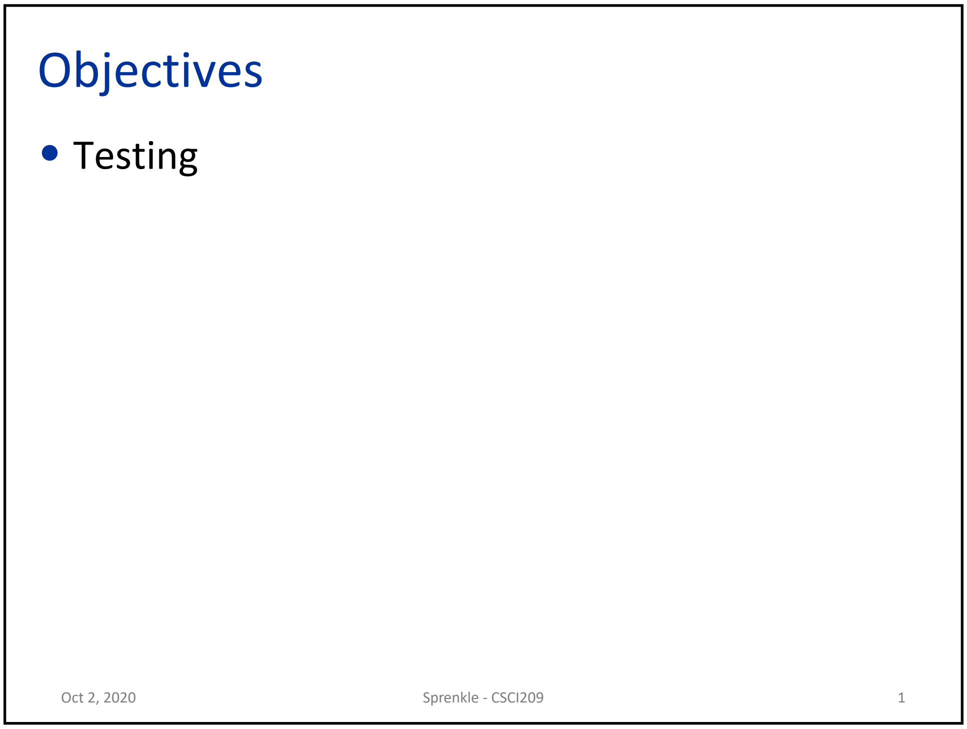
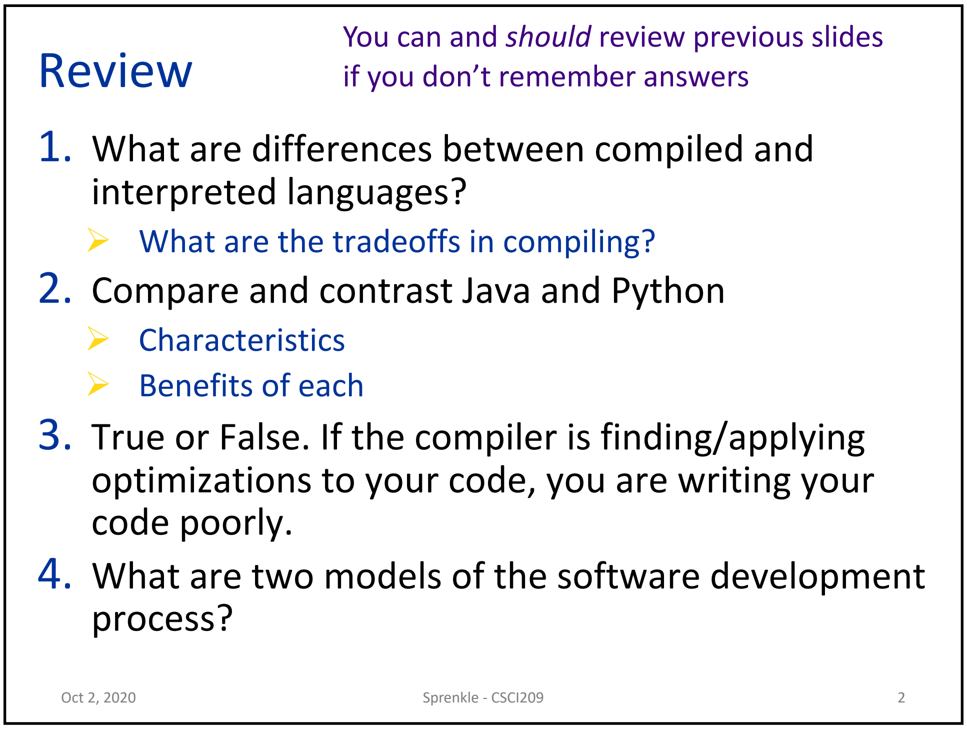


Objectives

- Testing

Oct 2, 2020 Sprenkle - CSCI209 1

1



Review

You can and *should* review previous slides if you don’t remember answers

1. What are differences between compiled and interpreted languages?
   - What are the tradeoffs in compiling?
2. Compare and contrast Java and Python
   - Characteristics
   - Benefits of each
3. True or False. If the compiler is finding/applying optimizations to your code, you are writing your code poorly.
4. What are two models of the software development process?

Oct 2, 2020 Sprenkle - CSCI209 2

2

Review: Compiled vs Interpreted Languages

In pure forms

Compiled

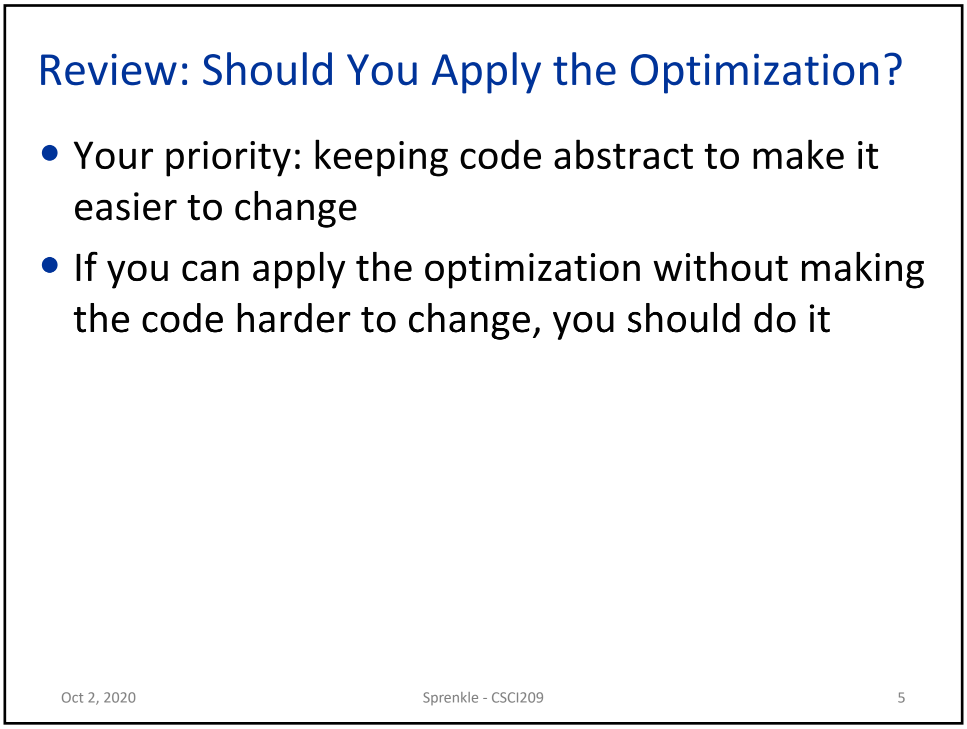
- − Spends a lot of time analyzing and processing the program
- Resulting executable is some form of machine- specific binary code
- Computer hardware interprets (executes) resulting code
- ✓ Program execution is fast
  - ➢ Efficient machine/byte code generation
  - ➢ Performance gains

Interpreted

- ✓ Relatively little time spent analyzing and processing the program
- Resulting code is some sort of intermediate code
- Another program interprets resulting code
- − Program execution is relatively slow
- ✓ Faster development/prototyping

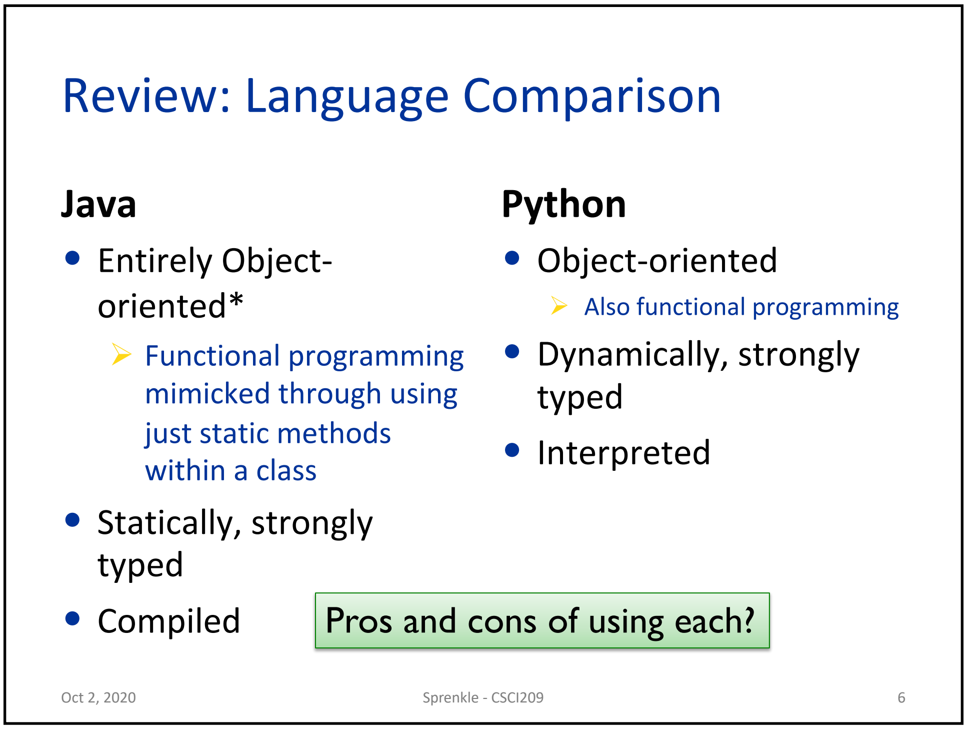
Review: Compiler Tradeoffs

- Upfront costs
  - ➢ Searching for optimizations
  - ➢ Make optimizations
    - Typically not Big-Oh efficiency improvements (unless program is really inefficient)
- Improved runtime
  - ➢ Expect executed many more times than compiled



Review: Should You Apply the Optimization?

- Your priority: keeping code abstract to make it easier to change
- If you can apply the optimization without making the code harder to change, you should do it



Review: Language Comparison

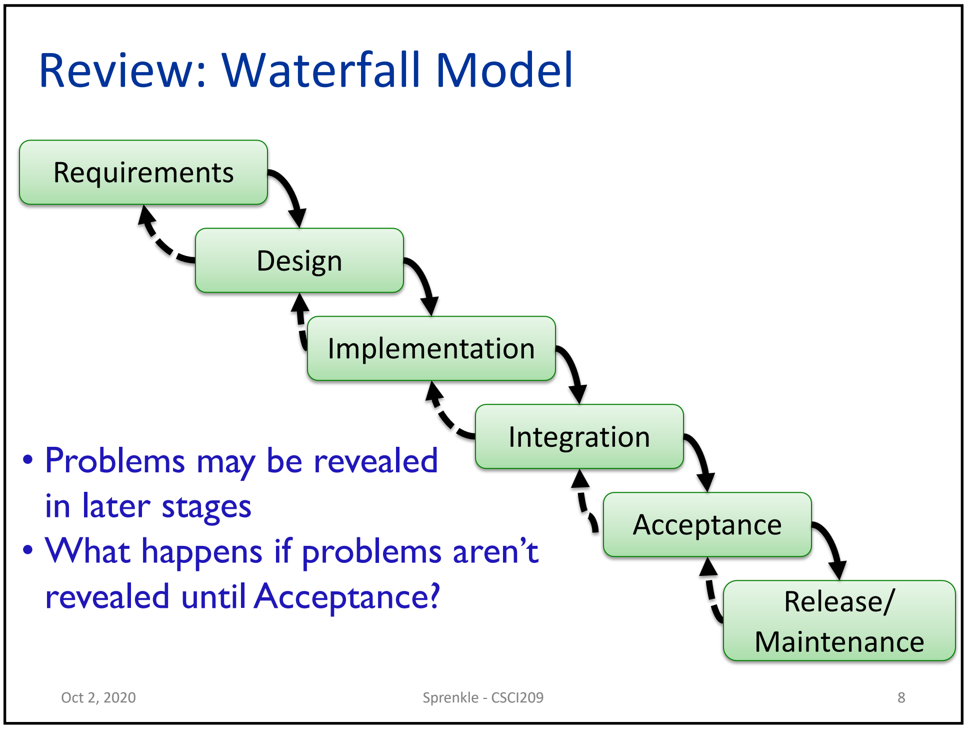
Java

- Entirely Object-oriented*
  - Functional programming mimicked through using just static methods within a class
- Statically, strongly typed
- Compiled

Python

- Object-oriented
  - Also functional programming
- Dynamically, strongly typed
- Interpreted

Pros and cons of using each?



Review: Waterfall Model

Requirements

Design

Implementation

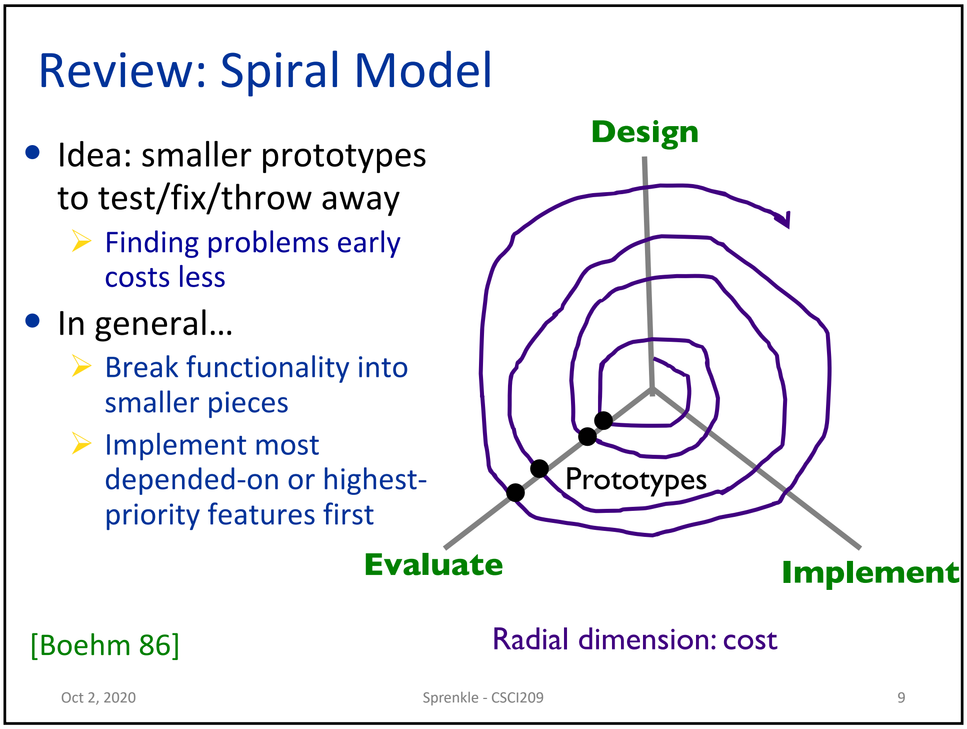
Integration

Acceptance

Release/ Maintenance

- Problems may be revealed in later stages
- What happens if problems aren't revealed until Acceptance?

8



Review: Spiral Model

- Idea: smaller prototypes to test/fix/throw away
  - Finding problems early costs less
- In general…
  - Break functionality into smaller pieces
  - Implement most depended-on or highest-priority features first

Design

Prototypes

Evaluate

Implement

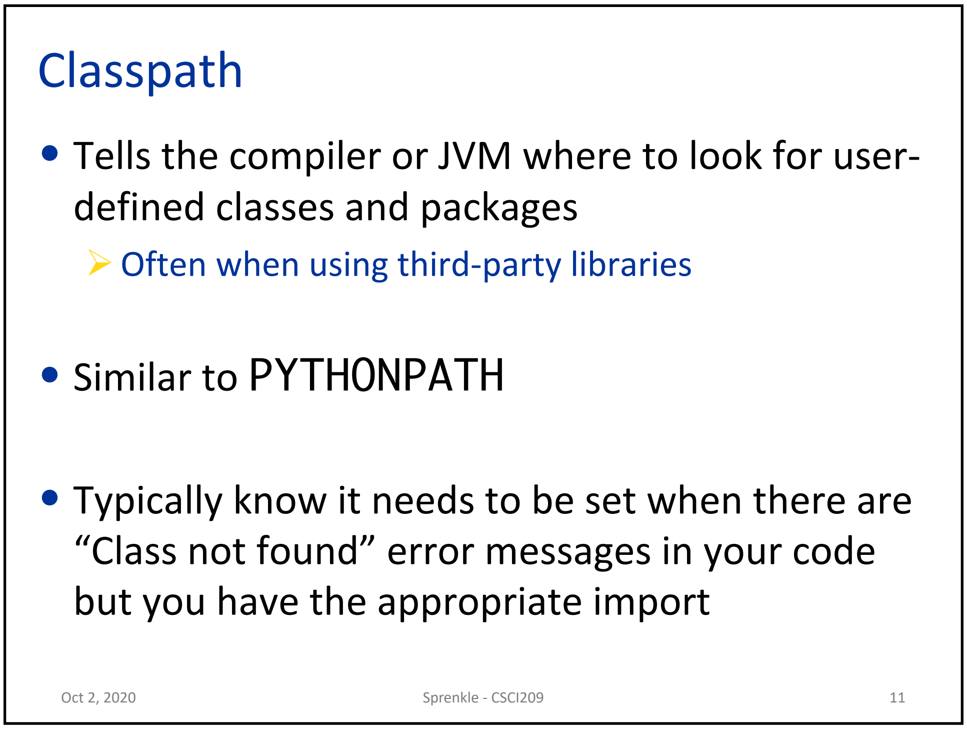
[Boehm 86]

Radial dimension: cost

9

CLASSPATH

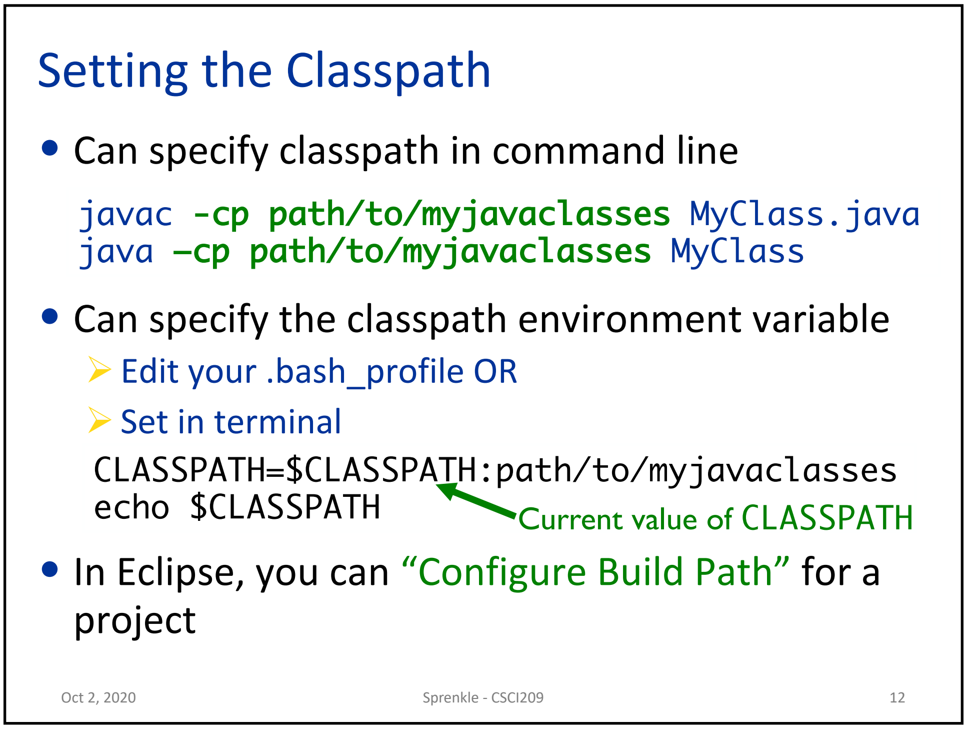
10



Classpath

- Tells the compiler or JVM where to look for user-defined classes and packages
    - ➢ Often when using third-party libraries
- Similar to `PYTHONPATH`
- Typically know it needs to be set when there are "Class not found" error messages in your code but you have the appropriate import

11



Setting the Classpath

- Can specify classpath in command line

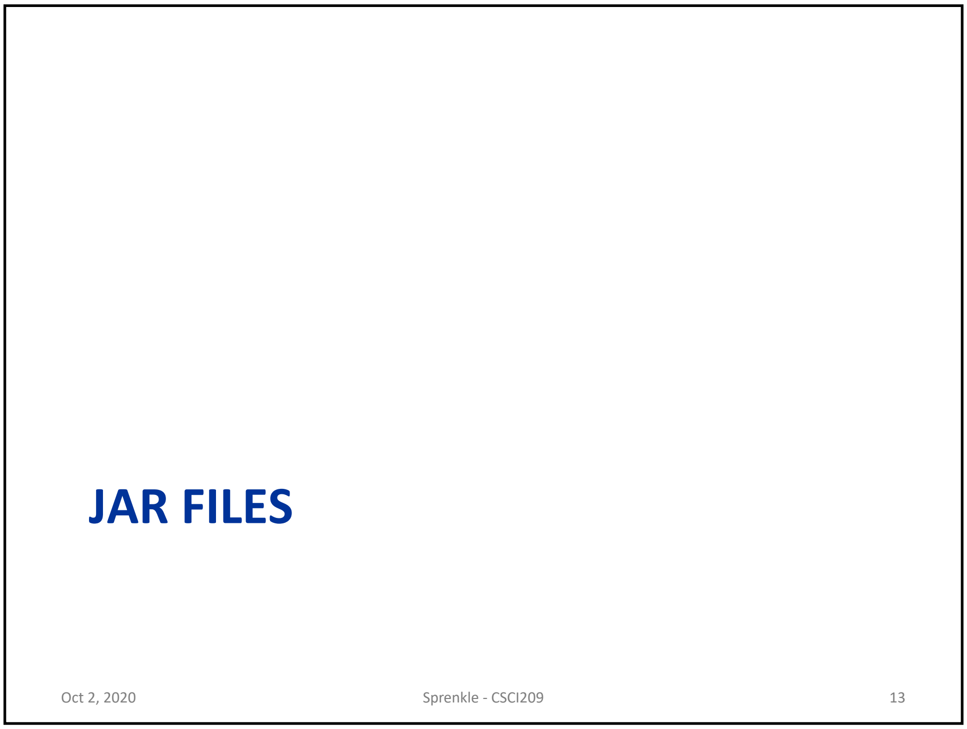
```
javac -cp path/to/myjavaclasses MyClass.java
java –cp path/to/myjavaclasses MyClass
```

- Can specify the classpath environment variable
  - Edit your .bash_profile OR
  - Set in terminal

```
CLASSPATH=$CLASSPATH:path/to/myjavaclasses
echo $CLASSPATH
```

Current value of CLASSPATH

- In Eclipse, you can “Configure Build Path” for a project



JAR FILES

Jar (Java Archive) Files

- Archives of Java files
- Package code into a neat bundle to distribute
  - Easier, faster to download
  - Easier for others to use
- `jar` command: create, view, and extract Jar files
  - Works similarly to `tar`

```
jar cf myapplication.jar *.class
```

- Run it using java

```
java -jar myapplication.jar
```

- Can include jar files in CLASSPATH

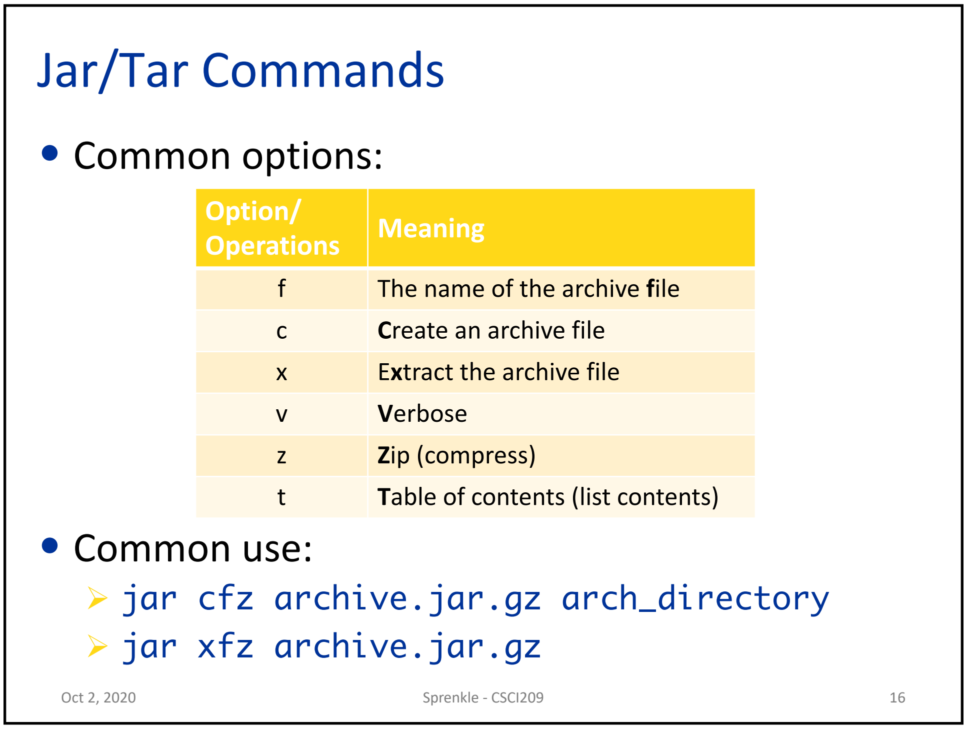
Examples from Class

- I provided you with the Game.jar file
  - Contained the .class files of my version of the code
- To run, you used the command

```
java -cp Game.jar Game
```

The name of the class to execute

- For today's lab, provided mutants.jar
  - Class files of the mutant versions of the implementation
  - Added to Eclipse's classpath



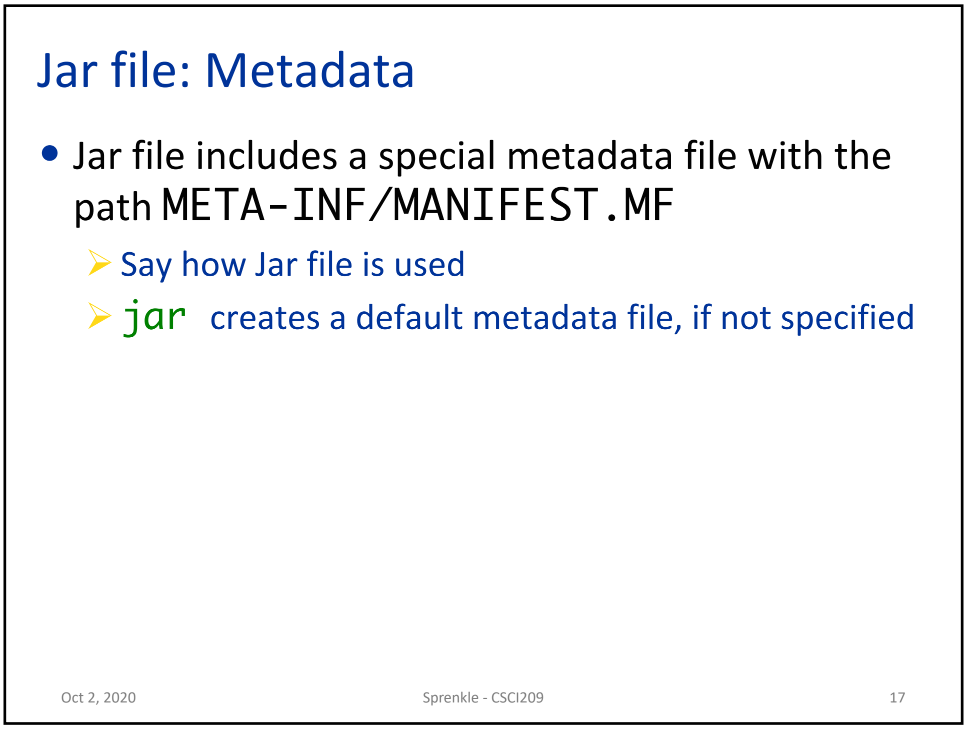
Jar/Tar Commands

- Common options:

Option/ Operations	Meaning
f	The name of the archive **f**ile
c	**C**reate an archive file
x	E**x**tract the archive file
v	**V**erbose
z	**Z**ip (compress)
t	**T**able of contents (list contents)

- Common use:
  - `jar cfz archive.jar.gz arch_directory`
  - `jar xfz archive.jar.gz`

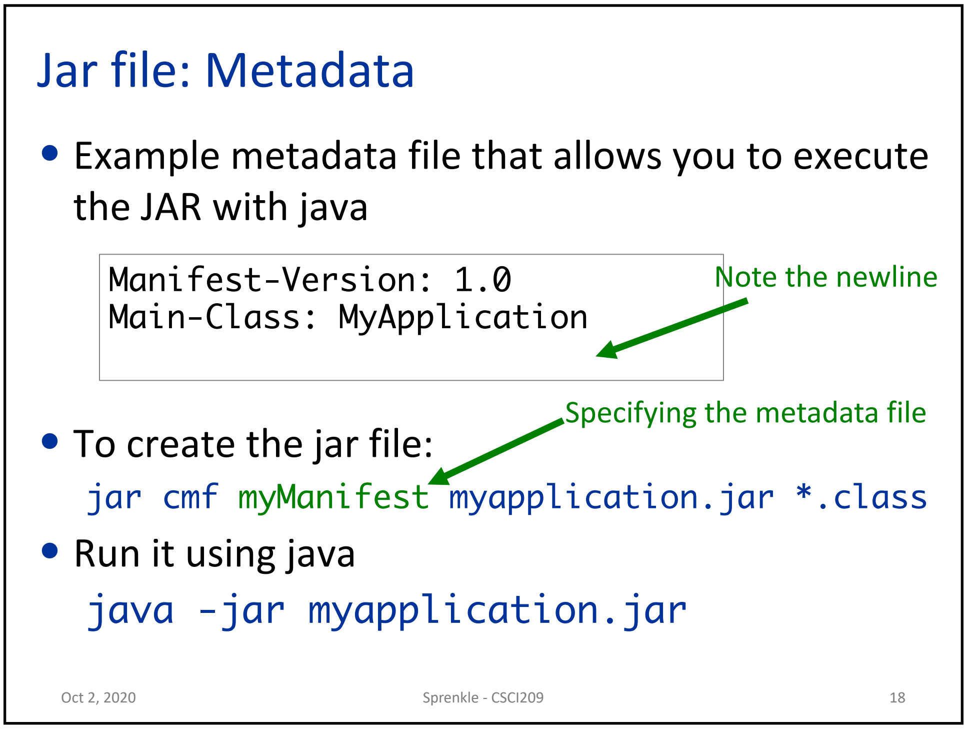
16



Jar file: Metadata

- Jar file includes a special metadata file with the path `META-INF/MANIFEST.MF`
  - Say how Jar file is used
  - `jar` creates a default metadata file, if not specified

17



Jar file: Metadata

- Example metadata file that allows you to execute the JAR with java

```
Manifest-Version: 1.0
Main-Class: MyApplication

```

Note the newline

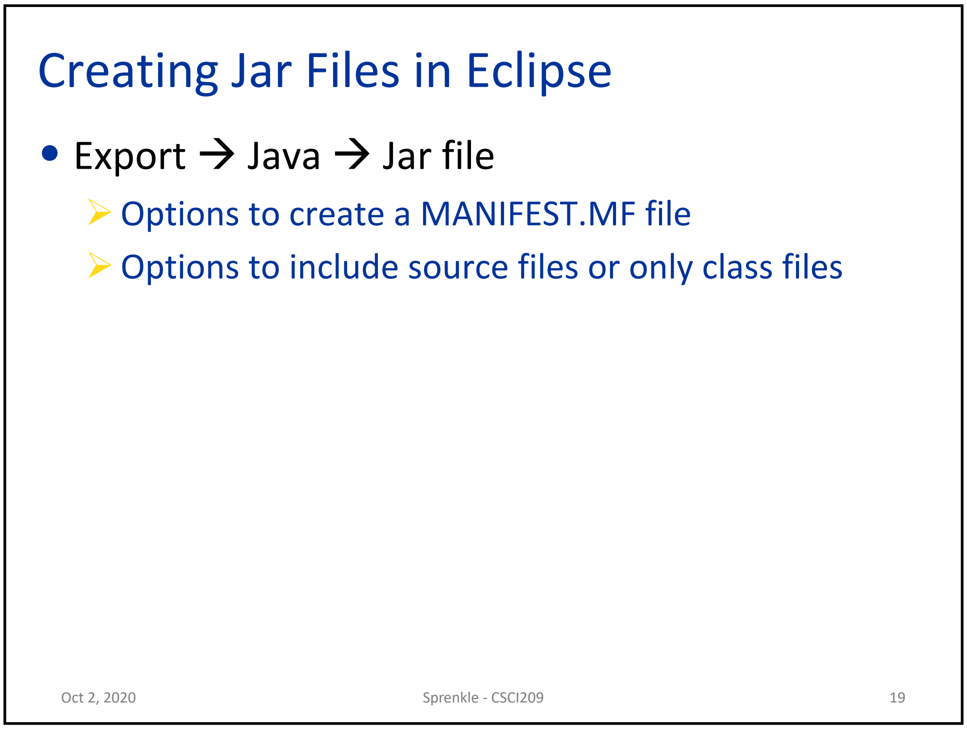
- To create the jar file:

Specifying the metadata file

```
jar cmf myManifest myapplication.jar *.class
```

- Run it using java

```
java -jar myapplication.jar
```



Creating Jar Files in Eclipse

- Export → Java → Jar file
  - Options to create a MANIFEST.MF file
  - Options to include source files or only class files

SOFTWARE TESTING PROCESS

A Bad Role Model

I DON'T ALWAYS TEST MY CODE
BUT WHEN I DO, I DO IT IN PRODUCTION. STAY ON CALL MY FRIENDS
Troll.me

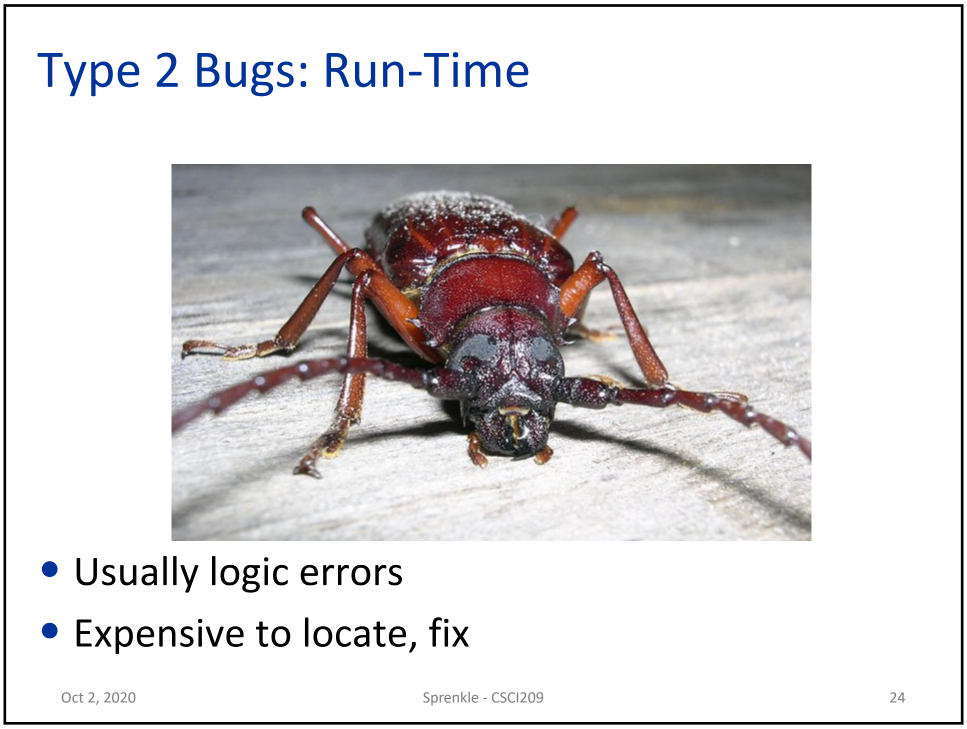
http://imgur.com/HBSbn

Microsoft Windows Vista Testing

- Beyond their internal testing …
  - 5 million people beta tested
  - 60+ years of performance testing
  - 1 Billion+ Office 2007 sessions
- Still, users found correctness, stability, robustness, and security bugs

Type 1 Bugs: Compile-Time

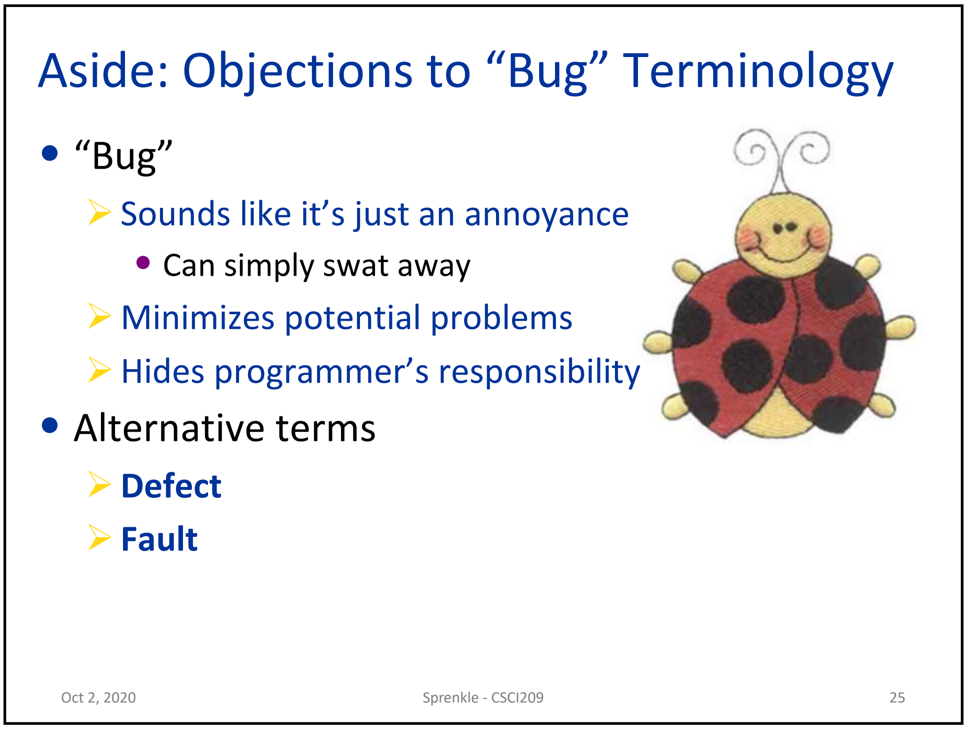
- Syntax errors
  - Missing semicolon, parentheses
- Compiler notifies of error
- Cheap, easy to fix



Type 2 Bugs: Run-Time

- Usually logic errors
- Expensive to locate, fix

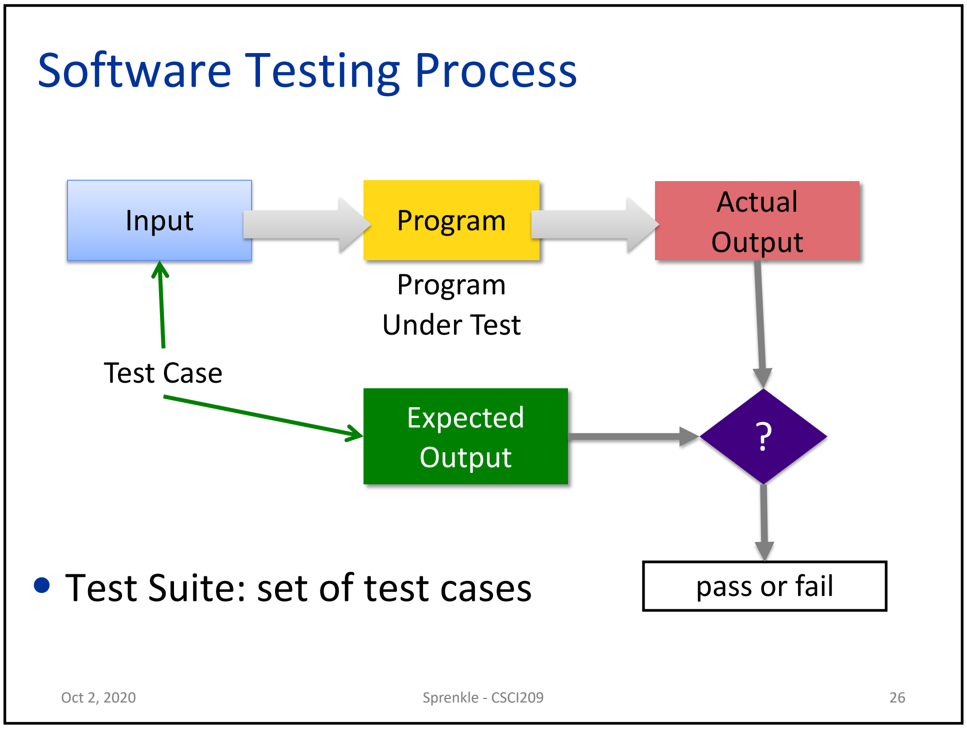
24



Aside: Objections to “Bug” Terminology

- “Bug”
  - Sounds like it’s just an annoyance
    - Can simply swat away
  - Minimizes potential problems
  - Hides programmer’s responsibility
- Alternative terms
  - **Defect**
  - **Fault**

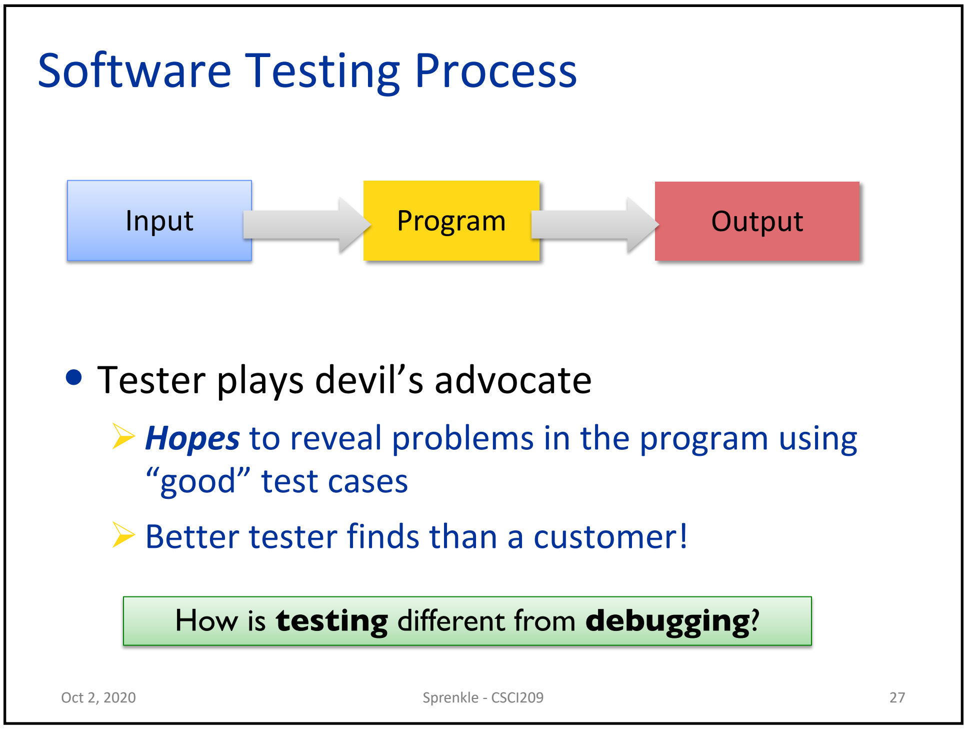
25



Software Testing Process

- Test Suite: set of test cases

26



Software Testing Process

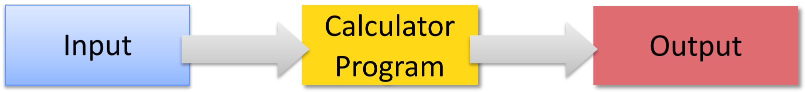
Input
Program
Output

- Tester plays devil’s advocate
  - ***Hopes*** to reveal problems in the program using “good” test cases
  - Better tester finds than a customer!

How is **testing** different from **debugging**?

27

How Would You Test a Calculator Program?



Operands, operators, expected output

adds, subtracts, multiplies, divides

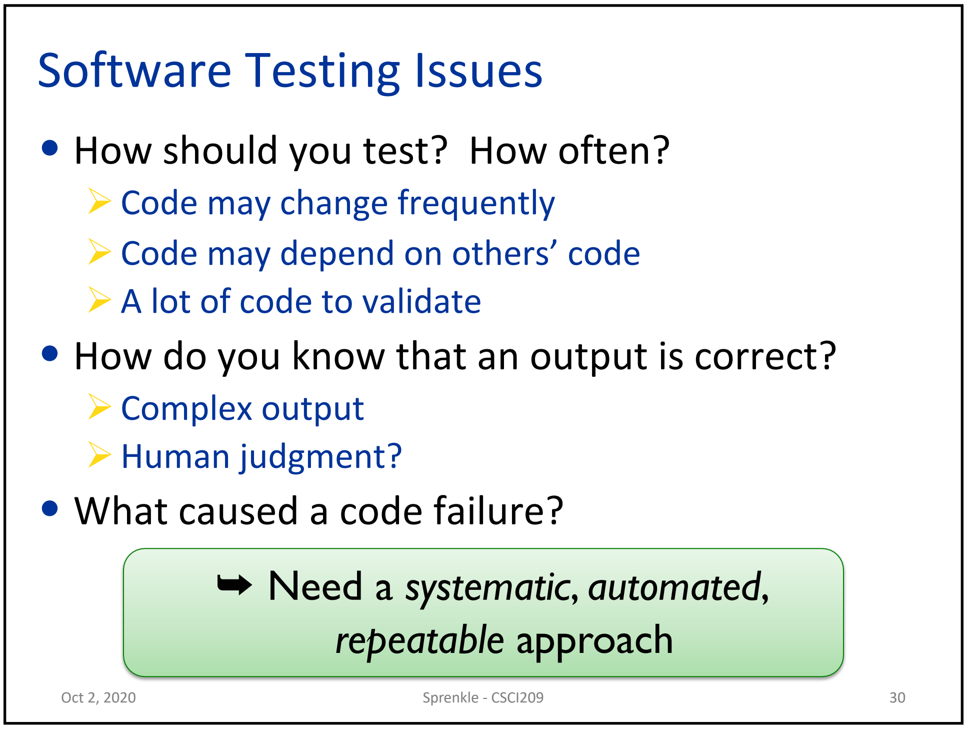
Numerical Answer

- What test cases: input and expected output?

Example Test Cases for Calculator Program

- Basic Functionality
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Order of operations
- Invalid Input
  - Letters, not-operation characters (&,$, …)

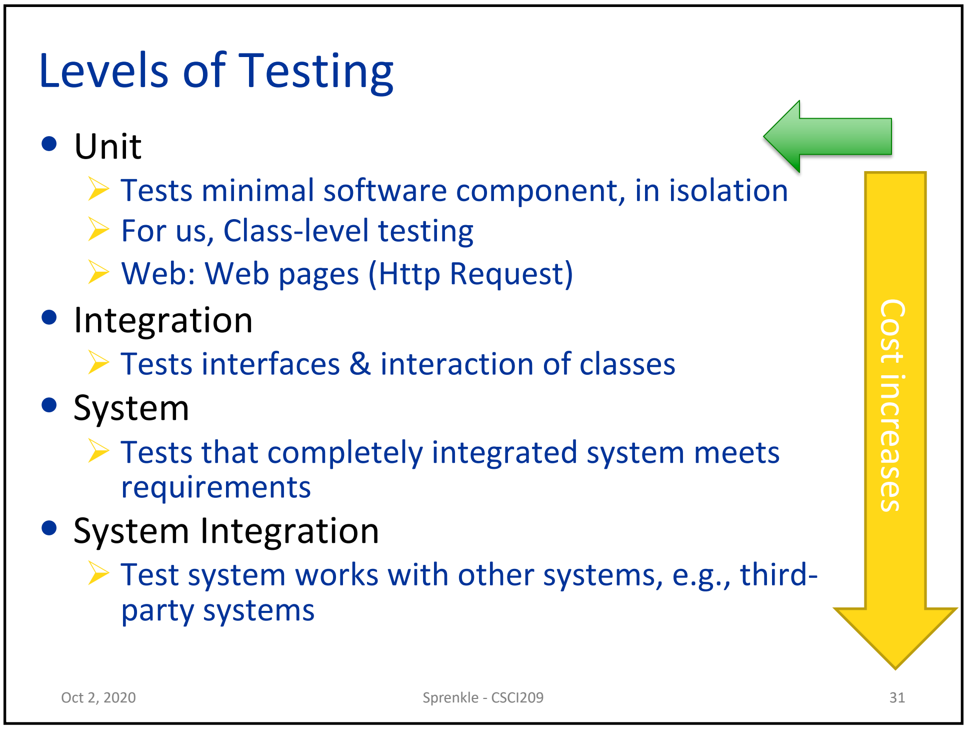
- “Tricky” Cases
  - Divide by 0
  - Negative Numbers
  - Long sequences of operands, operators
  - VERY large, VERY small numbers



Software Testing Issues

- How should you test? How often?
  - Code may change frequently
  - Code may depend on others' code
  - A lot of code to validate
- How do you know that an output is correct?
  - Complex output
  - Human judgment?
- What caused a code failure?

➥ Need a *systematic, automated, repeatable* approach



Levels of Testing

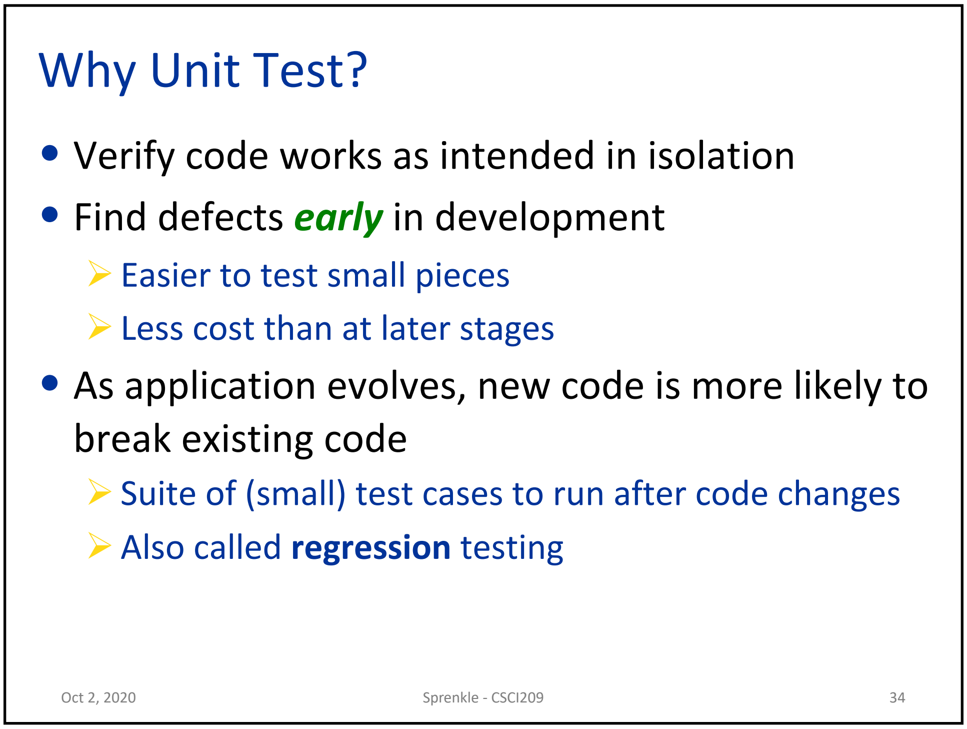
- Unit
  - Tests minimal software component, in isolation
  - For us, Class-level testing
  - Web: Web pages (Http Request)
- Integration
  - Tests interfaces & interaction of classes
- System
  - Tests that completely integrated system meets requirements
- System Integration
  - Test system works with other systems, e.g., third-party systems

Cost increases

UNIT TESTING

Why Unit Test?

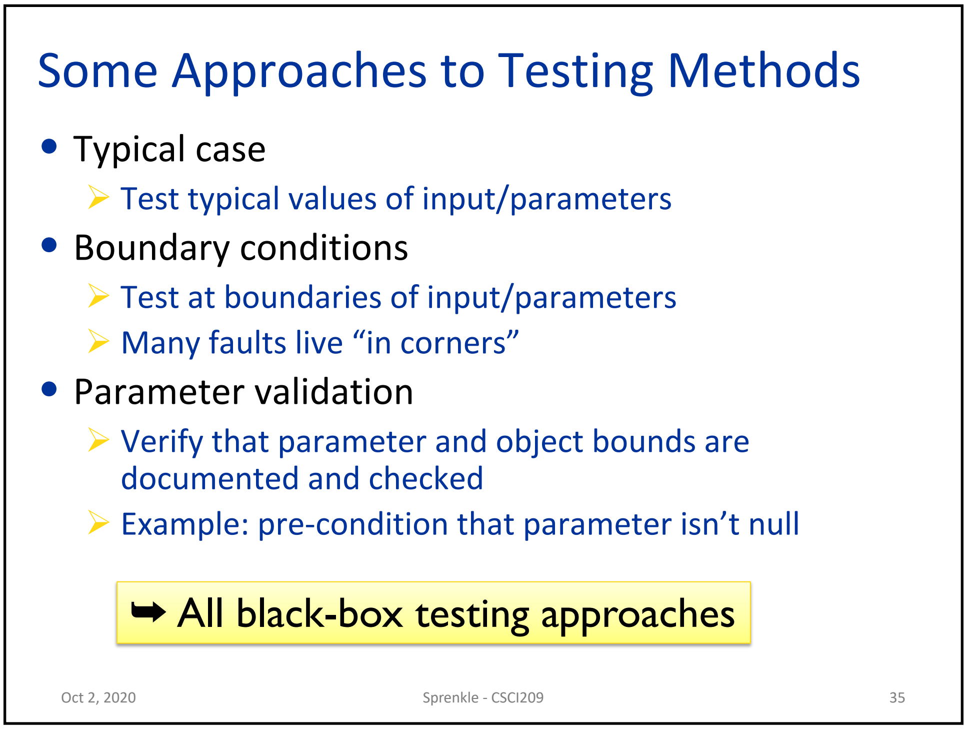
- Verify code works as intended in isolation
- Find defects ***early*** in development
  - Easier to test small pieces
  - Less cost than at later stages



Why Unit Test?

- Verify code works as intended in isolation
- Find defects ***early*** in development
  - Easier to test small pieces
  - Less cost than at later stages
- As application evolves, new code is more likely to break existing code
  - Suite of (small) test cases to run after code changes
  - Also called **regression** testing

34

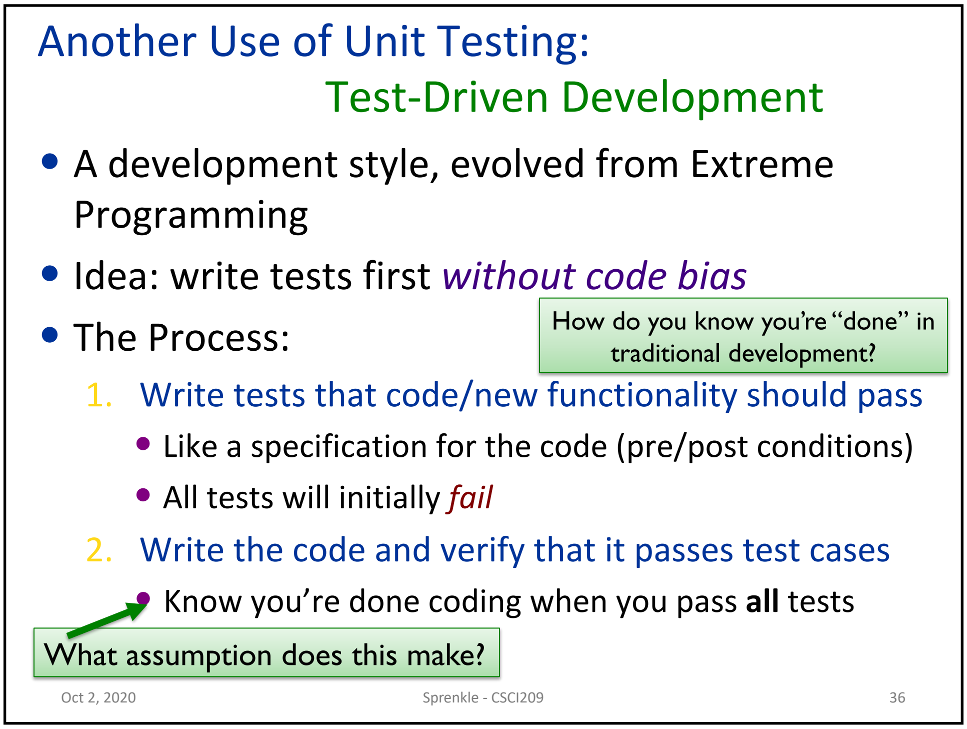


Some Approaches to Testing Methods

- Typical case
  - Test typical values of input/parameters
- Boundary conditions
  - Test at boundaries of input/parameters
  - Many faults live “in corners”
- Parameter validation
  - Verify that parameter and object bounds are documented and checked
  - Example: pre-condition that parameter isn’t null

➥ All black-box testing approaches

35



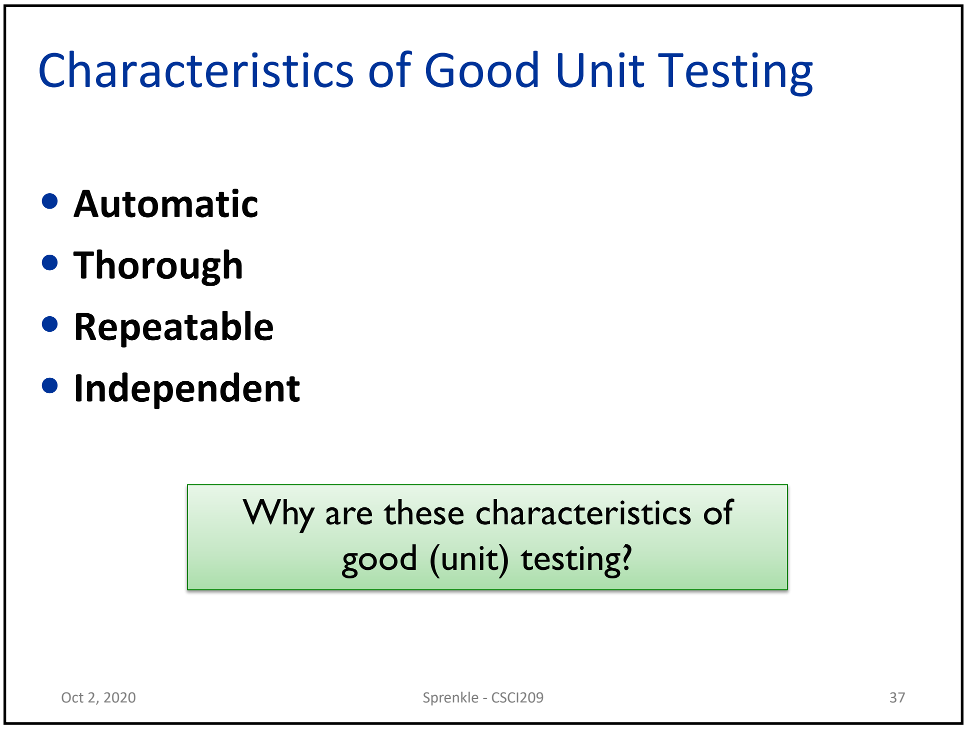
Another Use of Unit Testing: Test-Driven Development

- A development style, evolved from Extreme Programming
- Idea: write tests first *without code bias*
- The Process:
  1. Write tests that code/new functionality should pass
     - Like a specification for the code (pre/post conditions)
     - All tests will initially *fail*
  2. Write the code and verify that it passes test cases
     - Know you're done coding when you pass **all** tests

How do you know you're "done" in traditional development?

What assumption does this make?

36

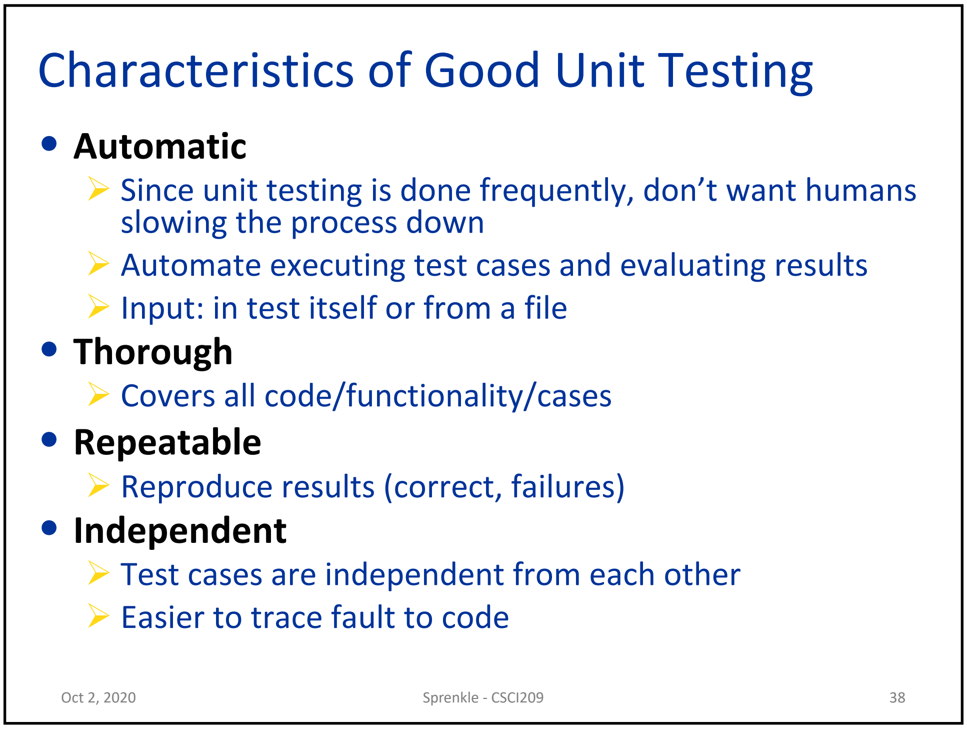


Characteristics of Good Unit Testing

- **Automatic**
- **Thorough**
- **Repeatable**
- **Independent**

Why are these characteristics of good (unit) testing?

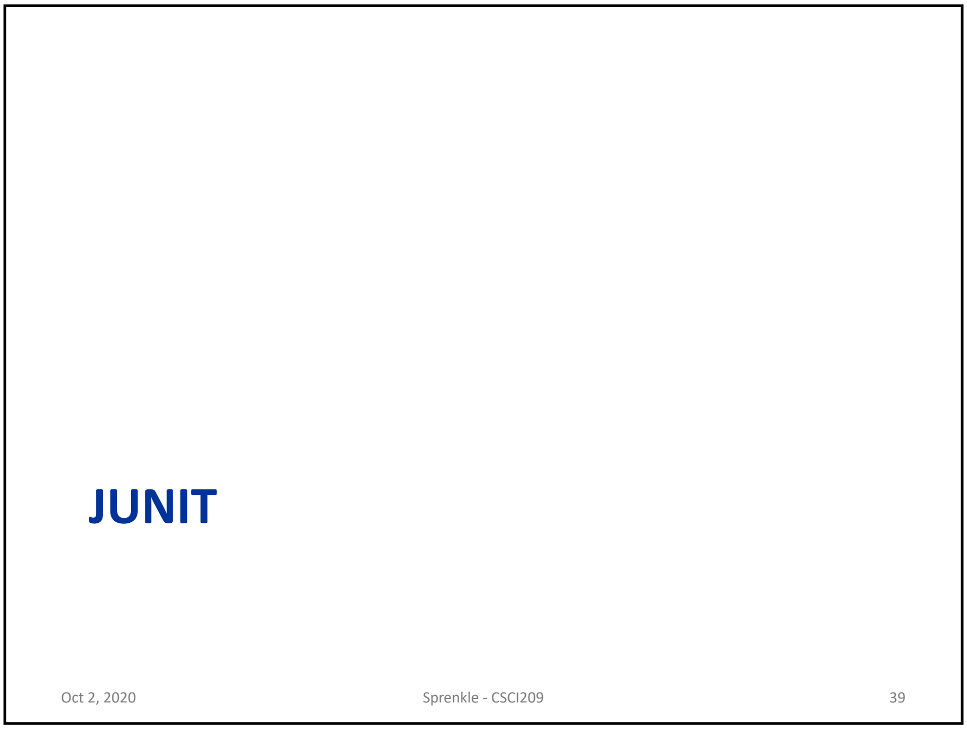
37



Characteristics of Good Unit Testing

- **Automatic**
  - Since unit testing is done frequently, don’t want humans slowing the process down
  - Automate executing test cases and evaluating results
  - Input: in test itself or from a file
- **Thorough**
  - Covers all code/functionality/cases
- **Repeatable**
  - Reproduce results (correct, failures)
- **Independent**
  - Test cases are independent from each other
  - Easier to trace fault to code

38



JUNIT

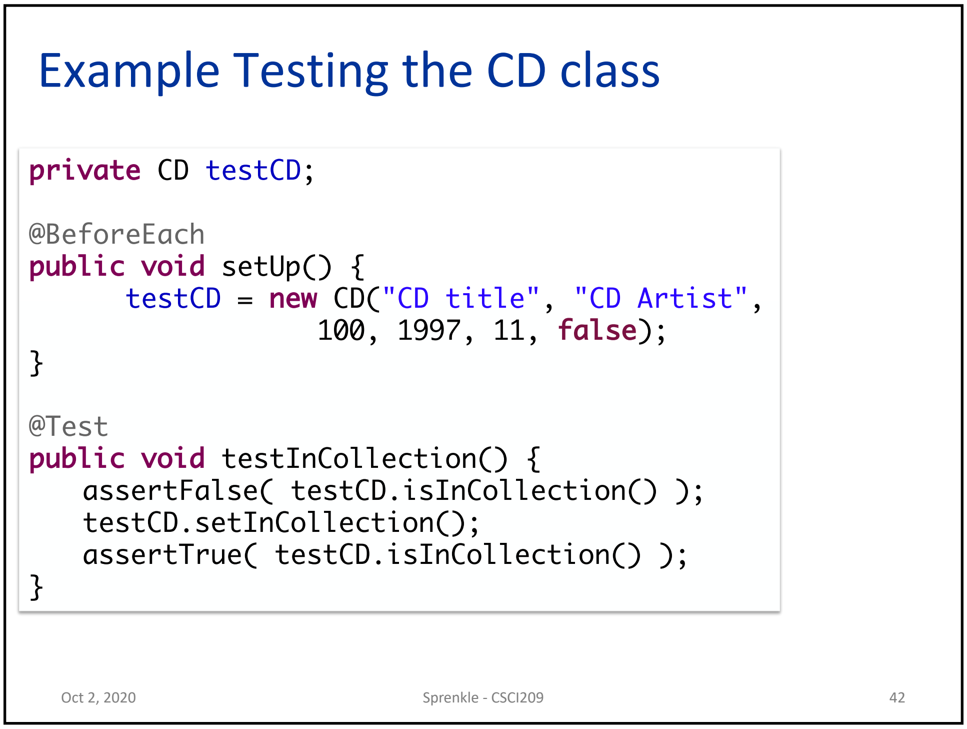
39

JUnit Framework

- A framework for unit testing Java programs
  - ➢ Supported by Eclipse and other IDEs
  - ➢ Developed by Erich Gamma and Kent Beck
- Functionality
  - ➢ Write tests
    - Validate output, automatically
  - ➢ Automate execution of test suites
  - ➢ Display pass/fail results of test execution
    - Stack trace where fails
  - ➢ Organize tests, separate from code
- But, you still need to come up with the tests!

Structure of a JUnit Test

1. Set up the test case (optional)
   - ➢ Example: Creating objects
2. Exercise the code under test
3. Verify the correctness of the results
4. Teardown (optional)
   - ➢ Example: reclaim created objects

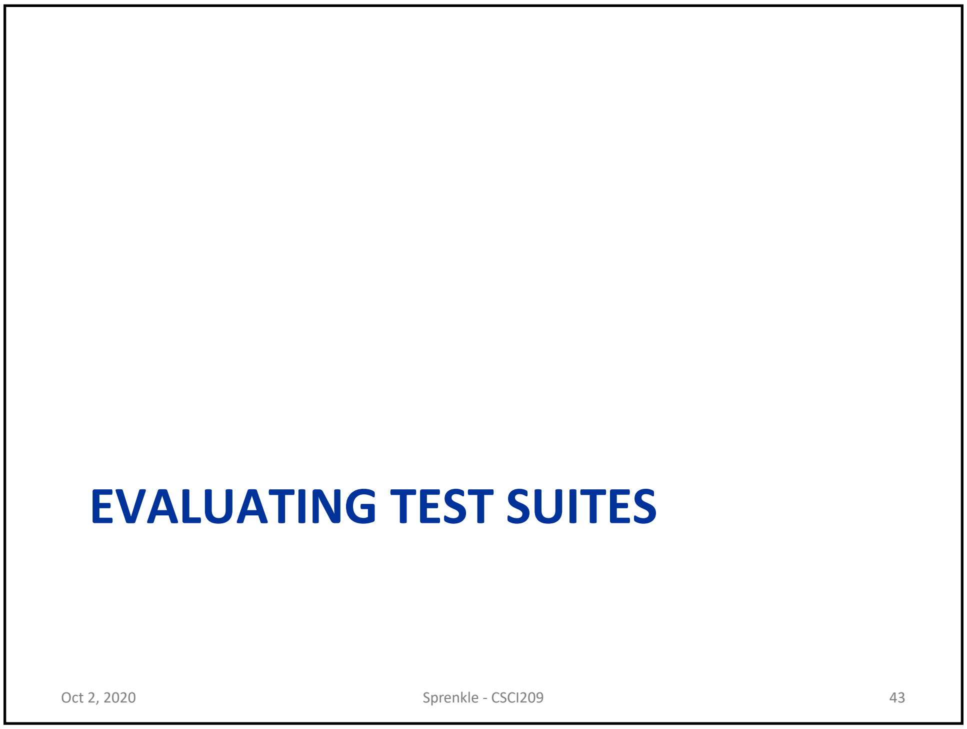


Example Testing the CD class

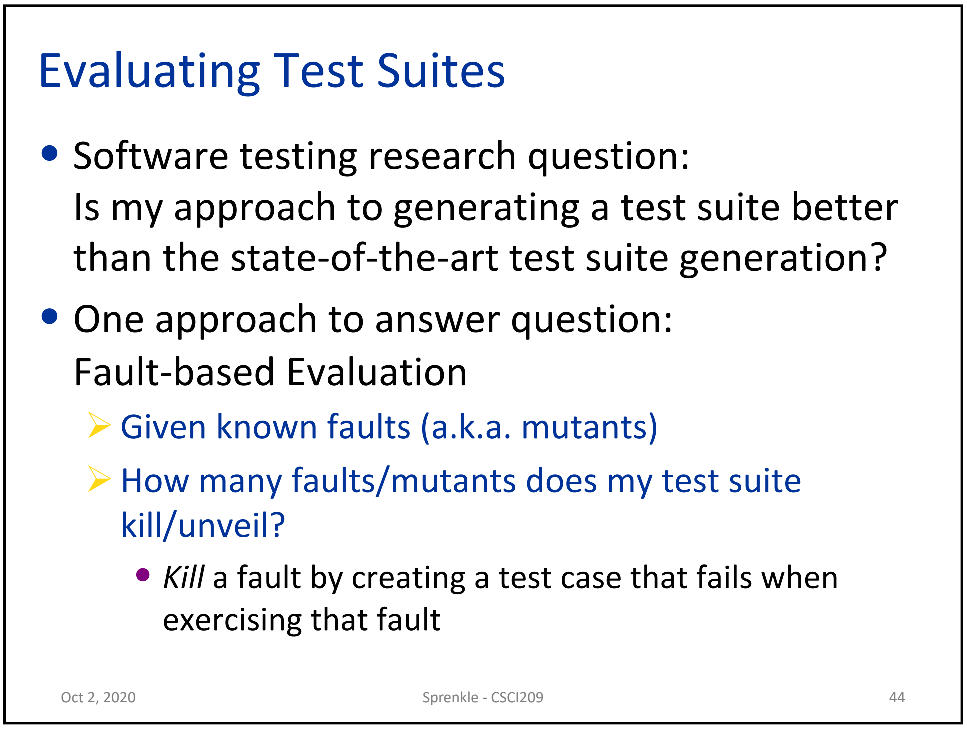
```
private CD testCD;

@BeforeEach
public void setUp() {
      testCD = new CD("CD title", "CD Artist",
                  100, 1997, 11, false);
}

@Test
public void testInCollection() {
   assertFalse( testCD.isInCollection() );
   testCD.setInCollection();
   assertTrue( testCD.isInCollection() );
}
```



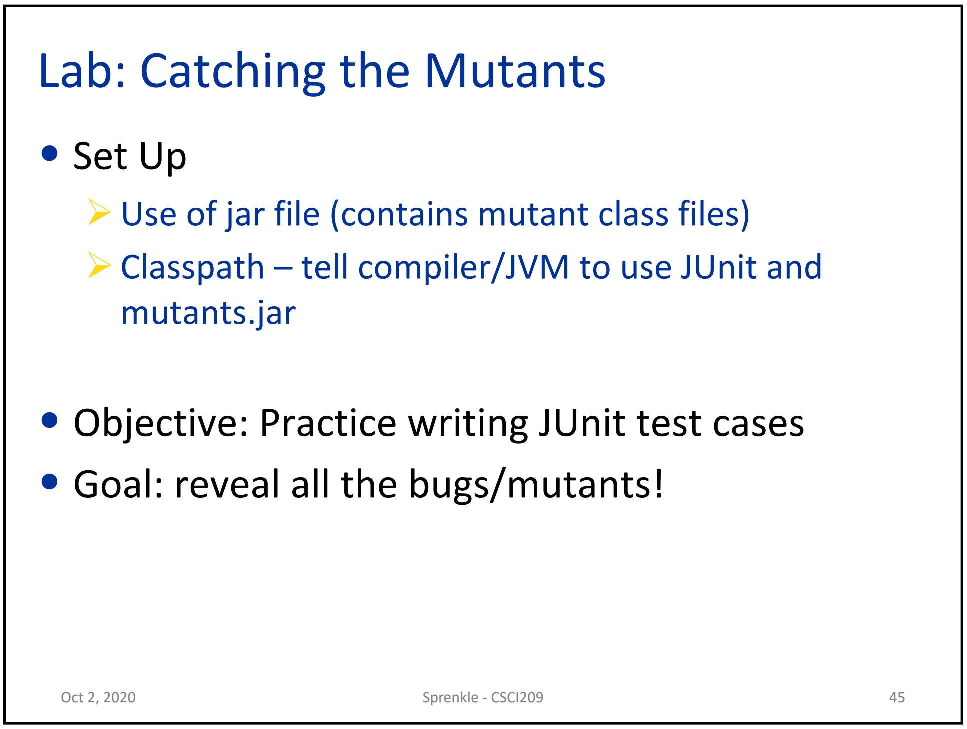
EVALUATING TEST SUITES



Evaluating Test Suites

- Software testing research question:
  Is my approach to generating a test suite better than the state-of-the-art test suite generation?
- One approach to answer question:
  Fault-based Evaluation
  - Given known faults (a.k.a. mutants)
  - How many faults/mutants does my test suite kill/unveil?
    - *Kill* a fault by creating a test case that fails when exercising that fault

44



Lab: Catching the Mutants

- Set Up
  - Use of jar file (contains mutant class files)
  - Classpath – tell compiler/JVM to use JUnit and mutants.jar
- Objective: Practice writing JUnit test cases
- Goal: reveal all the bugs/mutants!

45

Catching the Mutants: Post-Mortem

- What are the benefits of unit testing/using JUnit?
  - Consider if you were developing/maintaining the method
  - How would your testing/development process change?
- Why did the output come out in strange orders sometimes?
- Is it okay that some mutants passed some of the test cases?
- Recall the characteristics of good unit tests
  - How did you achieve them in your testing?

Project 1: Test-Driven Development

- Given: a Car class that only has enough code to compile
- Your job: Create a ***good*** set of test cases that ***thoroughly***/***effectively*** test Car class
  - Find faults in my faulty version of Car class
  - Start: look at code, think about how to test, set up JUnit tests
  - Written analysis of process
- First team project: teams of **3**
  - Practice collaboration (more on Monday)
  - Every student must commit code to the repository
- Due before class Monday, Oct 12
  - First step: create teams (and *team names!*) by Monday’s class