Sahara – the online retail store



13135511 – Donnchadh Murphy

14199041 – Pawel Pencherzewski

13128256 – Philip Condon

13148427 – Mark White

CS4227 Group Project

Table of Contents

[Requirements 3](#_Toc530580063)

[Outline 3](#_Toc530580064)

[Use Case Diagram 3](#_Toc530580065)

[Quality Attributes 5](#_Toc530580067)

[Tactics 5](#_Toc530580068)

[Patterns 6](#_Toc530580069)

[Factory 6](#_Toc530580070)

[Command 6](#_Toc530580071)

[Bridge 6](#_Toc530580072)

[Builder 6](#_Toc530580073)

[Façade 7](#_Toc530580074)

[Chain of Responsibility 7](#_Toc530580075)

[Interceptor 7](#_Toc530580076)

[System Architecture 8](#_Toc530580077)

[Package diagram 8](#_Toc530580078)

[Class diagram 9](#_Toc530580084)

[Cart 9](#_Toc530580085)

[Payment 9](#_Toc530580087)

[Receipt 9](#_Toc530580088)

[Store 10](#_Toc530580089)

[Command 10](#_Toc530580090)

[User 10](#_Toc530580091)

[Sequence Diagram 11](#_Toc530580092)

[Code Fragments 12](#_Toc530580093)

[Added value 13](#_Toc530580094)

[Problems encountered 15](#_Toc530580097)

[Evaluation of support for Non-Functional Requirements 16](#_Toc530580098)

[References 16](#_Toc530580099)

[Team Member Contribution 16](#_Toc530580100)

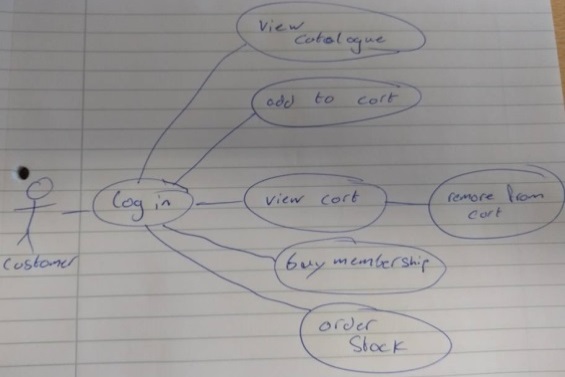
# Requirements

## Outline

The scenario we have chosen for our project is an online retail store system. The system has two levels of access, a customer and a staff member, each having different functionalities within the system. A customer user can surf the catalogue, add items to their cart, view the cart, pay for it and buy membership which applies discount on his/hers purchases. A staff member can view the catalogue, add new products, change prices and change quantities. By ordering more frequently the user will receive additional discount our project is programmed in Java using the IDE NetBeans.

## Use Case Diagram

## C:\Users\g13128256\Downloads\jj.jpg



|  |  |
| --- | --- |
| **Part** | **Description** |
| Use case | Buy Item(s) |
| Primary Actor | Customer |
| Scope | Buy item(s) |
| Brief | Item(s) add to cart, and then make purchase. |
| Preconditions | 1. Item(s) in stock. 2. Item(s) in cart. |
| Post conditions | 1. Item(s) removed from stock. 2. Cart emptied. 3. Receipt printed. |
| Triggers | The customer wants to buy item(s). |
| Basic flow | 1. Log in. 2. View Catalogue. 3. Add to cart. 4. Make purchase. |

|  |  |
| --- | --- |
| **Part** | **Description** |
| Use case | Buy membership |
| Primary Actor | Customer |
| Scope | Buy membership |
| Brief | Buy membership |
| Preconditions | 1. Doesn’t have a membership |
| Post conditions | 1. Member added. 2. Receipt printed |
| Triggers | The customer wants to buy membership. |
| Basic flow | 1. Log in. 2. Buy membership. |

|  |  |
| --- | --- |
| **Part** | **Description** |
| Use case | Add stock |
| Primary Actor | staff |
| Scope | Stock added |
| Brief | Add stock |
| Preconditions | Product exists. |
| Post conditions | Item(s) added to stock. |
| Triggers | Add stock. |
| Basic flow | 1. Log in. 2. Add stock. |

|  |  |
| --- | --- |
| **Part** | **Description** |
| Use case | Remove stock |
| Primary Actor | staff |
| Scope | Stock removed |
| Brief | Remove item from stock. |
| Preconditions | Product exists. |
| Post conditions | Item(s) removed from stock. |
| Triggers | Remove stock. |
| Basic flow | 1. Log in. 2. Remove stock. |

## Quality Attributes

Some of the quality attributes that are present in this framework are:

Extensibility - The framework supports extensibility as we could add another user, new types of product or another package, for example.

Maintainability – By using design patterns and programming to interfaces we have helped to ensure that our framework is easy to maintain.

Scalability – The framework can be scaled up to add new features like a database, adding more security features and storing more completed transactions and customer service.

## Tactics

A tactic is a design decision that influences the control of a quality attribute response. Some of the tactics present in the framework are:

Anticipate expected changes – Our framework uses this tactic as our implementation can be adapted with relative ease.

Information hiding - The tactic of information hiding is used in the receipt process where different information relating to a transaction is sent to each recipient. The User, Store and Bank all get different information relating to what is important to them.

Limit possible options – We use this tactic in our Menu system. There are two primary users that have different options available, customer and staff. Customer can view catalogue, add item to their cart, buy membership and purchase. Staff can add products, quantity and change prices. We are limiting what a ‘user’ can access depending on their activation type.

Use an intermediary - We have implemented Sahara and Command that act as intermediaries and every call will pass through either of them. This means that all classes and packages are connected in some way.

# Patterns

## Factory

The Factory is often used as the standard way for creating objects. It lets a class defer instantiation to subclasses. A superclass specifies all standard and generic behaviour and then delegates the creation details to subclasses that are supplied by the client.

Benefits

One of the most commonly used patterns, adds customizability.

Usage

The Factory is used for creating users. We have two types of users, a Customer user and a Staff user, created respectively by a customer user factory and a staff user factory.

## Command

The Command pattern is used to encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests. It supports undoable operations. It promotes “invocation of a method on an object” to full object status and an object-oriented call-back.

Benefits

Encapsulation, decoupling, promotes invocation.

Usage

The Command is used in the store to traverse through different functions of the system, such as ‘Show catalogue’ with the use of variable encapsulation, which allows you to perform a desired action.

## Bridge

The Bridge decouples the implementation from the interface, so both can vary independently. It publishes interface in an inheritance hierarchy and buries implementation in its own inheritance hierarchy.

Benefits

Run-time binding of the implementation, sharing implementation among multiple objects and mapping orthogonal class hierarchies.

Usage

The Bridge is used for the payment system, where we have a Credit Card payment type and a Bitcoin payment type.

## Builder

The Builder pattern builds a complex object using simple objects and using a step by step approach. It is a creational pattern and provides one of the best ways of creating an object. The builder is also independent of other objects. The Builder pattern separates the representation of an object from its construction, allowing to create different representations of an objects using the same construction method.

Benefits

Allows you to vary a product's internal representation, encapsulates code for construction and representation and provides control over steps of construction process.

Usage

The Builder is used in the creation of products in our store.

## Façade

The Façade pattern hides the complex parts of the system and provides an interface for which the client can use to access the system. It involves a single class which provides simplified methods required by the client and delegates calls to methods of existing system classes.

Benefits

Provide a unified interface to a set of interfaces in a subsystem and defines a higher-level interface that improves usability.

Usage

The Façade is used for the discount option in the cart. It allows users to get a different discount depending on the level of membership of the user and the number of items currently in their Cart.

## Chain of Responsibility

The Chain of Responsibility behaviour pattern creates a chain of receiver objects for a request. This pattern is used for decoupling the sender and receiver of a request. Normally each receiver contains a reference to another receiver in the chain. If one cannot handle the request, then it passes to the next receiver, etc.

Benefits

Avoid coupling the sender to its receiver by giving more than one object a chance to handle the request.

Usage

The Chain of Responsibility is used for the receipt subsystem. It will call through and return the 3 different receipts for a completed purchase. One for the User, Store and Bank.

## Interceptor

The Interceptor design pattern is used for when you want to augment or extend their cycle. It also allows for marshalling of requests. A key aspect is that the change is transparent and used automatically. This was they system does not have to change how it works; it doesn’t have to know something has happened. It uses a predefined interface, a dispatcher and context objects which allow access to the internal state.

It has 5 benefits:

Extensibility & Flexibility, Separation of concerns, Support for monitoring & control of frameworks, Layer symmetry and Reusability.

Its consequences:

Complex design issues, Malicious or erroneous interceptors, Potential interception cascades.

Usage

We use the Interceptor with the Pay for Cart function, which communicates with the payment system.

# System Architecture

## Package diagram

# hhhjkhjfgkfgfgfdlhgjdhklgjdhjalfjdklhfjdhkldhjgklhjdhdhd

# C:\Users\g13128256\Downloads\jj\package and class diagrams.png

# ksgvxnfjkfsgndfkbjnzhdkbznjb,vcnb,cvbncv,kbn

# ncjkbnbnvkbjnzvbjkzcvnb,vcnmcv,.n,.cnmcn,.cnmc,.nc.n

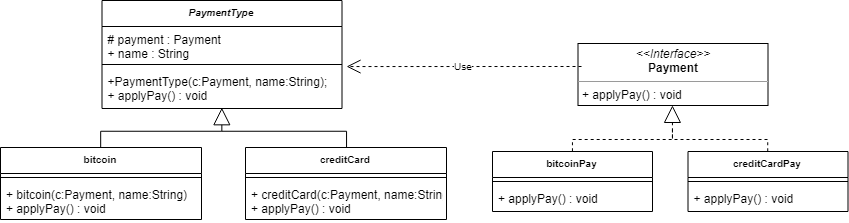
# fkljgdkfhjgdlhkhmjkghgmkhmghkmgmkdfgkbdbmdfbkbm

# Class diagram

## Cart

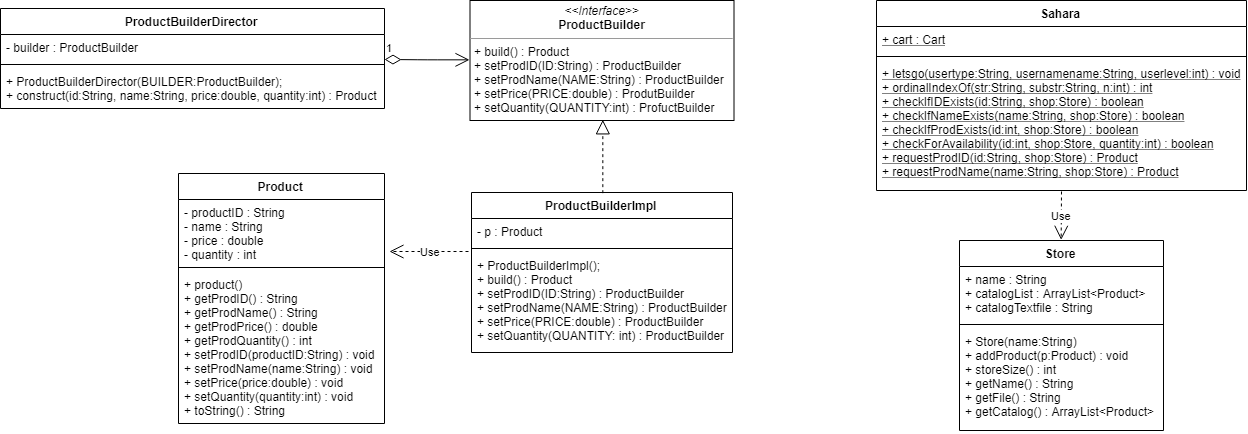
# C:\Users\g13128256\Downloads\jj\Cart (1).png

## Payment

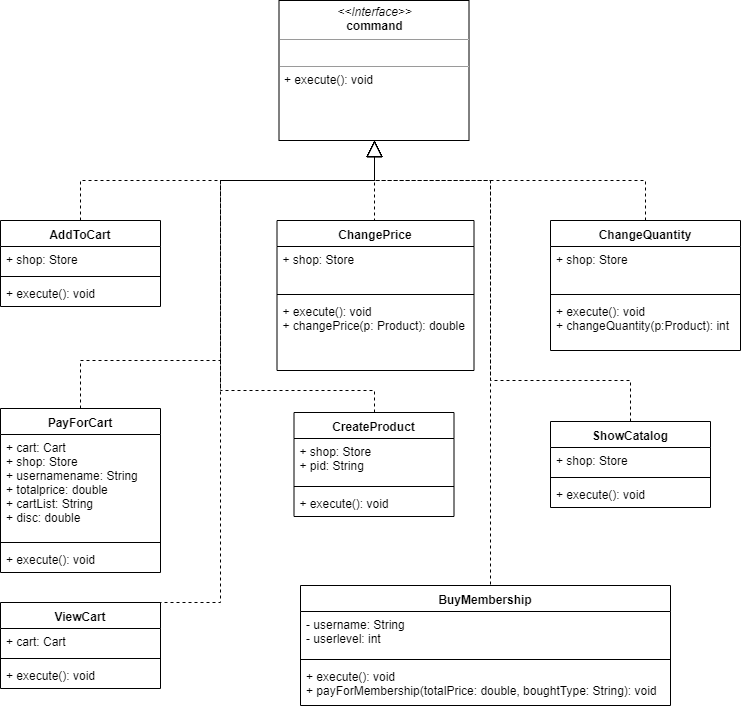


## C:\Users\g13128256\Downloads\jj\Receipt.pngReceipt

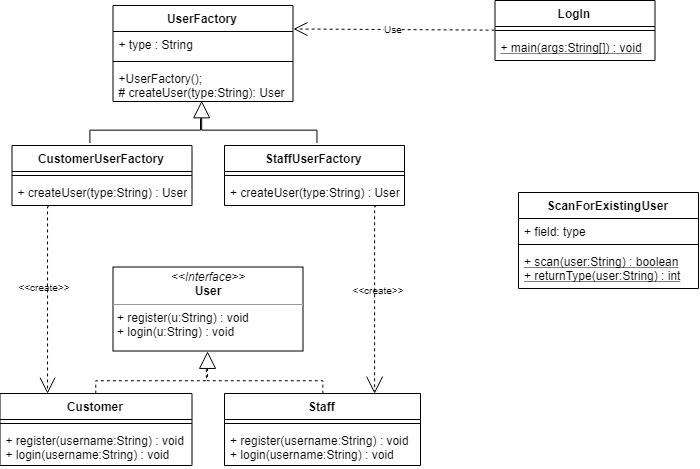
## Store



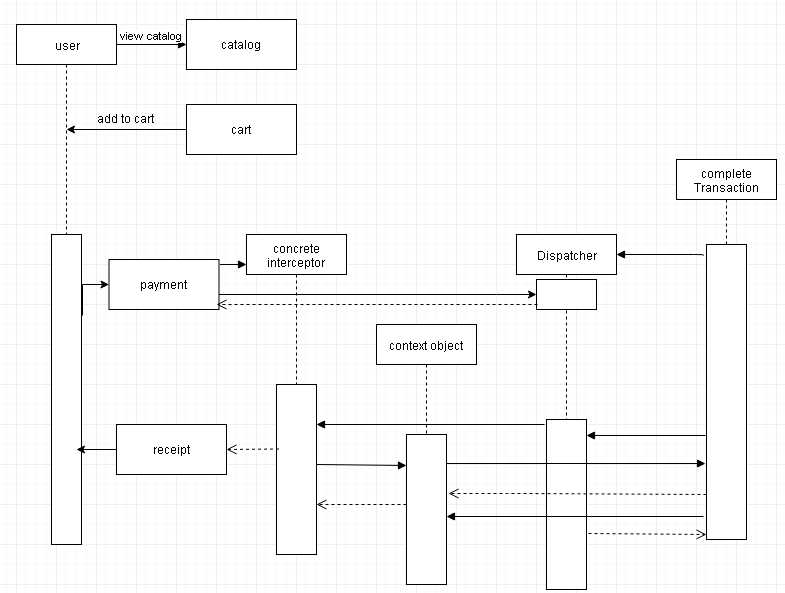
## Command



## User



# Sequence Diagram



# Code Fragments

# C:\Users\g13128256\Downloads\jj\9.jpg

Dfghgdhgh

Gdfghjgdkgj,

Fggfjhklhk;kh’

Ghdhgkjljk/;kj;’

The discount and DiscountFacade methods are particularly interesting as this was a nice feature for us to implement and made the idea come alive more for us. The implementation was difficult as it took a few builds to prefect but we are all pleased with the results.



Szghckjfkxf

Fxzgvj

Fdsggjbl;

Dfhghkj

Fgfgjl

Fdgxfgjhjk

Xhjjkblj

Gfhfhgkjl[

Fghjjho[

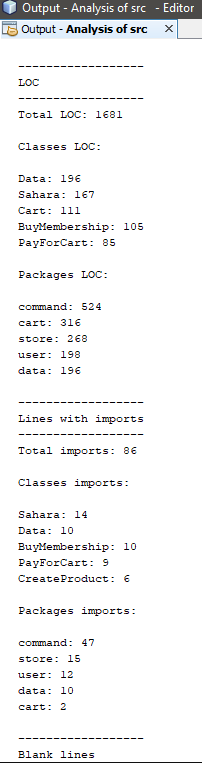
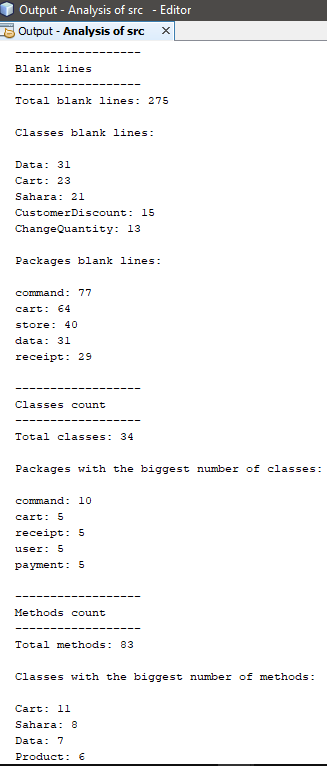
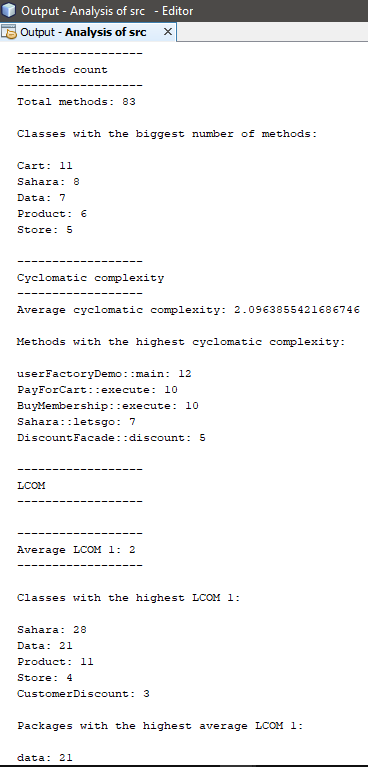
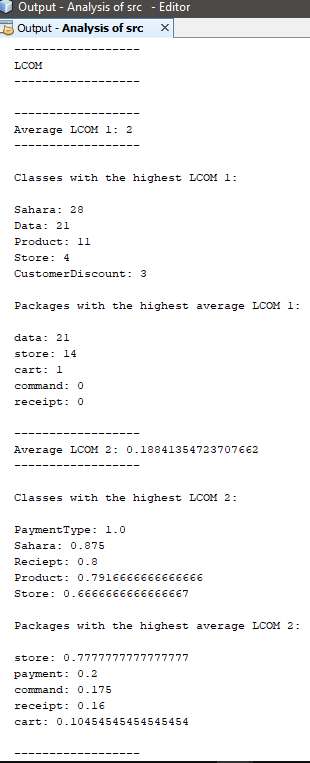
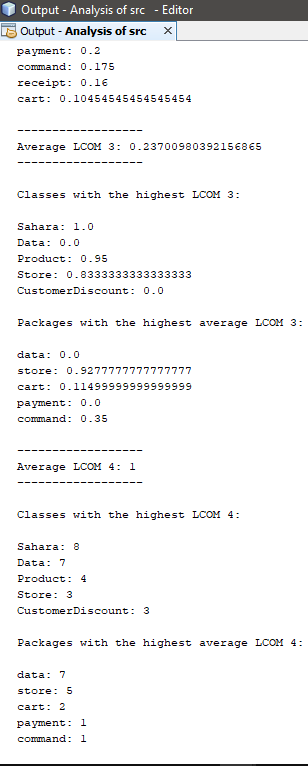
Fdghhkjk[

Gsshghjhkjfgj

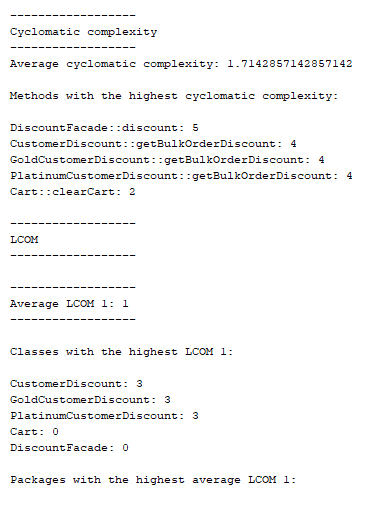
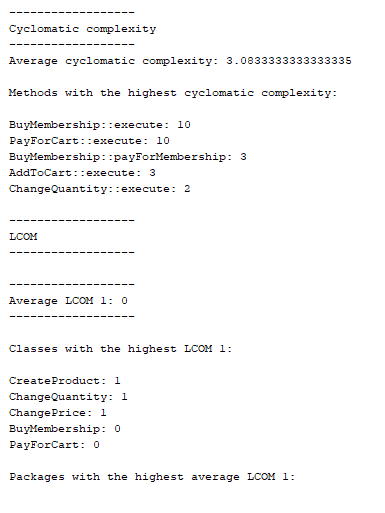
Gfgagsi

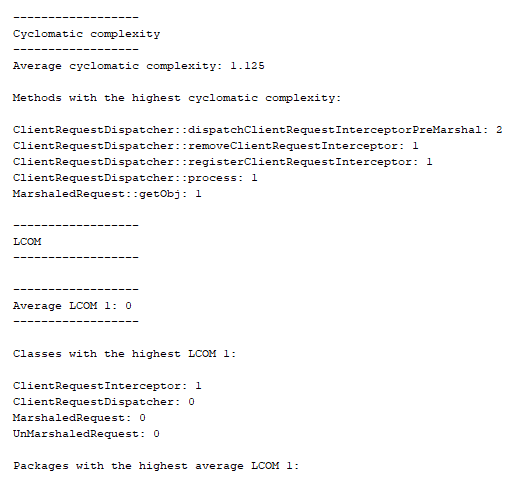
# Added value

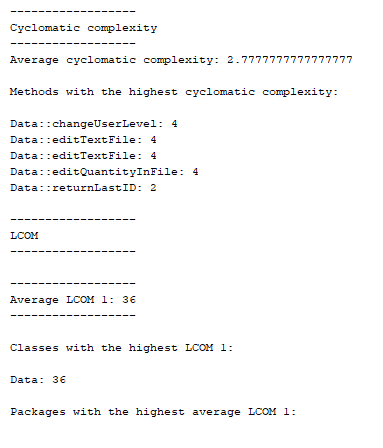
We are using Code Metrics for our Added Value. Our project is using the IDE NetBeans as we found a plugin called “Simple Code Metrics” (<http://plugins.netbeans.org/plugin/9494/simple-code-metrics>).

This is Pre utilization of the interceptor. As you can see our Cyclomatic Complexity is 2.09.

This is our Post utilization of Interceptor and the results look to have improved as you can see our Cyclomatic complexity is now 2.03. It is not a major improvement but it shows that the utilization of the Interceptor have a positive improvement.



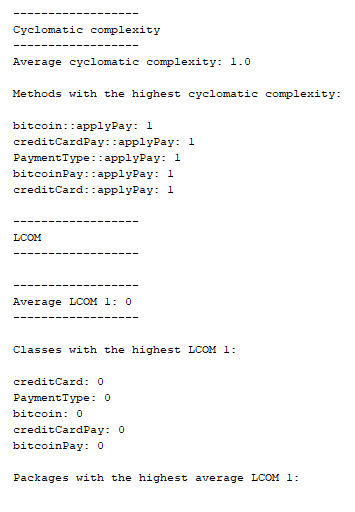
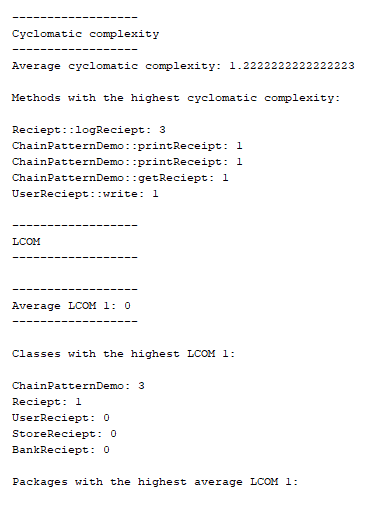




ffhgjdhjhgkgjhjgdhjghjgjkghkgj

hfjdhjhnkmdg

hbn,nmb.b,/.bm,.mvbnmcvbn



gfhxkdjgkbkhgjdghjgbkfnm

# E:\JJ project\user metrics.PNGE:\JJ project\src metric.PNG

# E:\JJ project\store metrics.PNG

# Problems encountered

One of the main problems that we have encountered was the implementation of the interceptor pattern. We had been focusing on getting the rest of the design patterns implemented before it so when we came to implementation of the interceptor, we had to find somewhere to put it that made sense and where it could be implemented correctly.

Once we found somewhere fitting, we then had to understand how the interceptor works which proved to be another challenge. It took us some time to understand how it works, and how it will work in relation to our implementation.

We had some smaller problems like initially deciding on what patterns we were going to use. This was solved quickly. Also, we had a problem with the diagrams as we were struggling to visualise the framework.

# Evaluation of support for Non-Functional Requirements

Extensibility - We have supported extensibility by programming to interfaces rather than implementation and using design patterns such as the Façade design pattern which defines a façade object to hide mask some underlying code

Scalability - In support of extensibility, we have developed our system with scalability in mind. For example, we have implemented the command design pattern to encapsulate many of the actions in our system and it allows us to create new commands to the system without having modify much of the existing code

Maintainability – The command design pattern, the factory design pattern and the builder design pattern have all helped to keep our system easy to understand and maintain.

# References

Wikipedia (2017) Global warming, available: https://en.wikipedia.org/ wiki/ Interceptor \_ pattern [accessed 20 Nov 2018, 13:42].

(Wikipedia 2017)

Tutorialspoint (2018) tutorialspoint simply easy learning, available:

<https://www.tutorialspoint.com/design_pattern/design_pattern_overview.htm> [accessed 20 Nov 2018].

(Tutorialspoint 2018)

Sourcemaking (2018) sourcemaking: online architecture learning:

<https://sourcemaking.com/design_patterns> [accessed 20 Nov 2018]

(Sourcemaking 2018)

Software Architecture by Bass et al (2010) specifically Chapter 5 on Tactics

<http://disi.unal.edu.co/dacursci/sistemasycomputacion/docs/SWEBOK/Addison%20Wesley%20-%20Software%20Architecture%20In%20Practice%202nd%20Edition.pdf> [accessed 21 Nov 2018]

The notes supplied in the lectures for reference in the selection and implementation of the design patterns.

Logo found here: [accessed on 21Nov 2018] <https://myelitedetail.us/clipart/sahara-clipart-transparent_128162.html>

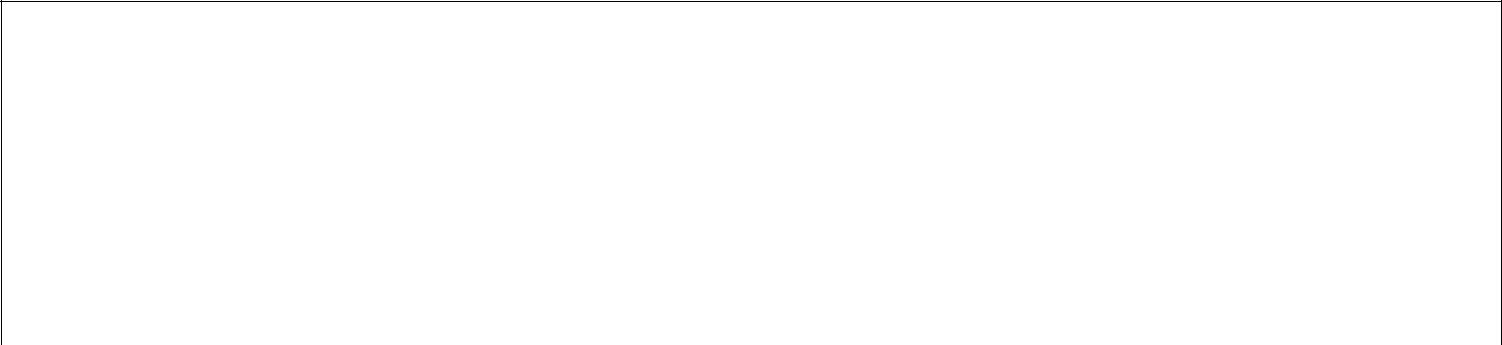
# Team Member Contribution

**CS4227: Software Design & Architecture**

**Guidance on Marking Scheme for Team-Based Project:**

**Semester 1, 2018-2019**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Name 1: Donnchadh Murphy | |  |  | ID1:13135511 |  |  |  |  |
| Name 2: Pawel Pencherzewski | |  |  | ID2: 14199041 |  |  |  |  |
| Name 3: Philip Condon | |  |  | ID3: 13128256 |  |  |  |  |
| Name 4: Mark White | |  |  | ID4: 13148327 |  |  |  |  |
|  | **Item** |  | **Detailed Description** | | **Marks** | | **Marks** |  |
|  |  |  |  |  | **Allocated** | | **Awarded** |  |
|  |  |  |  |  | Sub- | Total |  |  |
| 1-2 | Presentation |  | General Presentation | |  |  |  |  |
|  |  |  | Adherence to guidelines i.e. front cover | |  |  |  |
|  |  |  | sheet, blank marking scheme, table of | |  |  |  |  |
|  |  |  | contents | |  |  |  |  |
| 3 | Requirements |  | Narrative, Use Case diagram, and | |  |  |  |  |
|  |  |  | SAMPLE Use Case Description | |  |  |  |
|  |  |  | Discussion on NFRs and tactics | |  |  |  |
| 4 | Discussion on |  | The Interceptor architectural pattern. | |  |  |  |  |
|  | Architectural and |  | BRIEF discussion of 5 design patterns from | |  |  |  |  |
|  | Design Patterns |  | CS4227 | |  |  |  |  |
|  |  |  | Discussion on 7th pattern | |  |  |  |
|  |  |  |  |  |  |  |  |  |
| 5 | System |  |  |  |  |  |  |  |
|  | Architecture |  |  |  |  |  |  |  |
| 6 | Structural and |  | Class Diagram with package iconography | |  |  |  |  |
|  | Behavioural |  | Interaction diagram for key use case | |  |  |  |  |
|  | Diagram |  |  |  |  |  |  |  |
|  |  |  |  | |  |  |  |  |
| 7 | Code |  | Matches/Supports/Realises diagrams | |  |  |  |  |
|  |  |  | Interceptor pattern correctly implemented. | |  |  |  |  |
|  |  |  | 5 Design Patterns from CS4227 correctly | |  |  |  |  |
|  |  |  | implemented | |  |  |  |  |
|  |  |  | 7th Pattern | |  |  |  |
|  |  |  | Exposes intent, naming conventions clearly | |  |  |  |  |
|  |  |  | identify design patterns used | |  |  |  |  |
|  |  |  At least 4 packages, one developed by each | | |  |  |  |  |
|  |  |  | team member | |  |  |  |  |
|  |  |  |  |  |  |  |  |
| 8 | Added Value | Two examples, 3 marks each. | | |  |  |  |  |
| 9 | Testing |  | Design of test cases | |  |  |  |  |
|  |  |  | Automated | |  |  |  |  |
|  |  |  | Analysis of results | |  |  |  |  |
| 10 | Issues | Satisfactorily documented. | | |  |  |  |  |
|  |  | No marks awarded. | | |  |  |  |  |
| 11 | Evaluation / | Is it the case that the patterns selected | | |  |  |  |  |
|  | Critique | Supported relevant architectural use cases? If | | |  |  |  |  |
|  |  | Not, why not? Any alternatives? | | |  |  |  |  |
| 12 | References |  |  |  |  |  |  |  |
|  | Interview |  | Competent code inspection | |  |  |  |  |
|  | Week 11 or 12 |  | Working demo | |  |  |  |  |
|  |  |  | **SUB-TOTAL (A)** | |  | **60** |  |  |



**PENALTIES**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Description** | **S1** | **S2** | **S3** | **S4** |
| 1 | Late Submission |  |  |  |  |
| 2 | Failure to contribute to coding effort |  |  |  |  |
| 3 | Failure to contribute to writing of report |  |  |  |  |
| 4 | Failure to report problems with team dynamics |  |  |  |  |
| 5 | Failure to contribute to demo week 13 |  |  |  |  |
|  | **Sub-total (B)** |  |  |  |  |



|  |  |  |  |
| --- | --- | --- | --- |
|  |  | **FINAL MARKS AWARDED** |  |
| **Student1** | **Student2** | **Student 3** | **Student 4** |

**(A-B)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Package** | **Class** | **LOC** | **Author** |
| cart | Cart | 113 | Mark |
| cart | CustomerDiscount | 51 | Mark |
| cart | Discount | 22 | Mark |
| cart | DiscountFacade | 43 | Mark |
| cart | GoldCustomerDiscount | 51 | Mark |
| cart | Icart | 32 | Mark |
| cart | PlatinumCustomerDiscount | 51 | Mark |
| command | AddToCart | 53 | Mark |
| command | BuyMembrship | 112 | Pawel |
| command | ChangePrice | 50 | Mark + Donnchadh |
| command | ChangeQuantity | 53 | Mark + Donnchadh |
| command | Command | 15 | Mark |
| command | CreateProduct | 73 | Mark + Donnchadh |
| command | PayForCart | 115 | Pawel |
| command | ShowCatalog | 43 | Pawel |
| command | ViewCart | 25 | Pawel |
| data | Data | 230 | Philip + Donnchadh |
| interceptor | ClientRequestDispatcher | 36 | Mark + Philip |
| interceptor | ClientRequestInterceptor | 28 | Mark + Philip |
| interceptor | Interceptor | 17 | Mark + Philip |
| interceptor | MarshaledRequest | 26 | Mark + Philip |
| interceptor | UnMarshaledRequest | 33 | Mark + Philip |
| payment | Payment | 6 | Donnchadh |
| payment | PaymentTye | 16 | Donnchadh |
| payment | bitcoin | 14 | Donnchadh |
| payment | bitcoinPay | 9 | Donnchadh |
| payment | creditCard | 16 | Donnchadh |
| payment | creditCardPay | 9 | Donnchadh |
| receipt | BankReceipt | 13 | Philip |
| receipt | ChainPatternDemo | 50 | Philip |
| receipt | Receipt | 28 | Philip |
| receipt | StoreReceipt | 13 | Philip |
| receipt | UserReceipt | 13 | Philip |
| store | Product | 48 | Pawel |
| store | ProductBuilder | 18 | Pawel |
| store | ProductBuilderDirector | 22 | Pawel |
| store | ProductBuilderImpl | 48 | Pawel |
| store | Sahara | 165 | Pawel + Mark |
| store | Store | 34 | Pawel |
| user | Customer | 27 | Pawel |
| user | CustomerUserFactory | 18 | Pawel |
| user | LogIn | 51 | Pawel |
| user | ScanForExistingUser | 62 | Pawel |
| user | Staff | 27 | Pawel |
| user | StaffUserFactory | 18 | Pawel |
| user | User | 6 | Pawel |
| user | UserFactory | 10 | Pawel |
|  |  |  |  |
|  |  | **Total LOC** | **Author** |
|  |  | 668 | Mark |
|  |  | 302 | Philip |
|  |  | 272 | Donnchadh |
|  |  | 770 | Pawel |
|  |  | 2013 | Total |