

Lecture 17: this and unit testing

Last time:

1. Method overloading

Today:

1. `this`
2. Unit testing and JUnit



this

- **this**: keyword for referring to current object

- Example

- Recall definition of setName method in Student class:

```
public void setName (String newName) {  
    name = newName;  
}
```

- This is equivalent to:

```
public void setName (String newName) {  
    this.name = newName;  
}
```

Why this?

- One use: method-parameter names that match instance-variable names
 - Re-implementation of setName:

```
public void setName (String name) {  
    this.name = name;  
}
```
 - Without this, parameter would need different name (why?)
- A more interesting case is one where it is ***necessary*** to use this ... (see next example).

Why this?

```
public class Computer {
    private long clockSpeed;
    private String manufacturer;

    public Computer(String manufacturer, long clockSpeed) {
        this.manufacturer = manufacturer;
        this.clockSpeed = clockSpeed;
    }
    ...
    public String performPsychEvaluation(Student s) {
        if (s.getName().equals("Bonnie")) {
            return "totally crazy";
        } else {
            return "not as crazy as Bonnie";
        }
    }
}
```

Now add two lines to the “doAProject” student method:

```
Computer c = new Computer("Mac", 273847239847L);
c.performPsychEvaluation(this);
```

Note that “this” is the only way to refer to the current object above!!

Unit Testing

- So far projects have consisted of a single class and a driver
- Java programs typically contain many classes
- Locating errors can be tricky if multiple classes involved
- **Unit testing** helps overcome this problem
 - Unit testing: test each class (unit) individually
 - Goal is to eliminate errors within classes

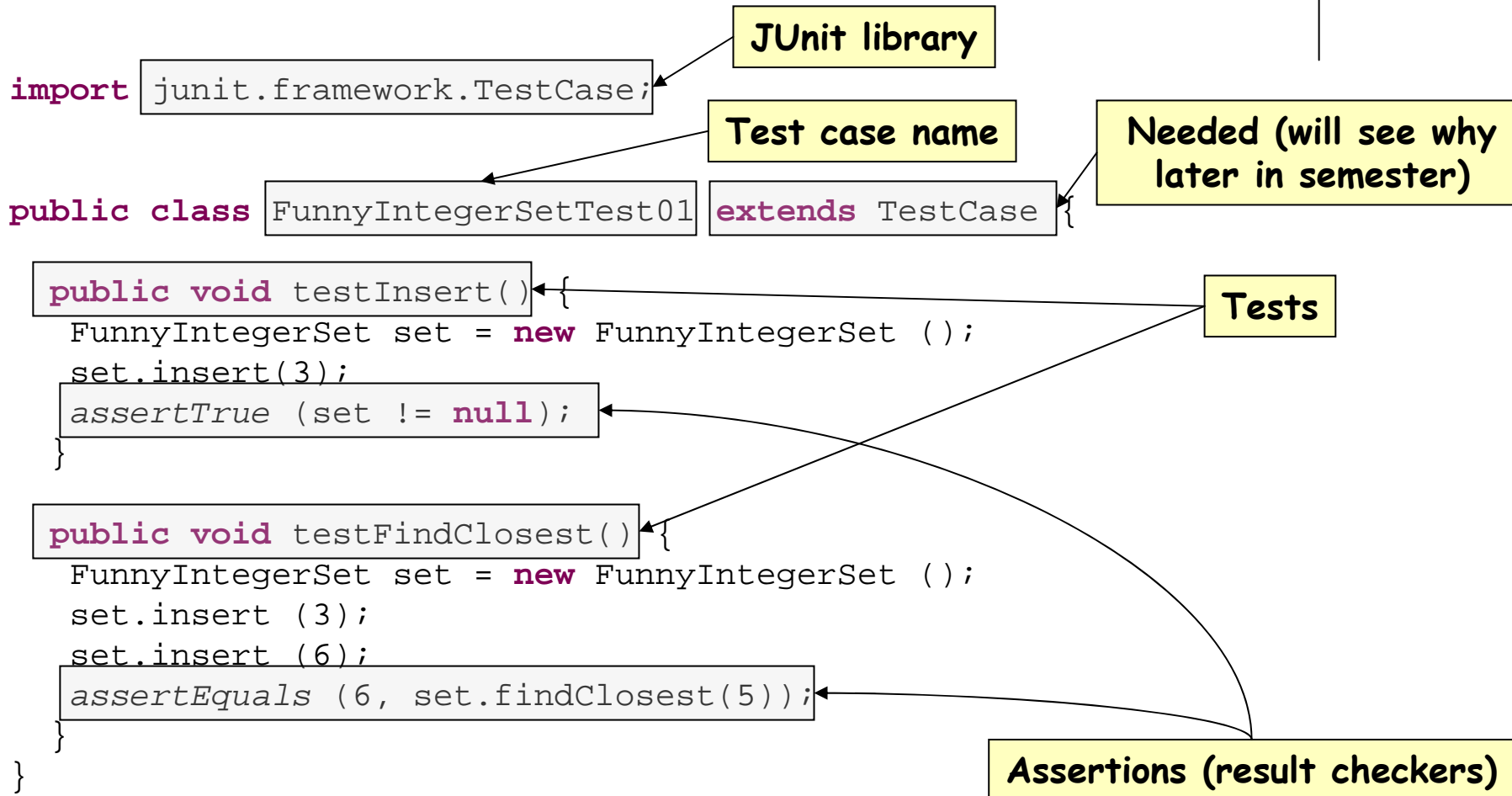
How To Do Unit Testing?

- Needs for unit testing
 - Method for defining tests = inputs, expected outputs
 - Method for running tests
 - Method for reporting results
- One possibility: write a driver for each class
 - Driver class contains main method
 - main method creates objects in class to be tested, calls methods, prints outputs
 - User checks outputs, determines correctness
 - Good: easy, no special tools needed
 - Bad: tedious, relies on human inspection of outputs
- Another approach: **JUnit**

JUnit

- A unit-testing tool for Java
- Includes capabilities for:
 - Test definition, including output checking
 - Test running (execution)
 - Result reporting
- Seamless integration with Eclipse
- **Note**
 - **In this class we will use JUnit 3.8.1**
 - A newer version, JUnit 4.0, has recently been released, but not yet evaluated by the CS dept.
 - JUnit 4.0 includes more features than JUnit 3.8.1, but basic principles are the same

Structure of a JUnit 3.8.1 Test Case



A Test Case Is ... A Class!

- “Inherits from” (**extends**) class `TestCase`, which is in library `junit.framework.TestCase` (we will learn about inheritance later)
- Contains methods like any class
- Some method names begin with the word `test`
- Method bodies contain calls to **assertion checkers**
 - E.g.
 - `assertTrue`
 - `assertFalse`
 - `assertEquals`
 - Assertion checkers are also defined in the above library

A Test Case Is ... A Collection of Tests



- In example, a class `FunnyIntegerSet` is being tested by the test case
 - Terminology: SUT = “software under test”
- Test case methods beginning with `test` are viewed as individual tests that the SUT either passes or fails
- Passing / failing determined by `assert` calls
 - `assertTrue(b)`
If `b` is true, keep running test; otherwise, halt test, report “fail”
 - `assertFalse(b)`
If `b` is false, keep running test; otherwise, halt test, report “fail”
 - `assertEquals (expected, actual)`
If `expected`, `actual` equal, keep running test; otherwise, halt test, report “fail”
- If test terminates without failing, it passes
- Test may create objects, call other methods, etc., just like any other method

Example Revisited

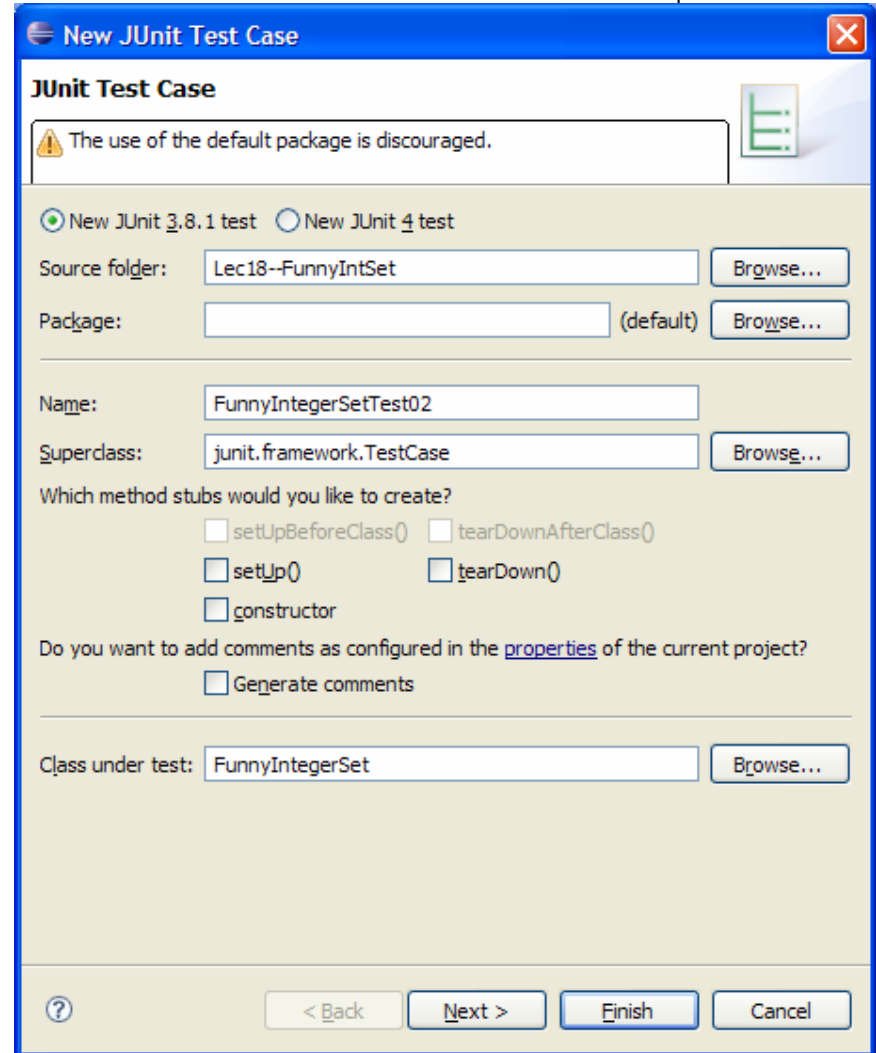
- FunnyIntegerSet is supposed to implement two operations on sets of integers
 - `public void insert (int i);`
add `i` into set
 - `public int findClosest (int i);`
return element of set closest to `i`
- What to test?
 - Does insert “really work”?
 - Does inserting yield a non-empty set?
 - etc.
 - Does findClosest really work?
 - Does it return the closest value in a two-member set?
 - Does it return the only member of the set in a one-member set?
 - etc.
- `FunnyIntegerSetTest01` tests two of these properties

Running Tests in JUnit

- JUnit test cases are Java classes
 - They are stored in .java files
 - They may be manipulated like any other file
- To run tests using JUnit, a **test runner** must be specified
 - Test runner executes each test in test case
 - Results about passing / failing reported
- Installation of JUnit typically identifies test runner to be used

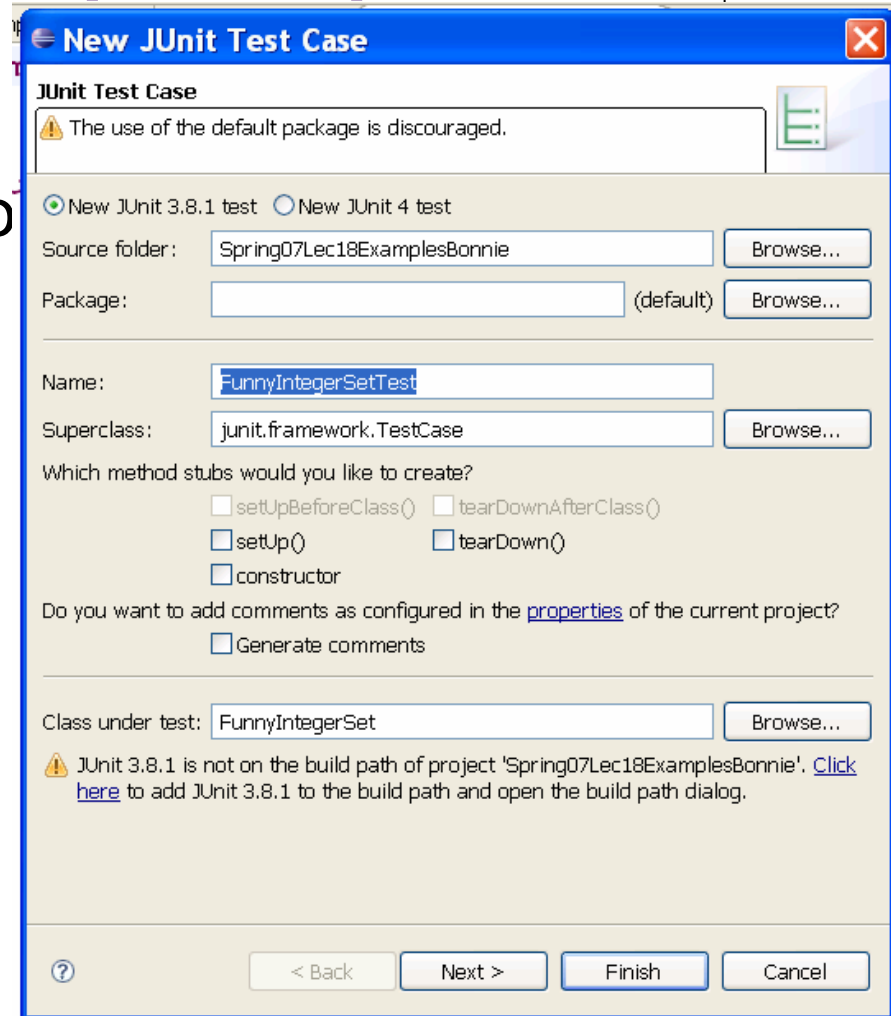
JUnit and Eclipse

- JUnit test cases may be created in Eclipse
 - Right click on the module (class) you are testing.
 - Select New → JUnit Test Case
 - You may be prompted to install JUnit if it is not already installed
 - Click Next and ...



JUnit and Eclipse (cont.)

- It puts in the name for you, but you may have to type in a new name if you are adding lots of JUnit tests.
- **Before clicking Finish,** click on “Click here”
- Then click Finish
- Resulting .java file is ...



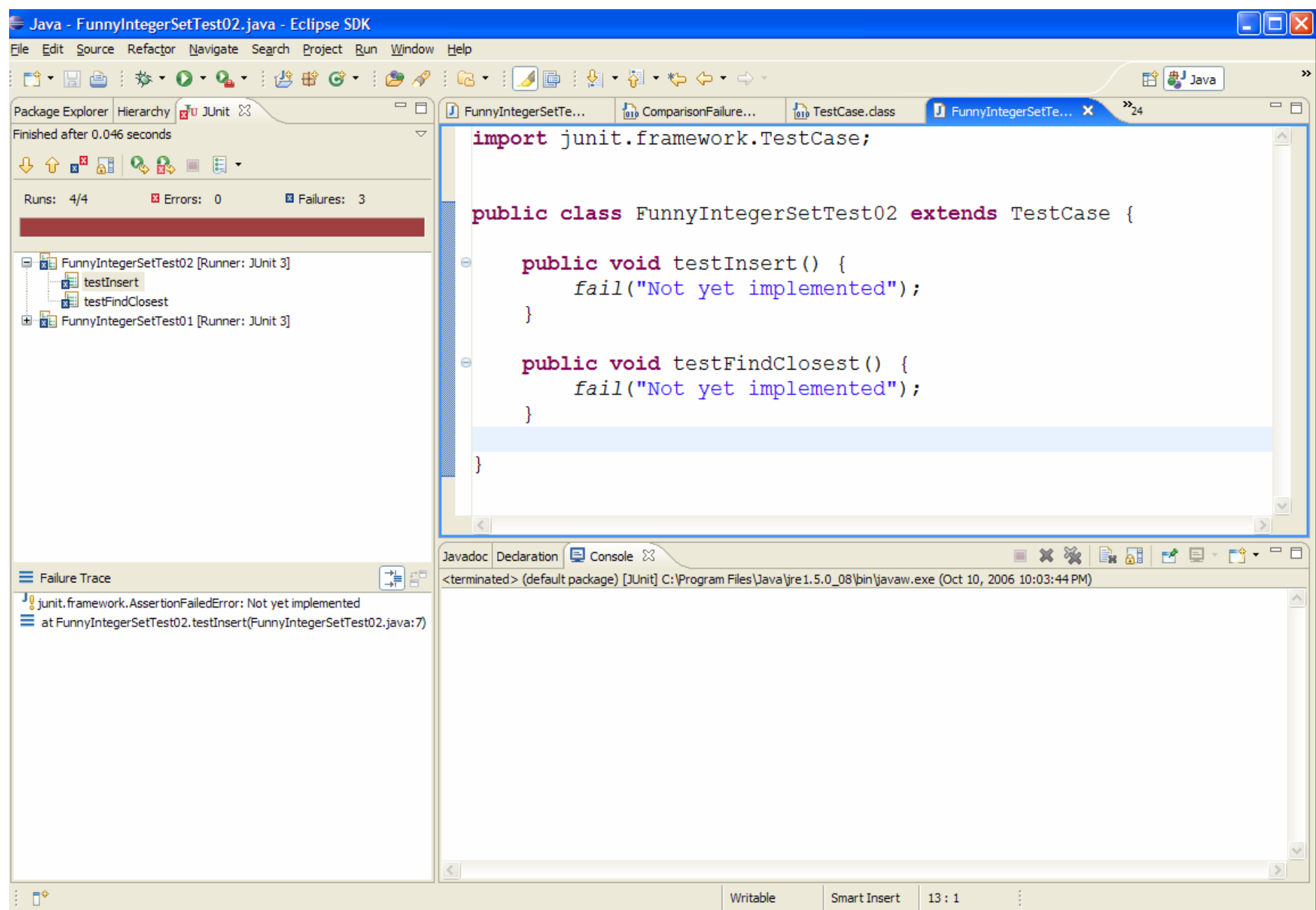
Java and Eclipse (cont.)

```
import junit.framework.TestCase;
```

```
public class FunnyIntegerSetTest extends  
    TestCase {  
  
}
```

Running Tests in Eclipse

- Right-click in Package Explorer
- Select Run As -> JUnit Test inside project
- New panel appears showing results of running all tests in same project



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