

1. Lifecycle Methods

Step 1.1: Writing code for `@BeforeAll` annotation, `@BeforeEach` annotation, `@AfterAll` annotation, and `@AfterEach` annotation:

Below is a sample program that includes all the annotations mentioned above. These annotations play a very specific role in the test execution order.

- `@BeforeAll` is used to signal that the annotated method must be executed before all the tests in the current test class.
- `@BeforeEach` is used to signal that the annotated method must be executed before each `@Test` method in the current class.
- `@AfterAll` is used to signal that the annotated method must be executed after all the tests in the current test class.
- `@AfterEach` is used to signal that the annotated method must be executed after each `@Test` method in the current class.

```
package com.simplilearn.unittest;
```

```
import org.junit.jupiter.api.AfterAll;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeAll;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
```

```
@DisplayName("The standard annotations for Junit")
public class JunitSdAnnotations {
```

```
    // Junit Fixture
    @BeforeEach
    public void setUp() {
        System.out.println("--- Before each is executed. ---");
    }
```

```
    @AfterEach
    public void cleanUp() {
        System.out.println("--- After each is executed. ---");
    }
```

```
    @BeforeAll
```

```

public static void setUpAll() {
    System.out.println("--- Before All test is executed. ---");
}

@AfterAll
public static void cleanUpAll() {
    System.out.println("--- After All test is executed. ---");
}

@Test
@DisplayName("Test One")
public void testOne() {
    System.out.println("--- Test One is executed. ---");
}

@Test
@DisplayName("Test Two")
public void testTwo() {
    System.out.println("--- Test Two is executed. ---");
}

@Test
@DisplayName("Test Three")
public void testThree() {
    System.out.println("--- Test Three is executed. ---");
}
}

```

•

```

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

@BeforeEach executed
=====TEST TWO EXECUTED=====
@AfterEach executed

@AfterAll executed

```

2. Assertions

Steps 2.1: Explaining Assertion Methods:

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

```
assertTrue(condition)
```

```
assertFalse(condition)
```

Here, the condition is a boolean value.

- **Identical:** If you want to check the initial value of an object/variable, you have the following methods:

```
assertNull(object)
```

```
assertNotNull(object)
```

Here, the object is a Java object, for e.g. `assertNull(actual)`;

- **Null object:** If you want to check whether the objects are identical (i.e. comparing two references to the same java object) or different, follow the below methods:

```
assertSame(expected, actual), It will return true if expected == actual
```

```
assertNotSame(expected, actual)
```

- **Assert Equals:** If you want to test the equality of two objects, you have the following methods:

```
assertEquals(expected, actual)
```

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

- **Assert Array Equals :**

```
assertArrayEquals(expected, actual)
```

The above method must be used if the arrays have the same length for each valid value for `i` as shown 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 the 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**

assertEquals(a , b) which relies on the equals() method of the Object class.

- 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.

For Example: Consider that the below-mentioned strings have the same values, let's test it using assertTrue.

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

The above assert statement will return true as obj1.equals(obj2) returns true.

- Floating point assertions

When you want to compare the 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:

- $\text{Math.abs(expected - actual)} \leq \text{delta}$

For example:

```
assertEquals( aDoubleValue, anotherDoubleValue, 0.001 )
```

Steps 2.2: Writing code for Assertions

```
package com.simplilearn.unittest;

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

import org.junit.jupiter.api.DisplayName;

public class AssertionTest {

    String userName = "Sasidhar";
    int age = 30;
    String email = null;

    Object actual = new Object();
    Object referance = actual;

    int[] aNumbers = new int[] { 10, 20, 30 };
    int[] eNumbers = new int[] { 10, 20, 30 };

    @Test
    @DisplayName("Should match expected and actual")
    public void test1() {
        assertEquals(userName, "Sasidhar");
        assertEquals(age, 30);
    }

    @Test
    @DisplayName("Should be true")
    void shouldBeTrue() {
```

```
        assertTrue(true);
        assertTrue(age > 20);
    }
```

```
@Test
@DisplayName("Should be false")
void shouldBeFalse() {
    assertFalse(false);
    assertFalse(age < 20);
}
```

```
@Test
@DisplayName("Should be null")
void shouldBeNull() {
    assertNull(email);
}
```

```
@Test
@DisplayName("Should be not null")
void shouldBeNotNull() {
    assertNotNull(userName);
}
```

```
@Test
@DisplayName("Should refer to the same object")
void shouldReferToSameObject() {
    assertSame(reference, actual);
}
```

@Test

@DisplayName("Should contain the same integers array element")

```
void shouldContainSameIntegers() {  
    assertEquals(aNumbers, eNumbers);  
}
```

@Test

@DisplayName("Should assert mulple Assertions")

```
public void lambdaExpressionTest3() {  
    int[] numbers = { 0, 1, 20, 3, 40 };  
    assertEquals("Numbers", () -> assertEquals(numbers[0], 0));  
    assertEquals("Numbers", () -> assertEquals(numbers[2], 20));  
    assertEquals("Numbers", () -> assertEquals(numbers[2], 20), () ->  
assertEquals(numbers[4], 40));  
    assertEquals("Numbers", () -> assertEquals(numbers[2], 20), () ->  
assertEquals(numbers[4], 40),  
        () -> assertEquals(numbers[3], 3));  
}
```

@Test

@DisplayName("Should throw the correct exception")

```
void shouldThrowCorrectException() {  
  
    assertEquals(NullPointerException.class, () -> {  
        throw new NullPointerException();  
    });  
}
```

@Test

@DisplayName("Should not throw an exception")

```

void shouldNotThrowException() {
    assertDoesNotThrow(()-> {
        // any logic -> should not throw exception
    });
}
}

```

- Next, create a java class file Calculator.java named in

```
package com.simplilearn.unittest;
```

```

public interface Calculator {

    // abstract method
    int add(int num1 , int num2);

    // default method
    default int multiply(int num1 , int num2) {
        return num1 * num2;
    }

}

```

Compile the Test Case and Test Runner classes



3. Disabling Tests

Step 3.1: Writing code to demonstrate @ignore annotation

Sometimes our code is not completely ready while running a test case. As a result, the test case fails. The **@Ignore** annotation helps in this scenario.

- A test method annotated with `@Ignore` will not be executed.
- If a test class is annotated with `@Ignore`, then none of its test methods will be executed.

Junit Test Example - Ignore

We can use `@Ignore` annotation to ignore a test or a group of tests.

Let's understand it using simple examples and in the scenarios given below:

1. Creating a simple test class without ignoring a test
2. Ignore a test method using @Ignore annotation
3. Ignore a test method using @Ignore annotation with proper reason
4. Ignore all test methods using @Ignore annotation

Creating a simple test class with ignoring a test method using @Ignore annotation

Let's write a program to disable a test. For this, you need to use `@Ignore` in the method you want to skip.

Let's do it for `testJUnitMessage()` of `JUnitTestExample.java`

DisableTest.Java

```
package com.simplilearn.junittest;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Disabled;
import org.junit.jupiter.api.DisplayName;
import org.junit.jupiter.api.Test;
```

```
import org.junit.jupiter.api.condition.DisabledOnJre;
```

```
import org.junit.jupiter.api.condition.DisabledOnOs;
```

```
import org.junit.jupiter.api.condition.JRE;
```

```
import org.junit.jupiter.api.condition.OS;
```

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

```
@DisplayName("Test Age calculator")
```

```
// @Disabled
```

```
public class DisableTest {
```

```
    AgeCalculator ageCalculator;
```

```
    @BeforeEach
```

```
    public void setUp() {
```

```
        ageCalculator = new AgeCalculator();
```

```
    }
```

```
    @AfterEach
```

```
    public void cleanUp() {
```

```
        if(ageCalculator!=null)
```

```
            ageCalculator = null;
```

```
    }
```

```
@Test

@Disabled

@DisplayName("Should return valid age for +ve year")

public void test1() {

    int eResult = 33;

    int aResult = ageCalculator.calculateAge(1990);

    assertEquals(eResult, aResult);

    // assertEquals(28, ageCalculator.calculateAge(1995));

}
```

```
@Test

@DisabledOnOs(value = OS.WINDOWS)

@DisplayName("Should return 0 age for -ve year")

public void test2() {

    assertEquals(0, ageCalculator.calculateAge(-1995));

}
```

```
@Test

@DisabledOnJre(value = JRE.JAVA_17)

@DisplayName("Should return 0 age for 0 year")

public void test3() {

    assertEquals(0, ageCalculator.calculateAge(0));

}
```

```
@Test

@DisabledOnJre({JRE.JAVA_17, JRE.JAVA_8})

@DisplayName("Should return 0 age for +ve future year")

public void test4() {

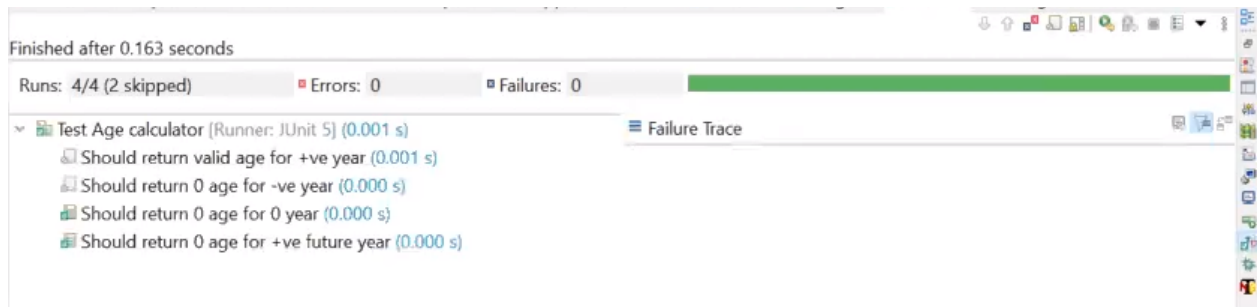
    assertEquals(0, ageCalculator.calculateAge(2050));

}

}
```

Output:

Let's execute and verify the output of the above example.



4. Assumptions

Steps 4.1: Importing a JUnit Assumption

- JUnit Assumptions class provides a useful collection of assumption methods. To import them in our test class, write the commands given below.

```
import static org.junit.jupiter.api.Assumptions.*;  
import static org.junit.Assume.assumeTrue;
```

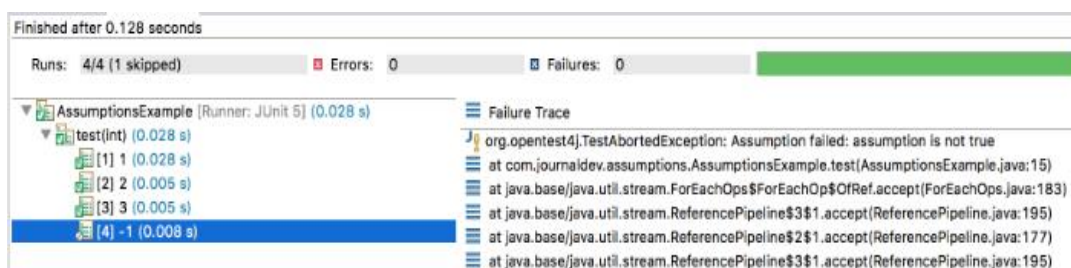
Steps 4.2: Writing a code to demonstrate the types of Assumptions

- assumeTrue()

We can use `assumeTrue()` to skip the test if the input number is negative. Below is the updated code:

```
11 |  
12 @ParameterizedTest  
13 @ValueSource(ints = {1,2,3,-1})  
14 void test(int i) {  
15     assumeTrue(i>=1);  
16     //assumeTrue(i >=0, "Wrong Input, Only positive ints please");  
17     try {  
18         Thread.sleep(i);  
19     } catch (InterruptedException e) {  
20         e.printStackTrace();  
21     }  
}
```

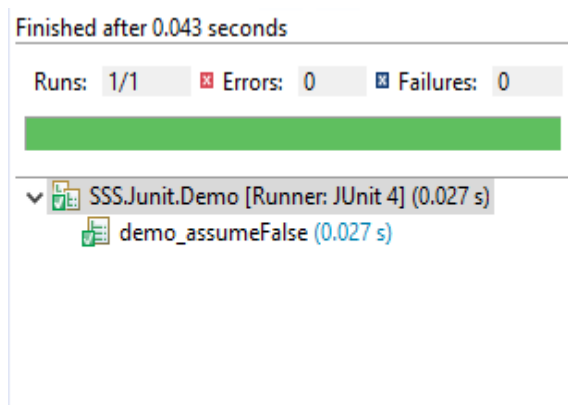
- JUnit will skip when the input number is negative. When we run the test, we will get the following output:



- assumeFalse()

`assumeFalse()` validates the given assumption to false and if the assumption is false then the test proceeds, otherwise, the test execution is aborted. It works just opposite to `assumeTrue()`.

```
11
12
13 @Test
14 public void demo_assumeFalse() {
15     assumeFalse("root".equals(System.getenv("USER")));
16     //I don't want to run as root user code
17 }
18
```



- `assumingThat()`

This method executes the supplied Executable if the assumption is valid. If the assumption is invalid, this method does nothing. We can use this for logging or notifications when our assumptions are valid.

```
13 @Test
14 void test_assumingThat() {
15     assumingThat("James".equals(System.getenv("USER")),
16         () -> {System.out.println("USER is James, continue further");});
17 }
18
```

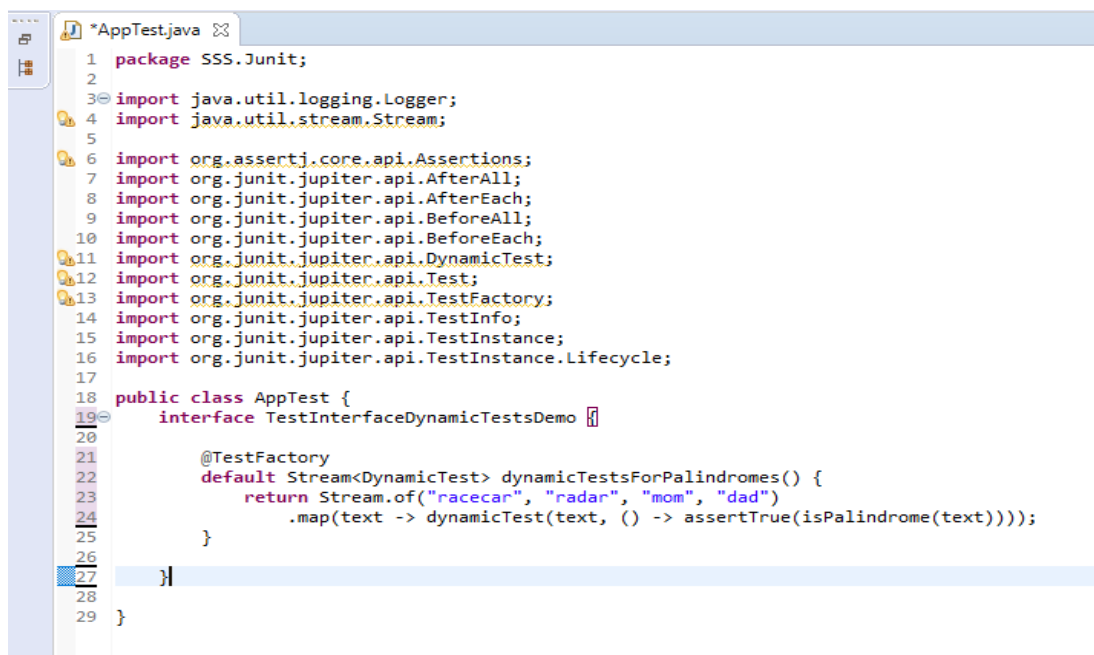
5. Test Interfaces and Default Methods

Step 5.1: Writing a code to demonstrate interfaces and default methods:

This section will guide you to write code with the annotations given below:

- Test
- RepeatedTest
- ParameterizedTest
- TestFactory
- TestTemplate
- BeforeEach
- AfterEach
- BeforeAll
- AfterAll

Step 5.2: Writing a code to demonstrate TestFactory



```
1 package SSS.Junit;
2
3 import java.util.logging.Logger;
4 import java.util.stream.Stream;
5
6 import org.assertj.core.api.Assertions;
7 import org.junit.jupiter.api.AfterAll;
8 import org.junit.jupiter.api.AfterEach;
9 import org.junit.jupiter.api.BeforeAll;
10 import org.junit.jupiter.api.BeforeEach;
11 import org.junit.jupiter.api.DynamicTest;
12 import org.junit.jupiter.api.Test;
13 import org.junit.jupiter.api.TestFactory;
14 import org.junit.jupiter.api.TestInfo;
15 import org.junit.jupiter.api.TestInstance;
16 import org.junit.jupiter.api.TestInstance.Lifecycle;
17
18 public class AppTest {
19     interface TestInterfaceDynamicTestsDemo {
20
21         @TestFactory
22         default Stream<DynamicTest> dynamicTestsForPalindromes() {
23             return Stream.of("racecar", "radar", "mom", "dad")
24                 .map(text -> dynamicTest(text, () -> assertTrue(isPalindrome(text))));
25         }
26     }
27 }
28
29 }
```

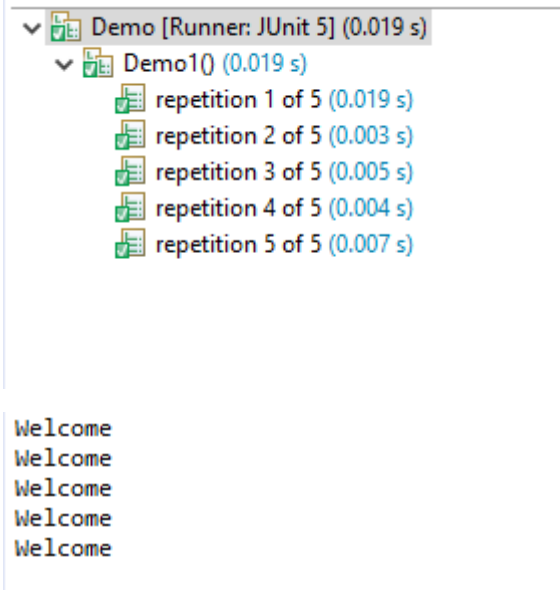
6. Repeating Tests

Step 6.1: Writing a code to demonstrate the execution of repeated tests

6.1.1 Repeated Tests Example

With the below example, we will use `@RepeatedTest` annotation (introduced in JUnit 5). This is more convenient to write the Junit test that we want to repeat several times.

```
11 public class Demo {  
12  
13     @RepeatedTest(5)  
14     void Demo1 () {  
15         System.out.println("Welcome");  
16     }  
17  
18  
19 }
```



6.1.2 @RepeatedTest DisplayName

We will be using `@DisplayName` annotation to declare a custom display name for the annotated test class or test method.

DisplayName value


```

1 package SSS.Junit;
2
3 import org.junit.jupiter.api.DisplayName;
4 import org.junit.jupiter.api.RepeatedTest;
5 import org.junit.jupiter.api.TestInfo;
6
7 public class Demo {
8
9
10 @RepeatedTest(value=3, name="{displayName} {currentRepetition}/{totalRepetitions}")
11 @DisplayName("Execution")
12 void test_with_cutom_DisplayName(TestInfo testInfo) {
13     System.out.println(testInfo.getDisplayName());
14 }
15
16 }

```

Finished after 0.168 seconds

Runs: 3/3 Errors: 0 Failures: 0

Demo [Runner: JUnit 5] (0.000 s)

- Execution (0.000 s)
 - Execution 1/3 (0.000 s)
 - Execution 2/3 (0.000 s)
 - Execution 3/3 (0.000 s)

Execution 1/3
 Execution 2/3
 Execution 3/3

6.1.3 JUnit @RepetitionInfo

When we injected TestInfo into our test method, JUnit Jupiter provides @RepetitionInfo annotation that we can inject into our test method.

```

14 public class Demo {
15
16 @RepeatedTest(5)
17 void test_with_RepetitionInfo_Injection(RepetitionInfo repetitionInfo) {
18     System.out.println("Number of excecution");
19     assertEquals(5, repetitionInfo.getTotalRepetitions());
20     System.out.println("Current Test Count = "+repetitionInfo.getCurrentRepetition());
21 }
22
23 }
24

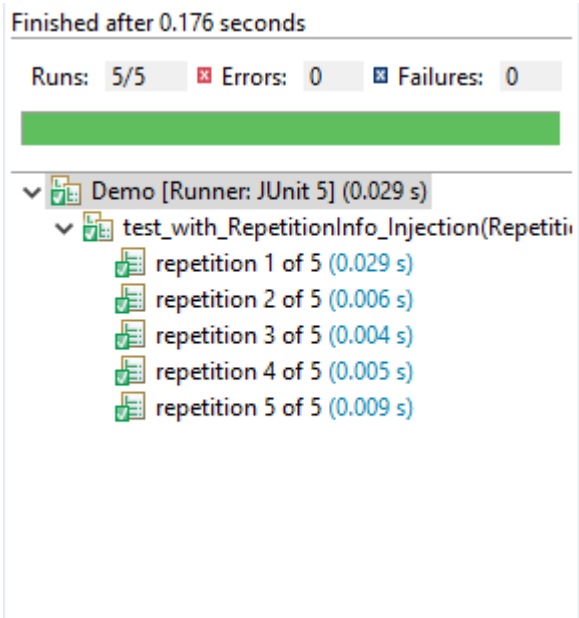
```

```

Number of execution
Current Test Count = 1
Number of execution
Current Test Count = 2
Number of execution
Current Test Count = 3
Number of execution
Current Test Count = 4
Number of execution
Current Test Count = 5

```

Activ



If we have many such methods, then we can move it to `@BeforeEach` or `@AfterEach` methods too.

```

@BeforeEach
void setUp(RepetitionInfo repetitionInfo, TestInfo testInfo) {
    System.out.println("Method = "+testInfo.getTestMethod().get().getName()+"",
        Execution Count = "+repetitionInfo.getCurrentRepetition());
}

```

The above mentioned `@BeforeEach` method will throw an error if all the test methods are not annotated with `@RepeatedTest`.

```

org.junit.jupiter.api.extension.ParameterResolutionException: No ParameterResolver
registered for parameter

```

```

[org.junit.jupiter.api.RepetitionInfo arg0] in executable

```

```

[void com.journaldev.repeatedtests.RepeatedTestExample.setUp

```

```

(org.junit.jupiter.api.RepetitionInfo,org.junit.jupiter.api.TestInfo)]

```

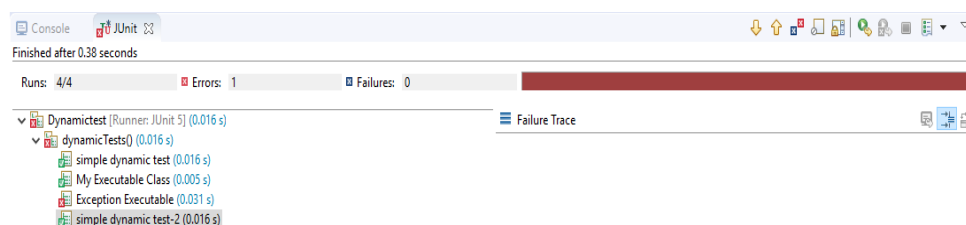
7. Dynamic Tests

Step 7.1: Writing a code to import Dynamic Test libraries

- import org.junit.jupiter.api.DynamicTest;
- import org.junit.jupiter.api.TestFactory;
- import org.junit.jupiter.api.function.Executable;

Step 7.2: Writing a code to demonstrate JUnit @TestFactory

```
*DynamicTest.java  ClearBlue/pom.xml
1 package Demo.ClearBlue;
2
3 import static org.junit.jupiter.api.Assertions.assertTrue;
4 import static org.junit.jupiter.api.DynamicTest.dynamicTest;
5 import java.util.Arrays;
6 import java.util.Collection;
7 import org.junit.jupiter.api.DynamicTest;
8 import org.junit.jupiter.api.TestFactory;
9 import org.junit.jupiter.api.function.Executable;
10 public class DynamicTest {
11
12     @TestFactory
13     Collection<DynamicTest> dynamicTests() {
14         return Arrays.asList(
15             dynamicTest("simple test", () -> assertTrue(true)),
16             dynamicTest("Executable Class", new MyExecutable()),
17             dynamicTest("Exception Executable", () -> {throw new Exception("Exception Example");}),
18             dynamicTest("simple test-2", () -> assertTrue(true))
19         );
20     }
21
22 }
23 class MyExecutable implements Executable {
24     @Override
25     public void execute() throws Throwable {
26         System.out.println("Dynamic test");
27     }
28 }
29
30
31
```



Step 7.3: Writing a code to demonstrate the execution of Dynamic Tests

- Screenshot 1:

```
"DynamicTest.java" "ClearBlue/pom.xml"
1 package Demo.ClearBlue;
2 import static org.junit.jupiter.api.DynamicTest.dynamicTest;
3 import java.util.ArrayList;
4 import java.util.Arrays;
5 import java.util.List;
6 import java.util.stream.Stream;
7 import org.junit.jupiter.api.DynamicTest;
8 import org.junit.jupiter.api.TestFactory;
9 import org.junit.jupiter.api.function.Executable;
10 public class DynamicTest {
11
```

• Screenshot 2:

```
"DynamicTest.java" "ClearBlue/pom.xml"
10 public class DynamicTest {
11
12     public static class MyUtils {
13         public static int add(int x, int y) {
14             return x+y;
15         }
16     }
17     @TestFactory
18     Stream<DynamicTest> dynamicTestsEx() {
19         List<Integer> inputList1 = Arrays.asList(1,2,3);
20         List<Integer> inputList2 = Arrays.asList(10,20,30);
21         List<DynamicTest> dynamicTests = new ArrayList<>();
22         for(int i=0; i < inputList1.size(); i++) {
23             int x = inputList1.get(i);
24             int y = inputList2.get(i);
25             DynamicTest dynamicTest = dynamicTest("Test for MyUtils.add(\"+x+\",\"+y+\"", () ->{inputVariables(x+y,MyUtils.add(x,y));});
26             dynamicTests.add(dynamicTest);
27         }
28         return dynamicTests.stream();
29     }
30     private void inputVariables(int i, int add) {
31     }
32 }
33 class MyExecutable implements Executable {
34     @Override
35     public void execute() throws Throwable {
36         System.out.println("Dynamic test");
37     }
38 }
39 }
40
41
42
```

```
Console JUnit
Finished after 0.313 seconds
Runs: 3/3 Errors: 0 Failures: 0
DynamicTest [Runner: JUnit 5] (0.000 s)
  dynamicTestsEx() (0.000 s)
    Test for MyUtils.add(1,10) (0.000 s)
    Test for MyUtils.add(2,20) (0.000 s)
    Test for MyUtils.add(3,30) (0.032 s)
```

8. Parameterized Tests

Step 8.1: Writing a code to demonstrate Parameterized Test with @ValueSource

```
1 package SSS.Junit;  
2  
3 import static org.junit.Assert.assertTrue;  
4  
5 import org.junit.jupiter.params.ParameterizedTest;  
6 import org.junit.jupiter.params.provider.ValueSource;  
7  
8 public class Demo {  
9  
10  
11 @ParameterizedTest  
12 @ValueSource(ints = { 4, 5, 6 })  
13 void test_ValueSource(int i) {  
14     System.out.println(i);  
15 }  
16  
17 @ParameterizedTest  
18 @ValueSource(strings = { "4", "5", "6" })  
19 void test_ValueSource_String(String s) {  
20     assertTrue(Integer.parseInt(s) < 5);  
21 }
```

```
4  
5  
6
```

Finished after 0.238 seconds

Runs: 6/6 Errors: 0 Failures: 2

The screenshot shows the JUnit 5 test results in an IDE. At the top, a summary bar indicates 'Runs: 6/6', 'Errors: 0', and 'Failures: 2'. Below this, a tree view expands the 'Demo' class. It shows two test methods: 'test_ValueSource_String(String)' and 'test_ValueSource(int)'. Each method has three parameterized test cases. The 'test_ValueSource_String' method has failures for the first two cases (4 and 5) and a success for the third (6). The 'test_ValueSource(int)' method has successes for all three cases (4, 5, and 6). Each test case is represented by a small icon (a green checkmark for success or a red X for failure) followed by the test name, the parameter value, and the execution time in seconds.

- ▼ Demo [Runner: JUnit 5] (0.071 s)
 - ▼ test_ValueSource_String(String) (0.045 s)
 - [1] 4 (0.045 s) [Failure]
 - [2] 5 (0.007 s) [Failure]
 - [3] 6 (0.004 s) [Success]
 - ▼ test_ValueSource(int) (0.005 s)
 - [1] 4 (0.005 s) [Success]
 - [2] 5 (0.006 s) [Success]
 - [3] 6 (0.008 s) [Success]



Step 8.2: Writing a code to demonstrate @ParameterizedTest with @EnumSource


```





1 package SSS.Junit;
2
3 import static org.junit.Assert.assertTrue;
4
5 import java.lang.annotation.ElementType;
6 import java.util.EnumSet;
7
8 import org.junit.jupiter.params.ParameterizedTest;
9 import org.junit.jupiter.params.provider.EnumSource;
10
11 public class Demo {
12
13     @ParameterizedTest
14     @EnumSource(value = ElementType.class, names = { "TYPE", "METHOD", "FIELD" })
15     void test_EnumSource_Extended(ElementType et) {
16         assertTrue(EnumSet.of(ElementType.FIELD, ElementType.TYPE, ElementType.METHOD).contains(et));
17     }
18 }
19

```

Finished after 0.209 seconds

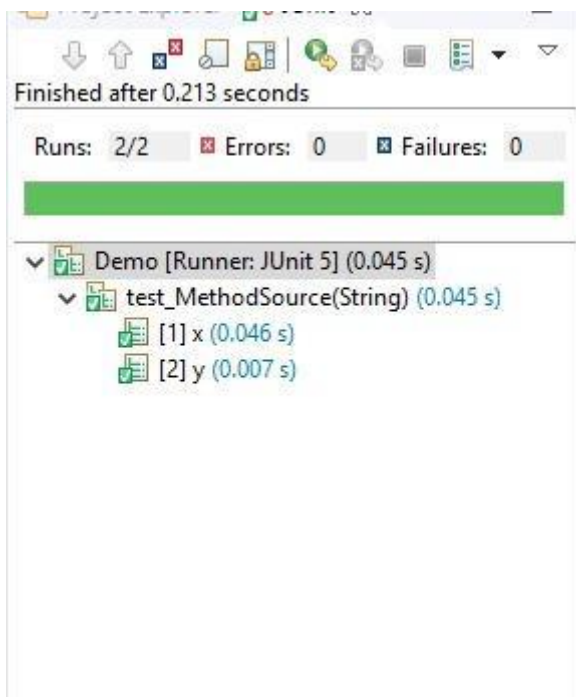
Runs: 3/3  Errors: 0  Failures: 0

 Demo [Runner: JUnit 5] (0.049 s)

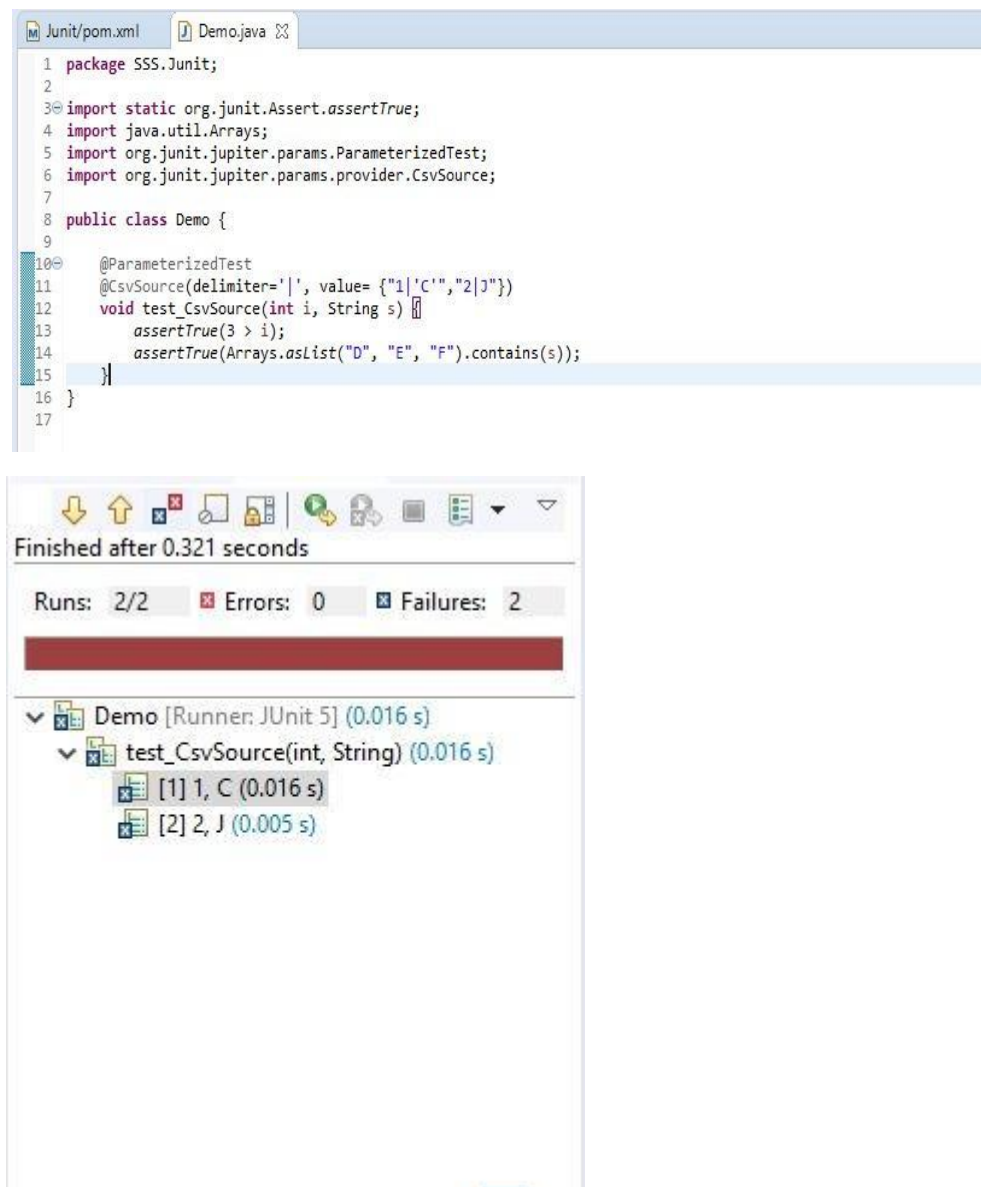
-  test_EnumSource_Extended(ElementType)
 -  [1] TYPE (0.049 s)
 -  [2] FIELD (0.004 s)
 -  [3] METHOD (0.006 s)

Step 8.3: Writing a code to demonstrate @ParameterizedTest with @MethodSource

```
1 package SSS.Junit;  
2  
3 import static org.junit.Assert.assertNotNull;  
4 import java.util.stream.Stream;  
5  
6 import org.junit.jupiter.params.ParameterizedTest;  
7 import org.junit.jupiter.params.provider.MethodSource;  
8  
9 public class Demo {  
10  
11     @ParameterizedTest  
12     @MethodSource("Dp")  
13     void test_MethodSource(String s) {  
14         assertNotNull(s);  
15     }  
16  
17     static Stream<String> Dp() {  
18         return Stream.of("x", "y");  
19     }  
20 }  
21
```



Step 8.4: Writing a code to demonstrate @ParameterizedTest with @CsvSource



The screenshot displays an IDE with two panels. The top panel shows the source code for `Demo.java`, and the bottom panel shows the test execution results.

Source Code (Demo.java):

```
1 package SSS.Junit;  
2  
3 import static org.junit.Assert.assertTrue;  
4 import java.util.Arrays;  
5 import org.junit.jupiter.params.ParameterizedTest;  
6 import org.junit.jupiter.params.provider.CsvSource;  
7  
8 public class Demo {  
9  
10     @ParameterizedTest  
11     @CsvSource(delimiter='|', value= {"1|C","2|J"})  
12     void test_CsvSource(int i, String s) {  
13         assertTrue(3 > i);  
14         assertTrue(Arrays.asList("D", "E", "F").contains(s));  
15     }  
16 }  
17
```

Test Results:

Finished after 0.321 seconds

Runs: 2/2 Errors: 0 Failures: 2

▼ Demo [Runner: JUnit 5] (0.016 s)

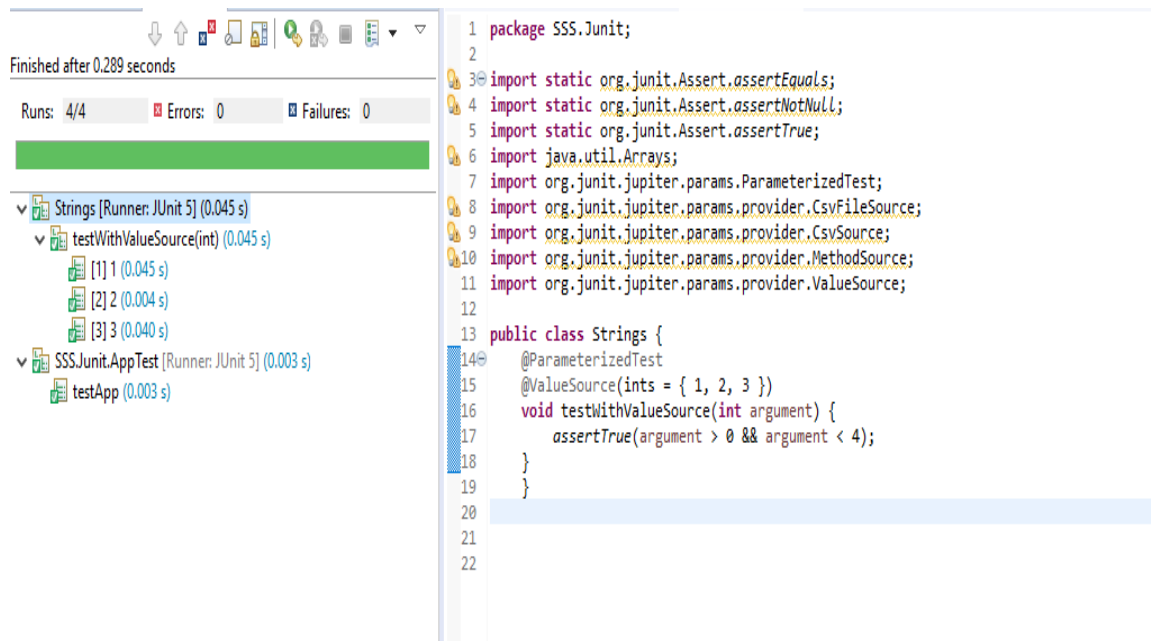
▼ test_CsvSource(int, String) (0.016 s)

- ▼ [1] 1, C (0.016 s)
- ▼ [2] 2, J (0.005 s)

9. Argument Sources

Step 9.1: Writing a code to demonstrate @ValueSource annotation

With the @ValueSource annotation, we can pass an array of literal values to the test method.



```
1 package SSS.Junit;  
2  
3 import static org.junit.Assert.assertEquals;  
4 import static org.junit.Assert.assertNotNull;  
5 import static org.junit.Assert.assertTrue;  
6 import java.util.Arrays;  
7 import org.junit.jupiter.params.ParameterizedTest;  
8 import org.junit.jupiter.params.provider.CsvFileSource;  
9 import org.junit.jupiter.params.provider.CsvSource;  
10 import org.junit.jupiter.params.provider.MethodSource;  
11 import org.junit.jupiter.params.provider.ValueSource;  
12  
13 public class Strings {  
14     @ParameterizedTest  
15     @ValueSource(ints = { 1, 2, 3 })  
16     void testWithValueSource(int argument) {  
17         assertTrue(argument > 0 && argument < 4);  
18     }  
19 }  
20  
21  
22
```

Step 9.2: Writing a code to demonstrate @CsvSource annotation

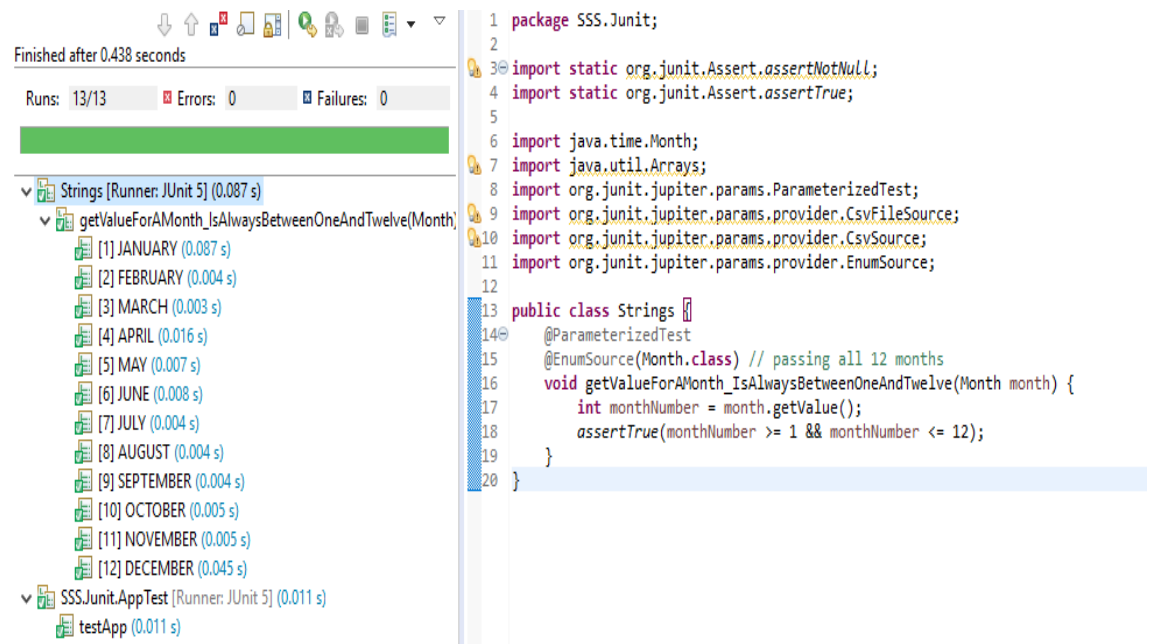
Suppose we're going to make sure that the toUpperCase() method from String generates the expected uppercase value. @ValueSource won't be enough.



```
1 package SSS.Junit;  
2  
3 import static org.junit.Assert.assertEquals;  
4 import static org.junit.Assert.assertNotNull;  
5 import static org.junit.Assert.assertTrue;  
6 import java.util.Arrays;  
7 import org.junit.jupiter.params.ParameterizedTest;  
8 import org.junit.jupiter.params.provider.CsvFileSource;  
9 import org.junit.jupiter.params.provider.CsvSource;  
10  
11 public class Strings {  
12     @ParameterizedTest  
13     @CsvSource({"test,TEST", "tEst,TEST", "Java,JAVA"})  
14     void toUpperCase_ShouldGenerateTheExpectedUppercaseValue(String input, String expected) {  
15         String actualValue = input.toUpperCase();  
16         assertEquals(expected, actualValue);  
17     }  
18 }  
19  
20
```

Step 9.3: Writing a code to demonstrate @EnumSource annotation

In order to run a test with different values from an enumeration, we can use the @EnumSource annotation.



The screenshot shows an IDE with two panels. The left panel displays test results for a JUnit 5 test suite. The right panel shows the corresponding Java source code.

Test Results (Left Panel):

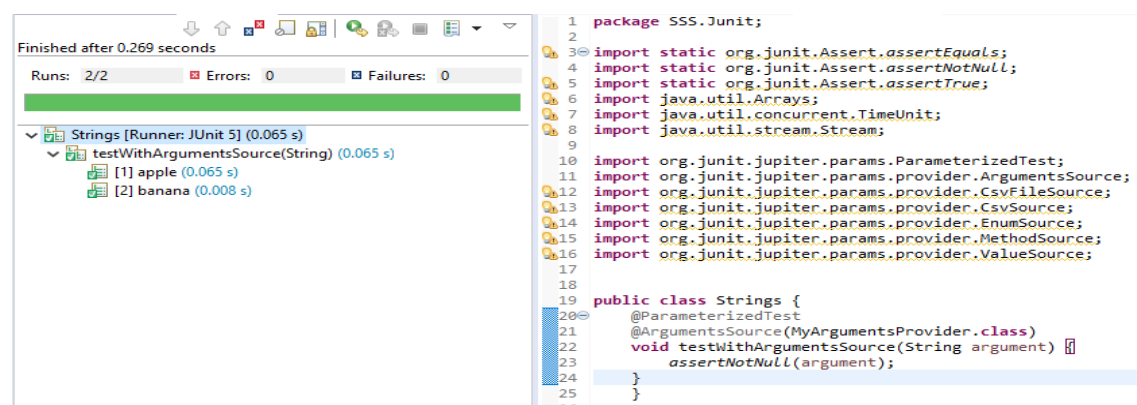
- Finished after 0.438 seconds
- Runs: 13/13, Errors: 0, Failures: 0
- Test Suite: Strings [Runner: JUnit 5] (0.087 s)
 - Method: getValueForAMonth_IsAlwaysBetweenOneAndTwelve(Month)
 - Parameters: [1] JANUARY (0.087 s), [2] FEBRUARY (0.004 s), [3] MARCH (0.003 s), [4] APRIL (0.016 s), [5] MAY (0.007 s), [6] JUNE (0.008 s), [7] JULY (0.004 s), [8] AUGUST (0.004 s), [9] SEPTEMBER (0.004 s), [10] OCTOBER (0.005 s), [11] NOVEMBER (0.005 s), [12] DECEMBER (0.045 s)
- Test Suite: SSS.Junit.AppTest [Runner: JUnit 5] (0.011 s)
 - Method: testApp (0.011 s)

Source Code (Right Panel):

```
1 package SSS.Junit;
2
3 import static org.junit.Assert.assertNotNull;
4 import static org.junit.Assert.assertTrue;
5
6 import java.time.Month;
7 import java.util.Arrays;
8 import org.junit.jupiter.params.ParameterizedTest;
9 import org.junit.jupiter.params.provider.CsvFileSource;
10 import org.junit.jupiter.params.provider.CsvSource;
11 import org.junit.jupiter.params.provider.EnumSource;
12
13 public class Strings {
14     @ParameterizedTest
15     @EnumSource(Month.class) // passing all 12 months
16     void getValueForAMonth_IsAlwaysBetweenOneAndTwelve(Month month) {
17         int monthNumber = month.getValue();
18         assertTrue(monthNumber >= 1 && monthNumber <= 12);
19     }
20 }
```

Step 9.4: Writing a code to demonstrate @ArgumentsSource annotation

@ArgumentsSource can be used to specify a custom, reusable ArgumentsProvider. Note that an implementation of ArgumentsProvider must be declared as either a top-level class or as a static nested class.



The screenshot shows an IDE with two panels. The left panel displays test results for a JUnit 5 test suite. The right panel shows the corresponding Java source code.

Test Results (Left Panel):

- Finished after 0.269 seconds
- Runs: 2/2, Errors: 0, Failures: 0
- Test Suite: Strings [Runner: JUnit 5] (0.065 s)
 - Method: testWithArgumentsSource(String) (0.065 s)
 - Parameters: [1] apple (0.065 s), [2] banana (0.008 s)

Source Code (Right Panel):

```
1 package SSS.Junit;
2
3 import static org.junit.Assert.assertEquals;
4 import static org.junit.Assert.assertNotNull;
5 import static org.junit.Assert.assertTrue;
6 import java.util.Arrays;
7 import java.util.concurrent.TimeUnit;
8 import java.util.stream.Stream;
9
10 import org.junit.jupiter.params.ParameterizedTest;
11 import org.junit.jupiter.params.provider.ArgumentsSource;
12 import org.junit.jupiter.params.provider.CsvFileSource;
13 import org.junit.jupiter.params.provider.CsvSource;
14 import org.junit.jupiter.params.provider.EnumSource;
15 import org.junit.jupiter.params.provider.MethodSource;
16 import org.junit.jupiter.params.provider.ValueSource;
17
18
19 public class Strings {
20     @ParameterizedTest
21     @ArgumentsSource(MyArgumentsProvider.class)
22     void testWithArgumentsSource(String argument) {
23         assertNotNull(argument);
24     }
25 }
26 }
```

10. Argument Conversion

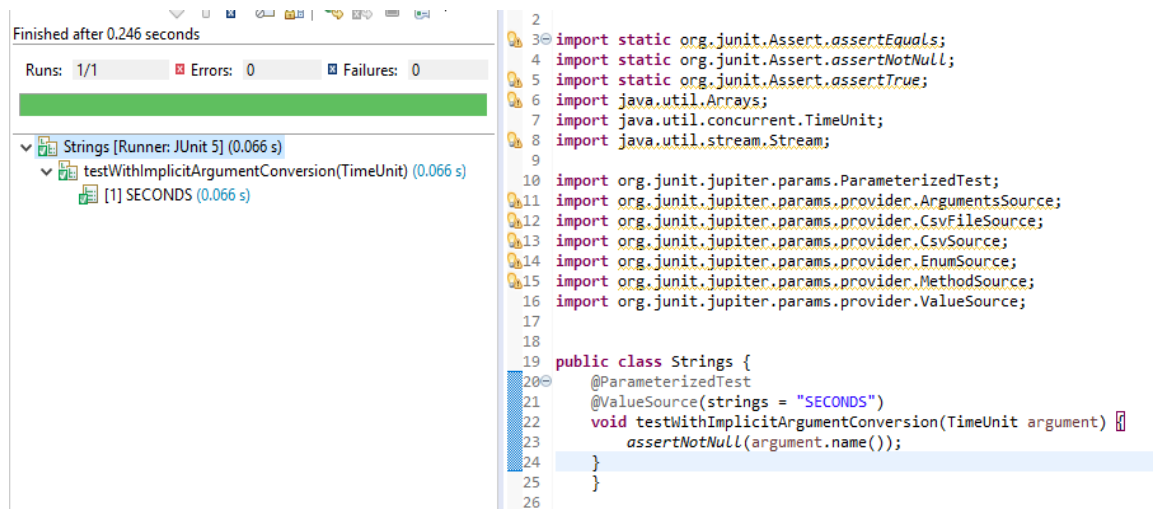
Step 10.1: Writing a code to demonstrate Implicit Conversion

To support use cases like `@CsvSource`, JUnit Jupiter provides a number of built-in implicit type converters. The conversion process depends on the declared type of each method parameter.

Create a class with test with `ToStringArgumentConverter.class`

`@Override`

```
protected Object convert(Object source, Class<?> targetType) {  
    assertEquals(String.class, targetType, "Can only convert to String");  
    return String.valueOf(source);  
}  
}
```

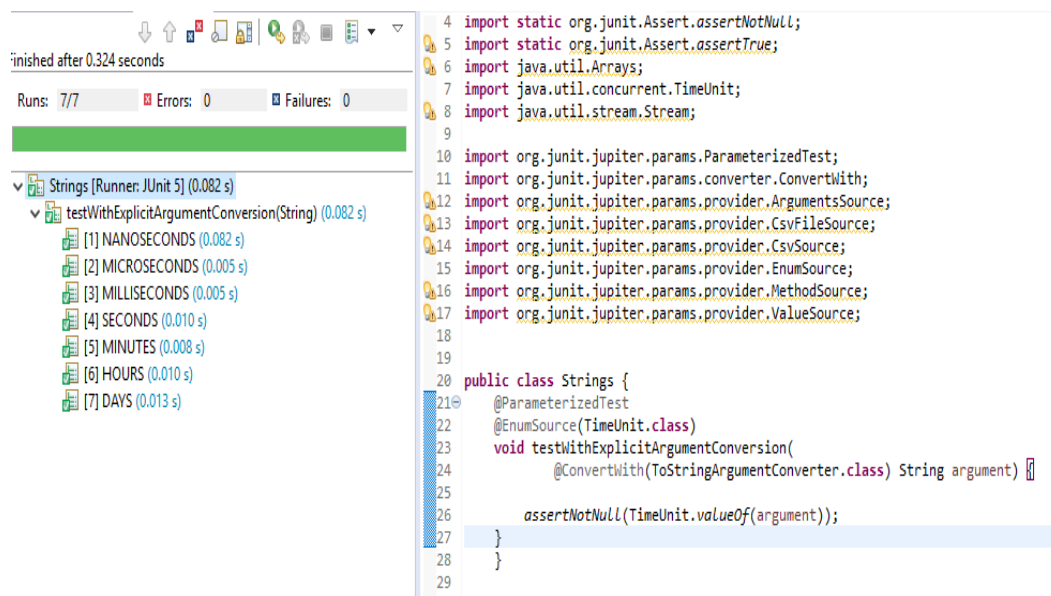


Step 10.2: Writing a code to demonstrate Explicit Conversion

Instead of relying on implicit argument conversion, you may explicitly specify an `ArgumentConverter` to use for a certain parameter using the `@ConvertWith` annotation as shown in the following example. Note that an implementation of `ArgumentConverter` must be declared as either a top-level class or a static nested class.

Create a class "ToStringArgumentConverter.class" with below code

```
public class ToStringArgumentConverter extends SimpleArgumentConverter
{
    @Override
    protected Object convert(Object source, Class<?> targetType)
    {
        assertEquals(String.class, targetType, "Can only convert to String");
        return String.valueOf(source);
    }
}
```



11. Extension Points

Steps 11.1: Writing a code to demonstrate Life Cycle Call Back

```
1 package Demo.ClearBlue;
2
3 import org.junit.jupiter.api.*;
4 public class extentionPoint {
5     public static class DefaultTestLifeCycleTest {
6         @BeforeAll
7         static void beforeAll() {
8             System.out.println("beforeAll()");
9         }
10        @BeforeEach
11        void beforeEach() {
12            System.out.println("beforeEach()");
13        }
14        @Test
15        void Test1() {
16            System.out.println("Test1()");
17        }
18        @Test
19        void Test2() {
20            System.out.println("Test2()");
21        }
22        @AfterEach
23        void afterBoth2() {
24            System.out.println("afterboth2()");
25        }
26        @AfterAll
27        static void afterAll() {
28            System.out.println("afterAll()");
29        }
30    }
31 }
32 }
33
```

```
beforeAll()
beforeEach()
Test1()
afterboth2()
beforeEach()
Test2()
afterboth2()
afterAll()
```

Steps 11.2: Writing code to demonstrate Conditional Test Execution

- Create the extension class and extend them in the base class.
- Create an extension as shown below:

```

1 package extensions;
2 import org.junit.jupiter.api.extension.ConditionEvaluationResult;
3 import org.junit.jupiter.api.extension.ExecutionCondition;
4 import org.junit.jupiter.api.extension.ExtensionContext;
5
6 import java.io.FileInputStream;
7 import java.io.IOException;
8 import java.util.Properties;
9 public class ExtensionPackageClass implements ExecutionCondition{
10     private static String propertyFilePath = System.getProperty("user.dir")+
11         "/src/test/java/test.properties";
12
13     @Override
14     public ConditionEvaluationResult evaluateExecutionCondition(ExtensionContext context) {
15
16         Properties prop = new Properties();
17         try {
18             prop.load(new FileInputStream(propertyFilePath));
19         } catch (IOException e) {
20             e.printStackTrace();
21         }
22
23         String environment = prop.getProperty("environment");
24         if (environment.equals("development")) {
25             return ConditionEvaluationResult.disabled("Test disabled on Development environment.");
26         }
27
28         return ConditionEvaluationResult.enabled("Test enabled on Test environment");
29     }
30 }
31
32

```

- Now

```

1
2 package Demo.ClearBlue;
3
4 import org.junit.jupiter.api.extension.ExtensionContext;
5 import org.junit.jupiter.api.extension.ParameterContext;
6 import org.junit.jupiter.api.extension.ParameterResolutionException;
7 import org.junit.jupiter.api.extension.ParameterResolver;
8
9 public abstract class FraudServiceParameterResolver implements ParameterResolver {
10     @Override
11     public Object resolveParameter(ParameterContext parameterContext,
12         ExtensionContext extensionContext) throws ParameterResolutionException {
13         return new FraudService();
14     }
15
16     @Override
17     public boolean supportsParameter(ParameterContext parameterContext,
18         ExtensionContext extensionContext) throws ParameterResolutionException {
19         return (parameterContext.getParameter().getType() == FraudService.class);
20     }
21 }

```

```

1 package Demo.ClearBlue;
2
3 import org.junit.jupiter.api.*;
4 import org.junit.jupiter.api.extension.ExtendWith;
5
6 import extensions.ExtentionPackageClass;
7 @ExtendWith(ExtentionPackageClass.class)
8
9 public class extentionPoint {
10     public static class TestLifecycleTest {
11         @BeforeAll
12         static void beforeAll() {
13             System.out.println("beforeAll()");
14         }
15         @BeforeEach
16         void beforeEach() {
17             System.out.println("beforeEach()");
18         }
19         @Test
20         void Test1() {
21             System.out.println("Test1()");
22         }
23         @Test
24         void Test2() {
25             System.out.println("Test2()");
26         }
27         @AfterEach
28         void aftertest() {
29             System.out.println("aftertest()");
30         }
31         @AfterAll
32         static void afterAll() {
33             System.out.println("afterAll()");

```

```

Test disabled on Development environment.

Test disabled on Development environment.

Process finished with exit code 0

```

- We should register this with `@ExtendWith(FraudServiceParameterResolver.class)` annotation on top of our test classes.

Steps 11.3: Writing a code to demonstrate Exception Handling Extension

- Create an extension class as shown below:

```

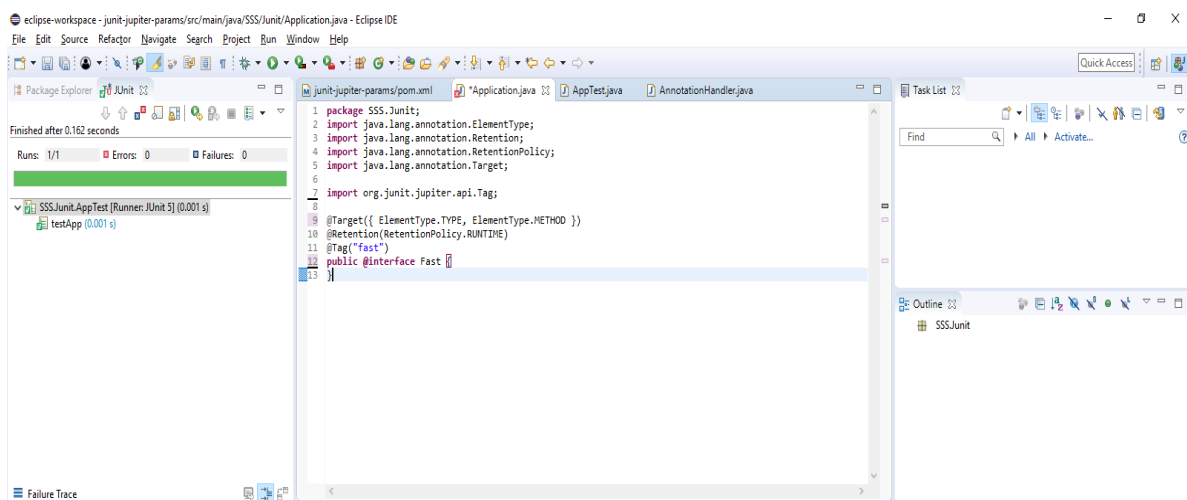
1 import org.junit.jupiter.api.extension.ExtensionContext;
2 import org.junit.jupiter.api.extension.TestExecutionExceptionHandler;
3
4 public abstract class IgnoreStaleElementReferenceExceptionExtension implements TestExecutionExceptionHandler {
5     @Override
6     public void handleTestExecutionException(ExtensionContext context, Throwable throwable)
7         throws Throwable {
8         if (throwable instanceof StaleElementReferenceException) {
9             return;
10        }
11        throw throwable;
12    }
13 }
14
15
16

```

12. Meta-Annotations

Step 12.1: Writing a code to demonstrate Meta-Annotations and Composed Annotations

- JUnit Jupiter annotations can be used as meta-annotations. That means that you can define your own composed annotation that will automatically inherit the semantics of its meta-annotations.
- Instead of copying and pasting `@Tag("fast")` throughout your code base (see Tagging and Filtering), you can create a custom composed annotation named `@Fast` as follows. `@Fast` can then be used as a drop-in replacement for `@Tag("fast")`.



- The following `@Test` method demonstrates the usage of the `@Fast` annotation.

```
@Fast
```

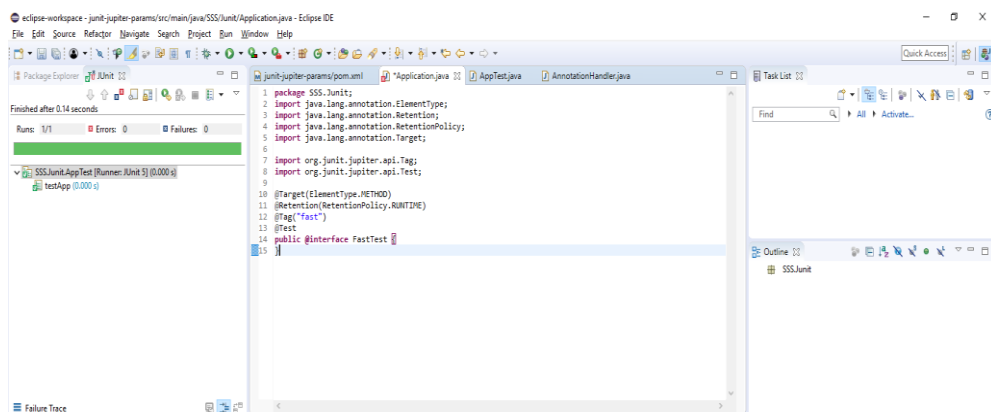
```
@Test
```

```
void myFastTest() {

    // ...

}
```

- You can even take that one step further by introducing a custom `@FastTest` annotation that can be used as a drop-in replacement for `@Tag("fast")` and `@Test`.



- JUnit automatically recognizes the following as a `@Test` method that is tagged with "fast."

```
@FastTest
```

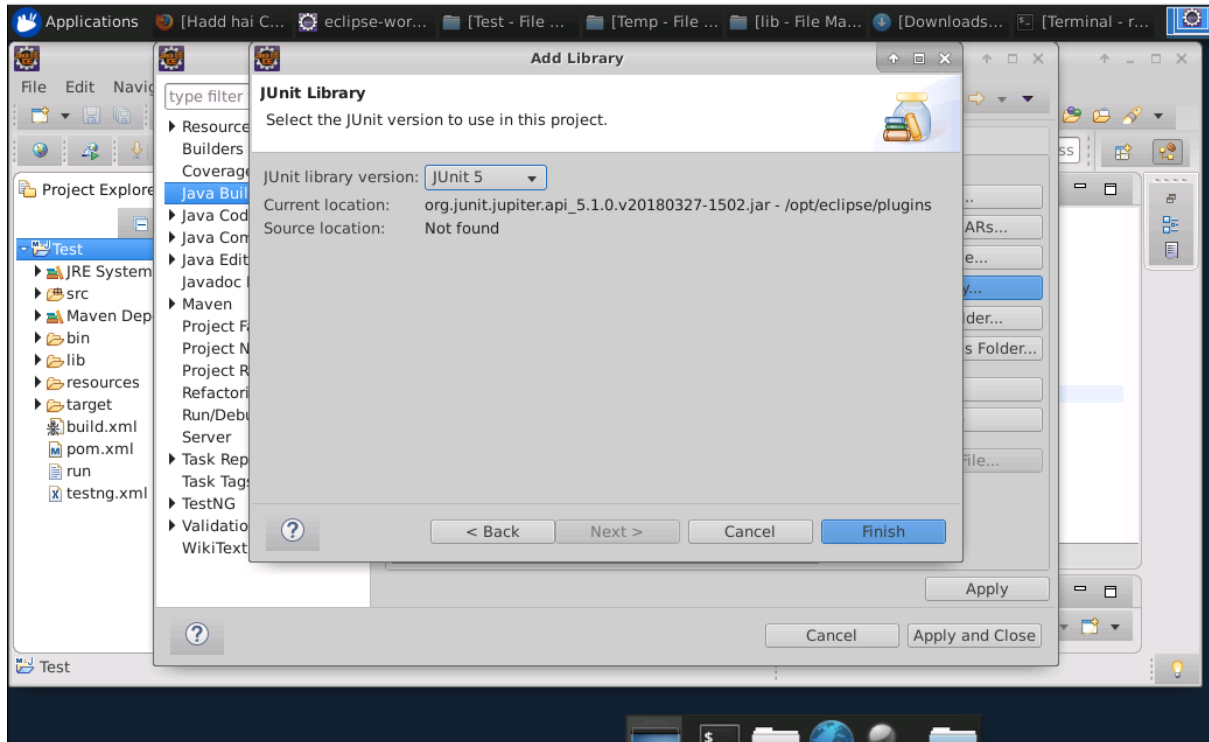
```
void myFastTest() {

    // ...

}
```

13. Running Test from the Console

Step 13.1: Configuring the JUnit-4 version from the eclipse build path to add libraries



- 1) Make sure there is no error in the script.
- 2) Open the command prompt and set the path folder where you saved the JavaScript file.

- 3) To compile the Java file, write the command given below:

```
javac -cp junit-4.12.0.jar;. Sample.java
```

- 4) Run the Junit java file in the command prompt.

Type the command: `javac -cp junit-4.12.jar;. UserDAOTest.java`

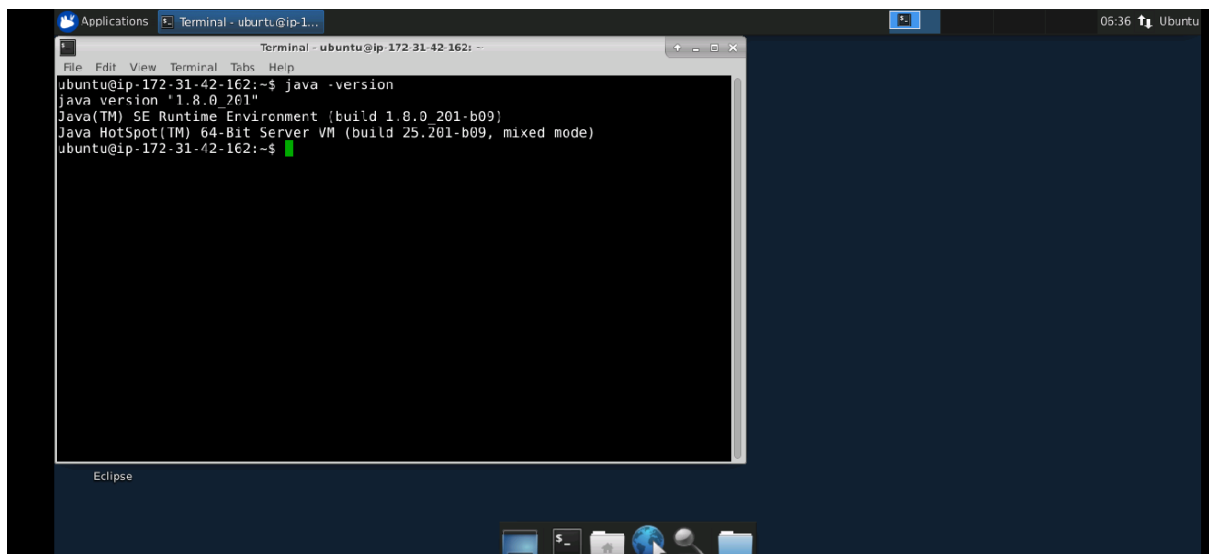
Type the command: `java -cp junit-4.12.jar;hamcrest-core-1.3.jar;. org.junit.runner.JUnitCore UserDAOTest JUnit version 4.12`

14. Running Tests with Gradle

Step 14.1: Installing Gradle from the official website

- Gradle requires Java development kit in order to work. Java has already been installed in your practice lab. To verify the installation, run the following command:

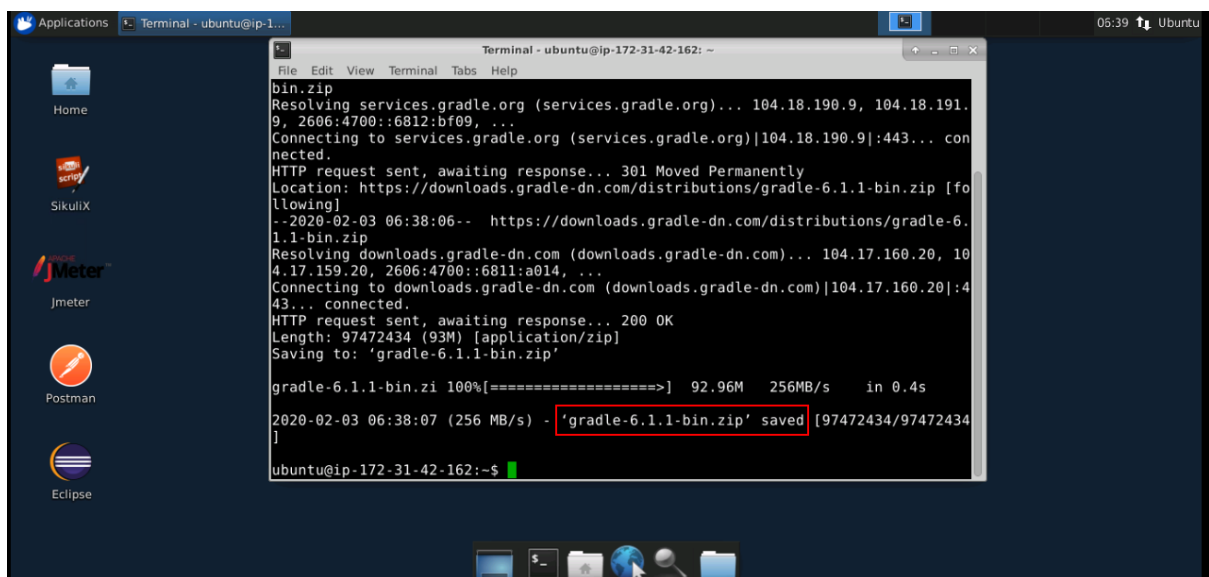
java -version

A terminal window titled 'Terminal - ubuntu@ip-172-31-42-162: ~' is open. The command 'java -version' has been executed, and the output is displayed: 'java version "1.8.0_261"', 'Java(TM) SE Runtime Environment (build 1.8.0_201-b09)', and 'Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)'. The terminal is running on an Ubuntu desktop environment with a taskbar at the bottom showing icons for applications, terminal, and other tools.

```
ubuntu@ip-172-31-42-162:~$ java -version
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_201-b09)
Java HotSpot(TM) 64-Bit Server VM (build 25.201-b09, mixed mode)
ubuntu@ip-172-31-42-162:~$
```

- Download Gradle using the following command:

wget <https://services.gradle.org/distributions/gradle-6.1.1-bin.zip>

A terminal window titled 'Terminal - ubuntu@ip-172-31-42-162: ~' is open. The command 'wget https://services.gradle.org/distributions/gradle-6.1.1-bin.zip' has been executed. The output shows the download progress: 'Resolving services.gradle.org (services.gradle.org)... 104.18.190.9, 104.18.191.9, 2606:4700::6812:bf09, ...', 'Connecting to services.gradle.org (services.gradle.org)|104.18.190.9|:443... connected.', 'HTTP request sent, awaiting response... 301 Moved Permanently', 'Location: https://downloads.gradle-dn.com/distributions/gradle-6.1.1-bin.zip [following]', '--2020-02-03 06:38:06-- https://downloads.gradle-dn.com/distributions/gradle-6.1.1-bin.zip', 'Resolving downloads.gradle-dn.com (downloads.gradle-dn.com)... 104.17.160.20, 104.17.159.20, 2606:4700::6811:a014, ...', 'Connecting to downloads.gradle-dn.com (downloads.gradle-dn.com)|104.17.160.20|:443... connected.', 'HTTP request sent, awaiting response... 200 OK', 'Length: 97472434 (93M) [application/zip]', 'Saving to: 'gradle-6.1.1-bin.zip'', 'gradle-6.1.1-bin.zip 100%[=====] 92.96M 256MB/s in 0.4s', '2020-02-03 06:38:07 (256 MB/s) - 'gradle-6.1.1-bin.zip' saved [97472434/97472434]', 'ubuntu@ip-172-31-42-162:~\$'. The terminal is running on an Ubuntu desktop environment with a taskbar at the bottom showing icons for applications, terminal, and other tools.

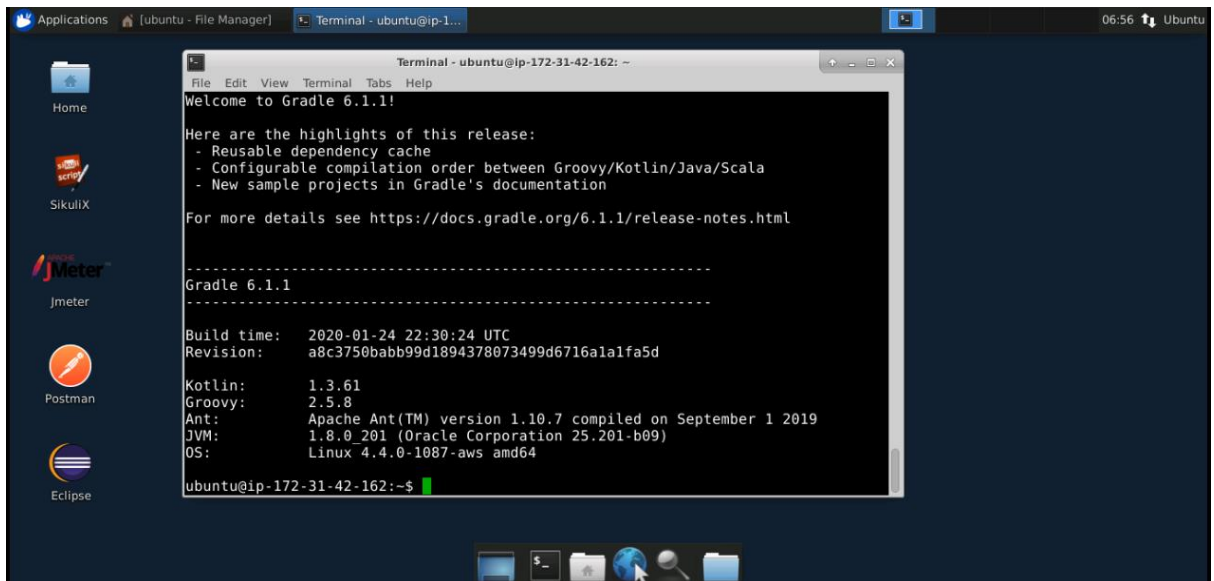
```
bin.zip
Resolving services.gradle.org (services.gradle.org)... 104.18.190.9, 104.18.191.9, 2606:4700::6812:bf09, ...
Connecting to services.gradle.org (services.gradle.org)|104.18.190.9|:443... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://downloads.gradle-dn.com/distributions/gradle-6.1.1-bin.zip [following]
--2020-02-03 06:38:06-- https://downloads.gradle-dn.com/distributions/gradle-6.1.1-bin.zip
Resolving downloads.gradle-dn.com (downloads.gradle-dn.com)... 104.17.160.20, 104.17.159.20, 2606:4700::6811:a014, ...
Connecting to downloads.gradle-dn.com (downloads.gradle-dn.com)|104.17.160.20|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 97472434 (93M) [application/zip]
Saving to: 'gradle-6.1.1-bin.zip'

gradle-6.1.1-bin.zip 100%[=====] 92.96M 256MB/s in 0.4s
2020-02-03 06:38:07 (256 MB/s) - 'gradle-6.1.1-bin.zip' saved [97472434/97472434]
ubuntu@ip-172-31-42-162:~$
```

- After downloading, create a directory for Gradle installation and unzip the downloaded file in the new directory.
- Configure the PATH environment variable using the following command:
`export PATH=$PATH:gradle/gradle-6.1.1/bin`

Step 14.2: Checking the Installed version

- Verify the installation using the following command:
`gradle -v`



```

Terminal - ubuntu@ip-172-31-42-162: ~
File Edit View Terminal Tabs Help
Welcome to Gradle 6.1.1!

Here are the highlights of this release:
- Reusable dependency cache
- Configurable compilation order between Groovy/Kotlin/Java/Scala
- New sample projects in Gradle's documentation

For more details see https://docs.gradle.org/6.1.1/release-notes.html

-----
Gradle 6.1.1
-----

Build time:   2020-01-24 22:30:24 UTC
Revision:     a8c3750babb99d1894378073499d6716a1a1fa5d

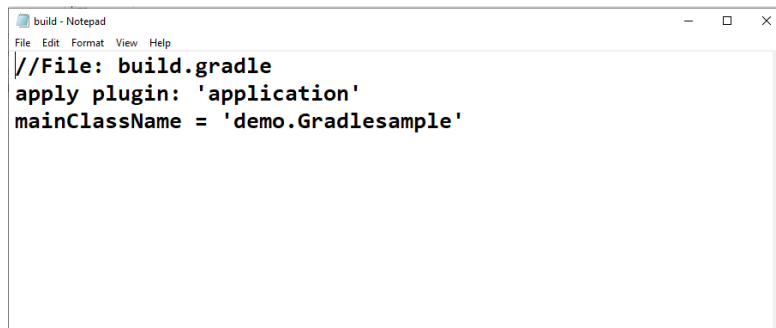
Kotlin:       1.3.61
Groovy:       2.5.8
Ant:          Apache Ant(TM) version 1.10.7 compiled on September 1 2019
JVM:          1.8.0_201 (Oracle Corporation 25.201-b09)
OS:           Linux 4.4.0-1087-aws amd64

ubuntu@ip-172-31-42-162:~$

```

Step 14.3: Scripting the Gradle built program

- Create a new file, type the script given below, and save it as “build.gradle” outside the Gradle folder.



```

build - Notepad
File Edit Format View Help
//File: build.gradle
apply plugin: 'application'
mainClassName = 'demo.Gradlesample'

```

- Go to the command prompt and type the following:
 - gradle tasks

```

> Task :tasks
-----
Tasks runnable from root project
-----

Build Setup tasks
-----
init - Initializes a new Gradle build.
wrapper - Generates Gradle wrapper files.

Help tasks
-----
buildEnvironment - Displays all buildscript dependencies declared in root project 'GradleDemo'.
components - Displays the components produced by root project 'GradleDemo'. [incubating]
dependencies - Displays all dependencies declared in root project 'GradleDemo'.
dependencyInsight - Displays the insight into a specific dependency in root project 'GradleDemo'.
dependencyComponents - Displays the dependent components of components in root project 'GradleDemo'. [incubating]
help - Displays a help message.
model - Displays the configuration model of root project 'GradleDemo'. [incubating]
projects - Displays the sub-projects of root project 'GradleDemo'.
properties - Displays the properties of root project 'GradleDemo'.
tasks - Displays the tasks runnable from root project 'GradleDemo'.

To see all tasks and more detail, run gradle tasks --all

To see more detail about a task, run gradle help --task <task>

BUILD SUCCESSFUL in 1s
1 actionable task: 1 executed

```

- gradle build

```

BUILD SUCCESSFUL in 2s
1 actionable task: 1 executed

```

- That will automatically generate the build folder file.

Step 14.4: Checking Final Gradle Output

- Run the Java program using the gradle command line prompt.
- Type gradle build
- Output: "Hello welcome to gradle project !!"

```

BUILD SUCCESSFUL in 2s
5 actionable tasks: 3 executed, 2 up-to-date

```

```

> Task :run
Hello welcome to gradle project!!

BUILD SUCCESSFUL in 1s
2 actionable tasks: 1 executed, 1 up-to-date

```

15. Running Tests with Maven

Step 15.1: Writing a code to demonstrate Maven-surefire-plugin

Below is the Surefire Plugin. This Plugin (code below) needs to be added to the Page Object Model “POM” file in xml format.

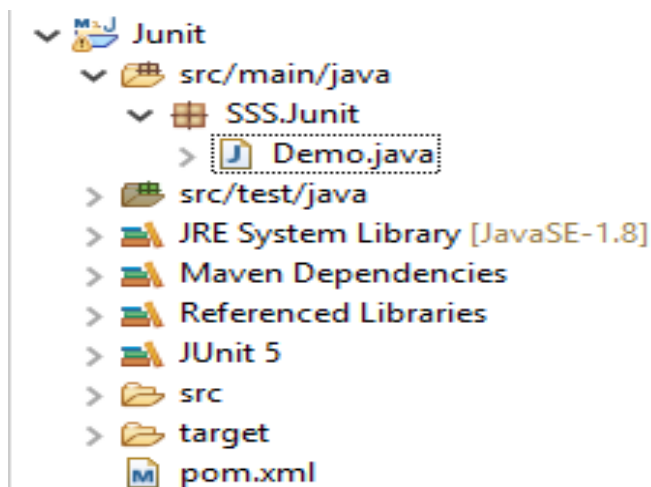
- Add the below Surefire to POM

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-surefire-plugin</artifactId>
      <version>2.22.0</version>
    </plugin>
  </plugins>
</build>
```

Step 15.2: Explaining the structure of a Maven project

- Below is a simple Java project which will guide us on how to run the unit test classes in the Maven project. This is the directory structure for creating a Maven project which essentially includes the Junit test cases, JRE system library, maven dependencies, Junit 5 library, and the POM XML file.

Directory Structure



Step 15.3: Demonstrating Maven + JUnit5 examples

- Below is the sample POM file with all the dependencies and other annotations:

```
<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>

  <artifactId>junit-jupiter-params</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>JUnit</name>
  <url>http://maven.apache.org</url>

  <properties>
    <maven.compiler.source>1.9</maven.compiler.source>
    <maven.compiler.target>1.9</maven.compiler.target>
  </properties>
  <build>
    <plugins>

      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.22.0</version>
      </plugin>
    </plugins>
  </build>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-api -->
    <dependency>
      <groupId>org.junit.jupiter</groupId>
      <artifactId>junit-jupiter-api</artifactId>
      <version>5.2.0</version>
      <scope>test</scope>
    </dependency>
```

```

<!-- https://mvnrepository.com/artifact/org.junit.jupiter/junit-jupiter-engine -->
<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-engine</artifactId>
  <version>5.2.0</version>
  <scope>test</scope>
</dependency>

<dependency>
  <groupId>org.junit.jupiter</groupId>
  <artifactId>junit-jupiter-params</artifactId>
  <version>5.0.0</version>
  <scope>test</scope>
</dependency>

</dependencies>
<groupId>org.junit.jupiter</groupId>
</project>

```

Step 15.4: Demonstrating Java Classes to run the Maven Script

- We have already created a complete Maven project structure above with Java source code. Now we will create different Java classes in the `./src/main/java/packages/class(s)`. It also created an example test class in `./src/test/`. In the root folder, there is a `pom.xml` file.

4.17.4.1 MagicBuilder.java

MagicBuilder.java

```

package SSS.Junit;
public class MagicBuilder {

    public static int getLucky() {
        return 7;
    }
}

```

Textbuild.java

```

package SSS.Junit;

public class Textbuild {

    public static String getHelloWorld(){
        return "hello world";
    }
}

```



```

public static int getNumber10(){
    return 10;
}
}

```

15.4.2 Test class for MagicBuilder

TestMagicBuilder.java

```

package SSS.Junit;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class TestMagicBuilder {

    @Test
    public void testLucky() {
        assertEquals(7, MagicBuilder.getLucky());
    }

}

```

4.17.4.3 Test class for Textbuild

pom.xml

```

package SSS.Junit;

import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class Msgbuild {

    @Test
    public void testHelloWorld() {
        assertEquals("hello world", Textbuild.getHelloWorld());
    }

    @Test
    public void testNumber10() {
        assertEquals(10, Textbuild.getNumber10());
    }

}

```

```
}
```

Step 15.5: Executing Maven Test

- While we run the POM and class files with Maven, the below results will be generated at the console:

4.17.5.1 Run all test classes

```
$ mvn test

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running SSS.Junit.TestMagicBuilder
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in
SSS.Junit.TestMagicBuilder
[INFO] Running SSS.Junit.Msgbuild
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.001 s - in
SSS.Junit.Msgbuild
[INFO]
```

4.17.5.2 Run a single test class Msgbuild

Terminal

```
$ mvn -Dtest=Msgbuild test

[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running SSS.Junit.Msgbuild
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - in
SSS.Junit.Msgbuild
[INFO]
```

4.17.5.3 Run a single test method testHelloWorld() from the test class Msgbuild

Terminal

```
$ mvn -Dtest=Msgbuild#testHelloWorld test




[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running SSS.Junit.Msgbuild
```

[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.004 s - [in](#)
SSS.Junit.Msgbuild
[INFO]



16. Include/Exclude Tests with Tags



Step 16.1: Writing a code to demonstrate Include Tag



```
1 package Demo.ClearBlue;
2 import org.junit.Test;
3 import org.junit.jupiter.api.Tag;
4 import org.junit.jupiter.api.TestInfo;
5 import org.junit.platform.runner.JUnit4Platform;
6 import org.junit.platform.suite.api.IncludeTags;
7 import org.junit.platform.suite.api.SelectPackages;
8 import org.junit.runner.RunWith;
9 @RunWith(JUnit4Platform.class)
10 @SelectPackages("com.howtodoinjava.junit5.examples")
11 @IncludeTags("production")
12 public class Include
13 {
14     public class ClassATest
15     {
16         @Test
17         @Tag("development")
18         @Tag("production")
19         void testCaseA(TestInfo testInfo) { //run in all environments
20         }
21     }
22     public class ClassBTest
23     {
24         @Test
25         @Tag("development")
26         void testCaseB(TestInfo testInfo) {
27         }
28     }
29     public class ClassCTest
30     {
31         @Test
32         @Tag("development")
33         void testCaseC(TestInfo testInfo) {
34         }
35     }
36 }
```

 Console  JUnit 

Finished after 0.27 seconds

Runs: 2/2  Errors: 0  Failures: 0

▼  Demo.ClearBlue.Include [Runner: JUnit 4] (0.002 s)  F

-  JUnit Jupiter (0.001 s)
-  JUnit Vintage (0.001 s)

Step 16.2: Writing a code to demonstrate Exclude Tag

```

3 import org.junit.jupiter.api.Tag;
4 import org.junit.jupiter.api.TestInfo;
5 import org.junit.platform.runner.JUnitPlatform;
6 import org.junit.platform.suite.api.ExcludeTags;
7 import org.junit.platform.suite.api.SelectPackages;
8 import org.junit.runner.RunWith;
9 @RunWith(JUnitPlatform.class)
10 @SelectPackages("com.howtodoinjava.junit5.examples")
11 @ExcludeTags("development")
12 public class Include
13 {
14     public class ClassATest
15     {
16         @Test
17         @Tag("development")
18         @Tag("production")
19         void testCaseA(TestInfo testInfo) { //run in all environments
20         }
21     }
22     public class ClassBTest
23     {
24         @Test
25         @Tag("development")
26         void testCaseB(TestInfo testInfo) {
27         }
28     }
29     public class ClassCTest
30     {
31         @Test
32         @Tag("development")
33         void testCaseC(TestInfo testInfo) {
34         }
35     }
36 }

```

Console
 JUnit

Finished after 0.248 seconds

Runs: 2/2
 Errors: 0
 Failures: 0

Demo.ClearBlue.Include [Runner: JUnit 4] (0.021 s)

JUnit Jupiter (0.006 s)
 JUnit Vintage (0.015 s)

17. Code Coverage

Step 17.1: Installing of EclEmma

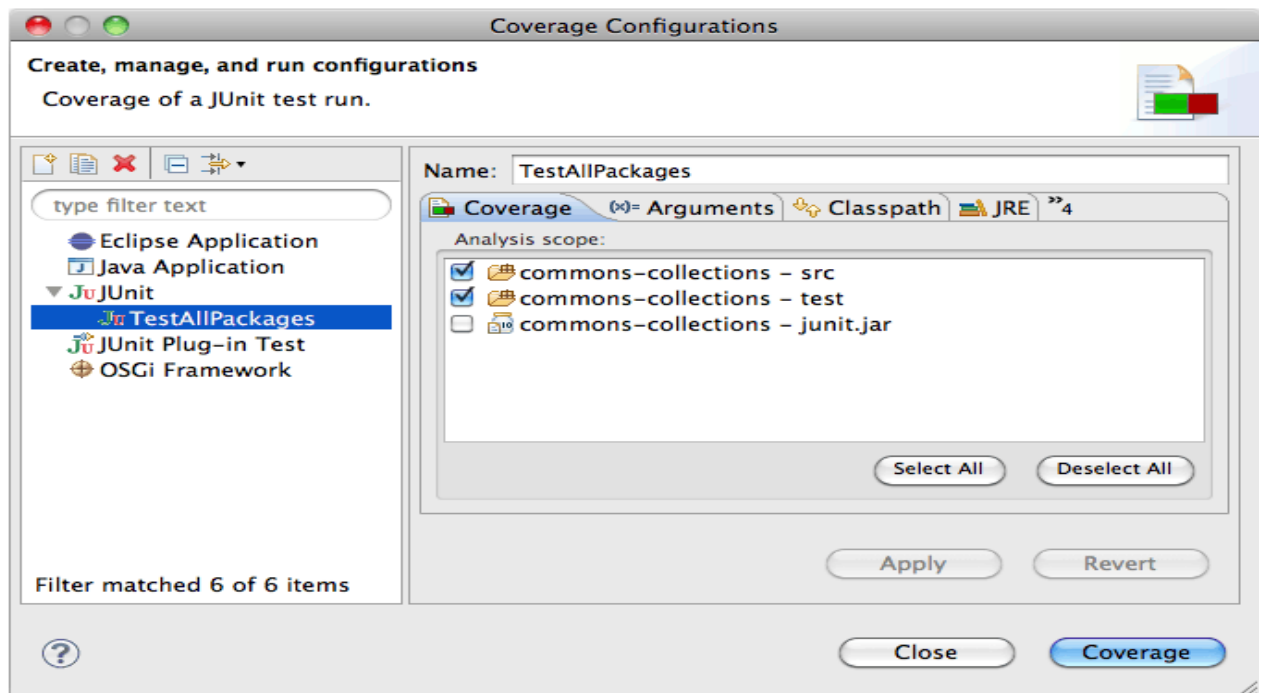
- Install from Eclipse Marketplace Client: EclEmma requires Eclipse 3.8 or higher versions and Java 1.5 or higher versions.
- Follow the steps below or drag and drop the button above into a running Indigo workspace:
 - From your Eclipse menu select Help → Eclipse Marketplace
 - Search for "EclEmma"
 - Hit Install for the entry "EclEmma Java Code Coverage"
 - Follow the steps in the installation wizard

Step 17.2: Verifying Installed EclEmma tool

- The installation was successful if you can see the coverage launcher in the toolbar of the Java perspective:



- Note: If the Coverage drop-down toolbar button is not visible in your current workbench perspective, open the **Customize Perspective...** dialog and enable the **Coverage command group** on the Commands tab.



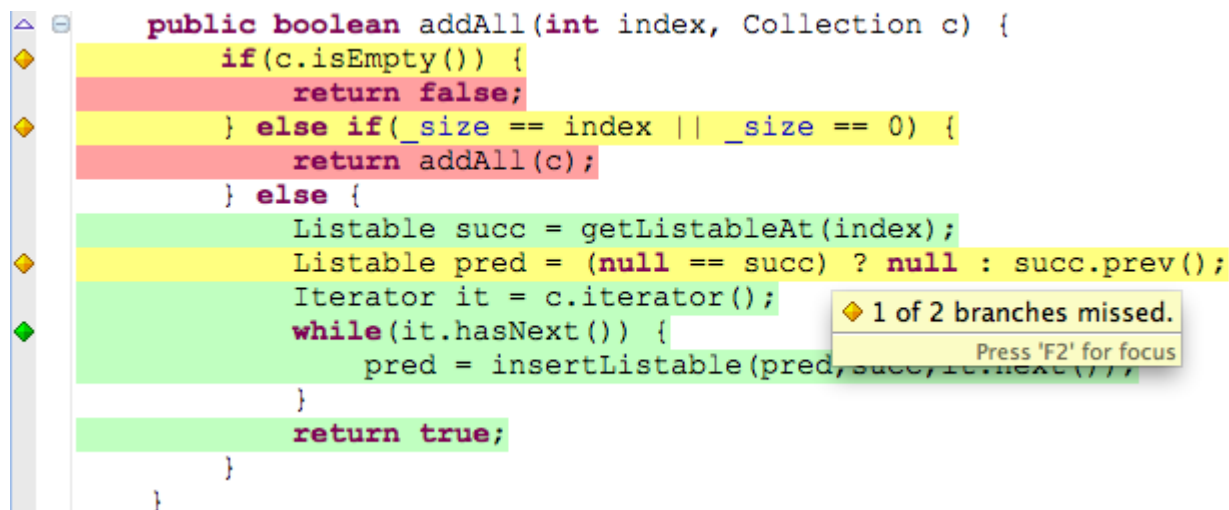
Step 17.3: Using the Coverage View

- The *Coverage* view automatically appears when a new coverage session is added or can be manually opened from the *Window* → *Show View* menu in the *Java* category. It shows coverage summaries for the active session.

Element		Coverage	Covered Lines	Missed Lines	Total Lines
▼ commons-collections		80.7 %	11092	2646	13738
▼ src		80.7 %	11092	2646	13738
▶ org.apache.commons.collections		77.1 %	3991	1188	5179
▶ org.apache.commons.collections.bag		66.9 %	234	116	350
▼ org.apache.commons.collections.bidimap		91.2 %	964	93	1057
▼ AbstractBidiMapDecorator.java		85.7 %	6	1	7
▼ AbstractBidiMapDecorator		85.7 %	6	1	7
AbstractBidiMapDecorator(BidiMap)		100.0 %	2	0	2
getBidiMap()		100.0 %	1	0	1
getKey(Object)		100.0 %	1	0	1
inverseBidiMap()		0.0 %	0	1	1

Step 17.4: Demonstrating Source Code Annotation

- Line coverage and branch coverage of the active coverage session is also directly displayed in the Java source editors. This works for Java source files contained in the project as well as source codes attached to binary libraries.



- Source lines containing executable code have the following color code:
 - **Green** for fully covered lines
 - **Yellow** for partly covered lines (some instructions or branches that are missed)
 - **Red** for lines that have not been executed at all
- In addition, colored diamonds are shown at the left for the lines containing decision branches. The colors for the diamonds have a similar semantic than the line highlighting the colors:
 - **Green** for fully covered branches
 - **Yellow** for partly covered branches
 - **Red** when no branches in the particular line have been executed
- These default colors can be modified in the *Preferences* dialog (see next section). The source annotations automatically disappear when you start editing a source file or delete the coverage session.

Step 17.5: Highlighting Preferences







The Eclipse preferences section *General* → *Appearance* → *Editors* → *Text Editors* → *Annotations* allows to modify the visual representation of coverage highlighting. The corresponding entries are:

- Full Coverage
- Partial Coverage
- No Coverage

Step 17.6: Demonstrating Coverage Properties

For each Java element (Java project, source folder, package, type, or method) EclEmma provides a *Coverage* property page summarizing all coverage counters:

Coverage
Run/Debug Settings

Counter	Coverage	Covered	Missed	Total
 Instructions	65.9 %	924	478	1402
 Branches	76.9 %	83	25	108
 Lines	66.9 %	234	116	350
 Methods	54.9 %	73	60	133
 Types	94.4 %	17	1	18
 Complexity	57.2 %	107	80	187



Cancel

OK