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:
  • http://www.cs.ubc.ca/~norm/211/2009W2
• Reminder: Assignment #1 is on the web site
  • Due Thursday, January 28th, 10:00pm
• NSERC Summer Undergrad Student Research Award
  • Applications due 26 Feb 2010 (earlier the better)
**Where are we?**

- Testing helps to develop confidence that an implementation matches its specification
- Unit testing, integration testing, system testing
- Black-Box vs. White-Box testing
- Partition the input into equivalence classes
- Select values from each equivalence class
  - Typical values
  - Boundary values

---

**Example 1**

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

- One equivalence class that satisfies the precondition: \( amount \geq 0 \)
  - 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 \}
Example 2

class Account {
  ...
  /**
   * @pre true
   * @throws IllegalValueException when amount < 0
   */
  public void deposit (double amount) { ... }
}

• Two equivalence classes. What are they?
  \[
  \begin{align*}
  \text{amount} & > 0 \\
  \text{amount} & < 0
  \end{align*}
  \]

• What test cases would you specify?
  \[
  \begin{align*}
  0 & \quad 200 \\
  -100 & \quad -0.01
  \end{align*}
  \]

Blackbox Testing : Selecting Test Cases

• For multiple inputs:
  • partition each input
  • take the Cartesian product of all input partitions to produce
    a set of equivalence classes for the unit tests
  • in some cases, it may be possible to combine some of the
    classes resulting from the Cartesian product.
Example 3

```java
/**
 * @invariant rate >= 0
 * @invariant hours >= 0
 */
public class Employee {

private double rate;  // dollars per hour
private int hours;    // number hours worked

/**
 * @post IF rate < 100.00 AND hours > 40
 *      THEN return 40*rate + (hours-40)*1.5*rate
 *      ELSE return hours * rate
 */
public double getPay() { … }
…
}
```

Example 3…

- What is the input to `getPay()`?
  - `rate`  `hours`

- What are the equivalence classes?
  - `rate > 100`  `rate < 100`
  - `hours > 40`  `hours <= 40`

- What are the test cases?
  - `100, 41`  `100, 50`  `110, 41`  `150, 50`
  - `100, 40`  `100, 30`  `110, 40`  `110, 30`
  - `99, 41`  `99, 50`  `90, 41`  `90, 50`
  - `99, 40`  `99, 30`  `90, 40`  `90, 30`
Example 4

/**
 * @invariant xUnits >= 0 && yUnits >= 0
 * @invariant cost = 4 * yUnits + 3 * xUnits;
 */

class Order {
    private int xUnits;
    private int yUnits;
    private int cost;
    /**
     * Determines if a discount applies to an order.
     * @post returns true if ((cost >= 60) && (xUnits >= 9))
     * @post returns false if ((cost < 60 ) || (xUnits < 9))
     */
    boolean isEligibleForDiscount() { … }
}

Example 4 …

• What are the equivalence classes?
  \[ \text{Cost} < 60 \quad \text{II} \quad \text{xUnits} < 9 \]
  \[ \text{Cost} \geq 60 \quad \text{II} \quad \text{xUnits} \geq 9 \]

• What are the test cases?
  \[ 56, 8 \]
  \[ 63, 9 \quad 60, 20 \]
Unit Testing a Class

- Consider unit testing the `Account` class using black-box testing techniques
- For each method in `Account` we need to
  - consider the implicit argument as well
  - determine appropriate set of test cases using equivalence partitioning with boundary condition testing
  - create a test driver that
    - initializes `Account` objects to an appropriate state
    - runs the test cases (which includes checking the results)
Unit Testing a Class …

• May need to be careful in the order in which test cases are run, because one method may call another in its implementation.
• Need to rerun the unit test cases each time the code of the Account class is changed (regression testing).

JUnit

• A framework for implementing unit testing in Java.
• Provides a uniform and hierarchical test design.
• Can even write tests before you develop the code for the classes.
• The specifics of how to create JUnit tests will be covered in the lab.
• Eclipse provides good support for JUnit
  • tests are run in a JUnit mode without the need of a main() method
  • test results are displayed in a special JUnit view.
Exercise

Consider the following method that determines if a year is a leap year:

class Year {

    /**
     * Determine by the Gregorian calendar if the given
     * year is a leap year.
     * @pre year >= 1582
     */
    public static boolean isLeap(int aYear) { ... }
}

- A year is a leap year if and only if:
  - (aYear % 4 == 0) && ((year % 100 != 0) || (year % 400 == 0))

- What test cases would you choose to test isLeap()?