

# Application design and development

*by* Thomas Evans

---

|                |                                                                          |                 |      |
|----------------|--------------------------------------------------------------------------|-----------------|------|
| FILE           | 2770_THOMAS_EVANS_APPLICATION_DESIGN_AND_DEVELOPMENT_93264_14719517.DOCX |                 |      |
| TIME SUBMITTED | 23-MAR-2018 06:45PM (UTC+0000)                                           | WORD COUNT      | 1297 |
| SUBMISSION ID  | 85480410                                                                 | CHARACTER COUNT | 6443 |

UWTSD

# Application Design & Development

Mike Dacey

Thomas Evans P145653  
Web Development

---

*Table of Contents*

*Part A*

*Part D*

*Part C*

*Part B*

*References*

---

## Part A

The brief for this report requests a discussion on the importance of object-oriented programming outlining a key importance on four key items that must be discussed during this section of the report.

Firstly, the brief asks for an explanation between classes and objects. Classes and objects are used to represent real life objects or entities. A class can represent an item whereas an object will represent its properties. For example, A car class would contain several object properties for example Doors, Seats, Wheels, Engine, etc. Going from this a class represents its an item and an object represents its characteristics a method on the other hand would represent its behaviours. For example, in the car class example the methods could be accelerate, break or turn.

Secondly the brief requests a debate over the use of inheritance of and polymorphism in large scale applications. Polymorphism and inheritance use classes and interfaces to outline general details to an object and then create a child class to that original class or interface to add specialisation. For example, if there was a student parent class would include properties like p number, name and course. Child classes could then be used to outline the different kind of students for example there could be a sub class for full time students and a sub class for part time students with properties that are only consistent to their classes. These principles are important as they can be very useful for when it comes to changing or adding to a program. Without using these principles, it would require a programmer to add sections of code to multiple places of code in order to add functionality. As oppose to with these principles adding functionality is as simple adding sub classes with new properties or functionalities.

Interfaces are an important part of object-oriented programming especially when using polymorphism. Interfaces should be used to outline what an object should do and then a class should be used to create the functionality outlined by the interface. For example, an interface could be used for pets. This could be used to outline that all pets should make a noise. Classes inheriting from this interface will all need to include a noise but could be different noises for example cats would 'meow' and dogs would 'woof'.

The dependency inversion principle states that High level modules shouldn't depend on low level but should depend on abstraction. Also, abstraction shouldn't depend on details, but details should depend on Abstraction. The clear advantage of this is that code will include more classes and interfaces and will be more flexible. The downside to this is that

## **PART D**

Software testing is essential to any projects development life cycle. Poor software testing can result in major bugs not being solved for the duration of a projects life cycle. The longer it takes for bug to be found in a project the higher the cost for the project.

A software bug can be defined by doing something that the product specification says it shouldn't or not doing something it should. By doing something the product specification doesn't mention but should do. Also, if the finished product is too slow or difficult to use.

Software testing can prove to be incredibly expensive with cost only growing within a projects lifecycle. It has been estimated that up to 50% of a projects budget can be taken up project testing [1].

There are many different types of software testing. One of which is White Box testing. White-Box testing is a sophisticated way of software testing where a tester must look at the inner workings of a program to identify any software bugs. White box testing can be great as it as seen as stable as most tests can be reused if the objects that compromise the code do not return [2].

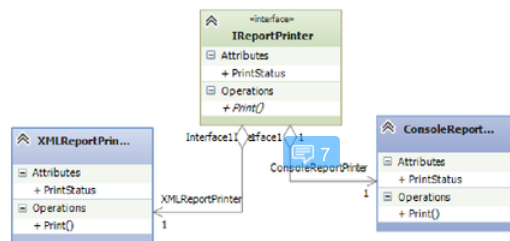
Black-Box testing is seen as testing looking at outputs alone. The benefits of this are great as this form of software testing is extremely easy in comparison to others. It is also by far one of the quickest forms of software testing and also seen as simple as it is based off of studying valid and invalid inputs and outputs.

Unit testing is a form of White-box testing normally done by programmers during the programming cycle of the projects as the code is being written. This form of testing is normally only used for small sections of code and would not be used to test anything larger than a class [1]. This form of testing is essential as it can outline bugs early in the project development cycle and can avoid being slowed down by small bugs late in the projects life [3]. The next step in the testing cycle is integration testing which is simply Unit testing on a slightly larger scale possibly using a few classes and interfaces together.

Moving on later into the testing cycle would be the use of a beta test. This test would be carried out late in the development cycle. This is normally conducted by possible customers and would be conducted on the full version of the product or at least a large chunk of it. The purpose of this is to gain feedback from possible clients and fix problems before a large-scale release.

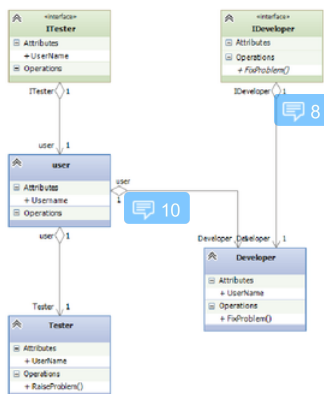
## Part C

The code supplied for this section of the report allows the user to conduct fake tests and asks if they were successful. The Program will then cycle back after it has asked for each test that has been conducted and will then



The brief supplies a program to be studied and demands that the program be taken and for the author of this report to create a set of UML diagrams to show how the structure of the program. The first UML diagram is below and shows the

inheritance between the printing interfaces and the printing classes. Both classes inherit the print status attribute and the Print() operation.



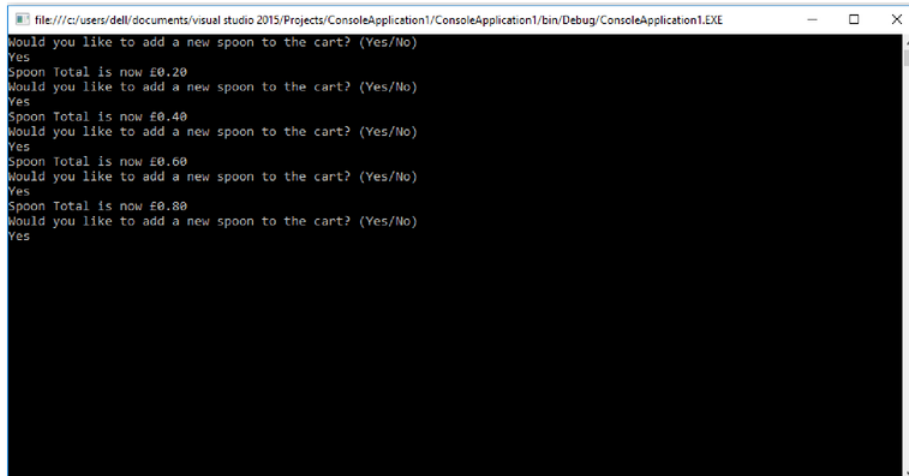
Also created are was a second UML Diagram showing the links from the 'IDeveloper' and 'ITester'. User inherits the 'username' attribute from the 'ITester'. Developers and testers will then have inherited username from 'user'. Developer also inherits 'F' ixProblem()

The program uses the Singleton design pattern to access the data on whether the test was successful or not. The pattern will then allow the program to ask the user if the problem has been solved. ixProblem()' from the IDeveloper interface.

## Part B

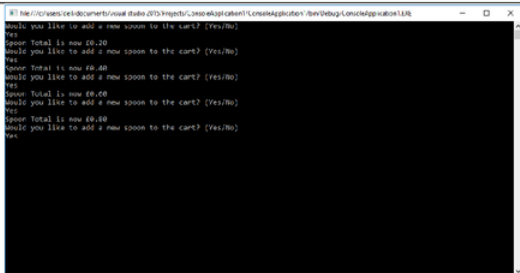
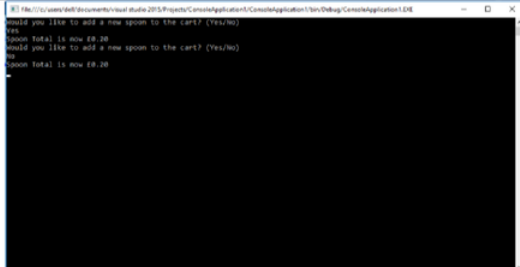
The brief provides 4 challenges to be attempted. The first of which is to create a class that enables the user to add new spoons. A simple class created simply adds the new spoon to the current total price. The completed program can be seen in the screenshot below.

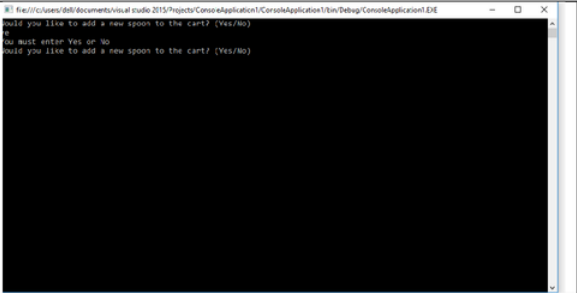
### Exercise 1: Spoon



```
file:///c:/users/dell/documents/visual studio 2015/Projects/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1.EXE
Would you like to add a new spoon to the cart? (Yes/No)
Yes
Spoon Total is now £0.20
Would you like to add a new spoon to the cart? (Yes/No)
Yes
Spoon Total is now £0.40
Would you like to add a new spoon to the cart? (Yes/No)
Yes
Spoon Total is now £0.60
Would you like to add a new spoon to the cart? (Yes/No)
Yes
Spoon Total is now £0.80
Would you like to add a new spoon to the cart? (Yes/No)
Yes
```

Above is a simple easy test to see if the program is working as it should. Below will be a table displaying a small black box test of the results of different tests.

| Input | Explanation                                                                                       | Screenshot                                                                           |
|-------|---------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
| "Yes" | Inputting the "Yes" Provides the expected output and adds a new spoon price to the current total. |  |
| "No"  | The "No" Input simply displays the current total and then closes.                                 |  |

|                     |                                                                                       |                                                                                                                                                                                                                                                                                                                                                                                    |
|---------------------|---------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Unwanted input "ye" | This input is unaccepted by the program and then asks the user to input a valid input |  <pre>81 file:///c:/users/dell/documents/visual studio 2015/Projects/ConsoleApplication1/ConsoleApplication1/Run/Debug/ConsoleApplication1.DEB Would you like to add a new spoon to the cart? (Yes/No) ye You must enter Yes or No Would you like to add a new spoon to the cart? (Yes/No)</pre> |
|---------------------|---------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

## References

- [1] Nidhra S.(2012). Black Box and White Box Testing Techniques – A Literature Review. International Journal of Embedded Systems and Applications, 2 (2), pp.29-50.
- [2] Cs.unh.edu. (2018). [online] Available at:  
[http://www.cs.unh.edu/~it666/reading\\_list/Defense/blackbox\\_vs\\_whitebox\\_testing.pdf](http://www.cs.unh.edu/~it666/reading_list/Defense/blackbox_vs_whitebox_testing.pdf) [Accessed 18 Mar. 2018].
- [3] Condor.depaul.edu. (2018). [online] Available at:  
<http://condor.depaul.edu/sjost/hci430/documents/testing/UnitTesting.pdf> [Accessed 18 Mar. 2018].
- [4] Chillarege R(1999). Software Testing Best Practices. IBM Research Technical Report RC 214457

# Application design and development

---

## GRADEMARK REPORT

---

### FINAL GRADE

40/100

### GENERAL COMMENTS

#### Instructor

Knowledge OOPs concepts (30) 22

Discussion of design principles (10) 3

Knowledge of testing (10) 6

UML (20) 7

Exercise 1 (5) 2

Exercise 2 (5) 0

Exercise 3 (10) 0

Exercise 4 (10) 0

Classes vs Objects: the explanation provided is a very confused, and I don't really agree with it.

Inheritance and Polymorphism: description needed to demonstrate a better understanding of the benefits.

Interfaces: you needed to talk about roles and how we write interfaces from the "user" perspective rather than from the classes implementing it.

Dependency Inversion: incomplete

Software Testing: you have briefly covered white, black and unit testing. But you could have outlined more of the benefits of these approaches.

UML: Class diagram was incomplete and no sequence diagrams provided. Seems rushed and a half hearted attempt.

Ex1: you have only counted the spoons

Ex2-4: not completed

---

PAGE 1

---

PAGE 2

---

PAGE 3

---



### **Comment 1**

I don't think this actually means what you think it means.



### **Comment 2**

attributes??



### **Comment 3**

No.

I would say a class represents a "type" of item, an object an instance of an item, the class attributes represent the characteristics and the methods the behaviours of that "type" of item.



### **Comment 4**

Again, you are on the correct lines here and you are implying "abstraction" to allow code to support multiple types of objects, in particular the Open Closed Principle. But you haven't really demonstrated you fully understood this concept.



### **Comment 5**

Why? How are they related?



### **Comment 6**

not finished!!

---

PAGE 4

---

PAGE 5

---



### **Comment 7**

Should be an implements relationship



### Comment 8

Should be an implements relationship



### Comment 9

??



### Comment 10

should be an inheritance relationship



### Comment 11

??

PAGE 6

---

PAGE 7

---

PAGE 8

---