Unit Testing and JUnit

Moinul Hossain
CS 791Z
03/02/2015

Outline

- What is Software Testing?
- What and Why Unit Testing?
- JUnit
- JUnit features and Examples
- Test Driven Development (TDD)

What is Software Testing?

- Software testing is an activity to find defects in software codes

- History of Software testing dates back to 1961!
  - The book Computer Programming Fundamentals by Gerald Weinberg and Herbert Leeds contains a chapter on testing softwares

- Software testing in 70’s and 80’s was destruction oriented

- Modern day software testing is prevention oriented

What is Software Testing?

- Modern day softwares usually consist of four levels of testing:
    - Unit Testing
    - Integration Testing
    - System Testing
    - Acceptance Testing

What is Unit Testing ?

- Unit testing is a software testing method by which individual units of source code are tested with automatic test cases

- The goal of unit testing is to isolate different parts of the program into units and show that the individual unit works as expected

- Unit testing is done by the developers. Unit testing is developers version of ‘acceptance test’

Why Unit Testing ?

- Unit testing finds problems early in the development cycle

- Unit testing simplifies integration

- Unit tests provide a living documentation of the system

- Unit testing helps the design by allowing to apply principles like decoupling

JUnit

- An unit testing framework for Java
- Developed by Kent Beck (creator of XP) and Erich Gamma
- Most popular testing framework for Java
- Descendent of xUnit framework family
- Other available frameworks: TestNG, Arquillian, JTest etc.

Current Version: 4.12

Website: junit.org

Why JUnit ?

- Open source !
- Being actively developed over a decade
- Has a very good ecosystem around the developers community
- Has good support for integration with all major IDEs
- Has good support integration with project management and build tools like Maven and Gradle

A Basic Example of Unit Testing

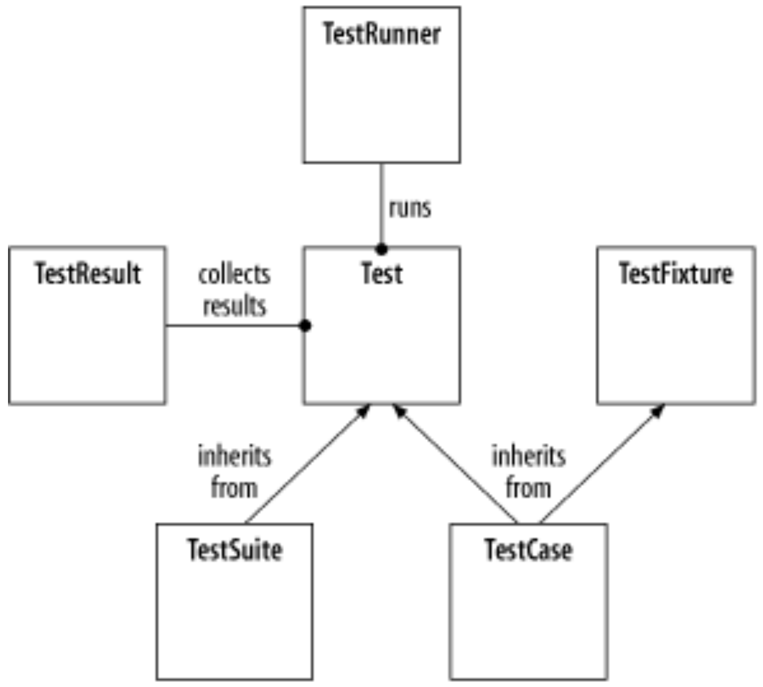
```java
interface Calculator {

    int add(int a, int b);

}



class CalculatorImpl implements Calculator {

    int add(int a, int b) {

        return a + b;

    }

}
```

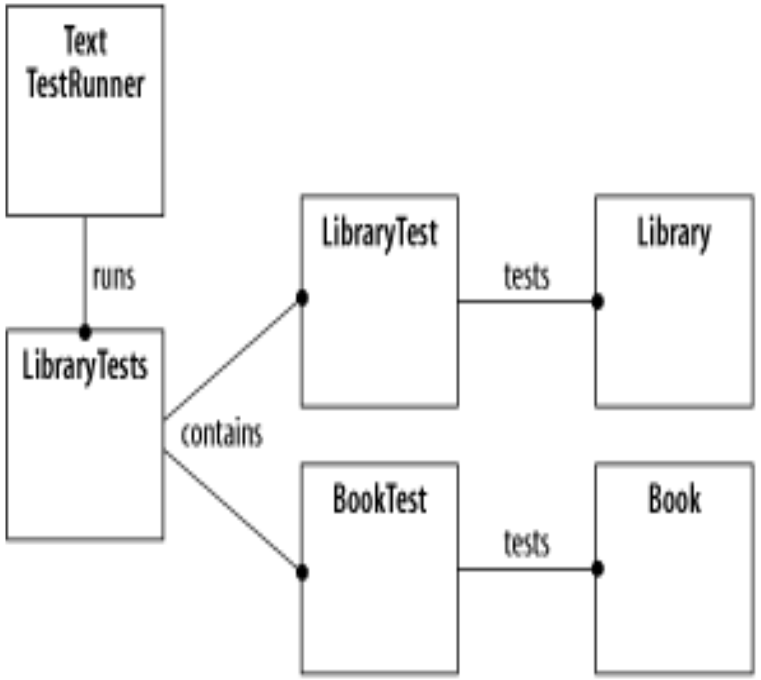
```java
public class TestCalculator {

    // can it add the positive numbers 1 and 2?
    public void testSumPositiveNumbersOneAndTwo() {
        Calculator calc= new CalculatorImpl();
        assert(calc.add(1, 2) == 3);
    }

    // can it add the negative numbers -1 and -2?
    public void testSumNegativeNumbers() {
        Calculator calc= new CalculatorImpl();
        assert(calc.add(-1, -2) == -3);
    }

    // can it add a positive and a negative?
    public void testSumPositiveAndNegative() {
        Calculator calc= new CalculatorImpl();
        assert(adder.add(-1, 1) == 0);
    }

    // how about larger numbers?
    public void testSumLargeNumbers() {
        Calculator calc= new CalculatorImpl();
        assert(adder.add(1234, 988) == 2222);
    }
}
```

A unit functionality

Covered tests for the functionality

JUnit Architecture





A Basic Example using JUnit

```java
interface Calculator {

    int add(int a, int b);

}



class CalculatorImpl implements Calculator {

    int add(int a, int b) {

        return a + b;

    }

}
```

```java
public class CalculatorTest {

    Calculator calc;

    @Before
    public void setUp(){
        calc = new CalculatorImpl();
    }

    @Test
    public void testSumPositiveNumbersOneAndTwo(){
        assertEqual(calc.add(1, 2) == 3);
    }
}
```

JUnit Features

- Assertions: Specify expected output of an unit and compare
- Test set up and teardown: As easy as using **@Before** and **@After**
- Exception Testing: Test whether an Exception has been thrown by a piece of code
- Test Suites: JUnit provides option to have suites of tests to better organize the tests
- Stub and Mock Objects: Provides support to use stub as placeholder for complex or yet to be developed code

JUnit Features

- Test Reports: provides options to generate test reports in different formats like HTML, XML etc

- Support for build systems: Has strong support for integration with different build systems like Ant, Maven and Gradle

Popular JUnit Extensions

- Cactus: For testing server side Java codes
- HTMLUnit: Models HTML documents and provides an API that allows you to invoke pages, fill out forms, click links, etc.
- Mockito: Popular object mocking framework built on top of JUnit

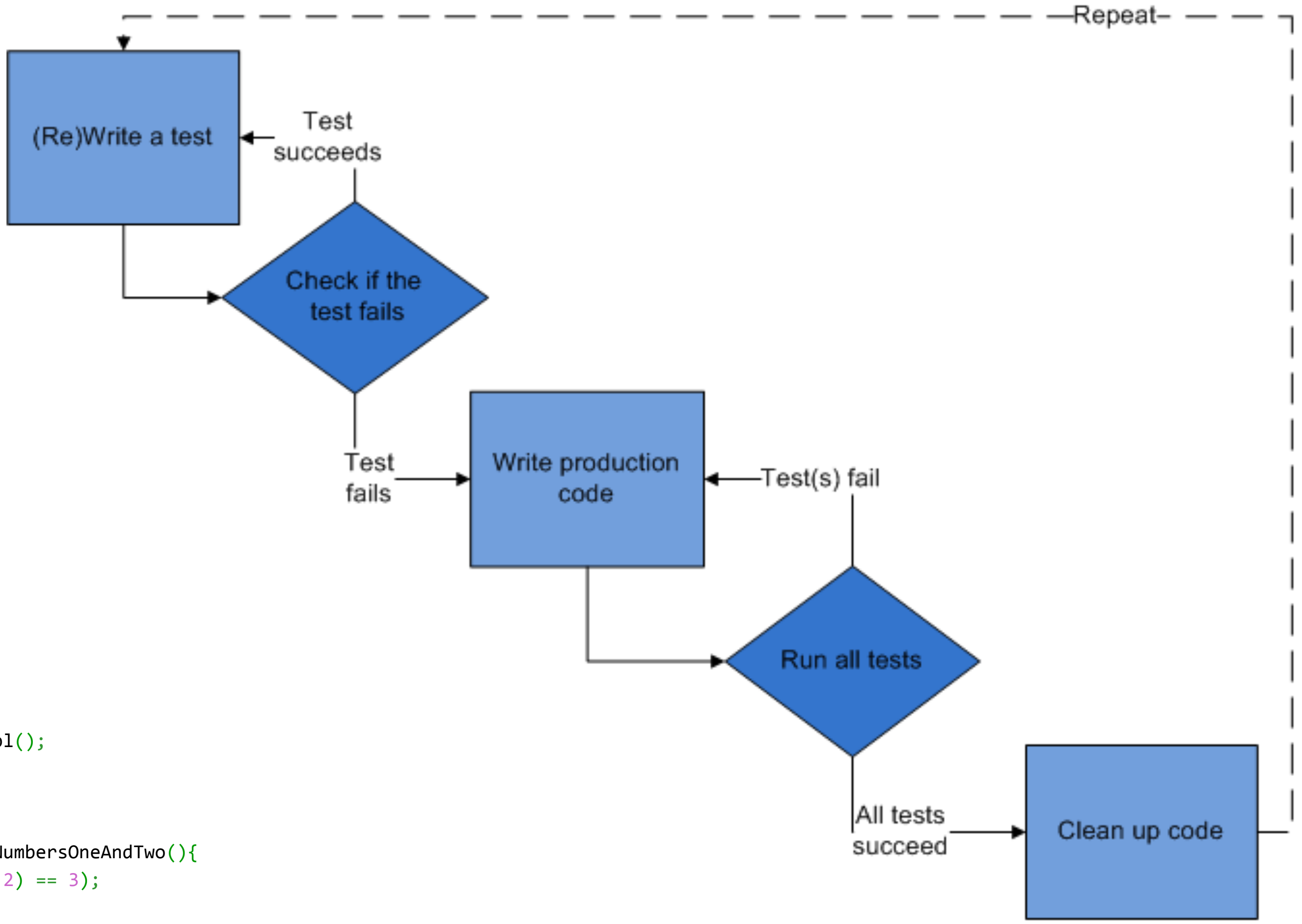
Test Driven Development (TDD)

- Popular software development process often used with agile process like SCRUM

- TDD: Write the test code first, then write the development code

- TDD forces to think **how to use a component** first and then **how to implement a component** consequently

Test Driven Development (TDD)



```
public class CalculatorTest {

    Calculator calc;

    @Before
    public void setUp(){
        calc = new CalculatorImpl();
    }

    @Test
    public void testSumPositiveNumbersOneAndTwo(){
        assertEqual(calc.add(1, 2) == 3);
    }
}
```

Questions?