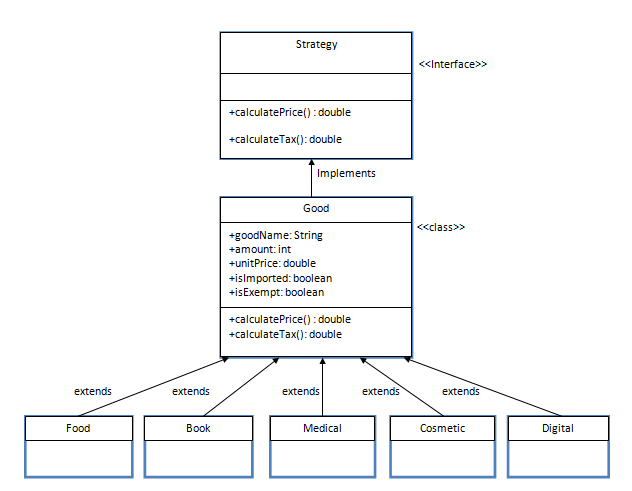
**Sales taxes problem**

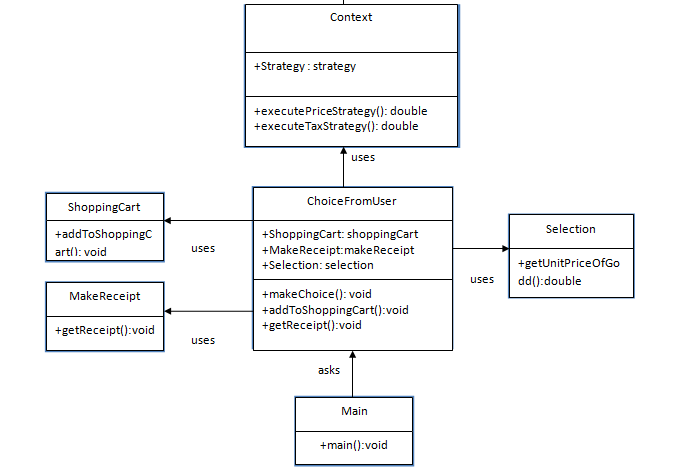
This program is created using JAVA programming. In this program, it is used Strategy Design Pattern to solve sales taxes problem. Strategy design template is used to select and apply a method according to the situation. Each method (algorithm) is implemented for a class. More clearly, the behavior or algorithm of a class at runtime can be changed according to a strategy. In order to apply the strategy design template, we need to define an interface named Strategy. Within this interface, there are methods to be implemented by subclasses.

Firstly, It is created an interface named Strategy and defined 2 important functions for all subclasses. These functions are calculatePrice() and calculateTax() functions. Then It is created a general class named “Good” that implements the Strategy interface.

Good class is a main class with several product subclasses inherited from it. Each subclass overrides our main functions of Good class that are inherited from Strategy interface. And each subclass has its own unique values for the common variables (e.g. unitCost, goodName.. etc.)

In the below, you will see the UML Diagram of the Project. The UML diagram explains code flows and class relationships.



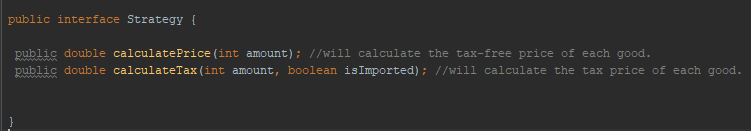
 uses

**Created Components**

**src/Main**

**Strategy.java**

This is the Strategy interface that has total price and tax price functions. It has 2 functions.

****

**src/Goods**

**Good.java**

It is the main class of all products. It implements Strategy interface. In this class, common variables such as goodName, unitCost, amounts, isImported, isExempt has defined.

* ***goodName:*** Every product has a product name.
* ***unitCost:*** It specifies unit costs for every product.
* ***amounts:*** It specifies how many/how much the user will buy the choosen product.
* ***isImported:*** It specifies if the product is an imported product. If yes then this variable returns true.
* ***isExempt:*** There is no sales tax for exempt goods (such as medical, food, boks). This variable defines if a product is exempt or not.

**+calculatePrice(int amount): double**

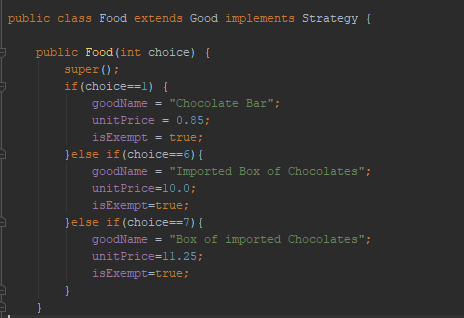
This function is inherited from Strategy interface and returns amount\*unitPrice for each product. It does not contain sales tax cost.

**+calculateTaxPrice(int amount, boolean isImported): double**

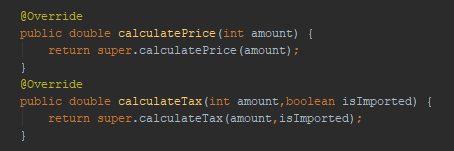
This function is inherited from Strategy interface and returns sales tax cost for each product looking if it is an imported and if it is an exempt product.

**Food.java**

In the food class there are 3 options for chocolate types. Whichever type of chocolate the user wants, the variables specific to that type, work in the construction. Every type has a different unit cost and good name. The “choice” variable is the variable that shows what product the user chooses.

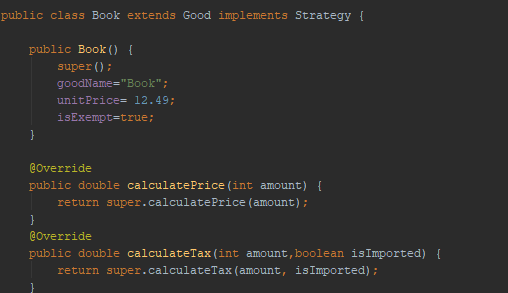


Food class overrides our 2 important functions.



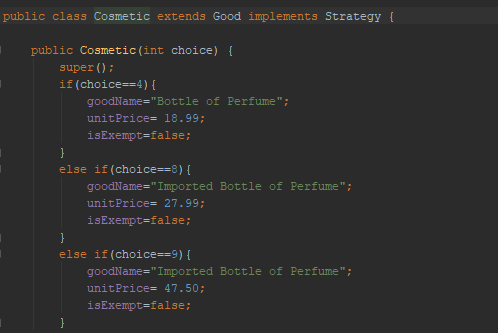
**Book.java**

In the Book class, unique values as goodName, unitPrice, isExempt have specified. The class overrides calculatePrice() and calculateTax() functions. In this class isExempt variable is true because there is no sales tax for books.



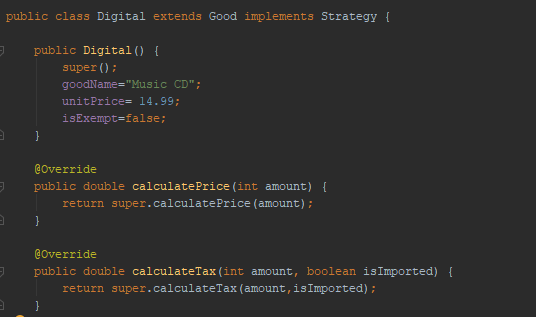
**Cosmetic.java**

Like the food class, there are different choices for perfume in the cosmetic class, and they are specified in the constructor. Whichever type of perfume the user wants, that type returns. This class also overrides calculatePrice() and calculateTax() functions.

****

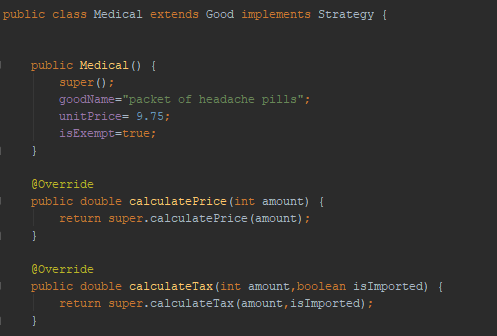
**Digital.java**

In this class, unique values as goodName, unitPrice, isExempt have specified for music CD. This class also overrides calculatePrice() and calculateTax() functions.



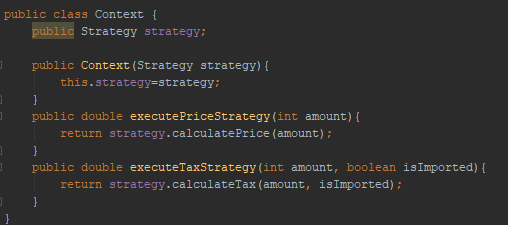
**Medical.java**

In this class, unique values as goodName, unitPrice, isExempt have specified for box of headache pills. This class also overrides calculatePrice() and calculateTax() functions.



**src/Main**

**Context.java**

Context class has been defined to run the related strategy through the Interface.

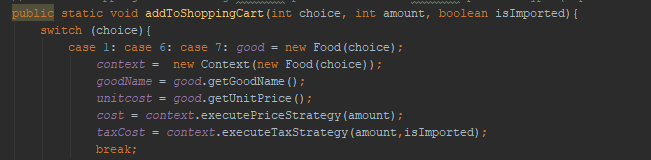
Now we are ready to create shopping cart, takes products from user and calculates their prices and tax prices.

**src/ShoppingCart**

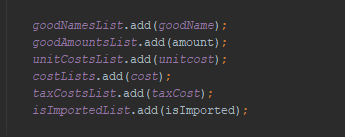
**ShoppingCart.java**

The aim of this class is to create a shopping cart with choosen products and calculate their prices. In this class, I created context object to calculate prices for every product. I created ArrayLists for keeping choosen products, their amounts, prices and their imported informations from the user. In this class, there is one function named addToShoppingCart(). This function takes inputs from user and returns the corresponding class using context object. For every choosen product, the created ArrayLists keep their values.

For example when the user selects chocolate option, the “choice” variable comes as 1 and first case runs. It reaches food class using context object.

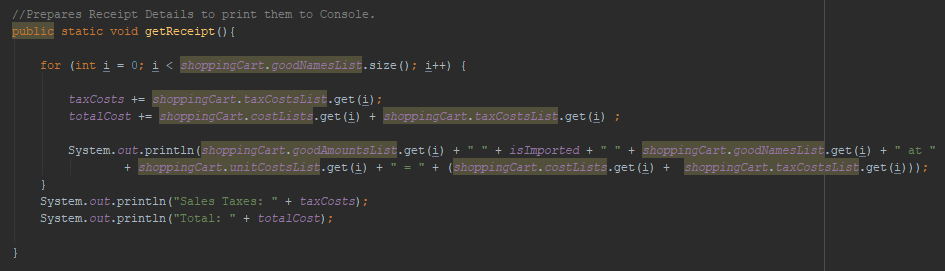


At the end of the function, ArrayLists are created.



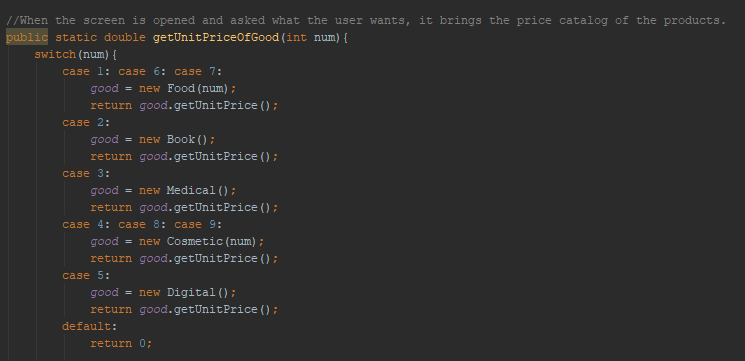
**MakeReceipt.java**

This class has getReceipt() function , it gets ArrayList values and print them as a receipt of the shopping. This function also calculates total costs and tax costs of the shopping.



**Selection.java**

This class has one function named getUnitPriceOfGood(). When the program is run, it asks the user which product he/she wants to buy and brings the unit price of the products. To do that, this function returns unit prices of goods.

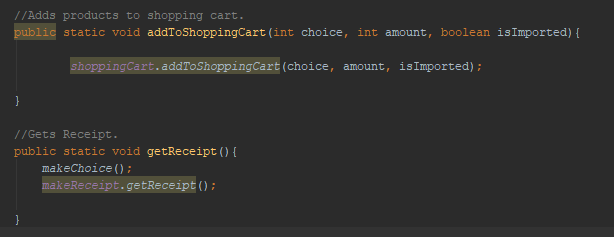


**ChoiceFromUser.java**

This class has 3 functions named makeChoice(), addToShoppingCart() and getReceipt().

makeChoice() function asks to user what he/she wants as a product and how much he/she wants from them. The function takes values and add them to shopping cart by calling ShoppingCart class. This event takes place in the addToShoppingCart() function.

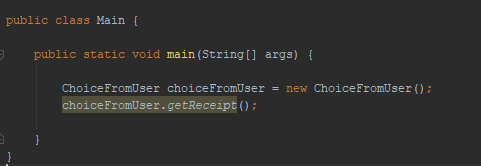
getReceipt() function returns the created receipt.



**Main/Main**

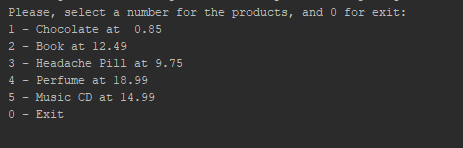
**Main.java**

In main method, it asks ChoiceFromUser class and gets the receipt when shopping is finished.

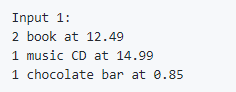


**How to Run the Program**

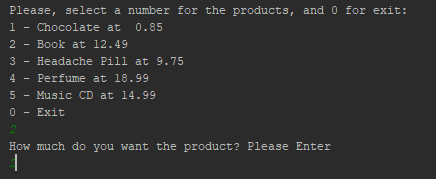
When the program is run, the product catalog is displayed.



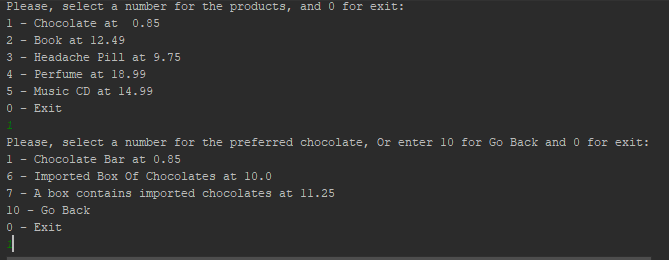
For the first input,



The user enter “2” for the Book selection. And the programs asks how much the user wants from that product. After entering the amount, the program goes on with asking other products until the user enters 0 to exit. This will be the same also in music CD selection.

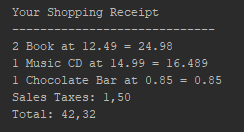


When the user selects chocolate. A chocolate catalog is displayed and asks the user which type of chocolate and how much he/she wants.



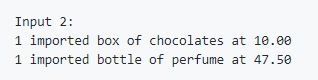
In chocolate katalog, there are 3 type of chocolates and each of them has different unit prices.

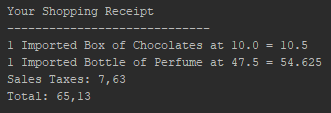
After all the selections, the user enter “0” and the receipt is diplayed.



The program is displayed fot other inputs too.

**For Input 2:**





**For Input 3:**

