

# **Core Java 8 and Development Tools**

Lesson 14 : Introduction to Junit 4

## Lesson Objectives

- After completing this lesson, participants will be able to
  - Understand importance of unit testing
  - Install and use JUnit 4
  - Use JUnit Within Eclipse



Copyright © Capgemini 2015. All Rights Reserved 2

This lesson covers the one of the important development tool called JUnit. It explains how to work with JUnit, install, configure tests, run test with use of an IDE.

Lesson outline:

- 14.1: Introduction
- 14.2: JUnit
- 14.3: Installing and Running JUnit
- 14.4: Testing with JUnit

14.1: Introduction

## Why is Testing Necessary

- To test a program implies adding value to it.
  - Testing means raising the reliability and quality of the program.
  - One should not test to show that the program works rather that it does not work.
  - Therefore testing is done with the intent of finding errors.
- Testing is a costly activity.



Copyright © Capgemini 2015. All Rights Reserved 3

### Why Testing?

#### Why is Testing Necessary?

In SDLC, testing plays a vital role. When one tests a program one adds value to the program, in turn raising the quality and reliability of the program.

When we say “reliable”, it implies finding and removing errors. Hence one should not test a program to show that it works, but to show that program does not work.

Testing cannot guarantee against software problems or even failures but it can minimize the risks of faults developing once the software is put to use.

Typically when testing one should start with assumptions that the program contains errors and the test the program to find as many errors as possible.

Testing is a costly activity. A test which does not find an error is a waste of time and money. “A test case that finds an error is a valuable investment”.

14.1: Introduction

## What is Unit Testing

- The process of testing the individual subprograms, subroutines, or procedures to compare the function of the module to its specifications is called Unit Testing.
  - Unit Testing is relatively inexpensive and an easy way to produce better code.
  - Unit testing is done with the intent that a piece of code does what it is supposed to do.



Copyright © Capgemini 2015. All Rights Reserved

4

### Why Testing?

#### What is Unit Testing?

There are various phases in testing. However, in this course we are mainly concentrating on unit testing. Testing individual subprograms or procedure to compare the functions of the module to its required specifications is Unit Testing. This is an inexpensive activity and a very easy way to produce better code. In other words, unit test is a small piece of code that is written by the developer that will exercise a specific area of functionality of the code being tested.

For example: You write a functionality to sort a list and then check if the sort works. You then modify the functionality to accept the sort order, as well, and then you test this newly added functionality.

The point to note here is that while doing Unit testing, the developers are not worried about verification and validation of the program. It is just that the functionality should be running as required.

Unit Testing is also called as Test Driven Development (TDD). A significant advantage of TDD is that it enables you to take small steps while writing software. Most of you will be already doing some amount of unit testing in an ad hoc manner.

14.1: Introduction

## What is Test-Driven Development (TDD)

- Test-Driven Development, also called Test-First Development, is a technique in which you write unit tests before writing the application functionality.
  - Tests are non-production code written in the same language as the application.
  - Tests return a simple pass or fail, giving the developer immediate feedback.



Copyright © Capgemini 2015. All Rights Reserved 5

Test Driven Development (TDD) requires developers to create automated unit tests that define code requirements before writing the code itself.

14.1: Introduction

## Why Unit Testing

- You can cite following reasons for doing a Unit Test:
  - Unit testing helps developers find errors in code.
  - It helps you write better code.
  - Unit testing saves time later in the production/development cycle.
  - Unit testing provides immediate feedback on the code.



Copyright © Capgemini 2015. All Rights Reserved

6

### Why Testing?

#### Why Unit Testing?

Why should a developer do unit testing? Some of the reasons that can be identified are given below:

Unit Testing helps developers find errors in code: When a developer starts writing unit tests, they can actually be surprised with how many errors are encountered in a small function that is written. It makes life of the developer easy. In case errors are “not found in time”, and are delayed till the end, then the entire module may fail.

Testing helps you write better code: Unit testing can help developer during the initial phase of development. Unit testing makes designs better and drastically reduce the time required while debugging.

Unit testing will save time later: To understand this concept, let us consider an example. Suppose you have written a small piece of code and you start testing. You identify that it is likely to fail in one tricky situation and the test finds out the error. You fix it, and continue the development. Since you have already fixed the bug, there is a little chance that the module might fail.

“Hence following the test driven development approach is beneficial”.

Unit testing provides immediate feedback on the code: When a developer is writing a critical module / functionality for a user interface, testing at that point will provide immediate feedback. There is no need to wait till the code becomes part of the application and then test it to check whether it works.

Overall Unit testing activity benefits the developer.

14.2: JUnit

## Need for Testing Framework

- Testing without a framework is mostly ad hoc.
- Testing without framework is difficult to reproduce.
- Unit testing framework provides the following advantages:
  - It allows to organize and group multiple tests.
  - It allows to invoke tests in simple steps.
  - It clearly notifies if a test has passed or failed.
  - It standardizes the way tests are written.



Copyright © Capgemini 2015. All Rights Reserved 7

Why use JUnit?

Need for Testing Framework:

After having understood the need for Unit testing, there is a requirement to understand how to do Unit testing.

Mostly the Unit Testing done by the developers is ad hoc. The tests done in this manner are not put across in code at all. If at all they are put up, then they are written in such a manner that they cannot be reused in future. If they can be used in future, then typically they might be reproduced differently every time they are used. "Hence it is said that testing without a framework is difficult to reproduce".

With the help of a framework, the tests get documented in code and are reproduced in the same manner, whenever required.

14.2: JUnit

## What is JUnit

- JUnit is a free, open source, software testing framework for Java.
- It is a library put in a jar file.
- It is not an automated testing tool.
- JUnit tests are Java classes that contain one or more unit test methods.



Copyright © Capgemini 2015. All Rights Reserved 8

Why use JUnit?

What is JUnit?

JUnit is an open source, software testing framework for Java developed by Kent Beck and Erich Gamma. JUnit allows developers to write Unit test cases for your Java code. It is library put in a jar file.

JUnit is not an automated testing tool. The developer has to write the test files and execute. JUnit offers some support so that the developer can easily write test files. It includes a tool which is called test runner to run your test files.

JUnit provides an easy way to state how the code should work. By expressing your intentions in code, you can use the JUnit test runners to verify that your code behaves according to your intentions.



14.2: JUnit

## Why JUnit

- JUnit allows you to write tests faster while increasing quality and stability.
- It is simple, elegant, and inexpensive.
- The tests check their own result and provide feedback immediately.
- JUnit tests can be put together in a hierarchy of test suites.
- The tests are written in Java.

Why use JUnit?

Why JUnit?

Many developers will agree to the fact that the code should be tested before it is delivered. We have already seen the reasons why a testing framework should be used. However, it is also necessary to understand why JUnit should be used. Some of the reasons are defined below:

JUnit allows you to write tests faster while increasing quality and stability:

Using JUnit, a developer spends less time debugging, and confidently makes changes to the code. With constant testing, any new functionality that is added can be verified for whether it is working or not. Hence the developer can be more positive about adding new features, because the developer now knows that it is less likely to fail. If a bug is detected while running tests, then the source code is fresh in your mind, so the bug is easily found. Also tests written in JUnit help you write code at a fast pace, and identify defects quickly. Writing tests builds the stability of the code and ensures that any changes that are made are working. As a result of the change, there is no effect on the software.

JUnit is simple, elegant, and inexpensive: "Simplicity" is the keyword while writing a test. Developers should not find it time consuming or difficult to write tests. With JUnit, the TDD can be followed very easily. It is simple and easy to put JUnit in practice. Developers can incrementally write tests as they increment their code. JUnit tests are such that they can be executed easily and frequently and it does not disturb the development process. This framework offers a cheap way of testing since it is an open source and freely downloadable ware.

JUnit tests check their own result and provide feedback immediately:

Manual Unit testing is a tedious task and obviously it is time consuming to compare the expected and the actual result. As a result, developers tend to do away with Unit testing. JUnit tests can be run easily and they check their own results. The developer immediately gets a feedback if the tests have passed or failed. Hence any manual intervention is not required while executing the tests.

JUnit tests can be put together in a hierarchy of test suites: JUnit tests can be organized into test suites containing test cases and even other test suites. The composite behavior of JUnit tests allows you to assemble collections of tests and automatically regression test the entire test suite in one go. You can also run the tests for any layer within the test suite hierarchy.

(Test Suites will be covered later in detail)

JUnit tests are written in Java: Testing Java software using Java tests forms a "seamless bond" between the test and the code under test. The tests become an extension to the overall software and code can be refactored from the tests into the software under test. The Java compiler helps the testing process by performing static syntax checking of the unit tests and ensuring that the software interface contracts are being obeyed.

14.3: Installing and Running JUnit

## Steps for Installing JUnit

- Following are the steps for installing and running JUnit:
  - Download JUnit from [www.junit.org](http://www.junit.org). You can download either the jar file or the zip file.
    - Unzip the JUnit zip file
  - Add the jar file to the CLASSPATH.
    - Set CLASSPATH=.,%CLASSPATH%;junit-4.3.1.jar



Copyright © Capgemini 2015. All Rights Reserved 11

### Installing and Running JUnit:

To start writing tests using JUnit, you will require the jar file for the latest version of JUnit. You can download the jar file or the zip file from the website. The zip file contains the source code and the documentation, as well.

Once downloaded, unzip the zip file that has downloaded. Add the jar file in the classpath. Alternatively the classpath can also be set through the Environment Variables option.

Once done you are ready to use JUnit.

14.3: Installing and Running JUnit

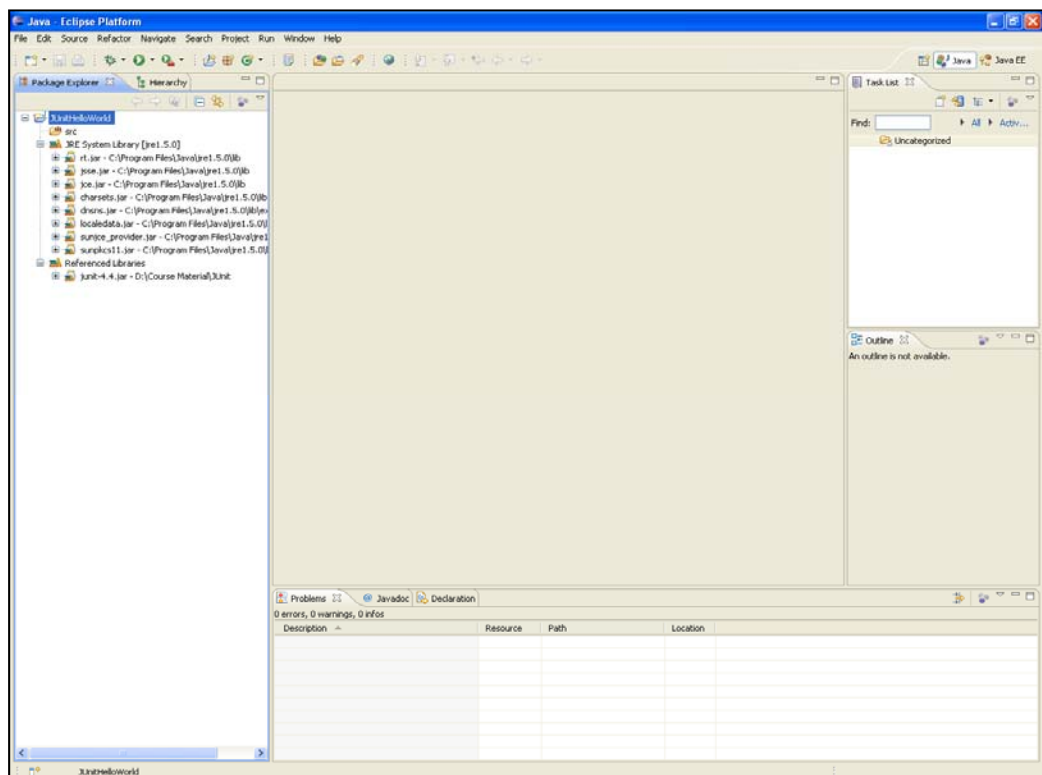
## Using JUnit within Eclipse

- JUnit can be easily plugged in with Eclipse.
- Let us understand how JUnit can be used within Eclipse.
  - Consider a simple “Hello World” program.
  - The code is tested using JUnit and Eclipse IDE.
- Steps for using JUnit within JUnit:
  - Open a new Java project.
  - Add junit.jar in the project Build Path.



Copyright © Capgemini 2015. All Rights Reserved 12

Note: The following figure depicts JUnit being added to the build path of the project.



14.3: Installing and Running JUnit

## Using JUnit within Eclipse (Contd.)

- Write the Test Case as follows:

```
import org.junit.Test;
import static org.junit.Assert.*;
public class TestHelloWorld {
    @Test
    public void testSay()
    {
        HelloWorld hi = new HelloWorld();
        assertEquals("Hello World!", hi.say());
    }
}
```

```
class HelloWorld{
    String say(){
        return "Hello World!";
    }
}
```



Copyright © Capgemini 2015. All Rights Reserved 13



**Note:**

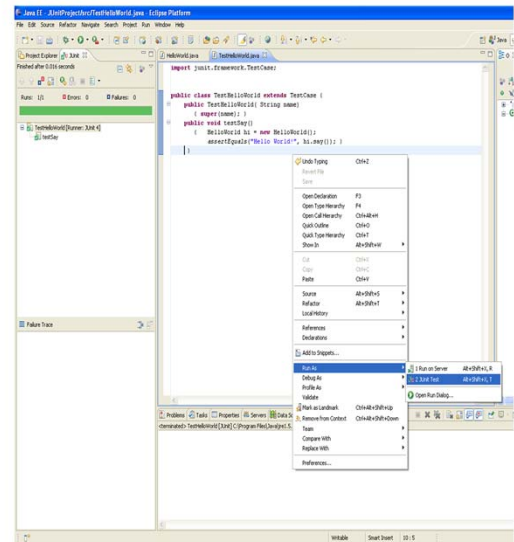
Test method must be annotated with @Test annotation then only Junit framework and eclipse will consider it as test method.

assertEquals(expected,actual) is performing the Test .This function internally uses Java assert feature which throws AssertionError exception when assertion fails.

## 14.3: Installing and Running JUnit

## Using JUnit within Eclipse (Contd.)


- Run the Test Case.
- Right-click the Project  Run As  JUnit Test
- The output of the test case is seen in Eclipse.




14.3: Installing and Running JUnit

## Demo

- Demo on:
  - Using JUnit with Eclipse
    - HelloWorld.java
    - TestHelloWorld.java



 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 15

14.4: Testing with JUnit

## Annotation Types in JUnit4.x

- JUnit4.x introduces support for the following annotations:

- `@Test` – used to signify a method is a test method
- `@Before` – can do initialization task before each test run
- `@After` – cleanup task after each test is executed
- `@BeforeClass` – execute task before start of tests
- `@AfterClass` – execute cleanup task after all tests have completed
- `@Ignore` – to ignore the test method



Copyright © Capgemini 2015. All Rights Reserved 16

### Testing with JUnit – Annotations:

#### Annotation Types in JUnit4.x:

JUnit4.x introduced annotations. They are as follows:

**@Test** : It is used to signify a method as a test method. There are some options which can be mentioned with this annotation. For example:

**@Test(timeout=100)** fails if the test takes longer than 100 milliseconds for execution. This can be used to test infinite loops. Earlier you would have to start every method with 'test' which is now not required. The method can be named whatsoever. You have to simply prefix it with **@Test** annotation.

**@Before** : It is used to carry some task before each test is run. This can be used for initialization required before the test.

**@After** : It is used to perform cleanup after each test is executed.

**@BeforeClass** : It will execute the method before the start of the tests. This can be used for initialization of intensive resources like database connection.

**@AfterClass** : It will execute the method after all tests have finished. Cleanup activities are carried out.

**@Ignore** : It will ignore the test method. This can be useful when the base code has been changed but the test is yet to be revised. You can also use this annotation when you do not want to run a test currently because it takes too long.

(As we go ahead, we will understand each of the annotation in detail.)



14.4: JUnit Framework

## Simple Example using Junit4.x

- Consider the following code snippet:

```
import static org.junit.Assert.*;
import org.junit.Test;
public class FirstJUnitTest {
    @Test
    public void simpleAdd() {
        int result = 1;
        int expected = 1;
        assertEquals(expected, actual);
    }
}
```



Copyright © Capgemini 2015. All Rights Reserved 17

### Understanding JUnit Framework:

Before we move any further, let us understand a very simple test case. Notice the static import. Since JUnit4.x is closely bound to Java 5, the static imports are permitted. JUnit4.x uses org.junit.\* package, which provides JUnit core classes and annotations. As mentioned earlier, the TestCase class is not required to be inherited by the test class.

The class, which includes at least one @Test annotation, is treated as the test class. In JUnit4.x, the assert methods are static. Hence you need to do either of the following:

- Call assert methods using Assert.assertEquals().

- Do a static import as we have done in the snippet shown on the slide.

You also need not tag test methods by starting their name with "test". You instead only need to specify the @Test annotation.

14.4: Testing with JUnit

## Assert Statements in JUnit

- Following are the methods in Assert class :

- Fail(String)
- assertTrue(boolean)
- assertEquals([String message],expected,actual)
- assertNull([message],object)
- assertNotNull([message],object)
- assertEquals([String],expected,actual)
- assertEquals([String],expected,actual)
- assertEquals(String,T actual, Matcher<T> matcher)



Copyright © Capgemini 2015. All Rights Reserved 18

### Testing with JUnit – Assertions:

#### Assert Statements in JUnit:

The org.junit.Assert class provides a set of useful assertions methods. You can call these methods by either using Assert.assertEquals() or by referencing through static import. Any failed assertions will be only recorded. Some assertion methods are as follows:

Fail([String]) : It signals the failure of a test. This method has two formats. If the String argument is not provided, then no message is displayed. Else the String argument message is displayed.

assertTrue(boolean) : It asserts if the condition is true. Similarly  
assertFalse(boolean) asserts if the condition is false.

assertEquals([String message],expected,actual) : It asserts whether the two objects passed as arguments are equal. This method can accept any kind of values for comparison like double long, etc..

assertNull([message],object) : It asserts that an object is null.

assertNotNull([message],object) : It asserts that an object is not null.

assertSame([String],expected,actual) : It asserts that two objects refer to the same object and assertNotSame([String],expected,actual) asserts that two objects do not refer to the same object.

assertThat(String, T actual, Matcher<T> matcher) : It asserts that “actual” satisfies the condition specified by the “matcher”.

14.4: Testing with JUnit

## Demo

- Demo on:
  - Using @Test Annotation
  - Using Assert Methods
    - Counter.java & Testcounter.java
    - Person.java & TestPerson.java



 Capgemini  
CONSULTING TECHNOLOGY OUTSOURCING

Copyright © Capgemini 2015. All Rights Reserved 19

**Note:**

Refer to JUnitProject.

Demo 1: Counter.java and testCounter.java

Demo 2: Person.java and TestPerson.java

In both the examples, we have used the @Test annotation and the assert methods to evaluate if the methods in the underlying class are working properly.

Remove the @Test annotation from one of the Test methods, and then observe the output. That particular method is not considered as a test method.

14.4: Testing with JUnit

## Using @Before and @After

- Test fixtures help in avoiding redundant code when several methods share the same initialization and cleanup code.
- Methods can be annotated with @Before and @After.
  - @Before: This method executes before every test.
  - @After: This method executes after every test.
- Any number of @Before and @After methods can exist.
- They can inherit the methods annotated with @Before and @After.



Copyright © Capgemini 2015. All Rights Reserved 20

### Testing with JUnit – Test Fixtures:

#### Using @Before and @After:

Many a times a set of common resources or data that you need to run one or more tests are required. To avoid the redundant code when several methods share the same initialization and cleanup code, you can use @Before and @After annotations. Prefix the methods with the respective annotations. In the previous version, you had to use setup() and teardown() methods to perform this task.

@Before annotated methods run before every test, and @After prefixed methods run after every test.

You can also have any number of @Before and @After prefixed methods as you need. It is possible to inherit the @Before and @After methods. The @Before methods in the superclass will be executed prior to the derived class @Before methods. The @After in the subclass are executed before the superclass @After methods. The superclass @Before and @After methods execute automatically and there is no need to call them explicitly. Also a logger can be initialized in @Before method of a Test Case.

When inherited, the overall execution process will be as follows:

- @Before methods in the superclass
- @Before methods in the current class
- @Test methods in the current class
- @After methods in the current class
- @After methods in the superclass

14.4: Testing with JUnit

## Using @Before and @After

- Example of @Before:

```
@Before
public void beforeEachTest() {
    Calculator cal=new Calculator();
    Calculator cal1=new Calculator("5", "2"); }
```

- Example of @After:

```
@After
public void afterEachTest() {
    Calculator cal=null;
    Calculator cal1=null; }
```



Copyright © Capgemini 2015. All Rights Reserved 21

Note: The methods annotated with @Before and @After should be declared as public void.

## Lab : Introduction to Junit

- Lab 9: Introduction to Junit



## Review Question

- Question 1: Why should one do Unit Testing?
  - Option 1: Helps to write code better
  - Option 2: Provides immediate feedback on the code
  - Option 3: Because it is one of the testing methods that has to be carried out
- Question 2: JUnit is a licensed product and can be purchased with Java.
  - True / False



## Review Question

- Question 3: To start working with JUnit in Eclipse, you need to add junit.jar in \_\_\_\_.
- Option 1: CLASSPATH
- Option 2: BUILDPATH
- Option 3: Project Settings

