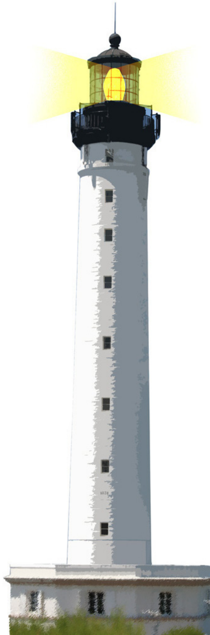
Learning Object-Oriented Programming and Design with TDD

Unit testing

S. Ducasse

http://stephane.ducasse.free.fr

http://www.pharo.org



Objectives

- Core concepts
- Examples of Unit tests
- In Java, C#, PHP and Pharo

Thanks Alexandre Bergel for parts of the materials used in this lecture!

First: Looking at a Test

In a test, we

- Create a context: Create an empty set
- Send a stimulus: Add twice the same element
- Check results: Check that the set contains only one element

Set TestCase In Pharo

```
TestCase subclass: # SetTest
  ...
```

```
SetTest >> testAdd
 | empty |
 empty := Set new.   "Context"
 empty add: 5.   "Stimulus"
 empty add: 5.
 self assert: empty size equals: 1.   "Check"
```

Counter in Java (JUnit 40)

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

class SetTest {

  @Test
  public void testAdd() {
      Set empty = new Counter(); //Context
      empty.add(5); //Stimulus
      empty.add(5);
      assertEquals(empty.size(),1); //Check
   }
}
```

Another example: CounterTest

```
TestCase subclass: #CounterTest
   ...
```

```
CounterTest >> testIncrement
  | counter |
  counter := Counter new.   "Context"
  counter value: 22.
  counter increment.   "Stimulus"
  counter increment.
  self assert: count value equals: 24.   "Verification"
```

Counter in Java (JUnit 40)

```
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.assertEquals;

class CounterTest {

  @Test
  public void testIncrement() {
      Counter count = new Counter();
      count.setValue(22);
      count.increment();
      count.increment();
      assertEquals(count.value(),24 );
   }
}
```

Summary: A Test

In a test, we

- Create a context
- Send a stimulus
- Check results

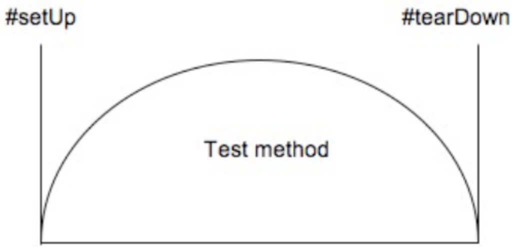
Success, Failures and Errors

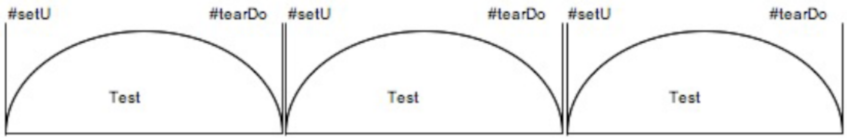
- Success: a test passes
- A failure is a failed assertion, i.e., an anticipated problem that you test.
- An error is a condition you didn't check for, i.e., a runtime error.

setUp and tearDown Messages

Executed systematically before and after each test run

- setUp allows us to specify and reuse the context
- tearDown to clean after





Defining a setUp Method

```
SetTestCase >> setUp
 empty := Set new
```

setUp is executed for you before any test execution

```
SetTestCase >> testOccurrences
 self
  assert: (empty occurrencesOf: 0)
  equals: 0.
 empty add: 5; add: 5.
 self
  assert: (empty occurrencesOf: 5)
  equals: 1
```

setUp in Java

```
@Before
public void setUp(){
  empty = new Set();
  }
```

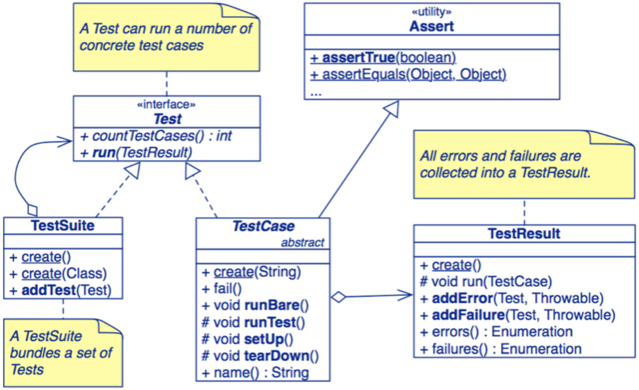
setUp is executed for you before any test execution

```
class CounterTest {

 @Test
 public void testOccurrences() {
   assertEquals(count.occurrencesOf(0), 0);
   empty.add(5);
   empty.add(5);
   assertEquals(count.occurrencesOf(5), 1);
}
```

Core framework graphically

- TestCase: a single test
- TestSuite: a groupe of tests
- TestResult: to represent
- TestResources: how to set up a context for a complete suite



Looking at Unit Framework variations

- All the frameworks are coming from SUnit
- Same concepts
  - TestCase
  - TestSuite
  - setUp/tearDown
- Passing, failures, errors

SUnit -> JUnit3

- Originally developed by K. Beck (agile programming father)
- Extremely simple (4 classes)
- Got copied all over the places: JUnit3.x, PHPUnit,...

JUnit 3.x is similar to SUnit 3.0/Pharo

Define a subclass of TestCase + setUp + testAdd method

```
import junit.framework.*;
public class MoneyTest extends TestCase {
  private Money f12CHF;       // fixtures
  private Money f14CHF;

  protected void setUp() {     // create the test data
    f12CHF = new Money(12, "CHF");
    f14CHF = new Money(14, "CHF");
  }
  public void testAdd() {          // create the test data
    Money expected = new Money(26, "CHF");
    assertEquals("amount not equal",
                expected, f12CHF.add(f14CHF));
  }
  ...
}
```

PHPUnit is close to JUnit 3/Pharo

```
<?php
class MoneyTest extends PHPUnit_Framework_TestCase
{
  // ...
  public function testCanBeNegated()
  {
    // Arrange
    $a = new Money(1);

    // Act
    $b = $a->negate();

    // Assert
    $this->assertEquals(-1, $b->getAmount());
  }
  // ...
}
```

In Ruby

```
# File:  tc_simple_number2.rb

require_relative "simple_number"
require "test/unit"

class TestSimpleNumber < Test::Unit::TestCase

  def test_simple
    assert_equal(4, SimpleNumber.new(2).add(2) )
    assert_equal(4, SimpleNumber.new(2).multiply(2) )
  end

  def test_typecheck
    assert_raise( RuntimeError ) { SimpleNumber.new('a') }
  end

end
```

JUnit 4 is based on annotations

- J2SE 5 introduced the Metadata feature
- Annotations allow you to add decorations to your code (remember javadoc tags: @author)
- Annotations are used for code documentation, compiler processing (@Deprecated), code generation, runtime processing

`http://java.sun.com/docs/books/tutorial/java/javaOO/annotations.html`

JUnit 4.x

- Annotations for marking methods as tests (@Test)
- Annotations for marking methods that setting up and cleaning up “fixtures” (@Before)
- methods for making assertions assertEquals()

JUnit 4.x Example Code

```
import org.junit.*;
import static org.junit.Assert.*;
public class MoneyTest {
  private Money f12CHF;
  private Money f14CHF;

  @Before public void setUp() {  // the fixture
    f12CHF = new Money(12, "CHF");
    f14CHF = new Money(14, "CHF");
  }

  @Test public void add() {
    Money expected = new Money(26, "CHF");
    assertEquals("amount not equal",
              expected,f12CHF.add(f14CHF));
  }
  ...
}
```

In CSharp similar idea

```
[TestMethod]
public void Withdraw_ValidAmount_ChangesBalance()
{
   double currentBalance = 10.0;  // fixture
   double withdrawal = 1.0;
   double expected = 9.0;
   var account = new CheckingAccount("JohnDoe", currentBalance);
   // stimulus
   account.Withdraw(withdrawal);
   double actual = account.Balance;
   // assertions
   Assert.AreEqual(expected, actual);
}
```

JUnit 5 :): the same but different

- Before -> BeforeEach , Ignore -> Disabled, BeforeClass -> BeforeAll
- JUnit 4 has everything bundled into single jar file.
- JUnit 5 is composed of 3 sub-projects
  - JUnit Platform. It defines the TestEngine API for developing new testing frameworks that runs on the platform.
  - JUnit Jupiter. has all new junit annotations and TestEngine implementation to run tests written with these annotations.
  - JUnit Vintage, To support running JUnit 3 and JUnit 4 written tests on the JUnit 5 platform.

The testing framework is not that importan, the tests are... Eagerly wanting for new old super exciting features :).

3 Testing practices

- During dev, write tests first
  - Specify what you want
  - You are done when the tests run
- When you redesign/improve your software
  - refactor in small steps and
  - run the tests to stop any regression
  - fix what is broken (get the bar green)
- During debugging
  - write a test that demonstrates the bug
  - then fix it.

What you should know

- What is a unit test
- Writing a test
- What is pass, failed, errors
- Tests are your best friends against bugs and evolution

A course by Stéphane Ducasse
http://stephane.ducasse.free.fr

Reusing some parts of the Pharo Mooc by

Damien Cassou, Stéphane Ducasse, Luc Fabresse
http://mooc.pharo.org

Except where otherwise noted, this work is licensed under CC BY-NC-ND 3.0 France
https://creativecommons.org/licenses/by-nc-nd/3.0/fr/