CS304 Software Engineering

Lab 5: Unit and System Testing

Self-introduction

- Name: Yushan Zhang(张雨姗) *Homepage: zhangyushao.site
- Supervised by Dr. Yuqun Zhang for undergraduate capstone and advanced research in software engineering (especially testing, repair, synthesis)
- Will join HKUST this fall as new research postgraduate student for a PhD degree in Cybersecurity team. (**Talk to me** if you are interested in our research team or company SourceBrella, we could offer possible PhD/RA or ~~research~~ intern positions.)

- Office hour: 2-3 pm Fri, 1009 Zhiyuan
- For questions email with Title **LecX+Question to** zhangys3@mail.sustc.edu.cn
- It is **recommended** if you could also cc to zhangyq@sustc.edu.cn

Note:

- Submit your assignment as **following folder structure**:

```
-/ 11310380 (student number)
-- / code
---/ src
---- main.java
---/ test (if you have)
---- testAll.java
--/ docs
--- README.md
--- comments.txt
```

- We only accept **PDF** as docs (except if noted).
- Using version control system if you can to keep track of revisions.

Lab Submission

- Every 20% penalty of total grade of each lab each day. (From Lab5 on)
- **NO points** received after 3 days.

- Possible all independent assignments from Lab 5 on.
- Be careful of plagiarism.

- If you cannot submit with Sakai: send email to zhangys3@mail.sustc.edu.cn with Title **LabX+stuNo**+reason

Any questions?

- QQ group: 397544953

- My WeChat: zhangysh1995

(PLEASE no course-related conversations, only through email or qq group)

Reminder:

- Please focus **more on concepts** rather than specific skill or tools.
- Software Engineering is not only code, but it is the **process** how the final application is planned and implemented.

- If you want to know more about **Best Practice**, you could choose to intern at a big company such as Tecent.

Outline

- Testing
- JUnit
- Fault-localization
- Tips

Kinds of Testing

- Unit testing: the execution of a complete class, routine, or small program or team of programmers
- Component testing: the execution of a class, package, small program, or other program element
- Integration testing: the combined execution of two or more classes, packages, components, or subsystems
- System testing: the execution of the software in its final configuration, including integration with other software and hardware systems
- Regression testing: the repetition of previously executed test cases for the purpose of finding defects

Types of Testing

- **Black-box testing**: tests in which the test cannot see the inner workings of the item being executed

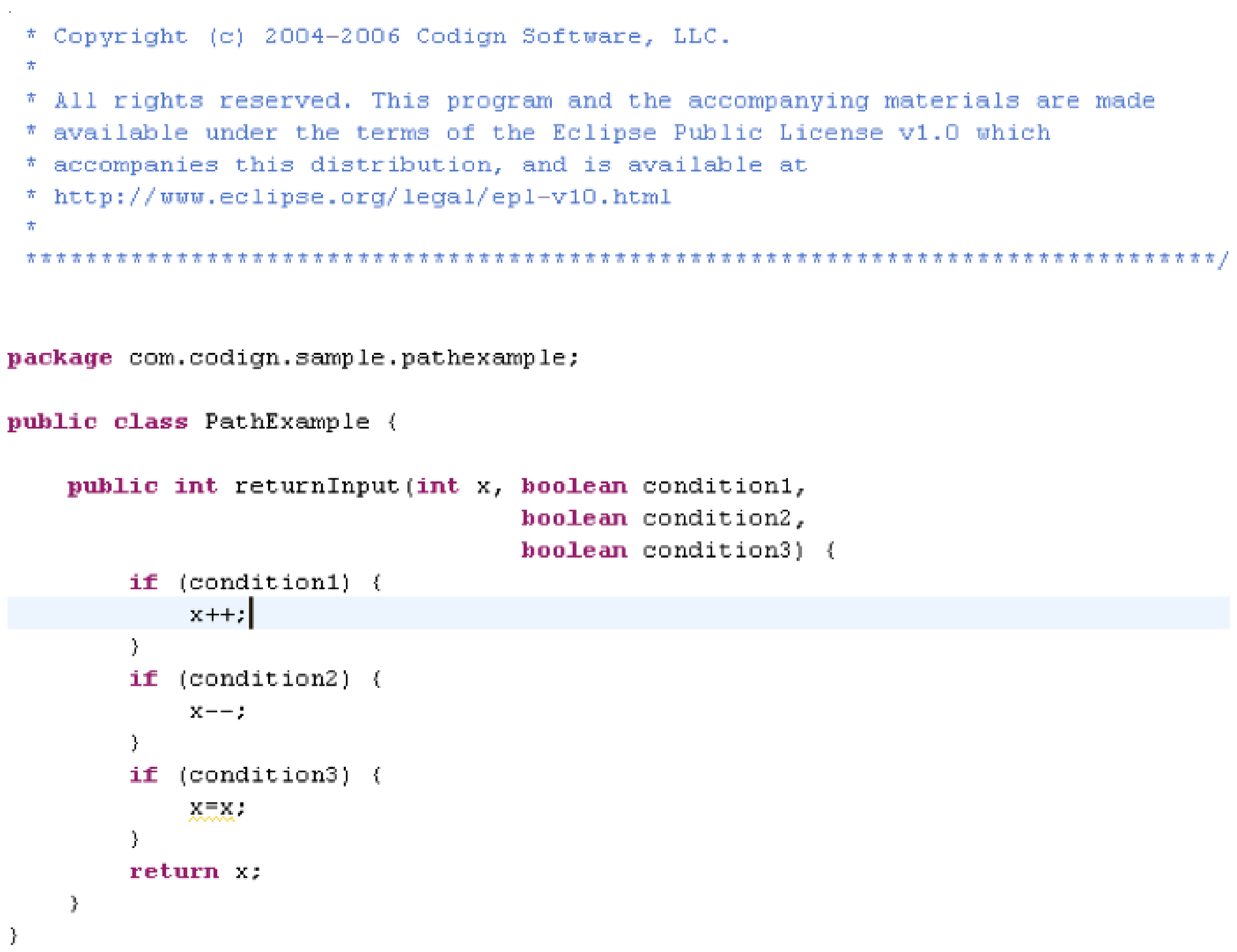
- **White-box testing** : tests in which the tester is aware of the inner workings of the item being tested

- **Gray-box testing, fuzzing (more offensive)…**

Test Coverage

- Statement coverage: has each statement been executed?
- Branch coverage: has each control structure evaluated to both true and false?
- Path coverage: has every possible route been executed?

Branch and Path Coverage Example



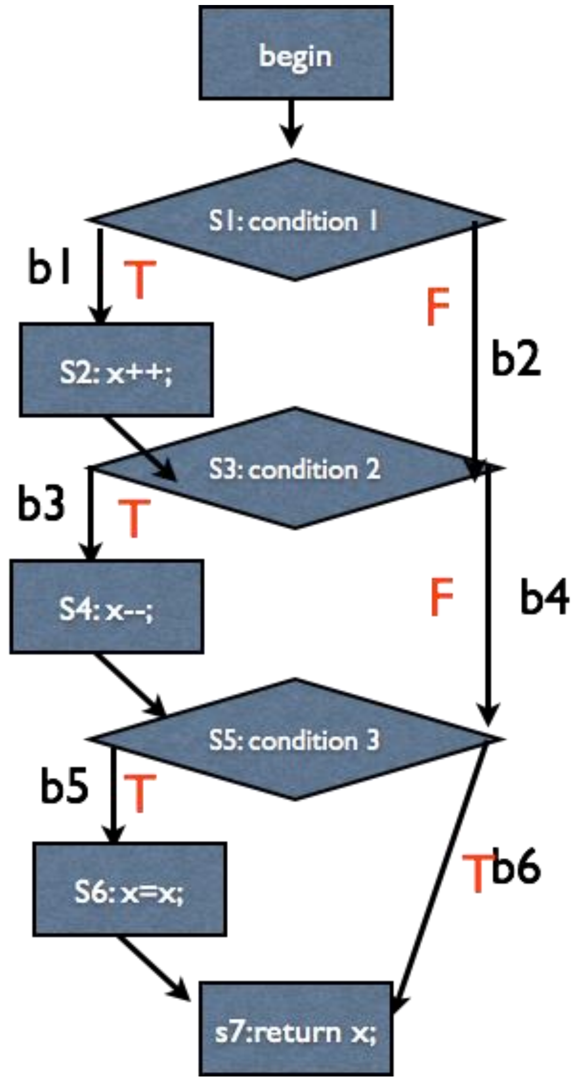
```
 * Copyright (c) 2004-2006 Codign Software, LLC.
 *
 * All rights reserved. This program and the accompanying materials are made
 * available under the terms of the Eclipse Public License v1.0 which
 * accompanies this distribution, and is available at
 * http://www.eclipse.org/legal/epl-v10.html
 *
 ********************************************************************************/


package com.codign.sample.pathexample;

public class PathExample {

    public int returnInput(int x, boolean condition1,
                                  boolean condition2,
                                  boolean condition3) {
        if (condition1) {
            x++;
        }
        if (condition2) {
            x--;
        }
        if (condition3) {
            x=x;
        }
        return x;
    }
}
```

Branch and Path Coverage Example



Fill out the following code coverage table by running the program with the following inputs

input	exercised statements	exercised branches	exercised paths
(cond1=true, cond2=true, cond3=true)	s1, s2, s3, s4, s5, s6, s7	b1, b3, b5	[b1, b3, b5]
Coverage			
(cond1=false, cond2=false, cond3=false)			
Coverage			
(cond1=false, cond2=true, cond3=true)			
Coverage			

This Can Quickly Get CRAZY

```
public static int fun1(int N) {
    int sum = 0;
    for (int i = 1; i <= N; i++) {
        for (int j = 1; j <= Math.pow(3, i); j++) {
            System.out.println("HelloWorld");
            if (new Random().nextInt() % 2 == 0)
                sum++;
        }
    }
    return sum;
}
```

Has an exponential number of paths

How to explore???

Executing Component Tests

- If a test fails, the subsequent test cases are no longer executed
- One should be able to run tests individually, independent of other test cases
- One should be able to group tests into *test suites*
- One should be able to grasp immediately whether tests have failed and, if so, which ones

Setting up Fixture

- Tests frequently need some **fixture** to execute
  - Configuration files that must be read and processed
  - External resources that must be requested and set up
  - Services of other components that must be initialized
- Setting up:
  - The method `setUp()` or **`@Before`** is called before each test of the class
- Tearing down:
  - The method `tearDown()` or **`@After`** is called after each test (it is used for releasing the fixture)

JUnit

- *Automated* unit testing framework
  - Provides the **required environment** for the component
  - Executes the **individual services** of the component
  - **Compares** the observed program state with the expected program state
  - Reports any **deviation** from the expectations
  - Does all of this automatically

JUnit TestCase Example

```
import junit.framework.*;
public class RationalTest extends TestCase {
// Create new test
    public RationalTest(String name) {
        super(name);
    }
    public void testEquality() {
        assertEquals(new Rational(1,3), new Rational(1,3));
        assertEquals(new Rational(2,6), new Rational(1,3));
        assertEquals(new Rational(3,3), new Rational(1,1));
        assertFalse(new Rational(2,3).equals(new Rational(1,3)));
    }
}
```

```
$ java -classpath .:/wherever/junit.jar junit.textui.TestRunner RationalTest
```

Example Test Fixture

```
public class RationalTest extends TestCase {
    private Rational a_third;

    // Set up fixture
    // Called before each testXXX() method
    protected void setUp() {
        a_third = new Rational(1,3);
    }
    // Tear down fixture
    protected void tearDown() {
        a_third = null;
    }
    ...
}
```

Another JUnit Example

```
public class VectorTest extends TestCase {
  protected Vector fEmpty;
  protected Vector fFull;
  // public VectorTest(String name);
  protected void setUp() {
    fEmpty = new Vector();
    fFull = new Vector();
    fFull.addElement(new Integer(1));
    fFull.addElement(new Integer(2));
    fFull.addElement(new Integer(3));
  }
  // continued…
```

Some state to refer to the SUT instance

Typically use the implicit constructor (why not?)

Set up the test fixture

https://stackoverflow.com/questions/6094081/junit-using-constructor-instead-of-before

Assignment 1 (mandatory) : Due 23rd, April

- **PDF Tutorial: Lab 5 --Unit and System Testing**
- **To submit:** all code with one report (please format) to show the run time results and write down your thoughts.

- It would be great if you could complete this assignment in Intellij IDEA, which is a more popular IDE for Java these days.

- I am more familiar with IDEA, if you have questions just ask. Or if you have problems with Eclipse, I will try my best to help. I recommend to migrate to IDEA for better user experience.

Assignment 2 (optional):

- Integrate testing with Gradle or Maven
- Write tests for you previous course work or project.
- Try to design tests for CS304 project. (recommended, easier for you to evaluate the code)

Tips

- Draw UML diagrams to specify program behavior. This is really helpful if you need API interface and want to split your project to several parts.
- Check-out TDD (Test Driven Development) if you have difficulty implementing modules as described in your docs.
- For teamwork, Git or Subversion is better for code review and version control.
- If you need continuous integration（可持续集成）, try out Travis-Ci on GitHub.