Sahara – the online retail store



13135511 – Donnchadh Murphy

14199041 – Pawel Pencherzewski

13128256 – Philip Condon

13148427 – Mark White

CS4227 Group Project

Table of Contents

[Requirements 2](#_Toc530481112)

[Outline 2](#_Toc530481113)

[Use Case Diagram 3](#_Toc530481114)

[Quality Attributes 3](#_Toc530481115)

[Tactics 3](#_Toc530481116)

[Patterns 3](#_Toc530481117)

[Factory 3](#_Toc530481118)

[Command 3](#_Toc530481119)

[Bridge 3](#_Toc530481120)

[Builder 3](#_Toc530481121)

[Façade 3](#_Toc530481122)

[Chain of Responsibility 4](#_Toc530481123)

[Interceptor 4](#_Toc530481124)

[System Architecture 4](#_Toc530481125)

[Package diagram 4](#_Toc530481126)

[Class diagram 5](#_Toc530481127)

[Sequence diagram 6](#_Toc530481128)

[Code fragments 7](#_Toc530481129)

[Added value 8](#_Toc530481130)

[Evidence of testing 11](#_Toc530481131)

[Problems encountered 11](#_Toc530481132)

[Evaluation of support for Non-Functional Requirements 12](#_Toc530481133)

[References 12](#_Toc530481134)

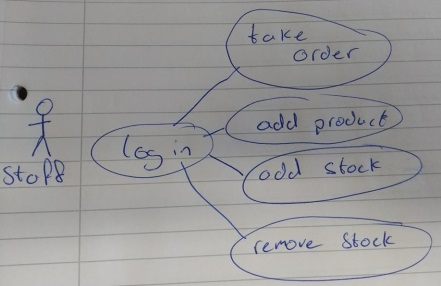
[Team Member Contribution 13](#_Toc530481135)

# Requirements

## Outline

The scenario we have chosen for our project is an online retail store system. The framework supports an end user and a staff member. An end user can surf the catalogue, add items to their cart and pay. A staff member can add products, change prices, quantities and respond to queries. An end user can subscribe to higher tier memberships which allow them to get a higher level of discount and by ordering more frequently they will also receive a discount too. Our project is programmed in Java using the IDE NetBeans.

## C:\Users\g13128256\Downloads\jj1.jpgUse Case Diagram



## 

|  |  |
| --- | --- |
| **Part** | **Description** |
| Use case | Buy Item(s) |
| Primary Actor | Customer |
| Scope | Buy item(s) |
| Brief | Item(s) add to cart, and then make purchase. |
| Post conditions | 1. Item(s) removed from stock. 2. Cart emptied. 3. Receipt printed. |
| Preconditions | 1. Item(s) in stock. 2. Item(s) in to cart. |
| Triggers | The customer want 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 |
| Post conditions | 1. Member added 2. Receipt printed. |
| Preconditions | 1. Doesn’t have a membership. |
| 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 |
| Post conditions | Item(s) added to stock. |
| Preconditions | Product exists. |
| 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. |
| Post conditions | Item(s) removed from stock. |
| Preconditions | Product exists. |
| Triggers | Remove stock. |
| Basic flow | 1. Log in. 2. Remove stock. |

## Quality Attributes

## Tactics

# 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.

Usage

The Factory is used for creating users. We have two types of user, Customer and Staff. Both have different options available.

## 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 callback.

Usage

The Command is used in the store for encapsulation and allows you to access different methods and options in it.

## Bridge

The Bridge decouples the implementation from the interface. It is also used for extensibility of the payment types.

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.

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.

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 amount of purchases they have made previously.

## 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.

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

# System Architecture

## Package diagram

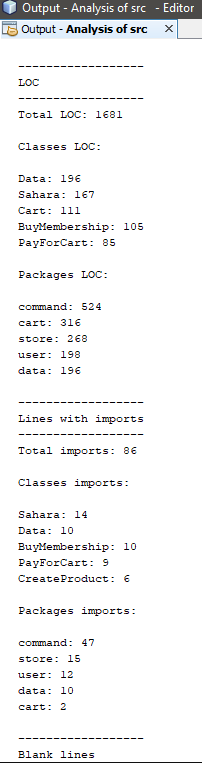
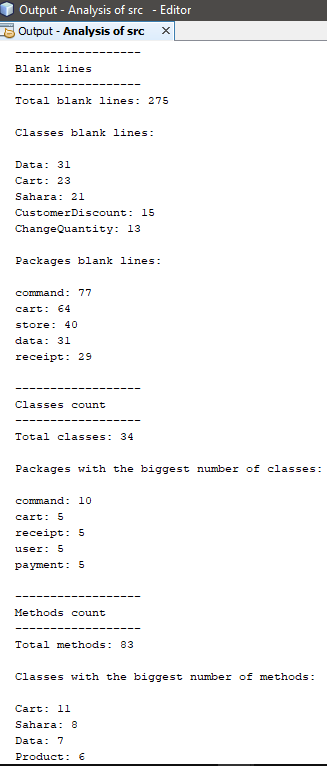
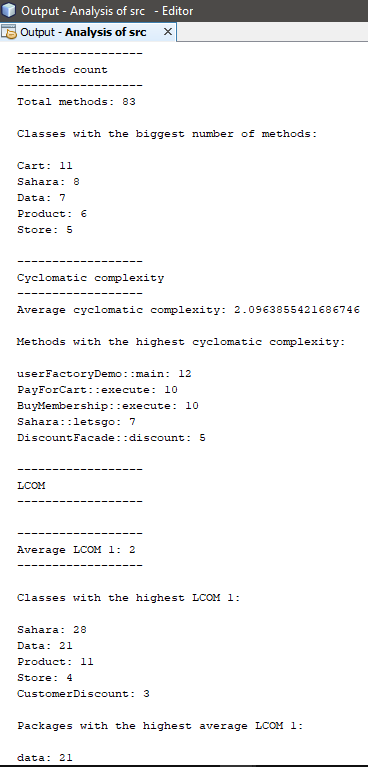
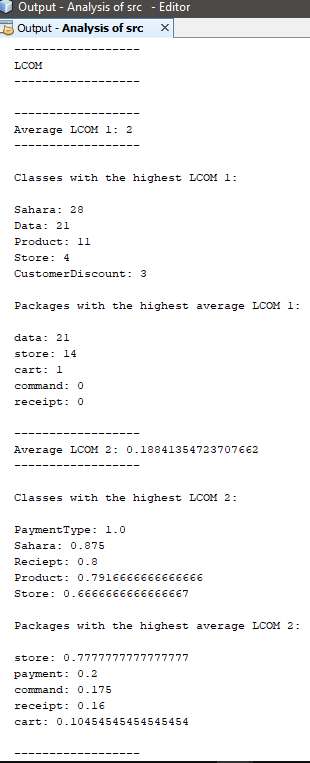
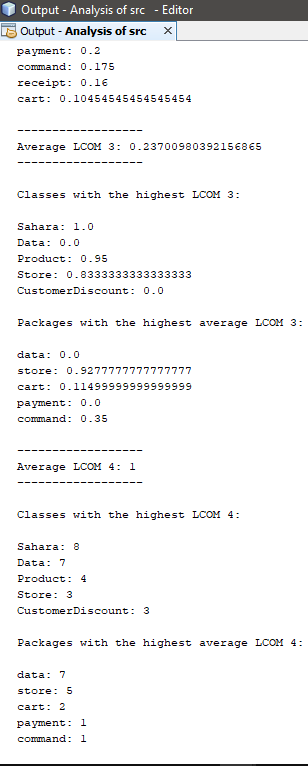
# Class diagram

# Sequence diagram

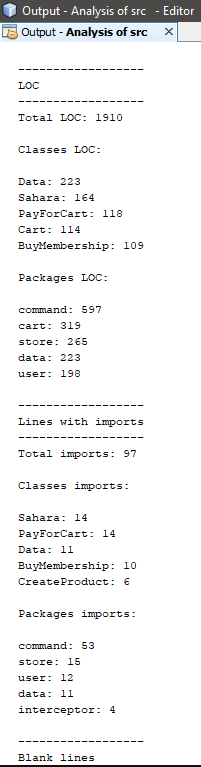
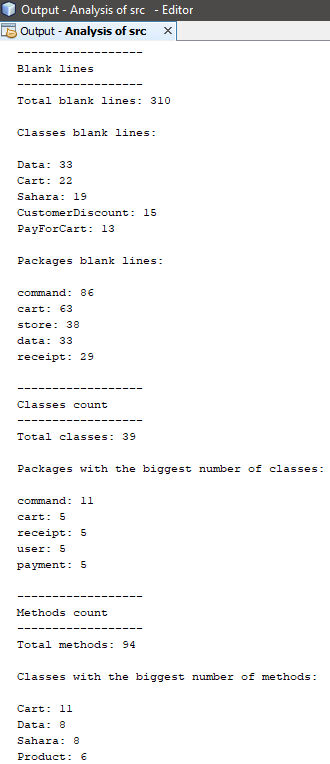
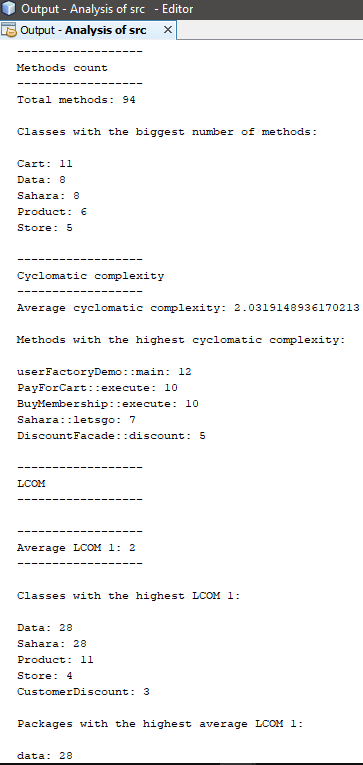
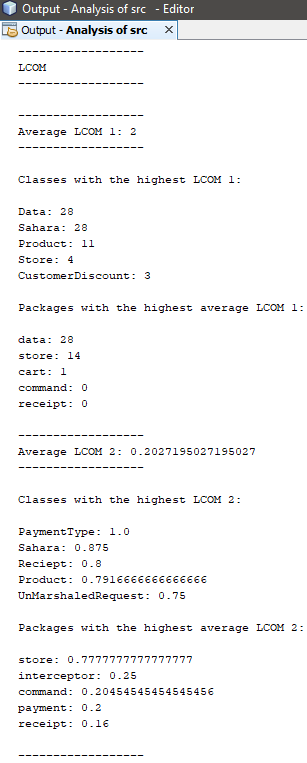
# Code fragments

# 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.



# Evidence of testing

# Problems encountered

# Evaluation of support for Non-Functional Requirements

# References

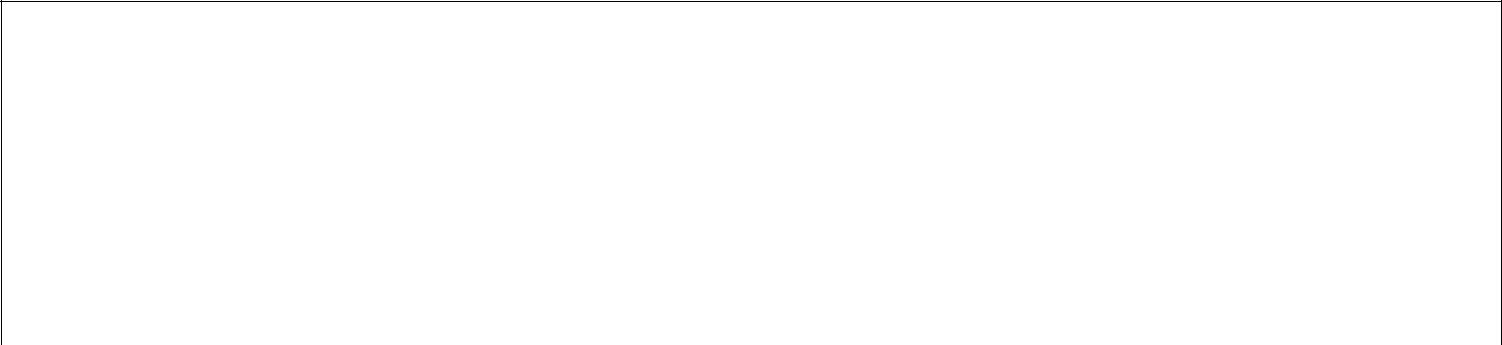
# Team Member Contribution

**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)**