CMT219 – Part 2 Report

A) The purpose of Strategy design pattern is to define a family of algorithms, all invoked in the same way, so algorithm can vary independently of clients that invoke it, which enables selecting an algorithm at runtime.

B) Figure 1 shows a UML class diagram representing the resulting Strategy design pattern in this program.

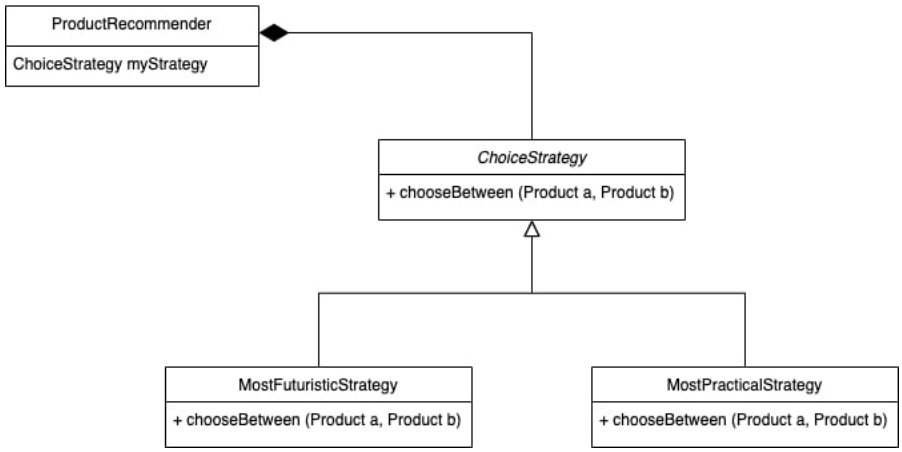

Figure 1. UML class diagram representing the resulting Strategy design pattern.

C) **The role of each of the classes.** There are totally five classes.

ChoiceStrategy: this class is a public interface, which provides a general method chooseBetween(Product a, Product b), choosing one Product between the two inputs Product a and Product b.

MostFuturisticStrategy: this class implements chooseBetween(Product a, Product b) in a specific way, which returns Product a if its futuristicness is greater than or equal to that of b; and returns Product b otherwise.

MostPracticalStrategy: this class implements chooseBetween(Product a, Product b) in another specific way, which returns Product a if its practicality is greater than or equal to that of b; and returns Product b otherwise.

Product: this class is to define a Product object, which contains attributes including name, futuristicness and practicality.

ProductRecommender: this class provides context of calling chooseBetween methods. It contains a general variable ChoiceStrategy myStrategy, to be instantiated later according to specific uses.

The changes made in realising this design pattern.

The two classes MostFuturisticStrategy and MostPracticalStrategy implement two different version of the method chooseBetween, which are both invoked in the same way. Depending on the instantiation of ChoiceStrategy myStrategy in the ProductRecommender class, calling chooseBetween method will be determined at runtime.

The relationships between these classes and the interface.

Both of the MostFuturisticStrategy and MostPracticalStrategy classes inherit ChoiceStrategy, and they implement the general method chooseBetween(Product a, Product b).

The Product class serves as the input and return type for the chooseBetween method.

The ProductRecommender class contains a variable of ChoiceStrategy myStrategy, which is to be instantiated later to call the chooseBetween method.

D) Reflection on the application of design patterns in OO Java programs.

The strategy design pattern can be naturally used in OO Java programs by defining abstract interfaces and general methods, without implementing details. This abstract class and method will serve as the template for a family of algorithms (ChoiceStrategy class and chooseBetween method in this case).

Then subclasses can be defined to inherit the general interface and implement specific versions of algorithms according to different needs (MostFuturisticStrategy and MostPracticalStrategy classes here).

A context class (ProductRecommender class here) can contain general class variables (ChoiceStrategy myStrategy), and which version of algorithms is called totally depends on which subclass is instantiated.