CMT219 Assessment 2

Part 2

Tom McVeigh

C21108264

Contents

[The purpose of the Strategy design pattern 3](#_Toc103272133)

[UML Diagram of the programs Strategy design pattern 3](#_Toc103272134)

[Explanation of the role of each class and the changes made 3](#_Toc103272135)

[Relationships between the classes and the interface 4](#_Toc103272136)

[Reflection on the application of design patterns in Object oriented Java Programs 4](#_Toc103272137)

# The purpose of the Strategy design pattern

The purpose of the Strategy design pattern is to allow for multiple algorithms to be created and make the objects contained in the algorithms interchangeable depending on what is needed. The example from the program to be completed during this assessment is that of changing what variable the algorithm checks. changing from the most futuristic strategy to the most practical strategy, changing the variable from the user input that is being analysed from the Futuristicness of the product to the Practicality of the product, giving a different result from the selected strategy.

# UML Diagram of the programs Strategy design pattern

Diagram

Description automatically generated

# Explanation of the role of each class and the changes made

There are five classes that are in the applications these are ChoiceStrategy, MostFuturisticStrategy, MostPracticalStrategy, Product and ProductRecommender. The first class that will be covered is that of the ChoiceStrategy, this is the interface of the application and is where the chooseBetween method is initialized, this class is then implemented in the MostFuturisticStrategy and MostPracticalStrategy. No changes were made to ChoiceStrategy.

MostFuturisticStrategy and MostPracticalStrategy both have a similar role in the application, they both edit the chooseBetween method to take the input products and look at different variables stored in the class of Product. The changes that I made to this class is by implementing the if statement that looks at the input given through ProductRecommender and compares the variable of futuristicness or practicality depending on the strategy.

The next class is one of the most important one and it is Product. Product is the class that initializes the class of project and states what a Product is made up of, being the name, futuristicness and practicality, these are then taken as input to create a new Product and used in the chooseBetween to give a value. No changes were made to Product.

The last class is ProductRecommender this is the class that is run to start the application. It holds the main method and the doExample method. The main method creates a new ProductRecmmender and tells it to start the doExample method. This method creates two products of p1 and p2, then it prints out messages informing the user on what is the currently selected strategy. The changes that I made for this class was changing the instance to first be set to MostFuturisticStrategy, creating a new product called result1 that stored the result of the choseBetween method that took the previously created p1 and p2 as input. This result was then compared against p1 in regards to the name. If the name given by the result was equal to the name stored in p1 then p1 is returned as the most futuristic Product, otherwise p2 is returned. This is the same for MostPracticalStrategy, the only differences are that the strategy is changed to MostPracticalStrategy and the result is stored as result2.

# Relationships between the classes and the interface

The interface for this program is that of ChoiceStrategy. The main related classes are that of MostFuturisticStrategy and MostPracticalStrategy as they directly implement ChoiceStrategy to be able to use the important method of chooseBetween that the two classes are based on. Without ChoiceStrategy would not function and the whole application would suffer as a result. The other class that relates to the interface is that of the ProductRecommender. ProductReccomender uses the interface to allow for ChoiceStrategy to be referred to as myStrategy as a global variable in the class allowing for it to be changed in the doExample method and for the other strategies to be used.

# Reflection on the application of design patterns in Object oriented Java Programs

The application of the design patterns for object oriented Java programs allow for programmers to follow a structure that ensures that code developed is developed to a high standard and any problems could possibly be fixed before they impact the project as a whole.

There are three main types of application design patterns, creational, behavioural and structural. These each specify the process of creating different types of objects and patterns. Creational patterns specify the how the objects of the program are created and help the developers in reusing code that has been previously developed to suit a new purpose. Behavioural patterns main purpose is to provide communication between objects. Structural patterns state different ways in which different class structures to be created.

These patterns allow for developers to integrate complex concepts in to the java programs that they create. Implementation of these patterns can improve the speed at which development is carried out on the programs in question and improve the quality of the programs created with patterns.