Cognizant | Academy
Passion for making a difference

Exploring JUnit 4.x

Targeted at: Entry Level Trainees

Session 3 & 4 : Jumpstart-JUnit 4.x

Academy

C3: Protected

About the Author

Created By:	B Sai Prasad, (105582)
Credential Information:	Sun Certified Java Programmer, Microsoft Certified Technology Specialist, PMI-certified Project Management Professional
Version and Date:	JUnit/PPT/1110/1.0

Cognizant Certified Official Curriculum

UNDER THE SEAL OF COGNIZANT ACADEMY

Icons Used

Questions

Tools

Hands-on Exercise

Coding Standards

Test Your Understanding

Reference

Try it Out

A Welcome Break

Cognizant Technology Solutions

Contacts

Jumpstart JUnit: Overview

- **Introduction:**
  - JUnit is a framework for unit tests initially developed by *Kent Beck and Erich Gamma*
  - JUnit provides a ready-made test harness for executing unit tests as well as an API for reporting the success or failure status of a test
  - Experience gained with JUnit is important in the development of *test-driven development*
  - In this chapter associates would learn the role of JUnit and how it can be used for writing unit tests

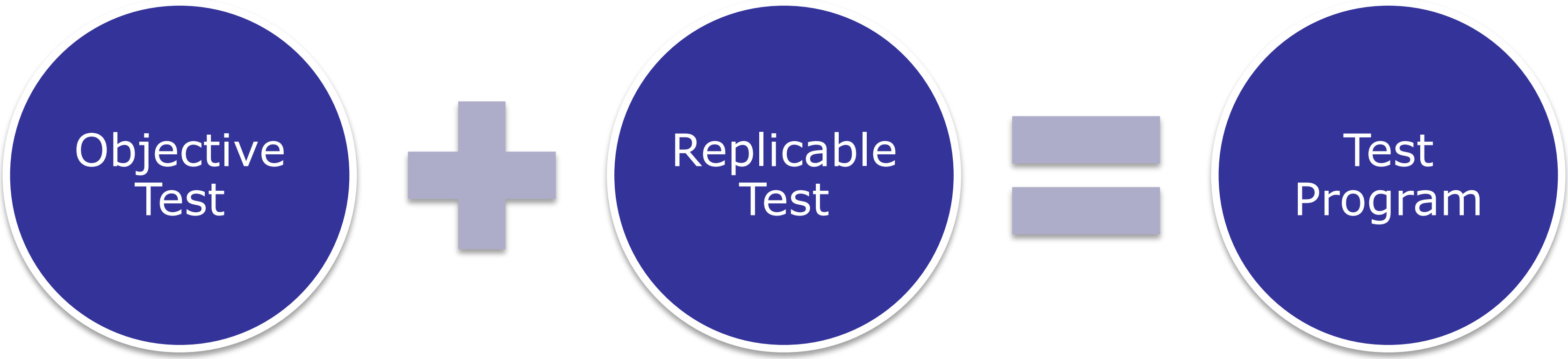
Jumpstart JUnit: Objectives

- **Objective:**

After completing this chapter, associate will be able to:

- Know the features of JUnit
- Write test programs using JUnit TestCase
- Write unit tests using *@Test* annotation
- Manage fixtures using *@Before, @After, @BeforeClass and @AfterClass* annotations
- Launch tests using *@RunWith* annotation
- Build test suite using *Suite.class* test runner

Understanding unit testing frameworks



Objective Test + Replicable Test = Test Program

- All unit testing frameworks should observe:
  - Rule 1: Each unit test must run *independently* of all other unit tests
  - Rule 2: Errors must be *detected* and *reported test by test*
  - Rule 3: It must be *easy to define* which unit tests will run

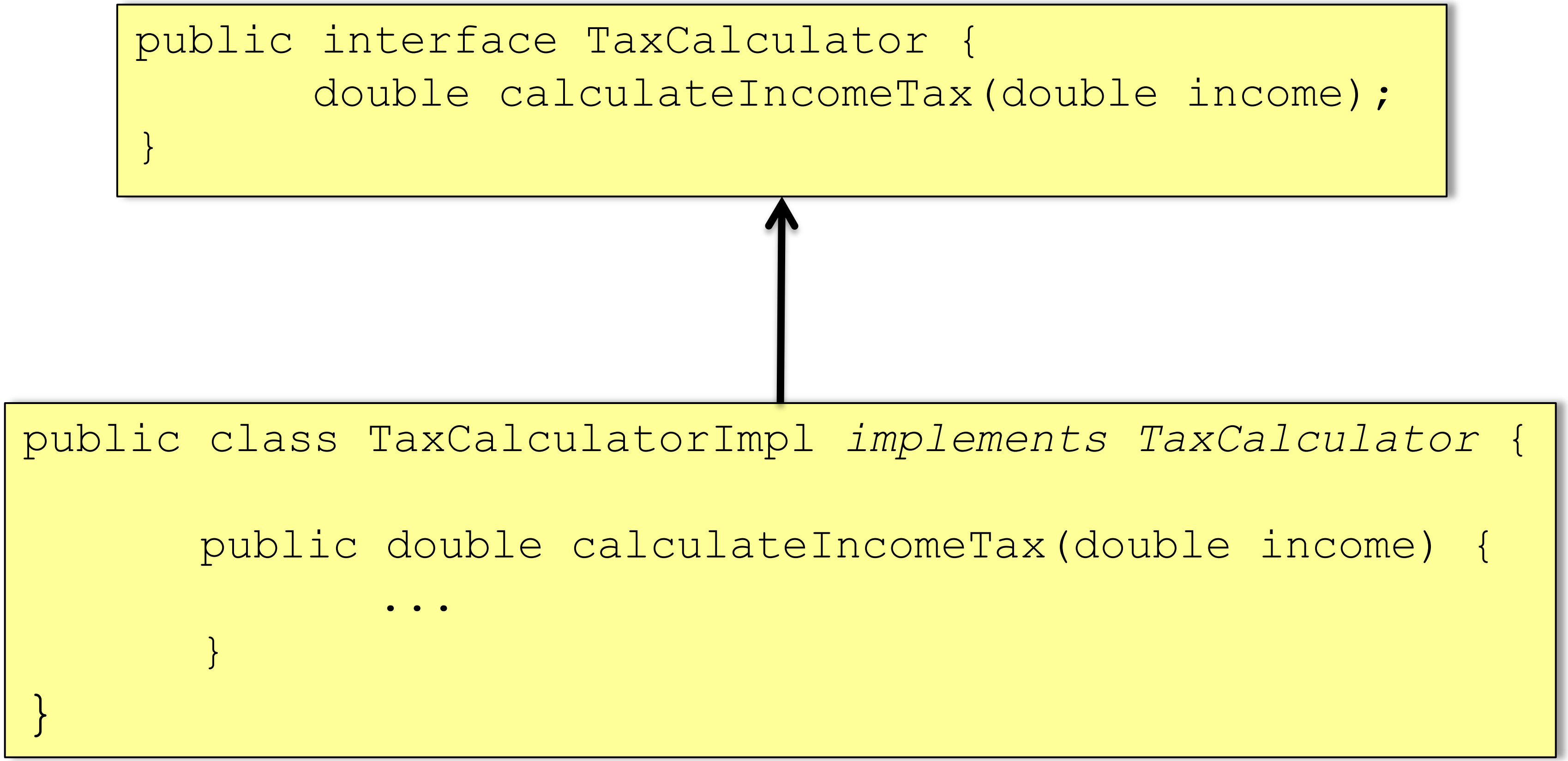
JUnit Features

- JUnit has features to make your tests easier to write and run:
  - Standard resource initialization and reclamation methods
  - Separate *classloaders* for each unit test to avoid side effects
  - A variety of *assert* methods to check the results of your tests
  - Alternate front-ends (or test runners) to display the result of your tests
  - Integration with popular tools like Ant, Maven and popular IDEs like Eclipse, Jbuilder

JUnit Design Goals

1. The framework must help to *write useful tests*
2. The framework must help us *lower the cost* of writing tests by reusing code
3. The framework must help us create tests that *retain value over time*

Class to be tested



```
public interface TaxCalculator {
      double calculateIncomeTax(double income);
}
```

```
public class TaxCalculatorImpl implements TaxCalculator {

      public double calculateIncomeTax(double income) {
            ...
      }
}
```

Test Case using JUnit

```
import static org.junit.Assert.*;
import org.junit.Test;
public class TaxCalculatorImplTest {
     @Test
     public void shouldUseLowestTaxRateForIncomeBelow38000() {
        TaxCalculatorImpl calc = new TaxCalculatorImpl();
        double expectedTax = 30000 * 0.195;
        double calculatedTax = calc.calculateIncomeTax(30000);
        assertEquals("Tax below 38000 should be taxed at
   19.5%“, expectedTax, calculatedTax, 0);
   }
}
```

“JUnit First Design Goal” Achieved
The framework must help to write useful tests.
Any class can be a test case and all test methods should have *@Test* annotation

Cognizant | Academy
Passion for making a difference

Exploring JUnit

Tests
Runner
Result

- Tests:
  - Any POJO class can be a test case
  - It contains one or more related tests; no special naming convention required
- Runner:
  - A launcher of tests
  - *@RunWith* annotation is used to indicate the runner to be used
- Result:
  - It collects any errors or failures that occur during a test
  - Every **`Runner`** has a **`Result`**

Writing the TestCase

- Creating a test case with JUnit framework requires:
  1. The test class does not need to extend any particular class
  2. Unit test methods to be marked by *@Test* annotation
  3. All unit test methods to be **`public void`** and take no parameters
  4. Test methods to make **`assert`** calls to validate the outcome

Annotated Methods

To run the method, JUnit first constructs a fresh instance then invokes the annotated method.

Explain the Failure Reason

When **`assert`** methods are used, make sure the signature that takes **`String`** is used.

JUnit Test Fixture

- *How do you ensure the results of a test are repeatable?*
- There should be a *well known* and *fixed environment* in which tests are run, so that the results are repeatable
- Examples:
  - Loading a database with a specific, known set of data
  - Copying a specific known set of files
  - Preparation of input data and setup/creation of fake or mock objects
- A test fixture is a fixed state of a set of objects used as a baseline for running tests

Managing Resources With Fixtures

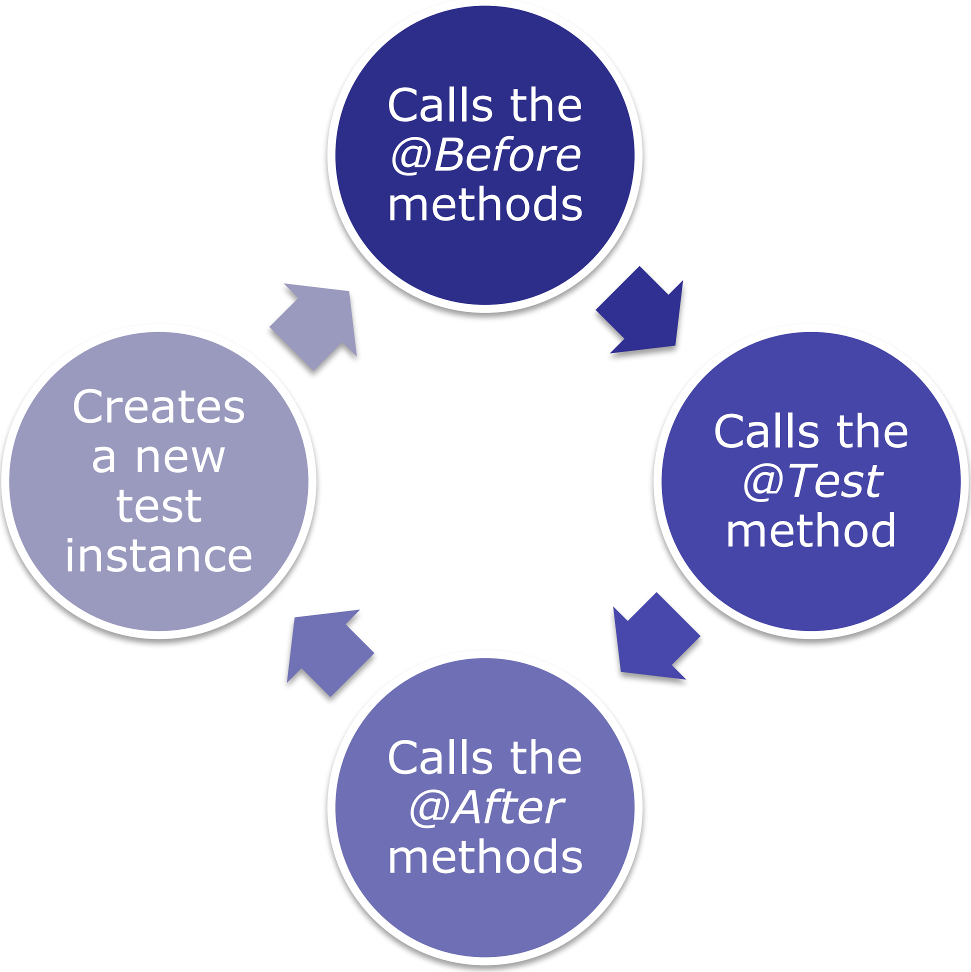
- Similar objects shared by several tests can be initialized and reclaimed using **`public void`** methods
- You should annotate the **`public void`** method with
  - *@BeforeClass* – run before any test has been executed
  - *@AfterClass* – run after all the tests have been executed
  - *@Before* – run before each test
  - *@After* – run after each test

"JUnit Second Design Goal" Achieved

The framework must help us lower the cost of writing tests by reusing code.

Each time you reuse the fixture, you decrease the initial investment made when the fixture was created

Share Similar Objects



Calls the *@Before* methods

Calls the *@Test* method

Calls the *@After* methods

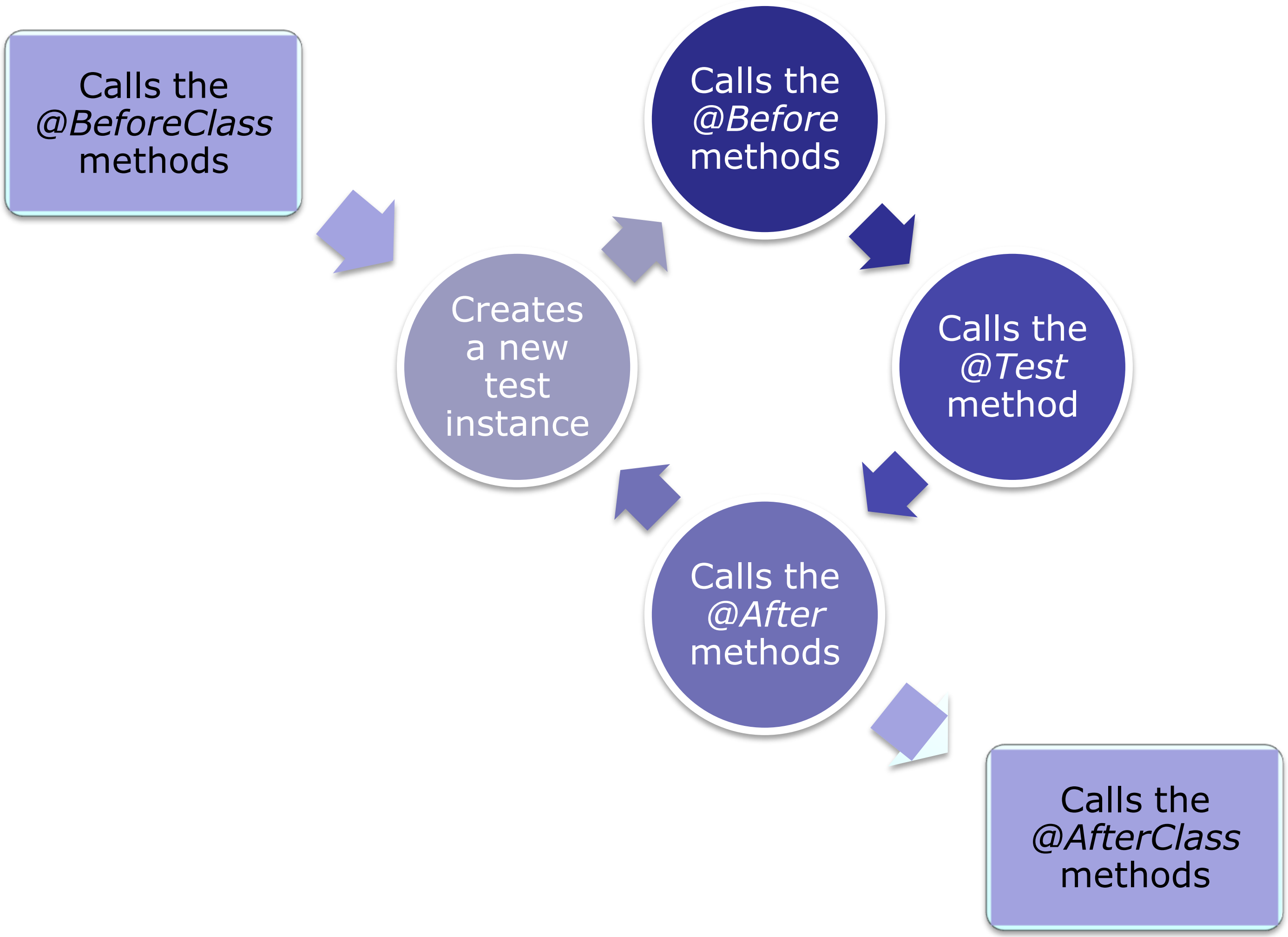
Creates a new test instance

Cognizant | Academy
Passion for making a difference

Share Similar Objects (Contd.)

```
public class TaxCalculatorImplTest {
   private TaxCalculatorImpl taxCalculator = null;
   @Before
   public void prepareTaxCalculator() {
        taxCalculator = new TaxCalculatorImpl();
   }
   @After
   public void cleanupTaxCalculator() {
        taxCalculator = null;
   }
   @Test
   public void shouldUseLowestTaxRateForIncomeBelow38000() {
        ...
   }
}
```

Share Expensive Setups


Calls the *@BeforeClass* methods
Creates a new test instance
Calls the *@Before* methods
Calls the *@Test* method
Calls the *@After* methods
Calls the *@AfterClass* methods

Share Expensive Setups (Contd.)

```
public class TaxCalculatorImplTest {
   private static TaxCalculatorImpl taxCalculator = null;
   @BeforeClass
   public static void initializeTaxCalculator() {
        taxCalculator = new TaxCalculatorImpl();
   }
   @AfterClass
   public static void releaseTaxCalculator() {
        taxCalculator = null;
   }
   @Test
   public void shouldUseLowestTaxRateForIncomeBelow38000() {
        ...
   }
}
```

@BeforeClass, @AfterClass
annotated methods must be **static**

Assert

- The assert methods are defined in **`org.junit.Assert`** class.

Method	Description
`assertTrue`	Asserts that a condition is true
`assertFalse`	Asserts that a condition is false
`assertEquals`	Asserts that two objects are equal
`assertNotNull`	Asserts that an object is not null
`assertNull`	Asserts that an object is null
`assertSame`	Asserts that two objects refer to the same object
`assertNotSame`	Asserts that two object don’t refer to the same object
`fail`	Fails a test with the given message

On failure, throws **`AssertionFailedError`**

Launching Tests

- Test runners are designed to *execute tests* and provide you with *statistics* regarding the outcome
- When a class is annotated with *@RunWith*, JUnit will invoke the class it refers to run the tests in that class
- **`JUnitCore`** is a facade for running tests. To run tests from the command line, run:

```
java –cp junit.jar org.junit.runner.JUnitCore AllTests
```

Failures vs Errors

Failures: *assert method* fails if the API contract cannot be fulfilled

Errors: These are unexpected condition that is not expected by the test.

Composing Tests Using Suite

- Default runner class scans the class for any methods that have *@Test* annotation
- Use *Suite.class* as a runner allows you to manually build a suite containing tests from many classes

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses ({TaxTest.class, BankAccountTest.class})

public class AllMyNewTests {

}
```

Composing Tests Using Suite

- JUnit 3.8.x-style test suites should use *AllTests.class* as a runner

```
import org.junit.runners.AllTests;

@RunWith(AllTests.class)
public class AllMyOldTests {
    public static Test suite() {  ... }
}
```

“JUnit Third Design Goal” Achieved

The framework must create tests that retain their value over time.

Combination of **`Runner`** and **`Suite`** makes it easy to run all tests, as well as, you can select a subset of tests that relate to the current development effort.

Demonstration

- Write unit tests using *@Test* annotation
- Share similar objects across tests by *@Before, @After annotation*
- Share computationally expensive setups by *@BeforeClass and @AfterClass* annotation
- Launch tests using *@RunWith* annotation
- Build test suite using *Suite.class* test runner

- Allow time for questions from participants

Test Your Understanding

- How do you test *protected* methods?
- How do you test *private* methods?
- How do you test a method that doesn’t return anything?
- Under what conditions should you test *get* and *set* methods?
- Why not just use *System.out.println* method instead of *assert* method?
- When should you write own test suite?

Jumpstart JUnit : Summary

- JUnit is a framework for unit tests initially developed by *Kent Beck and Erich Gamma*
- Design goals of JUnit is
  - Write useful tests
  - Create tests that retain value, and
  - To reuse code
- Write the test methods to test each discrete unit of work with *@Test* annotation
- Annotate the methods that create and destroy fixtures with *@Before, @After, @BeforeClass and @AfterClass*
- Use the `assert` methods to verify the behavior of the code being tested
- Run multiple suites using `Suite` runner in *@RunWith* annotation

Jumpstart JUnit : Source

- Books
  - JUnit Recipes: Practical Methods for Programmer Testing by *J. B. Rainsberger, Scott Stirling*
  - JUnit in Action by *Vincent Massol, Ted Husted*
- Web
  - Wiki: http://en.wikipedia.org/wiki/JUnit
  - JUnit: http://www.junit.org/

Disclaimer: Parts of the content of this course is based on the materials available from the Web sites and books listed above. The materials that can be accessed from linked sites are not maintained by Cognizant Academy and we are not responsible for the contents thereof. All trademarks, service marks, and trade names in this course are the marks of the respective owner(s).

Cognizant | Academy
Passion for making a difference

You have completed the Session 3 & 4 Jumpstart JUnit

Academy