CMSC 131

Object-Oriented Programming I

Testing, JUnit

Dept of Computer Science
University of Maryland College Park

This material is based on material provided by Ben Bederson, Bonnie Dorr, Fawzi Emad, David Mount, Jan Plane

Overview

- Testing
- JUnit

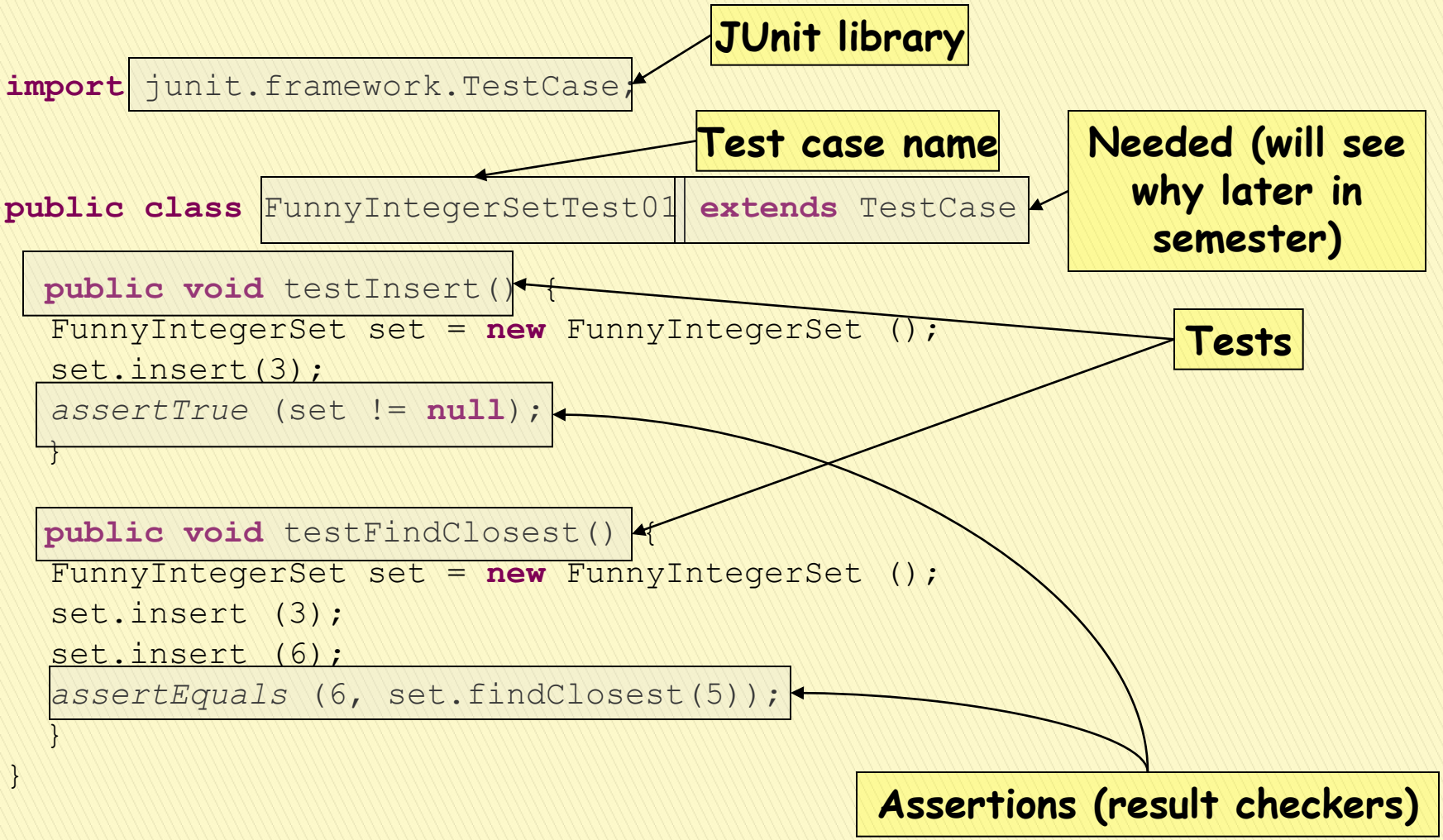
Testing: The Problem

- Problems:
  - Need to be able to make sure all parts are tested
  - Need to know in testing exactly which part was not as expected
  - Need to be able to keep the tests for modifications made later
- Unit testing helps overcome this problems of making sure everything is tested
  - Unit testing: test each class and each part of the class (unit) individually
  - Goal is to eliminate inconsistencies between the API and the actual working of the code

JUnit

- A unit-testing tool for Java
- Includes capabilities for:
  - Test definition, including output checking
  - Test running (execution)
  - Result reporting
- Seamless integration with Eclipse

Structure of a JUnit Test Case



JUnit Testing

- A class is used to write tests
  - PublicTests
  - ReleaseTests
  - SecretTests
- Methods in a class represent the actual tests
  - **Each test method must be named starting with "test" otherwise test will not work**
- The Junit test library needs to be added if not available
  - In Eclipse when creating a test the system will ask you to add the library if not available
- Assertion Checkers
  - **assertTrue(***expression***);**
    - If statement is true, keep running test; otherwise, halt test, report "fail"
  - **assertFalse(***expression***);**
    - If statement is false, keep running test; otherwise, halt test, report "fail"
  - **assertEquals(***expression1***, *expression2*);**
    - If expression1, expression2 equal, keep running test; otherwise, halt test, report "fail"

Junit Testing

- If test terminates without failing an assertion and without throwing an uncaught exception, then it passes that test
- You can have multiple assertions in a test
- It continues with all subsequent tests regardless of passing or failing the current test
- Notice that even if a test (method) does not compile, the other methods will be run
- B y using JUnit tests you no longer to define a Driver class to test your code.
- Notice that you can have output (e.g., System.out.println) statements in your tests.
- Instructions on how to create tests is available at:

http://www.cs.umd.edu/eclipse/EclipseTutorial/JUnitTesting.pdf

- Example: Let's define a class that computes maximum between two values and then define tests for it.

Hints on Testing

- Give names to tests that relate to class being tested
- Develop some tests before you code
  - Helps you to clarify what you are supposed to be doing
  - Gives you some ready-made tests to run while you code
- Use tests to debug
- How many tests?
  - Statement coverage: develop tests to make sure each statement in class is executed at least once (including constructors)
  - Decision coverage: develop tests to make each condition (if statement) in program both true and false
  - You should at least reach statement coverage in your own testing
- Beware of corner cases
- When software is not working come up with the simplest test that demonstrates the failure

Ternary Operator

- We can rewrite your maximum method by using the ternary operator:

Expr ? exprValueIfExprIsTrue :exprValueIfExp1sFalse;

- Rewriting maximum

int maximum = x > y ? x : y;

Quick Introduction to Interfaces

- Let's briefly introduce interfaces
- We will have more to say about them later on
- Pet interface
- Dog class implementing the interface