Unit Testing

Jin L.C. Guo

Objectives

- Concepts and Principles:

Unit testing, regression testing, test suites, test coverage;

- Programming Mechanisms:

Unit testing frameworks, JUnit, metaprogramming, annotations;

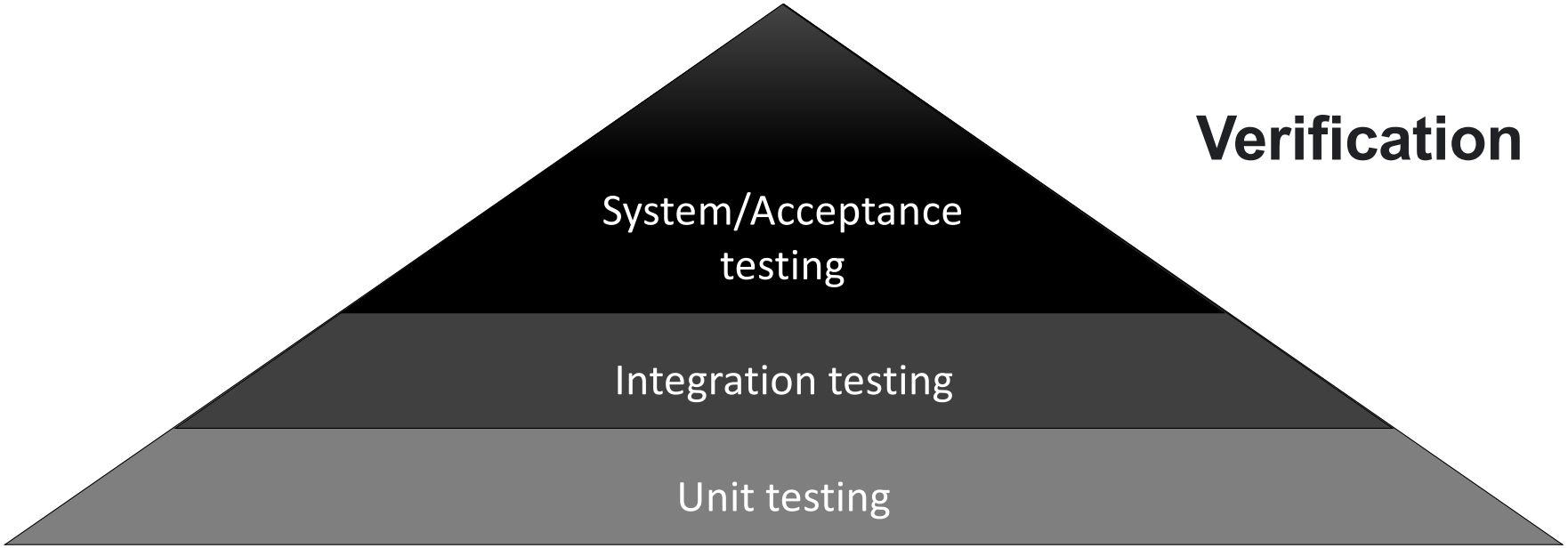
- Design Techniques:

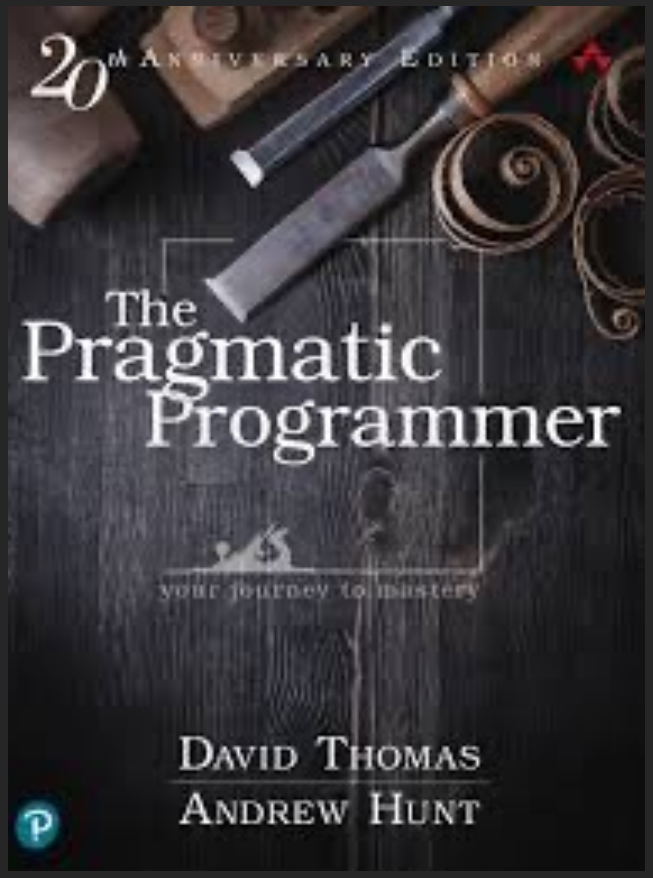
Test suite design and organization

Why testing?

Validation

Verification





—

“All software you write *will* be tested—if not by you and your team, then by the eventual users—so you might as well plan on testing it thoroughly … ”

Test is also a design tool

Reveal problems in poorly designed code

- Long setup code
- Long running tests
- Fragile tests
- …

What to test?

The program does what it is supposed to do.

Function Specifications

- Common cases
- Edge cases
- Code which has bugs before
- …

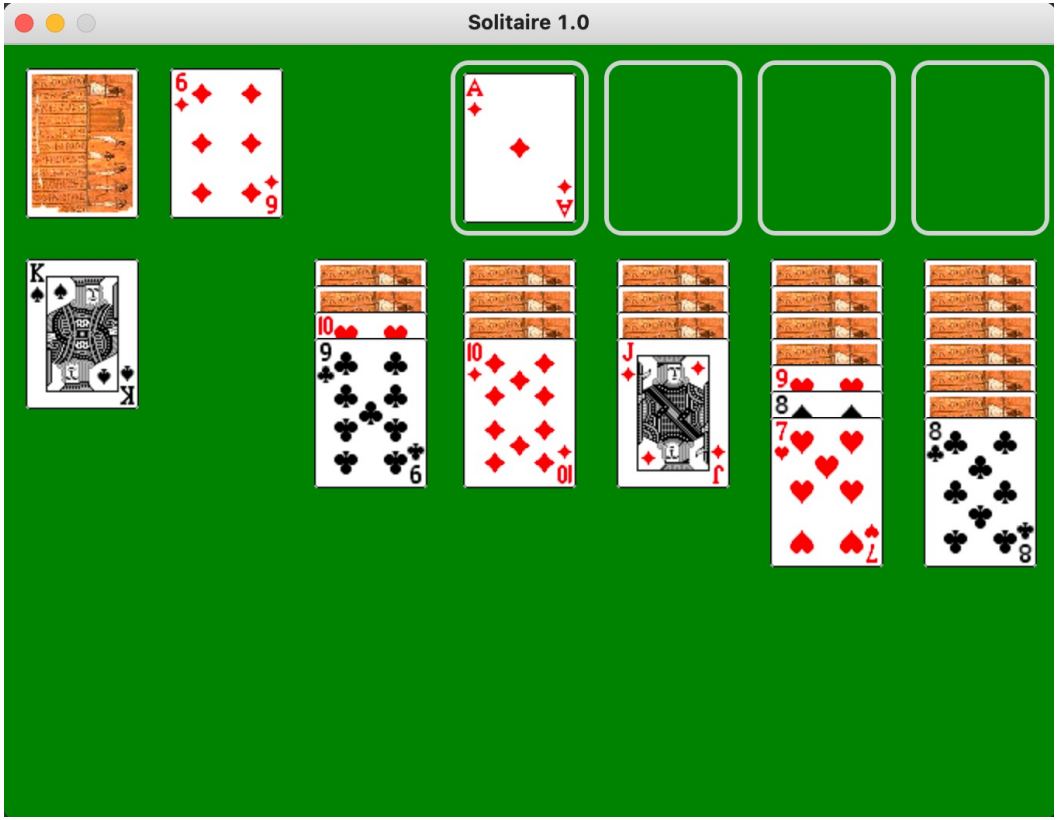
What constitute a “Unit”?

A small self-contained functionality

It runs fast

It is easy to know where to look for problems.

Activity: What constitute a “Unit”?



What is a test?

```
Student s1 = new Undergrad("001","Lily", "Joe");

System.out.println(s1.getFirstName());

System.out.println(s1.getFirstName().equals("Lily"));

assert s1.getFirstName().equals("Lily");
```

We need intuitive and automated solutions!

JUnit

- A Unit Testing framework

- Part of the xUnit family of frameworks (PHPUnit, PyUnit)

Example:

```
import org.junit.jupiter.api.Test;

import static org.junit.jupiter.api.Assertions.*;


class UndergradTest {

    @Test
    void getFirstName() {
        Student s = new Undergrad("001","Lily", "Joe");
        assertEquals("Lily", s.getFirstName());
    }
}
```

What is different from the code we normally write?

assertEquals method

Oracle

public static void assertEquals(Object expected, Object actual)

- *Asserts* that expected and actual are equal.
- If both are null, they are considered equal.

`@Test` Annotation

- Java Annotations -- a form of metadata
- Uses of annotation include:
  - Information for the compiler
  - Compile-time and deployment-time processing
  - Runtime processing

Java predefined annotation types:

- @Deprecated

  marked element is *deprecated* and should no longer be used.

- @Override

  informs the compiler that the element is meant to override an element declared in a superclass.

- @SuppressWarnings

  tells the compiler to suppress specific warnings that it would otherwise generate.

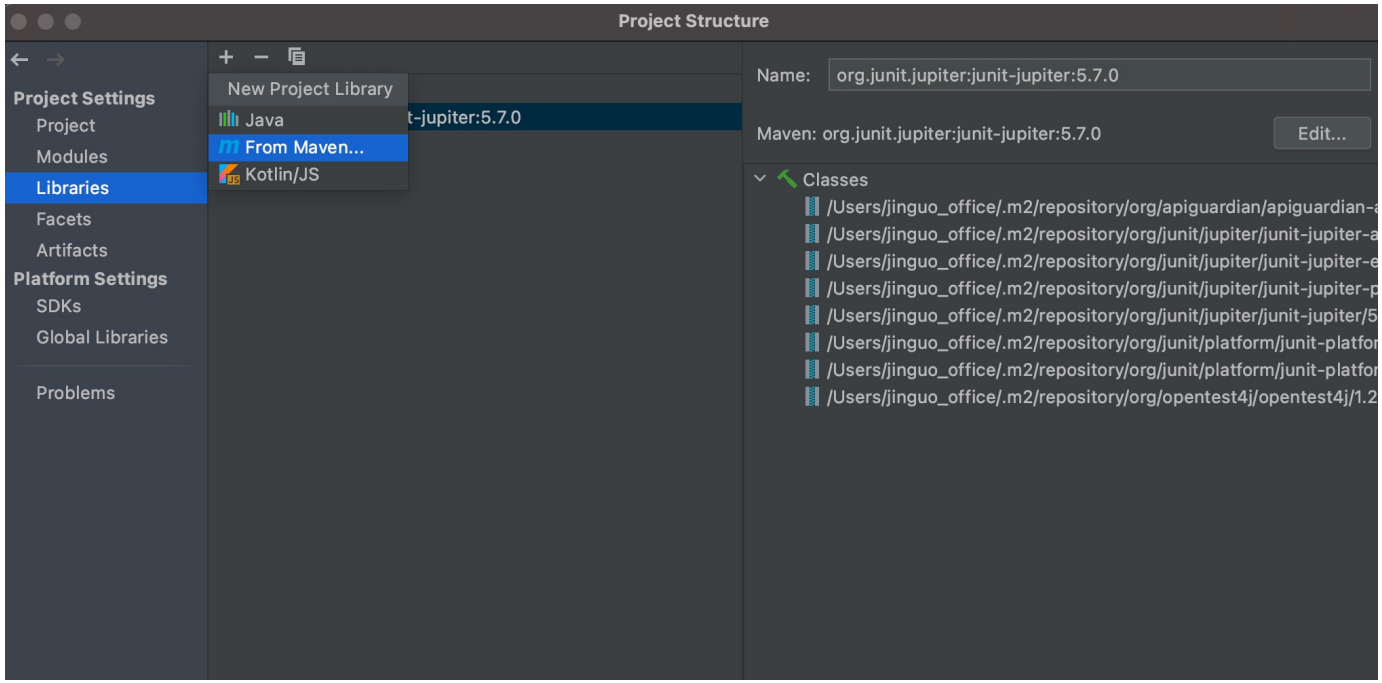
  ……

`@Test` Annotation

- To signal that the annotated method is a *test* method.
- Must not be private or static and must not return a value.

Configure JUnit using IntelliJ

- Instruction on M0-Exercise



Activity: Test this class

```
public class BrightnessFlashlight {
    private int brightness = 0;

    public void setBrightness(int pBrightness) {
        if (pBrightness < 0 || pBrightness > 5) {
            throw new IllegalArgumentException("Invalid brightness level");
        }
        this.brightness = pBrightness;
    }

    public int getBrightness() {
        return this.brightness;
    }

}
```

1. What are the units under test?
2. What are the expected behavior?
3. How to test if the expected behavior is the same with the actual behavior?

AAA (Arrange, Act, Assert) pattern

```
@Test
void testSetValidBrightness() {

    BrightnessFlashlight f = new BrightnessFlashlight();

    f.setBrightness(3);

    assertEquals(3, f.getBrightness());
}

@Test
void testDefaultBrightness() {

    BrightnessFlashlight f = new BrightnessFlashlight();

    assertEquals(0, f.getBrightness());
}
```

Arrange

Act

Assert

Test Invalid Brightness?

```
@Test
void testSetInvalidBrightness() {

        BrightnessFlashlight f = new BrightnessFlashlight();
      try {
        f.setBrightness(100);
          fail();

      } catch (IllegalArgumentException e) {
          // pass
      }
  }
}
```

public static <V> V fail(String message)

Fails a test with the given failure message.

What if we have a private method?

```
public class BrightnessFlashlight {

    private void incrementBrightness() {
        this.brightness = (this.brightness + 1) % 6;
    }
}
```

Objectives

- Concepts and Principles:

Unit testing, regression testing, test suites, test coverage;

- Programming Mechanisms:

Unit testing frameworks, JUnit, metaprogramming, annotations;

- Design Techniques:

Test suite design and organization