CSIS0403/COMP3403
Implementation, Testing, and Maintenance of Software Systems
Department of Computer Science
The University of Hong Kong

Assignment 2: White Box Testing in Practice

Description

This assignment features topics we've just covered in white-box testing and unit testing. It is split into two parts. The first, outlined here, gives experience in creating a set of white-box test cases to test a simple method of a class. In the second you'll build mock objects to support testing, execute the tests in *JUnit* and collect coverage statistics. The coverage tool we'll use is a plug-in for *Eclipse* named *Clover*. There will be a tutorial on JUnit and Clover in the last hour of our final class.

Details for Part 1

The IUT is adapted from past assignment work in the OO Programming and Java course, COMP2396/CSIS0396. The purpose of the assignment was to explore dynamic dispatch (dynamic binding) and it also contains a simple implementation of the double dispatch pattern (you will see this in the check for compatibility of ingredients). Your goal is to perform white box testing on the *makeOrder()* method of the *Order* class. While exploring the code in Part 1, you can begin planning for the implementation of unit testing that you will perform in Part 2. During Part 1 you will also perform a small restructuring to support the use of mock objects in testing.

You are supplied with a total of 9 files. That is for your convenience, to allow you to run the program and observe expected behaviour. HOWEVER, you will test the *makeOrder()* method of the *Order* class in isolation, and you must develop mock objects to support that form of testing. You must not simply copy the actual implementations of the ingredient classes to form mock objects since that would be missing the point of unit testing. In fact, you will find you do not need to implement mock versions of all types of ingredient in order to perform testing. For testing, assume you have access ONLY to the *Order* class and the *Ingredient* interface.

Tasks

I. Preparation

1) Unzip the "Assignment2.zip" file and create a new Java project in *Eclipse*.
2) Run the program by instantiating the *Order* class and invoking *makeOrder()* .
   For ease of testing, the constructor of *Order* is parametrized with a *Scanner* object for user input, and a *PrintStream* object for output. To run the application interactively, you can pass as arguments: a scanner object that scans the standard input stream, and a standard output stream object. In scripted unit tests, you can replace these, for example, with a scanner that reads from a String or file, and a PrintStream-decorated ByteArrayOutputStream.

   Thus, to run interactively requires only the following:

```
//...
   Order order = new Order(new Scanner(System.in), System.out);
   order.makeOrder();
//...
```

3) Study and understand the implementation of the class *Order*.
4) Observe that *makeOrder()* contains several direct instantiations of *Ingredient* objects of all types. To support the later use of mock objects, modify the code to use an *IngredientFactory* object (as in the example discussed in class) and replace the direct instantiations with calls to creation methods of the factory. We could pass the factory object as a constructor argument. However, since ingredient objects are instantiated only in *makeOrder()*, it is convenient to pass the factory as an argument of that method. Hence, the header of the method will become similar to:

```
//...
   public void makeOrder(IngredientFactory ingredientFactory) {
//...
```

II. Design your test cases

5) Draw the flow graph of the *makeOrder()* method.
6) Calculate the cyclomatic complexity of the method.
7) Apply the Baseline Method to derive a basis set of independent paths and hence test cases for the method.
8) Determine the conditions needed to drive the execution of each executable path in the basis set and present them as a table of test case inputs.

Note: in this assignment you are not concerned about the quality of the code provided. Your only interest is in the behaviour of the method and to cover it completely with white box test cases.

Deliverables

1. A report containing the following items for the *makeOrder()* method of the class *Order*:

a) Flow graph (with the nodes numbered).
b) Source code of the *makeOrder()* method, modified to use an IngredientFactory object, and with statements labelled with the corresponding node numbers from your flow graph (you can write the numbers directly on the code listing and submit it as an image).
c) Cyclomatic complexity of the method.
d) A table of your basis set of independent paths (describe each path by listing the nodes traversed using the numbering from your flow graph).
e) Table of concrete test cases.

Deadline

Part 1 does not contain any time-consuming work so let's set the deadline as:
Monday, May 2: 23:55. Please let us know if this is too tight for you.