CS31310 Agile Methodologies • Worksheet 2

Test Driven Development and Pair Programming

Introduction

This exercise gives you the opportunity to try Test Driven Development and Pair Programming. You will work on short programming exercise to develop a few stories using Test Driven Development. This example is based on an example from ThoughtWorks.

Steps

To complete this worksheet, follow the steps listed below.

1. Turn up to your allocated Worksheet session and participate in the TDD/Pair Programming exercise.
2. Following the exercise, write a short reflection on the exercise. Write between half and 1 page of A4 using an 11pt font. In your reflection, consider the following points:
    - How did you interact with the people that you worked with? Was the interaction helpful? Explain why you thought it was or wasn't helpful.
    - Comment on the use of the 'red-green-refactor' approach during the exercise. Was this a help or a hindrance to developing the code? Briefly discuss your thoughts on this.
    - Any other points that you would like to raise about the exercise.
3. Create a PDF version of your document and submit your document to Blackboard. This should be submitted by **3pm Monday 16th December**.

Marking

The worksheet is worth 3% of the module. You are being assessed on whether you attend the session to try TDD and Pair Programming and then write and submit a sensible reflection.

It is suggested that you only spend up to 30 minutes writing the reflection. Your reflection will only be marked if you have attended a session.

Questions?

If you have any questions about this worksheet, contact Neil Taylor (nst@aber.ac.uk).

Some Java & JUnit Notes

The following notes are intended as a brief reminder of aspects of Java.

Class structure

```
public class MyClassName {

   private int aValue;

   public void aMethodThatDoesNotReturnAnything(int number) {
       aValue = aValue + number;
   }

   public int multiplyValueByTen(int originalValue) {
      return originalValue * 10;
   }
}
```

Enum

Enumerations are used to have a named set of values. In Java, they can be more sophisticated in that the values can have more than one value and there can be methods in the enumeration. One possible implementation for Fruit could be achieved using Java enumeration. You don't have to do that, but an example of this approach is shown below.

```
public enum Fruit {
 apple(2),
      banana(1);

    public int pricePerItem;

    Fruit(int pricePerItem) {
        this.pricePerItem = pricePerItem;
    }
}
```

In the code, you see that *apple(2)* indicates that there is a value for *apple* and it is associated with the value 2. The methods give us a clue that the value represents the price of the item. *apple(2)* is actually calling the method Fruit(int pricePerItem). You don't call the Fruit method directly – it is implicitly called when the enumeration is initialized.

```
Fruit aFruit = Fruit.apple;
int price = aFruit.pricePerItem;
```

Statements

Typical statements that you might use in this exercise.

Using an if statement

```
if(someValue == false && otherValue >= 50) {
   // some statements
}
else if(otherValue >= 30) {
   // some statements
}
else {
   // some statements
}
```

Using a Switch Statement

```
int aValue = 50;

switch(aValue) {
   case 10:
      System.out.println("10");
      break;
   default:
      System.out.println("other value");
      break;
}
```

Iteration using a while loop.

```
while(someValue == true) {
   // code
}
```

Iterating using a counter.

```
for(int i = 0; i < 10; i++) {
   // code
}
```

Iterating over lists or other enumerable types.

```
ArrayList<Person> list = new ArrayList<Person>();

for(Person value : list) {
   // code
}
```

JUnit Notes

The key points are:

1. Setup the fixture (the initial state)
2. Exercise the code in some way, e.g. calling buyItem in the following example.
3. Make some assertions, e.g. assertNotNull()

The following example is in this exercise.

```
public class TillTest {

    @Test
    public void shouldNotFail() {
        // given I go to the till
        Till till = new Till();

        // when I buy two bananas
        Receipt done = till.buyItem(2, Fruit.banana);

        // then I should get a receipt
        assertNotNull(done);
    }
}
```

Example assert statements are:

```
assertTrue("Some comment to be shown if the test fails", <Boolean statement>)
```
e.g. `assertTrue("Value should be greater than 10", aValue > 10);`

```
assertEquals("Some comment to be shown if the test fails", <expected>, <actual>)
```
e.g. `assertEquals("Value should be 12", 12, value);`

```
assertNull("Some comment to be shown if the test fails", <reference value>)
```
e.g. `assertNull("Value isn't null", aStringValue);`

Document History

1.0	16/11/2012	Neil Taylor	Initial version.
1.1	19/11/2012	Neil Taylor	Corrected submission date.
1.2	03/12/2012	Neil Taylor	Updated for 2013.