Content

Unit testing

- In *computer programming*, **unit testing** is a software testing method by which **individual units** of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.

(*by wikipedia*)

- **Unit testing**, a testing technique using which **individual modules** are tested to determine if there are any issues by the developer himself. It is concerned with functional correctness of the standalone modules.
  - The main aim is to isolate each unit of the system to *identify*, *analyze* and fix the *defects*.

(by tutorialspoint)

Unit Test – What and Who ?

- **Unit Testing Actions:**
  - ★ **Validate** that individual units of software program are **working properly**.
  - ★ **A unit** is the smallest testable part of an application (In procedural programming a unit may be an individual program, function, procedure, etc., while in **object-oriented** programming, the smallest unit is always a **method**)

- **Unit Testing Deliverables:**
  - ★ Tested software units
  - ★ Related documents (Unit Test case, Unit Test Report)

- **Unit Testing Conductor:** Development team

Unit Test – Why ?

- Ensure **quality of software unit**.

- Detect **defects** and **issues** early.

- **Reduce** the Quality Effort & Correction **Cost**.

Section 2

INTRODUCTION TO JUNIT

What is JUnit?

- **JUnit** is an ***open source framework*** provided by JUnit.
  - JUnit enables us to write repeatable tests.
  - It is mainly used by Java developers to write unit test cases.
- The **objective of unit testing** is to **break code into multiple pieces** and **test each piece** of code separately to ensure that it works as it should be.
- Integrated nicely with many IDEs, Ant.
- Easy to use and to learn.
- **Version**: Junit 3, Junit 4, Junit 5

Why you need JUnit testing ?

- It **finds bugs early** in the code, which makes our code more reliable.
- JUnit is **useful for developers**, who work in a test-driven environment.
- Unit testing forces a developer to **read code more than writing**.
- You develop more **readable**, **reliable** and **bug-free code** which builds confidence during development.

What is JUnit 5?

- Unlike previous versions of JUnit, JUnit 5 is composed of several different modules from three different sub-projects.

  JUnit 5 = ***JUnit Platform* + *JUnit Jupiter* + *JUnit Vintage***

  - **JUnit Platform**: serves as a foundation for launching testing frameworks on the JVM. First-class support for the JUnit Platform also exists in popular **IDEs** (see IntelliJ IDEA, Eclipse, NetBeans, and Visual Studio Code) and build tools (see Gradle, Maven, and Ant).
  - **JUnit Jupiter**:
    - Blend of new programming model for writing tests and extension model for extensions
    - Addition of new annotations like `@BeforeEach`, `@AfterEach`, `@AfterAll`, `@BeforeAll` etc.
  - **JUnit Vintage**: provides a TestEngine for running JUnit 3 and JUnit 4 based tests on the platform.

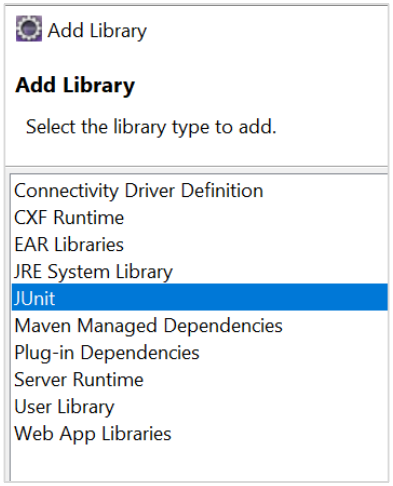
Installation JUnit 5

- JUnit 5 requires Java 8 (or higher) at runtime. However, you can still test code that has been compiled with previous versions of the JDK.
- **JUnit Maven Dependencies**

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.5.2</version>
</dependency>
<dependency>
    <groupId>org.junit.platform</groupId>
    <artifactId>junit-platform-runner</artifactId>
    <version>1.5.2</version>
</dependency>
```

Install JUnit jar file in Eclipse

- Add to Libs: R-click to your project | select **Java Build Path** | click button **Add Library** | **JUnit**:



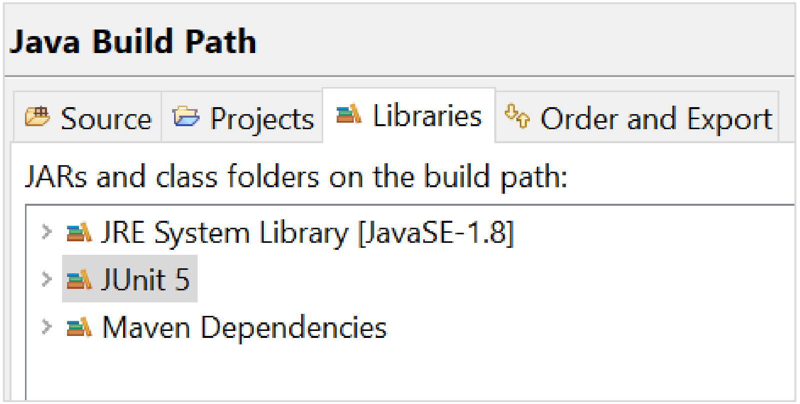
Add Library

Add Library

Select the library type to add.

Connectivity Driver Definition
CXF Runtime
EAR Libraries
JRE System Library
JUnit
Maven Managed Dependencies
Plug-in Dependencies
Server Runtime
User Library
Web App Libraries

- **Result**:



Java Build Path

JARs and class folders on the build path:

- JRE System Library [JavaSE-1.8]
- JUnit 5
- Maven Dependencies

Eclipse Eclipse IDE offers support for the JUnit Platform since the Eclipse Oxygen.1a (4.7.1a) release.

JUnit Annotations

- Listed below are some commonly used annotations provided in it:

Annotation	Description
@Test	Denotes a test method
@DisplayName	Declares a custom display name for the test class or test method
@BeforeEach	Denotes that the annotated method should be executed before each test method
@AfterEach	Denotes that the annotated method should be executed after each test method
@BeforeAll	Denotes that the annotated method should be executed before all test methods
@AfterAll	Denotes that the annotated method should be executed after all test methods
@Disable	Used to disable a test class or test method
@Nested	Denotes that the annotated class is a nested, non-static test class
@Tag	Declare tags for filtering tests
@ExtendWith	Register custom extensions

Section 3

JUNIT ASSERTION

Assertion Overview

- **JUnit 5 assertions** help in validating the expected output with actual output of a testcase. To keep things simple, all **JUnit Jupiter assertions** are static methods in the org.junit.jupiter.Assertions class.
  - `Assertions.assertEquals() and Assertions.assertNotEquals()`
  - `Assertions.assertArrayEquals()`
  - `Assertions.assertIterableEquals()`
  - `Assertions.assertLinesMatch()`
  - `Assertions.assertNotNull() and Assertions.assertNull()`
  - `Assertions.assertNotSame() and Assertions.assertSame()`
  - `Assertions.assertTimeout() and Assertions.assertTimeoutPreemptively()`
  - `Assertions.assertTrue() and Assertions.assertFalse()`
  - `Assertions.assertThrows()`
  - `Assertions.fail()`

JUnit Assertion methods

- **Boolean:** If you want to test the boolean conditions (true or false), you can use following assert methods
  - **assertTrue(condition)**
  - **assertFalse(condition)**

  Here the condition is a boolean value.
- **Null object:** If you want to check the initial value of an object/variable, you have the following methods:
  - **assertNull(object)**
  - **assertNotNull(object)**

  Here object is java object **e.g.** assertNull(actual).

JUnit Assertion methods

- **Identical:** If you want to check whether the objects are identical (i.e. comparing two references to the same java object), or different.
  - **assertSame(expected, actual),** It will return true if **expected == actual**
  - **assertNotSame(expected, actual).**
- **Assert Equals:** If you want to test equality of two objects, you have the following methods
  - **assertEquals(expected, actual)**

  It will return true if: **expected.equals(actual)** returns true.

JUnit Assertion methods

- **Assert Array Equals:** If you want to test equality of arrays, you have the following methods as given below:
  - **assertArrayEquals(expected, actual)**

  Above method must be used if arrays have the same length, for each valid value for **i**, you can check it as given below:
  - **assertEquals(expected[i],actual[i])**
  - **assertArrayEquals(expected[i],actual[i]).**
- **Fail Message:** If you want to throw any assertion error, you have **fail()** that always results in a fail verdict.
  - **Fail(message);**

  You can have assertion method with an additional **String** parameter as the first parameter. This string will be appended in the failure message if the assertion fails. E.g. **fail(message)** can be written as
  - **assertEquals(message, expected, actual).**

JUnit assertEquals

- You have **assertEquals(a,b)** which relies on the **equals()** method of the Object class.
  - Here it will be evaluated as **a.equals(b).**
- If a class does not override the **equals()** method of **Object** class, it will get the default behaviour of **equals()** method, i.e. object identity.
- If **a** and **b** are primitives such as **byte**, **int**, **boolean**, etc. then the following will be done for assertEquals(a,b) :
  - **a** and **b** will be converted to their equivalent wrapper object type (**Byte,Integer**, **Boolean**, etc.), and then **a.equals(b)** will be evaluated.
- Example:

```
String obj1="Junit";
String obj2="Junit";
assertEquals(obj1,obj2);
```

➔ Above assert statement will return true as obj1.equals(obj2) returns true.

Floating point assertions

- When you want to compare floating point types (e.g. **double** or **float**), you need an additional required parameter **delta** to avoid problems with round-off errors while doing floating point comparisons.
- The assertion evaluates as given below:
  - **Math.abs(expected – actual) <= delta**
- For example:
  - **assertEquals(aDoubleValue, anotherDoubleValue, 0.001)**

JUnit Assertion methods

- **Assertions.assertIterableEquals()**: It asserts that **expected and actual iterables are deeply equal**. Deeply equal means that number and order of elements in collection must be same; as well as iterated elements must be equal.
- It also has 3 overloaded methods.
  - public static void assertIterableEquals(Iterable<?> expected, Iterable> actual)
  - public static void assertIterableEquals(Iterable<?> expected, Iterable> actual, String message)
  - public static void assertIterableEquals(Iterable<?> expected, Iterable> actual, Supplier<String> messageSupplier)

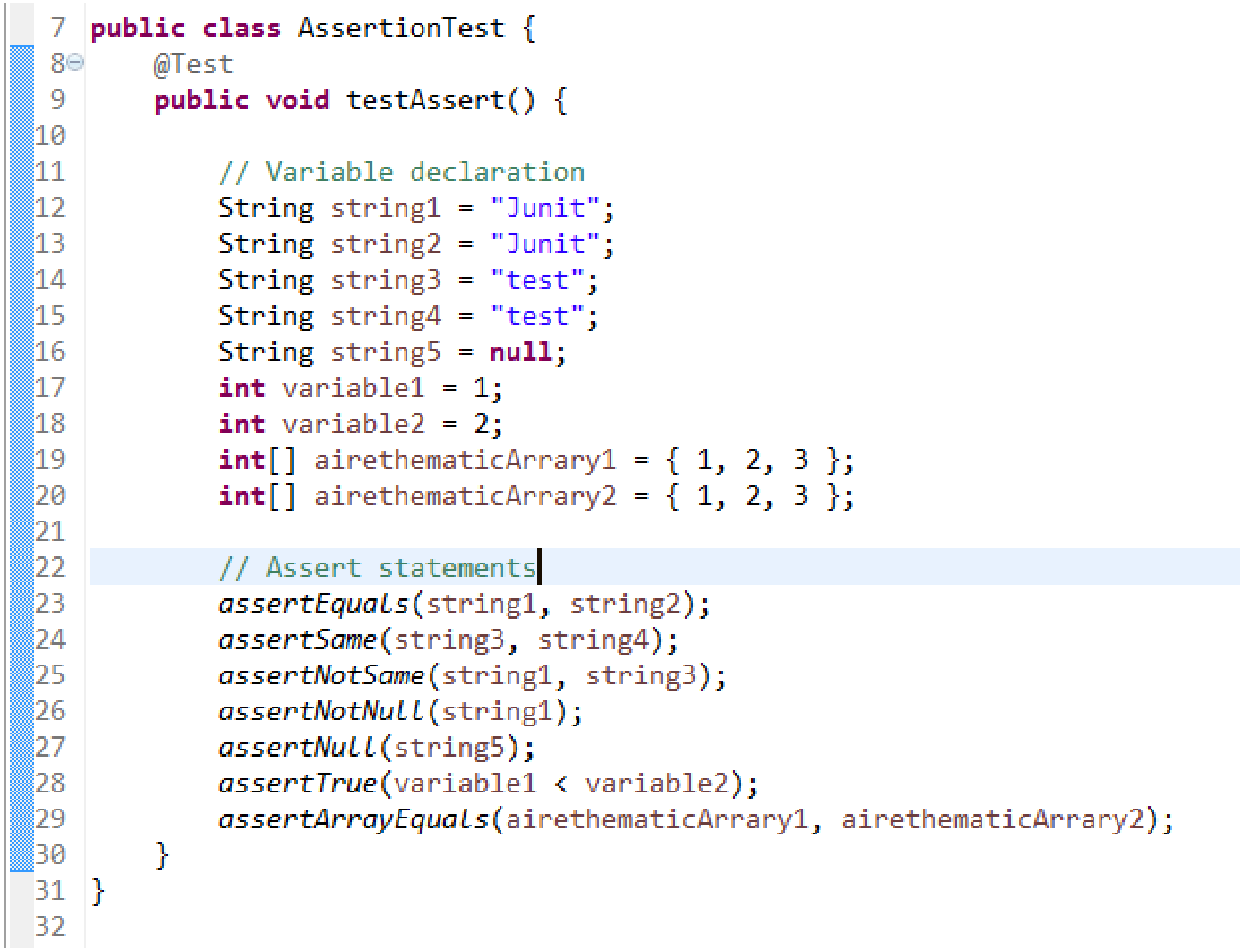
JUnit Assertion methods

- **Example:**

```
@Test
 void testCase() {
   Iterable<Integer> listOne = new ArrayList<>(Arrays.asList(1, 2, 3, 4));
   Iterable<Integer> listTwo = new ArrayList<>(Arrays.asList(1, 2, 3, 4));
   Iterable<Integer> listThree = new ArrayList<>(Arrays.asList(1, 2, 3));
   Iterable<Integer> listFour = new ArrayList<>(Arrays.asList(1, 2, 4, 3));

   // Test will pass
   Assertions.assertIterableEquals(listOne, listTwo);

   // Test will fail
   Assertions.assertIterableEquals(listOne, listThree);

   // Test will fail
   Assertions.assertIterableEquals(listOne, listFour);
 }
```

Practical time

- Let's create a simple test class named **AssertionTest.java.**
- You will create few variables and important assert statements in JUnit.



```java
public class AssertionTest {
    @Test
    public void testAssert() {

        // Variable declaration
        String string1 = "Junit";
        String string2 = "Junit";
        String string3 = "test";
        String string4 = "test";
        String string5 = null;
        int variable1 = 1;
        int variable2 = 2;
        int[] airethematicArrary1 = { 1, 2, 3 };
        int[] airethematicArrary2 = { 1, 2, 3 };

        // Assert statements
        assertEquals(string1, string2);
        assertSame(string3, string4);
        assertNotSame(string1, string3);
        assertNotNull(string1);
        assertNull(string5);
        assertTrue(variable1 < variable2);
        assertArrayEquals(airethematicArrary1, airethematicArrary2);
    }
}

```

Section 4

JUNIT 5 TEST LIFECYCLE

Before And After

- In JUnit 5, test lifecycle is driven by **4 primary annotations** i.e. @BeforeAll, @BeforeEach, @AfterEach and @AfterAll.
- Along with it, each test method must be marked with @Test annotation. @Test annotation is virtually unchanged, although it no longer takes optional arguments.
- **Why?**
  - You will primarily need to have some methods to **setup** and **tear down** the environment or test data on which the tests run.
  - @BeforeAll and @AfterAll annotations should be **called only once** in entire tests execution cycle. So they must be declared **static**.
  - @BeforeEach and @AfterEach are invoked for each instance of test so they need not to be static.

Writing Tests in JUnit 5

```java
package fa.training.jpe;

import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

/**
 * Unit test for simple App.
 */
public class AppTest {
  @BeforeAll
  static void setup() {
    System.out.println("@BeforeAll executed");
  }

  @BeforeEach
  void setupThis() {
    System.out.println("@BeforeEach executed");
  }

  @Tag("DEV")
  @Test
  void testCalcOne() {
    System.out.println("======TEST ONE EXECUTED=======");
    Assertions.assertEquals(4, Calculator.add(2, 2));
  }

  @Tag("PROD")
  @Disabled
  @org.junit.jupiter.api.Test
  void testCalcTwo() {
    System.out.println("======TEST TWO
                                    EXECUTED=======");
    Assertions.assertEquals(6,
                                    Calculator.add(2, 4));
  }

  @AfterEach
  void tearThis() {
    System.out.println("@AfterEach executed");
  }

  @AfterAll
  static void tear() {
    System.out.println("@AfterAll executed");
  }
}
```

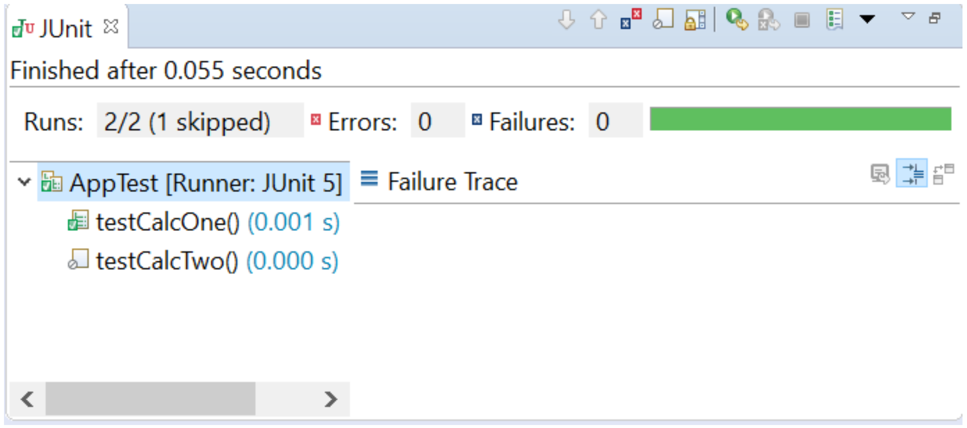
Writing Tests in JUnit 5

- Where Calculator class is:

```
package fa.training.jpe;


public class Calculator {
  public static int add(int a, int b) {
    return (a + b);
  }
}
```

Writing Tests in JUnit 5

- Result Execute:



JUnit

Finished after 0.055 seconds

Runs: 2/2 (1 skipped) Errors: 0 Failures: 0

AppTest [Runner: JUnit 5]
testCalcOne() (0.001 s)
testCalcTwo() (0.000 s)

Failure Trace

- Console Output:

```
@BeforeAll executed
@BeforeEach executed
======TEST ONE EXECUTED=======
@AfterEach executed
@AfterAll executed
```

Disabling Test

- To disable a test in JUnit 5, you will need to use @Disabled annotation.
- **@Disabled** annotation can be applied over test class (**disables all test methods** in that class) or **individual test methods** as well.

```
@Disabled
 @Test
 void testCalcTwo() {
     System.out.println("======TEST TWO EXECUTED=======");
     Assertions.assertEquals( 6 , Calculator.add(2, 4));
 }
```

@BeforeEach and @AfterEach annotations

- The **@BeforeEach** is used to signal that the **annotated method should be executed before each @Test method in the current class**.
- The **@AfterEach** is used to signal that the **annotated method should be executed after each @Test method in the current class**.
- @BeforeEach and @AfterEach annotated methods **MUST NOT be a static** method otherwise it will throw runtime error.
- **Example:**

```
package fa.training.jpe;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class BeforeEachTest {

@Test
void addNumber() {
      System.out.println("Running test -> 1");
      Assertions.assertEquals(2, Calculator.add(1, 1), "1 + 1 should equal 2");
}
```

@BeforeEach and @AfterEach annotations

- **Example:**

```java
  @Test
  void addNumber2() {
    System.out.println("Running test -> 2");
    Assertions.assertEquals(2, Calculator.add(1, 1), "1 + 1 should equal 2");
  }

  @Test
  void addNumber3() {
    System.out.println("Running test -> 3");
    Assertions.assertEquals(2, Calculator.add(1, 1), "1 + 1 should equal 2");
  }

  @BeforeAll
  public static void init() {
    System.out.println("BeforeAll init() method called");
  }

  @BeforeEach
  public void initEach() {
    System.out.println("BeforeEach initEach() method called");
  }

 @AfterEach
  public void cleanUpEach() {
    System.out.println("After Each cleanUpEach() method called");
  }

 @AfterAll
  public static void cleanUp() {
    System.out.println("After All cleanUp() method called");
  }
}
```

@BeforeEach and @AfterEach annotations

- **Console Output:**

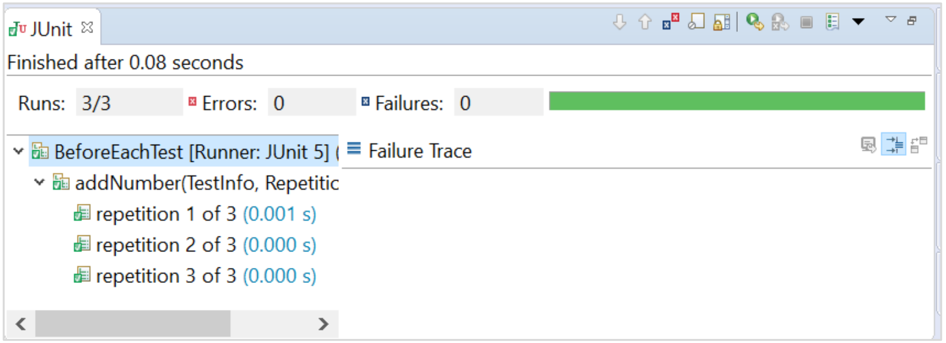
```
BeforeAll init() method called
BeforeEach initEach() method called
Running test -> 1
After Each cleanUpEach() method called
BeforeEach initEach() method called
Running test -> 2
After Each cleanUpEach() method called
BeforeEach initEach() method called
Running test -> 3
After Each cleanUpEach() method called
After All cleanUp() method called
```

@RepeatedTest Annotation

- JUnit 5 @RepeatedTest annotation enable to write **repeatable test templates** which could be run multiple times.
- **Example**:

```
@RepeatedTest(3)
 void addNumber(TestInfo testInfo, RepetitionInfo repetitionInfo) {
   System.out
        .println("Running test -> " + repetitionInfo.getCurrentRepetition());
   Assertions.assertEquals(2, Calculator.add(1, 1), "1 + 1 should equal 2");
 }
```

- **Result**:



```
JUnit
Finished after 0.08 seconds
Runs: 3/3    Errors: 0    Failures: 0

BeforeEachTest [Runner: JUnit 5] (
  addNumber(TestInfo, Repetitic
    repetition 1 of 3 (0.001 s)
    repetition 2 of 3 (0.000 s)
    repetition 3 of 3 (0.000 s)
```

Section 5

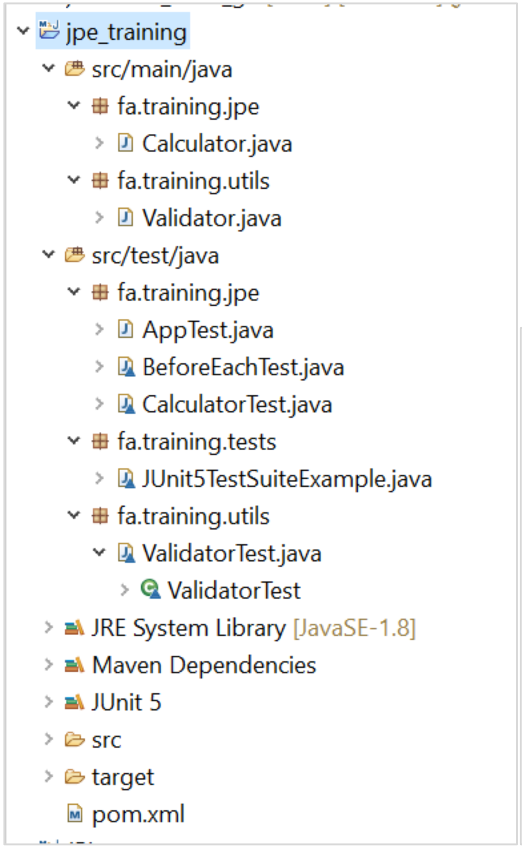
JUNIT 5 TEST SUITE

Test Suite Overview

- Using **JUnit 5 test suites**, you can run tests spread into multiple test classes and different packages.
- JUnit 5 provides two annotations: @SelectPackages and @SelectClasses to create test suites. Additionally, you can use other annotations for filtering test packages, classes or even test methods.
  - **@SelectClasses** specifies the classes to select when running a test suite
  - **@SelectPackages** specifies the names of packages to select when running a test suite
- As we learn that @SelectPackages causes all it's sub-packages as well to be scanned for test classes.
- If you want to exclude any specific sub-package, or include any package then you may use @IncludePackages and @ExcludePackages annotations.

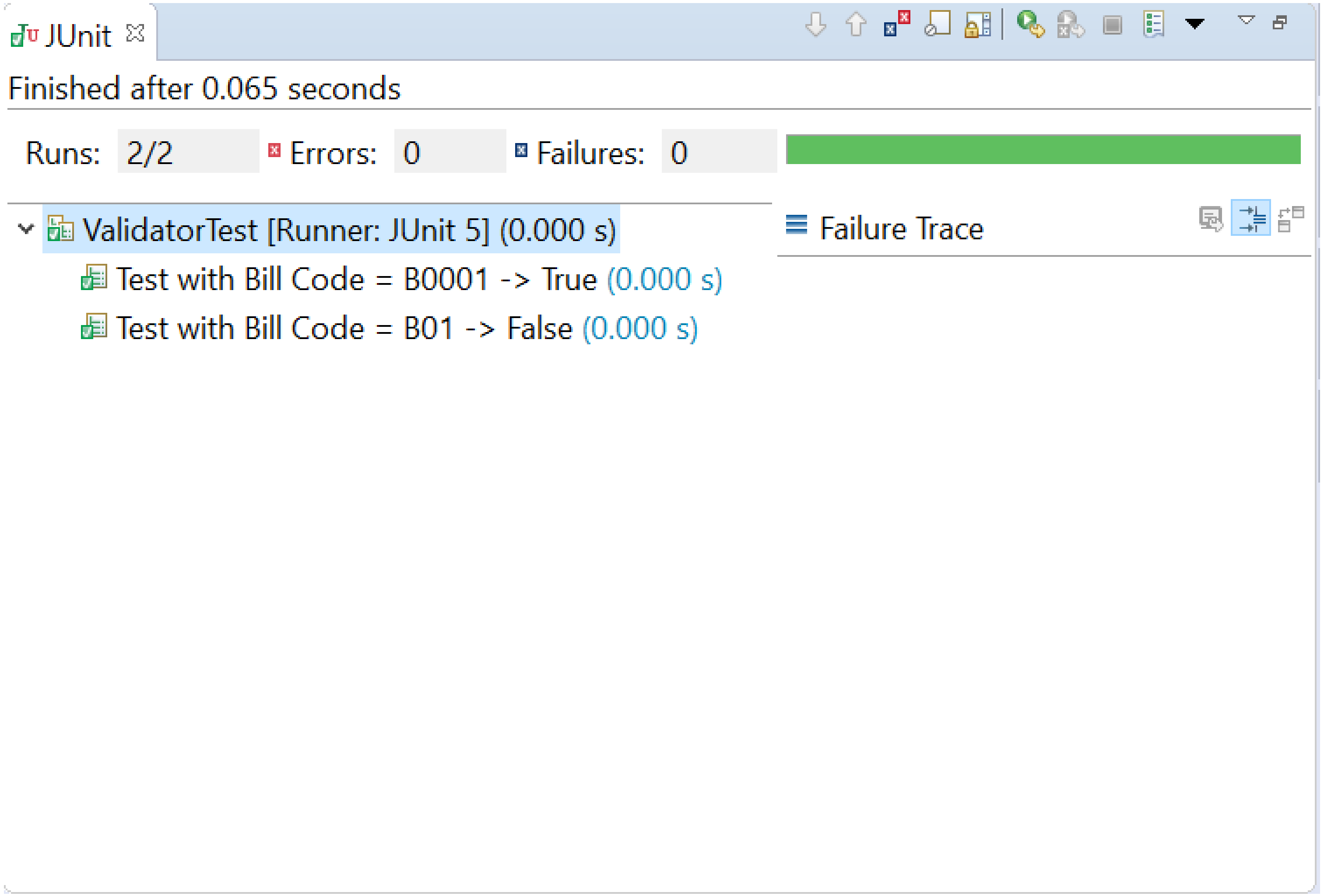
Test Suite Example

➤ Project Structure:



```
jpe_training
  src/main/java
    fa.training.jpe
      Calculator.java
    fa.training.utils
      Validator.java
  src/test/java
    fa.training.jpe
      AppTest.java
      BeforeEachTest.java
      CalculatorTest.java
    fa.training.tests
      JUnit5TestSuiteExample.java
    fa.training.utils
      ValidatorTest.java
        ValidatorTest
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  JUnit 5
  src
  target
  pom.xml
```

```java
package fa.training.tests;

import org.junit.platform.runner.JUnitPlatform;
import org.junit.platform.suite.api.SelectPackages;
import org.junit.runner.RunWith;

@RunWith(JUnitPlatform.class)
@SelectPackages({ "fa.training.jpe", "fa.training.utils" })
class JUnit5TestSuiteExample {

}
```



JUnit

Finished after 0.065 seconds

Runs: 2/2   Errors: 0   Failures: 0

ValidatorTest [Runner: JUnit 5] (0.000 s)
- Test with Bill Code = B0001 -> True (0.000 s)
- Test with Bill Code = B01 -> False (0.000 s)

Failure Trace

Test Result

- A test method in a test case can have one of three results:
  - Pass – all assertions matched expected values
  - Failed – an assertion did not match expected value
  - Error – an unexpected exception was thrown during execution of test method

JUnit Exception Test

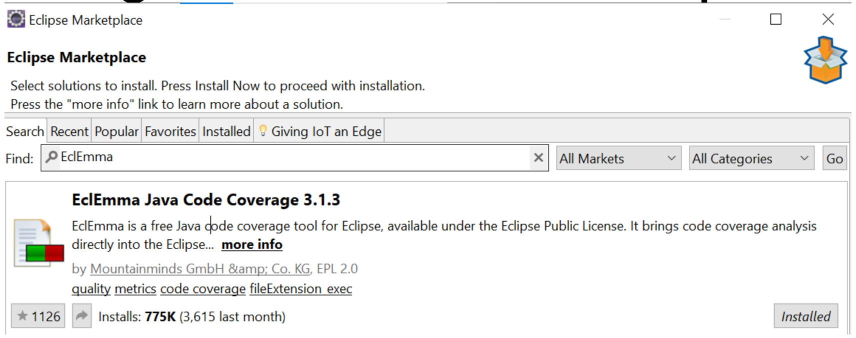
- JUnit provides the facility to trace the exception and also to check whether the code is throwing expected exception or not.
- Junit4 provides an easy and readable way for exception testing, you can use:
  - Optional parameter (**expected**) of @test annotation and
  - To trace the information ,"fail()" can be used.

Practical time

1. Create unit test class for Rectangle class.
2. Create unit test class for EmployeeRepository class
3. Create unit test class for HangKhong Project (Junit test cover 80%)

Resources & References

- Resources
  - www.junit.org/
  - http://junit.sourceforge.net
- Recommended readings
  - Manning – Junit in Action
  - Test Driven Development: By Example. Boston: Addison-Wesley, 2003
  - Plugin **EclEmma** in Eclipse IDE for check test coverage.


Eclipse Marketplace

Eclipse Marketplace

Select solutions to install. Press Install Now to proceed with installation.
Press the "more info" link to learn more about a solution.

Search | Recent | Popular | Favorites | Installed | Giving IoT an Edge
Find: EclEmma
All Markets
All Categories
Go

EclEmma Java Code Coverage 3.1.3

EclEmma is a free Java code coverage tool for Eclipse, available under the Eclipse Public License. It brings code coverage analysis directly into the Eclipse... **more info**

by Mountainminds GmbH &amp; Co. KG, EPL 2.0

quality metrics code coverage fileExtension_exec

1126 Installs: **775K** (3,615 last month)

Summary

- Introduction to Junit
- Setting up JUnit
- JUnit Test framework
- Junit Assert
- Junit TestSuite
- Examples

Thank you