CS3500: Object-Oriented Design
Spring 2014

Class 4
1.17.2014

Northeastern University
College *of* Computer and Information Science

Today...

- Testing
- Debugging
- Procedural Abstraction

Tutoring

- No tutoring when there are no classes.

Phases of the Software Process

- Requirements
- Design
- Implementation
- Testing
- Maintenance

Testing

Why Test?

- Meeting specifications/requirements
- Find bugs

Testing Terminology
[Liskov]

- Validation
- Verification
- Testing
- Exhaustive testing
- Unit testing
- Integration testing
- Regression testing

How much time is spent with testing?

Brooks' rule of thumb for scheduling a software project

- 1/3 design
- 1/6 coding
- 1/2 testing
  - 1/4 component testing
  - 1/4 system testing

Testing early and often

- **Test early:** start testing as soon as parts are implemented

- **Test often:** running tests at every reasonable opportunity

Testing

- **Black-box testing**
  - Based on specifications/requirements or what the software should do
- **White-box testing** (or glass-box testing)
  - Based on knowledge of specifications/ requirements, design, and implementation

Black-Box Testing

Advantages of Black-Box Testing

[Liskov]

- tests not adversely influenced by the component being tested
- robust with respect to changes in the implementation
- results of tests can be interpreted by people unfamiliar with implementation

Types of Black-Box Testing

- Specifications/Requirements
- Equivalence Partitioning
- Boundary Testing

Black-Box Testing: Specifications/Requirements

Black-Box Testing: Specifications/Requirements

```
MySet.size(MySet.empty())  =  0
MySet.size(MySet.insert(s0, k0))
  =  MySet.size(s0)                    if MySet.contains(s0, k0)
MySet.size(MySet.insert(s0, k0))
  =  1 + MySet.size(s0)                if !(MySet.contains(s0, k0))
```

```
Long one = new Long(1);
Long two = new Long(2);

MySet f0 = MySet.empty();
MySet f1 = MySet.insert(f0, one);
MySet f2 = MySet.insert(f1, two);
MySet f5 = MySet.insert(f2, one);

assertTrue("size {}", MySet.size(f0) == 0);
assertTrue("size {1}", MySet.size(f1) == 1);
assertTrue("size {1,2,1}", MySet.size(f5) == 2);
```

Black-Box Testing
Equivalence Partitioning

Black-Box Testing Equivalence Partitioning

- Types of Triangles
  - Equilateral: all sides greater than 0 and equal
  - Isosceles: all sides greater than 0, two side equal, and form a triangle
  - Scalene: all sides greater than 0, form a triangle, not equilateral or isosceles

Black-Box Testing
Boundary Value Analysis

Black-Box Testing
Boundary Value Analysis

X >= 93	A
90 <= X < 93	A-
87 <= X < 90	B+
83 <= X < 87	B
80 <= X < 83	B-
77 <= X < 80	C+
73 <= X < 77	C
70 <= X < 73	C-
67 <= X < 70	D+
63 <= X < 67	D
60 <= X < 63	D-
X < 60	F

Test Case Information

- Unique Identifier
- Description with preconditions and input
- Expected output from the program or program unit
- Actual results of running the test case

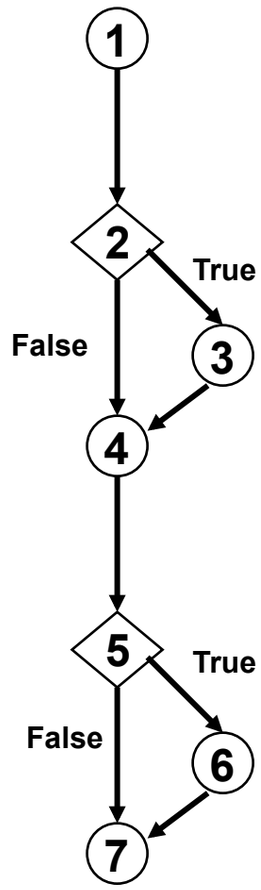
Test ID	Description	Expected Results	Actual Results
TestName (Test Author) Test Type:	Preconditions Test Inputs	Test outputs	Actual outputs

White-Box Testing

Types of White Box Testing

- Statement Coverage
- Branch Coverage
- Path Coverage
- Equivalence Partitioning
- Boundary Value

```
 int computeFine(int daysLate, boolean printOn){
1   int MAX_FINE_PERIOD=21; fine=0;
2   if (daysLate<=MAX_FINE_PERIOD){
3     fine = daysLate * DAILY_FINE;
    }
4   logFine(fine);
5   if (printOn == TRUE){
6     printFine(fine);
    }
7   return fine;
 }
```



Test Case #	dayLate	printOn	Path
1	1	TRUE	1-2-3-4-5-6-7

Figure from Braude & Bernstein

```
 int computeFine(int daysLate, boolean printOn){
1   int MAX_FINE_PERIOD=21; fine=MAX_FINE;
2   if (daysLate<=MAX_FINE_PERIOD){
3     fine = daysLate * DAILY_FINE;
    }
4   logFine(fine);
5   if (printOn == TRUE){
6     printFine(fine);
    }
7   return fine;
  }
```

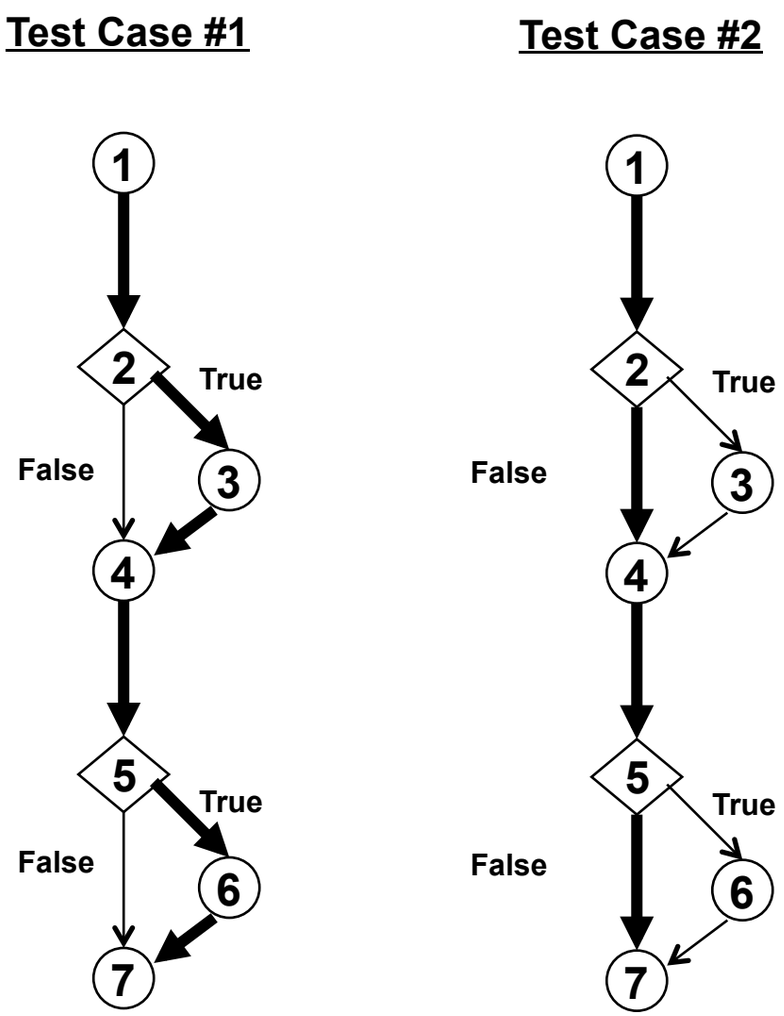


Figure from Braude & Bernstein

Path Coverage

- All distinct code paths are executed by at least one test case
- Complete set of sequences of branches and loop traversals
- Subsumes statement and branch coverage

Designing Test Harness

- JUnit
  - *Test
- Tester library
  - Example*
- Design own test harness
  - Test*

Assignments 2 & 3

- Assignment 2
  - TestMyList.java
  - Due Tuesday, January 21, 2014 at 11:59 pm
- Assignment 3
  - MyList implementation
  - Due Friday, January 24, 2014 at 11:59 pm

Abstraction Barrier

Client

- Knows the behavior of the data type
- Doesn't know how the data type was implemented, but can use the data type based on the specs

Abstraction Barrier

Implementor

- Knows the behavior of the data type
- Knows how the data type was implemented

Debugging

Writing New Code

- write tests first
- write a small amount of code
  - then test it
  - repeat

Modifying Code

- write new tests
- make a small change;
  - test it;
  - repeat

Debugging Code

- Do ***not*** make random or extensive changes to the program!
- Instead, examine the code to figure out ***what*** went wrong
  - Which tests are failing?
  - Find a test that is failing - is it repeatable? (sometimes it is not)
- ***Think*** before you change anything
- Figure out ***why*** it went wrong

Attitude Counts!