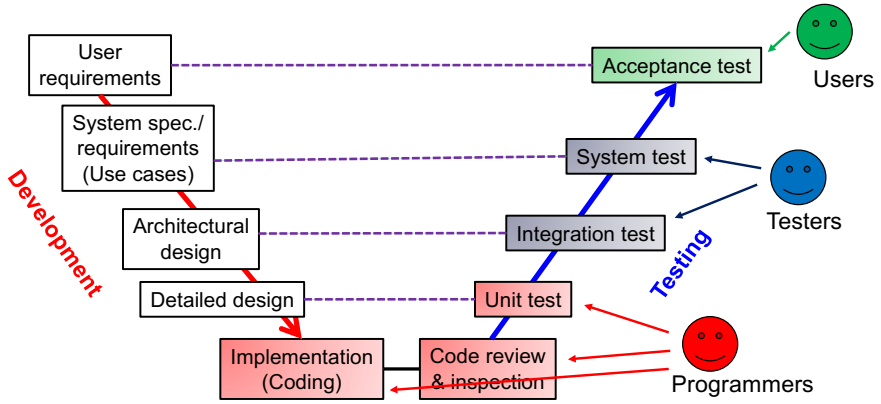
Unit Testing

Unit Tests

- Verify that each *program "unit"* works as it is intended and expected along with the system specification.
  - Units to be tested: classes (methods in each class) in OOPLs



Who Does it?

- You!
  - as a programmer
- Test cases are written *as programs* from a programmer's perspective.
  - A test case describes a test to verify a tested class in accordance with the system specification.
- Programmers and unit testers are no longer separated in most (both large-scale and small-scale) projects as
  - it has been a lot easier and less time-consuming to write and run unit tests.
  - programmers can write the best test cases for their own code in the least amount of time.

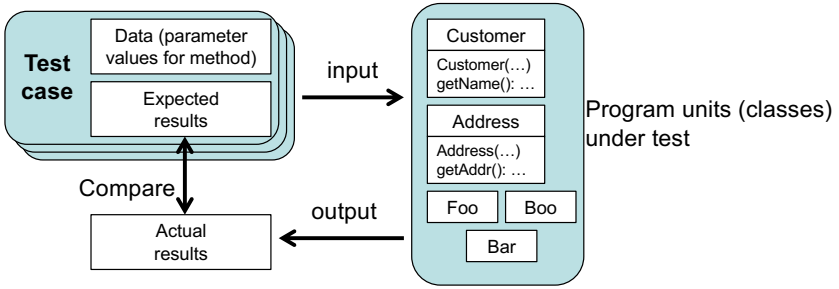
What to Do in Unit Testing?

- 4 tests (test types)
  - CS680 focuses on 3 of them: *functional*, *structural* and *confirmation* tests.

	Functional test	Non-functional test	Structural test	Confirmation test
Acceptance test				
System test				
Integration test				
Unit test	**X (B-box)**	**?**	**X (W-box)**	**X**
Code rev&insp.				

Functional Test in Unit Testing

– Ensure that each method of a class successfully performs a set of specific tasks.
  - Each test case confirms that a method produces the expected output when given a known input.
    – Black-box test
  - Well-known techniques: equivalence test, boundary value test



Structural Test in Unit Testing

- Verify the structure of each class.
- Revise the structure, if necessary, to improve maintainability, flexibility and extensibility.
  – White-box test

- To-dos
  – Refactoring
  – Use of design pattern
  – Control flow test
  – Data flow test

Confirmation Test in Unit Testing

- Re-testing
- Regression testing

HW: Reading Assignment

- Nick Jenkins, *A Software Testing Primer: An Introduction to Software Testing*, 2008.
  – http://www.nickjenkins.net/prose/testingPrimer.pdf

Unit Testing with JUnit

JUnit

- A unit testing framework for Java
  - Defines the format of a test case
    - Test case
      - a program to verify a method(s) of a given class with a set of inputs/conditions and expected results.
  - Provides APIs to write and run test cases
  - Reports test results
- Making unit testing as easy and automatic as possible.
- Version 4.x, http://junit.org/
- Integration with Ant and Eclipse (and other IDEs)
  - <junit> and <junitreport> for Ant

Test Classes and Test Methods

- Test class
  - A public class that has a set of "test methods"
  - Common naming convention: XYZTest
    - XYZ is a class under test.
  - One test class for one class under test
- Test method
  - A public method in a test class.
    - No parameters
    - No values returned ("void" return type)
    - Can have a "throws" clause
  - Annotated with @Test
    - org.junit.Test
  - One test method implements one test case.

Assertions

- Each test method verifies one or more *assertions*.
  - An assertion is a statement that a predicate (boolean function/expression) is expected to always be true at a particular point in code.
    - `String line = reader.readLine();`
      Assertion: `line != null`
    - `String str = foo.getPassword();`
      Assertion: `str.length() > 6`
- In JUnit, running unit tests means verifying assertions described in test methods.

An Example

- Class under test

```
public class Calculator{
 public float multiply(float x,
                       float y){
   return x * y;
 }
 public float divide(float x,
                     float y){
   if(y==0){ throw
    new IllegalArgumentException(
     "division by zero");}
   return x/y;
 }
]
```

- Test class

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.junit.Test;

public class CalculatorTest{
 @Test
 public void multiply3By4(){
   Calculator cut = new Calculator();
   float expected = 12;
   float actual = cut.multiply(3,4);
   assertThat(actual, is(expected)); }

 @Test
 public void divide3By2(){
   Calculator cut = new Calculator();
   float expected = 1.5f;
   float actual = cut.divide(3,2);
   assertThat(actual, is(expected)); }


@Test(expected=illegalArgumentException.class)
 public void divide5By0(){
   Calculator cut = new Calculator();
   cut.divide(5,0); }
}
```


Key APIs

- `org.junit.Assert`
  - Used to define an assentation and verify if it holds
- `org.hamcrest.CoreMatchers`
  - Provides a series of *matchers*, each of which performs a particular matching logic.

Key Annotations

- @Test
  - org.junit.Test
  - JUnit runs test methods that are annotated with @Test.
- @Ignore
  - org.junit.Ignore
  - JUnit ignores test methods that are annotated with @Ignore
    - No need to comment out the entire test method.

Static Imports

- `Assert` and `CoreMatchers` are typically referenced through *static import*.
  - `import static org.junit.Assert.*;`
    `import static org.hamcrest.CoreMatchers.*;`
  - With static import
    - `assertThat(actual, is(expected);`
      - "assert that actual is expected"
      - `assertThat()` is a static method of `Assert`.
      - `is()` is a static method of `CoreMatcher`.
  - With normal import
    - `Assert.assertThat(actual, CoreMatchers.is(expected));`

JUnit and Hamcrest

- Hamcrest provides many useful matchers for JUnit
  - http://hamcrest.org/JavaHamcrest/
  - junit.jar and hamcrest-core.jar available from http://junit.org
    - Both are available in Eclipse (and other IDEs) by default.

Principles in Unit Testing

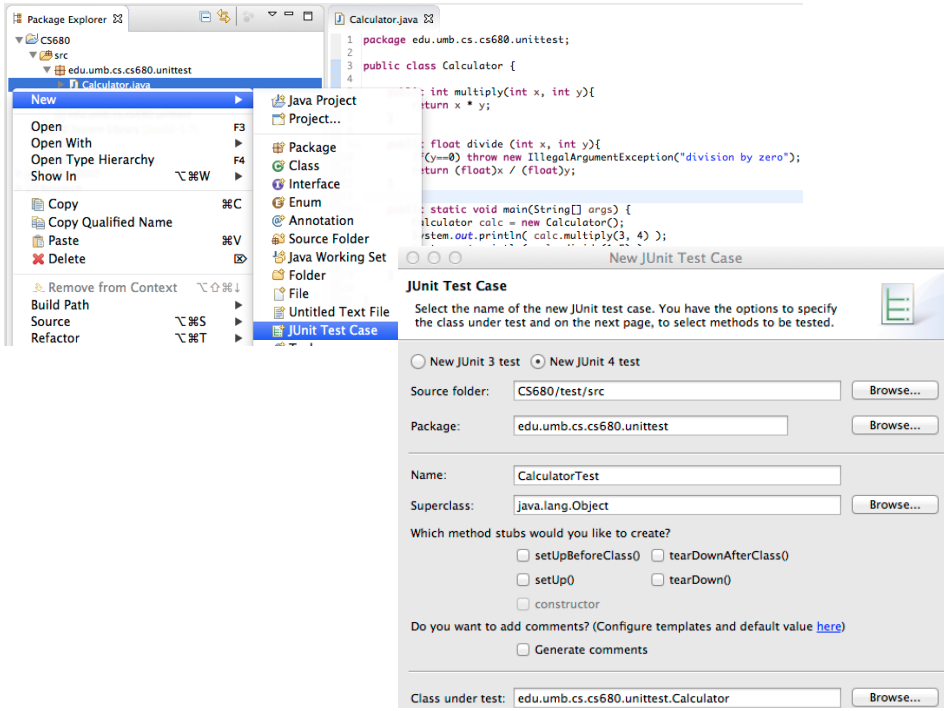
- Define one or more fine-grained specific test cases (test methods) for each method in a class under test.
- Give a concrete/specific and intuitive name to each test method.
  - e.g. “divide5by4”
- Use specific values and conditions, and detect design and coding errors.
  - Be detail-oriented. The devil resides in the details!
- No need to worry about redundancy in/among test methods.

Principles in Unit Testing (cont’d)

- Write simple, short, easy to understand test cases
  - Try to write many simple test cases, rather than a fewer number of complicated test cases.
    - Avoid a test case that perform multiple tasks.
    - You won’t feel bothered/overwhelmed by the number of test cases as far as they have intuitive names.
      - e.g. “divide5by4”

Test Suite with JUnit

- A set of test classes
  - ~/code/projectX/ [project directory]
    - build.xml
    - src [source code directory]
      - edu/umb/cs/cs680/Foo.java
      - edu/umb/cs/cs680/Boo.java
    - bin [byte code directory]
      - edu/umb/cs/cs680/Foo.class
      - edu/umb/cs/cs680/Boo.class
    - test [a test suite; a set of test classes]
      - src
        - » edu/umb/cs/cs680/FooTest.java
        - » edu/umb/cs/cs680/BooTest.java
      - bin
        - » edu/umb/cs/cs680/FooTest.class
        - » edu/umb/cs/cs680/BooTest.class



Things to Test

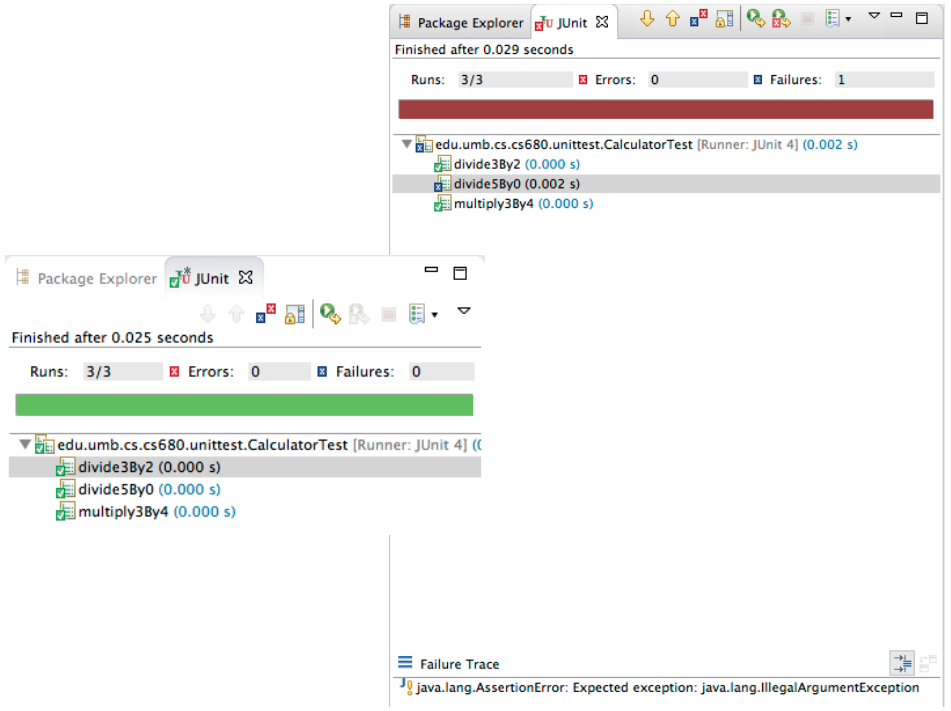
- Methods
- Exceptions
- Constructors

```
import static org.junit.Assert.*;
import static org.hamcrest.CoreMatchers.*;
import org.junit.Test;

public class StudentTest{
  @Test
  public void constructorWithName(){
    Student cut = new Student("John");
    assertThat(cut.getName(), is("John"));
    assertThat(cut.getAge(), is(nullValue()));
    assertThat(cut.getEmailAddr(), is(nullValue()));
  }
 @Test
  public void constructorWithoutName(){
    Student cut = new Student();
    ...
}
```


Test Runners

- How to run JUnit?
  - From command line
    - `java org.junit.runner.JUnitCore edu.umb.cs.cs680.CalculatorTest`
    - `java org.junit.runner.JUnitCore edu.umb.cs.cs680.FooTest, edu.umb.cs.cs680.BooTest`
  - From IDEs
    - Eclipse, etc.
  - From Ant
    - <junit> task
- How to run unit tests?
  - Test runners
    - org.junit.runners.JUnit4 (default runner)



HW 4

- Implement Calculator and CalculatorTest
  - edu.umb.cs.cs680.unittest.Calculator
  - edu.umb.cs.cs680.unittest.CalculatorTest
  - Define extra test methods in addition to `multiply3By4()`, `divide3By2()`, `divide5By0()`.
    - e.g., A float number times a float number
      - `Multiple2_5By5_5()`
    - e.g., A float number over a float number
      - `Multiple2_5By5_5()`
- Follow the directory structure shown in Slide 19.
  - <proj dir>/src/edu/umb/cs/cs680/unittest/Calculator.java
  - <proj dir>/bin/edu/umb/cs/cs680/unittest/Calculator.class
  - <proj dir>/test/src/edu/umb/cs/cs680/unittest/CalculatorTest.java
  - <proj dir>/test/bin/edu/umb/cs/cs680/unittest/CalculatorTest.class

- Use Ant to compile both Calculator and CalculatorTest and run test cases with JUnit
  - Run JUnit from Ant. Use <junit> task in Ant.
    - c.f. JUnit documentations (API docs, user manual, etc.)
  - No need to save test results in files. Just print them out on console
    - <formatter type="plain" usefile="false" /> in <junit>
  - Set junit.jar and hamcrest-core.jar to CLASSPATH.
    - <property environment="env"/>
    - ${env.HOME}/.p2/pool/plugins/org.junit_4.xxx/junit.jar
    - ${env.HOME}/.p2/pool/plugins/org.hamcrest.core_1.xxxx.jar