Exploring JUnit 4.x

Targeted at: Entry Level Trainees

Session 11: Parameterized Tests

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

Contacts

Session 11: Parameterized Tests overview

- **Introduction:**
  - » JUnit 4 comes with another special runner, *Parameterized*, which allows associates to run the same test with different data
  - » Parameterized test case run several sets of test data against the same test case
  - » It helps to reduce the number of unit tests to write and encourages developers to test more thoroughly
  - » In this chapter, associates would learn how to write parameterized tests

Session 11- Parameterized Tests: Objective

- **Objective:**

After completing this chapter, associates will be able to:

  - » Apply *Parameterized.class* as the test runner
  - » Write a feeder method using *@Parameters* annotation
  - » Write generic tests to support parameterized testing
  - » Explain the parameterized test execution cycle

Parameterize Test Case

- Step 1: To use a parameterized test case, associates need to use the **`org.junit.runners.Parameterized`** as the test runner

```
@RunWith(Parameterized.class)
public class TaxCalculationImplTest {


        ...
}
```

Parameters To Be Injected

- Step 2: To know which parameters to use, the test case needs a **`public static`** method that returns a **`Collection`** of **`Object`** array annotated with *@Parameters*

```
@RunWith(Parameterized.class)
public class TaxCalculationImplTest {

    @Parameters
    public static Collection<Object[]> data() {
        return Arrays.asList(
                new Object[][]{
                        {0.00, 2006, 0.00},
                        ...
                });
    }
}
```

Setting The Parameters

- Step 3: Associates need a **`public`** constructor that takes the parameters and sets it to the class member variables

```
public class TaxCalculationImplTest {
    ...
    private double income;
    private int year;
    private double expectedTax;

    public TaxCalculationImplTest(double income,
             int year, double expectedTax) {
        this.income = income;
        this.year = year;
        this.expectedTax = expectedTax;
    }
}
```

Write Test Method

- Step 4: Write unit tests that use the member variables to check the tested class

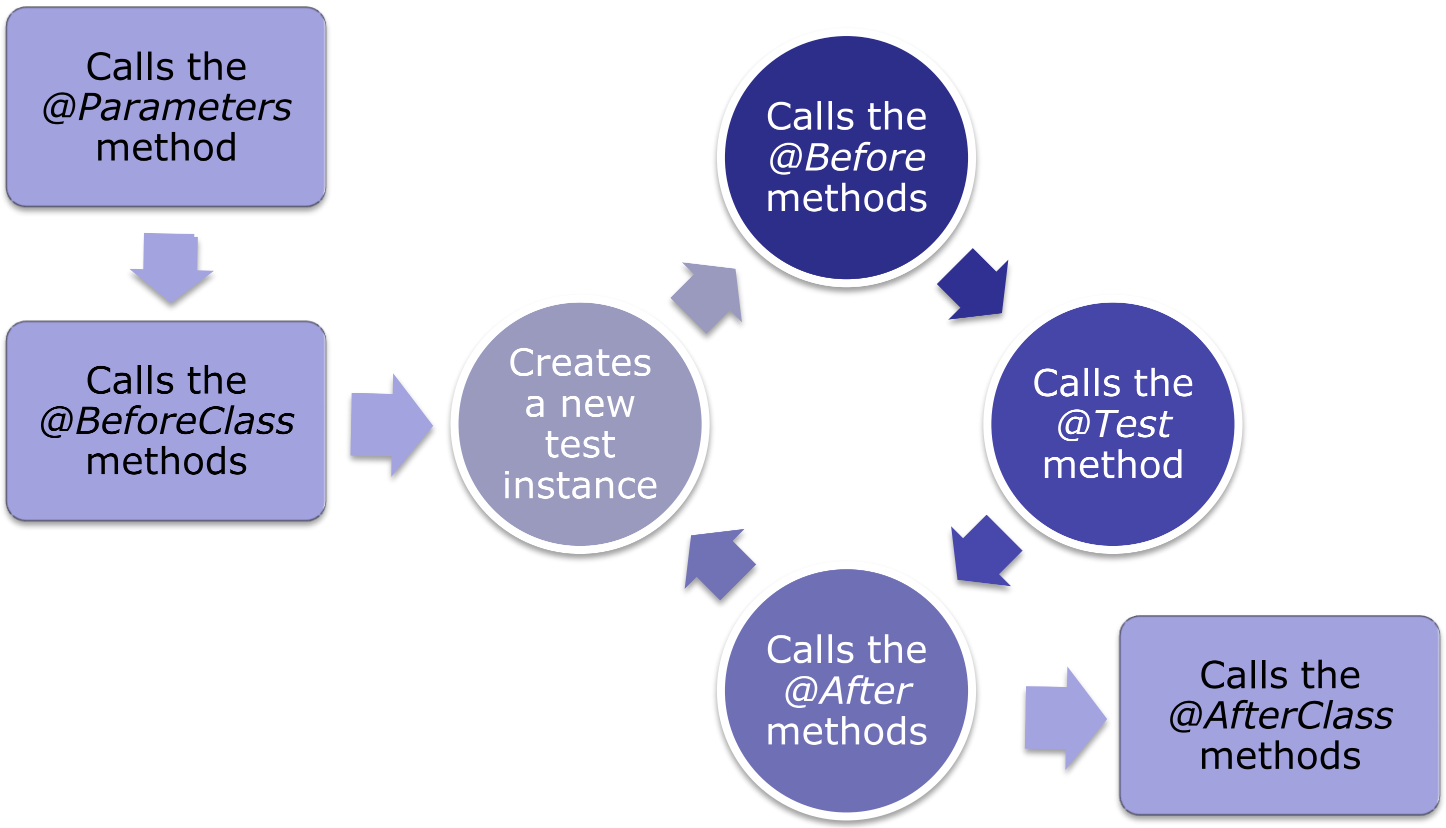
```
private double income;
private int year;
private double expectedTax;

@Test
public void shouldCalculateCorrectTax()
      throws InvalidYearException {

   TaxCalculator calculator = new TaxCalculatorImpl();
   double calculatedTax =
      calculator.calculateIncomeTax(income, year);
   assertEquals(expectedTax, calculatedTax, 0.0);
}
```

Cognizant | Academy
Passion for making a difference

Writing A Parameterized Test

- Creating a parameterized test case requires:
  1. Specify the test case to be run with the *Parameterized.class* via the *@RunWith* annotation
  2. Create a **`static`** feeder method that returns a **`Collection`** type with test data and decorate it with the @*Parameters* annotation
  3. Create class members for the parameter types required in the generic test
  4. Create a constructor that takes these parameter types and sets them to the class members
  5. Create a generic test and decorate it with @*Test* annotation

Test Execution Cycle


Calls the *@Parameters* method
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

Observations

- Parameterized test runner calls the `public static` method to obtain the `Collection` of test data
- For each `Object` array in the `Collection`, a new instance of the enclosing class is created
- JUnit calls all the test methods on the new instance

Non-parametric tests

It is better not to have non-parametric tests within your parameterized class.

Demonstration

- Use *Parameterized.class* as the test runner
- Write a feeder method using *@Parameters* annotation
- Write an argument constructor that takes the parameters and sets it to the class member variables
- Write generic tests to support parameterized testing

- Allow time for questions from participants

Cognizant | Academy
Passion for making a difference

Test Your Understanding

- What is the signature of feeder method?
- How many feeder methods can a parametric test case have?
- Can a test case have both non-parametric and parametric test methods?
- What is the execution cycle of a parametric test case?

Parameterized Tests -Session 11: Summary

- Parameterized test case run several sets of test data against the same test case
- To use a parameterized test case, associates need to use the `org.junit.runners.Parameterized` as the test runner
- The test case needs a **`public static`** method that returns a **`Collection`** of **`Object`** array annotated with *@Parameters*
- A **`public`** constructor that takes the parameters and sets it to the class member variables
- Write unit tests that use the member variables to check the tested class

Parameterized Tests Session 11: 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/
  - Test Early: http://www.testearly.com/2007/04/13/take-heed-of-mixing-junit-4s-parameterized-tests/

Cognizant | Academy
Passion for making a difference

You have completed the Session 11 Parameterized Tests

Academy