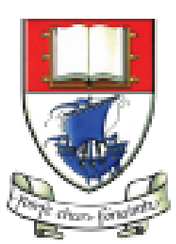
Annotations in Java (JUnit)

Produced by:

Eamonn de Leastar (edeleastar@wit.ie)

Dr. Siobhán Drohan (sdrohan@wit.ie)



Waterford Institute *of* Technology
INSTITIÚID TEICNEOLAÍOCHTA PHORT LÁIRGE

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

What are Annotations?

- They are metadata:
  - Provide information for the compiler (and humans) about the program.
    - Not part of the program itself and don’t affect the code they are annotating.
- Some software tools use annotations to generate code; we will do this when we start working with Xtend.

Where are Annotations used?

- Annotations are typically applied to declarations e.g.
  - classes
  - fields
  - methods, and
  - other program elements.

Activity.java from lab04 → @Override

```java
@Override
public int hashCode()
{
    return Objects.hashCode(this.id, this.type,
    this.location, this.distance);
}

@Override
public String toString()
{
    return toStringHelper(this).addValue(id)
                               .addValue(type)
                               .addValue(location)
                               .addValue(distance)
                               .addValue(route)
                               .toString();
}
```

Compiler checks methods are actually overridden. Also makes the code more human-readable.

Some Built in Annotations

- There are three annotation types that are predefined by the language specification itself:

  - @Deprecated— indicates that the marked element is deprecated and should no longer be used. The compiler generates a warning whenever a program uses a method, class, or field with the @Deprecated annotation.

  - @Override annotation informs the compiler that the element is meant to override an element declared in a superclass. It not required to use this annotation when overriding a method, it helps to prevent errors. If a method marked with @Override fails to correctly override a method in one of its superclasses, the compiler generates an error.

  - @SuppressWarnings annotation tells the compiler to suppress specific warnings that it would otherwise generate e.g. casting errors.

https://docs.oracle.com/javase/tutorial/java/annotations/predefined.html

JUnit 3

- Last week's slides used JUnit 3 conventions.
- Test class extend TestCase
- setUp/tearDown are overridden from TestCase
- test methods must begin with "test" word.

```
import junit.framework.TestCase;

public class TestLargest extends TestCase
{
  private int[] arr;

  public TestLargest (String name)
  {
    super(name);
  }

  public void setUp()
  {
    arr = new int[] {8,9,7};
  }

  public void tearDown()
  {
    arr = null;
  }

  public void testOrder ()
  {
    assertEquals(9, Largest.largest(arr));
  }

  public void testOrder2 ()
  {
    assertEquals(9, Largest.largest(new int[] { 9, 8, 7 }));
    assertEquals(9, Largest.largest(new int[] { 8, 9, 7 }));
    assertEquals(9, Largest.largest(new int[] { 7, 8, 9 }));
  }
}
```

JUnit 4 included Annotations

- **@Before -** run before each test case
- **@After** - run after each test case
- **@Test** - the test case itself
- No need to extend TestCase
- We will use Junit 4 from here on in.

```
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import static org.junit.Assert.fail;
import static org.junit.Assert.assertTrue;
import static org.junit.Assert.assertEquals;

public class TestLargest
{
  private int[] arr;

  @Before
  public void setUp()
  {
    arr = new int[] {8,9,7};
  }

  @After
  public void tearDown()
  {
    arr = null;
  }

  @Test
  public void order ()
  {
    assertEquals(9, Largest.largest(arr));
  }

  @Test
  public void dups ()
  {
    assertEquals(9, Largest.largest(new int[] { 9, 7, 9, 8 }));
  }
```

Exceptions: JUnit 3 vs JUnit 4

- Use @Test (expected = ...) to specify exception
- Simpler, but less verbose

```
public void testEmpty ()
{
  try
  {
    Largest.largest(new int[] {});
    fail("Should have thrown an exception");
  }
  catch (RuntimeException e)
  {
    assertTrue(true);
  }
}
```

```
@Test (expected = RuntimeException.class)
public void testEmpty ()
{
  Largest.largest(new int[] {});
}
```

JUnit 4 Annotations (1)

Annotation	Parameters	Use
`@After`	None	Method will be executed after each test method (similar to the `tearDown()` method in JUnit 3.x). Multiple methods may be tagged with the `@After` annotation, however no order is guaranteed.
`@AfterClass`	None	Method will be executed after all of the test methods and teardown methods have been executed within the class. Multiple methods may be tagged with the `@AfterClass` annotation, however no order is guaranteed.
`@Before`	None	Method will be executed before each test method (similar to the `setUp()` method in JUnit 3.x). Multiple methods may be tagged with the `@Before` annotation, however no order is guaranteed.
`@BeforeClass`	None	Executed before any other methods are executed within the class. Multiple methods may be tagged with the `@BeforeClass` annotation, however no order is guaranteed.
`@Ignore`	String (optional)	Used to temporarily exclude a test method from test execution. Accepts an optional String reason parameter.

https://dzone.com/refcardz/junit-and-easymock

JUnit 4 Annotations (2)

`@Parameters`	None	Indicates a method that will return a Collection of objects that match the parameters for an available constructor in your test. This is used for parameter driven tests.
`@RunWith`	Class	Used to tell JUnit the class to use as the test runner. The parameter must implement the interface `junit.runner.Runner`.
`@SuiteClasses`	Class []	Tells JUnit a collection of classes to run. Used with the `@RunWith(Suite.class)` annotation is used.
`@Test`	■ Class(optional) ■ Timeout(optional)	Used to indicate a test method. Same functionality as naming a method `public void testXYZ()` in JUnit 3.x. The class parameter is used to indicate an exception is expected to be thrown and what the exception is. The timeout parameter specifies in milliseconds how long to allow a single test to run. If the test takes longer than the timeout, it will be considered a failure.

https://dzone.com/refcardz/junit-and-easymock

Creating your own annotations

- So, can you create your own annotations?
- Why yes you can!
- And this link steps you through the process:

https://docs.oracle.com/javase/tutorial/java/annotations/declaring.html

junit.org/junit5/

JUnit 5

JUnit 4

The new major version of the programmer-friendly testing framework for Java 8

User Guide | Javadoc | Code & Issues | Q & A

About

JUnit 5 is the next generation of JUnit. The goal is to create an up-to-date foundation for developer-side testing on the JVM. This includes focusing on Java 8 and above, as well as enabling many different styles of testing.

JUnit 5 is the result of JUnit Lambda and its crowdfunding campaign on Indiegogo.

The JUnit 5 team released Milestone 2 on July 23, 2016, and is currently working on additional milestones and ultimately a GA release (due late 2016).

Upcoming Events

2016-10-10
JAX London 2016 in London, UK
Nicolai Parlog

2016-11-09
Deep Dive into JUnit 5 - Devoxx Belgium 2016 in Antwerp, Belgium
Sam Brannen

JUnit 5 vs. JUnit

Features	JUnit 5	JUnit 4
Declares a test method	**@Test**	**@Test**
Denotes that the annotated method will be executed before all test methods in the current class	**@BeforeAll**	**@BeforeClass**
Denotes that the annotated method will be executed after all test methods in the current class	**@AfterAll**	**@AfterClass**
Denotes that the annotated method will be executed before each test method	**@BeforeEach**	**@Before**
Denotes that the annotated method will be executed after each test method	**@AfterEach**	**@After**
Disable a test method or a test class	**@Disable**	**@Ignore**
Denotes a method is a test factory for dynamic tests in JUnit 5	**@TestFactory**	N/A
Denotes that the annotated class is a nested, non-static test class	**@Nested**	N/A
Declare tags for filtering tests	**@Tag**	**@Category**
Register custom extensions in JUnit 5	**@ExtendWith**	N/A

Table source: http://howtoprogram.xyz/2016/08/10/junit-5-vs-junit-4/

More information: https://dzone.com/articles/first-look-at-junit-5

JUnit 5 vs. JUnit

- This course is using JUnit4.

- http://junit.org/junit5/docs/current/user-guide/#running-tests-ide

> ## 4.1. IDE Support
>
> At the time of this writing there is no direct support for running tests on the JUnit Platform within IDEs. However, the JUnit team provides two intermediate solutions so that you can go ahead and try out JUnit 5 within your IDE today. You can use the Console Launcher manually or execute tests with a JUnit 4 based Runner.

- If you wish to investigate JUnit5, please go to this link: http://junit.org/junit5/