CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Testing and Correctness

Department of Computer Science
University of Maryland, College Park

Debugging Is Harder Than Coding!

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it"

– Brian W. Kernighan and P. J. Plauger, *The Elements of Programming*

Overview

- Program correctness is determined by the presence / absence of program defects (errors)
- Issues
  - Types of program errors
    - Compile-time
    - Run-time
    - Logic
  - Testing
  - Debugging

Program Errors (Compile-Time)

- Errors in code construction
  - Lexical (typographical), grammatical, types
- Detected during compilation
- Usually easy to correct quickly
- Examples
  - Misspelled Keyword
  - Missing or misplaced symbol
  - Incorrect operator for variable type

Program Errors (Run-time)

- Operations illegal / impossible to execute
- Detected during program execution
  - But not detectable at compile time
- Treated as exceptions in Java
- Examples
  - Division by zero
  - Array index out of bounds
  - Using null pointer
  - Illegal format conversion

Program Errors (Logical)

- Logical errors
  - Operations leading to incorrect program state
  - May (or may not) lead to run-time errors
  - Problem in design or implementation of algorithm
- Examples
  - Computing incorrect arithmetic value
  - Ignoring illegal input
- Hardest error to handle
  - Detect by testing
  - Fix by debugging

Testing

- Run program (or part of program) under controlled conditions to verify behavior
  - **Detects run-time error if exception thrown**
  - **Detects logical error if behavior is incorrect**
  - **Use of debugger is important**
- Issues
  - Selecting test cases
    - Think of them as you develop code or before
  - Test coverage
  - Others

Unit Test

- Test individual units extensively
  - Classes
  - Methods
- Central part of Extreme Programming (XP)
  - **Extensive unit testing during development**
    - Pair programming
  - **Design unit tests along with specification**
- Approach
  - Test each method of class
  - Test every possible flow path through method

Flow Path

- Unique execution sequence through program
- Example

```
S1
while (B1) {
    if (B2)
        S2
    else
        S3
}
```

→

Flows

S1
S1, S2
S1, S3
S1, S2, S2
S1, S2, S3
S1, S3, S2
S1, S3, S3
…

Test Coverage

- **Not possible to test all flow paths**
  - Many paths by combining conditionals, switches
  - **Infinite number of paths for loops**
  - New paths caused by exceptions
- **Test coverage**
  - Whether code is executed by some test case
  - Alternative to flow path
  - Ensure high % (if not all) of lines of code tested
    - What does 100% test coverage mean?
  - **Does not capture all possible flow paths**
    - Even if all lines of code tested by some test case

Test Coverage, Continued

- Branch coverage is stronger than statement coverage
  - Generally achievable
- Can be tricky to cover all exceptions and error cases
- Control flow coverage doesn't tell you about **data coverage**
  - Did you try it with negative integers, or with non-ASCII characters?
- Coverage won't tell you about functionality you forgot to implement or test
- Java Code Coverage for Eclipse
  - You can get code coverage information by using the option

Run→Coverage As→Java Application

Developing Quality Test Cases

- Tips on developing test cases
  - **Develop test data during analysis & design phases**
    - Use cases $\rightarrow$ Test cases
    - Pay close attention to problem specification
  - **Check boundary conditions**
    - 1st and last iterations of loop
    - 1st and last values added to data structure
  - **Improve code coverage**

About Testing

- **JUnit**
  - Review the information available at
    - http://www.cs.umd.edu/eclipse/junit/
  - Notice the problem you may experience while using static and JUnit
- **Findbugs (Static analysis to find coding mistakes)**
  - http://findbugs.sourceforge.net/

Debugger

- Be familiar with a Java Debugger
- Important operations
  - Setting breakpoints
  - Stepping into functions
  - Stepping over
  - Looking at variable values
- Additional information at
  - http://www.cs.umd.edu/eclipse/debugging/

Rubber Duck Debugging

- https://en.wikipedia.org/wiki/Rubber_duck_debugging

Eclipse's Code Coverage Support

- Java Code Coverage for Eclipse
  - You can get code coverage information by using the option

    Run→Coverage As→Java Application