RMIT UNIVERSITY

Assignment name: Individual Project

COSC2440 – Further Programming

Lecturer name: Dang Tran Tri

Tutor name: Dang Tran Tri

Subject code: COSC2440

Campus: Saigon South

Date of submission: 2nd April 2023

Student Name: Do Quang Thang
ID: S3891873

Table of Contents

1. OOP Principles

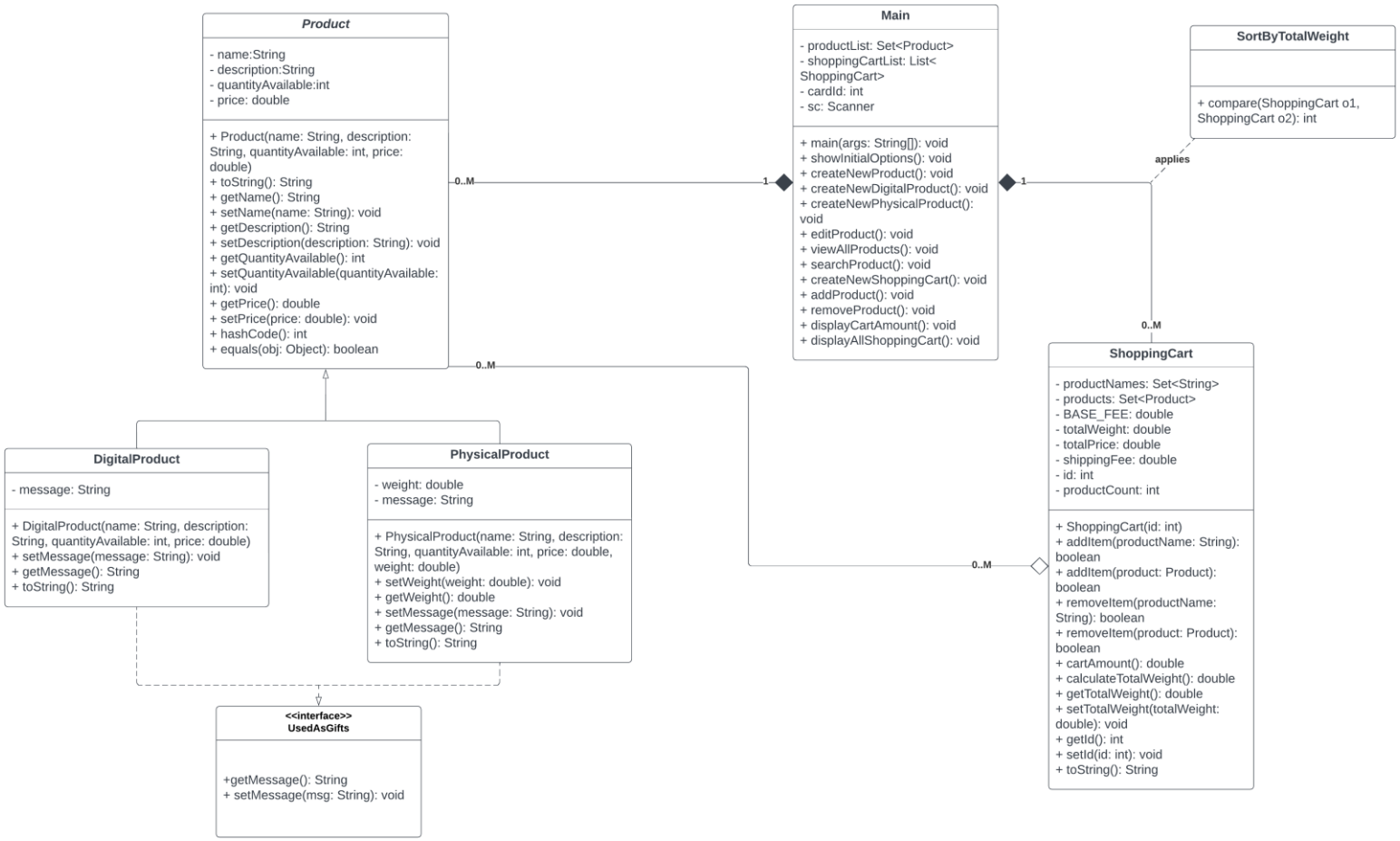
Object-oriented programming is one of the most popular styles of application development in the world. OOP principles make the software development process much simpler and easier. With that in mind, in this project I have applied 4 important OOP principles throughout the implementation process, including abstraction, encapsulation, inheritance, and polymorphism.

- ***Abstraction***: abstraction can be understood as hiding unimportant information and showing only the required features. For example, in this project, when the program is running, the user only needs to interact with the interface via input without knowing the logic in the code. In programming, specifically the Java language, abstraction can be achieved through the abstract keyword. My Product class is declared as an abstract class and implements an abstract method totring to ensure the principle of abstraction in OOP. In addition, the UsedAsGifts interface along with the two methods getMessage and setMessage also contribute to achieving abstraction. As a result, users only need to implement them without worrying about internal implementation.
- ***Encapsulation***: encapsulation can be defined as the encapsulation of attributes or methods in a single module. These attributes or methods will have limited access to the outside world and can only be accessed from internal components. In this project, every attribute in the Product, ShoppingCart, DigitalProduct, PhysicalProduct, and Main classes have access modifiers that are private and the corresponding getter and setter methods are also properly implemented. As a result, external classes cannot arbitrarily access these attributes, but must call the corresponding getter function. This ensures the encapsulation of the whole program.
- ***Inheritance***: Inheritance can be thought of as a child class absorbing all the properties of the parent class and combining the properties of the child class itself. In this project, the Product class is designed as a parent class that has two child classes, DigitalProduct and PhysicalProduct. Both these derived classes inherit all attributes and methods from the Product class such as name, description, quantityAvailable, etc. However, PhysicalProduct can still add an attribute which is weight to match its inherent properties. This design minimizes code repetition as the same properties do not need to be declared multiple times.
- ***Polymorphism***: polymorphism can be understood as many forms. A particular object or method with many different names can be considered polymorphism. In this project, I applied polymorphism both to objects and methods. For object, every time I

declare an instance of DigitalProduct or PhysicalProduct, I declare them according to the type of Product. This makes the program more flexible later on. For method, this program has achieved polymorphism through the use of method overloading and method overriding. First, I used the method overloading over the addItem and removeItem functions. Therefore, the system will automatically implement the appropriate method if I call this method with a product object or with a String name. Also, throughout the project, I have used method overriding such as toString, hashCode, equals, setMessage, and getMessage. Thus, when a child object calls these functions the system will call the overridden methods.

* The reason I passed an entire Product object to the cart was that I realized the program needed access to other attributes of the Product class, not just the product name. Even if we have the product name as a reference, the program needs a list of product objects to search for. I was thinking of using the Map interface to manage the product name and the corresponding object, however, this probably requires a middle class which in my opinion, is not sensible as embedding an entire object into the cart.

2. UML Class Diagram



The entire code skeleton of the project is described in the Class Diagram above:

Description: As we can see, there are 7 entities appear in the diagram, which includes 6 classes (Product, DigitalProduct, PhysicalProduct, SortByTotalWeight, ShoppingCart, and Main) and an interface UsedAsGifts.

- First, the Product class is declared as an abstract class. Therefore, no instance of Product is allowed to exist in the system. Product includes 4 attributes(name, description, quantityAvailable, and price) with access modifier private to ensure encapsulation of object-oriented design. Accordingly, I have implemented setter and getter methods corresponding to each attribute. This class also has an abstract method called toString so that subclasses can override it later. In addition, this class overrides two methods from the Object class, which are hashCode and equals. These methods decide how to compare two Product objects (in this case, by name).
- Next, the UsedAsGifts interface defines a contract with some product types that determine whether they can be used as a gift. This interface includes two methods setMessage and getMessage.
- The Product class has two main subclasses, DigitalProduct and PhysicalProduct. In other words, the relationship between these two classes and the Product class is inheritance. These two classes also implement the UsedAsGifts interface, so they have quite similar properties. DigitalProduct carries the same attributes and methods as the parent class and also overrides three methods including setMessage, getMessage, and toString. With PhysicalProduct, in addition to the attributes and methods similar to DigitalProduct, it also adds an attribute called weight. As a result, there are two more methods, setWeight and getWeight.
- Another important class is the ShoppingCart class. This class includes 8 private attributes (a set of productNames, a set of products, BASE_FEE, totalWeight, totalPrice, shippingFee, id, and productCount) and the corresponding setter and getter methods. In addition, there are three more important methods including addItem (adding a specific product and its name to the cart), removeItem (removing a specific product and its name from the cart), and cartAmount (calculating the total cost of the whole cart). Since this class includes a set of products, its relationship with the Product class should be Aggregation and the cardinality should be many to many because a shopping cart can contain zero or more products and a product can also belong to zero or more shopping carts.

- The most important class of the entire program is the Main class. This class acts as the entire shopping service class associated with the user interface. There are a total of 4 private attributes in this class including a set of products(productList), a list of shopping cart(shoppingCartList), cardId, and sc(Scanner). In addition, this class also implements a series of methods that represent user actions, which we will see more clearly in the Screenshots section below. The relationship between Main class and Product class is Composition because this class represents the whole system, once this class is deleted, the entire product in the system will also be deleted. The cardinality is one to many because a product belongs to a single system and a system can own many different products. Similarly, the relationship between Main and the ShoppingCart class is also Composition one to many.
- The last class is SortByTotalWeight. This class implements the Comparator interface and overrides the compare method to perform comparisons between shoppingCarts based on their totalWeight value. This class uses the ShoppingCart object as a parameter so their relationship can be Association. As for the Main class, the relationship between them is Composition.

3. Screenshots of solution

My project guides users step by step through specific instructions, the result of the project is as follows:

When the program starts running, a menu like the one above will appear, the user needs to select one of the actions by inputting the corresponding number. I do not validate user input so let's assume all user input is valid. Every time the user completes an action flow, the program will also automatically return to this menu.

```
Creating new physical product....
Please enter product name:
laptop
Please enter product description:
Acer Nitro 5
Please enter product available quantity:
10
Please enter product price:
100
Please enter product weight:
1.5
Please enter product message (optional):
no Message
laptop added successfully!
```

These are the steps to create a physical product, users need to enter all necessary and reasonable information to succeed. For digital products, the same process applies.

```
Please enter product name to edit:
laptop
Start edit product information....
Please enter new product description(name cannot be changed):
Acer Nitro 6
Please enter new product available quantity:
12
Please enter new product price:
100.5
Please enter new product weight:
1.6
Please enter new product message (optional):
Happy birthday!
The product is edited successfully!
```

The second function of the program is to edit product information. The user needs to enter the name of an existing product and start changing the information. Note that the product name cannot be changed.

```
Physical - Name: laptop        Description: Acer Nitro 6            Available Quantity: 12        Price: 100.5          Weight: 1.6           Message: Happy birthday!
##########################################
```

With the third function, users can view all products that exist in the system as shown above.

```
Please enter a product name:
laptop
************************************
Physical - Name: laptop        Description: Acer Nitro 6            Available Quantity: 12        Price: 100.5          Weight: 1.6           Message: Happy birthday!
************************************
```

The fourth function allows users to search for a certain product by product name. The user needs to enter the name of an already existing product to be successful.

```
5
Shopping Cart ID: 1 has been created successfully!
```

The 5th option will automatically create a new shopping cart with an id number starting from 1 (new cart followed by 2 and so on). This function requires no user input.

```
6
Please enter shopping cart ID to add:
1
Please enter product name to be added into the cart:
laptop
The product: laptop has been added successfully!
```

The 6th function is to add a specific product to the cart. The user needs to enter cart Id first (in this case 1) and then the name of an existing product.

```
8
Please enter the cart ID to check total price:
1
The total price of this cart is: 100.66
```

Next, the 8th option will allow the user to see the total amount of a particular cart. The user needs to enter an already existing cart id to succeed.

```
9
Shopping Cart ID: 2            Total weight: 1.4            Number of product: 1
Shopping Cart ID: 1            Total weight: 1.6            Number of product: 1
```

Function number 9 allows the user to view a list of all shopping carts in the system. This list is sorted by the total weight of each cart (the cart with the smaller weight comes first).

```
7
Please enter shopping cart ID to remove:
1
Please enter product name to be removed from the cart:
laptop
The product: laptop has been removed successfully!
```

The last function (number 7) is to remove the product from the shopping cart. The user needs to enter the id of a specific cart then enter the name of the product that exists in that cart. Note, this function only deletes products from the cart, not from the system.