SOFTWARE ENGINEERING 3XA3

Software Engineering Practice & Experience: Software Project Management

Dr. Spencer Smith

McMaster University, Fall 2016

Laboratory 07 Unit Testing Frameworks

Revised: September 12, 2016

Components of Lab

1. Introduction to types of software testing
2. Introduction to test-driven development
3. Introduction to unit testing frameworks
4. GitLab Exercises

Details

- There are two main types of software testing: Unit Testing and Integration testing
  - Unit testing (aka white-box testing) is centered on testing the smallest logical unit of your code; typically this means a single function or method
  - Integration testing (also known as end-to-end testing or black-box testing) is a different style of testing where the entire system is tested at once
  - The following article goes into more detail about the kinds of testing: `http://softwaretestingfundamentals.com/unit-testing/`
- Test driven development is an iterative workflow where, the programmers write unit and integration tests before they write any code. Then, after they have finished a logical section of their code, they run the tests to see if they have written the code correctly.

- Most languages have some sort of testing framework, either as part of the standard library (such as in Python) or as a third-party library (such as in Java).
  - For today's lab, we will be providing examples in Python (2.7) and in Java. If your team has decided on a different language, you must find and familiarize yourself with that language's framework.
    - Documentation for Python 2's unittest: `https://docs.python.org/2/library/unittest.html`
    - Documentation for Java's JUnit: `http://junit.org/junit4/`
  - Here are some tutorial videos for other languages:
    - Unit Tests in Visual C++ (For C and C++ languages): `https://www.youtube.com/watch?v=2fxl7zMzfFM`
    - PHP-PHPUnit Testing (PHP): `https://www.youtube.com/watch?v=Iq6wvboGU-A`
  - The JavaScript community hasn't settled on one "best" unit testing framework; you'll have to choose one from the many that exist. Some suggestions:
    - Mocha: `http://mochajs.org`
    - Buster.JS: `http://docs.busterjs.org/en/latest/`
    - Jasmine: `https://github.com/jasmine/jasmine`
    - Ava: `https://github.com/avajs/ava`
    - and many more online...

Examples

Suppose we had a Line2D class that modeled lines. This class has the following methods:

- constructor `Line2D(float slope, float yIntercept)`
- `float getY(float xValue)`
- `float getX(float yValue)`

Our test cases would look like this:

Java

```java
import static org.junit.Assert.*;
import org.junit.BeforeClass;
import org.junit.Test;

public class LineTest {
    private static Line2D line;

    @BeforeClass
    public void setUp() {
        line = new Line2D(3, 2);  // y = 3x + 2
    }

    @Test
    private void testGetY() {
        // assertEquals is a JUnit method
        // the first arg (the string) is optional
        assertEquals("x value of 2 returns y value of 8",
line.getY(2),  8);
        assertEquals("x value of -2 returns y value of -4",
line.getY(2), -4);
        assertEquals("x value of 0 returns y value of 2",
line.getY(2),  2);
    }

    @Test
    private void testGetX() {
        assertEquals("y value of 5 returns x value of 1",
line.getX(5),  1);
        assertEquals("y value of -1 returns x value of -1",
line.getX(2), -1);
        assertEquals("y value of 0 returns x value of -2/3",
line.getX(2), (-2.0/3));
    }
}
```

Python

```
import unittest

class LineTest(unittest.TestCase):
    def setUp(self):
        self.line = Line2D(3, 2) # y = 3x + 2

    def testGetY(self):
        # assertEquals is a unittest method.
        # the third arg (the string) is optional.
        assertEquals(line.getY(2),  8,
"x value of 2 returns y value of 8")
        assertEquals(line.getY(2), -4,
"x value of -2 returns y value of -4")
        assertEquals(line.getY(2),  2,
"x value of 0 returns y value of 2")

    def testGetX():
        assertEquals(line.getX(5),  1,
"y value of 5 returns x value of 1")
        assertEquals(line.getX(2), -1,
"y value of -1 returns x value of -1")
        assertEquals(line.getX(2), (-2.0/3),
"y value of 0 returns x value of -2/3")

if __name__ == '__main__':
    # loadTestsFromTestCase looks for methods that
    # start with the word "test"
    suite = unittest.TestLoader().loadTestsFromTestCase(LineTest)
    suite.run()
```

Exercises

- On the gitlab repository hosting this document, you will find two files: `Box3D.Java` and `Box3D.py`. Select the one that corresponds to your prefered language.
- Write tests to verify that the following methods work correctly:
  - `getDimensionsOfFace`: accepts a `Face` enum and returns the dimensions of that face of the box.
  - `getAreaOfFace`: accepts a `Face` enum and returns the area of that face of the box.
  - `getPerimeterOfFace`: accepts a `Face` enum and returns the perimeter of that face of the box.
  - `getVolume`: returns the volume of the box.
  - `getSurfaceArea`: returns the total surface area of the box.
- Additionally, the code as written does some error handling, but it is lacking quite a bit of important error handling. Write some unit tests to demonstrate as many cases of missing error handling as you can.

For The TAs (Students can ignore this section)

Before the conclusion of the lab, please make sure that you have tested each student to verify that they understand the basics of the lab exercise. For students that understand the basics, please give them a grade of 1 for this Lab Exercise.