Events this week

Drop-in Resume/Cover Letter Editing
Date:       Tues., Jan 19
Time:       12:30 – 2 pm
Location:   Rm 255, ICICS/CS Bldg.

Interview Skills Workshop
Date:       Thurs., Jan 21
Time:       12:30 – 2 pm
Location:   DMP 201
Registration: Email dianejoh@cs.ubc.ca

Project Management Workshop
Speaker:    David Hunter (ex-VP, SAP)
Date:       Thurs., Jan 21
Time:       5:30 – 7 pm
Location:   DMP 110

CSSS Laser Tag
Date:       Sun., Jan 24
Time:       7 – 9 pm
Location:   Planet Laser @ 100 Braid St., New Westminster

Event next week

Public Speaking 101
Date:       Mon., Jan 25
Time:       5 – 6 pm
Location:   DMP 101

Administrivia

- Lecture slides (day by day) are on the web:
- Reminder: Assignment #1 is on the web site
  - Due Thursday, January 28th, 10:00pm
A quick backwards step

- In response to Rui’s question about subclasses and exceptions last time, I said categorically that a subclass can't override a method and extend the list of exceptions that are thrown. At the time, I wasn’t actually positive that I was right (but that has never stopped me from categorically claiming to be right).
- Turns out, I was right.

```java
class MyFirstException extends Exception {
    MyFirstException() {}
    MyFirstException(String msg) { super(msg); }
}
class MySecondException extends Exception{
    MySecondException() {}
    MySecondException(String msg) { super(msg); }
}
class ExceptionTest {
    void dostuff() throws MyFirstException {
    }
}
class SubExceptionTest extends ExceptionTest {
    void dostuff() throws MyFirstException, MySecondException {
    }
}
public class ExceptionTests {
    public void main(String argv[]) {
    ExceptionTest t = new SubExceptionTest();
    }
```
Unit Testing

You will be expected to:
• compare and contrast blackbox and whitebox testing (at the level of what each type of testing provides)
• use blackbox testing with equivalence classes to test a method
• describe how unit testing is applied to a class
• write a suite of tests to apply unit testing to a class using Junit (putting the above into practice with a particular tool)

Reading:
2nd Ed: Sections 10.1 to 10.5
3rd Ed, 4th Ed: 3.6T, 5.5T, 7.8T, 8.10T

Some “Famous” Software Problems

Ariane 5, June 4, 1996

Vancouver Stock Exchange Rounding Problem, 1983

Therac-25, mid-80s

See comp.risks for more each day.
Testing

• Terminology:
  • test case: a set of inputs and expected outputs that test a single use of a piece of the system (e.g., a method, a class, a subsystem)
  • test: a set of test cases
  • test driver: code that sets up any context needed to run a test, calls the test case(s), and displays the results
  • test stub: code that simulates the behaviour of the actual code that is still to be written

Testing Activities

• Unit Testing (for individual classes or small groups of classes)
  • find differences between what an object does and what it is supposed to do
  • testing one (or a few) class(es) is easier than testing the whole system
  • Enables incremental and parallel testing

  – There are other kinds of testing
    • Integration Testing (for a group of classes or subsystems)
    • System Testing (check if system does what is intended)
Unit Testing Types

- There are two major types of unit testing

**Blackbox testing**
- focuses on input/outputs only
- cases are derived from class specification

**Whitebox testing**
- focuses on the component's internal structure
- attempts to test all states and interactions

- is good for testing interfaces
- does not effectively test all cases

- also known as structural testing
- complementary to black-box testing

Blackbox Testing : Input Partition

- In general, we can't fully test an application.
  - applications often accept many different inputs
  - testing every different combination of inputs is practically impossible.

- To test a method, divide its inputs into equivalence classes (*here we use the term class as category, not a Java class!*)
  - all values within an equivalence class behave similarly with respect to the specification
  - equivalence classes are disjoint
  - they should cover the entire input space
Blackbox Testing : Input Partition (cont'd)

• Use preconditions, postconditions and class invariants to determine the equivalence classes for the input partition
• The method preconditions will divide the input into
  • *Valid* space that satisfies the preconditions and
  • *Invalid* space that violates the preconditions
• In general we need to test the valid input space only.

Blackbox Testing : Selecting Test Cases

• First, identify the valid input space and divide it into equivalence classes
• From each equivalence class, select:
  • at least one typical value - equivalence partition testing (sometimes called equivalence class testing)
  • some boundary values – boundary testing
Example 1

class Account {
    …
    /**
     * @pre amount >= 0
     * …
     */
    public void deposit(double amount) { … }
}

• One equivalence class that satisfies the precondition:
  amount >= 0
  • Select at least one typical member of the class, amount = 200
  • Select values at boundaries, only one boundary, amount = 0

• Test cases are then: {amount = 200, amount = 0}