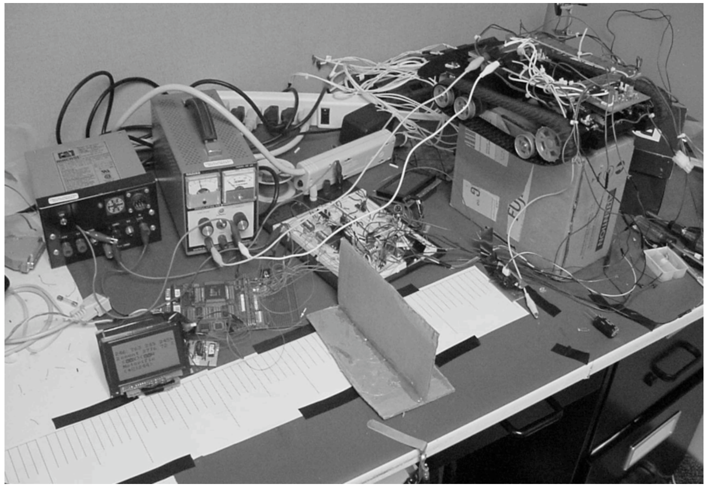
Unit Testing
JUnit and Coverage

Software Engineering



Chair of Programming Methodology

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Agenda for Today

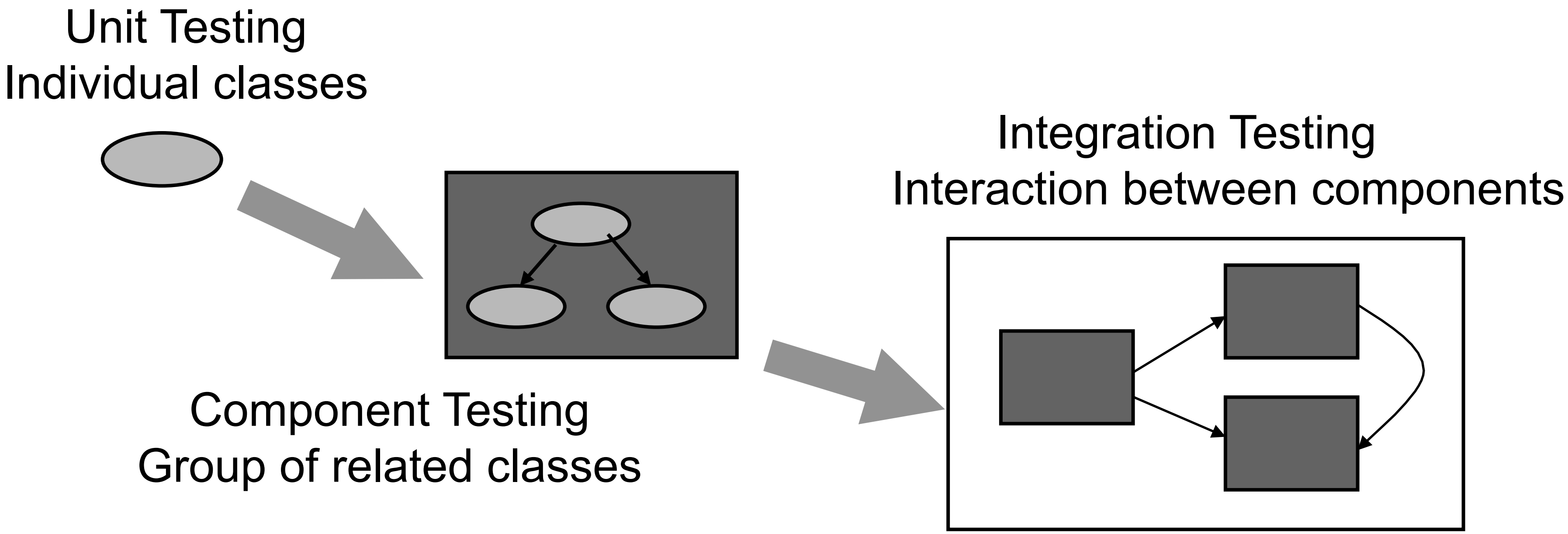
1. Testing
2. Main Concepts
3. Unit Testing – JUnit
4. Test Evaluation – Coverage
5. Reference

Software Testing

- Goal: find many errors before shipping software
  - Higher cost to fix errors after deployment
  - Higher acceptance and confidence of users
- Scientific approach
  - Proof correctness and completeness of code
- Pragmatic approach
  - Try out software in typical usage scenarios
- Fact
  - Testing does not guarantee the absence of errors

Testing Scope

- Testing in the small
  - Exercising the smallest executable units of the system
- Testing in the large
  - Putting the entire system to the test



Unit Testing
Individual classes

Integration Testing
Interaction between components

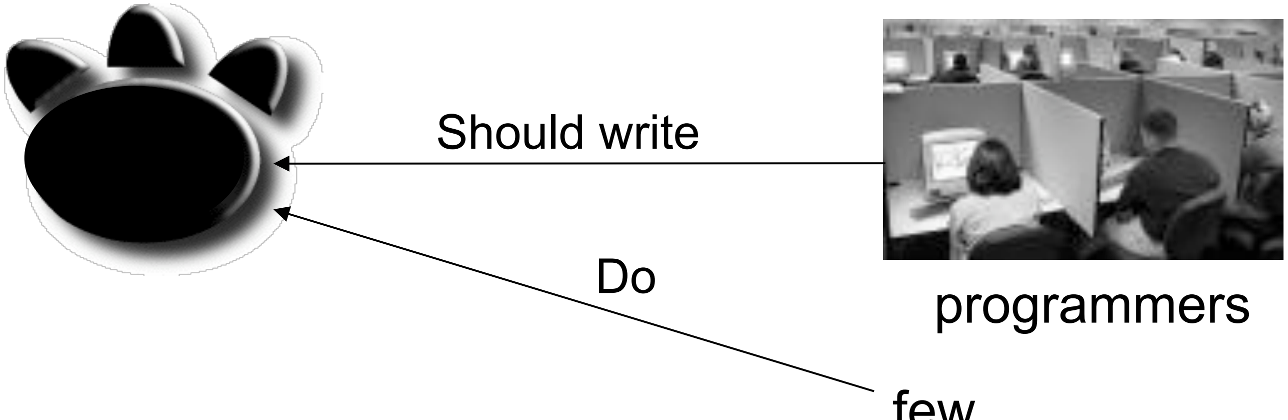
Component Testing
Group of related classes

Unit Testing

- Exercising the smallest individually executable units
- Objectives: find faults in the units, assure correct functional behavior of units
- Usually performed by programmers

- The Typical Test Cycle
  - Develop a suite of test cases
  - Create test fixtures to support each test case
  - Clean-up fixtures, if necessary
  - Run the test and capture test results
  - Report and analyze the test results

Testing Problem



Should write

Do

programmers

few

Why?

I am so busy…

It is difficult…

- Programmers need a tool to:

  "Write a few lines of code, then a test that should run; or even better, write a test that won't run, then write the code that will make it run."

JUnit

- Almost indisputably the single most important third-party Java library ever developed
- Fueled the testing explosion
  - Inspired a whole family of xUnit tools:
    NUnit(.net), PyUnit(Python), CppUnit(c++), DUnit(delphi), …
  - increasing number of extensions: JMLUnit, SQLUnit, XMLUnit
- IDE integration
  - NetBeans, BlueJ, IntelliJ, JBuilder, Eclipse, …
- open source

"Never in the field of software development was so much owed by so many to so few lines of code."
Martin Fowler

Detailed Look

- Written by Erich Gamma (of Design Patterns fame) and Kent Beck (creator of XP methodology)
- A simple framework to write and run repeatable tests
- JUnit features include:
  - Assertions for testing expected results
  - Test fixtures for sharing common test data
  - Test suites for easily organizing and running tests
  - Graphical and textual test runners

- JUnit 3.8.2: old stable version
  - Naming conventions and reflection
- JUnit 4.5: new stable version
  - Using Java 5 annotations

Terminology

- Test Case
  - defines a method to run a set of tests
- Test Suite
  - a collection of related test cases
- Test Fixture
  - a common set of test data and collaborating objects shared by many tests
  - Generally are implemented as instance variables in the test class
- Test Runner
  - runs tests and reports results
- Errors and failures
  - An error is some unanticipated failure (e.g., an exception thrown inside the tested code)
  - A failure is anticipated, and is produced by a call of an assertXXX method

JUnit 3.x Testing Steps

1. Create a test class
    - Import junit.framework.*
    - Declared as a subclass of TestCase
- Create Test Case
    - Name the test method as testXXX
    - Asserts the expected results on the object under test
- Use Test Fixture when necessary

1. Check for expected exceptions
2. Run the tests in the console or IDE

Example: Money class

```
import org.junit.*;
import static org.junit.Assert.*;
import java.util.*;

public class Money {
	private int fAmount;
  private String fCurrency;

	public Money(int amount, String currency) {
    fAmount=amount; fCurrency=currency; }

	public int amount() { return fAmount; }

	public String currency() { return fCurrency; }

	public Money add(Money m) {
    return new Money(  amount()+m.amount(),
                         currency()); }
}
```

Test method

```
public class MoneyTest extends TestCase {// test class
  public void testadd() {                    // test method
    // create the test fixture
     Money m12CHF = new Money(12, "CHF");
     Money m14CHF = new Money(14, "CHF");
    Money expected = new Money(26, "CHF");
    Money result = m12CHF.add(m14CHF);
    // verify the result, use assertEquals in this example
     Assert.assertEquals(expected,result);
  }
}
```

JUnit Assertions

- Within a test
  - Call the method being tested and get the actual result
  - Assert what the correct result should be with one of the provided assert methods
  - These steps can be repeated as many times as necessary

- An assert method
  - Is a JUnit method that performs a test
  - Throws an AssertionFailedError if the test fails
  - JUnit catches these Errors and shows you the result

JUnit Assertions (Cont’d)

- assertTrue(boolean test)
  assertFalse(boolean test)
  assertEquals(Object expected, Object actual)
  assertSame(Object expected, Object actual)
  assertNotSame(Object expected, Object actual)
  assertNull(Object object)
  assertNotNull(Object object)
  fail()
- assertXXX(**String** ***message***, …)
  - Throws an AssertionFailedError if the test fails
  - The optional *message* is included in the Error

Use of Fixtures

- Some test cases act on similar sets of objects
  - Create a fixture instead of declaring them in all methods
  - Write as many Test Cases as you like
  - Add as many test methods as you like

- Use in detail
  - Add fields for each part of the fixture
  - Define setUp to initialize the fields
  - Define tearDown to release any permanent resources

Test Runners

- Run the tests and collect the results
- Make sure that the **`junit.jar`** file is on classpath
- Textual/graphical user interface
  - Command line:

```
java junit.textui.TestRunner  <test class name>
java junit.swingui.TestRunner  <test class name>
```

  - Ant task (See `<junit>` tag)
- May use a main() method:

```
 public static void void
main(String
a
rgs
[]) {          junit.textui.TestRunner.run(suite());}
java junitfaq.SimpleTest
```

JUnit 4

- Forward and backward compatibility
  - JUnit4 can run JUnit 3 tests without any changes
  - To enable JUnit 4 tests to run in JUnit 3 environments, use JUnit4TestAdapter (see next slide)
- Java 5 annotations instead of naming conventions
- Free to use any superclass for tests
- Identify fixture methods with annotations, possible to have multiple fixture methods
- More annotations to simplify tests

Making a test class for Money

```
public class MoneyTest {
  @Test public void SimpleAdd() {
     Money m12CHF= new Money(12, "CHF");
     Money m14CHF= new Money(14, "CHF");
     Money expected= new Money(26, "CHF");
     Money result= m12CHF.add(m14CHF);

     Assert.assertTrue(expected.equals(result));
  }
}
```

Use of Fixtures

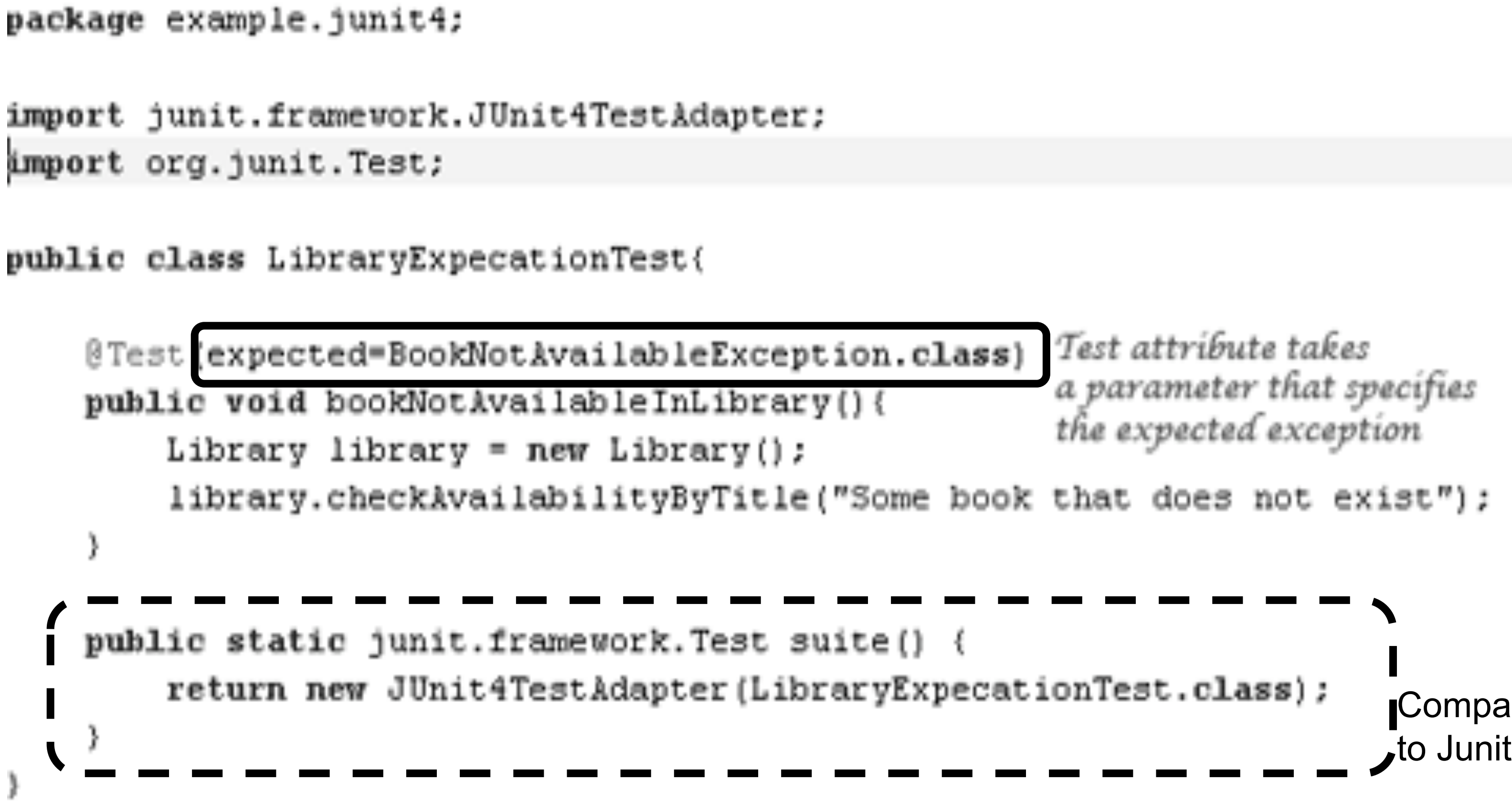
- when some test cases act on similar sets of objects
  - creating a fixture instead of declaring in all methods
  - write as many Test Cases as you'd like
  - add as many test methods as you'd like
- How to do
  - Add fields for each part of the fixture
  - Annotate a method with @Before and initialize the variables in that method
  - Annotate a method with @After to release any permanent resources you allocated in setUp
  - One-time setup and teardown for all classes: @_Class

Example: Use of Fixtures

```
public class MoneyTest {
  private Money m12CHF;  // fixture data
  private Money m14CHF;
  @Before public void setUp() {    // setting up fixture
     m12CHF= new Money(12, "CHF");
     m14CHF= new Money(14, "CHF");
  }
  @Test public void SimpleAdd() { // [12 CHF] + [14 CHF] = [26 CHF]
     Money expected= new Money(26, "CHF");
     Money result= m12CHF.add(m14CHF);
     Assert.assertTrue(expected.equals(result));
  }
}
```

JUnit 4: Expecting Exceptions

- @Test annotation takes a parameter declaring the type of Exception thrown (test fails if no exception is thrown) .



```java
package example.junit4;

import junit.framework.JUnit4TestAdapter;
import org.junit.Test;

public class LibraryExpecationTest{

    @Test(expected=BookNotAvailableException.class)
    public void bookNotAvailableInLibrary(){
        Library library = new Library();
        library.checkAvailabilityByTitle("Some book that does not exist");
    }

    public static junit.framework.Test suite() {
        return new JUnit4TestAdapter(LibraryExpecationTest.class);
    }
}
```

Test attribute takes a parameter that specifies the expected exception

Compatible to Junit 3.x

JUnit 4: Other Annotations

- Ignoring a test
  - @Ignore annotation tells the runner to ignore the test
  - @Ignore(“reason of why to ignore the test”) to pass in a string message to the runner and report it

- Timing out a test
  - @Test (timeout=10)
  - pass in a timeout parameter to the test annotation to specify the timeout period in milliseconds
  - If the test takes more, it fails

More Unit Testing Issues

- How do I test database dependent code?
  - dbUnit
- Should I test my web application? How?
  - HttpUnit
    - Parses HTML results into DOM
    - Easy link navigation and form population
    - Useful for automated acceptance tests
  - Canoo WebTest
    - HttpUnit inside Ant

A partial List of xUnit framworks:

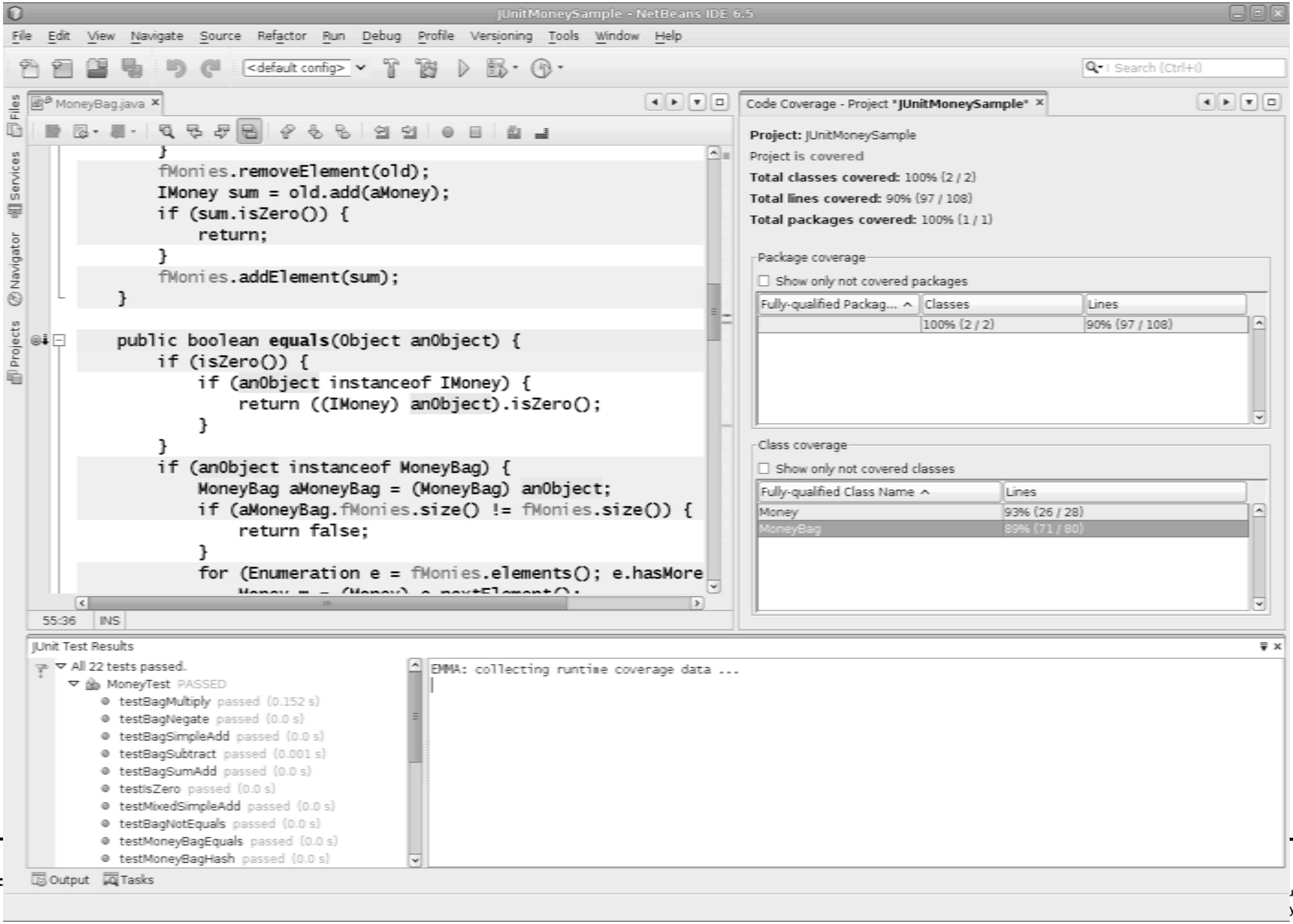
http://en.wikipedia.org/wiki/XUnit

Test Evaluation: Code Coverage

- How good is a test?
- Do we have enough test cases?
- Testing is inherently incomplete
  - Coverage metrics: quantitative evaluation of test suite
  - A test evaluation tool helps in assessing whether the test cases achieve good coverage or not

- Tools
  - Clover, Quilt, Emma, Coverlipse, JDepend, Cobertura, Java Test Coverage, …

Coverage Netbeans Plugin



```
        }
        fMonies.removeElement(old);
        IMoney sum = old.add(aMoney);
        if (sum.isZero()) {
            return;
        }
        fMonies.addElement(sum);
    }

    public boolean equals(Object anObject) {
        if (isZero()) {
            if (anObject instanceof IMoney) {
                return ((IMoney) anObject).isZero();
            }
        }
        if (anObject instanceof MoneyBag) {
            MoneyBag aMoneyBag = (MoneyBag) anObject;
            if (aMoneyBag.fMonies.size() != fMonies.size()) {
                return false;
            }
            for (Enumeration e = fMonies.elements(); e.hasMore
                Money m = (Money) e.nextElement();
```

55:36 INS

Code Coverage - Project "**JUnitMoneySample**"

Project: JUnitMoneySample

Project is covered

Total classes covered: 100% (2 / 2)

Total lines covered: 90% (97 / 108)

Total packages covered: 100% (1 / 1)

Package coverage

☐ Show only not covered packages

Fully-qualified Packag... ^	Classes	Lines
	100% (2 / 2)	90% (97 / 108)

Class coverage

☐ Show only not covered classes

Fully-qualified Class Name ^	Lines
Money	93% (26 / 28)
MoneyBag	89% (71 / 80)

JUnit Test Results

- All 22 tests passed.
  - MoneyTest PASSED
    - testBagMultiply passed (0.152 s)
    - testBagNegate passed (0.0 s)
    - testBagSimpleAdd passed (0.0 s)
    - testBagSubtract passed (0.001 s)
    - testBagSumAdd passed (0.0 s)
    - testIsZero passed (0.0 s)
    - testMixedSimpleAdd passed (0.0 s)
    - testBagNotEquals passed (0.0 s)
    - testMoneyBagEquals passed (0.0 s)
    - testMoneyBagHash passed (0.0 s)

```
EMMA: collecting runtime coverage data ...
```

JMLUnit

- A model-driven test generation tool, from Iowa State University
- One of a suite of tools based on the JML behavioral interface specification language
- Automatic generation of test oracles using
  - Formal specifications and
  - Runtime assertion checker
- License: open source

Summary

- “Any program feature without an automated test simply doesn’t exist”
- Testable code improves confidence and design
- Programmers can sleep better
- “Keep the bar green to keep the code clean!”

Reference

- JUnit http://www.junit.org
  http://junit.sourceforge.net/doc/cookbook/cookbook.htm
- Extreme programming

  http://www.xprogramming.com
- Coverage http://codecoverage.netbeans.org
- dbUnit http://www.dbunit.org
- HttpUnit http://www.httpunit.org
- Canoo WebTest http://webtest.canoo.com
- Software QA and Testing Resource Center http://www.softwareqatest.com
- JMLUnit http://jmlspecs.org