Course Code	COSC2440
Course Name	Further Programming
Location and Campus	Saigon South Campus
Title of Assignment	Group Project
Group Name	Group 9
Group Members	Nguyen Anh Duy (s3878141) Pham Minh Hoang (s3930051) Kang Junsik (s3916884) Yoo Christina (s3938331)
Lecturer	Tri Dang Tran
Submission Date	9 May, 2023
Number of Pages	8

I. Goals and objectives

The objective of this group project is to extend the Java console application for a shopping service that was done individually. This application allows users to view, create, and edit products, view, create, and edit carts, set gift messages for the product, apply, remove coupons, and print receipts of the carts. The application is designed with an object-oriented approach, utilizing classes and interfaces.

This report will provide an overview of the available features, the UML diagram which illustrates the design structure of the system, a detailed explanation of the coding techniques and design patterns implemented.

II. Scope and Features

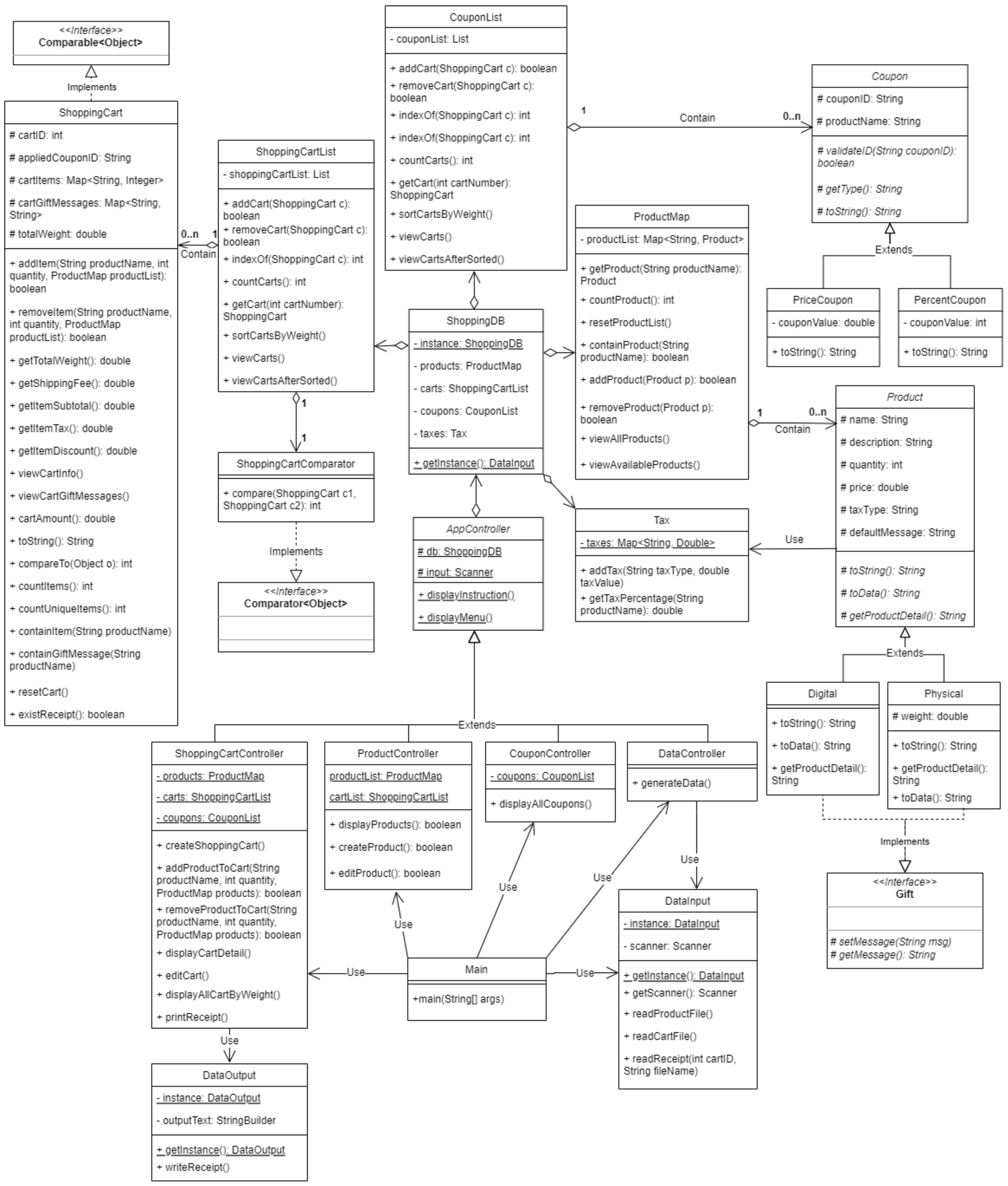
The extended Java shopping service application has the following features:

- View products: Users can view all products (digital/physical with different tax types).
- Create new products: Users can add new products to the system.
- Edit product: Users can edit the product information.
- Create a new shopping cart: Users can create new shopping carts.
- Add and remove products: Users can add and remove items in the cart.
- Display shopping carts: Users can view all shopping carts in the system.
- Set gift message: Users can set gift messages for a product in a cart.
- Apply and remove coupons: Users can apply or remove a coupon (price/percent type for a specific product) from the cart.
- Get a receipt of the cart: The user can view the final price of the cart with all conditions (tax, coupon discounts) applied to finalize the cart.

Out of scope (features that are not included):

- Writing to database files, changes made when using the program aren't saved.
- Addition and deletion of coupons, as well as assigning coupons to different a different product during system usage
- Removal of shopping carts.

III. Class Diagram



IV. Design and Implementation

1. Design pattern and the architecture

Regarding the system architecture, to mitigate some issues, the classes and files are organized into relevant groups. The packages here represent the feature that the classes or the files are associated with to help the structure visually look convenient and coherent for accessing and modifying. There are 3 main packages: controllers, database, and models. The controllers stored the functions that can deliver the features of the system, the database stored all the necessary data for the controllers to retrieve and execute, and finally the models are all the required classes needed in the system architecture.

With a detailed focus on the code design, the system has implemented the Singleton, Iterator, and Factory Method. The input, output, and database classes use Singleton pattern to ensure only one instance of these classes can be instantiated, since these classes are responsible for the data flow and create multiple instances would be extremely redundant and costful for the memory, especially, we only need one database only to store and retrieve data from to avoid misleading data. For further clarification, the DataInput class stores a scanner, and DataOutput class stores an output attribute of StringBuilder type, so it can be reused again in the system to mitigate the cost of creating a new instance. The Iterator pattern is applied for iterating the carts in the ShoppingCartList to avoid the undergoing data structure which is used, and it will not affect the iteration process if the structure is adjusted in the future. Lastly, The Factory Method pattern is used for Product and Coupon classes which use abstract parent classes and have different child classes and based on different occasions, the appropriate class will be instantiated to align with the system needs.

To address other minor issues on the design concept, the coding techniques that have been applied include the usage of wrapper classes, utilities methods, and the models + controllers approaches. In particular, the wrapper classes are the manual classes such as ProductMap, ShoppingCartList, and CouponList that store a suitable abstract data type to store the values, for instance, a TreeMap for the product items and quantity in a cart, or an ArrayList for the coupons. The database does not store directly these data types such as List, Map, or Set but store these wrapper classes, in order to create a flexible design and avoid revealing the implemented data structure, as well as the stored data type in these wrapper classes can be modified conveniently in the future. For the utility methods, these functions are constructed to reduce duplicated codes such as checking for a data existence or counting. Finally, the models and controllers packages are partially adopted from the MVC design, to divide the features of the whole Main class into relevant areas (controller), to easily append/remove/adjust the functions when needed, without interfering with the main class. Moreover, including all the functions to the Main would create a chaos composite relationship with all the other parts and can make the class overwhelmed with a lot of complex functions, therefore, it is more approachable if there are multiple controllers to manipulate the flow of the system, and the main class is only the final controller to call for the needed feature to be executed.

2. Classes and Interfaces

(For clarification, the UML class will not contain the general getters and setters of the classes)

In general, according to the class diagram, the overall relationship between each class can be illustrated with different combinations of association and aggregation relationships. The ShoppingDB are associated with the ProductMap, ShoppingCartList, CouponList and Tax classes because the database must have a location to store data after retrieving them from the text files. Having said that the database would not fully operate without these attributes, however, to create a flexible design, association relationship would be preferable compared to compositions. Moreover, the AppController also uses the data retrieved from the database to execute the features, therefore, it is also strongly associated with the ShoppingDB. For the Main class, it uses all the required components to run the system, hence, all the four controllers and DataInput are associated with Main class The other aggregations relationships are between the class container and its model to store. For example, the ShoppingCartList stores the ShoppingCart, the ProductMap stores the Product, and the CouponList stores the Coupon. Due to the limitation of the UML class diagram, there are more classes that are associated with each other. As a clarification, for instance, not only the DataController that used the DataInput class but also the DataOutput and all the controllers when asking for inputs from users would associate with this class. With regards to the further classes implementation details, the application is using the following approaches:

Class / Interface	**Design implementation**	**Description**
ProductMap & ShoppingCart	TreeMap: for productList, cartItems, and cartGiftMessages	Map a productName with the corresponding value (Product instance, quantity in the cart, or gift message in the cart). TreeMap allows user to search for items with case insensitive.
ShoppingCartList & CouponList	ArrayList: for cartList and couponList	Store all the created shopping carts and coupons and easily to extract information
Tax	HashMap	To store all the tax values with the tax types. Can easily append and adjust in the future if there are more tax types
AppController, Product & Coupon	Abstract (parent) classes	The system will decide which controller to use, or which product/coupon should be created based on particular scenario
Gift (Interface)	Set and get a default gift message for a gift product	Gift interface is implemented to Physical and Digital products, can be implemented to other products in the future if needed.
Comparable & Comparator (Interface)	Compare shopping carts by weights	For sorting the shopping carts based on their total weights in ascending orders
DataInput and DataOutput	try, catch InputOutputException	For reading the text files and writing the cart receipts

Looking into the methods, most of the required return types will be boolean, unless these methods are displaying messages to the console or getting/calculating a value. For instance, adding, removing, and updating actions in the system will have a boolean result to indicate whether it is successful. Even though the usage of boolean values might not be too identical in the scope of this system, for the future implementation, having a status to ensure the success of the feature is extremely important to produce extra input/output or alerting messages to users, if the system can be further enhanced. For users to view the information, due to the scope of the application, there are no separated view packages or view classes, but every message's output is controlled by the controllers, since the message is output differently according to each feature. Furthermore, even for input validations, there are identical differences when checking for the item existence or a valid number, therefore, there are no separated utilities methods for validating, but it is controlled by the controllers.

V. Testing

1. Automatic Testing

There are a total eight test classes created for automatic testing. Contents are provided in the following table:

Test Classes	Tested methods
ProductMapTest	- countProduct(): returns the number of products in the map - resetProductList(): clears all the products in the map - containProduct(): returns a boolean value based on the existence - removeProduct(): removes the selected product from the map
DigitalTest	- toString(): returns a formatted string representing a digital product - setMessage(): set a gift message to the product - getMessage(): returns the gift message of the product
PhysicalTest	- toString(): returns a formatted string representing a physical product - setMessage(): set a gift message to the product - getMessage(): returns the gift message of the product
CouponTest	- validateID(): checks if the ID of the coupon is valid - getType(): determines the type (percent/price) of the coupon based on the ID's last character ('a' or 'b')
PriceCouponTest	- toString(): returns a formatted string representing a price coupon
PercentCouponTest	- toString(): returns a formatted string representing a percent coupon
ShoppingCartTest	- addItem(): add a product to the cart - removeItem(): removes a product from the cart - getItemDiscount(): returns the discount based on the applied coupon - getItemTax(): returns the tax based on the tax type percentage
ShoppingCartListTest	- addCart(): add a cart to the cart list - removeCart(): remove a cart from the list - getCart(): returns a cart from the list

2. Manual Testing

Several methods were tested by manual testing. Contents are provided in the following table:

Class	Tested methods
ProductMap	- viewAllProducts(): displays all products in the system in alphabetical

	order. Information includes the type (digital/physical), name, remaining quantity per product. - viewAvailableProducts(): similar to above method, but only displays products with their remaining quantity larger than 0.
Product (both Digital and Physical)	- getProductDetail(): displays every detailed information of the selected product. Information includes type, name, quantity, tax type, weight, and gift message.
ShoppingCartList	- viewCarts(): displays all carts in the system. Information includes cart id with the products in it. - viewCartsAfterSorted(): same with viewCart but is in ascending order
ShoppingCart	- viewCartInfo(): displays the information of the selected cart. All the detailed information is given as the applied coupons, total weight, etc. - viewCartGiftMessages(): displays the gift messages saved in the cart.
CouponList	- viewAllCoupons(): displays all coupons in the system. Information includes id, type (price/percent), discount amount, and its product. Order is in Physical to Digital and then alphabetical.
DataOutput	- writeReceipt(): displays the receipt of the selected cart. It includes a receipt number, date, and the username. Information includes the list of products, purchased quantity, single price, price calculated by quantity. At the bottom, the total item number and the final price including subtotal, discount and tax is displayed.

VI. Conclusion

The system provides a console application for a shopping service that was successfully extended with additional features, such as applying coupons, tax types, data input methods, returning receipts including all the conditions. The application's object-oriented and organized architecture allows easy testing, modification, and integration of features. For further implementation, this application can be extended with additional features that is not covered in the project's scope, as well as some essential functionalities, for instance, registration for accounts, more product and coupon types, having a delivery system for the shopping cart, and finally, writing to products.txt and carts.txt after all the modification from the user in the future.