

OOPS Assignment 1

by Alexander Hall

FILE	2655_ALEXANDER_HALL_OOPS_ASSIGNMENT_1_108537_1240856158.D OCX		
TIME SUBMITTED	21-MAR-2018 04:19PM (UTC+0000)	WORD COUNT	1524
SUBMISSION ID	85237212	CHARACTER COUNT	12199

Assignment Title:

Development of a manufacturing system simulation.

MODULE CODE: M2X8523

MODULE LECTURER: JAYMIE L. PHILLIPS

MODULE TITLE: OBJECT ORIENTED PROGRAMMING

STUDENT NUMBER: ALEXANDER HALL (P151746)

Contents

Introduction	2
Background Theory and Coding	2
Classes	2-3
Objects	3-4
Patterns	4
Singleton	4
Inheritance	4
Overloading	5
Overriding	5
Composition	5
Aggregation	5-6
File Access/Stream	6
System design and UML models	6
Design Specifications	6-7
Structural: -	7
Class Diagram	7
Behavioural: -	8
Use-Case diagram	8-10
Sequence Diagram	10-12
References	13

Introduction

This assignment is used to show a level of understanding of C++ programming language as well as use the object-oriented programming techniques, these techniques will be outlined below and will be used to create a manufacturing and stocking system for a manufacturing business, also there will be design models that outline what the system does using UML (Unified Modelling language) diagrams.

The manufacturing business must have a system for manufacturing and selling a range of components, this system should include features such as ordering raw materials, ordering new components, making user accounts for both individual customers and companies and should include a stocking system that can be used by the users mentioned above as well as being maintained by the manufacturing business itself.

The components that need manufacturing are resistors, capacitors, transistors, diodes and inductors.

Background Theory and Coding

This section is used to describe the various parts of Object oriented programming, it will also show the code that has been written for the system using the object-oriented programming techniques.

Classes

Classes are usually made up of a header (.h files) and a source files (.cpp files), both files have the same name, here is an example of a classes header and source files: -

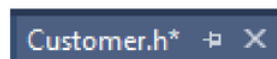


Figure 1-Customer Class header file

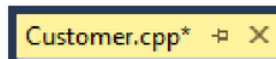


Figure 2-Customer Class Source file

The class files are used to define data types (attributes) such as int, char, string or float, which normally go within the 'private' section of the class meaning they can only be used within the scope of the Class itself, this means the methods or functions usually go into the 'public' section of the class files, this way the functions can be used within the bigger program outside of the specified class files, meaning that the Class files are used overall as a type of blueprint to get data when an instance or object is instantiated .

Here are the images of the class files for the manufacturing business's system filled with the data types, member functions and Macros: -

```

1  #ifndef CUSTOMER_H
2  #define CUSTOMER_H
3  #include <string.h> // allows user to use string (The #include keyword shows it's a preprocessor and to include the string library)
4  #include <iostream> // allows user to use input and outputs (This preprocessor is used to include the input and output streams library)
5  using namespace std;
6  class Customer // This is a user defined Class name and is using the Camel case naming convention.
7  {
8  private: // Private keyword is used to describe data that is only used within the class files.
9      int id; //
10     int order_number; // These are the data types of the customer class, they can also be called its attributes.
11     string user_name; //
12     string password; //
13 public: // Public keyword means that the methods below can be used by the entire program not just the Class files.
14     Customer(int Customer_Id, int Customer_Order_Number, string Customer_User_Name, string Password);
15     ~Customer(); // This is the default destructor for this class file.
16
17     int GetId(); //
18     int GetOrder_Number(); //
19     string GetUser_Name(); // These are member functions that are needed to define the "Customer" class.
20     string GetPassword(); //
21 };
22
23 #endif
24

```

Figure 3-Customer header file filled with Attributes, Member Functions, Pre-processor's and Macro's

```

1  #include "Customer.h" // The "#include" keyword is used so that the source file is linked with the Customer header file.
2
3  Customer::Customer(int Customer_Id, int Customer_Order_Number, string Customer_User_Name, string Password)
4  {
5      id = Customer_Id; //
6      order_number = Customer_Order_Number; //
7      user_name = Customer_User_Name; // Equating the basic defined data variables to data variables that can be used within the whole program.
8      password = Password; //
9  }
10
11
12  Customer::~Customer()
13  {
14  }
15  // Below are the implementations of the member functions declared in the Customer class header file.
16  int Customer::GetId() // This "Get" member function is used to return the integer of "id".
17  {
18      return id;
19  }
20
21
22  int Customer::GetOrder_Number() // This "Get" member function is used to return the integer of "order_number".
23  {
24      return order_number;
25  }
26
27  string Customer::GetUser_Name() // This "Get" member function is used to return the string data of "user_name".
28  {
29      return user_name;
30  }
31
32  string Customer::GetPassword() // This "Get" member is used to return the string data of "password".
33  {
34      return password;
35  }
36

```

Figure 4-The Customer Source file

Objects

Objects are what are created when a class is used with a program needs to use a variety of data types at the same time to carry out a specific purpose, objects can also be called

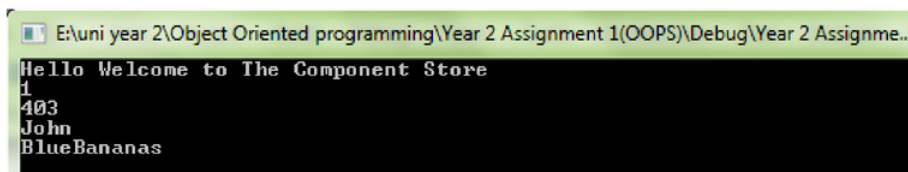
instances as you can use an object many different times, creating many different instances depending what the object is being used for, here is an example of an object that was part of the manufacturing businesses program:-

```

//getting data from the customer class and printing out the new customer's details//// creating an instance of object.
Customer*customer1 = new Customer( 01, 403, "John","BlueBananas");//Giving values from the overloaded customer.h file for each interger or string.
cout << customer1->GetId()<< endl;//Prints out the Value given above for Customer Id.
cout << customer1->GetOrder_Number()<< endl;//Prints out the value given above for the Order_Number of the Customer.
cout << customer1->GetUser_Name().c_str()<< endl;//Prints out the data given above for the Customer's User_Name.
cout << customer1->GetPassword().c_str() << endl<<endl;//Prints out the data given above for the Customers Password.

```

Figure 5- Instantiation of a customer object



```

E:\uni year 2\Object Oriented programming\Year 2 Assignment 1(OOPS)\Debug\Year 2 Assignme...
Hello Welcome to The Component Store
1
403
John
BlueBananas

```

Figure 6-Output when the Customer object has been instantiated

Patterns

There are many three types of patterns within the subject of C++ object-oriented programming, which are creational, structural and behavioural patterns. The type that will be outlined below is the singleton Creational design pattern, these types of patterns are usually used within the creations of classes and objects and programs use these patterns to solve various problems that may appear during the creation of objects or make the creation of new objects simpler by the reuse of patterns.[1]

Singleton

The singleton pattern is a creational pattern that is used when only a singular instance of the singleton class is needed within a given program. it is also used when the class needs to be called-upon by the entire program, but the program only being able to call upon the one singular instance giving the program one place to access it when an object using the singleton class is instantiated. The Singleton pattern is used as a class, where the only class has statically defined member functions and attributes.[2]

Inheritance

Inheritance within C++ programming is when a Specific 'derived class' such as the transistor class will use/gain some of the 'base class' such as the component classes data types, methods and functions here is an example of inheritance within the manufacturing businesses program: -

```

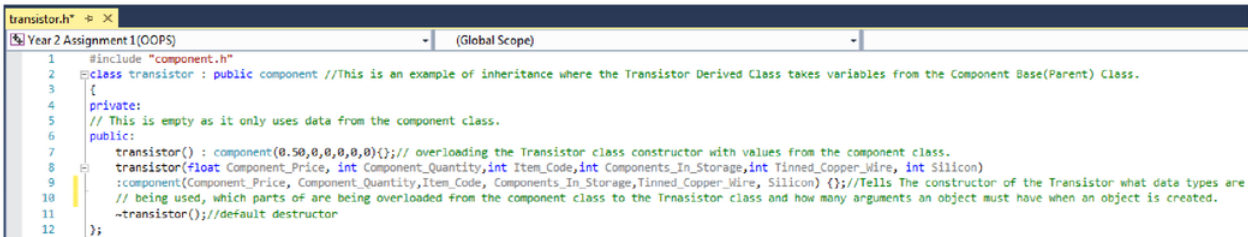
1 #include "component.h"
2 class transistor : public component //This is an example of inheritance where the Transistor Derived Class takes variables from the Component Base(Parent) Class.
3 {

```

Figure 7-Inheritance of component class by the transistor class

Overloading

Constructor Overloading is the act of passing in specific data to a classes constructor which is seen as a type of class function, function overloading is used so that when an object is created within a given C++ program there is a reference of what data needs to be used and how many different arguments need to be called depending on the what data and how many arguments are needed by the object created. The constructor can be seen as being within the public part of the class and the constructor below within the transistor class header file is overloaded with data and the amount of arguments of the component base class:-

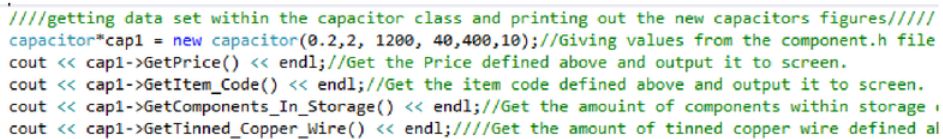


```
transistor.h
1 #include "component.h"
2 class transistor : public component //This is an example of inheritance where the Transistor Derived Class takes variables from the Component Base(Parent) Class.
3 {
4 private:
5 // This is empty as it only uses data from the component class.
6 public:
7 transistor() : component(0.50,0,0,0,0,0){}; // overloading the Transistor class constructor with values from the component class.
8 transistor(float Component_Price, int Component_Quantity, int Item_Code, int Components_In_Storage, int Tinned_Copper_Wire, int Silicon)
9 :component(Component_Price, Component_Quantity, Item_Code, Components_In_Storage, Tinned_Copper_Wire, Silicon) {};//Tells The constructor of the Transistor what data types are
10 // being used, which parts of are being overloaded from the component class to the Transistor class and how many arguments an object must have when an object is created.
11 ~transistor();//default destructor
12 };
```

Figure 8 - Overloading the Transistor class's constructor with the data types and amount of arguments of the component class.

Overriding

Overriding can be part of inheritance, it happens when a child (derived) class has a function with the same name and amount of defined data types as its parent (Base) class, when the function is called by an instantiated object within the program, the child class's function will override the function defined in the parent class.[3]



```
1 //getting data set within the capacitor class and printing out the new capacitors figures/////
2 capacitor*cap1 = new capacitor(0.2,2, 1200, 40,400,10);//Giving values from the component.h file
3 cout << cap1->GetPrice() << endl;//Get the Price defined above and output it to screen.
4 cout << cap1->GetItem_Code() << endl;//Get the item code defined above and output it to screen.
5 cout << cap1->GetComponents_In_Storage() << endl;//Get the amount of components within storage
6 cout << cap1->GetTinned_Copper_Wire() << endl;//Get the amount of tinned copper wire defined al
```

Figure 9- Overriding the default values of the capacitor given by the component class

Composition

Composition is used within C++ programming and is the practice of grouping classes together to make a parent class, it's used when a relationship between classes is exclusively needed. It is done in the way that other subclasses end up being part of a parent class, this means that an object can be created from that parent class that must contain all parts of the other subclasses within it, but if any part the object or subclasses are destroyed then the parent class and all other parts are destroyed and the parent class would become useless and will then be unable to be used to create other objects.

Aggregation

Aggregation is where the parent class "can" contain other subclasses, but the relationship within aggregation is that the parent class may need to use the subclass's data or functions, this way when an object is created it may use all of the data and functions, or some of the data and functions of

subclass's mentioned within the parent class, but if parts of the object that are given by the subclasses are destroyed the object can continue to function using the remaining parts of the parent class.

File Access/Stream

File streams and access are used within C++ to store data within external files, there are three key words when it comes to the accessing of files these keywords are ifstream for reading from a file, ofstream for when data needs to be written to a file or there is the easier option of using fstream which can make a stream for reading out of and writing into a file, here is an example of using fstream to write into a file and read out of a file within the manufacturing businesses system:-

```
fstream stock; // fstream is used so that data can be written into or read from the Stock.txt file.

//writing to the Stock file////(writes the text below but deletes other data within Stock.txt file)
stock.open("Stock.txt",stock.out|stock.ate); //Opening file to write into,using the file for outputting and putting the pointer at the end of the stock file
stock << "Stock Room\n" << endl; //Writes the text Stock Room to the Stock.txt file.
stock.close(); //Closes the stock file after data has been written to it.
```

Figure 10-Writing to a Stock.txt File Using the fstream keyword

```
fstream stock; // fstream is used so that data can be written into or read from the Stock.txt file.

// ...

//reading from a file//
stock.open("Stock.txt",stock.in); // open the stock file for reading
string Stocklist=""; // Using a string to read over the file
getline(stock, Stocklist, '!'); // looking at all the lines within the Stock.txt file
cout << Stocklist << endl; //output everything within the Stock.txt
stock.close(); //closing the Stock.txt file
```

Figure 10-Reading from the Stock.txt file using the fstream keyword

```
E:\uni year 2\Object Oriented programming\Year 2 Assignment 1(OOPS)
Hello Welcome to The Component Store
Stock Room
Component  Item Code  Price  Stock Quantity
Resistor   RES      0.15   3000
Capacitor  CAP      0.30   2500
Diode      DIO      0.45   2500
Inductor   IND      0.25   3000
Transistor TRAN     0.55   3000
Press any key to continue . . .
```

Figure 11-Output window showing the data from within the Stock.txt file

System design and UML models

This section is used to outline the designing of the component manufacturing system, it will include many UML (Unified Modelling Language) diagrams that show the systems structure and behaviour.

Design Specifications

This is just an initial list of specifications that the system should include: -

Allow user (whether it is a customer or company) to make an account.

Check user account details so that they can purchase components.

Show an interface for users so that Components can be bought.

Only allow users to buy specific quantities of Capacitors, Transistors and inductors.

Give users an output of their specified order (including the price of each component they have bought and the overall price of their order).

Include a piece of code that lets customers know that their order is to be delivered.

Allow the business's employees to view the stock levels and amount of sales.

Manage stock levels (order new components or subcomponents when the business has run out of them)

Include a visual output of a graph for the levels of the sales or levels of the stock for the business employees.

Include the use of external files to keep records of sales and stock levels.

Give deals to specific customers depending on specific promotions (20% off order to any user who buys 250 or more of any Components and 25% discount off next order if the user buys components more than 3 time within one week.)

Structural: -

Class Diagram

This diagram is used to show the relationships between classes, the diagram shown below is a basic representation of the classes within the manufacturing business's system.

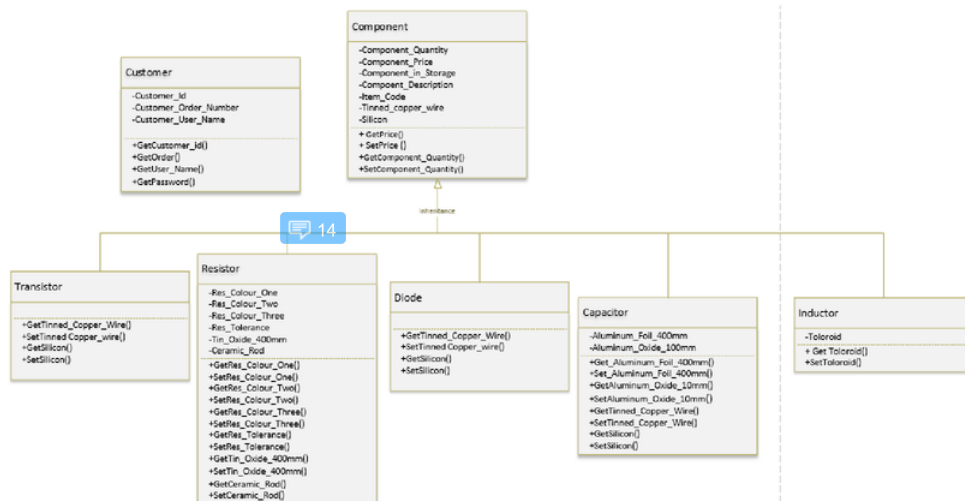


Diagram 1- Manufacturing business classes. (class diagram)

Behavioural: -

Use-Case diagram

Here are three use-case diagrams that describe the behaviour of user Account control, the Sales/Stock of the system and the re-stocking of components.

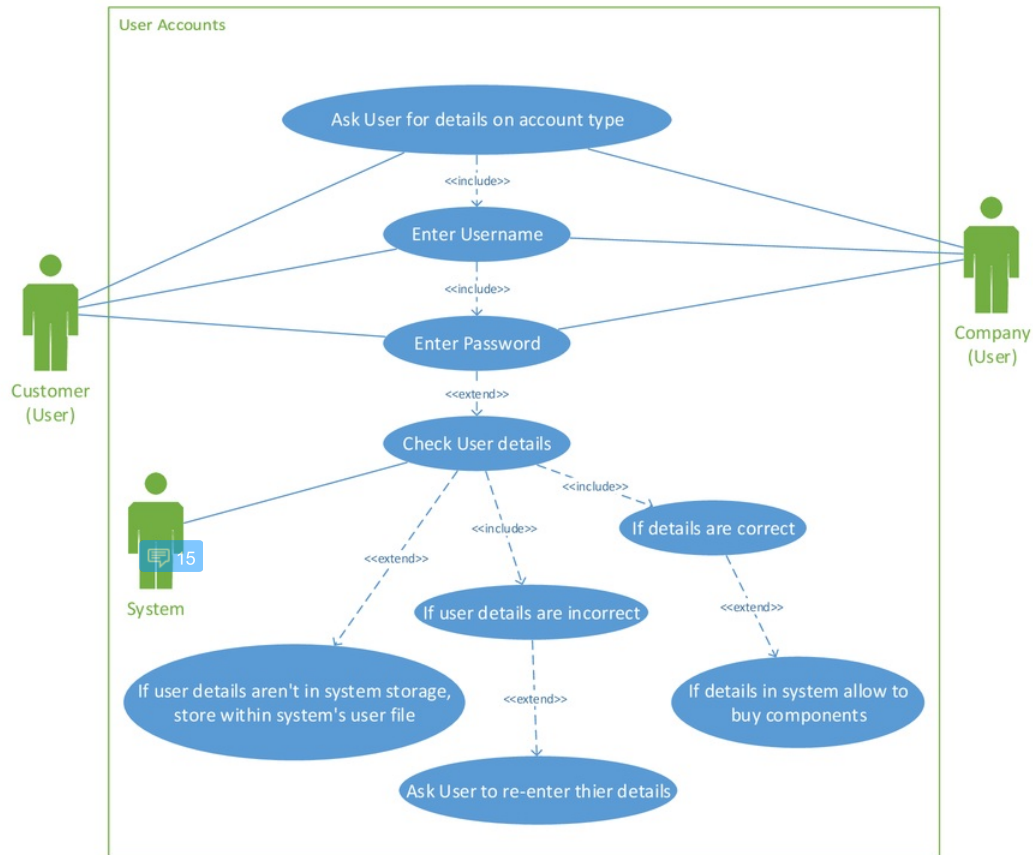


Diagram 2- User Account system use-Case diagram.

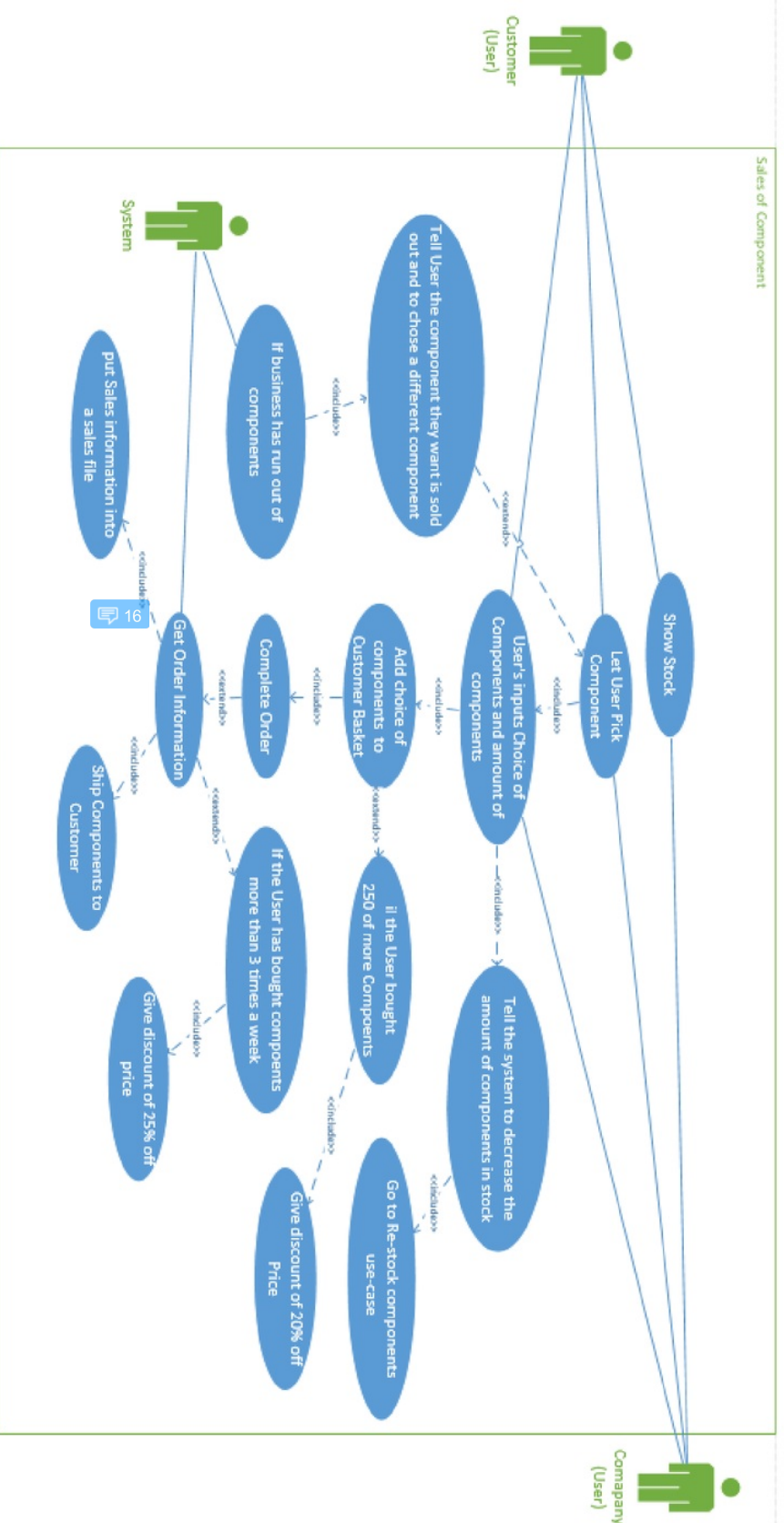


Diagram 3- Sales of component Use Case Diagram.

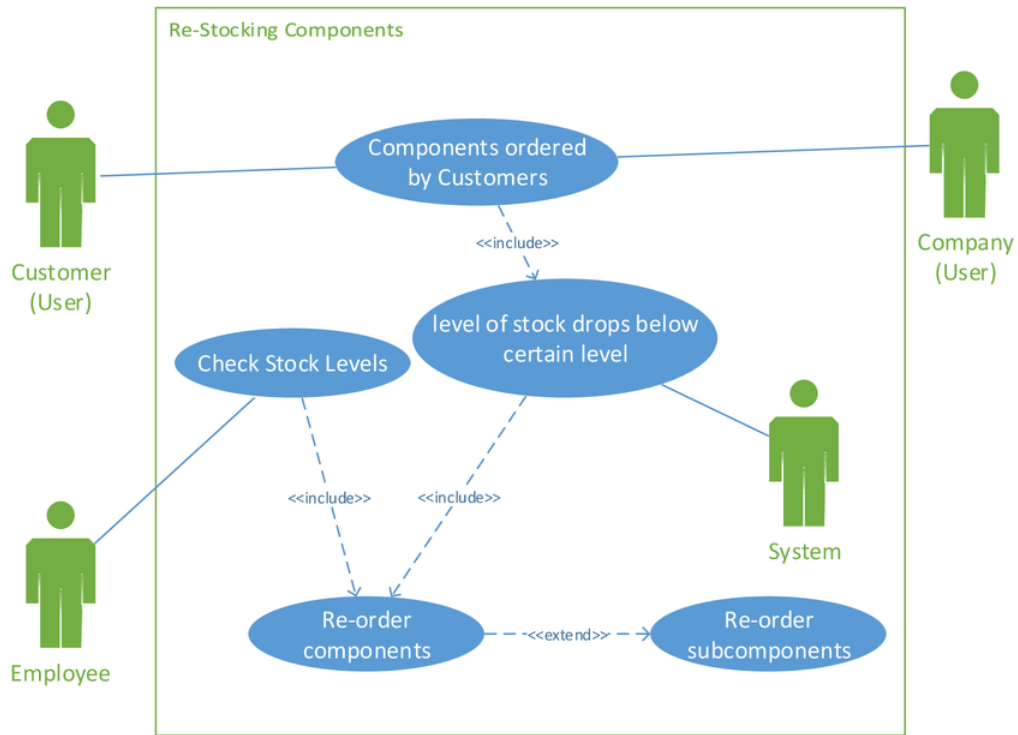


Diagram 4- Re-Stocking of Components Use-Case diagram

Sequence Diagram

A sequence diagram is used to show the flow of information within a given system and this is done with messages between users (actors) and various parts of the system.

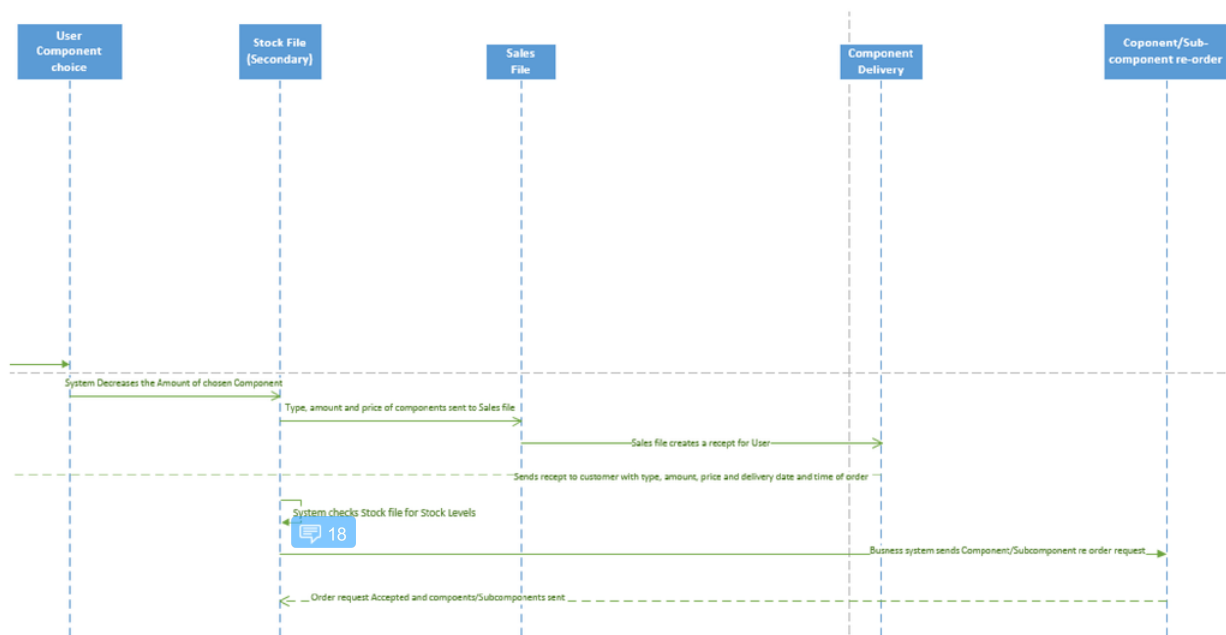
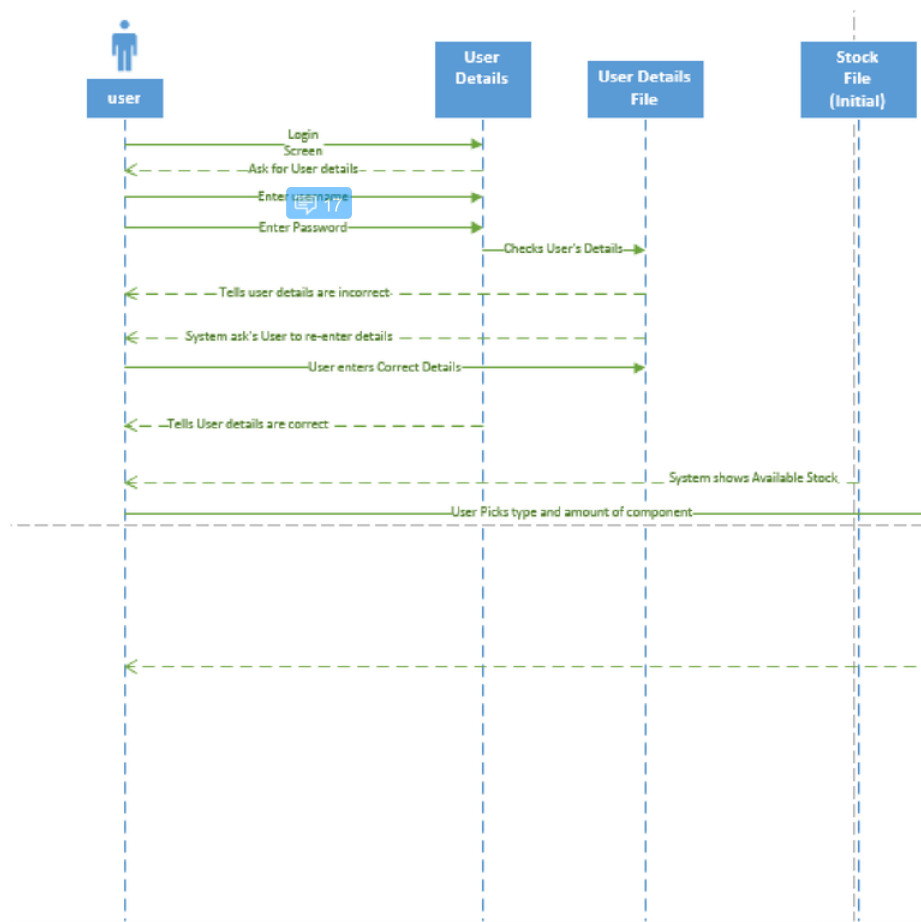


Diagram 5-System and user Interaction Sequence diagram.

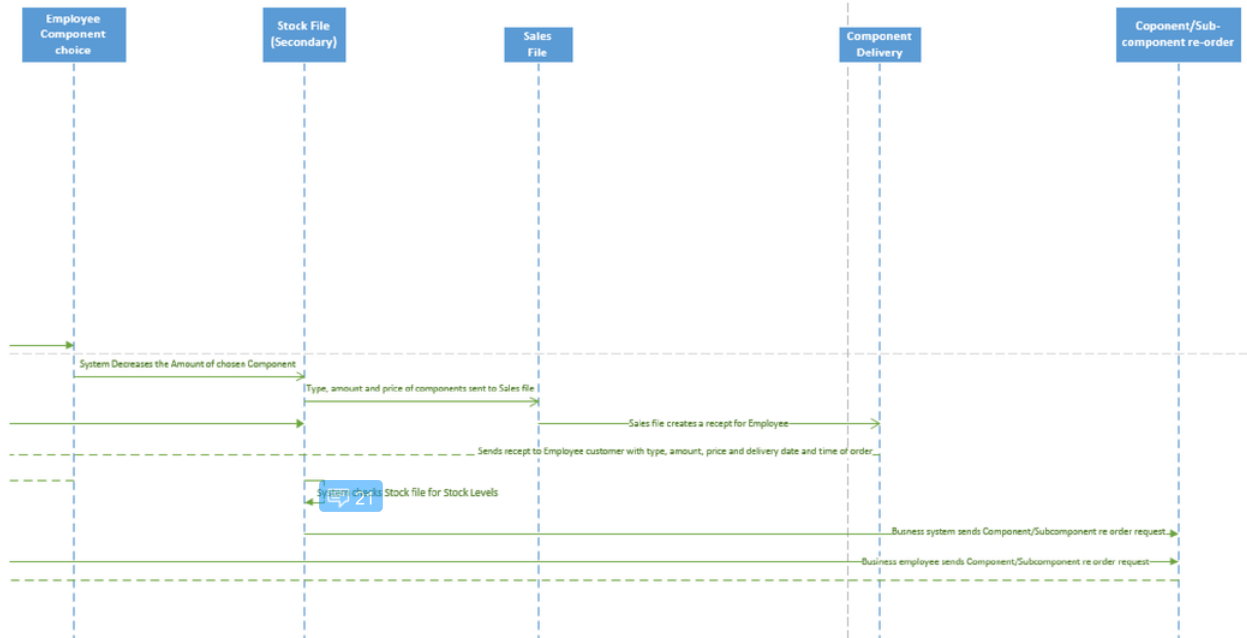
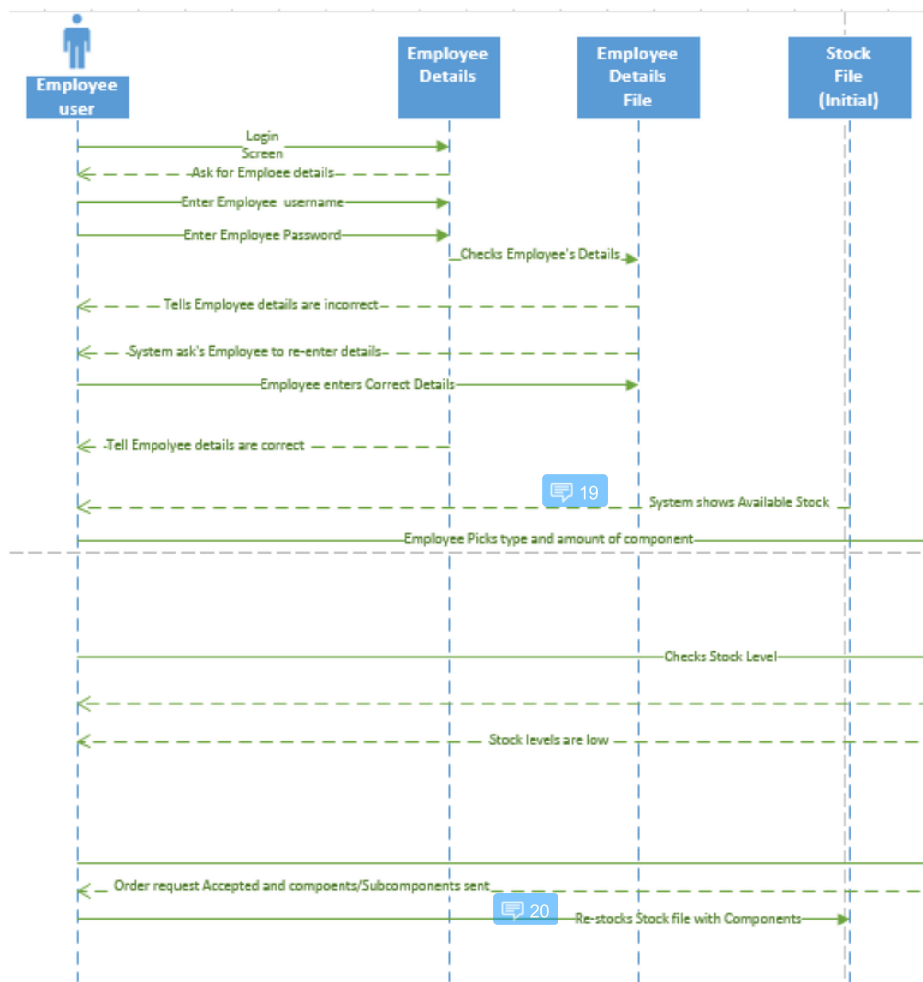


Diagram 6-Employee User and System Interaction Sequence Diagram.

References

- [1] SourceMaking.com, "Creational Patterns." [Online]. Available: https://sourcemaking.com/design_patterns/creational_patterns. [Accessed: 19-Mar-2018].
- [2] W. B. Sanders and C. Cumaranatunge, "Singleton Pattern," in *ActionScript 3.0 Design Patterns: Object Oriented Programming Techniques*, 1st ed., Steve weiss, Robyn G.Thomas, and Phillip Dangler, Eds. Sebastopol,CA: O'Reilly, 2007, pp. 101–102.
- [3] Programiz.com, "C++ Function Overriding," *Programiz*. [Online]. Available: <https://www.programiz.com/cpp-programming/function-overriding>. [Accessed: 19-Mar-2018].

OOPS Assignment 1

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

36/50

PAGE 1

PAGE 2

PAGE 3



Comment 1

Whilst classes do essentially define a data type, they include attributes and not of themselves the same



Comment 2

Shows good understanding here



Comment 3

again good understanding of the purpose and usage of files shown here

PAGE 4

PAGE 5



Comment 4

good understanding of the instantiation here



Comment 5

Careful of the language you use to describe things here, are there many or three



Comment 6

Evidence of background reading shown



Comment 7

Some understanding of the underlying concepts shown here, may have noted that an instance is created within the class definition



Comment 8

Good understanding here but use of the word encapsulation may have made the explanation clearer

PAGE 6



Comment 9

Overloading is a little more than just specifying the parameters, but having two or more function signatures holding the same name but differing parameter lists

.

You have however, evidenced this in your code



Comment 10

very good understanding shown here



Comment 11

Your initial statements regarding the concept are good however, the latter statements are a little confusing. Whilst i can see what you are trying to say it is a little misleading

PAGE 7



Comment 12

this is an accurate description of aggregation



Comment 13

Good use made of file handling

PAGE 8



Comment 14

Very good example of a class diagram, accurately represents the system created in your software

PAGE 9



Comment 15

Good attempt made however,

actors should never appear inside the system boundary

not all use-cases need to connect

PAGE 10



Comment 16

Good example, see comments for previous diagram

PAGE 11

PAGE 12



Comment 17

With the exception of the fact that you dont appear to have a userdetails class or subsystem within your software this is a good attempt at a sequence diagram



Comment 18

A file cannot necessarrily check itself , if you had a stock manager this may have better represented what you are trying to achieve

PAGE 13



Comment 19

Coud use a retrieve stock call here



Comment 20

this is another good attempt



Comment 21

See previous comment with regards to file operating upon itself

PAGE 14
