JUNIT IN ACTION
Software Engineering (Exercise) Class

Prof. Adriano Peron
May, 31st 2013

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

XUNIT 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 an extend it to meet their specific needs.

- **Unit Test**: A unit test examines the behavior of a distinct unit of work.
  - The “distinct unit of work” is often (but not always) a single method.

IMPROVED (NAIVE) SOLUTION

```java
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)");
        }
    }
}
```

LESSON LEARNED

Objective Test + Repeatable Test = Simple Test Program

**Disclaimer**: The previous example showed a naive way to test (a.k.a. the **wrong** one)

That was **not** JUnit!!
JUnit Test Annotations

- @Test public void method()
  - Annotation @Test identifies that this method is a test method.

- @Before public void method()
  - Will perform the method() before each test.
  - This method can prepare the test environment
  - E.g. read input data, initialize the class, ...

- @After public void method()

JUnit Assert Statements

- assertNotEqual([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

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() {
    // ... test code ...
  }
}
```

Testing the Exception Handling the New Way!

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

```java
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?!
  }
}
```
JUnit Words Cloud

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

- Testing
- xUnit
- Java
- Unit Testing
- Testing framework
- Black Box Testing
- Test Suite
- Testing Automation
- Test Fixtures
- Simple Test Program
- Test Runners
**fixture:** The set of common resources or data that you need to run one or more tests.
EXERCISE 1

Calculator
**CALCULATOR CLASS**

- **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!

```
<table>
<thead>
<tr>
<th>Calculator</th>
</tr>
</thead>
<tbody>
<tr>
<td>- memory: double</td>
</tr>
<tr>
<td>+ MAX_DIGITS_LEN: int = 5</td>
</tr>
<tr>
<td>+ add (double augend, double addend): double</td>
</tr>
<tr>
<td>+ subtract (double minuend, double subtrahend): double</td>
</tr>
<tr>
<td>+ multiply (double multiplicand, double multiplier): double</td>
</tr>
<tr>
<td>+ divide (double dividend, double divisor): double</td>
</tr>
<tr>
<td>+ addToMemory(double number): void</td>
</tr>
<tr>
<td>+ recallNumber(): double</td>
</tr>
</tbody>
</table>
```

• Add method `parseExpression` to the Calculator class and corresponding bunch of tests!

• The method takes in input an expression string
  • (e.g., `1+2-3*4/2`)

• and returns the correct output result!
  • (i.e., `-3`)

```
Calculator
| - memory: double |
| + MAX_DIGITS_LEN: int = 5 |
| + 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 |
| + parseExpression(String expression): double |
```
EXERCISE II
Stack
STACK: **LIFO** QUEUE

**Stack**
- **<<constructor>>**
- `Stack(capacity: int)`
- `pop(): Process`
- `push(Process p): void`

**Process**
- `name: String`
- `pid: Integer`
- `priority: Integer (default=-1)`
- `getName(): String`
- `setName(String n): void`
- `getPid(): Integer`
- `setPid(Integer pid): void`
- `getPriority(): Integer`
- `setPriority(Integer p): void`
Q: How would you **test** Scheduler?

---

**Remember:** Unit tests run in **isolation**!