Objectives

- Coverage
  - Strengths, Limitations
  - Tools: EclEmma
- Debugger

1

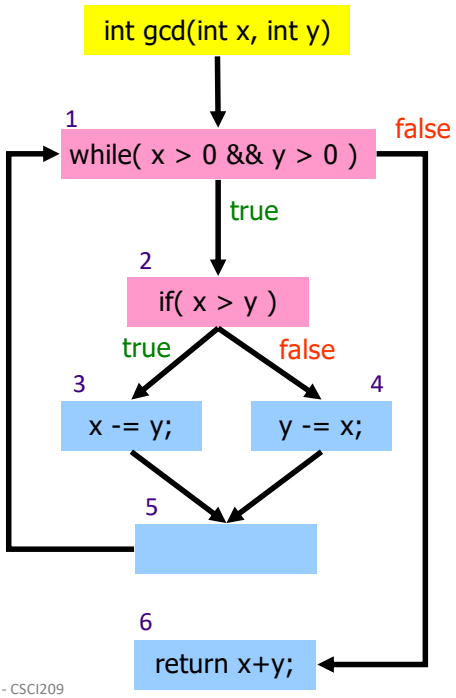
Project Notes

- Do NOT change the Car class's API/the method specifications
  - Your test cases need to compile with my code
- You are writing the test cases that my code needs to pass
  - The tests encode the requirements
- When using assertEquals
  - Parameter order is expected, actual
  - Use a third error tolerance parameter for comparing doubles

2

From Last Time

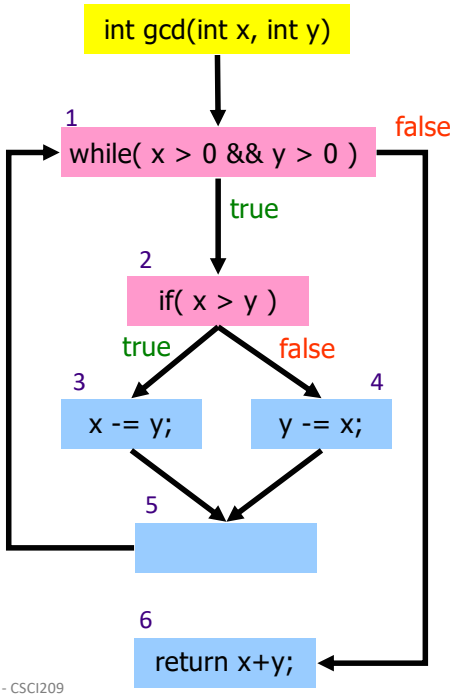
```
/**
 * Euclid's algorithm to calculate
 * greatest common divisor
 */
public int gcd( int x, int y ) {
    while ( x > 0 && y > 0 ) {
        if( x > y ) {
                x -=y ;
        } else {
                y -=x;
        }
    }
    return  x+y;
}
```



Path Coverage

- How many paths through this method?
  - Too many to count, test them all!

1-6
1-2-3-5-1-6
1-2-4-5-1-6
1-2-3-5-1-2-3-5-1-6
1-2-4-5-1-2-4-5-1-6
1-[2-(3|4)-5-1]*-6



Review

1. What are two workflows for collaboration in Git?
   - Even more important since we're working on a team project
2. True or False. Our team's GitHub repository and git are all my team needs to effectively collaborate.
3. How is writing/running the tests for the project different from the tests for the mutant lab?
4. What is code coverage?
5. What code coverage criteria did we discuss?
   - What has been our goal for coverage?
   - Synthesize: Compare the strengths/limitations of each of those criteria
6. Can we use code coverage in the testing project?

5

Review: Collaboration: Workflow – Seeking Feedback

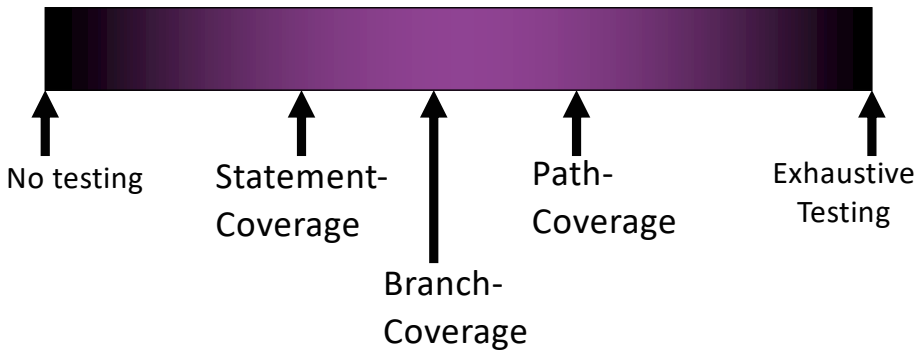
1. Create a branch for your work
   - Commit periodically
   - Write descriptive comments so your team members know what you did and why
2. Push your branch
3. Open a ***Pull Request*** on your branch in GitHub
   - You can tag your teammates to let them know that you've completed your work
   - Team: discuss and review potential changes – can still update
4. Merge pull request into `main` branch
5. Pull the `main` branch to get the latest code

6

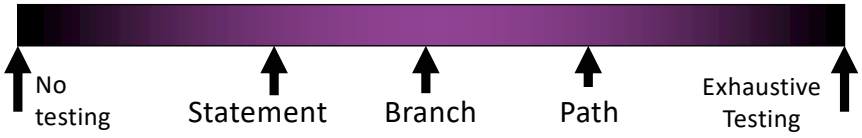
Review: Collaboration: Workflow

1. Create a branch for your work
   - Commit periodically
   - Write descriptive comments so your team members know what you did and why
2. Switch to `main`
3. Pull `main` branch
4. Merge your branch into the `main` branch
   - Handle merge conflicts
   - Commit
5. Push `main` branch

Testing Continuum



Comparison of Coverage



Coverage Criterion	Advantages	Disadvantages
Statement		
Branch		
Path		

For 100% coverage with each criteria

9

Comparison of Coverage

Coverage Criterion	Advantages	Disadvantages
Statement	Practical	Weak, may miss many faults
Branch	Practical, Stronger than Statement	Weaker than Path
Path	Strongest	Infeasible, too many paths to be practical

10

Note: Coverage and Testing Project

- You won't be able to leverage coverage for the testing project
  - Even if you write code for the Car class, it's not *my* code.
- Challenge of test-driven development (TDD)
  - Common practice in industry

11

How Can We Use Coverage Criteria?

12

Uses of Coverage Criteria

- Specify test cases
  - Describe additional test cases needed
- "Stopping" rule → sufficient testing
  - Avoid unnecessary, redundant tests
- Measure test quality
  - Dependability estimate
  - Confidence in estimate

Coverage Criteria Discussion

- Is it always possible for a test suite to cover all the statements in a given program?
  - No. Could be infeasible statements
    - Unreachable code
    - Legacy code
    - Configuration that is not on site
- Do we need the test suite to cover 100% of statements/branches to believe it is adequate?
  - Yes.
  - While 100% coverage does not mean correct program, < 100% coverage means testing inadequacy

True/False Quiz

- A program that passes all test cases in a test suite with 100% path coverage is bug-free.
  - **False.**
  - Examples:
    - The test suite may cover a faulty path with data values that don't expose the fault.
      - Coverage is about *control flow* but *data* plays a role too
    - Errors of omission
      - Missing a whole if

Example

Test Suite:

3-7: a=3

4-6: a=30

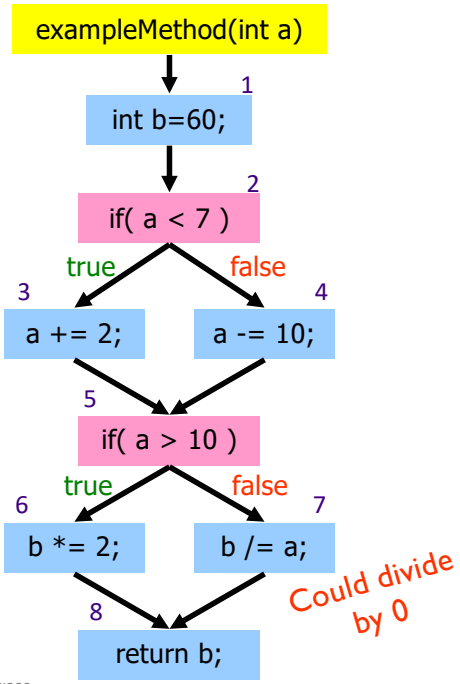
3-6: a=6

4-7: a=9

But, error shows up with

3-7: a=0

4-7: a=10



Example

Test Suite:

3-7: a=3

4-6: a=30

3-6: a=6

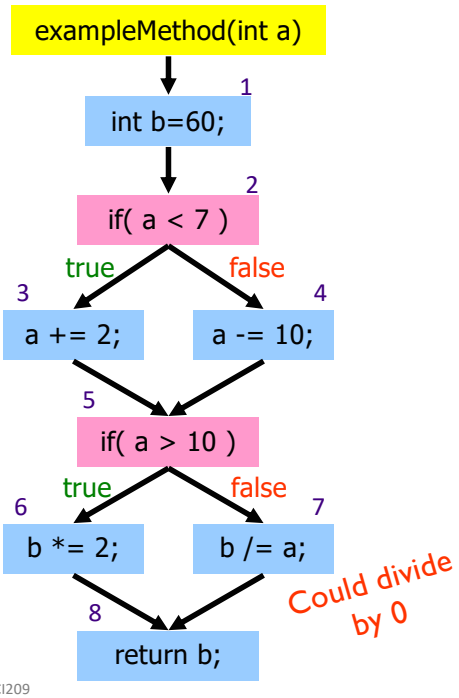
4-7: a=9

But, error shows up with

3-7: a=0

4-7: a=10

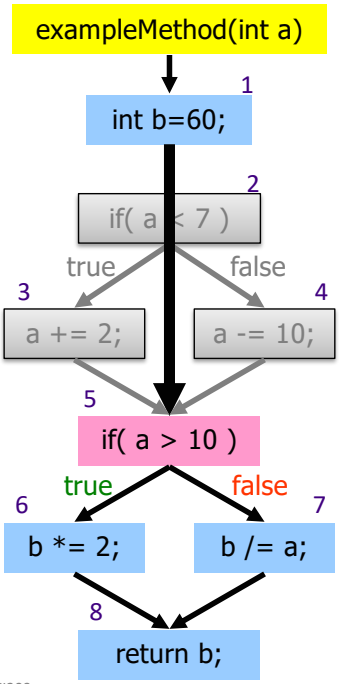
Also an error of omission:
Should have statement 7 within an if statement that checks value of a



Omission Example

Consider if the first `if` block wasn't in the code.

You could cover all the paths, but you're missing a crucial condition.



True/False Quiz

- When you add test cases to a test suite that covers all statements so that it covers all branches, the new test suite is more likely to be better at exposing faults.
  - **True**.
  - You're adding test cases and covering new paths, which may have faults.
    - Also, may be using new values that expose faults

Which Test Suite Is Better?

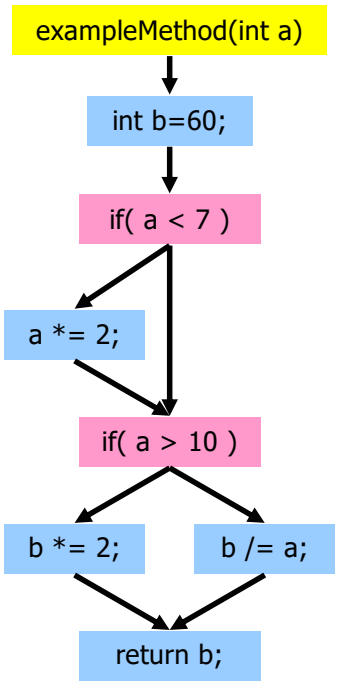
Statement-adequate Test Suite

Branch-adequate Test Suite

- Branch-adequate suite is not *necessarily* better than Statement-adequate suite
  - Statement-adequate suite could cover paths with faults and include input values that Branch-adequate suite doesn't

Example

- TS1 (Statement-Adequate):
  - a=0, 6
- TS2 (Branch-Adequate):
  - a=3, 30
- Statement-adequate will find fault but branch- adequate won't
  - Covers the path that exposes the fault



Software Testing: When is Enough Enough?

- Need to decide when tested enough
  - Balance goals of releasing application, high quality standards
- Can use program coverage as "stopping" rule
  - Also measure of *confidence* in test suite
  - Statement, Branch, Path and their tradeoffs
  - Use coverage tools to measure statement, branch coverage
- Still, need to use some other "smarts" besides program coverage for creating test cases

No Silver Bullet

- Recall the Fred Brooks' quote:
  - "There is no single development, in either technology or in management technique, that by itself promises even one order-of-magnitude improvement in productivity, in reliability, in simplicity."
  - Known as "no silver bullet"
- Test coverage is one tool that will help us improve the quality of our code, but it will not solve everything

COVERAGE TOOLS

Coverage Tools

- Coverage is used in practice
- Don't need to figure out coverage manually
- Available tools to calculate coverage
  - Examples for Java programs: Cobertura, Clover, JCoverage, **Emma**
  - Measure statement, branch/conditional, method coverage

Eclipse Plugin: EclEmma for Coverage

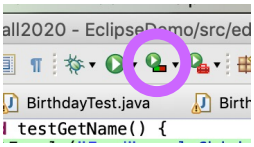
- Eclipse can be extended through ***plugins***
  - Provide additional functionality
- EclEmma Plugin
  - Records executing program's (or JUnit test case's) coverage
  - Displays coverage graphically
- Built into Enterprise Edition of Eclipse
  - What you were supposed to install
  - If you got the regular version of Eclipse, you'll need to install the EclEmma plugin

Installing Emma in Eclipse

1. From your Eclipse menu select *Help → Eclipse Marketplace*.
2. Search for "EclEmma".
3. Hit *Install* for the entry "EclEmma Java Code Coverage".
4. Follow the steps in the installation wizard.

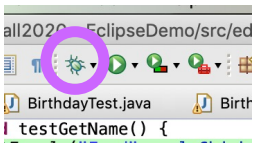
Demonstration

- Execute test with coverage



ECLIPSE DEBUGGER

Eclipse Debugger



1. Set breakpoint
   - Near and BEFORE point of failure
2. Run program in debug mode
3. Inspect variables
4. Step through program, inspecting variables
   - Step into, over, and return

More Testing Tools, Frameworks

- "Tasty mocking framework for unit tests in Java"
- Mock objects before have other code
- Allows you to test in isolation, e.g., mock the payment system so you focus on your code

- Cucumber
  - Behavior-driven development
  - Language parser: Gherkin

- Many more

Looking Ahead

- Testing Project due
  - Has an FAQ
    - Updated as I get more questions
  - Due Tuesday at 11:59 p.m.
- Monday:
  - Design in the Small