CISC 3120

C07: Class Projects, and Testing and JUnit

Hui Chen

Department of Computer & Information Science

CUNY Brooklyn College

Outline

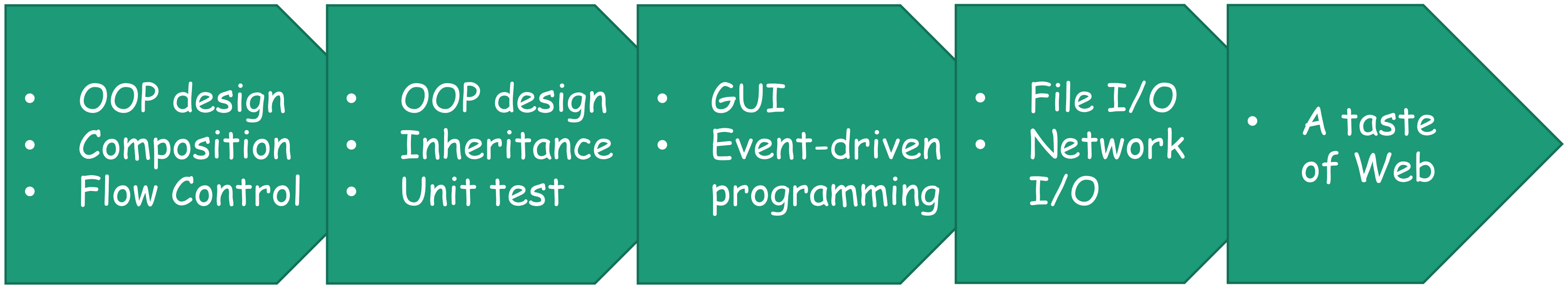
- Recap and issues
  - Grades and feedback
  - Practice assignments and CodeLabs
  - Review guide and Test #1
- Project 1 and Semester Project Road Map
- Verification and Testing
- Unit Tests and JUnit
- Assignments
  - Project 1
  - Practice assignment (JUnit)

Recap and Issues

- Any questions?
  - Grades and feedback
  - Practice assignments and CodeLabs
  - Review guide and Test #1

Design & Implement Applications

- 5 team projects (phases) to build a Treasure Hunt game application
  - Each built from start-up code or built from previous project (phase)
  - Text-based to GUI
  - Standalone to networked/distributed



Project 1

- Start to build the Treasure Hunt game application
  - A simple desktop application with text-based user interface (a.k.a., the command line user interface)
  - Game rules
    - A "treasure" is buried in a field, unearthing the treasure earns the player a score
    - Clues are given when the player solves a puzzler

Project 1: Objectives

- Design, implement, and test a simple Java application
- Apply the “composition” pattern
- Observe the “single responsibility principle”
- Use simple data structures and algorithms
- Be proficient in flow controls

Design Applications: Classes & Objects

- Single responsibility principle
  - *A class should have only a single responsibility and responsible for its own behavior*
  - Objects interacts with only their methods

Treasure Hunt

- The game
  - Game interface (Game frame)
  - Player scores
- One class or two classes? Your choice?
  1. One class controls game interface and keeps track of player scores
  2. Two classes, one controls game interface, the other keeps track of player scores

Project 1 Start-up Code

- To develop and enhance the “Treasure Hunt” from the start-up code
    - https://github.com/CISC3120SP18/ProjectStartupCode/tree/master/TreasureHuntOnConsole

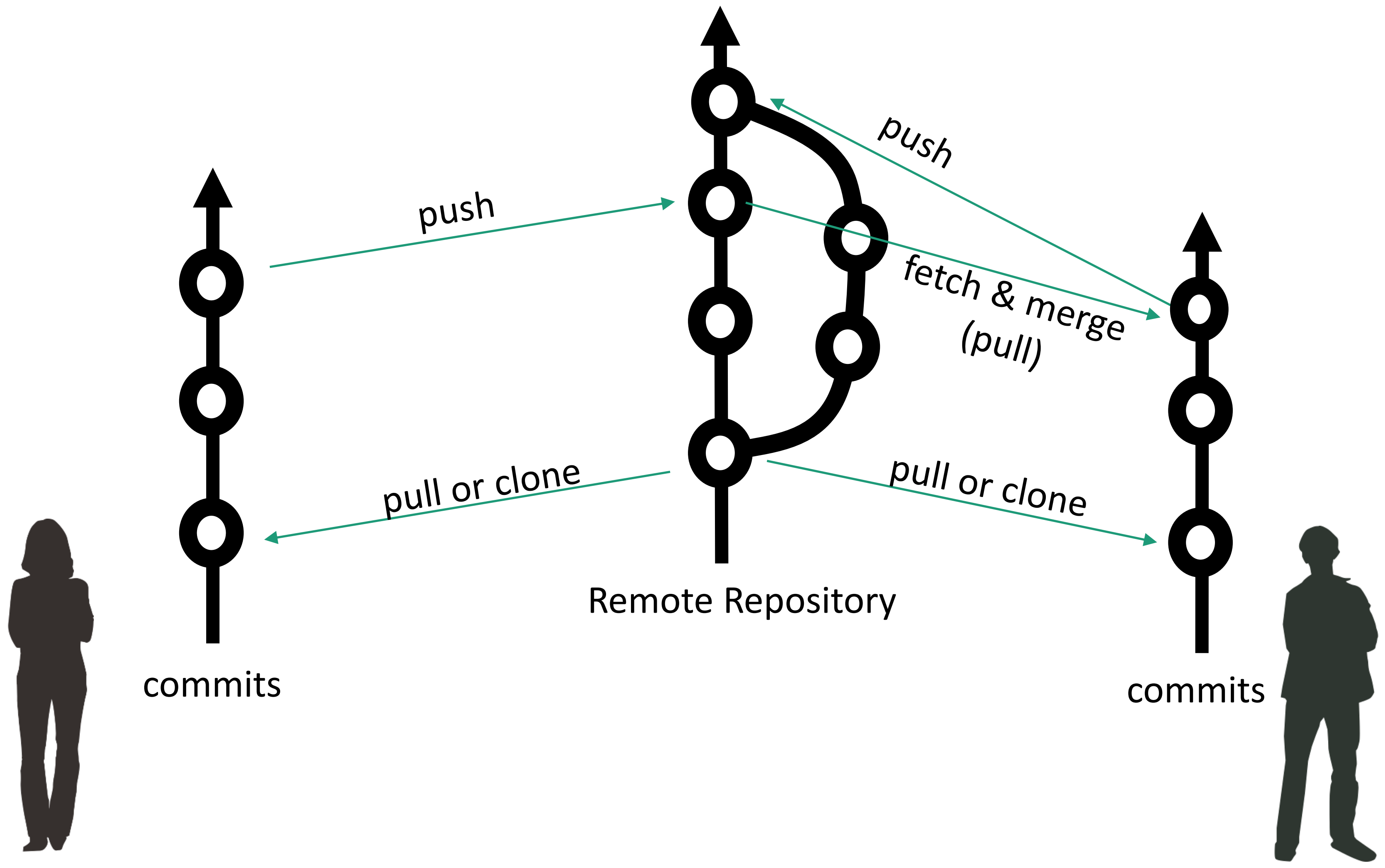
Project 1 In-Class Team Discussion

- Select project coordinator: coordinator's responsibility
    - Accept the assignment
    - Clone the team project repository
    - Copy and add the start-up project to your own project repository (if not already done by Github)
    - Commit and push the project
- Members' responsibility
    - Clone the repository and make a contribution as a team member
- Discuss initial tasks and steps:
    - Use Github issue tracking to create tasks (issues) and assign them team members
- Conflict resolution in Git

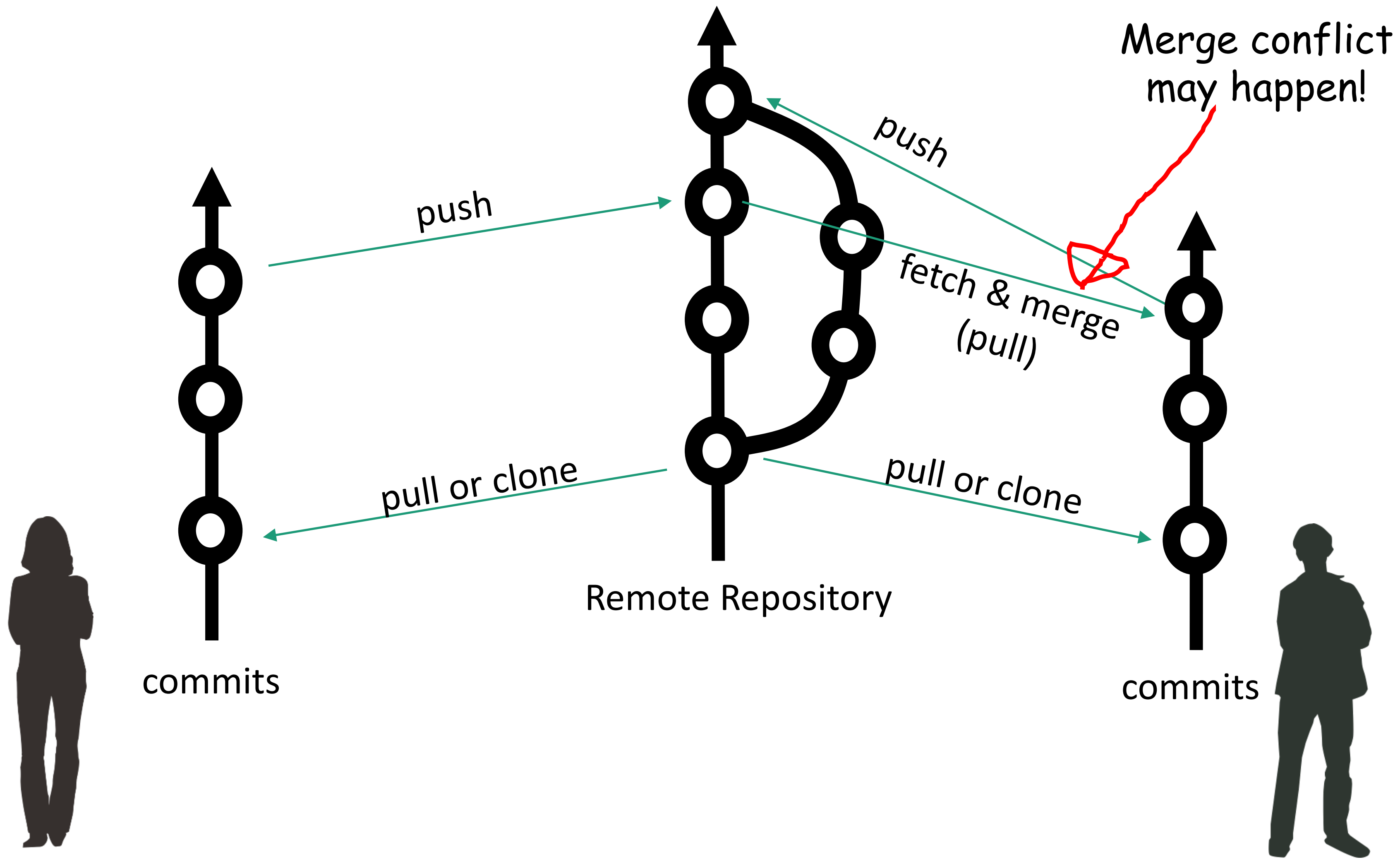
Conflict Resolution in Git

- Have you experienced it?
  - Have you experienced that Git rejects your push?

Distributed Workflow

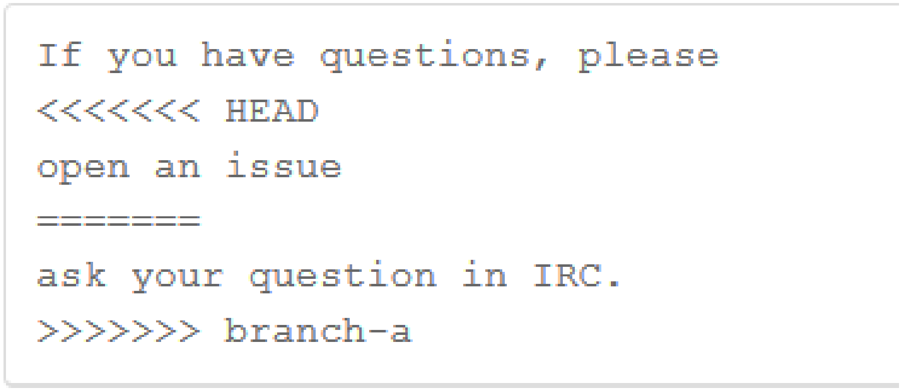


Merge Conflict



Merge Conflict

- When it happens
  - Your and your team member made changes to the same file.
  - The remote repository carries your team member's change
  - Your local repository (committed made) contains you change
- The merge attempt results the file having both changes and marked with markers
  - Edit the file
  - Add and commit the edit file
  - Push it again



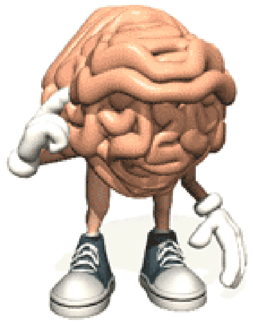
```
If you have questions, please
<<<<<<< HEAD
open an issue
=======
ask your question in IRC.
>>>>>>> branch-a
```

Work in team using Git and Github

- Do not create named branches
- A GitHub guide (well written) is linked in the class website for merge
- Do use commit commenting and GitHub issue tracking

Questions?

- Project 1



"A well built physique is a status symbol. It reflects you worked hard for it, no money can buy it. You cannot borrow it, you cannot inherit it, you cannot steal it. You cannot hold onto it without constant work. It shows discipline, it shows self respect, it shows patience, work ethic and passion. That is why I do what I do."

--Arnold Schwarzenegger

Software Failures

- Ariane 5 rocket explosion (June 4, 1996)
  - "Overflow from conversion from a 64-bit floating point number to a 16-bit signed integer value …"
- Therac-25 lethal radiation overdose (June 1985 ~ Jan 1987)
  - "Some basic software engineering principles were apparently violated …"
- Mars Climate Orbiter disintegration (December 11, 1998)
  - "... In the case of the ground software, … was in English units of pounds (force)-seconds (lbf-s) rather than the metric units specified …"
- FBI Virtual Case File project abandonment (2000 ~ 2005)
  - " …a systematic failure of software engineering practices …"

Recent Stories in the Airlines Industry

- "The big computer systems that get airplanes, passengers and baggage to their destinations every day are having a bad summer." (NY Times, August 8, 2016)
  - 1,000 Of 6,000 Delta flights canceled, August 8-9, 2016
  - 2,300 canceled flights over 4 days at Southwest Airlines, ~July 22, 2016
  - Hundreds of flights grounded at United Airlines, July 2015
  - An iPad software glitch caused two days of problems for American Airlines, the airline said Wednesday …

Software Quality Assurance

- Verification
  - Did you build the thing right? (Did you meet the specification?)
- Validation
  - Did you build the right thing? (Is this what the customer wants? That is, is the specification correct?)
- Two approaches
  - Testing
  - Formal methods

Formal Methods for Software Verification

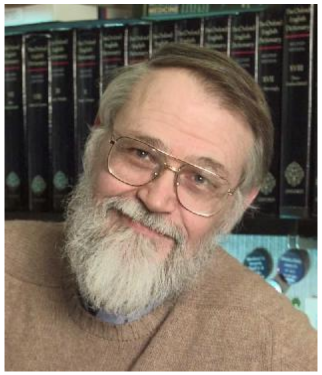
- Start with a formal specification and prove code behavior follows that of specification
  - Mathematical proofs
  - Humans do the proofs
  - Computers do the proofs
    - Automatic theorem proving
    - Model checking
  - In practice, mostly done for hardware; done in very limited cases for software
    - Computational intensive: hard to test, not too large spec., error repair cost prohibitive, critical components or systems

Model Checking

- Determines whether a hardware or software design satisfies a formal specification
  - Design expressed in an abstract model
  - Formal specification expressed as a temporal logic formula
  - An algorithmic means
- Identifies a counter-example execution showing the source of the problem if the property does not hold
- Examples:
  - Verification of VLSI circuits
  - Communication protocols
  - Software device drivers,
  - Real-time embedded systems
  - Security algorithms

Edmund M. Clarke, E. Allen Emerson, and Joseph Sifakis

Testing for Software Verification



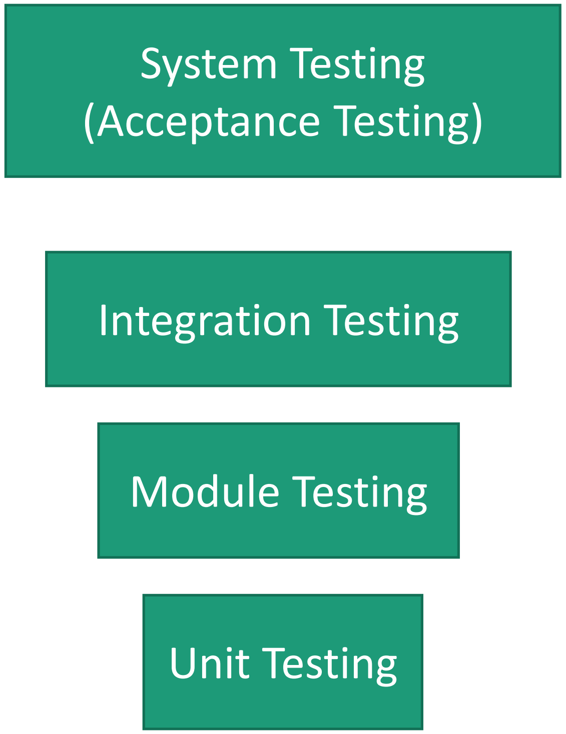
- Everyone knows that debugging is twice as hard as writing a program in the first place. So if you're as clever as you can be when you write it, how will you ever debug it? -- Brian Kernighan
- Programming testing can be used to show the presence of bugs, but never to show their absence! – Edsger W. Dijkstra

Testing

- Exhaustive testing is infeasible
  - e.g., 1 nanosecond for test a program that has one 64-bit input, how long does it take to test all possible input values?
- Reduce the space for testing
  - Perform different tests at different phases of software development

Different Types of Tests

- System testing (acceptance testing): if the integrated program meets its specification
- Integration testing: interfaces between units have consistent assumption and communicate correctly
- Module testing: tests across individual units (e.g., across classes)
- Unit testing: single method does what was expected (e.g., within a single class)



Perspectives: Black-box vs. White-box Tests

- Black-box tests
  - Test design is solely based on the program’s external specifications
- White-box (glass-box) tests
  - Test design reflects knowledge about the program’s implementation, e.g., developers’ doing the tests
- For it or against it?

Test Coverage

- The fraction of the possible program execution paths that have been tested
  - 100% of test coverage is no guarantee of design reliability
  - Quality of tests also matters

Common Test Coverage Levels

- S0 (method coverage)
    - Is every method executed at least once by the test suite?
- S1 (call coverage or entry/exit coverage)
    - Has each method been called from every place it can be called?
- C0 (statement coverage)
    - Is every statement of the source code executed at least once by the test suite?
- C1 (branch coverage)
    - Has each branch been taken in each direction at least once?
- C2 (path coverage)
    - Has every possible route through the code been executed?

Sample Code to Test

```
public class MyClass {
   public void foo(boolean x, boolean y, boolean z) {
     if (x)
        if (y && z) bar(0);
        else
           bar(1);
   }
  public boolean bar(x) {
     return x;
   }
}
```

Test Coverage S0

```
public class MyClass {
    public void foo(boolean x, boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
  }
}
```

Satisfying S0 requiring calling foo and bar at least once each in the tests

Test Coverage S1

```
1.    public class MyClass {
2.       public void foo(boolean x,
      boolean y, boolean z) {
3.          if (x)
4.             if (y && z) bar(0);
5.             else
6.                bar(1);
7.      }
8.     public boolean bar(x) {
9.         return x;
10.     }
11.   }
```

Satisfying S1 requiring calling bar from both line 4 and line 6 in the test suites

Test Coverage C0

```
public class MyClass {
    public void foo(boolean x,
boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
  }
}
```

Counting both branches of a conditional as a single statement, satisfying C0 requiring calling foo at least once with x true, and at least once with y false

Test Coverage C1

```
public class MyClass {
    public void foo(boolean x, boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
  }
}
```

Satisfying C1 requiring calling foo at least once with x true, and with x false, and with y && z true and false.

Test Coverage C2

```
public class MyClass {
    public void foo(boolean x,
boolean y, boolean z) {
        if (x)
            if (y && z) bar(0);
            else
                bar(1);
    }
  public boolean bar(x) {
      return x;
  }
}
```

Satisfying C2 requiring calling foo with all 8 combinations of values of x, y, and z

Modified Condition/Decision Coverage (MCDC)

- Combines a subset of the above levels
  - Each point of entry and exit in the program have been invoked at least once
  - Every decision in the code has taken all possible outcomes at least once
  - Each condition in a decision has been shown to independently affect that decision's outcome

Achieving Test Coverage

- 100% of C0 coverage is not unreasonable.
- Achieving C1 coverage requires careful construction of tests.
- C2 is the most difficult of all, and the additional value of 100% of C2 is debatable.

Questions?

- Examples of catastrophic software failures
- Software quality assurance
- A few important concepts in software testing

Unit and Functional Testing

- Recall …
  - Unit testing: single method does what was expected (e.g., within a single class)
  - Functional testing: a well-defined subset of the code does what was expected (e.g., several methods and classes)
- Tests
  - Calls to methods with different input parameters
  - Asserts on the effects of method calls
  - Aims for high coverage
  - Almost always white-box, and performed by developers

Test Assertion

- An expression encapsulates some testable logic about a target under test

JUnit

- A unit testing framework for Java
- Test assertion in JUnit
  - It throws an exception if it evaluates to false

Unit Test Example with JUnit

- Some of you completed the Array and ArrayList assignment
- Let's use it as an example
  - Test whether the "delete" method functions as specified.

```
public class FruitArray
{ …
   public void delete(String fruitName) { … }
…
}
```

What is the Specification?

```
public class FruitArray
{ …
   public void delete(String fruitName) { … }
…
}
```

What is the Specification?

- Remove an element, “shrink” the “list”, as if the element had never been in the “list”.

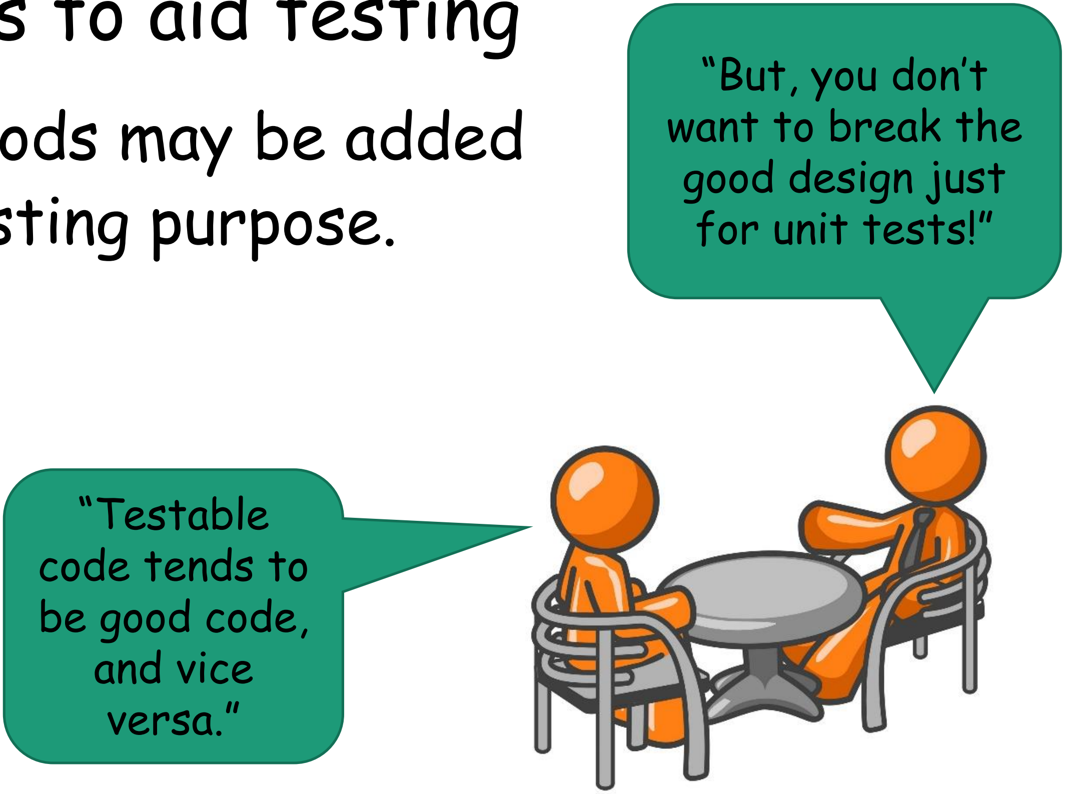
```
public class FruitArray
{ …
   public void delete(String fruitName) { … }
…
}
```

What is the Test Assertion?

- It is expected that upon an item is deleted, the object should be identical to another object of the class that does not have the item from beginning with, but has all the other items.

Design for Testing

- Add methods to aid testing
  - Some methods may be added just for testing purpose.


"But, you don't want to break the good design just for unit tests!"
"Testable code tends to be good code, and vice versa."

Example Implementation in JUnit 4

- Use JUnit 4, assume Maven project in Eclipse using the Quickstart archetype 1.1
- A few steps
  - Update pom.xml

```
<!-- https://mvnrepository.com/artifact/junit/junit -->
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.12</version>
    <scope>test</scope>
</dependency>
```

  - Remove AppTest.Java
  - Create your own test class (See https://github.com/junit-team/junit4/wiki/getting-started)
  - Complete the example in "ArrayArrayList" in the "sampleprograms" repository

FruitArrayTest.java

```java
import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertArrayEquals;
import org.junit.Test;
public class FruitArrayTest
{
  @Test // Important: annotate a test
  public void testDelete()
  {
    testDeleteByDeleting1st();
    testDeleteByDeleting3rd();
    testDeleteByDeletingLast();
    testDeleteIgnoreCaseByDeleting1st();
    testDeleteIgnoreCaseByDeletingLast();
  }
```

```java
private void testDeleteByDeleting1st() {
    String[] fruits = {new String("Apple"),
                       new String("Banana"),
                       new String("Kiwi"),
                       new String("Mango"),
                       new String("Orange")};
    FruitArray fruitArray = new FruitArray(fruits);
    fruitArray.delete(new String("Apple"));
    assertEquals(fruitArray.getSize(), fruits.length - 1);
    assertEquals(fruitArray.getCapacity(), fruits.length);
    assertArrayEquals(fruits, fruitArray.getFruitsAsArray());
  }
```

Note: the purpose of the test is to show how you write tests in a JUnit. This test is actually poorly designed. See the sample code in the repo for better ones.

Did We Pass the Tests?

Questions

- JUnit
- Examples in JUnit 4
  - Update pom.xml
  - Use the sample code as the template or the one in the JUnit 4 wiki page

Assignments

- Practice Assignment
- Project 1