CSE – 443 OBJECT ORIENTED ANALYSIS AND DESIGN

MIDTERM REPORT

OĞUZHAN SEZGİN

QUESTION 1

In this part of midterm I used **Abstract Factory Design Pattern**. Because client wants to product family. There are 3 types of phone and their ingredients may change by their model also their region.

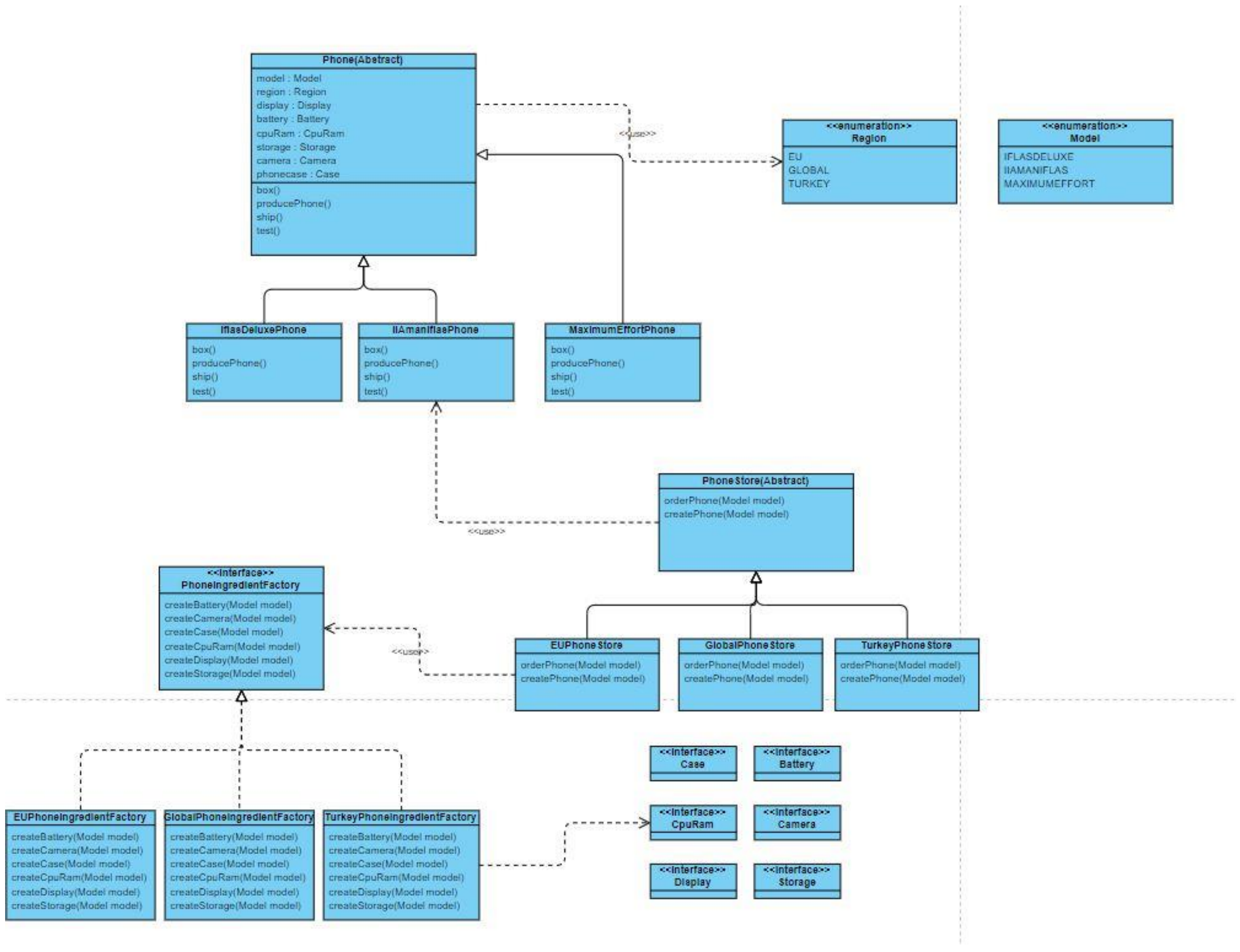
I created a abstract **PhoneStore** class. This class places phone order by user and call **createPhone** method and then test , box and finally ship the phone. Because of the properties of ingredients can change from the region, all region has their own **PhoneStore** class. This classes extends **PhoneStore** class. This classes create phone by the their region properties so they implements their **createPhone** method.

The local store classes also has a local factory classes. These factory classes create ingredients by their region ingredients and for this reason all phone has their region property. The local factory classes implements **PhoneIngredientFactory** interface.

I also created **Model** and **Region** enum classes . User choose his/her region and phone model from these enum classes.

Phones extends abstract **Phone** class. They are created according to their region and model.

The class diagram showing below :



QUESTION 3

In this part of midterm I use **Command Design Pattern**. I created Command interface for make transaction . Transaction classes implements this class by their process. If the transaction require int parameter(e.g. increase money by X) it use execute(int) parameter or require string parameter (e.g. add Y property) it use execute(String) parameter. This execute method ,calls **store** method. This method create the transaction class with use private constructor. This constructor takes a update value and if they **execundo** method called , they do the opposite of what you did before according to update data. The store method push this classes to process stack which in the **BankAccount**.

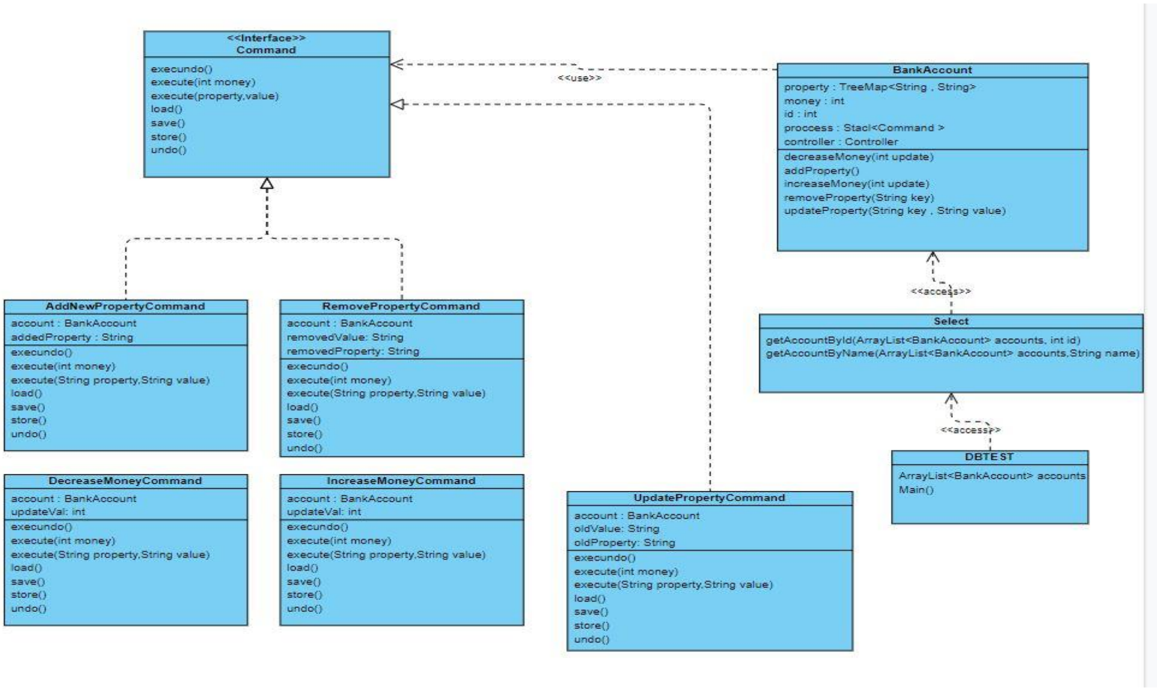
BankAccount class represents database member. It has id, money and other property column. Other property may change by user alter operation. User can add or remove property. This class also has process and controller.

Process attribute is a stack. This stack stores all transactions for making the account until the user save changes. This stack store Command type If user wants to **undo** transactions , the process stack is popped and call **execundo** method so the transaction undo one step and if user wants to undo again and the stack popped and execute again. This operation can continue until stack become empty. If user saved transactions , process stack is emptied and user can not undo until new make new transactions. If user make wrong transaction and he/she did not save before , the **load** method called which in the command interface, and process stack pop and execute until the stack become empty.

Controller class has Commands. User can make all transactions with use this class. All accounts has their Controller class and user control account with this controller class.

User can get the account from database with use **select** class. This class has two method and it can get user by id or name with this methods.

The class diagram showing below :



QUESTION 4

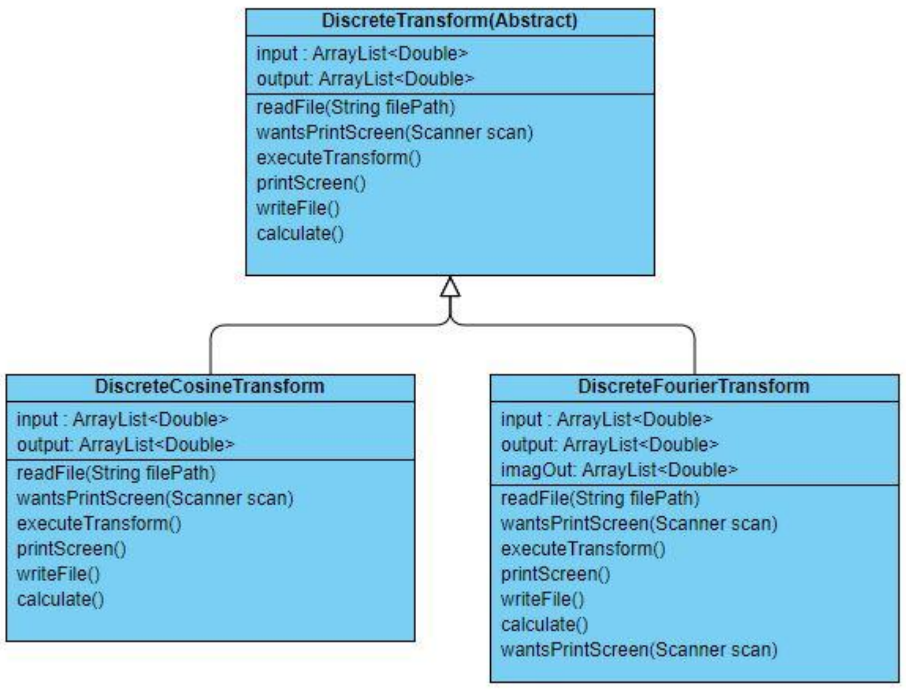
In this part of midterm I used **Template Design Pattern**. I created abstract **DiscreteTransform** class. **DiscreteFourierTransform** and **DiscreteCosineTransform** extends from this class. **DiscreteTransform** implements **readFile** and **executeTransform** methods because of this methods are common. **ExecuteTransform** calls other methods by order , it represents the general algorithm of the program.

Calculate and write methods implemented by concrete classes. They implement this methods by their calculation and their output.

wantsPrintScreen.method is a **hook** methods. It returns false by default but the **DiscreteFourierTransform** class ask the user to print results on screen ,so **DiscreteFourierTransform** override this method. User decides print screen process. On the other hand **DiscreteCosineTransform** does not ask the user because it does not override **wantsPrintScreen**.

The outputs are written to "**result.txt**" file..

The class diagram showing below :


DiscreteTransform(Abstract)
input : ArrayList<Double>
output: ArrayList<Double>
readFile(String filePath)
wantsPrintScreen(Scanner scan)
executeTransform()
printScreen()
writeFile()
calculate()
DiscreteCosineTransform
input : ArrayList<Double>
output: ArrayList<Double>
readFile(String filePath)
wantsPrintScreen(Scanner scan)
executeTransform()
printScreen()
writeFile()
calculate()
DiscreteFourierTransform
input : ArrayList<Double>
output: ArrayList<Double>
imagOut: ArrayList<Double>
readFile(String filePath)
wantsPrintScreen(Scanner scan)
executeTransform()
printScreen()
writeFile()
calculate()
wantsPrintScreen(Scanner scan)

All of these packages has their own Tester class.

Tester classes tests all methods of their package class.

Javadoc document is in doc file(index.html).