UNIT TESTING WITH JUNIT

Software Engineering Class

Prof. Adriano Peron

Dec, 6th 2013

Valerio Maggio, Ph.D.
valerio.maggio@unina.it

BRIEF INTRO TO THE CLASS

It's about Unit Testing

+

It's about JUnit

It's about ...?

It's about
Unit Testing with JUnit

Make a guess....

You say?!

DISCLAIMER

This is not a **Tutorial Class**

- At the end of the class you (should)...

1. ..have learnt something more about unit testing;

2. ..have learnt **what** is JUnit, **how** to use it and **when**;

3. ..have realized how much important are *testing activities*!

(maybe you *already noticed* that slides are in English...)

JUNIT PRELIMINARIES

- **Q**: How many "types" of testing do you know?
  - **A**: System Testing, Integration Testing, Unit Testing....

- **Q**: How many "testing techniques" do you know?
  - **A**: Black Box and White Box Testing

    Which is the difference?

- **Q**: What type and technique do you think JUnit covers?

JUNIT WORDS CLOUD

a.k.a. *some random words (almost) related to JUnit*

Testing

xUnit

Java

Testing framework

Unit Testing

Test Suite

Black Box Testing

Testing Automation

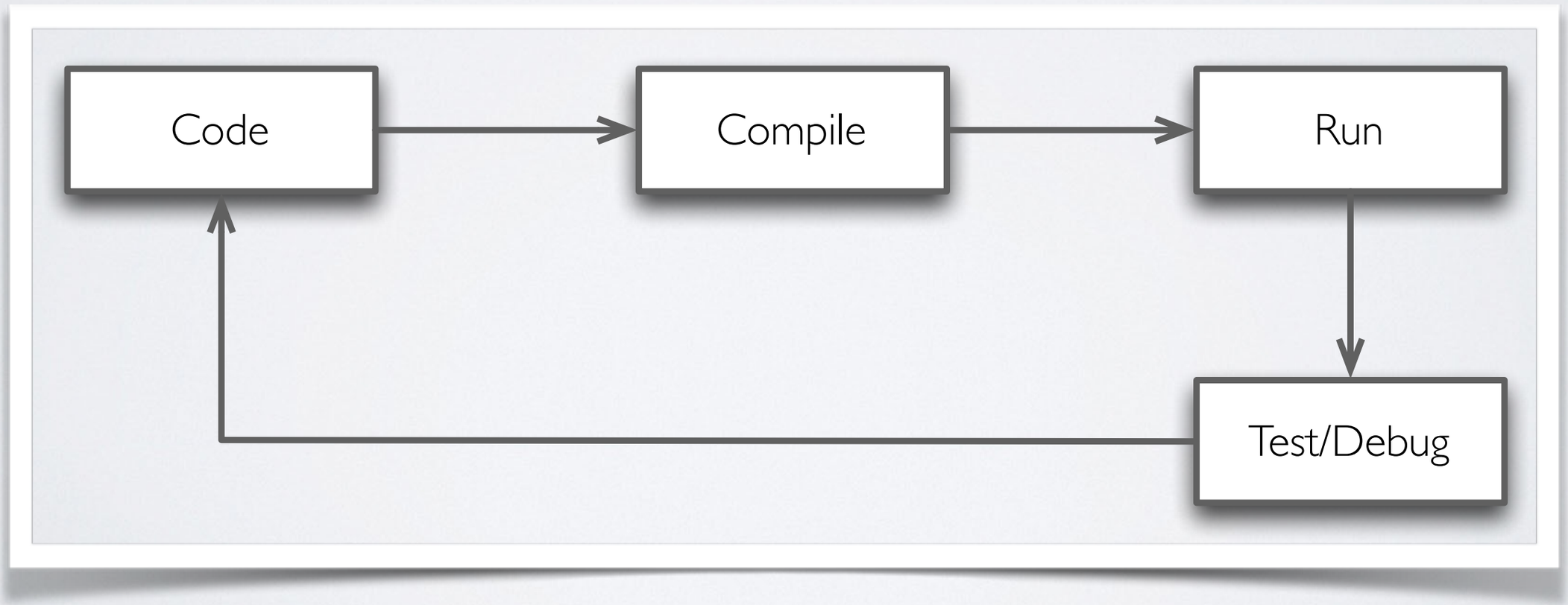
Test Fixtures

Simple Test Program

Test Runners

THE IMPORTANCE OF TESTING

- During development, the first thing we do is run our own program
  - This is *(**sometimes**)* called *Acceptance Testing*



JUnit: *a.k.a.* A programmer-oriented testing framework for Java

JUnit is a simple framework to write repeatable tests. It is an instance of the **xUnit** architecture for unit testing frameworks.

UNIT TEST

Unit Test: A unit test examines the behaviour of a distinct *unit of work*.

The "distinct unit of work" is often (*but not always*) a single **class**.

Unit tests run **fast**! (*If they don't, they're **not** Unit tests*)

Unit tests run **in isolation**! (*no dependencies*)

why UNIT TESTING

- Unit tests ensure that code works **as intended**.
- (**Also**) very helpful to ensure that the code still works as intended in case you need to modify code
  - e.g., *fixing* a bug or *extending* functionality.
- **Test Coverage:** the *percentage of code* tested by unit tests
  - Test coverage should be as high as possible. why?
- **Test Fixture:** a fixed state of the software under test used as a baseline for running tests

Other TYPES OF TESTING

- **Integration Tests:** test the **behaviour** of a component;
  - (or) the **integration** between a set of components.
  - *(Sometimes)* Called *Functional Tests*
  - These kind of tests would resemble a expected user interaction with the application.
- **Performance Tests:** measure performance of software components in a **repeatable** way.

JUnit: *a.k.a.* A programmer-oriented testing framework for Java

JUnit is a simple framework to write repeatable tests. It is an instance of the **xUnit** architecture for unit testing frameworks.

FRAMEWORK

*A **framework** is a semi-complete application that provides a **reusable**, common structure that can be shared between applications.*

*Developers **incorporate** the framework in their own application and extend it to meet their specific needs.*

FRAMEWORK *(in other words)*

- A framework provides a working infrastructure to be used for a specific task
    - i.e., the **same** task the framework was designed for.
- Developers integrate the framework by:
    - adopting the **design rules** of the framework (in **design**)
    - Importing, subclassing, …extending framework functionalities (in **code**)
    - Btw, they avoid writing **boilerplate code**

WHY A FRAMEWORK IS NEEDED?

Let's do a very dummy example...

```
public class Calculator {

    public double add(double number1, double number2) {
        return number1 + number2;
    }
}
```

Q: How would you test this method?

VERY SIMPLE TESTING STRATEGY

```
public class TestCalculator {

    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        double result = calculator.add(10,50);
        if (result != 60){
            System.out.println("Bad result: " + result);
        }// end if
    }//end main
}
```

Q: How would you improve it?

IMPROVED (Naive) SOLUTION

```
public class TestCalculator {
    private int nbErrors = 0;

    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        if (result != 60) {
            throw new RuntimeException("Bad result: " + result);
        }
    }

    public static void main(String[] args) {

        TestCalculator test = new TestCalculator();
        try {
            test.testAdd();
        } catch (Throwable e) {
            test.nbErrors++;
            e.printStackTrace();
        }

        if (test.nbErrors > 0) {
            throw new RuntimeException("There were " + test.nbErrors +
                                       " error(s)");
        }
    }// end main
}
```

LESSON LEARNED

Objective Test **+** Repeatable Test **=** Simple Test Program

Beware:
Previous Code showed a
naive way of testing
(a.k.a. the wrong one)

That was **not** JUnit!!

JUNIT'S *way*

```java
import static org.junit.Assert.assertEquals;

import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.JUnit4;

@RunWith(JUnit4.class)
public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals("10 + 50 must be 60", 60, result);
    }
}
```

Q: Do you already know these constructs (@) (in **Java**)?

JUNIT (*main*) FEATURES

- **JUnit** features include:
  - **Assertions** for testing expected results
  - **Test fixtures** for sharing common test data
  - **Test runners** for running tests

TESTCALCULATOR JUNIT 4

```java
public class Calculator {

    public double add(double number1, double number2) {
        return number1 + number2;
    }
}
```

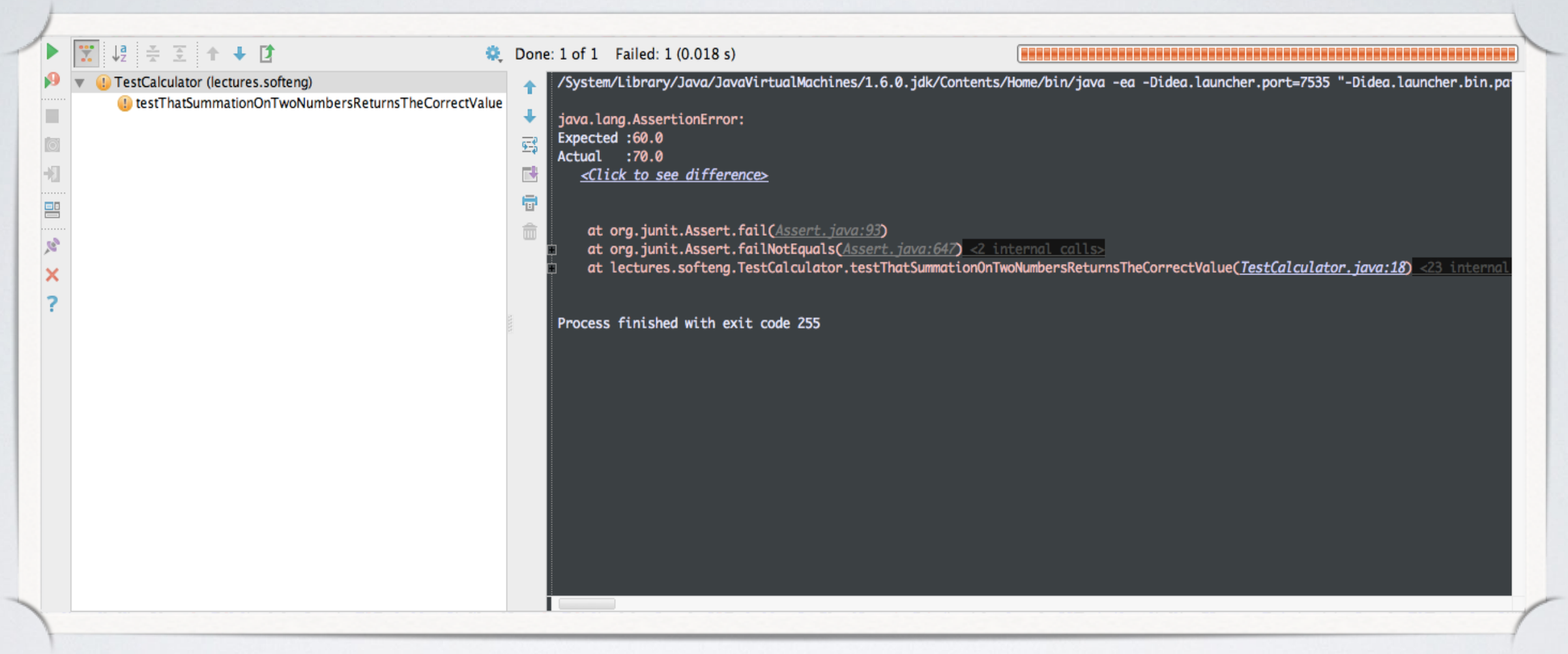
```java
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class TestCalculator {

    @Test
    public void testThatSummationOnTwoNumbersReturnsTheCorrectValue(){
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 60);
        assertEquals(60, result, 0);
    }
}
```

TestAnnotation

JUnit Assert

THE TEST FAILS!

(as expected)



IDE: IntelliJ IDEA 12 CE

http://www.jetbrains.com/idea/

JUNIT 4.X DESIGN

- Main features inspired from other Java Unit Testing Frameworks
  - **TestNG**
- Test Method **Annotations**
  - Requires Java5+
- Main Method Annotations
  - `@Before, @After`
  - `@Test, @Ignore`
  - `@SuiteClasses, @RunWith`

JUNIT **TEST** ANNOTATIONS

- `@Test public void testMethodName()`
  - Annotation `@Test` identifies that `testMethodName` is a **test** method.
- `@Before public void beforeMethodName()`
  - `beforeMethodName()` will be executed **before** each test.
    - This method can prepare the test environment
      - e.g., read input data, initialise the class, …
    - Traditionally named `setUp()` **(for "historical" reasons)**
- `@After public void afterMethodName()`
  - `afterMethodName()` will be executed **after** each test.
  - Traditionally named `tearDown()` **(for "historical" reasons)**

JUNIT ANNOTATIONS (2)

- `@Ignore`
  - Will ignore the test method
  - E.g. Useful if the underlying code has been changed and the test has not yet been adapted.

- `@Test(expected=Exception.class)`
  - Tests if the method throws the named exception.

- `@Test(timeout=100)`
  - Fails if the method takes longer than 100 milliseconds.

`java.lang.Annotation` AT GLANCE

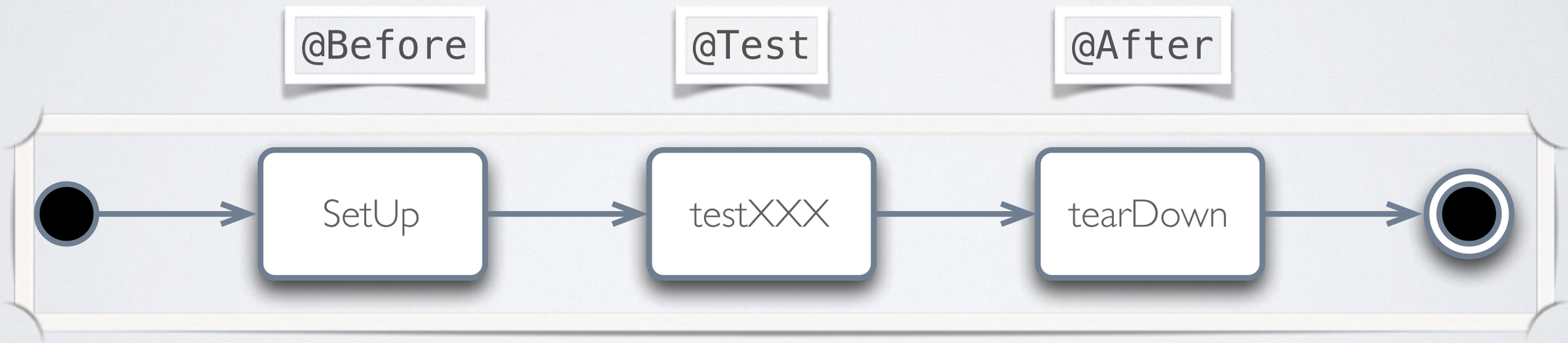
- Meta Data Tagging (package `java.lang.annotation`)
- **Target** (`java.lang.annotation.ElementType`)
  - Specify to which element the annotation is applied
    - `FIELD`
    - `METHOD`
    - `CLASS`
- **Retention**
  - Specify how long annotation should be available

@TEST ANNOTATION

```
package org.junit;

@java.lang.annotation.Retention(java.lang.annotation.RetentionPolicy.RUNTIME)
@java.lang.annotation.Target({java.lang.annotation.ElementType.METHOD})
public @interface Test {
    java.lang.Class<? extends java.lang.Throwable> expected() default org.junit.Test.None.class;

    long timeout() default 0L;

    static class None extends java.lang.Throwable {
        private static final long serialVersionUID = 1L;

        private None() { /* compiled code */ }
    }
}
```

TEST FIXTURES

fixture: *The set of common resources or data that you need to run one or more tests.*



JUNIT **ASSERT** STATEMENTS

- `assertNotNull([message], object)`
  - Test passes if Object is not null.
- `assertNull([message], object)`
  - Test passes if Object is null.
- `assertEquals([message],expected, actual)`
  - Asserts equality of two values
- `assertTrue(true|false)`
  - Test passes if condition is True
- `assertNotSame([message], expected, actual)`
  - Test passes if the two Objects are not the same Object
- `assertSame([message], expected, actual)`
  - Test passes if the two Objects are the same Object

TESTING EXCEPTION HANDLING

try-catch trick!

```
import org.junit.TestCase;

public class TestCalculator extends TestCase {

    public void testThatSummationRaisesAnExceptionOnNegativeInputNumbers(){

        try {
            Calculator calculator = new Calculator();
            calculator.add(-1, -3);
            this.fail(); // Fail the test if no exception has been thrown!
        } catch (RuntimeException) {
            this.assertTrue(True); // Pass the test! ;)
        }
    }
}
```

TESTING THE EXCEPTION HANDLING THE NEW WAY!

Use the **`expected`** parameter of **`@Test`** annotation

```
import org.junit.Test;

public class TestCalculator {

    @Test(expected=RuntimeException.class)
    public void testThatSummationRaisesAnExceptionOnNegativeInputNumbers(){
        Calculator calculator = new Calculator();
        calculator.add(-1, -3);
    } // This is very short, isn't it?!
}
```

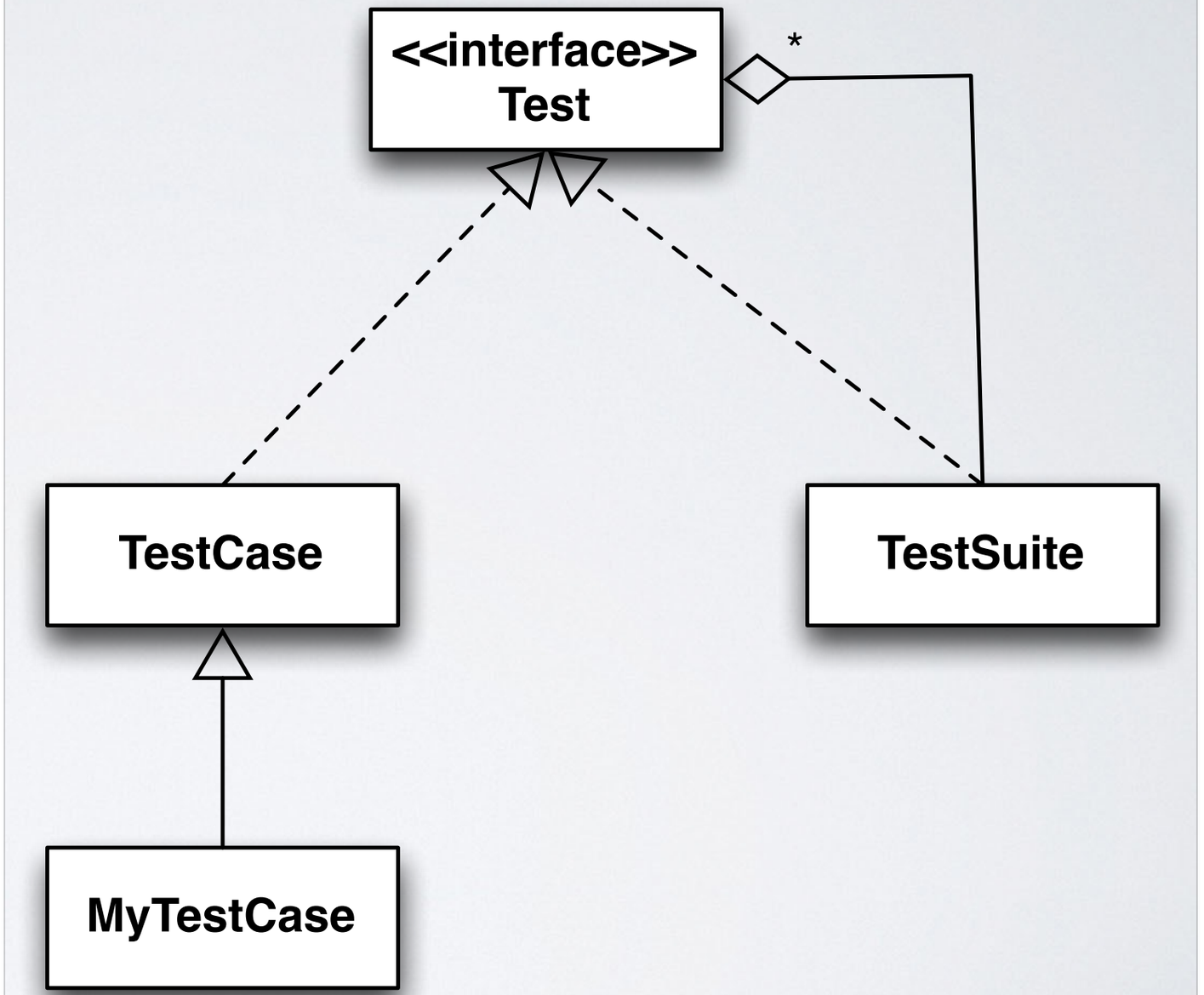
A look at the past: JUNIT 3.X

- All the Test classes **must extend** `junit.framework.TestCase`
  - Functionalities by inheritance
- All the test method's names **must** start with the "keyword" **test** in order to be executed by the framework
  - `testSomething()`
  - `testSomethingElse()`

xUNIT DESIGN

JUnit 3.x design was compliant with xUnit framework guidelines

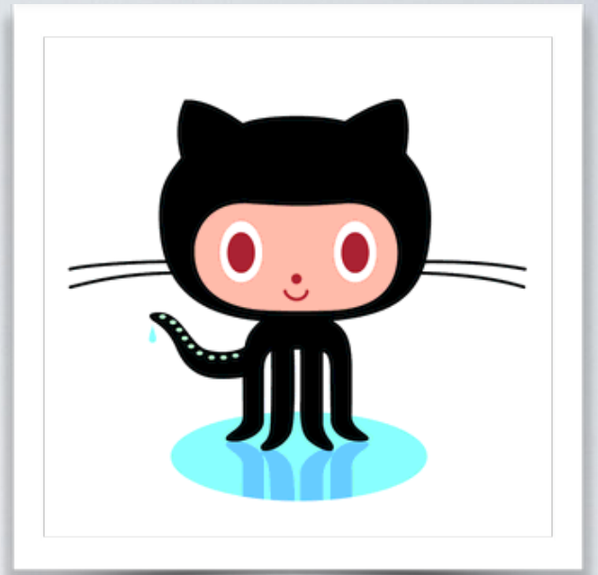
- JUnit
- CppUnit
- PyUnit
- NUnit
- XMLUnit
- PHPUnit
- RUnit
- SUnit
- .....



(vintage !-)
JUNIT TEST EXAMPLE

```
import junit.framework.TestCase;

public class TestCalculator extends TestCase {

    public void testAdd() {
        Calculator calculator = new Calculator();
        double result = calculator.add(10, 50);
        assertEquals(60, result, 0);
    }
}
```

GET THE CODE



PUBLIC junit-team / junit — Watch 272 | Star 2,126 | Fork 676

A programmer-oriented testing framework for Java.

1,697 commits | 3 branches | 15 releases | 58 contributors

branch: master — junit /

Merge pull request #764 from ferstl/classrule-in-public-class

dsaff authored 15 days ago — latest commit 1029290bc9

.settings	restored settings files to try to make git happy	a month ago
build	Delete .cvsignore files, because we're using Git now.	4 months ago
doc	added ReleaseNotes for 4.5	a month ago
lib	Add Hamcrest source JAR for easy reference	a year ago
src	Merge pull request #764 from ferstl/classrule-in-public-class	15 days ago
.classpath	Add Hamcrest source JAR for easy reference	a year ago
.gitignore	restored .gitignore to make git happy	a month ago
.project	Added TemporaryFile and ExternalResource interceptors	5 years ago
BUILDING	updated references from KentBeck github account to junit-team	10 months ago
CODING_STYLE	Issue #426 Added suggestions by dsaff	a year ago
CONTRIBUTING.md	Make numbered list format correctly in markdown	24 days ago
LICENSE.txt	Relicense JUnit from CPL to EPL	7 months ago
NOTICE.txt	NOTICE.txt	11 months ago
README.md	updated references from KentBeck github account to junit-team	10 months ago
acknowledgements.txt	updated references from KentBeck github account to junit-team	10 months ago
build.xml	Fix path to JavaDoc style sheet.	4 months ago
build_tests.sh	Fixes gh-309: build.xml uploads junitX.jar as junit-depX.jar	2 years ago
done.txt	Patched javadoc, thanks to Matthias Schmidt	8 years ago
epl-v10.html	Relicense JUnit from CPL to EPL	7 months ago

Code | Issues 155 | Pull Requests 17 | Wiki | Pulse | Graphs | Network

HTTPS clone URL

https://github.com/

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop

Download ZIP

github.com/junit-team/junit

- **Requirements:**
  - Input numbers cannot have more than 5 digits;
  - The calculator can remember a given (unique) number;
  - Only non-negative numbers are allowed.
  - In case of negative numbers, an exception is thrown!

Calculator
- memory: double + MAX_DIGITS_LEN: int = 5 <<final>> <<static>>
+ add (double augend, double addend): double + subtract (double minuend, double subtrahend): double + multiply (double multiplicand, double multiplier): double + divide (double dividend, double divisor): double + addToMemory(double number): void + recallNumber(): double

Some REFERENCES

Support free tutorials:

Donate

by Lars Vogel

Unit Testing with JUnit - Tutorial

Lars Vogel

Version 2.4

28.10.2013

Revision History			
Revision 0.1 - 2.4	03.09.2007 - 28.10.2013	Lars Vogel	created, bugfixes and enh

JUnit

This tutorial explains unit testing with JUnit 4.x. It explains the creation of JUnit tests and hc or via own code.

Table of Contents

vogella.com/articles/JUnit/article.html

github.com/junit-team/junit/wiki

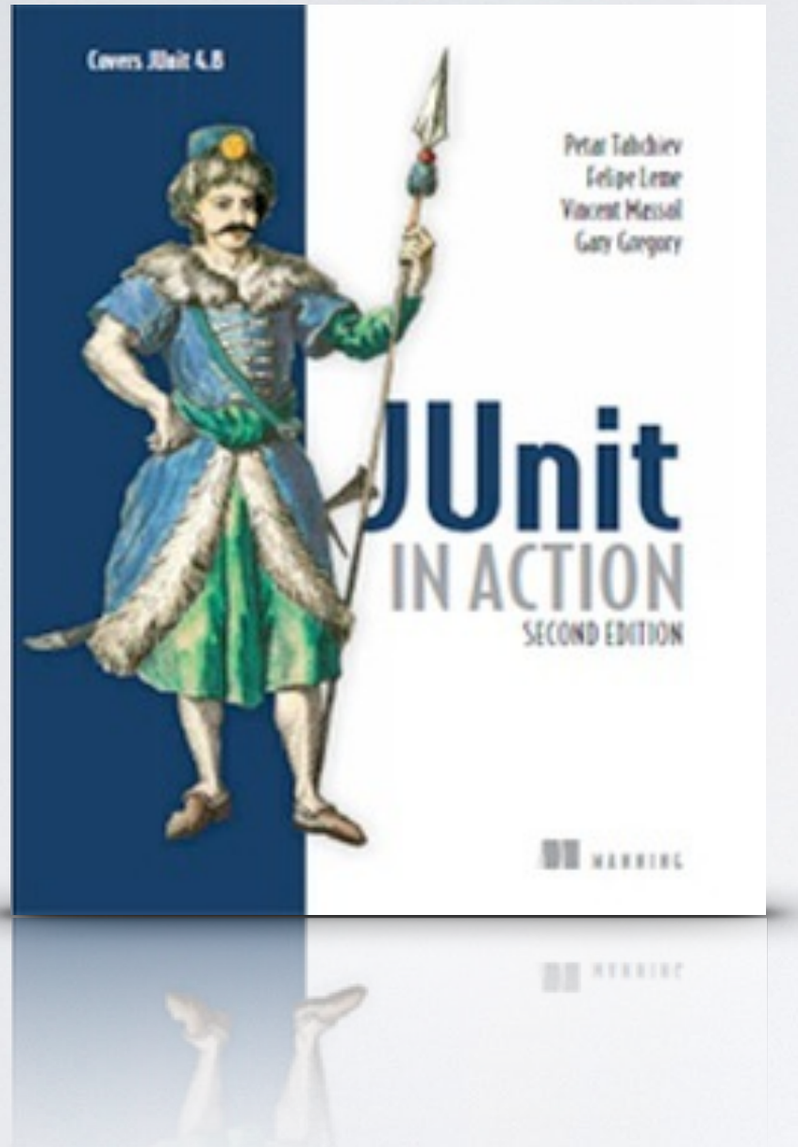
JUnit Usage and Idioms

- Assertions
- Test Runners
- Aggregating tests in Suites
- Test Execution Order
- Exception Testing
- Matchers and assertThat
- Ignoring Tests
- Timeout for Tests
- Parameterized Tests
- Assumptions with Assume
- Rules
- Theories
- Test Fixtures
- Categories
- Use with Maven
- Multithreaded code and Concurrency
- Java contract test helpers
- Continuous Testing

Third-party extensions

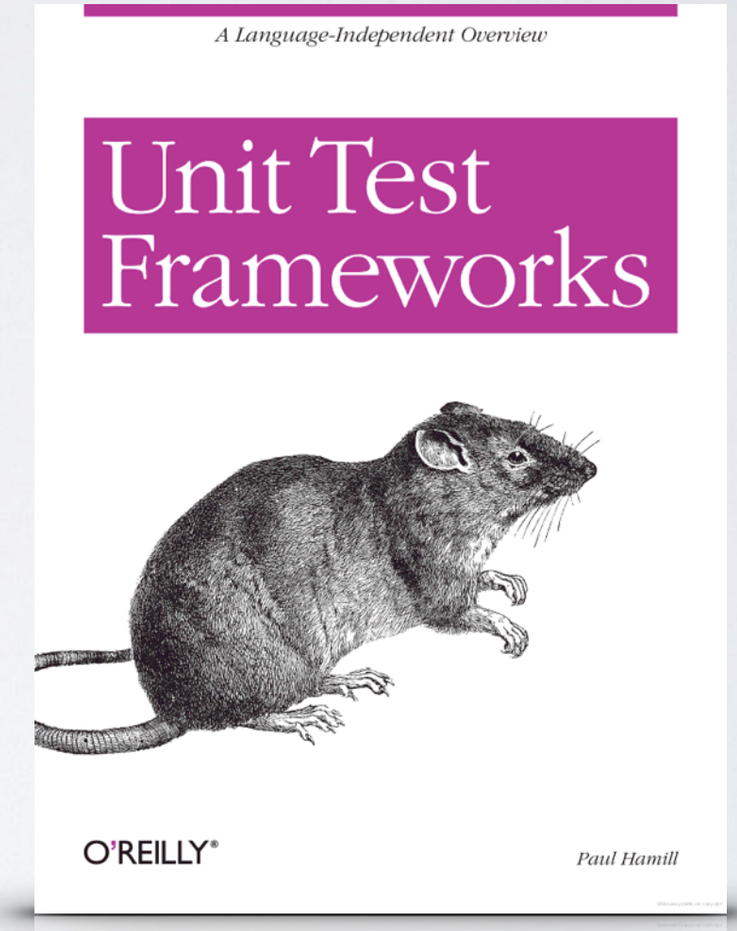
- Custom Runners
- net.trajano.commons:commons-testing for UtilityClassTestUtil per #646
- System Rules – A collection of JUnit rules for testing code that uses java.lang.System.
- JUnit Toolbox - Provides runners for parallel testing, a `PoolingWait` class to ease asynchronous testing, and a `WildcardPatternSuite` which allow you to specify wildcard patterns instead of explicitly listing all classes when you create a suite class.
- junit-quickcheck - QuickCheck-style parameter suppliers for JUnit theories. Uses junit.contrib's version of the theories machinery,

Some REFERENCES



JUnit in Action 2nd Ed.
Massol et al., Manning Pubs 2009

Unit Test Frameworks
Tools for High-Quality Software Development
Paul Hamill, O'Reilly Media 2004



CONTACTS

http://wpage.unina.it/valerio.maggio

MAIL:

valerio.maggio@unina.it