

Object Oriented Programming with C++

by Ashleigh Thomas George Edwin Robert Elder

FILE	2822_ASHLEIGH_THOMAS_GEORGE_EDWIN_ROBERT_ELDER_OBJECT_ORIENTED_PROGRAMMING_WITH_C__108537_861758083.DOCX		
TIME SUBMITTED	21-MAR-2018 07:43PM (UTC+0000)	WORD COUNT	490
SUBMISSION ID	85250106	CHARACTER COUNT	6998

Object Orientated programming with C++

Ashleigh Elder P141924

March 21st 2018

Contents

- 1 Introduction
- 2 Terms and Diagrams
 - 2.0 Terminology
 - 2.1 Class UML
 - 2.2 Use Case UML
- 3 The Program
- 4 Bibliography

1 Introduction

For this assignment a “company” required a system to control its electronics manufacturing business. It was required that this system cover all aspects of running a business from ordering parts and producing components to receiving customer orders and storing account information.

2 Terms and Diagrams

2.0 Terminology

Use Case

A “Use Case Diagram” is a diagram that shows the relationship of different “Actors” with a program, the actors can only access specific functions that then use information that is either inherited or extended from the previous function in order to complete an action the actor has initiated. 1

Class Diagram

A “Class Diagram” is a diagram that provides detailed information about the relationship of each class in the program to each other. This includes any methods or variables initiated by their associated classes (parents and associates). 2

Object

An object is an instance of a class that operates as part of another function in order to keep similar information in one place kind of like a struct, in the same way an array of structs can exist to keep similar data in the same location an array of objects can also be created. 3

Inheritance 5

When a subclass (or “child”) is created using the constructor of its “parent” the child inherits specific methods and variables from its parents. These can be used by an object of the “child” class when called by main. When the “parent” class is deconstructed the children of that class are also deconstructed. 6

Aggregation 7

Aggregation is similar to Inheritance in that a main class controls the fate of its subclasses with one main difference, the subclass is created as its own object therefore when one part of an aggregated class “family” is deconstructed, the other classes are not.

Overloading

A function can be overloaded by giving the same function multiple definitions with the constraint of having different types (i.e int float and string). When an overloaded function is called the most appropriate type is chosen by default and used as the function. 8

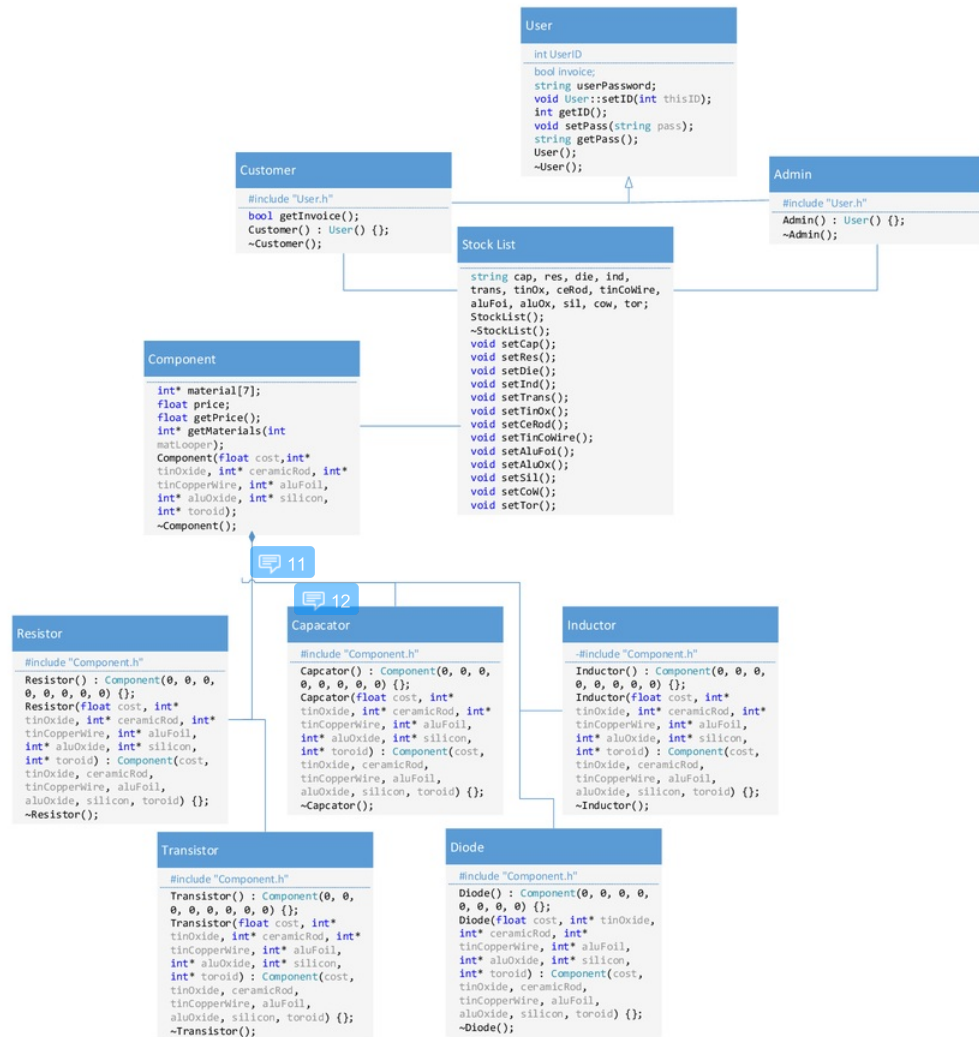
Overriding

An overridden class has a base and derived class with the same name and arguments and the member function of the derived class is used instead of the base class which gets ignored.

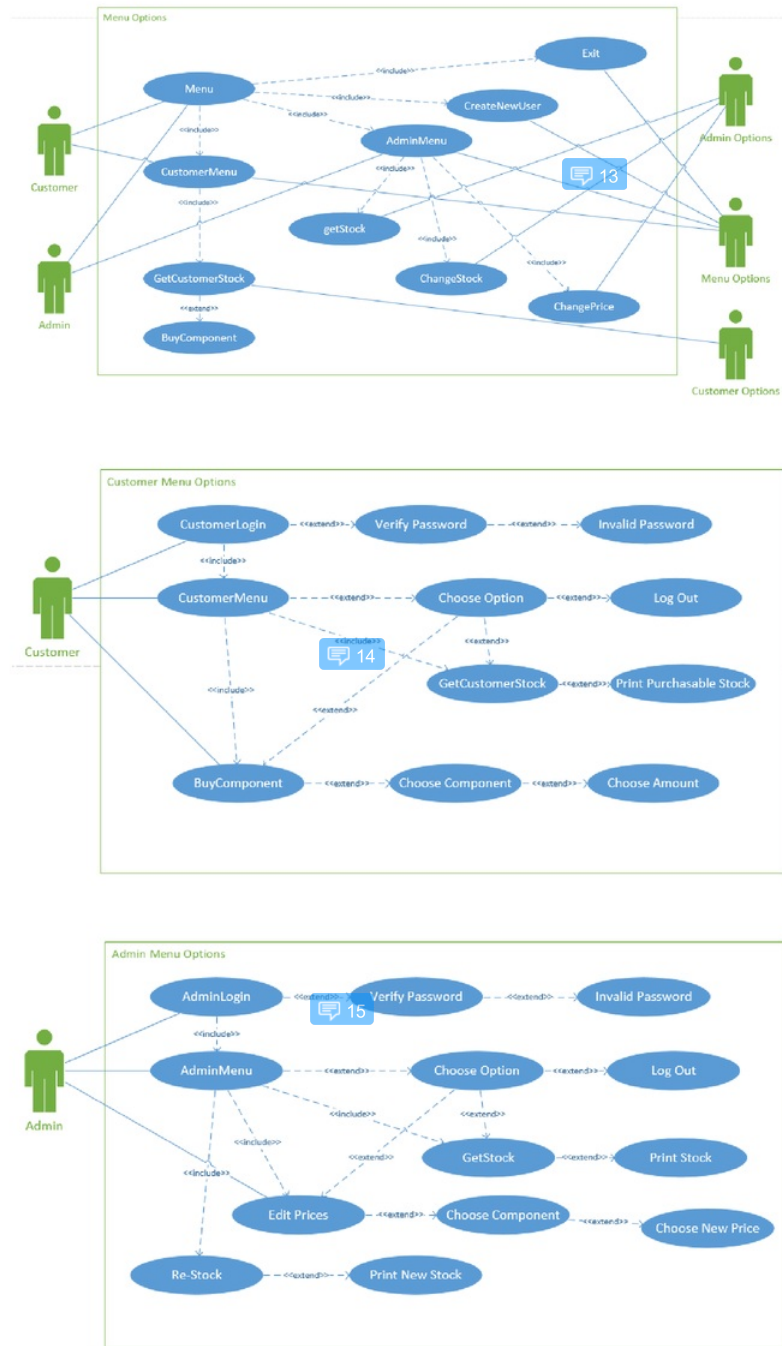
Singleton Pattern

A class following the "Singleton Pattern" can only ever be created statically one time by its self. This static object can be used by other methods to read data but cannot in and of itself be changed. This can be used when a large number of variables need declaring in order to be used by the rest of the code without needing global variables, as they can be unstable and inefficient.

2.1 Class UML



2.2 Use Case UML



3 The Program

```
C:\Users\mei\Documents\Visual Studio 2013\Projects\Prog Assign\Debug\Prog Assign.exe
1. Customer
2. Admin
3. New Account
4. Exit
>
User ID:
0000
New Password:
end
Incorrect User ID or Password!
User ID:
0000
New Password:
new
Welcome Customer0000

1. View Stock List
2. Buy Component
3. Checkout
4. Log out
>
ComponentID Component Quantity Price
0000 Resistor 100000 0.65
0001 Capacitor 100000 0.75
0002 Inductor 100000 0.35
0003 Transistor 100000 0.35
0004 Diode 100000 0.35

1. View Stock List
2. Buy Component
3. Checkout
4. Log out
>
2
Buy Component

1. View Stock List
2. Buy Component
3. Checkout
4. Log out
>
3
Go to Checkout

1. View Stock List
2. Buy Component
3. Checkout
4. Log out
>
```

```
C:\Users\mei\Documents\Visual Studio 2013\Projects\Prog Assign\Debug\Prog Assign.exe
1. Customer
2. Admin
3. New Account
4. Exit
>
Admin Password:
0011
Incorrect Admin Password!
Admin Password:
Password
Welcome Admin

1. View Stock List
2. Edit Prices
3. Edit Stock
4. Log out
>
ComponentID Component Quantity Price
0000 Resistor 100000 0.65
0001 Capacitor 100000 0.75
0002 Inductor 100000 0.35
0003 Transistor 100000 0.35
0004 Diode 100000 0.35
0015 Tintoxide 40000000 0.001
0016 Ceramicd 100000 0.01
0017 Tinnedcopperwire 10000000 0.001
0018 Aluminiumfoil 40000000 0.001
0019 Aluminiumoxide 1000000 0.01
0020 Silicon 1000000 0.01
0021 Capacitwire 1000000 0.001
0022 Tantal 100000 0.01
CompanyFunds 100000

1. View Stock List
2. Edit Prices
3. Edit Stock
4. Log out
>
3
edit prices

1. View Stock List
2. Edit Prices
3. Edit Stock
4. Log out
>
```

```

C:\Users\elder\Documents\Visual Studio 2015\Projects\Pog Assign\Debug\Pog Assign.exe
0022  Toroid  100000  0.01
Company Funds 130000
1. View Stock List
2. Edit Prices
3. Edit Stock
4. Log out
5.
6. Customer
7. Admin
8. New Account
9. Exit
10.
Thank you for using "Ashleigh's Rudimentary Shopping Experience"!
Press any key to continue . . .

```

Singleton

```

    StockList *instance = 0; //declare static object to make a singleton pattern

    void StockList::setCap()
    {
        cap = "100000";
    }
    void StockList::setRes()
    {
        res = "100000";
    }
    void StockList::setDie()
    {
        die = "100000";
    }
    void StockList::setInd()
    {
        ind = "100000";
    }

```

```
void StockList::setAluFoi()
{
    aluFoi = "40000000";
}
void StockList::setAluOx()
{
    aluOx = "1000000";
}
void StockList::setSil()
{
    sil = "1000000";
}
void StockList::setCoW()
{
    coW = "100000";
}
void StockList::setTor()
{
    tor = "100000";
}

//each method sets the variables of the singleton to a specific value that can be called by other functions
```

Overloading

```
void changePrice(int whatComp, float price); //edit the price of a component and write to file
void changeStock(StockList *instance, string cap, string res, string die, string ind, string tran
//^^ restock and print to file
float buyComponent(int whatComp, float amount); //customer can purchase a component
```

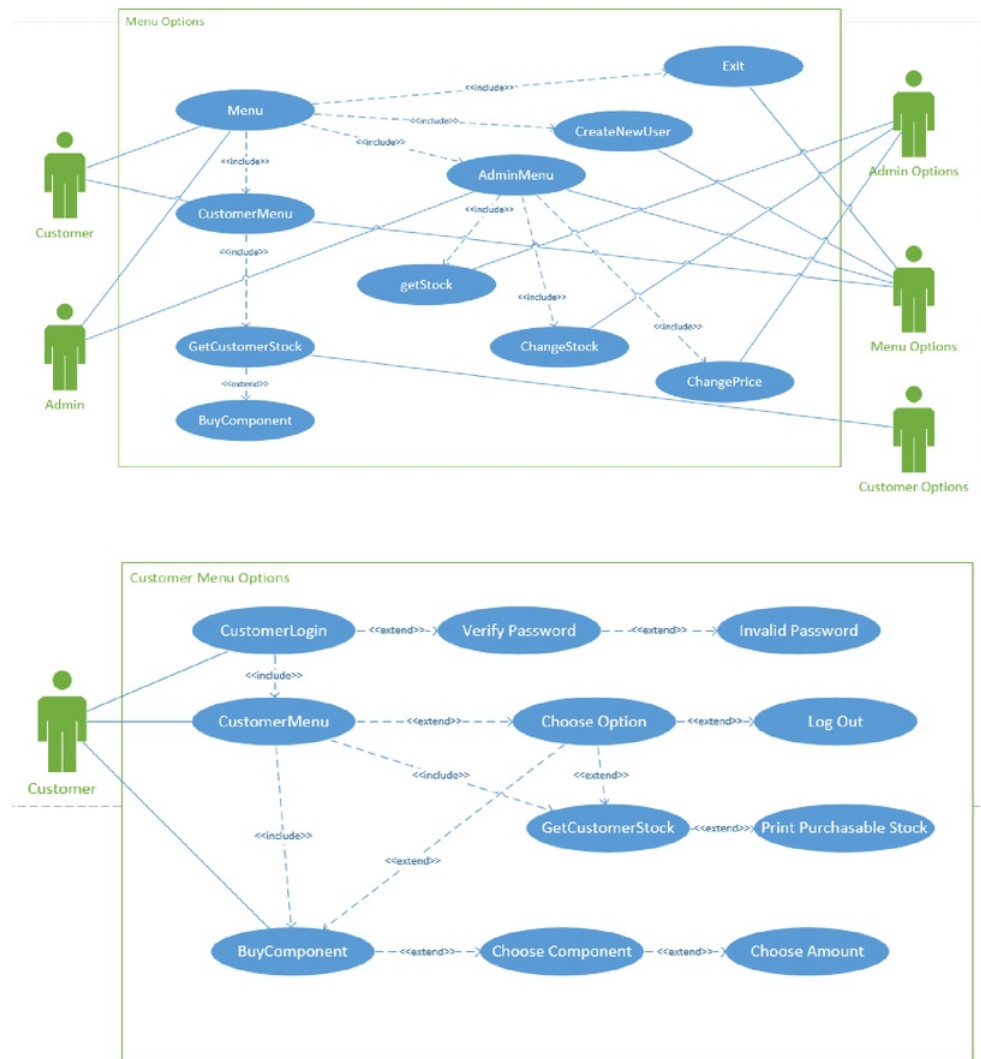
Overriding

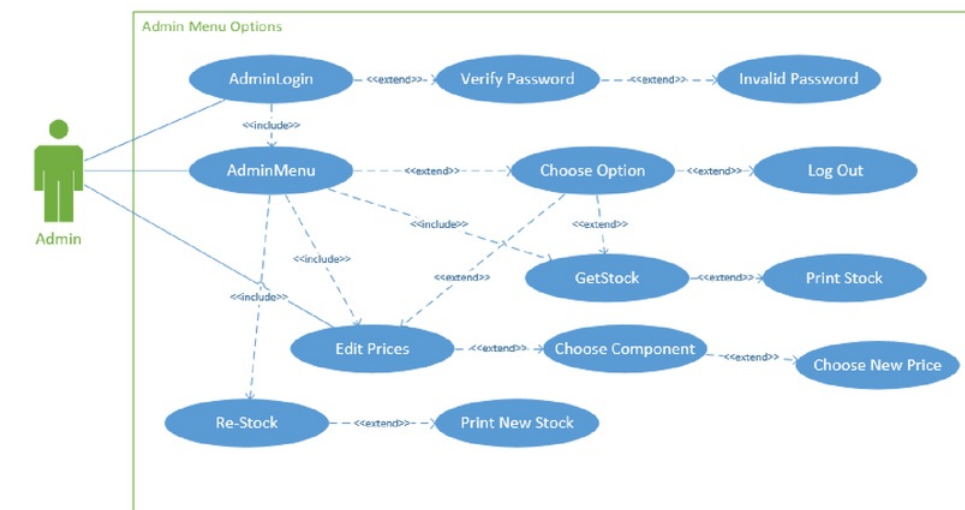
```
void changeStock(StockList *instance, string cap, string res, string die, string ind, string trans, string tinOx, string ceRod, string
{
    if (!instance)
        instance = new StockList;
    fstream stock;
    int x, y = 0, a = 0, b = 0, vary[58];
    string text, hunt, var[58], q;
    stock.open("Stocklist.txt", stock.in);
    if (stock.is_open())
    {
        while (getline(stock, text))
        {
            var[b] = text; //create an array and put each line in each part of the array
            b++; //go to next array location
        }
        stock.close();
    }
}
```

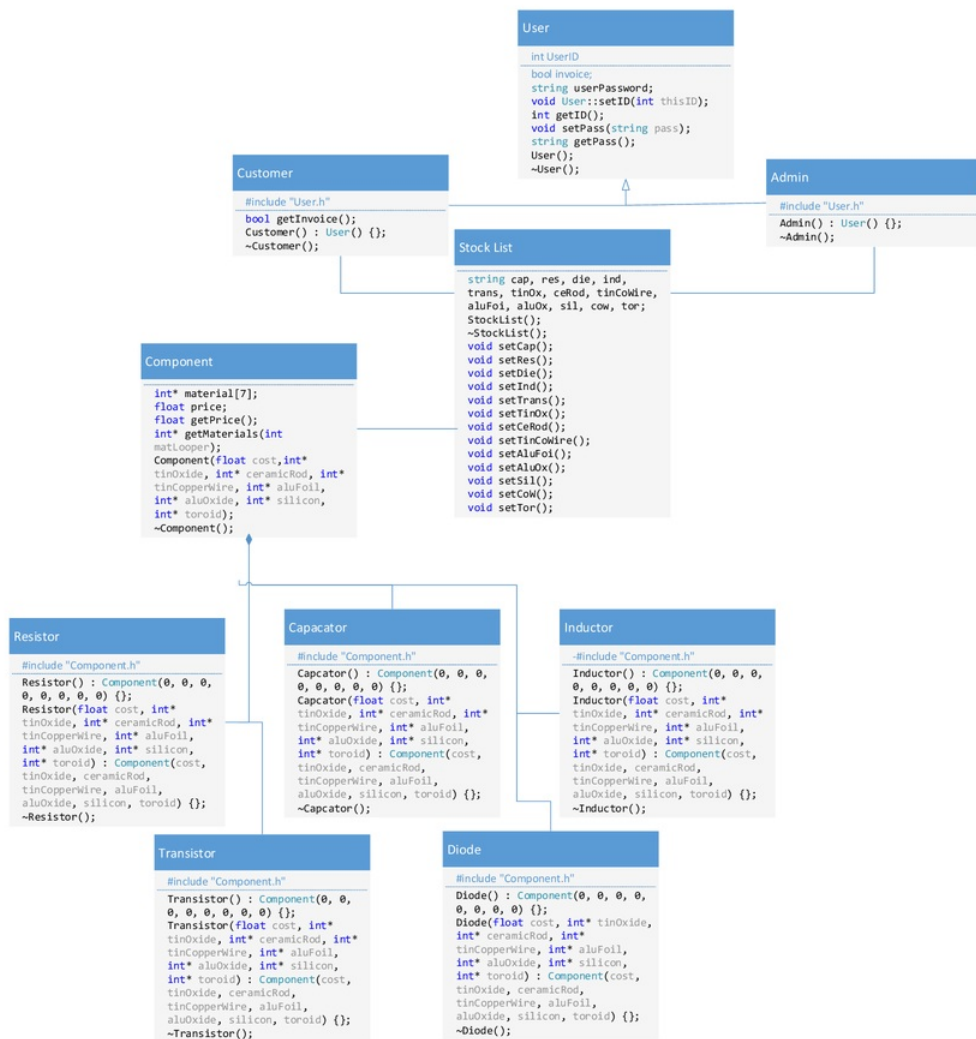
Inheritance

```
1  #pragma once
2  #include "User.h"
3  class Customer : public User //customer is set up using users constructor as it's a child class of User
4  {
5  public:
6      bool getInvoice();
7      Customer() : User() {}
8      ~Customer();
9  };
10
```

5 Bibliography







Object Oriented Programming with C++

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

29/50

PAGE 1

PAGE 2

PAGE 3



Comment 1

Careful of the terminology utilised here, i understand what you are trying to explain however the terminology here doesnt really reflect what an use case diagram is for



Comment 2

Good explanation of purpose and usage given here



Comment 3

Good definition given here



Comment 4

Use of informal language "Kind of"



Comment 5

May want to change font to indicate headings here



Comment 6

Good definition for inheritance given here with clarification



Comment 7

A little unclear as to what you mean here, however, the latter indicates an understanding on a fundamental level



Comment 8

Again a little unclear but the concepts are given



Comment 9

Not technically correct., the base class cannot have the same name as the derived class. The member functions and parameters however, must match



Comment 10

Not sure i agree with the final reasoning given here but this shows good understanding



Comment 11

Thgis should be inheritance not composition,

composition is where a class contains another, inheritance references or uses the properties of another class



Comment 12

What is this abstract association ?



Comment 13

These associated lines are not necessary here. The actors do not make sense in thise case. The only ones that are required are the "User" and "Admin".

In addition the use-case diagram should include use cases within the system boundary



Comment 14

Far better attempt at a use-case diagram, some of the extends should be includes within this diagram.

(e.g: should the verify password be optional)



Comment 15

Again far better attempt, but there are issues still hold from the previous diagram



Comment 16

Good evidence of pattern used here



Comment 17

Not technically evidence of overloading



Comment 18

Good evidence of inheritance here