Gebze Technical University

Computer Engineering

Object Oriented Analysis and Design

CSE 443

MIDTERM REPORT

Değer MANDAL

161044096

Important: I used “jdk-14.0.2” in my homework.

Part1

Abstract Factory Design Pattern

I used the Abstract Factory design pattern for the problem because of the problem asks us to make a more detailed change on the level of materials.

Display, Battery, CPU&RAM, Storage, Camera, Case Interfaces: I used the interfaces to the smartPhone materials. I write a toString() method and that will return the features of ordered smartPhone in subclasses.

MaximumEffortSP, IflasDeluxeSP, I_I_Aman_IflasSP: The classes extended from SmartPhone abstract class. Also take SmartPhoneComponentFactory reference that used the to market features. PrepareSP() method combines phone and market features to create the phone.

TurkeySmartPhoneComponentFactory, EUSmartPhoneComponentFactory, GlobalSmartPhoneComponentFactory: The classes implements from SmartPhoneComponentFactory and using to create suitable market feautures.

TurkeySmartPhoneStore, EUSmartPhoneStore, GlobalSmartPhoneStore: The classes extends from SmartPhoneStore and using to decide which smartPhone ordered.

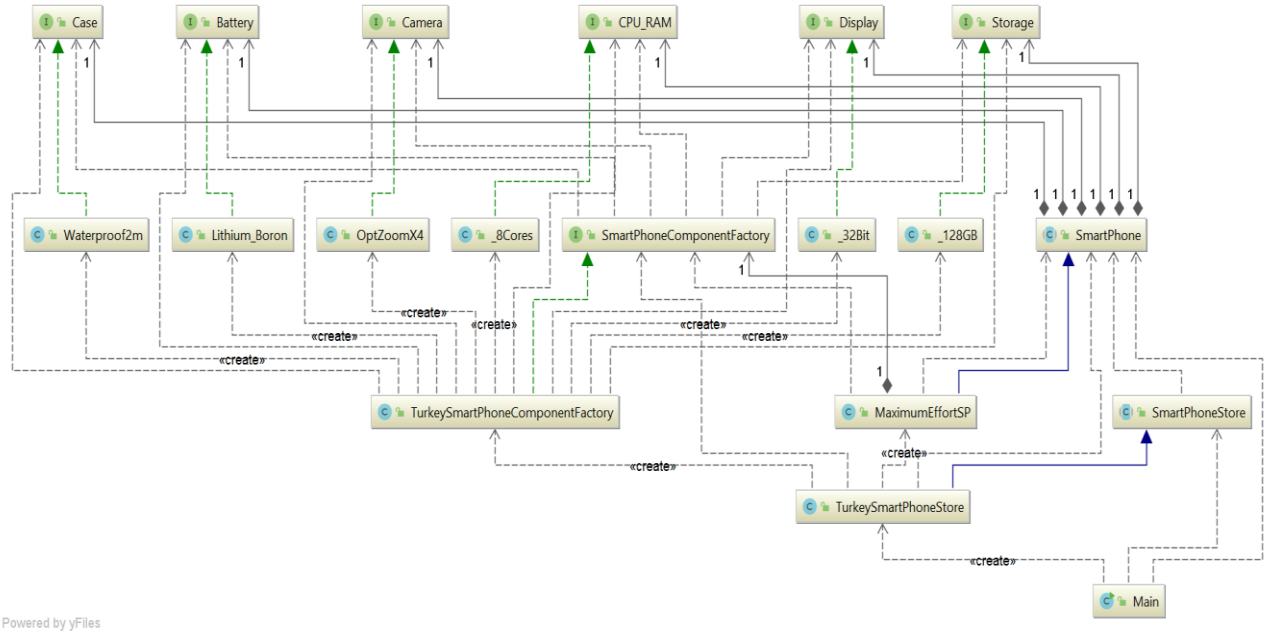
SmartPhone: The abstract class has one abstract class and six concrete class. Concrete classes printing the common informations.

SmartPhoneStore: The class also is an abstract class that used to call the all smartphone concrete and abstract methods.

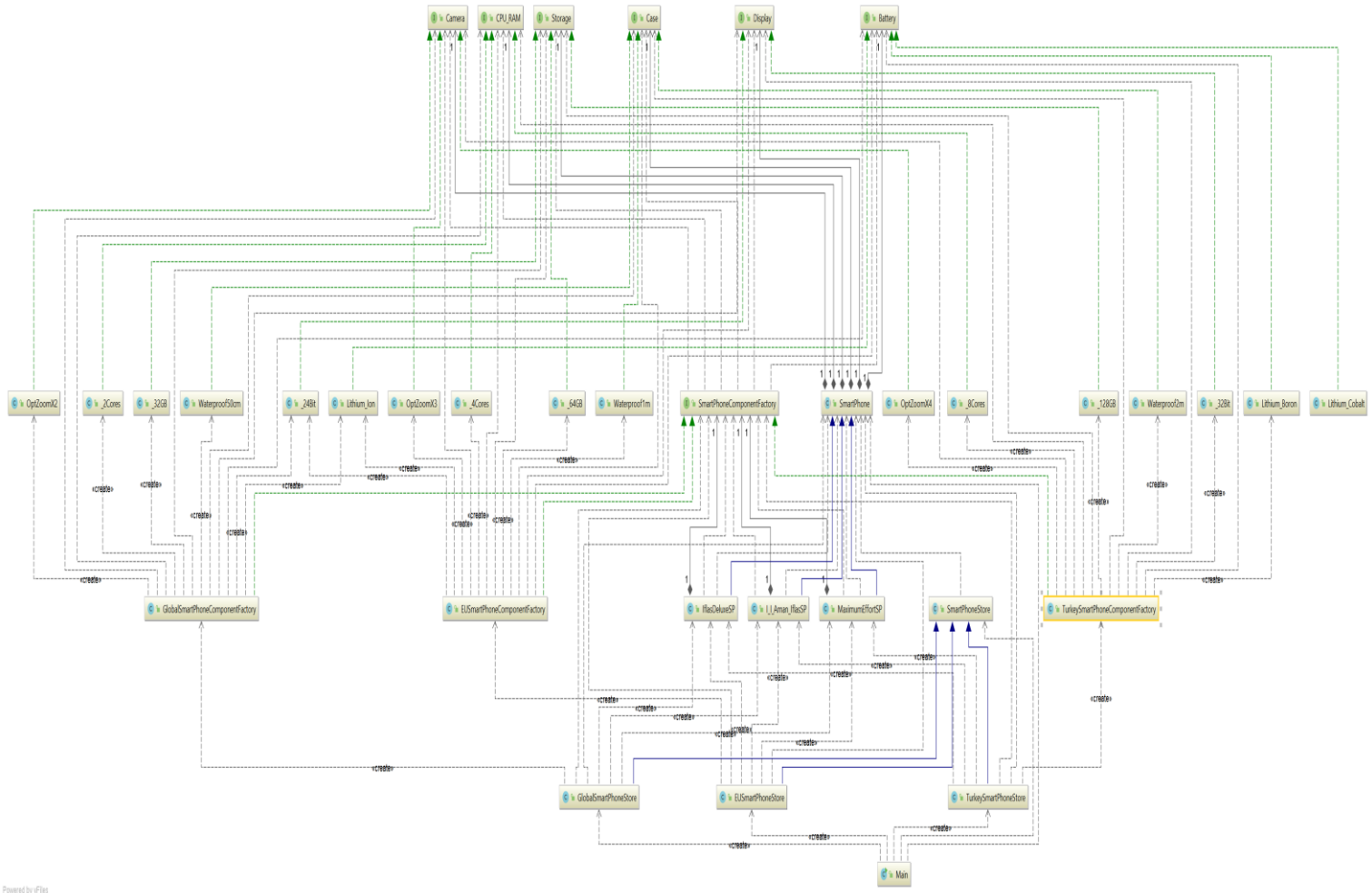
Other Classes: These classes that used to return market features.

Class Diagram:

Turkey style Maximum Effort SmartPhone class diagram:



All Classes:



Example Output:

```
public static void main(String[] args) {

    SmartPhoneStore turkeyStore = new TurkeySmartPhoneStore();
    SmartPhone smartPhone = turkeyStore.orderSmartPhone( type: "IflasDeluxe");
    System.out.println(smartPhone);
    System.out.println("-------------------------------------------------------------");

    SmartPhoneStore turkeyStore1 = new TurkeySmartPhoneStore();
    SmartPhone smartPhone1 = turkeyStore1.orderSmartPhone( type: "I_I_Aman_Iflas");
    System.out.println(smartPhone1);
    System.out.println("-------------------------------------------------------------");

    SmartPhoneStore turkeyStore2 = new TurkeySmartPhoneStore();
    SmartPhone smartPhone2 = turkeyStore2.orderSmartPhone( type: "MaximumEffort");
    System.out.println(smartPhone2);
    System.out.println("-------------------------------------------------------------");

    SmartPhoneStore euStore = new EUSmartPhoneStore();
    SmartPhone smartPhoneEU = euStore.orderSmartPhone( type: "IflasDeluxe");
    System.out.println(smartPhoneEU);
    System.out.println("-------------------------------------------------------------");


    SmartPhoneStore globalStore = new GlobalSmartPhoneStore();
    SmartPhone smartPhoneG = globalStore.orderSmartPhone( type: "I_I_Aman_Iflas");
    System.out.println(smartPhoneG);
    System.out.println("-------------------------------------------------------------");
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Didea.launcher.port=62084 "-Didea.lau
----- Making a Turkey Style IflasDeluxe SmartPhone -----
Preparing a Turkey Style IflasDeluxe SmartPhone !!!
Attach Display
Attach Battery
Attach CPU & RAM to the board
Attach Storage
Attach Camera
Enclose the Phone Case
Attached Display # 5.3 inches , 32 Bit #
Attached Battery # 20h, 2800mAh , Lithium-Boron #
Attached CPU & RAM to the board # 2.2GHz, 6GB , 8 Cores #
Attached Storage # MicroSD support, 32GB , Max 128 GB #
Attached Camera # 12Mp front, 5Mp rear , Optimum Zoom X4 #
Enclosed the Phone Case # 149x73x7.7 mm waterproof, aluminum , Waterproof up to 2m #
```

```
------------------------------------------------------------
----- Making a Turkey Style I-I-Aman-Iflas SmartPhone -----
Preparing a Turkey Style I-I-Aman-Iflas SmartPhone !!!
Attach Display
Attach Battery
Attach CPU & RAM to the board
Attach Storage
Attach Camera
Enclose the Phone Case
Attached Display # 4.5 inches , 32 Bit #
Attached Battery # 16h, 2000mAh , Lithium-Boron #
Attached CPU & RAM to the board # 2.2GHz, 4GB , 8 Cores #
Attached Storage # MicroSD support, 16GB , Max 128 GB #
Attached Camera # 8Mp front, 5Mp rear , Optimum Zoom X4 #
Enclosed the Phone Case # 143x69x7.3 mm waterproof, plastic , Waterproof up to 2m #
```

```
------------------------------------------------------------
----- Making a Turkey Style MaximumEffort SmartPhone -----
Preparing a Turkey Style MaximumEffort SmartPhone !!!
Attach Display
Attach Battery
Attach CPU & RAM to the board
Attach Storage
Attach Camera
Enclose the Phone Case
Attached Display # 5.5 inches , 32 Bit #
Attached Battery # 27h, 3600mAh , Lithium-Boron #
Attached CPU & RAM to the board # 2.8GHz, 8GB , 8 Cores #
Attached Storage # MicroSD support, 64GB , Max 128 GB #
Attached Camera # 12Mp front, 8Mp rear , Optimum Zoom X4 #
Enclosed the Phone Case # 151x73x7.7 mm dustproof, waterproof, aluminum , Waterproof up to 2m #
```

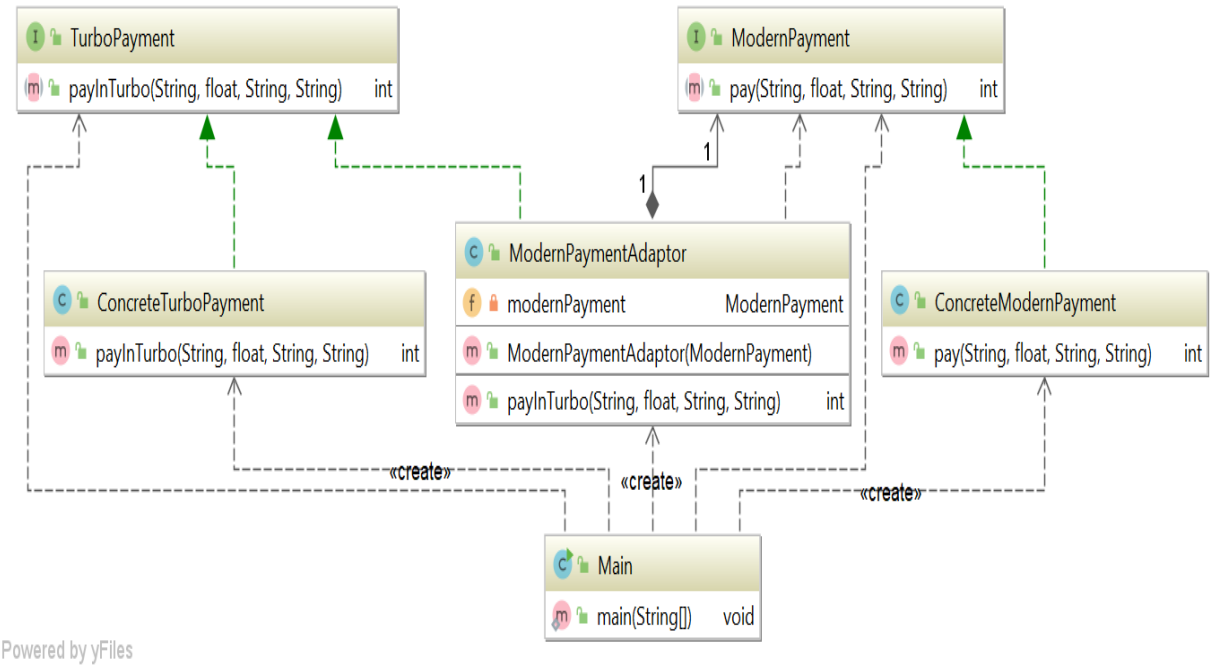
```
------------------------------------------------------------
----- Making a EU Style IflasDeluxe SmartPhone -----
Preparing a EU Style IflasDeluxe SmartPhone !!!
Attach Display
Attach Battery
Attach CPU & RAM to the board
Attach Storage
Attach Camera
Enclose the Phone Case
Attached Display # 5.3 inches , 24 Bit #
Attached Battery # 20h, 2800mAh , Lithium-Ion #
Attached CPU & RAM to the board # 2.2GHz, 6GB , 4 Cores #
Attached Storage # MicroSD support, 32GB , Max 64 GB #
Attached Camera # 12Mp front, 5Mp rear , Optimum Zoom X3 #
Enclosed the Phone Case # 149x73x7.7 mm waterproof, aluminum , Waterproof up to 1m #
```

```
------------------------------------------------------------
-----Making a Global Style I-I-Aman-Iflas SmartPhone -----
Preparing a Global Style I-I-Aman-Iflas SmartPhone !!!
Attach Display
Attach Battery
Attach CPU & RAM to the board
Attach Storage
Attach Camera
Enclose the Phone Case
Attached Display # 4.5 inches , 24 Bit #
Attached Battery # 16h, 2000mAh , Lithium-Ion #
Attached CPU & RAM to the board # 2.2GHz, 4GB , 2 Cores #
Attached Storage # MicroSD support, 16GB , Max 32 GB #
Attached Camera # 8Mp front, 5Mp rear , Optimum Zoom X2 #
Enclosed the Phone Case # 143x69x7.3 mm waterproof, plastic , Waterproof up to 50cm #
```

Part2
Adapter Design Pattern

It will be beneficial for us to use the Adapter Design Pattern in this problem. Thus, we will be able to use ModernPayment, our new payment method, wherever we used to use TurboPayment using an adapter. Thus, we will be able to use existing codes that we know to be working in new ones.
To do this, to use the old method, TurboPayment, we need to create an adapter that implements this interface. In this adapter, we provide our new payment method in constructure so that we can call the methods of this method within the methods of the old method.

Class Diagram:



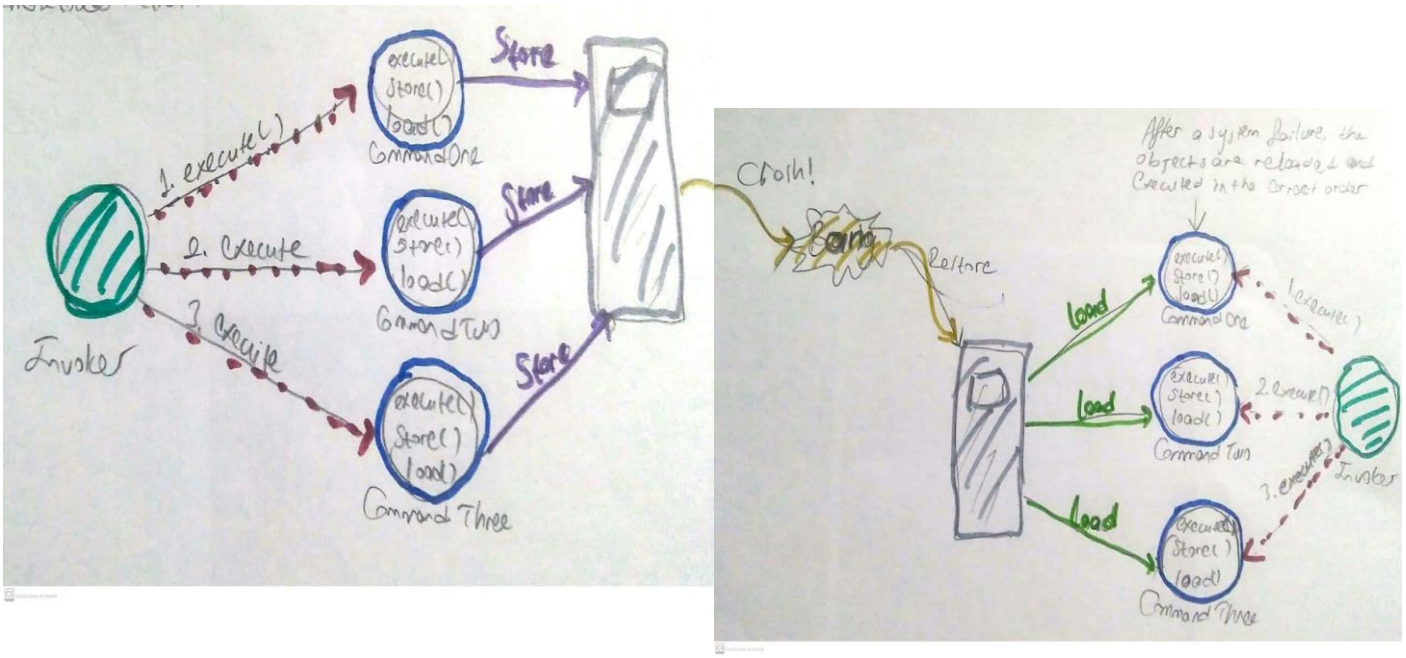
Example Output:

```
lic static void main(String[] args) {

TurboPayment turboPayment = new ConcreteTurboPayment();
System.out.println("---- Turbo Payment ----");
turboPayment.payInTurbo( turboCardNo: "3456", turboAmount: 230.99f, destinationTurboOfCourse: "Istanbul", installmentsButInTurbo: "3");
System.out.println("-----------------------------------------------------------");

System.out.println("---- Modern Payment ----");
ModernPayment modernPayment = new ConcreteModernPayment();
modernPayment.pay( cardNo: "4123", amount: 219.99f, destination: "Kocaeli", installments: "5");
System.out.println("-----------------------------------------------------------");

System.out.println("---- Modern with Turbo Payment ----");
TurboPayment payment = new ModernPaymentAdaptor(modernPayment);
payment.payInTurbo( turboCardNo: "4202", turboAmount: 100.0f, destinationTurboOfCourse: "Konya", installmentsButInTurbo: "9");
System.out.println("-----------------------------------------------------------");
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Didea.launcher.
---- Turbo Payment ----
Paying with Turbo Payment=>
Card No: 3456
Destination: Istanbul
Installments: 3
Done!
-----------------------------------------------------------
---- Modern Payment ----
Paying with Modern Payment =>
Card No: 4123
Destination: Kocaeli
Installments: 5
Done!
-----------------------------------------------------------
---- Modern with Turbo Payment ----
Paying with Modern Payment =>
Card No: 4202
Destination: Konya
Installments: 9
Done!
-----------------------------------------------------------
```

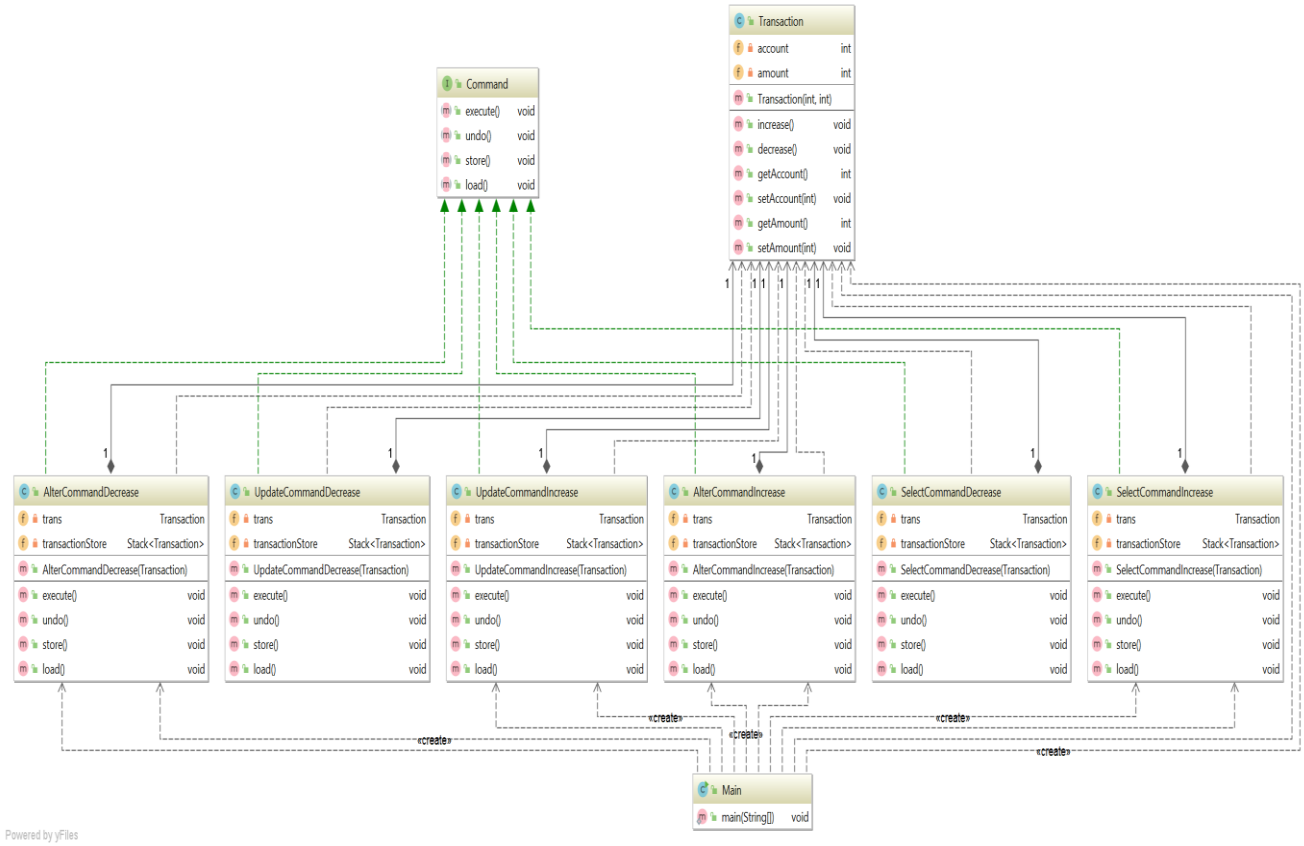
Part3
Command Design Pattern

I used Command Design Pattern because of all system using same commands and the system work with commands.
My thought was that like the Picture:



I implemented a Transaction class that includes increase and decrease methods. I used a Command interface that includes execute(), undo(), store() and load() methods. Also I wrote two class for each database operation like AlterCommandIncrease and AlterCommandDecrease. Also these classes implements from Command interface. I used stack data structure for storing the transactions.

Class Diagram:



Example Output:

```
public static void main(String[] args) {
    Transaction trans = new Transaction( account: 100,  amount: 10);
    //Transaction trans1 = new Transaction(200, 20);
    //Transaction trans2 = new Transaction(300, 30);
    AlterCommandIncrease alterInc = new AlterCommandIncrease(trans);
    SelectCommandIncrease selectInc = new SelectCommandIncrease(trans);
    UpdateCommandIncrease updateInc = new UpdateCommandIncrease(trans);
    AlterCommandDecrease alterdec = new AlterCommandDecrease(trans);

    alterInc.execute();
    alterInc.store();
    System.out.println("------------------------------------------------------------------------");
    selectInc.execute();
    selectInc.store();
    System.out.println("------------------------------------------------------------------------");
    updateInc.execute();
    updateInc.store();
    System.out.println("------------------------------------------------------------------------");
    alterdec.execute();
    alterdec.store();
    System.out.println("####### CRASH!!!! #######");
    alterInc.load();
    selectInc.load();
    updateInc.load();
    alterdec.load();
    System.out.println("------------------------------------------------------------------------");
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Didea.launcher.port=64404
Transaction Inc:  110 = 100 + 10
Done increase transaction and Alter Table => account: 110 amount: 10
TransStoreIncreaseAlter: Account: 110  Amount: 10
------------------------------------------------------------------------
Transaction Inc:  120 = 110 + 10
Done increase transaction and Select Table => account: 120 amount: 10
TransStoreIncreaseSelect: Account: 120  Amount: 10
------------------------------------------------------------------------
Transaction Inc:  130 = 120 + 10
Done increase transaction and Update Table => account: 130 amount: 10
TransStoreIncreaseUpdate: Account: 130  Amount: 10
------------------------------------------------------------------------
Transaction Dec:  120 = 130 - 10
Done decrease transaction and Alter Table => account: 120 amount: 10
TransStoreDecreaseAlter: Account: 120  Amount: 10
####### CRASH!!!! #######
TransLoadIncreaseAlter: Account: 110  Amount: 10
TransLoadIncreaseSelect: Account: 100  Amount: 10
TransLoadIncreaseUpdate: Account: 90  Amount: 10
TransLoadDecreaseAlter: Account: 100  Amount: 10
------------------------------------------------------------------------
```

Part4
Template Method Design Pattern

Since we will use the same algorithm differently, we need a Template Method design pattern. According to this pattern:

Discrete Fourier Transform

I implemented the transform method for the Discrete Fourier Transform class with the following formula.

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} kn}$$
$$= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)],$$

Xn = (a + bi)
A loop k, for returning from 0 to N. Within this loop, a for loop returning from 0 to N and cos sin operations in this second loop were added.

Discrete Cosine Transform

DCT-II

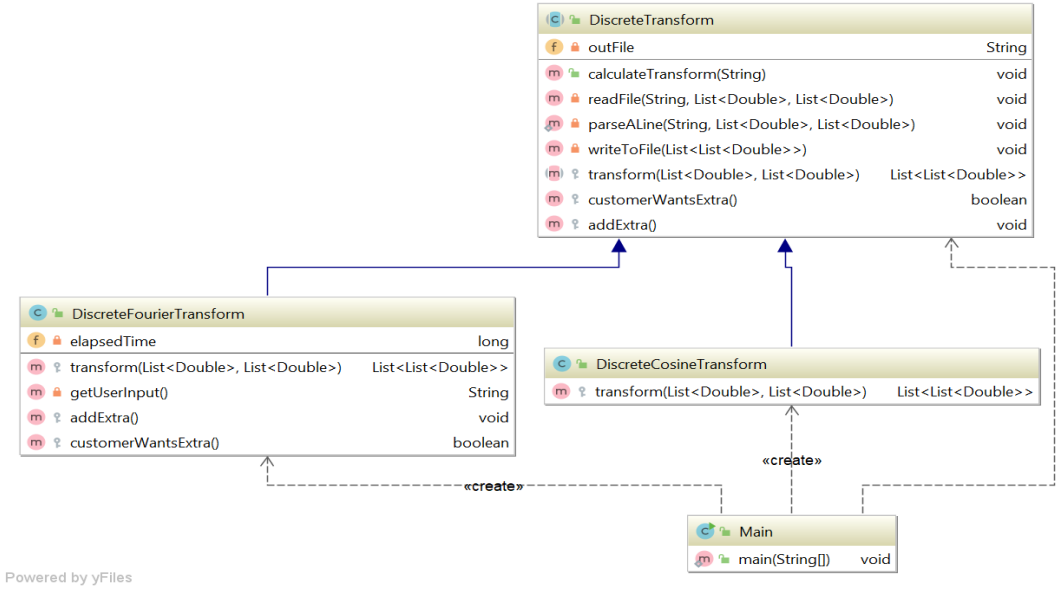
$$X_k = \sum_{n=0}^{N-1} x_n \cos\left[\frac{\pi}{N}\left(n + \frac{1}{2}\right)k\right] \qquad k = 0, \ldots, N-1.$$

A loop k, for returning from 0 to N. In this loop, a for loop returning from 0 to N and cos operations in this second loop were added.

An abstract class named "DiscreateTransform" was created and a method containing the solution algorithm for this class was written (calculateTransform). To implement this algorithm, the methods required to read the file and write the results to the file are implemented in this abstract class. The main method to perform the transform is left as the abstract method and subclass is expected to implement. After performing the transform in DiscreteFourierTransform, an if has been added to the algorithm (calculateTransform) to ask the user an extra question. The customerWantsExtra method, which returns boolean if the condition of this if is returned, this method returns "false" in the abtract class. If the subclasses want any extras, they can override this method and override the "addExtra" method that will be called when the extras are in the if.

For DiscreteCosineTransform, "customerWantsExtra" and "addExtra" methods are not overrided because no extra is requested. For DiscreteFourierTransform, this time was overridden by asking the user (whether they want to see how long the transform takes place) in the "customerWantsExtra" method and by printing this time on the screen in the "addExtra" method.

Class Diagram:



Example Output:

```
public static void main(String[] args)  {

    String filename = "inputFile.txt";
    DiscreteTransform discreteCosineTransform= new DiscreteCosineTransform();
    DiscreteTransform discreteFourierTransform = new DiscreteFourierTransform();

    System.out.println("Discrete Cosine Transform");
    discreteCosineTransform.calculateTransform(filename);
    System.out.println("Discrete Fourier Transform");
    discreteFourierTransform.calculateTransform(filename);


}
```

```
"C:\Program Files\Java\jdk-14.0.2\bin\java.exe" -Dide
Discrete Cosine Transform
----  Results  ----
21.0
5.598076211353318
-13.499999999999998

Discrete Fourier Transform
----  Results  ----
21.0 + 0.29999999999999893i,
-8.07846096908266 + 18.09615242270663i,
-3.9215390309173443 + 7.7038475772933594i,

Would you like see time of execution (y/n)? y
Time of execution DFT in nanotime: 80500

Process finished with exit code 0
```

inputFile.txt

```
3   8.7 6   -5.4    12   -3
```

outFile.txt

```
21.0 + 0.29999999999999893i,
-8.07846096908266 + 18.09615242270663i,
-3.9215390309173443 + 7.7038475772933594i,
```