JUnit Framework

Terminology: assertions, annotations, fixtures

Produced by: Dr. Siobhán Drohan
Mairead Meagher
Siobhan Roche

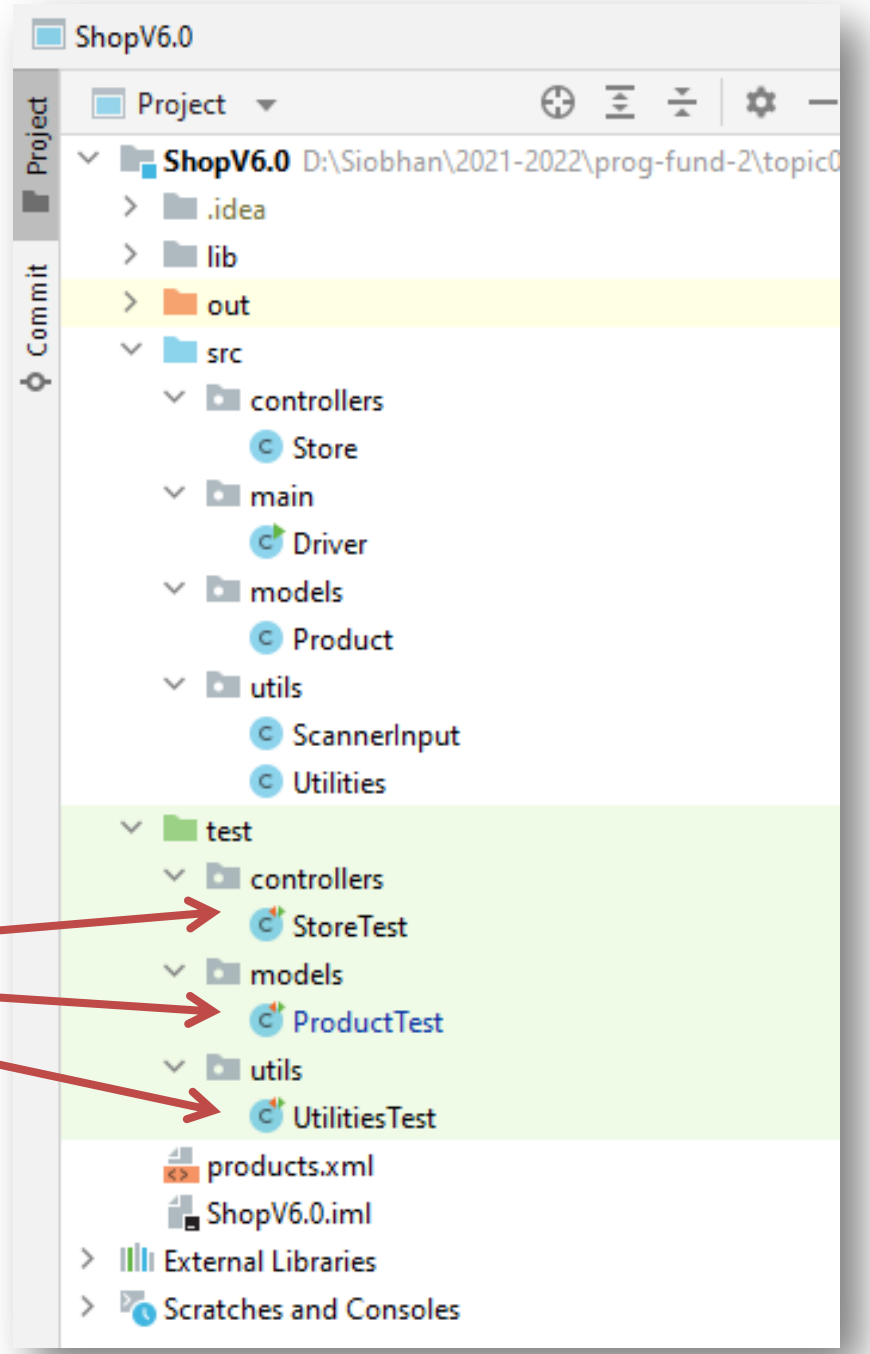
Ollscoil Teicneolaíochta an Oirdheiscirt
South East Technological University

Department of Computing and Mathematics
http://www.wit.ie/

Topic List

- General Terminology
- Assertions
- Annotations
- Fixtures

JUnit Test Class / Test Case



JUnit 5

Setting up the **test fixture**

Tearing down the **test fixture**

Nesting similar tests together

Test Methods

ProductTest.java

```java
package models;

import ...

class ProductTest {

    private Product productBelow, productExact, productAbove, productZero;

    @BeforeEach
    void setUp() {
        //name, 19 chars, code 999, unitCost 1, inCurrentProductLine true.
        productBelow = new Product( productName: "Television 42Inches", productCode: 999, unitCost: 1, inCurrentProductLine: true);
        //name, 20 chars, code 1000, unitCost 999, inCurrentProductLine true.
        productExact = new Product( productName: "Television 50 Inches", productCode: 1000, unitCost: 999, inCurrentProductLine: true);
        //name, 21 chars, code 10000, unitCost 1000, inCurrentProductLine false.
        productAbove = new Product( productName: "Television 60 Inches.", productCode: 10000, unitCost: 1000, inCurrentProductLine: false);
        //name, 0 chars, code 9999, unitCost 0, inCurrentProductLine false.
        productZero = new Product( productName: "", productCode: 9999, unitCost: 0, inCurrentProductLine: false);
    }

    @AfterEach
    void tearDown() { productBelow = productExact = productAbove = productZero = null; }

    @Nested
    class Getters {

        @Test
        void getProductName() {...}

        @Test
        void getUnitCost() {...}

        @Test
        void getProductCode() {...}
```

Topic List

- General Terminology
- Assertions
- Annotations
- Fixtures

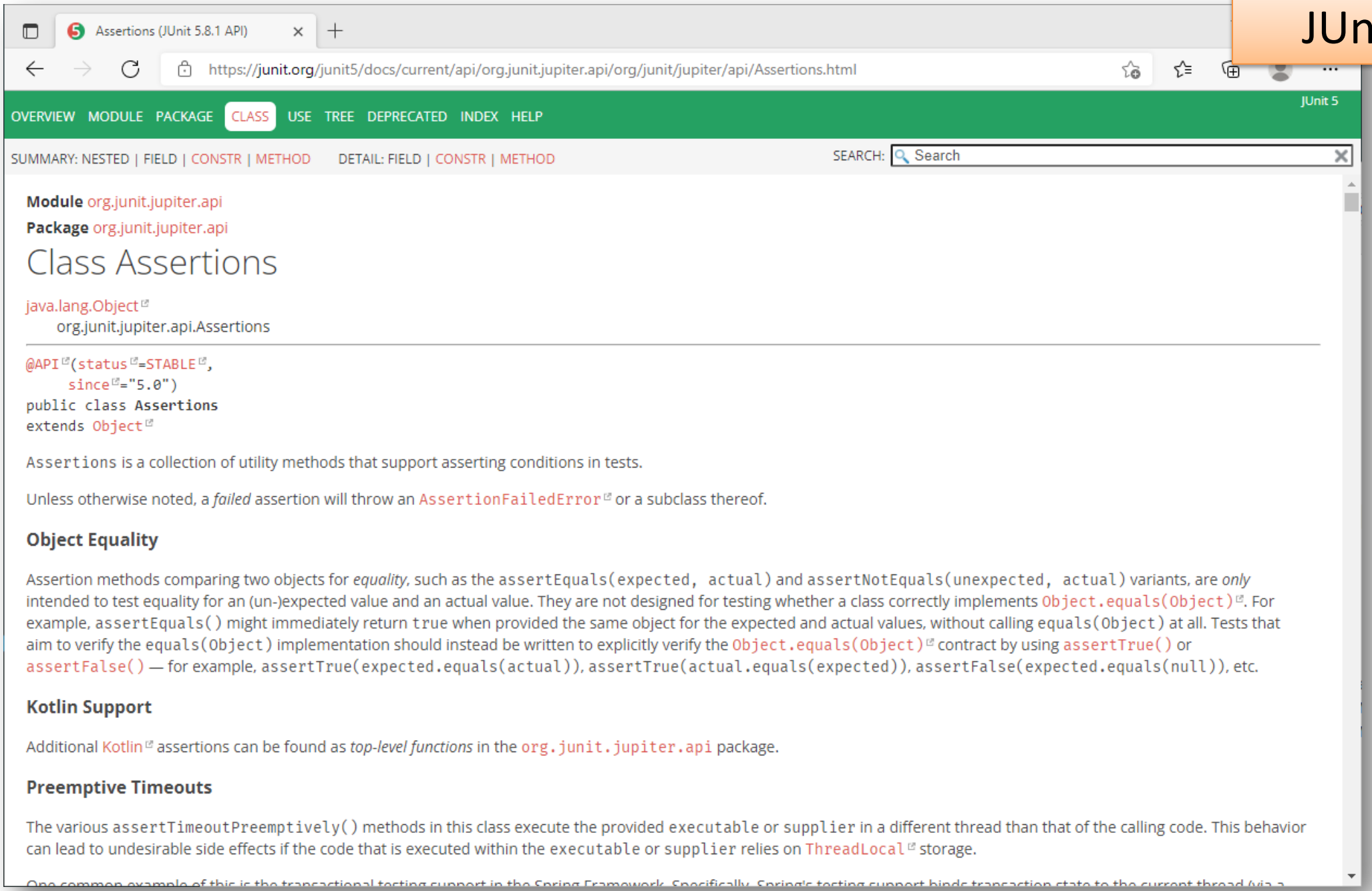
First you import the **Assertions** class from **org.junit…**

Test Methods then contain assertions

JUnit **5**

ProductTest.java

```
9   import static org.junit.jupiter.api.Assertions.*;
10
11  class ProductTest {
12
13      private Product productBelow, productExact, productAbove, productZero;
14
15      @BeforeEach
16      void setUp() {...}
26
27      @AfterEach
28      void tearDown() { productBelow = productExact = productAbove = productZero = null; }
31
32      @Nested
33      class Getters {
34
35          @Test
36          void getProductName() {
37              assertEquals( expected: "Television 42Inches", productBelow.getProductName());
38              assertEquals( expected: "Television 50 Inches", productExact.getProductName());
39              assertEquals( expected: "Television 60 Inches", productAbove.getProductName());
40              assertEquals( expected: "", productZero.getProductName());
41          }
42
43          @Test
44          void getUnitCost() {...}
50
51          @Test
52          void getProductCode() {...}
58
59          @Test
60          void isInCurrentProductLine() {
61              assertTrue(productBelow.isInCurrentProductLine());
62              assertTrue(productExact.isInCurrentProductLine());
63              assertFalse(productAbove.isInCurrentProductLine());
64              assertFalse(productZero.isInCurrentProductLine());
65          }
```



Assertions (JUnit 5.8.1 API)

https://junit.org/junit5/docs/current/api/org.junit.jupiter.api/org/junit/jupiter/api/Assertions.html

OVERVIEW MODULE PACKAGE CLASS USE TREE DEPRECATED INDEX HELP

JUnit 5

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

SEARCH: Search

Module org.junit.jupiter.api

Package org.junit.jupiter.api

Class Assertions

java.lang.Object

org.junit.jupiter.api.Assertions

```
@API(status=STABLE,
     since="5.0")
public class Assertions
extends Object
```

`Assertions` is a collection of utility methods that support asserting conditions in tests.

Unless otherwise noted, a *failed* assertion will throw an `AssertionFailedError` or a subclass thereof.

Object Equality

Assertion methods comparing two objects for *equality*, such as the `assertEquals(expected, actual)` and `assertNotEquals(unexpected, actual)` variants, are *only* intended to test equality for an (un-)expected value and an actual value. They are not designed for testing whether a class correctly implements `Object.equals(Object)`. For example, `assertEquals()` might immediately return `true` when provided the same object for the expected and actual values, without calling `equals(Object)` at all. Tests that aim to verify the `equals(Object)` implementation should instead be written to explicitly verify the `Object.equals(Object)` contract by using `assertTrue()` or `assertFalse()` — for example, `assertTrue(expected.equals(actual))`, `assertTrue(actual.equals(expected))`, `assertFalse(expected.equals(null))`, etc.

Kotlin Support

Additional Kotlin assertions can be found as *top-level functions* in the `org.junit.jupiter.api` package.

Preemptive Timeouts

The various `assertTimeoutPreemptively()` methods in this class execute the provided `executable` or `supplier` in a different thread than that of the calling code. This behavior can lead to undesirable side effects if the code that is executed within the `executable` or `supplier` relies on `ThreadLocal` storage.

One common example of this is the transactional testing support in the Spring Framework. Specifically, Spring's testing support binds transaction state to the current thread (via a

Assertions (JUnit 5.8.1 API)

The Assertions Class

- To check if code is behaving as you expect, you use an assertion.
- An assertion is a simple method call that verifies that something is true.
- The Assertions class Contains a set of assertion methods useful for writing JUnit tests.
- Only failed assertions are recorded i.e. an **AssertionFailedError** is thrown and handled by JUnit.

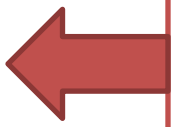
The Assertions Class

These methods can be used directly:

```
Assertions.assertEquals(...);
```

However, they read better if they are referenced through a static import:

```
import static org.junit.jupiter.api.Assertions.*;
//some code
assertEquals(...);
```



We will use this approach.

Some common Assert methods (1)

Method Summary	
static void	**assertEquals**(double expected, double actual, double delta) Asserts that two doubles are equal to within a positive delta.
static void	**assertEquals**(long expected, long actual) Asserts that two longs are equal.
static void	**assertEquals**(Object expected, Object actual) Asserts that two objects are equal.
static void	**assertNotEquals**(double unexpected, double actual, double delta) Asserts that two doubles are **not** equal to within a positive delta.
static void	**assertNotEquals**(Object unexpected, Object actual) Asserts that two objects are **not** equals.
static void	**assertNotSame**(Object unexpected, Object actual) Asserts that two objects do not refer to the same object.
static void	**assertSame**(Object expected, Object actual) Asserts that two objects refer to the same object.

Assertions (JUnit 5.8.1 API)

Some common Assert methods (2)

Method Summary	
static void	**assertNotNull**(Object object) Asserts that an object isn't null.
static void	**assertNotNull**(String message, Object object) Asserts that an object isn't null.
static void	**assertNull**(Object object) Asserts that an object is null.
static void	**assertFalse**(boolean condition) Asserts that a condition is false.
static void	**assertTrue**(String message, boolean condition) Asserts that a condition is true.
static void	**fail**() Fails a test with no message.
static void	**fail**(String message) Fails a test with the given message.

Assertions (JUnit 5.8.1 API)

JUnit 5

assertEquals

assertTrue

assertFalse

ProductTest.java

```
9   import static org.junit.jupiter.api.Assertions.*;
10
11  class ProductTest {
12
13      private Product productBelow, productExact, productAbove, productZero;
14
15      @BeforeEach
16      void setUp() {...}
26
27      @AfterEach
28      void tearDown() { productBelow = productExact = productAbove = productZero = null; }
31
32      @Nested
33      class Getters {
34
35          @Test
36          void getProductName() {
37              assertEquals( expected: "Television 42Inches", productBelow.getProductName());
38              assertEquals( expected: "Television 50 Inches", productExact.getProductName());
39              assertEquals( expected: "Television 60 Inches", productAbove.getProductName());
40              assertEquals( expected: "", productZero.getProductName());
41          }
42
43          @Test
44          void getUnitCost() {...}
50
51          @Test
52          void getProductCode() {...}
58
59          @Test
60          void isInCurrentProductLine() {
61              assertTrue(productBelow.isInCurrentProductLine());
62              assertTrue(productExact.isInCurrentProductLine());
63              assertFalse(productAbove.isInCurrentProductLine());
64              assertFalse(productZero.isInCurrentProductLine());
65          }
```

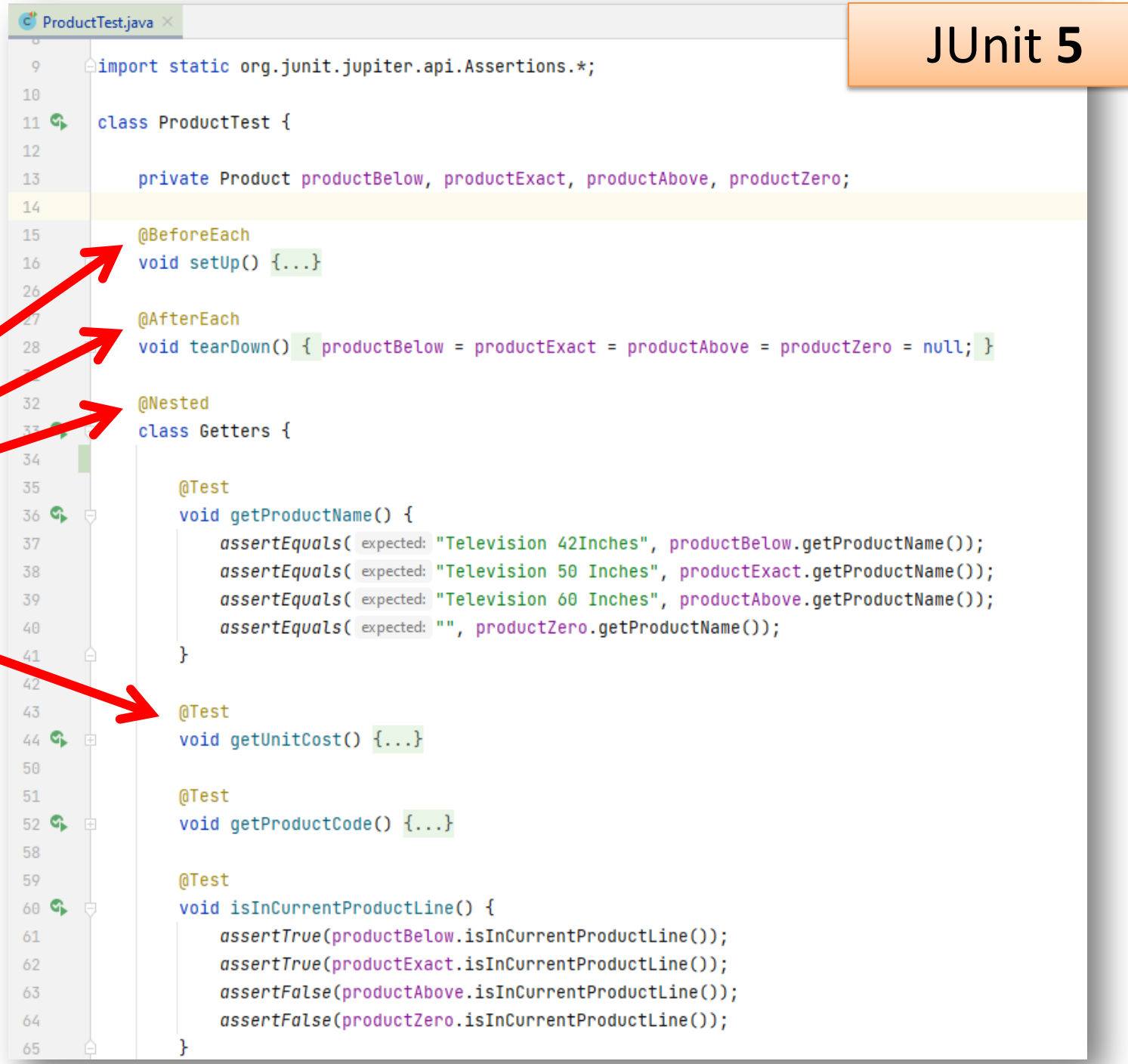
Topic List

- General Terminology
- Assertions
- Annotations
- Fixtures

JUnit **5**

These methods have **Annotations**

@someword



ProductTest.java

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

class ProductTest {

    private Product productBelow, productExact, productAbove, productZero;

    @BeforeEach
    void setUp() {...}

    @AfterEach
    void tearDown() { productBelow = productExact = productAbove = productZero = null; }

    @Nested
    class Getters {

        @Test
        void getProductName() {
            assertEquals( expected: "Television 42Inches", productBelow.getProductName());
            assertEquals( expected: "Television 50 Inches", productExact.getProductName());
            assertEquals( expected: "Television 60 Inches", productAbove.getProductName());
            assertEquals( expected: "", productZero.getProductName());
        }

        @Test
        void getUnitCost() {...}

        @Test
        void getProductCode() {...}

        @Test
        void isInCurrentProductLine() {
            assertTrue(productBelow.isInCurrentProductLine());
            assertTrue(productExact.isInCurrentProductLine());
            assertFalse(productAbove.isInCurrentProductLine());
            assertFalse(productZero.isInCurrentProductLine());
        }
```

What are Annotations?

- Annotations provide data about a program that is not part of the program itself. They have no direct effect on the operation of the code they annotate.

- Annotations can be applied to a program's declarations of classes, fields, methods, and other program elements.

What are Annotations?

- Annotations have a number of uses, among them:
  - Information for the compiler — Annotations can be used by the compiler to detect errors or suppress warnings.
  - Compiler-time and deployment-time processing — Software tools can process annotation information to generate code, XML files, and so forth.
  - Runtime processing — Some annotations are available to be examined at runtime.

JUnit 5 Annotations

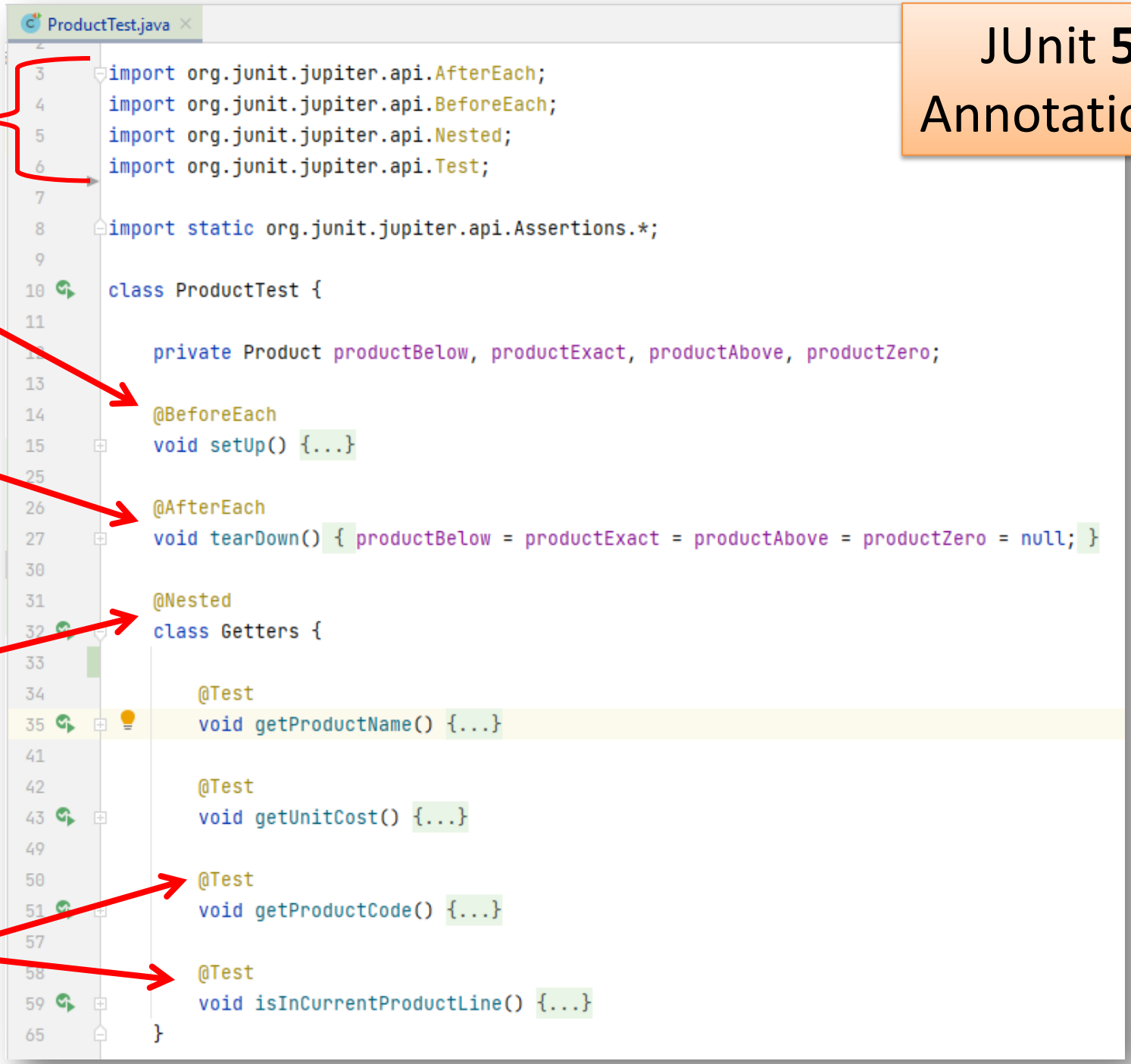
Import the required Annotation class(es) from org.junit.

@BeforeEach runs the method before each test.

@AfterEach
runs the method after each test.

@Nested
Groups together a series of methods into a nested class.

@Test
identifies that a method is a test method.



Other Useful **JUnit5** Annotations

@BeforeAll **public void method()**	Will execute the method once, before the start of all tests in the current class.
@AfterAll **public void method()**	Will execute the method once, after all tests in the current class have finished.
@Disabled **public void method()**	Used to disable a test class or test method.

Topic List

- General Terminology
- Assertions
- Annotations
- Fixtures

JUnit 5

Test fixture fields.

Setting up the test fixture.

Tearing down the test fixture.

ProductTest.java

```java
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Nested;
import org.junit.jupiter.api.Test;

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

class ProductTest {

    private Product productBelow, productExact, productAbove, productZero;

    @BeforeEach
    void setUp() {
        //name, 19 chars, code 999, unitCost 1, inCurrentProductLine true.
        productBelow = new Product( productName: "Television 42Inches", productCode: 999, unitCost: 1, inCurrentProductLine: true);
        //name, 20 chars, code 1000, unitCost 999, inCurrentProductLine true.
        productExact = new Product( productName: "Television 50 Inches", productCode: 1000, unitCost: 999, inCurrentProductLine: true);
        //name, 21 chars, code 10000, unitCost 1000, inCurrentProductLine false.
        productAbove = new Product( productName: "Television 60 Inches.", productCode: 10000, unitCost: 1000, inCurrentProductLine: false);
        //name, 0 chars, code 9999, unitCost 0, inCurrentProductLine false.
        productZero = new Product( productName: "", productCode: 9999, unitCost: 0, inCurrentProductLine: false);
    }

    @AfterEach
    void tearDown() {
        productBelow = productExact = productAbove = productZero = null;
    }
```

Fixtures

- A **Fixture** is a fixed state of a set of objects used as a baseline for running tests.
- Test fixtures allow tests to share common test data.
- The purpose of a test fixture is to ensure that there is a fixed environment in which tests are run so that results are repeatable.
- It includes:
  - setUp() method which runs before every test method.
  - tearDown() method which runs after every test method.

http://www.tutorialspoint.com/junit/junit_test_framework.htm

Any Questions?