School of Computer Science

Contemporary Software Development

COMP47480

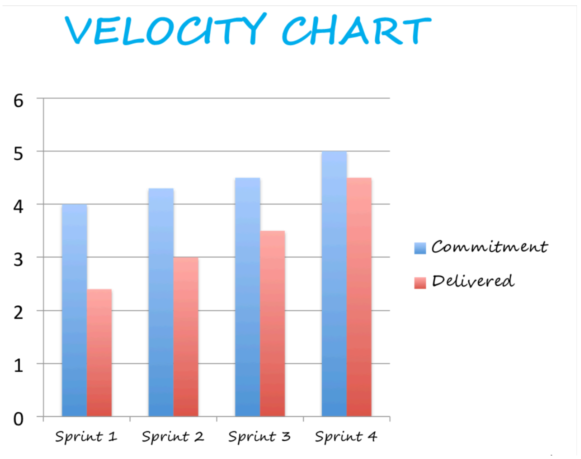
Lab Journal 1

Xiaosheng Liang - 15211913

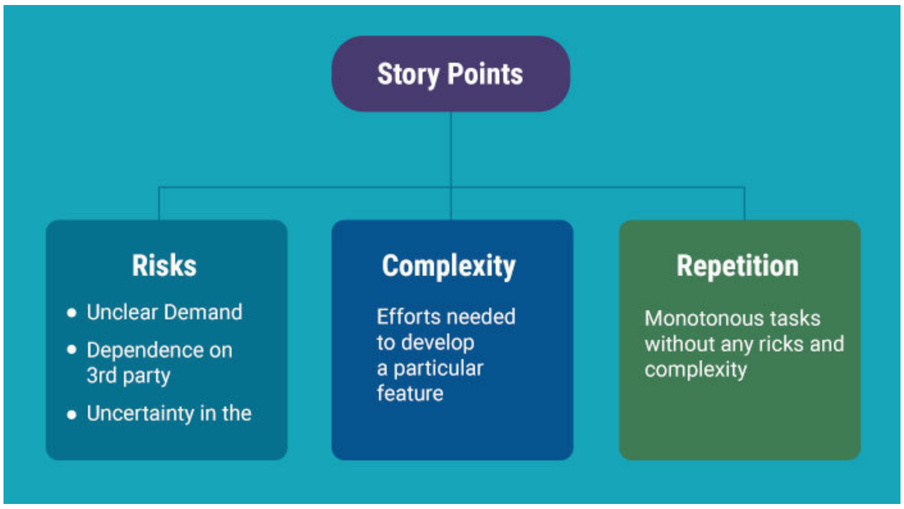
Agile “Berlin Bunker” Exercise

Work done:

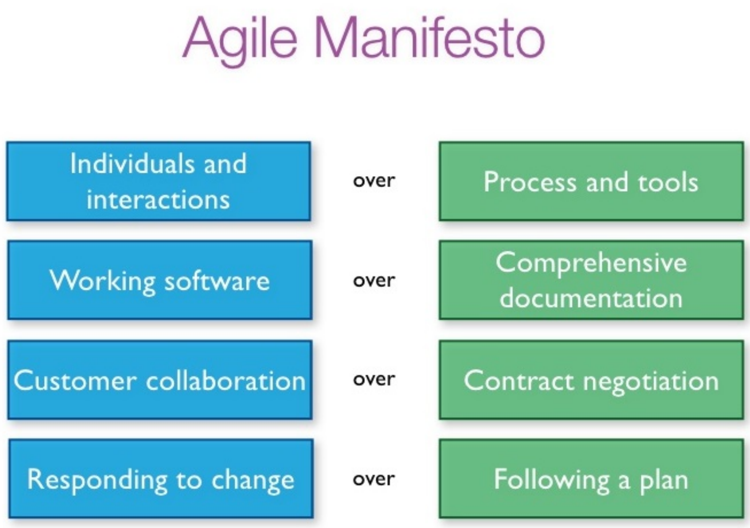
- In Lab1 Agile “Berlin Bunker” Exercise, we are provided a number of terms to explain and describe their context in the video. All the terms are divided into three groups.
- My group of terms are velocity, hourly build, story point, Agile Manifesto and burndown chart.
- After finished, three group members discussed, compared notes and suggested the improvement.
- velocity: The velocity is how much work (or how many tasks) have been done in a certain time- period. The tasks are counted by the number of same units of project which have been decided at the beginning of the project. The manager asks about the velocity, indicating that he wants to know how long it would take to finish the project.



- hourly build: Hourly build means frequent and regular builds to make sure the integration occurs frequently and avoid the problem. The manager asks his team to run the build once an hour and integrate frequently.
- story point: Story points are used to calculate the effort, complexity and risk needed for user stories. It’s a unit of estimating the effort it would take to finish a story. The manager requires his team get credit for story points after all the test done.

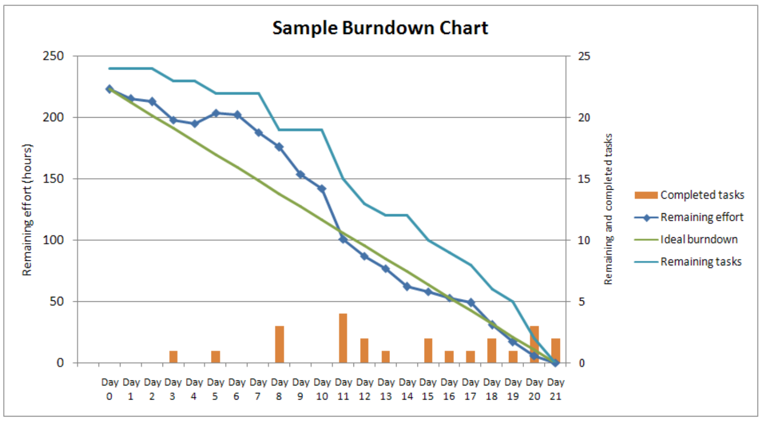

Story Points
Risks
Unclear Demand
Dependence on 3rd party
Uncertainty in the
Complexity
Efforts needed to develop a particular feature
Repetition
Monotonous tasks without any ricks and complexity

- Agile Manifesto: Agile Manifesto officially announces the four core values and twelve principles that guide the software development approach of people-centered iteration. It's foundation of agile development, which the manager still believes, despite current failure to deliver.



  -

- waterfall: The "traditional" approach to software development involving a sequence of phases, analysis, design, implementation, testing, maintenance. It's the antithesis of Agility, so the manager mocks his team that they want to "go back to waterfall."
- burndown chart: Burndown chart is a visual representation of the work that needs to be done before the project is completed. Burndown charts do look hopeless means all the works are almost done.



Lessons Learned:

- The methodology is a splitting of software development work into distinct phases and the approach we use to system building.
- We used to use traditional methodologies, such as Code and Fix and Waterfall Process. The problems of these methodologies are hard to maintain, have any changes or make mistakes during development.
- For incremental development, system could be tested and got feedback earlier.
- For iterative Development, it begins by specifying and implementing just part of the software, which can be reviewed in order to identify further requirements. And it is used to solve risk problems in development.
- The two key features of any current methodology are incremental and iterative. In the unified process,, iterative should be controlled and it applies to process a group of user case to extend availability. Increment is improvement of system design in the initial phase of life cycle, and increase of original products in the subsequent stages.

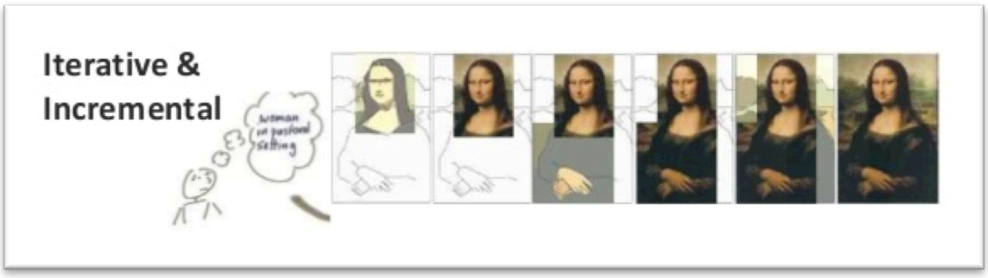


Figure 1. Iterative and Incremental Methodology

- Extreme programming is a software development methodology which intended to improve software quality and responsiveness to changing customer requirements.
- In extreme programming, testing happens first. The test case should be finished before the code is implemented any functionality.

- Refactoring is the process to make the design more flexible.
- Other features of extreme programming are pair programming, single coding style and original authorship irrelevant.

Reflections:

The reflections of exercise:

All the terms have been learnt in the lecture. After describing the terms base on the context of video, all words related to software methodology and extreme programming can be understand deeply.

Compared with previous development methods. Agile development enables some benefits to be realized early as the product continues to develop. Small incremental releases made visible to the product owner and product team through its development help to identify and issues early ad make it easier to respond to change. Above all of points, the ability for agile development requirements to emerge and evolve, and the ability to embrace change (with the appropriate trade-offs), the team build the right products.

A deeper study of extreme programming:

- The principle of extreme programming: keep awake, accommodating and changing. Just like driving a car, keep focus and keep continuous adjusting steering wheel so that it should be in the straight line.
- For Implementing extreme programming, a cross-functional team, information workspace, pair and personal workspace, weekly cycle, relaxation, ten-minute build, test-first programming are essential.
- Inclusion of the client means that we are able to capture many the real requirements early enough. A complete team should include the client.
- Once a mistake found, root cause analysis should be done so that the problems are no longer appear.
- Pay-per-use: making money in advance and delaying spending can improve the value of software development. Pay-per-use gives us a convenient way to do this: once the function deployed, the income generate.
- Sharing code: anyone in the team can improve any part of the system at any time. Everyone in the team should be responsible to the system.

School of Computer Science

Contemporary Software Development

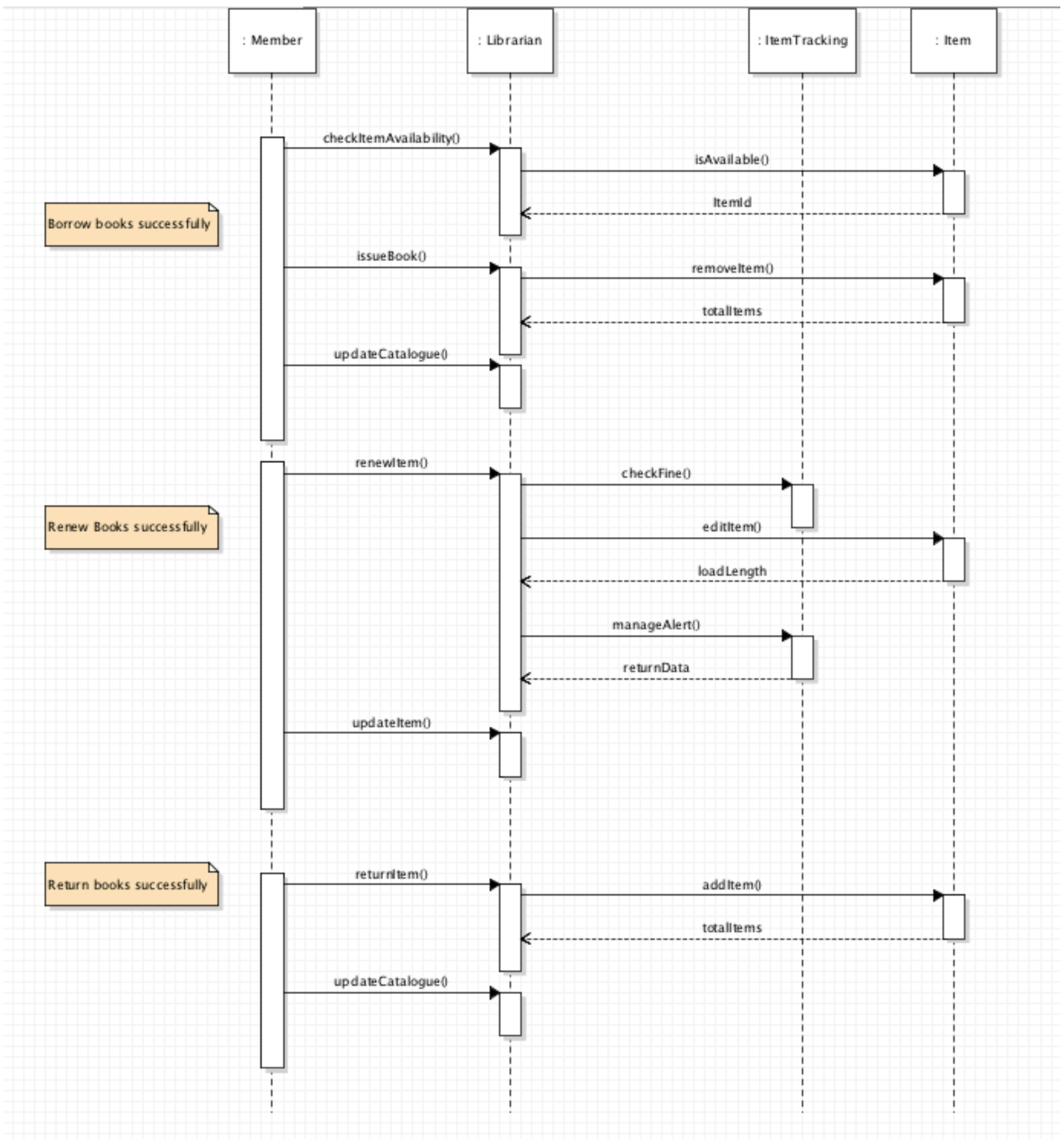
COMP47480

Lab Journal 2

Xiaosheng Liang - 15211913

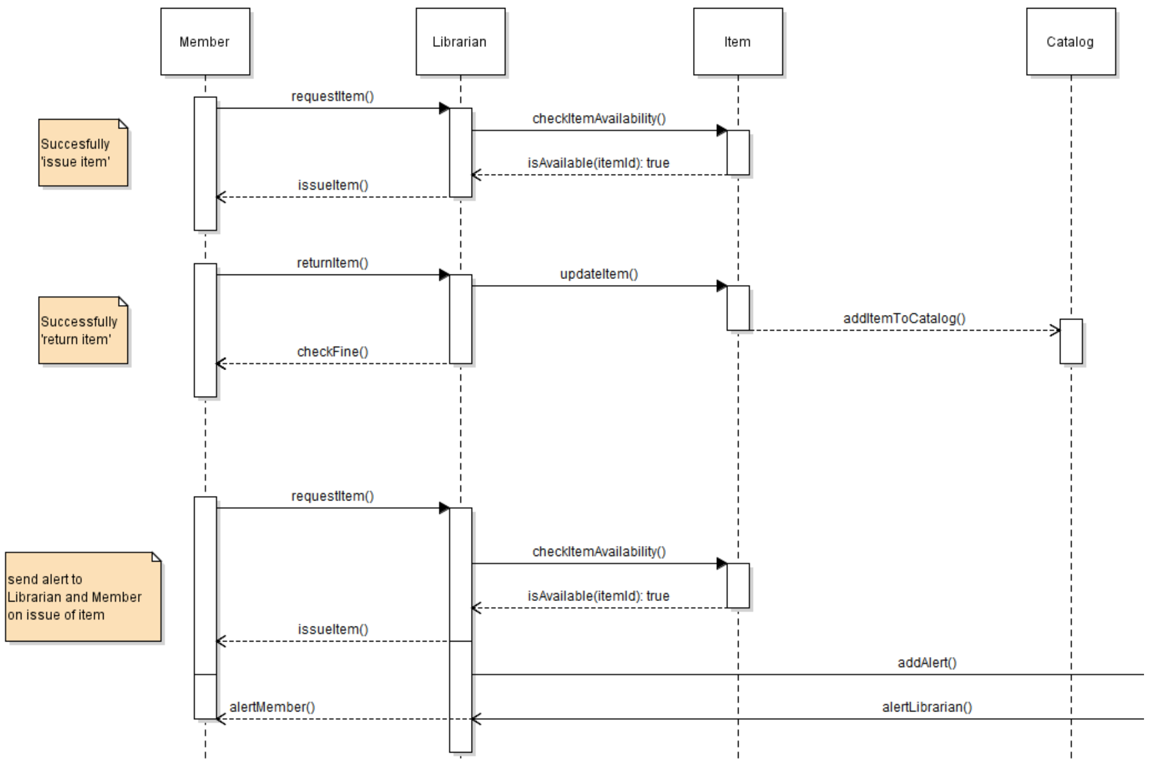
Work done:

In Lab2, ”Modeling with UML”, my group members and I finished the use case diagram and class diagram in class. Then we had the second group meeting the next day to fix use case diagram and class diagram and finish state diagram. At the weekends, I did sequence diagram for borrowing books, returning books and renewing books. As follow:

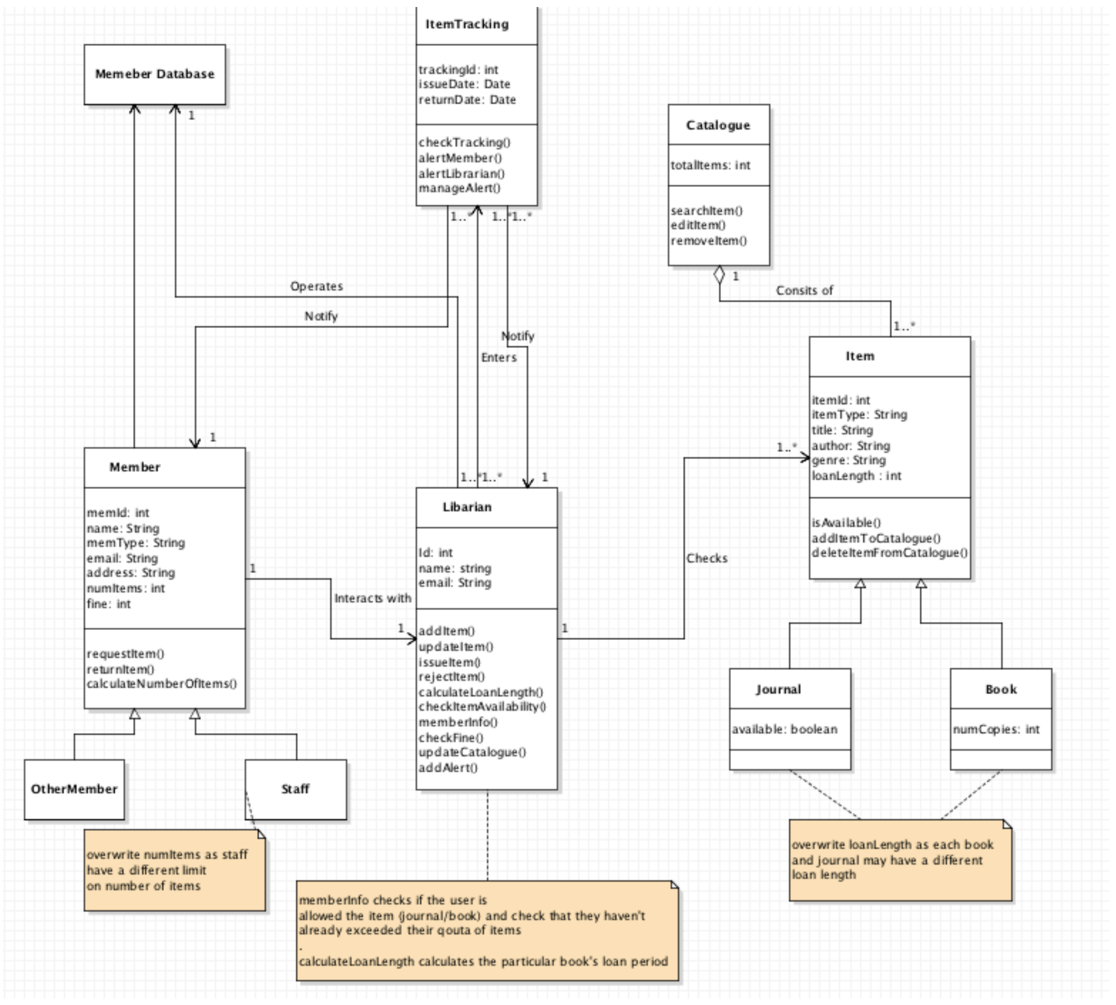


In the third group meeting my group members pointed out the problems in this diagram. In this diagram, all the arrows are pointed from method caller to implementer. In library system we did, members is one of class not the actor, in other word, it’s not a self-service system, so the method should be pointed from its own class. During this process, we still fixed use case diagram and class diagram to make sure they are consistent.

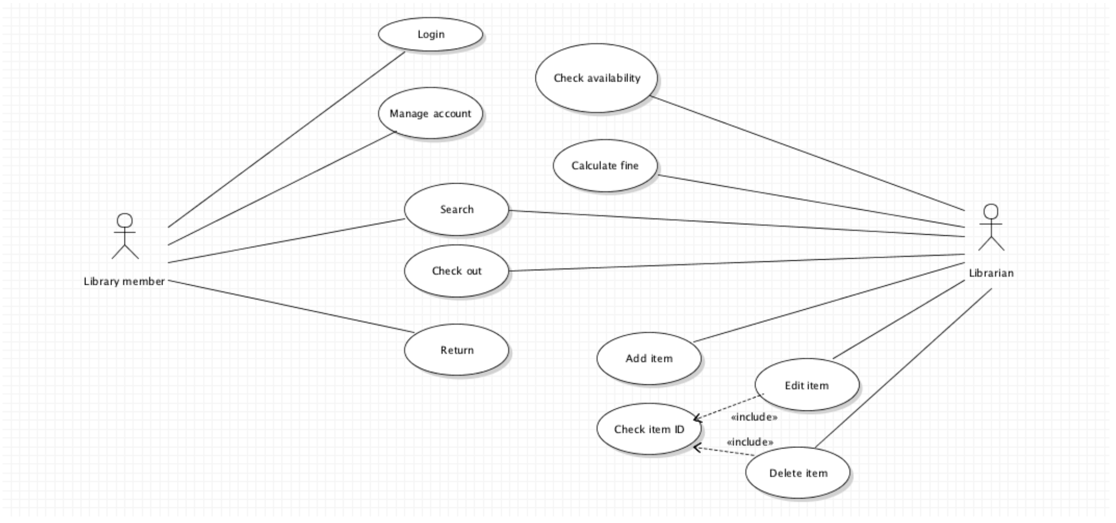
After discussing, we decided to delete the renew sequence diagram and add alert function into borrowing diagram, as follow:



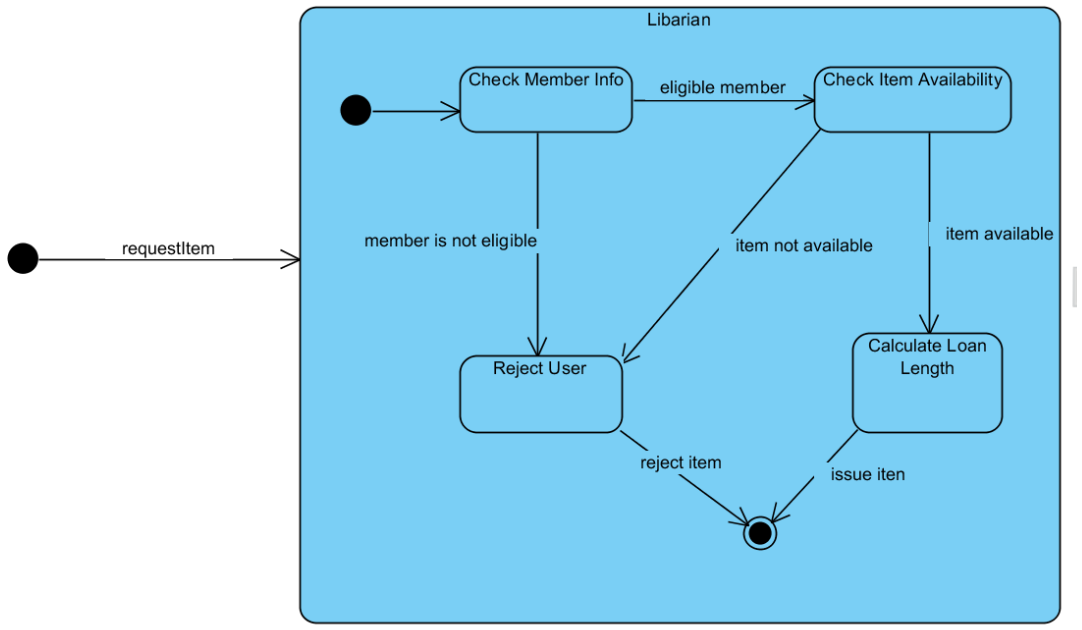
Use case diagram and Class diagram which are fixed are as follow:



The biggest change in class diagram is that we removed methods Librarían called from Catalogue, all the methods of Item should be accessed to Item directly. After fixing class diagram, we fixed use case diagram as well.



As for state diagram, we did Librarian class' state diagram.



Lessons Learned:

The Unified Modeling Language (UML) is a general purpose, developmental, modeling in the field of software engineering, that is intended to provede a standard way to visualize the design of system.

The use case is to describe the functionality of system. Class diagram is first-cut class model which design can evolve from initial domain model. A sequence leads to a set of use case realization and a refined class diagram. State diagram is a summary of what happen to an object which is used for a specific class.

Reflections:

After this Lab, I have deep understanding of UML. All diagrams can not be finished once. The constant revision is just the part of UML process, and it's essential to do it. Having iterations of design allows we to explore a clearer structure and a complete design of the system and make sure all the classes and all the methods are consistent.

During the processing of practical, I learned different diagrams have different functions. As Class diagrams are arguably the most used UML diagram type. It's the main building block of any object oriented solution. As for use case diagram, it gives a graphic overview of the actors involved in a system, different functions are interacted. State diagram is very useful to describe the behavior of objects that act differently according to the state they are in at the moment. In addition, sequence diagram is important to note that it shows the interactions for a particular scenario.

Since all the diagrams are not able to be done once. We have to be very patient when we are doing diagrams and fix it again and again to make it perfect.

School of Computer Science

Contemporary Software Development

COMP47480

Lab Journal 3

Xiaosheng Liang - 15211913

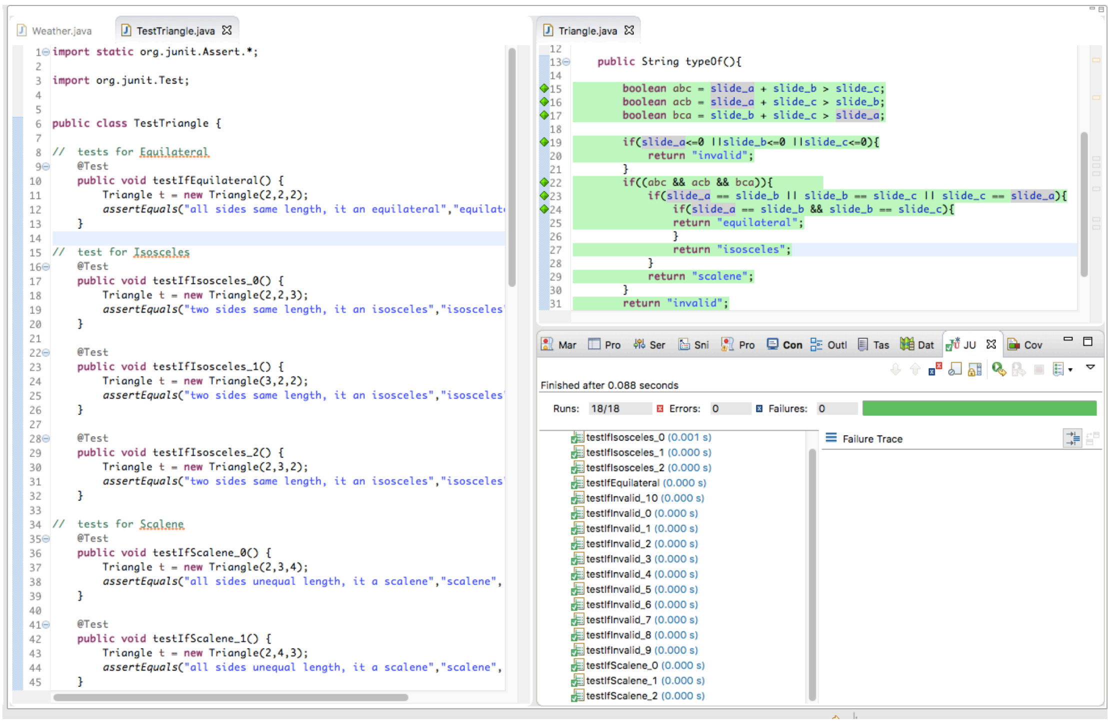
Work Done:

In Lab3, JUnit, my group members and I finished it individually. In the last Tuesday's lab, I finished the first part and just started the second part in class. As for part 1, a warm-up exercise, it made me easy to understand JUnit test basic processing by following the instructors. Learning the thinking of TDD approach and the method of implementing is the key of this exercise.

In the next part, it is to test a Triangle class, a more complicated program. It is a multiple conditions nested program. We are supposed to come up with sufficient tests to make sure that every case would be covered. All the tests are as follow:

- One test is for "Equilateral".
- Three tests are for "Isosceles".
- There tests are for "Scalene".
- Eleven tests are for "Invalid"

The result as shown below:



Weather.java | TestTriangle.java

```
import static org.junit.Assert.*;

import org.junit.Test;


public class TestTriangle {

//  tests for Equilateral
    @Test
    public void testIfEquilateral() {
        Triangle t = new Triangle(2,2,2);
        assertEquals("all sides same length, it an equilateral","equilat
    }

//  test for Isosceles
    @Test
    public void testIfIsosceles_0() {
        Triangle t = new Triangle(2,2,3);
        assertEquals("two sides same length, it an isosceles","isosceles
    }

    @Test
    public void testIfIsosceles_1() {
        Triangle t = new Triangle(3,2,2);
        assertEquals("two sides same length, it an isosceles","isosceles
    }

    @Test
    public void testIfIsosceles_2() {
        Triangle t = new Triangle(2,3,2);
        assertEquals("two sides same length, it an isosceles","isosceles
    }

//  tests for Scalene
    @Test
    public void testIfScalene_0() {
        Triangle t = new Triangle(2,3,4);
        assertEquals("all sides unequal length, it a scalene","scalene",
    }

    @Test
    public void testIfScalene_1() {
        Triangle t = new Triangle(2,4,3);
        assertEquals("all sides unequal length, it a scalene","scalene",
    }
```

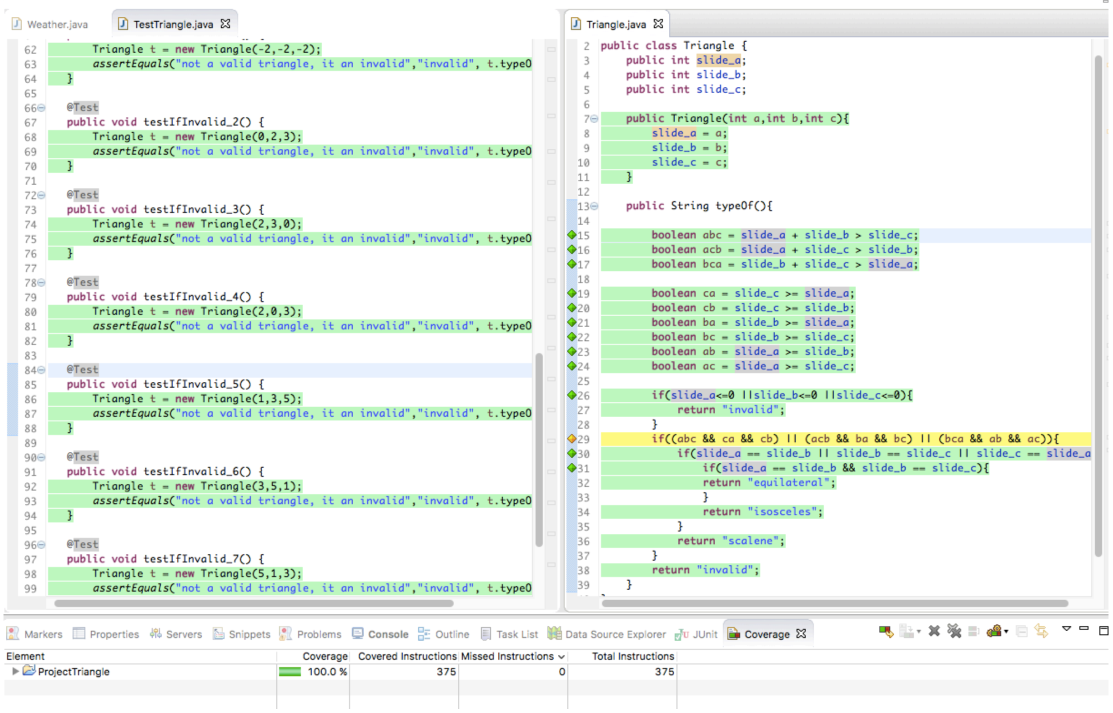
Triangle.java

```

    public String typeOf(){

        boolean abc = slide_a + slide_b > slide_c;
        boolean acb = slide_a + slide_c > slide_b;
        boolean bca = slide_b + slide_c > slide_a;

        if(slide_a<=0 ||slide_b<=0 ||slide_c<=0){
            return "invalid";
        }
        if((abc && acb && bca)){
            if(slide_a == slide_b || slide_b == slide_c || slide_c == slide_a){
                if(slide_a == slide_b && slide_b == slide_c){
                return "equilateral";
                }
                return "isosceles";
            }
            return "scalene";
        }
        return "invalid";
```

```
Finished after 0.088 seconds

Runs:  18/18    Errors:  0    Failures:  0

testIfIsosceles_0 (0.001 s)
testIfIsosceles_1 (0.000 s)
testIfIsosceles_2 (0.000 s)
testIfEquilateral (0.000 s)
testIfInvalid_10 (0.000 s)
testIfInvalid_0 (0.000 s)
testIfInvalid_1 (0.000 s)
testIfInvalid_2 (0.000 s)
testIfInvalid_3 (0.000 s)
testIfInvalid_4 (0.000 s)
testIfInvalid_5 (0.000 s)
testIfInvalid_6 (0.000 s)
testIfInvalid_7 (0.000 s)
testIfInvalid_8 (0.000 s)
testIfInvalid_9 (0.000 s)
testIfScalene_0 (0.000 s)
testIfScalene_1 (0.000 s)
testIfScalene_2 (0.000 s)

Failure Trace
```

Figure 1. JUnit Test

In part 3, EclEmma is used to determine the level of code achieve by test cases. At first, the code wasn't covered 100%.



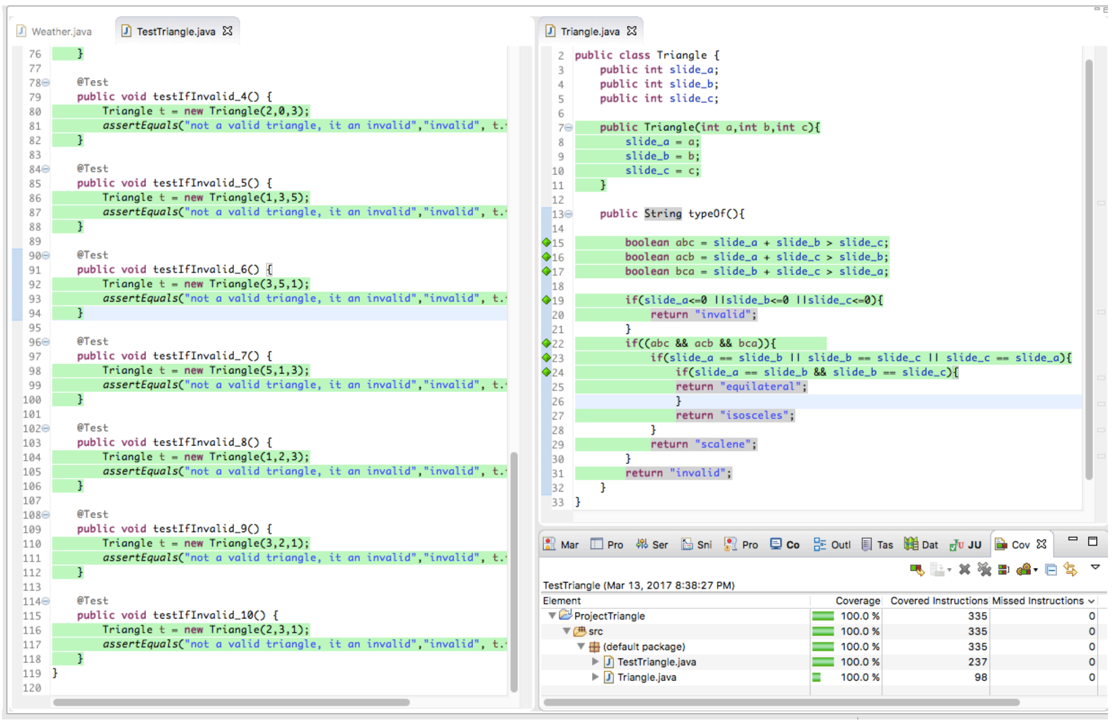
```java
        Triangle t = new Triangle(-2,-2,-2);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
    }

    @Test
    public void testIfInvalid_2() {
        Triangle t = new Triangle(0,2,3);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
    }

    @Test
    public void testIfInvalid_3() {
        Triangle t = new Triangle(2,3,0);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
    }

    @Test
    public void testIfInvalid_4() {
        Triangle t = new Triangle(2,0,3);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
    }

    @Test
    public void testIfInvalid_5() {
        Triangle t = new Triangle(1,3,5);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
    }

    @Test
    public void testIfInvalid_6() {
        Triangle t = new Triangle(3,5,1);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
    }

    @Test
    public void testIfInvalid_7() {
        Triangle t = new Triangle(5,1,3);
        assertEquals("not a valid triangle, it an invalid","invalid", t.typeO
```

```java
public class Triangle {
    public int slide_a;
    public int slide_b;
    public int slide_c;

    public Triangle(int a,int b,int c){
        slide_a = a;
        slide_b = b;
        slide_c = c;
    }

    public String typeOf(){

        boolean abc = slide_a + slide_b > slide_c;
        boolean acb = slide_a + slide_c > slide_b;
        boolean bca = slide_b + slide_c > slide_a;

        boolean ca = slide_c >= slide_a;
        boolean cb = slide_c >= slide_b;
        boolean ba = slide_b >= slide_a;
        boolean bc = slide_b >= slide_c;
        boolean ab = slide_a >= slide_b;
        boolean ac = slide_a >= slide_c;

        if(slide_a<=0 ||slide_b<=0 ||slide_c<=0){
            return "invalid";
        }
        if((abc && ca && cb) || (acb && ba && bc) || (bca && ab && ac)){
            if(slide_a == slide_b || slide_b == slide_c || slide_c == slide_a
                if(slide_a == slide_b && slide_b == slide_c){
                return "equilateral";
                }
                return "isosceles";
            }
            return "scalene";
        }
        return "invalid";
    }
```

```
Element                 Coverage   Covered Instructions  Missed Instructions   Total Instructions
▶ ProjectTriangle       100.0 %                     375                     0                  375
```

Figure 2. Not 100% Code Coverage

After the program was fixed, it was 100% covered.



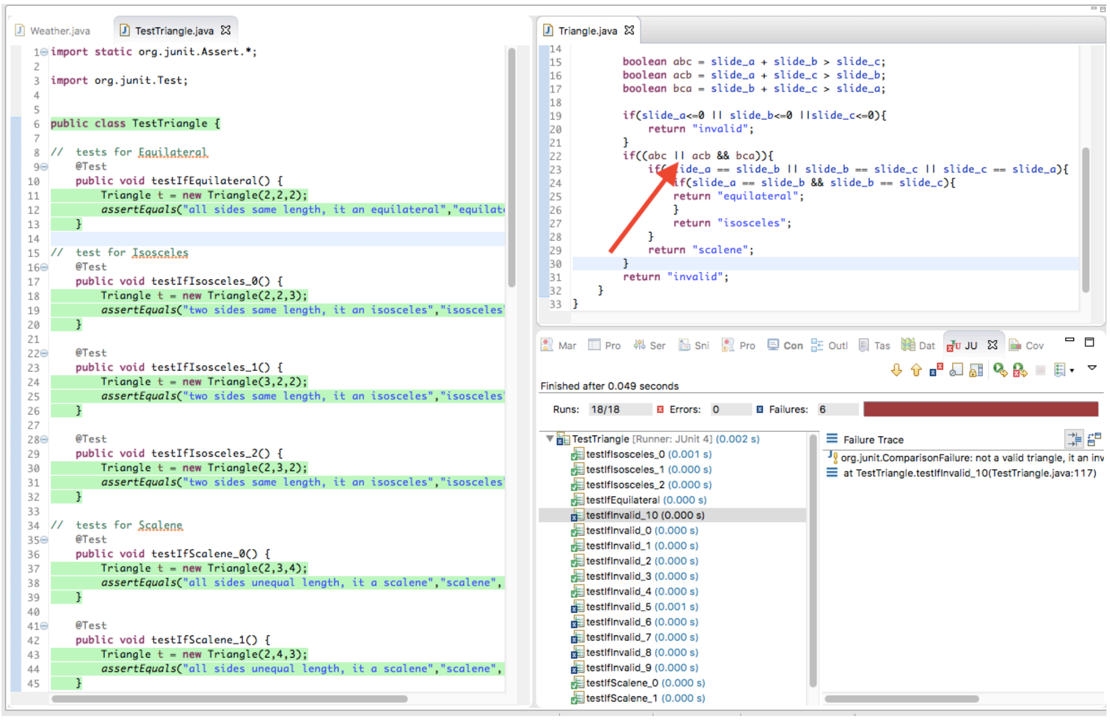
```java
    }

    @Test
    public void testIfInvalid_4() {
        Triangle t = new Triangle(2,0,3);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }

    @Test
    public void testIfInvalid_5() {
        Triangle t = new Triangle(1,3,5);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }

    @Test
    public void testIfInvalid_6() {
        Triangle t = new Triangle(3,5,1);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }

    @Test
    public void testIfInvalid_7() {
        Triangle t = new Triangle(5,1,3);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }

    @Test
    public void testIfInvalid_8() {
        Triangle t = new Triangle(1,2,3);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }

    @Test
    public void testIfInvalid_9() {
        Triangle t = new Triangle(3,2,1);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }

    @Test
    public void testIfInvalid_10() {
        Triangle t = new Triangle(2,3,1);
        assertEquals("not a valid triangle, it an invalid","invalid", t.
    }
}
```

```java
public class Triangle {
    public int slide_a;
    public int slide_b;
    public int slide_c;

    public Triangle(int a,int b,int c){
        slide_a = a;
        slide_b = b;
        slide_c = c;
    }

    public String typeOf(){

        boolean abc = slide_a + slide_b > slide_c;
        boolean acb = slide_a + slide_c > slide_b;
        boolean bca = slide_b + slide_c > slide_a;

        if(slide_a<=0 ||slide_b<=0 ||slide_c<=0){
            return "invalid";
        }
        if((abc && acb && bca)){
            if(slide_a == slide_b || slide_b == slide_c || slide_c == slide_a){
                if(slide_a == slide_b && slide_b == slide_c){
                return "equilateral";
                }
                return "isosceles";
            }
            return "scalene";
        }
        return "invalid";
    }
}
```

```
TestTriangle (Mar 13, 2017 8:38:27 PM)
Element                      Coverage   Covered Instructions  Missed Instructions
▼ ProjectTriangle            100.0 %                     335                    0
  ▼ src                      100.0 %                     335                    0
    ▼ (default package)      100.0 %                     335                    0
      ▶ TestTriangle.java    100.0 %                     237                    0
      ▶ Triangle.java        100.0 %                      98                    0
```

Figure 3. 100% Code Coverage

But in instructor, it is not guarantee even we obtained 100%coverage. So I use the second way provided in instructor, and obtained six test cases failed



Weather.java | TestTriangle.java

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

import org.junit.Test;


public class TestTriangle {

//  tests for Equilateral
    @Test
    public void testIfEquilateral() {
        Triangle t = new Triangle(2,2,2);
        assertEquals("all sides same length, it an equilateral","equilat
    }

//  test for Isosceles
    @Test
    public void testIfIsosceles_0() {
        Triangle t = new Triangle(2,2,3);
        assertEquals("two sides same length, it an isosceles","isosceles'
    }

    @Test
    public void testIfIsosceles_1() {
        Triangle t = new Triangle(3,2,2);
        assertEquals("two sides same length, it an isosceles","isosceles'
    }

    @Test
    public void testIfIsosceles_2() {
        Triangle t = new Triangle(2,3,2);
        assertEquals("two sides same length, it an isosceles","isosceles'
    }

//  tests for Scalene
    @Test
    public void testIfScalene_0() {
        Triangle t = new Triangle(2,3,4);
        assertEquals("all sides unequal length, it a scalene","scalene",
    }

    @Test
    public void testIfScalene_1() {
        Triangle t = new Triangle(2,4,3);
        assertEquals("all sides unequal length, it a scalene","scalene",
    }
```

Triangle.java

```java

        boolean abc = slide_a + slide_b > slide_c;
        boolean acb = slide_a + slide_c > slide_b;
        boolean bca = slide_b + slide_c > slide_a;

        if(slide_a<=0 || slide_b<=0 ||slide_c<=0){
            return "invalid";
        }
        if((abc || acb && bca)){
            if(slide_a == slide_b || slide_b == slide_c || slide_c == slide_a){
                if(slide_a == slide_b && slide_b == slide_c){
                return "equilateral";
                }
                return "isosceles";
            }
            return "scalene";
        }
        return "invalid";
    }
}
```

```
Finished after 0.049 seconds

Runs: 18/18     Errors: 0     Failures: 6

TestTriangle [Runner: JUnit 4] (0.002 s)
    testIfIsosceles_0 (0.001 s)
    testIfIsosceles_1 (0.000 s)
    testIfIsosceles_2 (0.000 s)
    testIfEquilateral (0.000 s)
    testIfInvalid_10 (0.000 s)
    testIfInvalid_0 (0.000 s)
    testIfInvalid_1 (0.000 s)
    testIfInvalid_2 (0.000 s)
    testIfInvalid_3 (0.000 s)
    testIfInvalid_4 (0.000 s)
    testIfInvalid_5 (0.001 s)
    testIfInvalid_6 (0.000 s)
    testIfInvalid_7 (0.000 s)
    testIfInvalid_8 (0.000 s)
    testIfInvalid_9 (0.000 s)
    testIfScalene_0 (0.000 s)
    testIfScalene_1 (0.000 s)

Failure Trace
org.junit.ComparisonFailure: not a valid triangle, it an inv
at TestTriangle.testIfInvalid_10(TestTriangle.java:117)
```

Lesson Learned:

A software test is a piece of software, which executes another piece of software. It validates if that code results in the expected state (state testing) or executes the expected sequence of events(behavior testing). Software unit tests help the developer to verify that the logic of a piece of the program is correct. Running tests automatically helps to identify software regressions introduced by changes in the source code. Having a high test coverage of your code allows you to continue developing features without having to perform lots of manual tests.Unit tests are not suitable fot testing complex user interfave or component interaction.

JUnit is a unit testing framework for Java programming language. JUnit has been important in the development of test-driven development. JUnot tests can be run automatically and they check their own results and provide immediate feedback. A JUnit test is a method contained in a class which is only used for testing. This is called a Test class. To define that a certain method is a test method, annotate it with the @Test annotation.

Reflection:

In this journal, I learned the approach to test codes and determine the coverage by using EclEmma tool. The most important point I have learned in this Lab is the thinking of TDD. It’s not a way as usual that we test the program after we finish the codes but let test drive a better code.

Apart from that, JUnit assumes that all test methods can be executed in an arbitrary order. Well-written test code should not assume any order, for example, tests should bot depend on other tests.

This method executes the code under test. I use an assert method, provided by JUnit or another assert framework, to check an expected result versus the actual result. These method calls are typically called asserts or assert statements.

It would be better if I provide meaningful messages in assert statements. That makes it easer for user to identify and fix the problem. This is especially true of someone else at the problem, who did not write the code user test or test code.

School of Computer Science

Contemporary Software Development

COMP47480

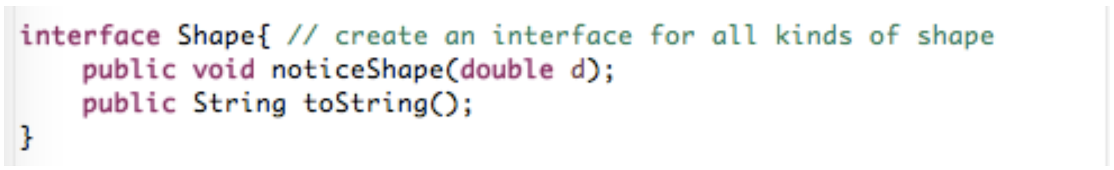
Lab Journal 4

Xiaosheng Liang - 15211913

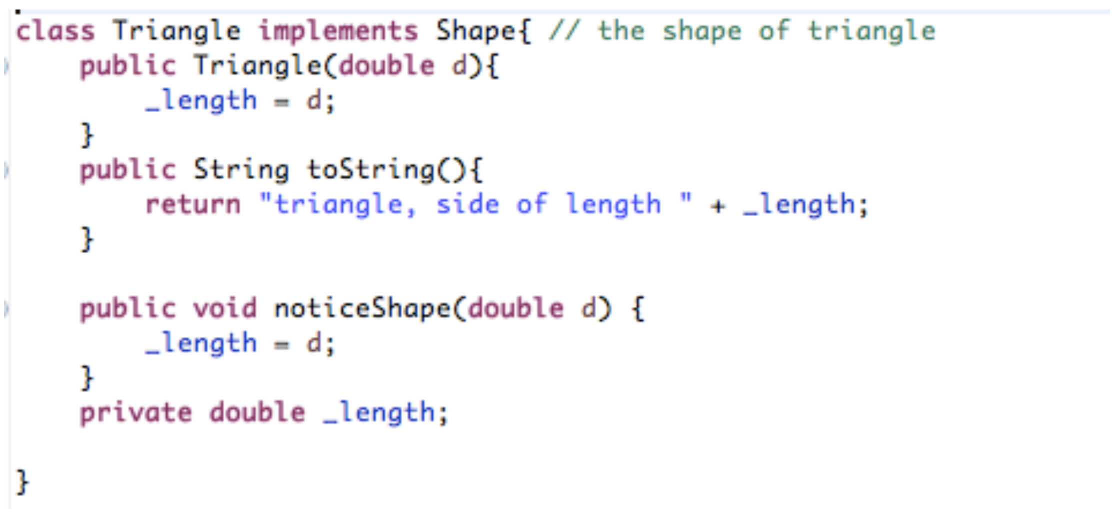
Work Done:

In Lab 4, I learned the principle of Object-Oriented design. Dependency is one of the most important principle which decides if another class has to change when it changes. There are two edicts in dependency rule, one is separate the stable from the unstable, the other is the stable must not depend on the unstable. Another important principle is solid principle which is divided into SRP, OCP, LSP, ISP, DIP, No Concrete Superclasses and The Law of Demeter.

In this practical, all of my group members finished it individually. In open and closed principle, I created an interface for all kinds of shape so that square, triangle or other shapes could implements it directly.

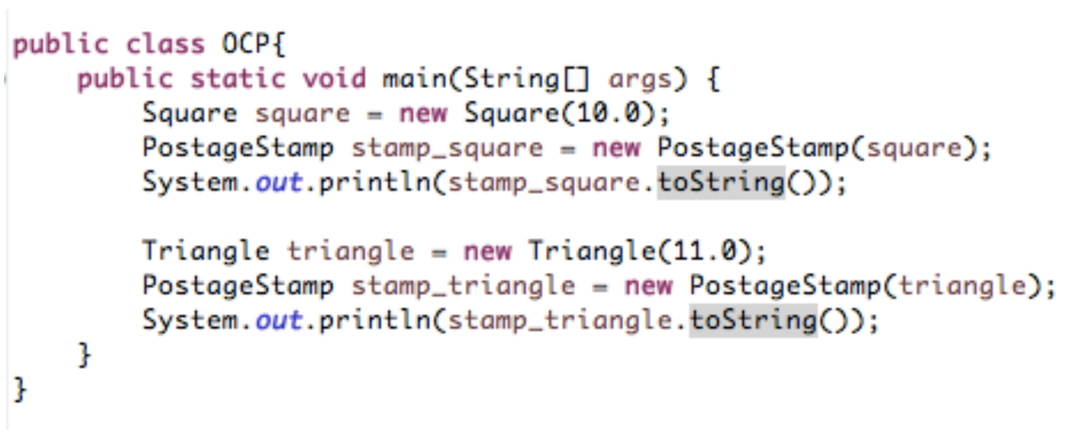


```
interface Shape{ // create an interface for all kinds of shape
    public void noticeShape(double d);
    public String toString();
}
```



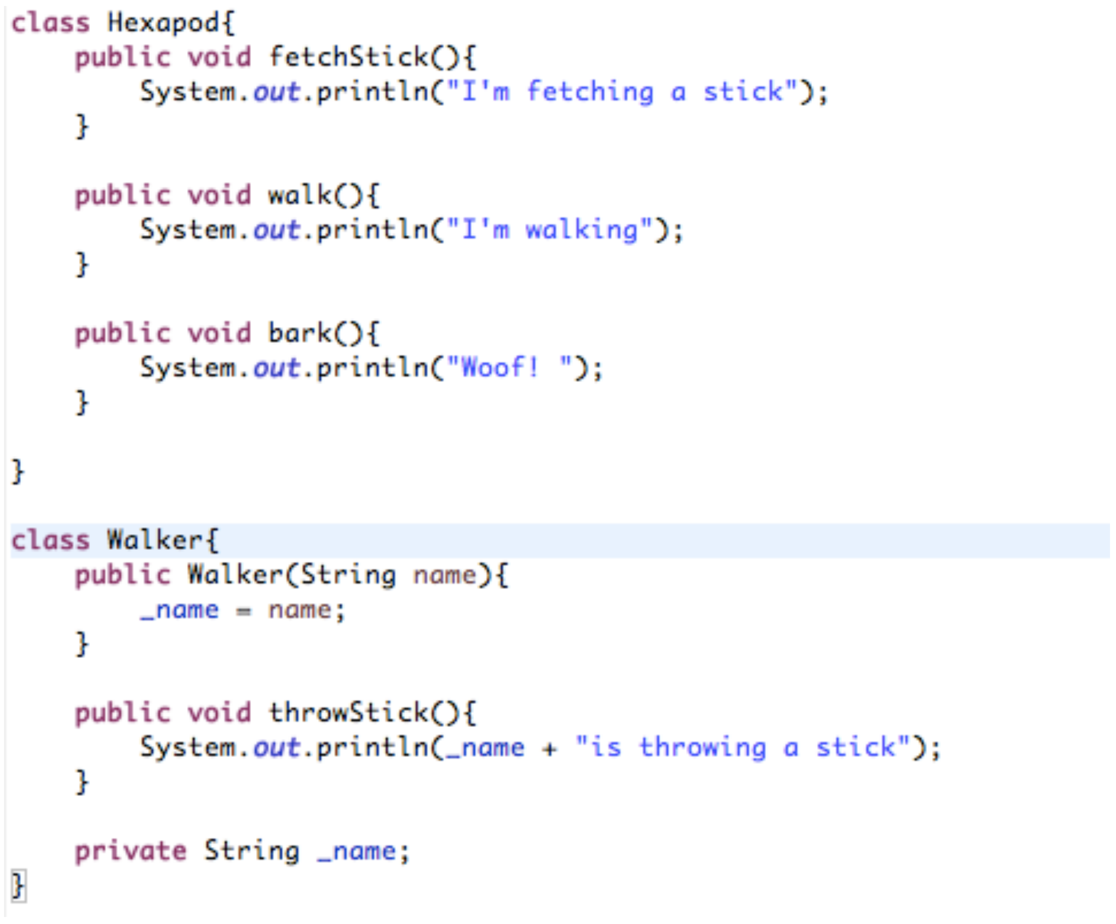
```
class Triangle implements Shape{ // the shape of triangle
    public Triangle(double d){
        _length = d;
    }
    public String toString(){
        return "triangle, side of length " + _length;
    }

    public void noticeShape(double d) {
        _length = d;
    }
    private double _length;

}
```

In Single Responsibility Principe, I divided class Hexapod into two classes. The methods Hexapod and throwStick are moved to the class Walker, since they are invoked by walker not hexapod.



```java
public class OCP{
    public static void main(String[] args) {
        Square square = new Square(10.0);
        PostageStamp stamp_square = new PostageStamp(square);
        System.out.println(stamp_square.toString());

        Triangle triangle = new Triangle(11.0);
        PostageStamp stamp_triangle = new PostageStamp(triangle);
        System.out.println(stamp_triangle.toString());
    }
}
```

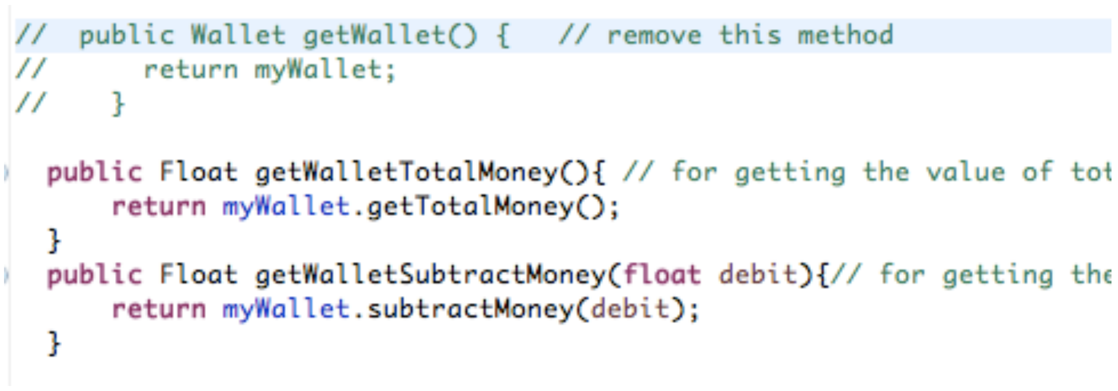
In the main class, we just need to create the instance of different shapes and get a new PostageStamp.



```java
class Hexapod{
    public void fetchStick(){
        System.out.println("I'm fetching a stick");
    }

    public void walk(){
        System.out.println("I'm walking");
    }

    public void bark(){
        System.out.println("Woof! ");
    }

}

class Walker{
    public Walker(String name){
        _name = name;
    }

    public void throwStick(){
        System.out.println(_name + "is throwing a stick");
    }

    private String _name;
}
```

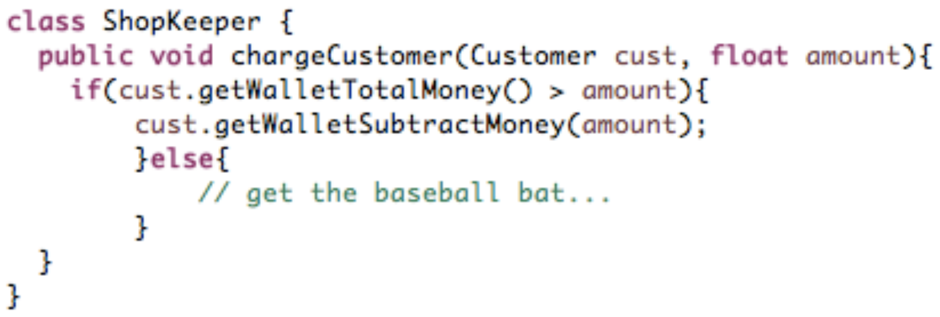
In addition, I added another two classes which are time and place, based on the information provided in problem statement.

In Law of Demeter, since the class ShopKeeper doesn't know about class Wallet, I removed getWallet method in Customer and add another methods to get the value of total money and subtract money in Wallet.



```
//  public Wallet getWallet() {   // remove this method
//      return myWallet;
//    }

  public Float getWalletTotalMoney(){ // for getting the value of tot
      return myWallet.getTotalMoney();
  }
  public Float getWalletSubtractMoney(float debit){// for getting the
      return myWallet.subtractMoney(debit);
  }
```

After fixing Wallet class we don't need to create the instance of class Wallet in ShopKeeper.



```
class ShopKeeper {
  public void chargeCustomer(Customer cust, float amount){
    if(cust.getWalletTotalMoney() > amount){
        cust.getWalletSubtractMoney(amount);
        }else{
            // get the baseball bat...
        }
  }
}
```

Lesson Learned :

Object-oriented analysis and design (OOAD) is a popular technical approach for analyzing, designing an application, system, or business by applying the object-oriented paradigm and visual modeling throughout the development life cycles to foster better stakeholder communication and product quality.

In OOD, the technology-independent concepts in the analysis model are mapped onto implementing classes, constraints are identified, and interfaces are designed, resulting in a model for the solution domain. In a nutshell, a detailed description is constructed specifying how the system is to be built on concrete technologies

Reflection:

In this practical, we practice to apply main principles in our code. For single-responsibility principle, a class is required to have one and only one reason to change, meaning that a class should have only job. As for open-closed principle, objects or entities should be open for extension, but closed for modification. It means that the class should be easily extendable without modifying the class itself.

What I learned from this practical is the conception and implement of single responsibility principle, open/closed principle and the law of Demeter. For single principle, it is a good way of identifying classes during the design phase of an application and good separation of responsibility. But in practice it is sometimes hard to get it right. As for open and close principle, classes, modules and functions should be open for extension but closed for modifications. So this principle is supposed to be applied in those areas which are most likely to be changed. In the law of Demeter, it shows that an object should avoid invoking methods of a member object return by another method. So in Wikipedia, it is called “one dot” rule.

Main principle might seen to be a handful at first, bit with continuous usage and adherence to its guidelines, it becomes a part of you and your code which can sassily be extended, modified, tested and refactored without any problems.

School of Computer Science

Contemporary Software Development

COMP47480

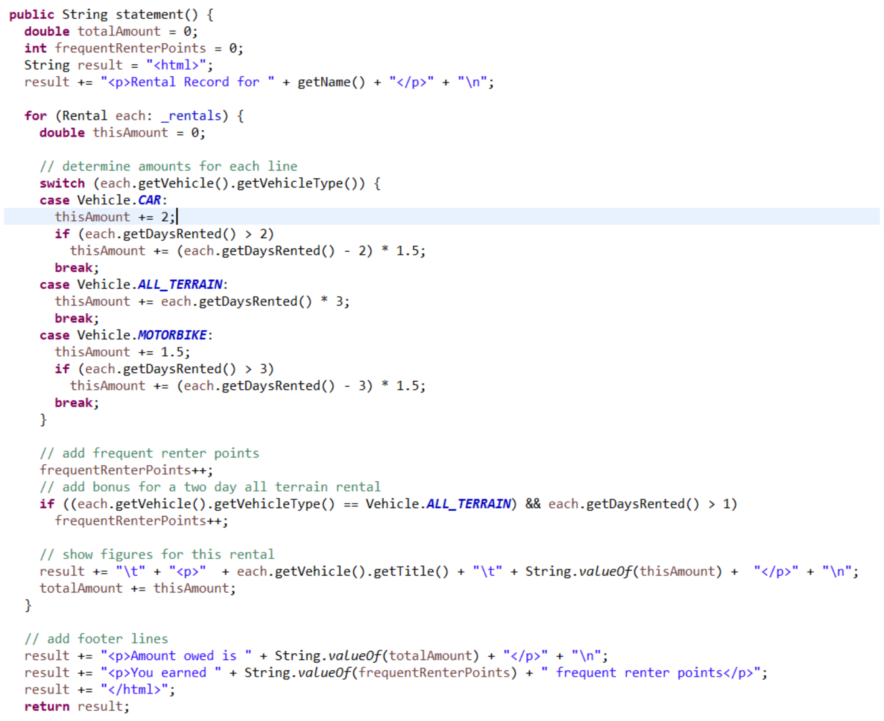
Lab Journal 5

Xiaosheng Liang - 15211913

Work Done:

- Future Evolution:

Changing the HTML output was ambiguous task as to what was actually needed to refactor. Due to this, the changes simply return HTML tags inside the pre-existing strings in the statement() method.



```java
public String statement() {
  double totalAmount = 0;
  int frequentRenterPoints = 0;
  String result = "<html>";
  result += "<p>Rental Record for " + getName() + "</p>" + "\n";

  for (Rental each: _rentals) {
    double thisAmount = 0;

    // determine amounts for each line
    switch (each.getVehicle().getVehicleType()) {
    case Vehicle.CAR:
      thisAmount += 2;
      if (each.getDaysRented() > 2)
        thisAmount += (each.getDaysRented() - 2) * 1.5;
      break;
    case Vehicle.ALL_TERRAIN:
      thisAmount += each.getDaysRented() * 3;
      break;
    case Vehicle.MOTORBIKE:
      thisAmount += 1.5;
      if (each.getDaysRented() > 3)
        thisAmount += (each.getDaysRented() - 3) * 1.5;
      break;
    }

    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a two day all terrain rental
    if ((each.getVehicle().getVehicleType() == Vehicle.ALL_TERRAIN) && each.getDaysRented() > 1)
      frequentRenterPoints++;

    // show figures for this rental
    result += "\t" + "<p>"  + each.getVehicle().getTitle() + "\t" + String.valueOf(thisAmount) +  "</p>" + "\n";
    totalAmount += thisAmount;
  }

  // add footer lines
  result += "<p>Amount owed is " + String.valueOf(totalAmount) + "</p>" + "\n";
  result += "<p>You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points</p>";
  result += "</html>";
  return result;
```

- Code Smells:

Vehicle and Rental both suffer from Data Class, both Vehicle and Rental both consist only of getters and setters, some of which do nearly the same thing (getVehicle() & getVehicleType()). Its methods are just there to hold data and are manipulated in the Consumer class.

To deal with this code smell the refactoring practices you could use Move Method for where these getter and setters are used by other classes to move the behaviour into the data class.

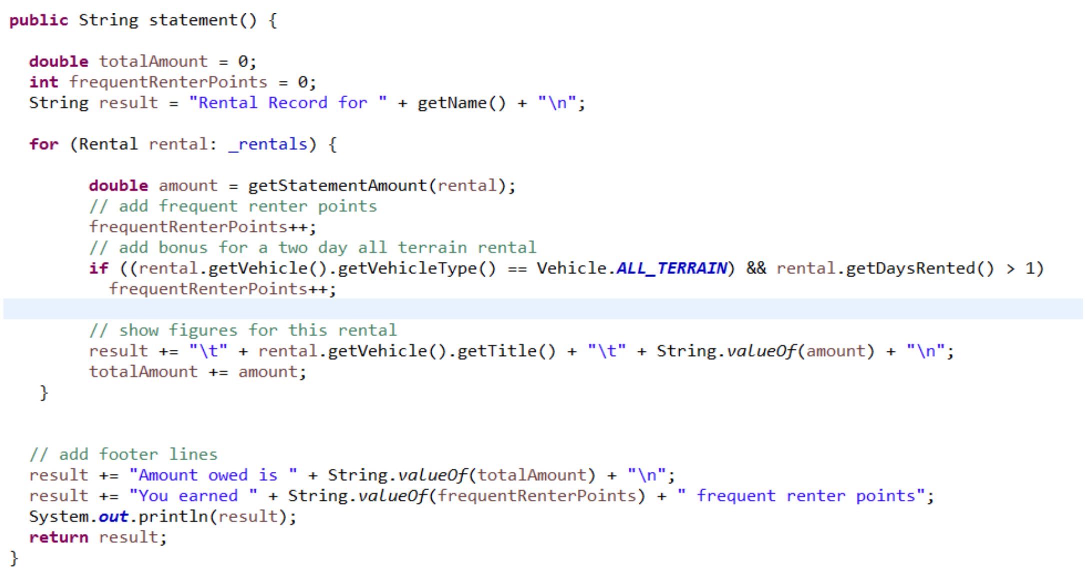
The Customer class suffers from a few different code smells such as long method and feature envy which are both apparent in the statement method. Feature envy also appears in the form of the addRental() method which should be moved into the Rental class using Move Method.

- Splitting up a Long Method: the Extract Method refactoring

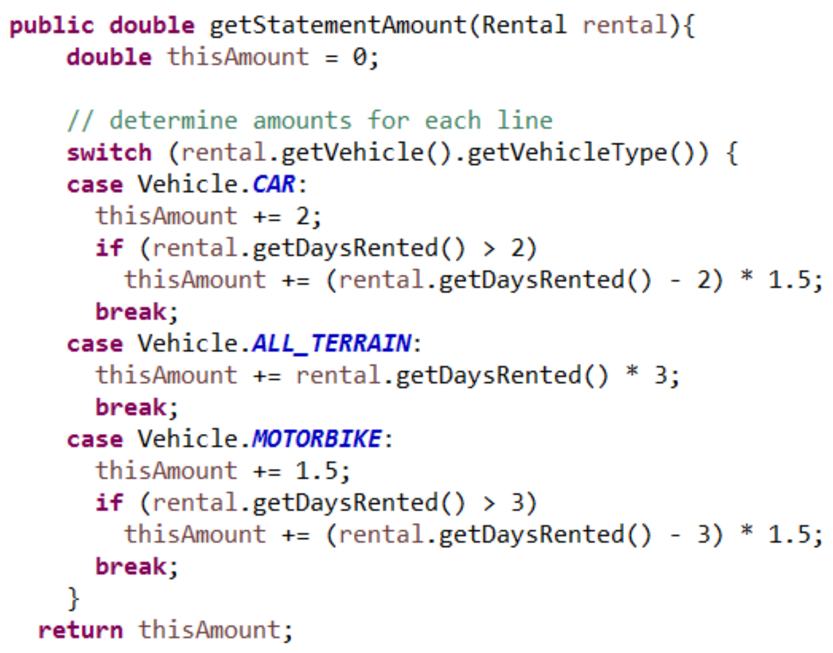
This exercise required fixing a long method violation. The statement is too long and too complex to represent any one functionality, it's doing too many things that it really should not be doing.

To rectify this, the statement method was split into two. A new method called getStatementAmount() was created. This method is responsible for getting the amount for the requested statement, it embodies the entire switch statement that was previously in the statement() method.

After completing the refactoring by hand the results can be seen in the images below:



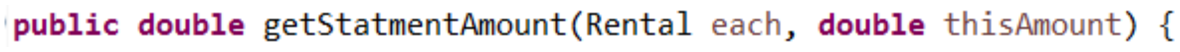
```java
public String statement() {

  double totalAmount = 0;
  int frequentRenterPoints = 0;
  String result = "Rental Record for " + getName() + "\n";

  for (Rental rental: _rentals) {

        double amount = getStatementAmount(rental);
        // add frequent renter points
        frequentRenterPoints++;
        // add bonus for a two day all terrain rental
        if ((rental.getVehicle().getVehicleType() == Vehicle.ALL_TERRAIN) && rental.getDaysRented() > 1)
          frequentRenterPoints++;

        // show figures for this rental
        result += "\t" + rental.getVehicle().getTitle() + "\t" + String.valueOf(amount) + "\n";
        totalAmount += amount;
   }


  // add footer lines
  result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
  result += "You earned " + String.valueOf(frequentRenterPoints) + " frequent renter points";
  System.out.println(result);
  return result;
}
```



```java
public double getStatementAmount(Rental rental){
    double thisAmount = 0;

    // determine amounts for each line
    switch (rental.getVehicle().getVehicleType()) {
    case Vehicle.CAR:
      thisAmount += 2;
      if (rental.getDaysRented() > 2)
        thisAmount += (rental.getDaysRented() - 2) * 1.5;
      break;
    case Vehicle.ALL_TERRAIN:
      thisAmount += rental.getDaysRented() * 3;
      break;
    case Vehicle.MOTORBIKE:
      thisAmount += 1.5;
      if (rental.getDaysRented() > 3)
        thisAmount += (rental.getDaysRented() - 3) * 1.5;
      break;
    }
  return thisAmount;
```

As you can see the refactoring from hand was successful. The getStatementAmount() method was successfully extracted and created a new single responsibility method for returning the amount of the requested statement. The statement() method significantly shortened and now represents more closely to its intended purpose.

When doing the refactoring through eclipse, the extract method insists on an extra parameter for the thisAmount variable, while keeping the variable itself in the statement method.



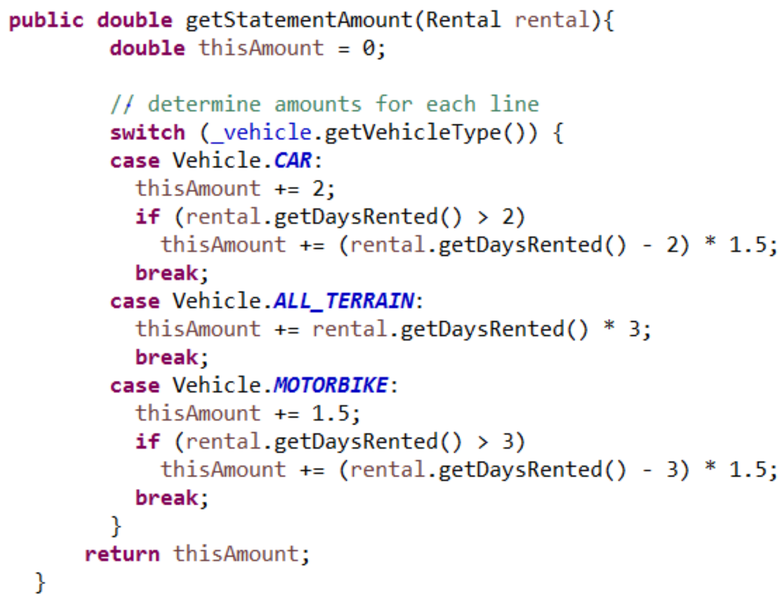
```java
public double getStatmentAmount(Rental each, double thisAmount) {
```

- Fixing poor names with the Rename refactoring

For renaming any inappropriate name, I changed the ‘each’ which represents each individual rental to ‘rental’. Also, in the Vehicle class, the setVehicleType method take a int called args, this was changed to the more appropriate name vehicleTypeNumber

- Feature Envy! Use Move Method to get the method to it rightful home

The new method, getStatementAmount() method, suffers from feature envy of the Rental class. Firstly, by hand, the method was moved in its entirety into the Rental class. This also meant that some of the method calls in getStatementAmount() had to be changed to remove the feature envy.



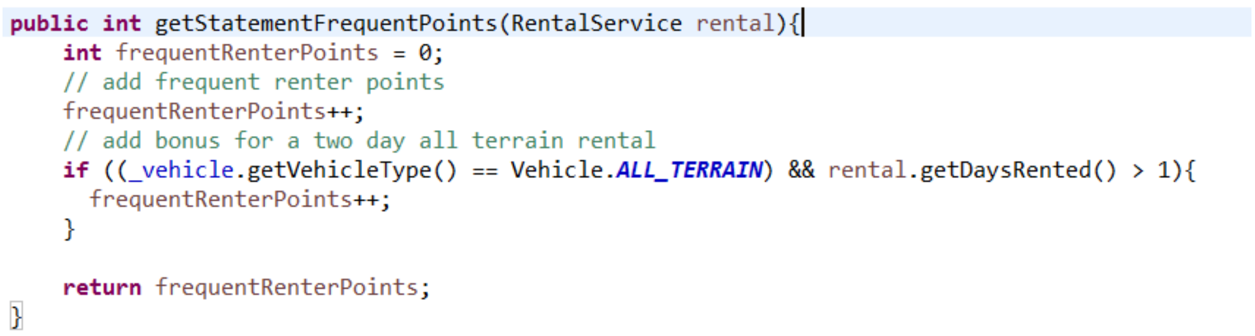
```java
public double getStatementAmount(Rental rental){
        double thisAmount = 0;

        // determine amounts for each line
        switch (_vehicle.getVehicleType()) {
        case Vehicle.CAR:
          thisAmount += 2;
          if (rental.getDaysRented() > 2)
            thisAmount += (rental.getDaysRented() - 2) * 1.5;
          break;
        case Vehicle.ALL_TERRAIN:
          thisAmount += rental.getDaysRented() * 3;
          break;
        case Vehicle.MOTORBIKE:
          thisAmount += 1.5;
          if (rental.getDaysRented() > 3)
            thisAmount += (rental.getDaysRented() - 3) * 1.5;
          break;
        }
      return thisAmount;
  }
```

Changes also had to be made in the Customer class with reference to getStatementAmount(), now having to be instantiated with the Rental class. The Rental class was also Renamed with the eclipse rename tool to RentalService, which gives a much overview of the functionality, as the class provides more of a service than anything else.

When this refactoring is done through the eclipse tools 'Move Method' function it doesn't give as quite a good result. The main issue with using Move Method is that it didn't keep the parameter initially given to it, thus, leaving a blank parameter list and an error to anywhere the method was being called. I preferred the none delegated version of this refactoring , largely due to the fact with the delegate option adds an additional method to make a call to the moved method, thus cluttering the class with an extra method it doesn't need.

- Now Refactor Mercilessly (don't worry if you don't get to this)

Due to time constraints, not much more refactoring was done to the code. One refactor we did make was adding the functionality for adding frequentPoints to the Rental class, which involved creating a method getStatementFrequentPoints() which I called by statement() method to return the points for the requested statement, the feature previously seen in this functionality was also removed due to it being in its rightful place.



```
public int getStatementFrequentPoints(RentalService rental){
    int frequentRenterPoints = 0;
    // add frequent renter points
    frequentRenterPoints++;
    // add bonus for a two day all terrain rental
    if ((_vehicle.getVehicleType() == Vehicle.ALL_TERRAIN) && rental.getDaysRented() > 1){
      frequentRenterPoints++;
    }

    return frequentRenterPoints;
}
```

Lessons learned:

Refactoring is a controllable process of improving code without creating new functionality. Refactoring transforms a mess into clean code and simple design.

Refactoring is a controlled technique for improving the design of an existing code base. Its essence is applying a series of small behavior-preserving transformations, each of which "too small to be worth doing".

The key insight is that it's easier to rearrange the code correctly if you don't simultaneously try to change its functionality. The secondary insight is that it's easier to change functionality when you have clean (refactored) code.

Refactoring is a kind of reorganization. Technically, it comes from mathematics when you factor an expression into an equivalence - the factors are cleaner ways of expressing the same statement. Refactoring implies equivalence; the beginning and end products must be functionally identical. You can view refactoring as a special case of reworking

Reflection:

After finished the lab, I understand Code Refactoring is the process of clarifying and simplifying the design of existing code, without changing its behavior. Agile teams are maintaining and extending their code a lot from iteration to iteration, and without continuous refactoring, this is hard to do. This is because un-refactored code tends to rot.

Every time I change code without refactoring it, rot worsens and spreads. Code rot frustrates us, costs us time, and unduly shortens the lifespan of useful systems. In an agile context, it can mean the difference between meeting or not meeting an iteration deadline.
Refactoring code ruthlessly prevents rot, keeping the code easy to maintain and extend. This extensibility is the reason to refactor and the measure of its success.

Practically, refactoring means making code clearer and cleaner and simpler and elegant. Or, in other words, clean up after yourself when you code. Refactoring is **not** rewriting, as I thought before. Refactoring is a good thing because complex expressions are typically built from simpler, more grokable components. Refactoring either exposes those simpler components or reduces them to the more efficient complex expression.

School of Computer Science

Contemporary Software Development

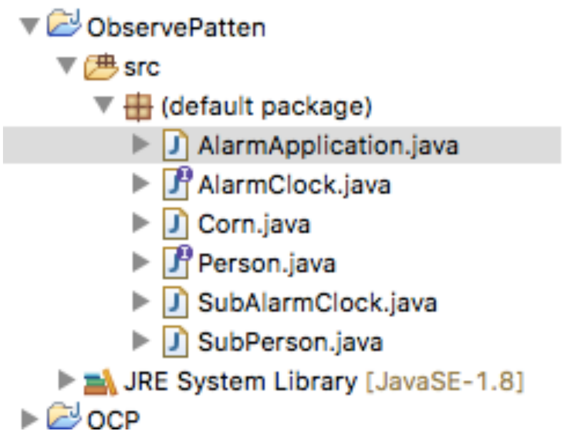
COMP47480

Lab Journal 6

Xiaosheng Liang - 15211913

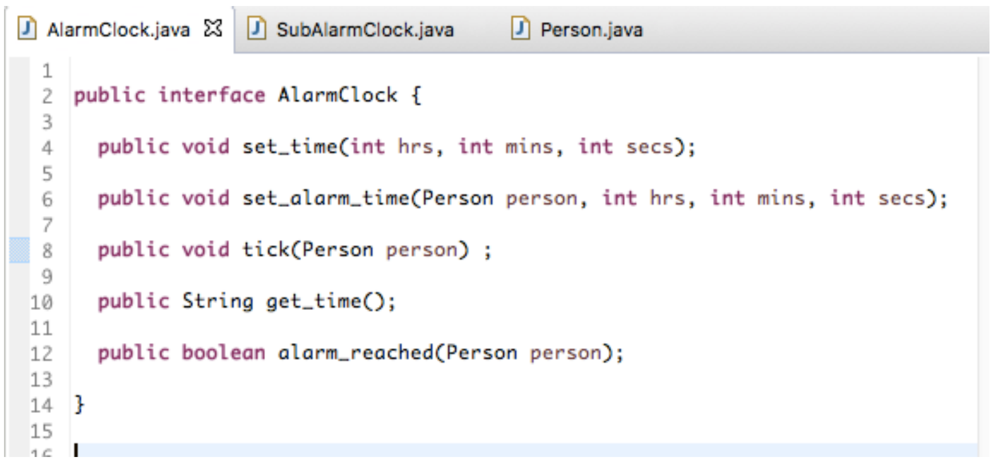
Work Done:

In this practical, we finished individually. Follow the guide, first I created SubAlarmClock and SubPerson, as two subclasses for Person and AlarmClock.



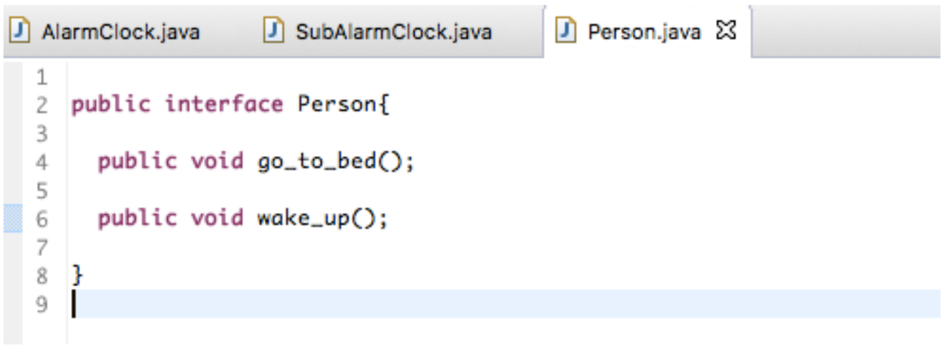
```
ObservePatten
  src
    (default package)
      AlarmApplication.java
      AlarmClock.java
      Corn.java
      Person.java
      SubAlarmClock.java
      SubPerson.java
  JRE System Library [JavaSE-1.8]
OCP
```

Change Person and AlarmClock classed into Interfaces for those subclasses.



AlarmClock.java SubAlarmClock.java Person.java

```java

public interface AlarmClock {

  public void set_time(int hrs, int mins, int secs);

  public void set_alarm_time(Person person, int hrs, int mins, int secs);

  public void tick(Person person) ;

  public String get_time();

  public boolean alarm_reached(Person person);

}

```



AlarmClock.java SubAlarmClock.java Person.java

```java

public interface Person{

  public void go_to_bed();

  public void wake_up();

}

```

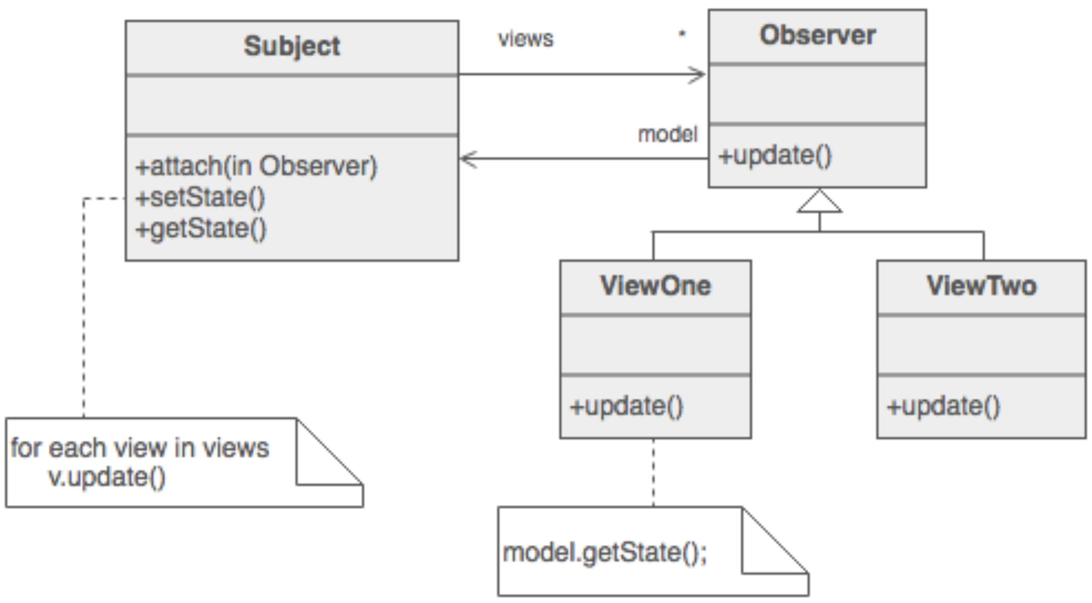
Create a list of Person into SubAlarmClock and add instance of person into method of SubAlarmClock. Apart from that, add method of Person (wake_up and go_to_bed) into method of SubAlarmClock respectively to make sure that, set the alarm when go to bed and wake up when alarm reached.

```
public void set_alarm_time(Person person, int hrs, int mins, int secs){
  alarm_hours=hrs; alarm_minutes=mins; alarm_seconds=secs;
  personlist.add(person);
  person.go_to_bed();
}
```

Lessons learned:

Observer design pattern is defining a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. Encapsulate the core (or common or engine) components in a Subject abstraction, and the variable (or optional or user interface) components in an Observer hierarchy. And it's the "View" part of Model-View-Controller.

Here is the structure of bserver design pattern:



The Observer design pattern lets several observer objects be notified when a subject object is changed in some way. Each observer registers with the subject, and when a change occurs, the subject notifies them all. Each of the observers is notified in parallel

Observer pattern uses three actor classes. Subject, Observer and Client. Subject is an object having methods to attach and detach observers to a client object. We have created an abstract class Observer and a concrete class Subject that is extending class Observer.

Reflection:

The Observer Pattern is an appropriate design pattern to apply in any situation where we have several objects which are dependent on another object and are required to perform an action when the state of that object changes, or an object needs to notify others without knowing who they are or how many there are.

An object with a one-to-many relationship with other objects who are interested in its state is called the subject or publisher. Its dependent objects are called observers or subscribers. The observers are notified whenever the state of the subject changes and can act accordingly. The subject can have any number of dependent observers which it notifies, and any number of observers can subscribe to the subject to receive such notifications.

In this lab, I have learned you the basic Subject-Observer pattern and praltised concrete examples of how you can use this behavioral pattern to easily extend the functionality of an existing class without tightly coupling any new requirements. This pattern enables us to achieve a higher level of consistency between related and dependent objects without sacrificing code re-usability.

School of Computer Science

Contemporary Software Development

COMP47480

Seminar Journal 1

Xiaosheng Liang - 15211913

Demonware

In the beginning of the seminar, the speaker introduced Demonware, whose productions enable games publishers to outsource their networking requirements, allowing them to concentrate on playability and allowing hundreds of thousands of games player around the world play against with each other simultaneously.

The speaker started from introduced the key components of the system. The system consists of the following parts:

- C++: For building the platform of system (engine).
- Erlang: For building the connection, which connects distributed databases through Message after data clustering. It is for distributed system.
- Django: For creating the interface of the application. (MVC framework)
- MySQL: Database management system. Managing all the data of the system.
- Python: Since it run fast, python is used to respond the action between distributed databases.

The State Engine is a high-performance state synchronization C++ programming framework.
In the Django, the MVC frame work of this system. Databased is managed by MySQL and controller is implemented by Erlang.

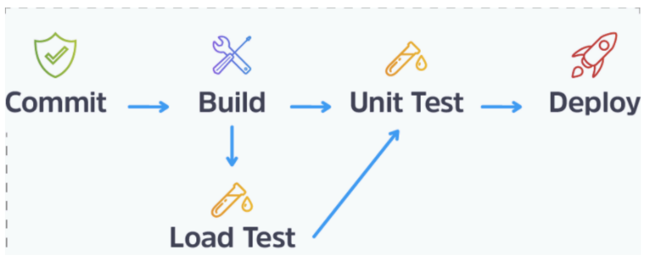
As for Erlang, this is my first time I heard this language. So I search some information about it. Erlang is a programming language which is well suited for systems which are distributed, soft real-time, non-stop systems. These characters are just the characters of Demonware system.
As far as I know about Django, it's a python framework which is similar to MVC architecture.

After that, the speaker mentioned there are two mechanisms for database protection:

- Replication: In database, once datasets set up, the system would replicate it to SQL master, then the data would be copied. As for the masters, they are not just one, so all the datasets will be copied to all the masters. If one of the master died, the system can change to another immediately in case that the system crash.
- Sharing: Distributing all the data for all the databases. It's a kind of multiple management. For example, for dataset from A to Z, the first distribution stores the data from A to L, the second distribution stores L to R, and the last distribution store the data from R to Z. Then, build the connection to all the user.
-

As far as I know from other materials, this kind of mechanism has been used in lots of productions. Take “MIMIX Share” as a example, it provides easy, automated data replication between databases and real-time data sharing which can improve efficiency by keeping your databases in sync.

For testing, the speaker said there are kinds of tests for system, such as load test, unit test and integration test. The load test is a process of putting demand on the games and estimate their performances. It’s performed to determine a game’s behavior under both normal and anticipated peak load conditions. Unit test and integration test are testing individual units of modules. And combining individual units and modules, then testing them as groups, which is integration test. Usually, load test is before unit test and integration test.



As for teamwork, Demonware has two branch offices in Canada and China respectively. So how to cooperate with distributed teams across the world used to be a problem to their company. The solution is each team has its own data and server so that they can make the decisions by their own. Once one of the team blocked, the system would split it up immediately by using multiple controlling as mentioned in the databases management.

School of Computer Science

Contemporary Software Development

COMP47480

Seminar Journal 2

Xiaosheng Liang - 15211913

Dimension data

We were given a presentation on Tuesday about company Dimension Data. First the speaker introducing the their company. Dimension Data is a company specializing in information technology services. Dimension Data focused on services including network integration, security and data centers. Dimension Data, as a global cloud platform, manages the database in cloud platform and make data get access to the network. After that, Dimension Data could use the power of technology to help organizations achieve great things in the digital area.

As for agile development, she mentioned some key points of it. Plan, developer should have a plan and design before they develop a software. Then a software should be throw into a technical practices. The culture of company is very important for all the staffs. And quality and quality and knowledge creation.

Agile development is to make sure fail faster, sooner and cheaper. Agile approaches help teams respond to unpredictability through incremental, iterative work and empirical feedback. For agile development in Dimension Data, they need two weeks to sprints followed. After codes come out from sprints, they go into cloud control center. System is located higher than higher than cloud control center, each team should be responsible for the system. In addition, team for 5-6 persons would be perfect in agile development. There would be a lot of communication between teams so that they can work together better.

Software methodology

AGILE (SCRUM)
Presenter's thoughts: Agile does not say how to do it, it just encompasses some values, and we are free to do the way we want. It does not guarantee success; all it does is fail quicker so that loss is minimized.
2 week sprints followed, and backlogs are resolved in parallel. Daily stand-ups happen.
After code comes out of sprints, it goes into a cloud control layer and later into production.
TEAM SIZE: 5-6 preferred (Scrum masters collaborate between teams).
QA team comes after engineering team.
Positives of Agile: Recycle, rebalance, re-adjust, easy to move people between teams, mix people.
Elevation week: developers stop coding and involve in discussing out-of-the-box ideas.

For Technical tools:

Java, Spring, intelliJ IDEA (Java IDE for professional developer) MySQL/Hibernate(Database Management), Linux, Apache Tomcat(for web apps)

As for Refactoring, good code is that human can understand when it is updated.

Build Tools:

Maven, Git (started using before 6 months. Before, subversion was used. But it had problems with code merge and lots of manual work), Jenkins (to integrate systems), JUnit (automated tests), NexusOSS (for deployment – an open-source repository).
For Build tools n Maven: It is software tool that helps manage java project structure. And automate application builds. In intelliJ IDEA, we memtioned before, it fully integrates with Maven version 2.2 and later versions, allowing developers to create or import Maven modules, download artifacts and perform the goals of the build lifecycle and plugins.

Jenkins: It is an server-based system written in Java to and running in Apache Tomcat in Dimension Data system. The basic functionality of Jenkins is to execute a predefined list of steps. The trigger for this execution can be time or event based. For example, every 20 minutes or after a new commit in a Git repository.

JUnit : It is a unit testing framework got Java development and it's used for test-driven development. With IDE integrated support for the JUnit testing framework enables developers to test quickly.

Nexus OSS: offer system version n Git: It doesn't do well when code merges.

When build fails, Jenkins gives a UI output that's easy to read. It is also designed to send emails to the commit owners.
Inter-dependencies: one project is dependent on latest commit of other project. For example, messaging library is common between projects and so, all projects have to be updated so that library works well.
Changing VCS (version control system) from subversion to GIT was tough and very important.

School of Computer Science

Contemporary Software Development

COMP47480

Seminar Journal 3

Xiaosheng Liang - 15211913

OpenJaw

About OpenJaw:

OpenJaw is a company which develops the software for travel industry. It transforms travel companies into travel retailers and enables them to sell other travel products, including hotel rooms and rental cars. And its platform, t-Retail, is the most complete travel retailing platform available. It provides everything that a large travel company needs to create the best shopping experiences for their customers and derive maximum benefit for their business.

Software methodology

OpenJaw creates t-retail, a powerful technology platform which gives airlines, OTAs and loyalty brands everything they need to retail all travel products to increase revenue and enable redemption.

Openjaw uses agile and "waterfall" development cycle. There are 4-5 people in each team and they use GitLab to keep continuous integration. It a web-based Git repository manager and issue tracking features, using an open source license, developed by GitLab. the demonstrator mentioned, all the teams have to be one and the same or at least work together all the time to reduce overhead.

Modeling

The are kinds of tools they use when the software is developed, such as GitLab, SonarQube, Veracode, Slack, Selenium and Manual. SonarQube is for code analysis, Veracode is a securith static analysis tool and Slack is for team communication. Selenium and Manual is for quality assurance. But the speaker said they don't use UML.

OpenJaw uses Amazon Web Services (AWS) to collaborate for continuous integration because there are around 23 individual projects that need to be merged often. The QA testers review the test cases and consolidate them into a single script which runs overnight, for e.g., in booking engine. They also use Swagger, an API tool, Spring Boot for Dependency injection and JaCoCo for code coverage.

Testing

In the old platform of OpenJaw, the developer always have to face the difficulty of unit testing. For solving this issue, OpenJaw has been moved to REST-ful API and developed in latest software tools. The QA testers review the test cases and consolidate them into a single script which runs overnight

OpenJaw uses Amazon Web Services (AWS) to collaborate for continuous integration because there are around 23 individual projects that need to be merged often. The QA testers review the test cases and consolidate them into a single script which runs overnight, for e.g., in booking engine. They also use Swagger, an API tool, Spring Boot for Dependency injection and JaCoCo for code coverage.