# CE0825a: Object Oriented Programming II
# 10: Project, Testing, Randomness

James A Sutherland

Abertay University

Monday, 14th March 2016

# Coursework Project

**Submission Deadline** End of Monday 18th April (week 13)

**Submission Format** **Two** files in Blackboard

- Standalone runnable JAR file containing binary and source
- Project report (Word or PDF)

**Demonstration** Week 13 timeslot, chosen during week 12 lab session

# Coursework Requirements

| | |
|---|---|
| Standalone | Include the SWT library (Eclipse does this for you!) |
| Runnable | Double-click (on Windows with x86 JRE) and it opens |
| JAR file | Hopefully you know what that is now! |
| Including source | The .java text files you edit and compile |

# Report

- What your code does
- How it does it
- Why it does that
- Screenshots
- Explanations

# Demonstration

- Run your application, on a 32 bit Windows PC . . . and impress me!
- Running on Mac, 64 bit Windows or Linux **as well** is good. **Only** running on one of those is not!
- Embedding the map images, or fetching from Driesh on demand, is fine.
- Remember, you don't have Eclipse there, just a headless JRE . . .

# Coursework Summary

- **Two** files in Blackboard by midnight Monday 18th April
- During week 12's lab, pick a timeslot in week 13 to demonstrate
- Demonstrate during that timeslot
- Collect 50% of your module grade, pass go

# Automated Testing

- No, not being applied to your coursework! (Yet...)
- Good practice for development

# Test Terminology

**Unit Test** Test of a single component

**Integration Test** Test of interaction between components

**Performance Test** Still responds OK under load?

**Regression Test** Check old bugs stay dead

**Test Coverage** How much of the total code gets tested

**Continuous Integration** Constant automated build & test

**Behaviour Test** Check for certain functions, arguments

**State Test** Check for certain outcomes: a file, message...

# What & When to Test?

- From the start (Test Driven Development)
- Legacy code: add to most problematic parts first
- When you make changes or fix bugs

# Static Imports

- Plain old "import" makes a class available by short name
  - `new java.io.File();`
  - `import java.io.File;`
  - `new File();`
- So "import static" does that for static methods:
  - `java.io.File.createTempFile();`
  - `import static java.io.File.createTempFile;`
  - `createTempFile();`

# Annotations

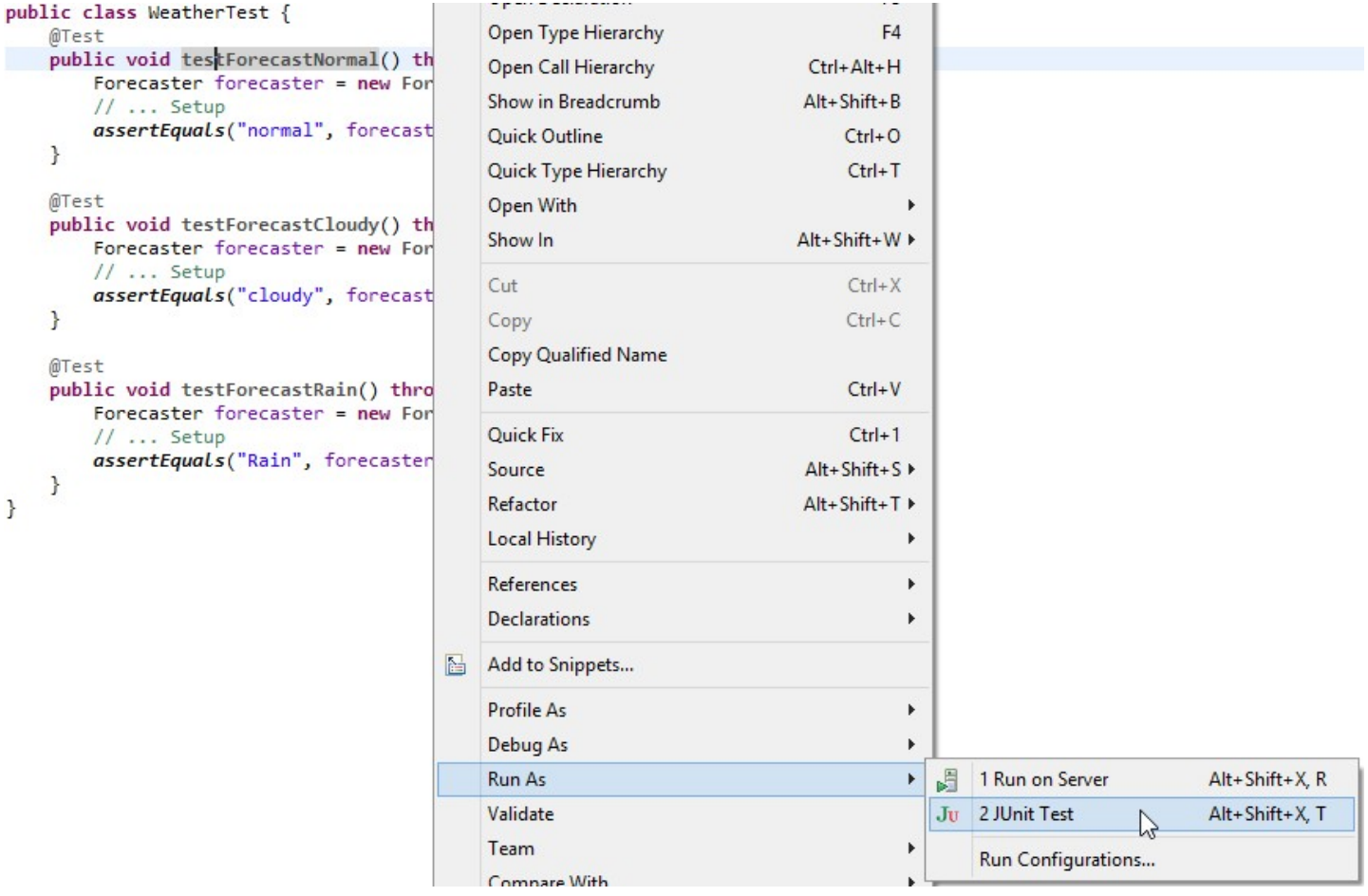
Annotations: little notes for various purposes, all starting with an @ symbol. For example, `@Deprecated` marks a function which shouldn't be used any more: each tiem you compile code that uses it, the compiler will warn you about that. Also used in JUnit, with @Test, which we're about to meet ...

# JUnit Introduction

JUnit: Testing framework

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class MyTests {
  @Test
  public void fooMustBeZero() {
    MyClass tester = new MyClass();
    assertEquals("foo returns 0", 0,
        tester.foo());
  }
}
```

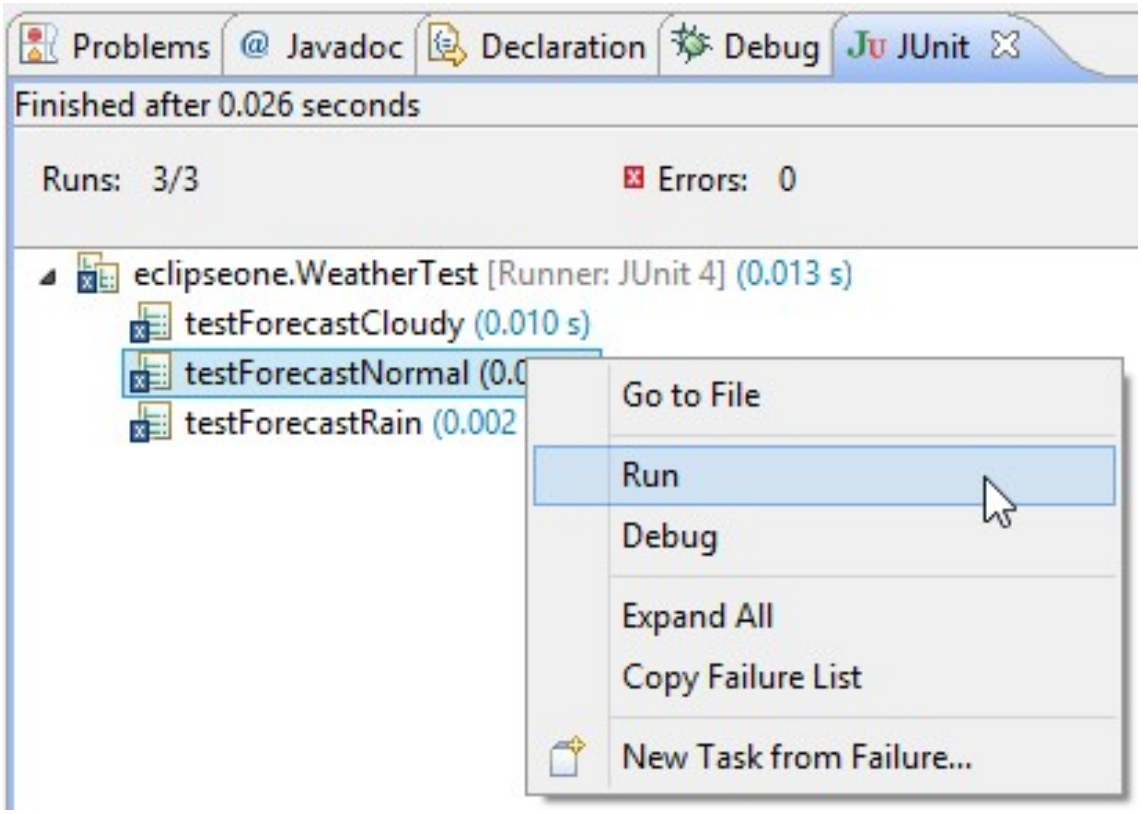
# Running Individual JUnit Tests



# JUnit View in Eclipse

- Eclipse has a JUnit View which lists all the tests in your project.
- Ctrl+F11 there will re-run *all* your tests
- Right-click, Run to re-run a particular test

# Running Tests From JUnit View



# Randomness & SecureRandom

- Random numbers are surprisingly important for games, encryption and other things. Unfortunately, also surprisingly difficult for computers!
- One of the 'Snowden revelations' was that the NSA had put a back door in one of the new standard RNGs. (Then, that another government had copied it into the US government's own firewalls...)
- `SecureRandom` is intended to be a cryptographically strong random number generator.

```
import java.security.SecureRandom;
SecureRandom sr=new SecureRandom();
int x=sr.next(8); // new 8 bit value; or
    .nextBytes(array)
```

# Lab Task Week 10

1. Write your own test code for some earlier lab work
2. Add a test that gets a random number and tests it's 42