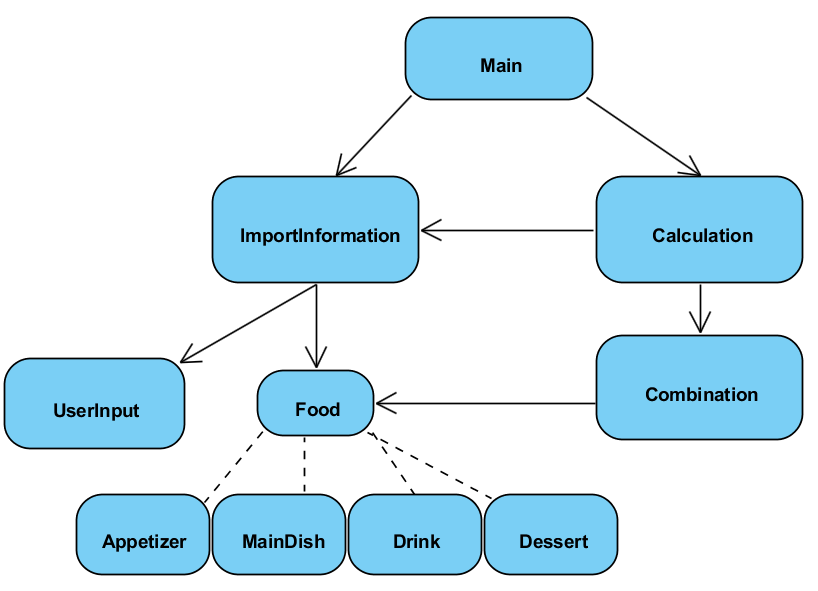
Test report – Release 2.0

1. Dependency diagram
2. Testing method
3. Test plan
4. Unit test
5. Integration test
6. System test
7. Branch coverage
8. C/DC coverage
9. Loop coverage

Dependency diagram

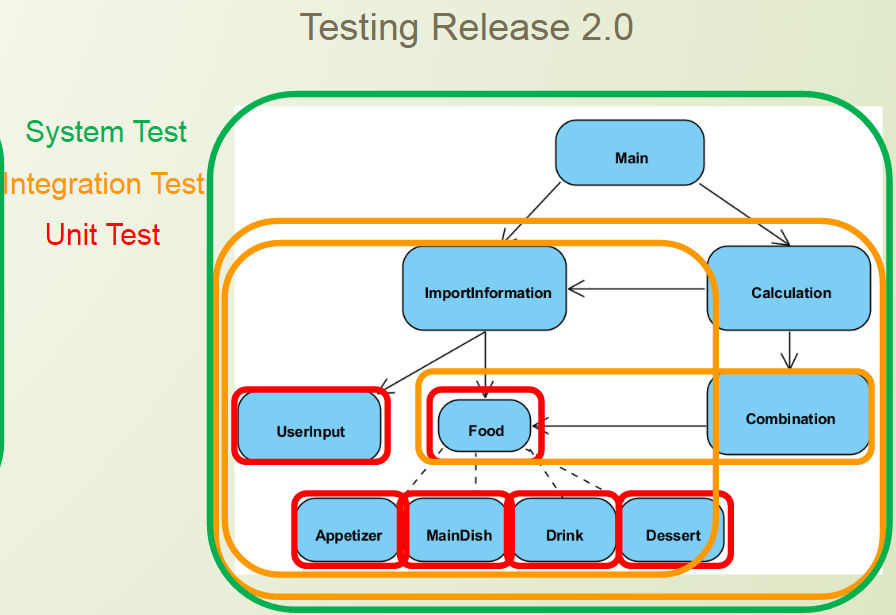


This diagram show the dependency relation between each classes and the general execution flow of the program. Although the number of class had been increased from 7 to 10 after performing debugging and refactoring process in the release 1.0 of the program, the main algorithm of calculating combination and importing user input remain mostly the same. The main class which control the main execution flow call the functions from two classes namely “ImportInformation” which help to import the menu and users input and “Calcualtion” which is mainly responsible for calculating valid food combination based on the imported user’s preferences and the imported menu. However, the testing method that we used in release 2.0 is different from release 1.0 which was used “Big Bang” approach. Instead, we implemented “Bottom-up” approach which is a more systematic and effective approach to help us test the program.

Testing method

Since the “Big Bang” approach that we used in release 1.0 has increased complexity and efforts to locate the bugs causing code in the program, more time is needed to spend on testing and debugging which in turn the project plan had been slightly delayed. In order to give greater confidence that program is being test properly and increase the ability to trace the bugs more easily, the “Bottom up” approach is then being used in the program testing of the release 2.0. The “Bottom-up” strategy required integration testing before performing system testing while all bottom level classes had been tested by unit testing. Hence, we can ensure that the lower level classes were being tested and confirmed that the code in those classes were performed as expected and prevent fault causing section from propagating to other classes, while the integration test can help to isolate and locate the bugs causing section in the sub-system. Unlike other testing strategies such as “Top-down”, ”Sandwich”, “Modified top-down”…etc. which test tubs have to be created when doing integration test, the “Bottom-up” approach requires no test stub during the testing process which means some efforts and time can be saved while the program is still being tested properly. Therefore, we decided to use “Bottom-up” strategy due to limited development time for our project.

Test plan



The diagram from above showed a general view of the testing sequence and classes involved in unit-testing, integration and system testing. Since we were using “Bottom-up” strategy to test our program, there will be a total of 6 unit tests, 3 integration tests and 1 system test need to be validated and checked individually. The testing will begin with unit testing, followed by integrated testing and finally system testing. The following table shows the test sequence:

|  |
| --- |
| Unit Testing:  Food (Stub: N/A)  Appetizer (Stub: N/A)  MainDish (Stub: N/A)  Drink (Stub: N/A)  Dessert (Stub: N/A)  UserInput (Stub: N/A) |
| Integration Testing:  Combination + Food (Stub: N/A)  Information + UserInput + Food + Appetizer + MainDish + Drink + Dessert (Stub: N/A)  Calculation + Combination + Information + UserInput + Food + Appetizer + MainDish + Drink + Dessert (Stub: N/A) |
| System Testing:  Main + Calculation + Combination + Information + UserInput + Food + Appetizer + MainDish + Drink + Dessert (Stub: N/A) |

The testing was done by using Junit.4 library in Eclipse.