Exploring JUnit 4.x

Targeted at: Entry Level Trainees

Session 12 & 14: Theories

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/1110/1.1

Cognizant Certified Official Curriculum

UNDER THE SEAL OF COGNIZANT ACADEMY

Cognizant | Academy
Passion for making a difference

Icons Used

Questions

Tools

Hands-on Exercise

Coding Standards

Test Your Understanding

Reference

Try it Out

A Welcome Break

Contacts

Theories: Overview

- **Introduction:**
  - A *test* captures the intended behavior in one particular scenario
  - A *theory* captures the intended behavior in possibly infinite numbers of potential scenarios
  - JUnit absorbed the support for theories from the *Popper* project
  - In this chapter, associates would learn how to write theory enabled tests

Theories: Objective

- **Objective:**

After completing this chapter, associates will be able to:

» Use *Theories.class* as the test runner

» Specify a set of data points using *@DataPoint* annotation

» Write generic test using *@Theory* annotation

» Use **`assume*`** methods to filter out data values in theory enabled test

» Explain the theory-enabled test execution cycle

Writing Theory Enabled Test

- A theory is a statement that is true for many data sets
- Creating a theory enabled test case requires:
  1. Data Points: Data to be injected into the theory methods according to their type
  2. Theories: Theories look like test methods, but are universally quantified; all assertions must hold for any possible parameter values that pass the assumptions

Write Datapoints

```
@DataPoint public static double INCOME_1 = 0;
@DataPoint public static double INCOME_2 = 1000;
...
@DataPoint public static double INCOME_15 = 60000;

@DataPoint public static int YEAR_2006 = 2006;
@DataPoint public static int YEAR_2007 = 2007;
@DataPoint public static int YEAR_2008 = 2008;

@Theory
public void incomeUpTo38000(double income, int year)
            throws InvalidYearException {
    ...
}
```

Write Datapoints (Contd.)

1. Test data are indicated by *@DataPoint* annotation
2. Datapoints are **`public static`** variables of different types

```
@DataPoint public static double INCOME_1 = 0;
@DataPoint public static double INCOME_2 = 1000;
@DataPoint public static double INCOME_3 = 5000;

@DataPoint public static int YEAR_2006 = 2006;
@DataPoint public static int YEAR_2007 = 2007;
@DataPoint public static int YEAR_2008 = 2008;
```

3. *@Datapoint* values are injected into the *@Theory* methods according to their type

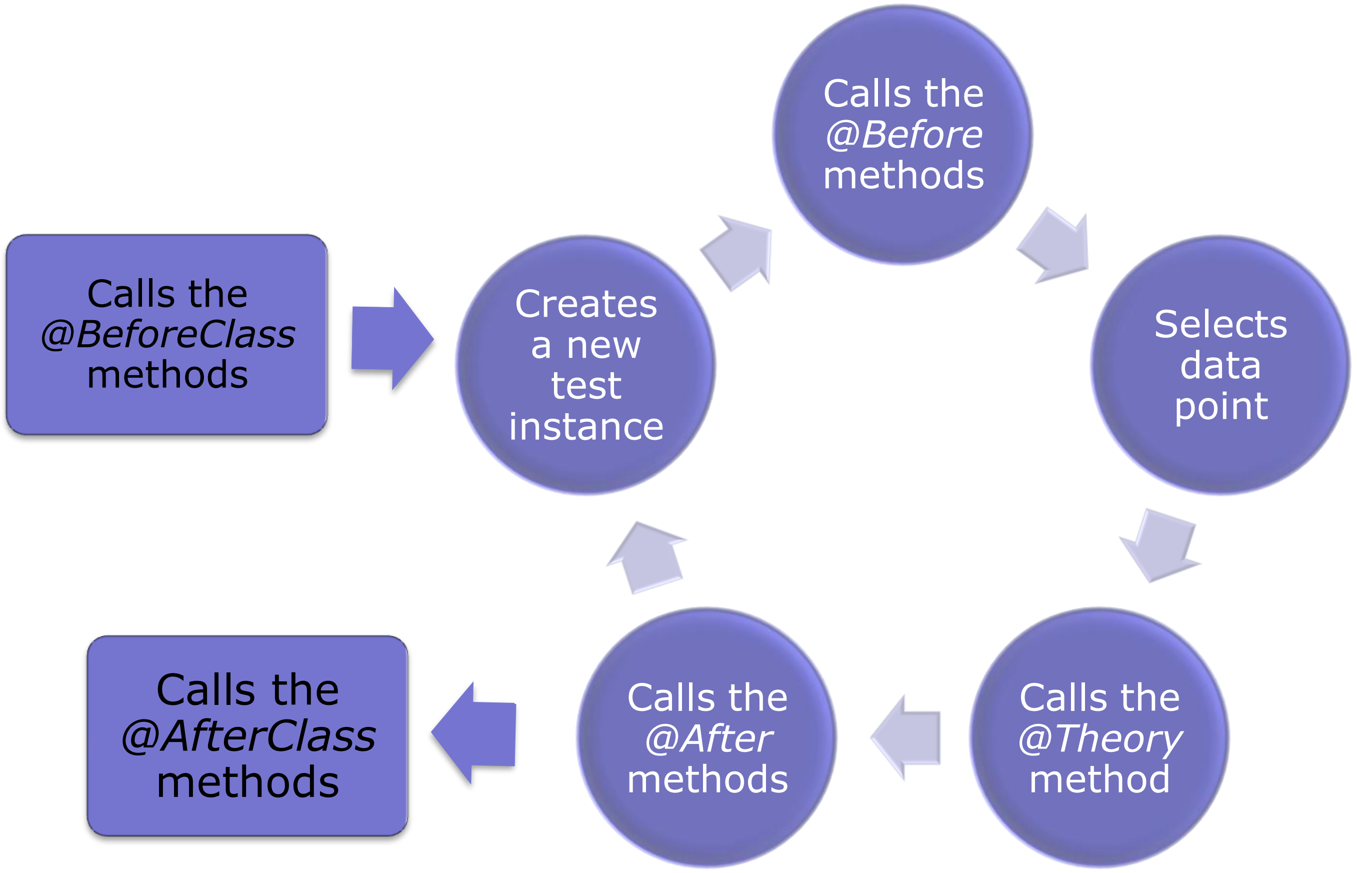
Write Theories

```
@RunWith(Theories.class)
public class TaxCalculationTheoryTest {
  ...

  @Theory
  public void incomeUpTo38000(double income, int year)
            throws InvalidYearException {
    assumeThat(year, anyOf(is(2007), is(2008)));
    assumeThat(income, lessThanOrEqualTo(38000.00));

    TaxCalculator calculator = new TaxCalculatorImpl();
    double calculatedTax = ...
    double expectedTax = income * 0.195;

    assertThat(expectedTax, is(calculatedTax));
 }
}
```

Write Theories(Contd.)

1. Specify the test case to be run with the *Theories.class* via the *@RunWith* annotation
2. Methods that test a theory are declared using *@Theory* annotation
3. A *@Theory* method has parameters, which is used to inject data
4. Theory methods filter out the test data values using **`assume*`** method

Test Execution Cycle


Calls the *@BeforeClass* methods
Creates a new test instance
Calls the *@Before* methods
Selects data point
Calls the *@Theory* method
Calls the *@After* methods
Calls the *@AfterClass* methods

Observations

- Junit will execute the below steps for each combination of corresponding datapoint values
  1. Creates a new instance of the test
  2. Calls the *@Before* annotated methods
  3. Injects the selected datapoint into the *@Theory* method
  4. Calls the *@After* annotated methods

Demonstration

- Use *Theories.class* as the test runner
- Specify a set of data points using *@DataPoint* annotation
- Write generic test using *@Theory* annotation
- Use **`assume*`** methods to filter out data values in theory enabled test

Cognizant | Academy
Passion for making a difference

- Allow time for questions from participants

Test Your Understanding

- Is the following theory declaration correct?

```
@Theory
public void saveEmployee() { … }
```

- Test case has 5 integer datapoints, 2 double datapoints, 3 String datapoints and a theory. Theory takes 2 integer parameters. How many instances would be created?
- What is the execution cycle of a theory enabled test case?

Theories: Summary

- A *test* captures the intended behavior in one particular scenario
- A *theory* captures the intended behavior in possibly infinite numbers of potential scenarios
- Creating a theory enabled test case requires
  - Theories
  - Datapoints
- Specifying the test case to be run with the *Theories.class* via the *@RunWith* annotation
- Methods that test a theory are declared using *@Theory* annotation
- Test data are **`public static`** variables indicated by *@DataPoint* annotation

Theories : Source

- Books:
  - JUnit Recipes: Practical Methods for Programmer Testing by *J. B. Rainsberger, Scott Stirling*
  - JUnit in Action by *Vincent Massol, Ted Husted*
  - Java Power Tools by *John Ferguson Smart*
- Web:
  - Wiki: http://en.wikipedia.org/wiki/JUnit
  - JUnit: http://www.junit.org/
  - Theories: http://www.markhneedham.com/blog/2008/12/12/junit-theories-first-thoughts/

Cognizant | Academy
Passion for making a difference

You have completed the Session 13&14 Theories

Academy