Test with JUnit

Version 2 - April 2013

- Test cases can be
  - Informal, undocumented
    - Also called 'informal testing'
  - Documented
    - As text, tables, pseudo code
    - As source code (usually in same language as tested code)

Testing with JUnit

- JUnit is a testing framework for Java programs
  - Idea of Kent Beck
- It is a framework with unit-testing functionalities
- Integrated in Eclipse development Environment
- *http://www.junit.org*

SoftEng
http://softeng.polito.it

What JUnit does (1)

- JUnit runs a suite of tests and reports results
- For *each* test method in the test suite:
  - JUnit calls the method
  - If the methods terminates without problems
    - The test is considered passed.

SoftEng
http://softeng.polito.it

Example

- Test Stack class

SoftEng
http://softeng.polito.it

Stack

```
public class Stack{
  public Stack(){   //…
   }
 public boolean isEmpty(){
   }
 public int pop(){
   }
 public void push(int value){
   }
```

SoftEng
http://softeng.polito.it

High level test cases

T1	Create stack Stack empty Push(10) Stack not empty
T2	Create stack Push(10) Push(-4) Pop() → -4 Pop() → 10 Pop() → empty

SOftEng
http://softeng.polito.it

Junit test cases

```
public void testStackT1() {
   Stack aStack = new Stack();
   assertTrue("Stack should be empty!",
                    aStack.isEmpty());
   aStack.push(10);
   assertTrue("Stack should not be empty!",
                    !aStack.isEmpty());
}
public void testStackT2() {
   Stack aStack = new Stack();
   aStack.push(10);
   aStack.push(-4);
   assertEquals(-4, aStack.pop());
   assertEquals(10, aStack.pop());
}
```

SOftEng
http://softeng.polito.it

Junit test cases

Extends TestCase

```
public class StackTest extends TestCase {
   public void testStackT1() {
       Stack aStack = new Stack();
       assertTrue("Stack should
                                aStack
       aStack.push(10);
       assertTrue("Stack should n
                    !aStack.isEmp
   }
   public void testStackT2() {
       Stack aStack = new Stack();
       aStack.push(10);
       aStack.push(-4);
       assertEquals(-4, aStack.pop());
       assertEquals(10, aStack.pop());
   }
}
```

Test method name:
testSomething

SoftEng

Execution

- Junit executes each test...() method in the Test class
- If the tests run correctly, nothing is done
- If a test fails, it throws an *AssertionFailedError*
- The JUnit framework catches the error and deals with it; you don't have to do anything

SoftEng

Execution (2)

- setUp() method is run before each test..() method
  - Useful to create the needed clean context for the test
- tearDown() method is run after
  - Useful to clean up, when needed
- They can be override

SoftEng
http://softeng.polito.it

- protected void runTest() {
  setUp();
  test…();
  tearDown();
}

SoftEng
http://softeng.polito.it

Assert*()

- For a condition
  - assertTrue("message when test fails", condition);
- If the tested condition is
  - True => execute the following instruction
  - False => break to the end of the test method, print out the optional message

SOftEng
http://softeng.polito.it

Assert (2)

- For objects, int, long, byte:
  - assertEquals(expected value, expression);
  - Es. `assertEquals( 2 , aStack.size() );`
- For floating point values:
  - assertEquals(expected value, expression, error);
  - Es. `assertEquals(1.0, Math.cos(3.14), 0.01);`

SOftEng
http://softeng.polito.it

Other assert*X* methods

```
assertTrue(boolean test)
assertFalse(boolean test)
assertEquals(expected, actual)
assertSame(Object expected,
           Object actual)
assertNotSame(Object expected,
              Object actual)
assertNull(Object object)
```

SoftEng http://softeng.polito.it

Other assert*X* methods

- `assertNotNull(Object object)`
- `fail()`
- All the above may take an optional String message as the first argument, e.g.

```
static void assertTrue(
           String message,
           boolean test)
```

SoftEng http://softeng.polito.it

Running a JUnit test case

- Running a JUnit test case :
  - Executes all public methods starting with "test"
  - Ignores the rest
- The class can contain helper methods
  - They are not public
  - Or they don't start with "test"

SoftEng
http://softeng.polito.it

Creating a test class in JUnit

- Define a subclass of TestCase
- Override the setUp() method to initialize object(s) under test.
- Override the tearDown() method to release object(s) under test.
- Define one or more public testXXX() methods that exercise the object(s) under test and assert expected results.

SoftEng
http://softeng.polito.it

Implementing setUp() method

- Override setUp() to initialize the variables, and objects
- Since setUp() is your code, you can modify it any way you like (such as creating new objects in it)
- Reduces the duplication of code

SoftEng http://softeng.polito.it

The tearDown() method

- In most cases, the tearDown() method doesn't need to do anything
  - The next time you run setUp(), your objects will be replaced, and the old objects will be available for garbage collection
  - Like the finally clause in a try-catch-finally statement, tearDown() is where you would release system resources (such as streams)

SoftEng http://softeng.polito.it

TestSuite

- Combine many test cases in a test suite:

```
public class AllTests extends TestSuite {

public AllTests(String name) {
  super(name);
}

public static TestSuite suite() {
  TestSuite suite = new TestSuite();
  suite.addTestSuite(StackTest.class);
  suite.addTestSuite(AnotherTest.class);
}
```

SoftEng http://softeng.polito.it

Organization of test cases

- Free, but suggested
  - One class – one test class
  - One package – one test package
  - Use TestSuite to call same tests in different ways
  - Ex: all unit tests
  - all (part of) integration tests
  - all (part of) system tests

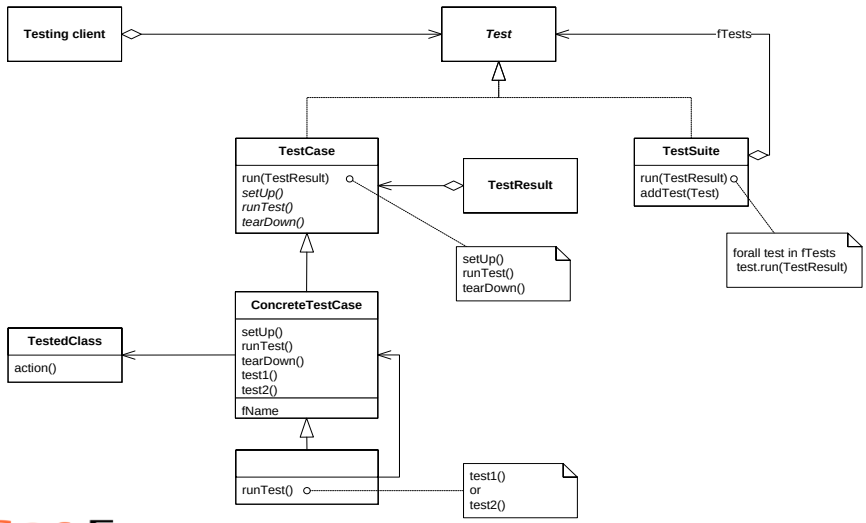
SoftEng http://softeng.polito.it

Organize The Tests

- Create test cases in the same package as the code under test
- For each Java package in your application, define a TestSuite class that contains all the tests for validating the code in the package
- Define similar TestSuite classes that create higher-level and lower-level test suites in the other packages (and sub-packages) of the application
- Make sure your build process includes the compilation of all tests

SoftEng http://softeng.polito.it

JUnit framework



SoftEng http://softeng.polito.it

Example: Counter class

- The constructor will create a counter and set it to zero
- The increment method will add one to the counter and return the new value
- The decrement method will subtract one from the counter and return the new value

SoftEng http://softeng.polito.it

Counter class

```
public class Counter {
   int count = 0;
   public int increment() {
        return ++count;
   }
   public int decrement() {
        return --count;
   }
   public int getCount() {
        return count;
}  }
```

SoftEng http://softeng.polito.it

JUnit tests for Counter

```
public class CounterTest extends
junit.framework.TestCase {
    Counter counter1;

    public CounterTest() { } // default ctor

    protected void setUp() {
        // creates a (simple) test fixture
        counter1 = new Counter();
    }
    protected void tearDown() { }
        // no resources to release
```

SOftEng http://softeng.polito.it

JUnit tests for Counter…

```
    public void testIncrement() {
        assertTrue(counter1.increment()== 1);
        assertTrue(counter1.increment()== 2);
    }
    public void testDecrement() {
        assertTrue(counter1.decrement()==-1);
    }
}
```

SOftEng http://softeng.polito.it

Problems with unit testing

- JUnit is designed to call methods and compare the results they return against expected results
  - This ignores:
    - Programs that do work in response to GUI commands
    - Methods that are used primary to produce output

SoftEng
http://softeng.polito.it

Problems with unit testing…

- Heavy use of JUnit encourages a "functional" style, where most methods are called to compute a value, rather than to have side effects
  - This can actually be a good thing
  - Methods that *just* return results, without side effects (such as printing), are simpler, more general, and easier to reuse

SoftEng
http://softeng.polito.it

Summary: elements of JUnit

- assert*()
  - Comparison functions
- TestCase
  - Class containing a set of tests
  - One per each class in production code
  - Many tests for each method of a class in production code
- TestSuite
  - Class containing a sequence of TestCase

SoftEng http://softeng.polito.it

ECLIPSE JUNIT PLUG-IN

SoftEng http://softeng.polito.it

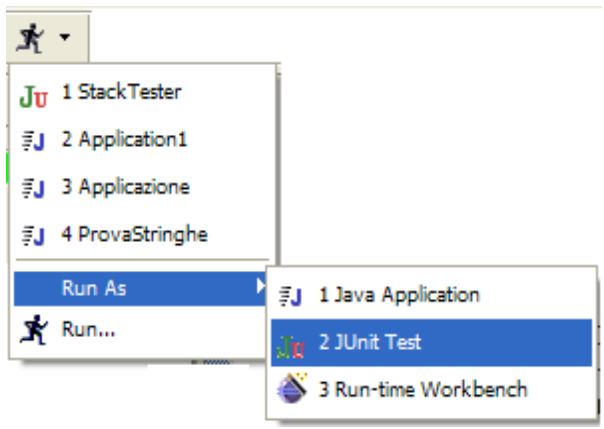
Junit in Eclipse – Setup

- In Eclipse
- open project's property window
- java build path
- libraries
- Add external jar
  - add org.junit

SOftEng

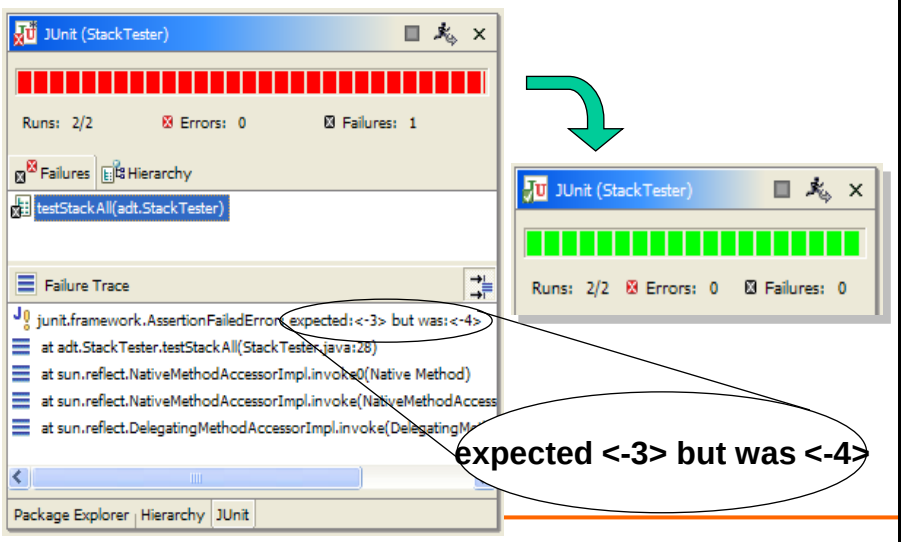
Junit in Eclipse – Run as JUnit Test

- Run
- Run As..
- Junit Test



SOftEng

Red / Green Bar



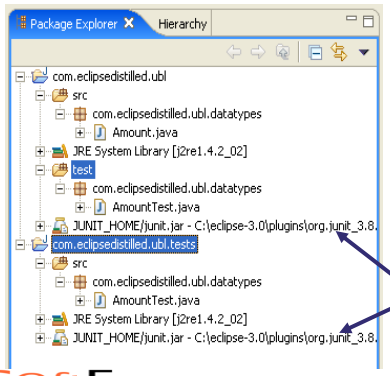
Unit Testing New Code

Unit testing can support several general strategies for validating the behavior of your software. When developing new code, write tests that:

- Specify the intended outcome of code yet to be written; then write code until the tests pass. This is the ideal test-first strategy advocated by agile development principles.
- Specify the correct operation for bug reports; then modify the code until these bug-fix tests pass.

SoftEng

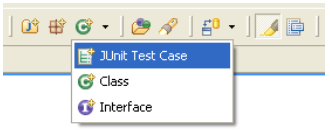
Organizing Unit Tests in Eclipse

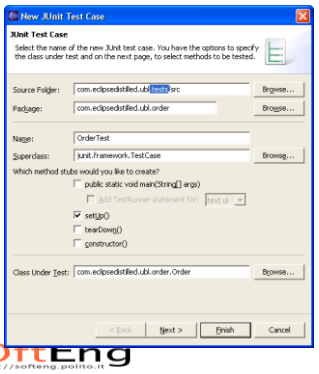


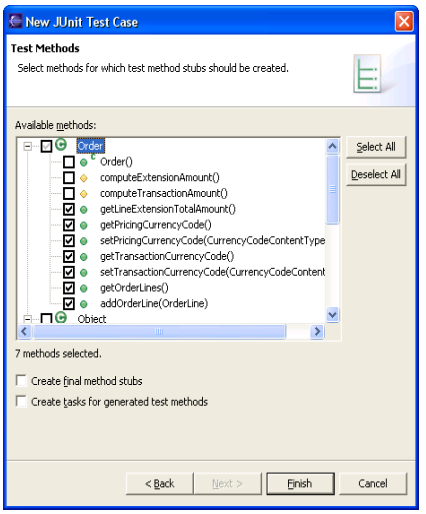
- Second source folder
  - Do not put JUnit tests in the same source folder as your project code
  - A second source folder allows clear separation
- Separate Project
  - Preferred configuration
  - No unit test libraries are added to your primary project classpath

Add junit.jar to the project classpath

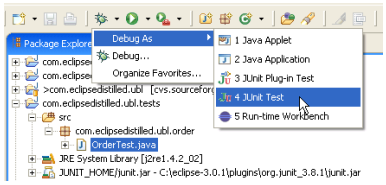
JUnit Test Case Wizard

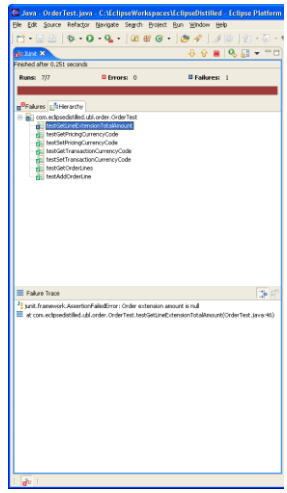






Running JUnit in Eclipse





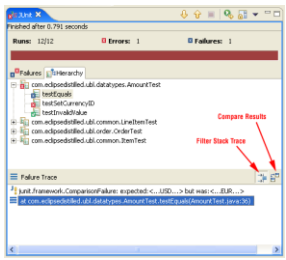
- Configure JUnit View as a **Fast View**

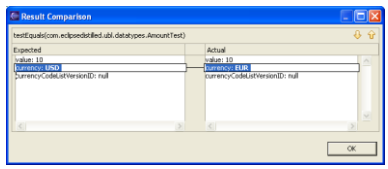
  Test results summary on status bar icon
- Double click error trace to go to test/app source

- **Failure**: JUnit assertion or fail was invoked
- **Error**: Unexpected error, e.g. NullPointerException

SoftEng http://softeng.polito.it

Additional Controls in JUnit View





- Filter Stack Trace
  - remove stack trace entries related to JUnit infrastructure
  - filter is configurable in preferences

    Java > JUnit
- Compare Results
  - available when `assertEquals()` is used to compare two string value

SoftEng http://softeng.polito.it

…use JUnit

Keep the bar green to keep the code clean…

SOftEng
http://softeng.polito.it