

## Objectives

- Unit testing
- JUnit Framework
  - In Eclipse

Oct 17, 2008

Sprenkle - CS209

1

## Review

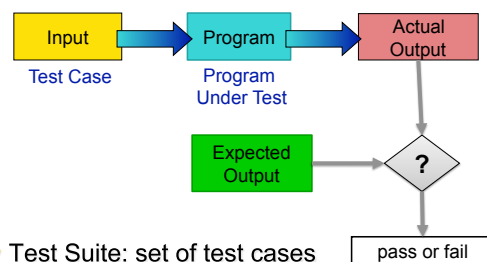
- What is the operator that allows us to use the output from one Unix command as the input to another Unix command?
- How do we use a Unix command in another Unix command?
- Describe and compare the two software development models we discussed
- Describe the general testing process
- What is a set of test cases called?

Oct 17, 2008

Sprenkle - CS209

2

## Review: Software Testing Process



- Test Suite: set of test cases

Oct 17, 2008

Sprenkle - CS209

3

## Review: Levels of Testing

- Unit
  - Tests minimal software component, in isolation
  - For us, Class-level testing
  - Web: Web pages
- 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

Oct 15, 2008

Sprenkle - CS209

4

## 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

Oct 17, 2008

Sprenkle - CS209

5

## Levels of Testing

- Unit
  - Tests minimal software component, in isolation
  - For us, Class-level testing
  - Web: Web pages
- 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

Oct 15, 2008

Sprenkle - CS209

6

## 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**

Oct 17, 2008

Sprenkle - CS209

7

## Review: Example Test Cases for Calculator Program

- Basic Functionality
  - Addition
  - Subtraction
  - Multiplication
  - Division
  - Order of operations
- "Tricky" Cases
  - Divide by 0
  - Negative Numbers
  - Long sequences of operands, operators
  - VERY large, VERY small numbers
- Invalid Input
  - Letters, not-operation characters (&,\$, ...)

Generalize types of test cases

Oct 15, 2008

Sprenkle - CS209

8

## Some Approaches to Testing Methods

- Typical case
  - Test typical values of parameters
- Boundary conditions
  - Test at boundaries of parameters
  - Many bugs 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  
We'll discuss others later...

Oct 17, 2008

Sprenkle - CS209

9

## Another Use of Unit Testing: Test-Driven Development

- A development style
  - Evolved from Extreme Programming
- Idea: write tests first, without code bias
- How it works:
  - Write the tests that the code/new functionality should pass
    - Like a specification for the code (pre/post conditions)
    - All tests will initially fail
  - Write the new code and make sure that it passes all test cases

Oct 17, 2008

Sprenkle - CS209

10

## 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

Oct 17, 2008

Sprenkle - CS209

11

## Good Unit Testing

- Automatic
  - Since unit testing is done frequently, don't want humans slowing the process down
  - Running test cases
  - Evaluating results
  - Input: in test itself or from a file
- Thorough
- Repeatable
  - Reproduce results (correct, failures)
- Independent
  - Test cases are independent from each other
  - Easier to trace fault to code

Oct 17, 2008

Sprenkle - CS209

12

## 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!

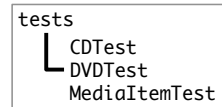
Oct 17, 2008

Sprenkle - CS209

13

## Testing with JUnit

- Typical organization:
  - Set of testing classes
  - Testing classes packaged together in a **tests** package
    - Separate package from code testing
- A test class typically
  - Focuses on a specific class
  - Contains methods, each of which represents another test of the class



Oct 17, 2008

Sprenkle - CS209

14

## Structure of a JUnit Test

- Set up the test case (optional)
  - Example: Creating objects
- Exercise the code under test
- Verify the correctness of the results
- Teardown (optional)
  - Example: reclaim created objects

Oct 17, 2008

Sprenkle - CS209

15

## Annotations

- Testing in JUnit 4: uses **annotations**
- Provide data about a program that is not part of program itself
- Have no direct effect on operation of the code
- Example uses:
  - **@Override**: method declaration is intended to override a method declaration in parent class
    - If method does not override parent class method, compilers generates error message
  - Information for the compiler to suppress warnings (**@SuppressWarnings**)

Oct 17, 2008

Sprenkle - CS209

16

## Tests are Methods

- Mark your testing method with **@Test**
  - From `org.junit.Test`

```

public class CalculatorTest {
    @Test
    public void add() {
        ...
    }
}
  
```

Class for testing the Calculator class

A method to test the "add" functionality

- Convention: Method name describes what you're testing

Oct 17, 2008

Sprenkle - CS209

17

## Assert Methods

- Variety of assert methods available
- If fail, throw an exception
- All **static void**
- Example:
 

```
assertEquals(Object expected, Object actual)
```

```

@Test
public void add() {
    ...
    assertEquals(4, calculator.add(3, 1));
}
  
```

Oct 17, 2008

Sprenkle - CS209

18

## Assert Methods

- To use asserts, need *static* import:
 

```
import static org.junit.Assert.*;
```

  - Static allows us to not have to use classname
- More examples
  - `assertTrue(boolean condition)`
  - `assertSame(Object expected, Object actual)`
    - Refer to same object

```
@Test
public void testEmptyCollection() {
    Collection collection = new ArrayList();
    assertTrue(collection.isEmpty());
}
```

Oct 17, 2008

Sprenkle - CS209

19

## Set Up/Tear Down

- May want methods to set up objects for every test in the class
  - Called **fixtures**
  - If have multiple, no guarantees for order executed

```
@Before
public void prepareTestData() { ... }

@Before
public void setupMocks() { ... }

@After
public void cleanupTestData() { ... }
```

Executed before each test method

Oct 17, 2008

Sprenkle - CS209

20

## Using JUnit in Eclipse

- Eclipse can help make our job easier
  - Automatically execute tests (i.e., methods)
  - We can focus on coming up with tests

Oct 17, 2008

Sprenkle - CS209

21

## Using JUnit in Eclipse

- In Eclipse, go to your MediaItems project
- Create a new JUnit Test Case
  - Use JUnit 4
    - Add junit to build path
  - Put in package `media.tests`
  - Name: `DVDTest`
  - Choose to test `DVD` class
    - Select `setUp` and `tearDown`
    - Select methods to test
- Run the class as a JUnit Test Case

Oct 17, 2008

Sprenkle - CS209

22

## Example

- Testing `getPlayingTime`
  - Revise: Add code to `setUp` method that creates a DVD
- Notes
  - FastView vs Detached
- Hint: CTL-Spacebar to get auto-complete options

Oct 17, 2008

Sprenkle - CS209

23

## Set Up/Tear Down For Class

- May want methods to set up objects for set of tests
  - Executed once for all tests in class

```
@BeforeClass
public static void
setupDatabaseConnection() { ... }

@AfterClass
public static void
tearDownDatabaseConnection() { ... }
```

Oct 17, 2008

Sprenkle - CS209

24

## Unit Testing & JUnit Summary

- Unit Testing: testing smallest component of your code
  - For us: class and its methods
- JUnit provides framework to write test cases and run test cases automatically
  - Easy to run again after code changes
- JUnit Resources available from Course Page's "Resource" Link, under Java
  - API
  - Tutorials

Oct 17, 2008

Sprenkle - CS209

25

## Project 1: Testing Practice

- Due next Friday
- 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 buggy version of Car class
  - Start: look at code, think about how to test, set up JUnit tests
- We're going to continue talking about software testing issues, e.g., what makes a test suite good

Oct 17, 2008

Sprenkle - CS209

26