CSE443 – Object Oriented Analysis and Desing

MIDTERM PROJECT - REPORT

Furkan ÖZEV

161044036

PART 1:

In the Iflas-Technologies Ltd project, the company has decided to make and sell smartphones.

Every smartphone consists of 6 components: Display, Battery, CPU&RAM, Storage, Camera, Case

The company produces 3 distinct model series: Maximum Effort, IflasDeluxe and I-I-Aman-Iflas. Each phone model has different features for these components. Some features of these components are unique to the model and do not change. For example, for Display, the MaximumEffort model is 5.5 inches, the IflasDeluxe model is 5.3 inches, and the I-I-Aman-Iflas model is 4.5 inches.

This company sells the same models, with different specifications to different markets. Some features of the components of these phone models vary depending on the market. For instance the same model MaximumEffort is sold in Turkey with a 5.5 inch, 32bit display while it's sold at the EU market with a 5.5 inch but 24-bit display.

- Turkey-Maximum Effort Display Specifications: 5.5 inches, 32 bit
- EU-Maximum Effort Display Specifications: 5.5 inches, 24 bit
- Global-Maximum Effort Display Specifications: 5.5 inches, 24 bit

The production of a phone is carried out in the following order:

- attach cpu & ram to the board
- attach display
- attach battery
- attach storage
- attach camera
- enclose the phone case

A piece of software will be produced to manage the production of these smartphones.

Thus, I designed the Iflas-Technologies Ltd project using the **Abstract Factory Design Pattern**.

The Abstract Factory Design Pattern provides an interface for creating families of related or dependent objects without specifying their concrete classes.

There are three types of models of phone companies in project and they are intended to be sold in three different markets.

Some features of the 6 components vary according to the model and some features vary according to the market to be sold.

Each of these 6 components is an abstract class and can contain both market and model features.

The production steps of each model are similar but perform differently. So we have an abstract class called phoneComponentFactory. Each market has its own componentFactory class and these classes derive from the phoneComponentFactory class.

There is an abstract parent class called Market. All market classes derive from this class. It includes methods createPhone () and orderPhone (). Therefore, phone production steps are the same in every market, but they can produce different features because they are produced in their own factories.

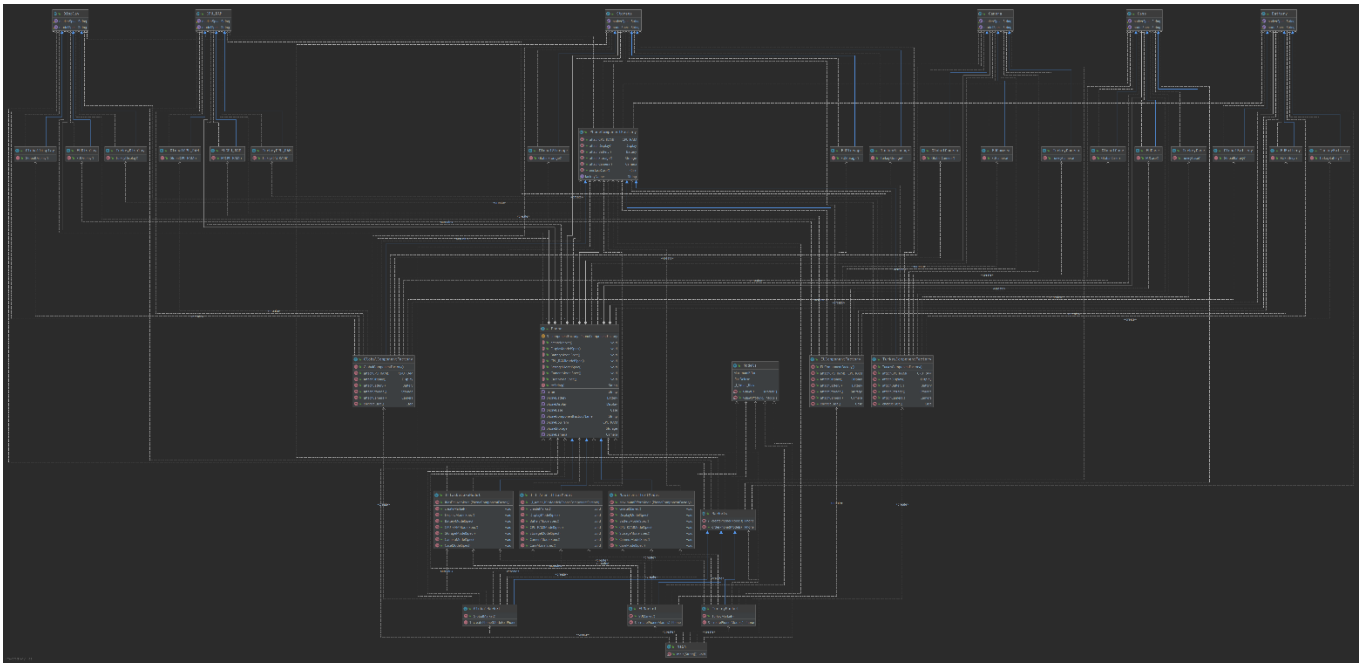
There is an abstract class called Phone. All phone models are derived from this class. It contains the common features and methods of phones.

It will print on screen step by step the production phases of every model from every market.

When a new phone model with same component wants to be added, the things to do are simply as follows:

1- The name of the new phone model name must be added to the "Models" enum class.
2- The class of the new phone model must extend the "Phone" abstract class. Then, It should implement own methods with own specifications.

Class Diagram:



Output:

```
----- TURKEY MARKET TESTS -----
### Turkey Market is created. ###

-> Phone order has been received.
-> Phone model is: Turkey - MaximumEffort Model
-> Processes:

    * attach 2.8GHz, 8GB, 8 cores CPU in Turkey Component Factory
    * attach 5.5 inches, 32 bit Display in Turkey Component Factory
    * attach 27h, 3600mAh, Lithium-Boron Battery in Turkey Component Factory
    * attach MicroSD support, 64GB, Max 128 GB Storage in Turkey Component Factory
    * attach 12Mp front, 8Mp rear, Opt. zoom x4 Camera in Turkey Component Factory
    * attach 151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 2m Case in Turkey Component Factory

After processes, Result:
---- Turkey - MaximumEffort Model ----
DISPLAY: 5.5 inches, 32 bit
BATTERY: 27h, 3600mAh, Lithium-Boron
CPU&RAM: 2.8GHz, 8GB, 8 cores
STORAGE: MicroSD support, 64GB, Max 128 GB
CAMERA: 12Mp front, 8Mp rear, Opt. zoom x4
CASE: 151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 2m

--------------------------------------------------
```

```
-> Phone order has been received.
-> Phone model is: Turkey - IflasDeluxe Model
-> Processes:

    * attach 2.2GHz, 6GB, 8 cores CPU in Turkey Component Factory
    * attach 5.3 inches, 32 bit Display in Turkey Component Factory
    * attach 20h, 2800mAh, Lithium-Boron Battery in Turkey Component Factory
    * attach MicroSD support, 32GB, Max 128 GB Storage in Turkey Component Factory
    * attach 12Mp front, 5Mp rear, Opt. zoom x4 Camera in Turkey Component Factory
    * attach 149x73x7.7 mm waterproof, aluminum, Waterproof up to 2m Case in Turkey Component Factory

After processes, Result:
---- Turkey - IflasDeluxe Model ----
DISPLAY: 5.3 inches, 32 bit
BATTERY: 20h, 2800mAh, Lithium-Boron
CPU&RAM: 2.2GHz, 6GB, 8 cores
STORAGE: MicroSD support, 32GB, Max 128 GB
CAMERA: 12Mp front, 5Mp rear, Opt. zoom x4
CASE: 149x73x7.7 mm waterproof, aluminum, Waterproof up to 2m

--------------------------------------------------
```

```
-> Phone order has been received.
-> Phone model is: Turkey - I-I-Aman-Iflas Model
-> Processes:

    * attach 2.2GHz, 4GB, 8 cores CPU in Turkey Component Factory
    * attach 4.5 inches, 32 bit Display in Turkey Component Factory
    * attach 16h, 2000mAh, Lithium-Boron Battery in Turkey Component Factory
    * attach MicroSD support, 16GB, Max 128 GB Storage in Turkey Component Factory
    * attach 8Mp front, 5Mp rear, Opt. zoom x4 Camera in Turkey Component Factory
    * attach 143x69x7.3 mm waterproof, plastic, Waterproof up to 2m Case in Turkey Component Factory

After processes, Result:
---- Turkey - I-I-Aman-Iflas Model ----
DISPLAY: 4.5 inches, 32 bit
BATTERY: 16h, 2000mAh, Lithium-Boron
CPU&RAM: 2.2GHz, 4GB, 8 cores
STORAGE: MicroSD support, 16GB, Max 128 GB
CAMERA: 8Mp front, 5Mp rear, Opt. zoom x4
CASE: 143x69x7.3 mm waterproof, plastic, Waterproof up to 2m

--------------------------------------------------
```

```
----- EU MARKET TESTS -----
### EU Market is created. ###

-> Phone order has been received.
-> Phone model is: EU - MaximumEffort Model
-> Processes:

    * attach 2.8GHz, 8GB, 4 cores CPU in EU Component Factory
    * attach 5.5 inches, 24 bit Display in EU Component Factory
    * attach 27h, 3600mAh, Lithium-Ion Battery in EU Component Factory
    * attach MicroSD support, 64GB, Max 64 GB Storage in EU Component Factory
    * attach 12Mp front, 8Mp rear, Opt. zoom x3 Camera in EU Component Factory
    * attach 151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 1m Case in EU Component Factory

After processes, Result:
---- EU - MaximumEffort Model ----
DISPLAY: 5.5 inches, 24 bit
BATTERY: 27h, 3600mAh, Lithium-Ion
CPU&RAM: 2.8GHz, 8GB, 4 cores
STORAGE: MicroSD support, 64GB, Max 64 GB
CAMERA: 12Mp front, 8Mp rear, Opt. zoom x3
CASE: 151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 1m

--------------------------------------------------
```

```
-> Phone order has been received.
-> Phone model is: EU - IflasDeluxe Model
-> Processes:

    * attach 2.2GHz, 6GB, 4 cores CPU in EU Component Factory
    * attach 5.3 inches, 24 bit Display in EU Component Factory
    * attach 20h, 2800mAh, Lithium-Ion Battery in EU Component Factory
    * attach MicroSD support, 32GB, Max 64 GB Storage in EU Component Factory
    * attach 12Mp front, 5Mp rear, Opt. zoom x3 Camera in EU Component Factory
    * attach 149x73x7.7 mm waterproof, aluminum, Waterproof up to 1m Case in EU Component Factory

After processes, Result:
---- EU - IflasDeluxe Model ----
DISPLAY: 5.3 inches, 24 bit
BATTERY: 20h, 2800mAh, Lithium-Ion
CPU&RAM: 2.2GHz, 6GB, 4 cores
STORAGE: MicroSD support, 32GB, Max 64 GB
CAMERA: 12Mp front, 5Mp rear, Opt. zoom x3
CASE: 149x73x7.7 mm waterproof, aluminum, Waterproof up to 1m

--------------------------------------------------
```

```
-> Phone order has been received.
-> Phone model is: EU - I-I-Aman-Iflas Model
-> Processes:

    * attach 2.2GHz, 4GB, 4 cores CPU in EU Component Factory
    * attach 4.5 inches, 24 bit Display in EU Component Factory
    * attach 16h, 2000mAh, Lithium-Ion Battery in EU Component Factory
    * attach MicroSD support, 16GB, Max 64 GB Storage in EU Component Factory
    * attach 8Mp front, 5Mp rear, Opt. zoom x3 Camera in EU Component Factory
    * attach 143x69x7.3 mm waterproof, plastic, Waterproof up to 1m Case in EU Component Factory

After processes, Result:
---- EU - I-I-Aman-Iflas Model ----
DISPLAY: 4.5 inches, 24 bit
BATTERY: 16h, 2000mAh, Lithium-Ion
CPU&RAM: 2.2GHz, 4GB, 4 cores
STORAGE: MicroSD support, 16GB, Max 64 GB
CAMERA: 8Mp front, 5Mp rear, Opt. zoom x3
CASE: 143x69x7.3 mm waterproof, plastic, Waterproof up to 1m

--------------------------------------------------
```

```
----- GLOBAL MARKET TESTS -----
### Global Market is created. ###

-> Phone order has been received.
-> Phone model is: Global - MaximumEffort Model
-> Processes:

    * attach 2.8GHz, 8GB, 2 cores CPU in Global Component Factory
    * attach 5.5 inches, 24 bit Display in Global Component Factory
    * attach 27h, 3600mAh, Lithium-Cobalt Battery in Global Component Factory
    * attach MicroSD support, 64GB, Max 32 GB Storage in Global Component Factory
    * attach 12Mp front, 8Mp rear, Opt. zoom x2 Camera in Global Component Factory
    * attach 151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 50cm Case in Global Component Factory

After processes, Result:
---- Global - MaximumEffort Model ----
DISPLAY: 5.5 inches, 24 bit
BATTERY: 27h, 3600mAh, Lithium-Cobalt
CPU&RAM: 2.8GHz, 8GB, 2 cores
STORAGE: MicroSD support, 64GB, Max 32 GB
CAMERA: 12Mp front, 8Mp rear, Opt. zoom x2
CASE: 151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 50cm

-------------------------------------------------
```

```
-> Phone order has been received.
-> Phone model is: Global - IflasDeluxe Model
-> Processes:

    * attach 2.2GHz, 6GB, 2 cores CPU in Global Component Factory
    * attach 5.3 inches, 24 bit Display in Global Component Factory
    * attach 20h, 2800mAh, Lithium-Cobalt Battery in Global Component Factory
    * attach MicroSD support, 32GB, Max 32 GB Storage in Global Component Factory
    * attach 12Mp front, 5Mp rear, Opt. zoom x2 Camera in Global Component Factory
    * attach 149x73x7.7 mm waterproof, aluminum, Waterproof up to 50cm Case in Global Component Factory

After processes, Result:
---- Global - IflasDeluxe Model ----
DISPLAY: 5.3 inches, 24 bit
BATTERY: 20h, 2800mAh, Lithium-Cobalt
CPU&RAM: 2.2GHz, 6GB, 2 cores
STORAGE: MicroSD support, 32GB, Max 32 GB
CAMERA: 12Mp front, 5Mp rear, Opt. zoom x2
CASE: 149x73x7.7 mm waterproof, aluminum, Waterproof up to 50cm

-------------------------------------------------
```

```
-> Phone order has been received.
-> Phone model is: Global - I-I-Aman-Iflas Model
-> Processes:

    * attach 2.2GHz, 4GB, 2 cores CPU in Global Component Factory
    * attach 4.5 inches, 24 bit Display in Global Component Factory
    * attach 16h, 2000mAh, Lithium-Cobalt Battery in Global Component Factory
    * attach MicroSD support, 16GB, Max 32 GB Storage in Global Component Factory
    * attach 8Mp front, 5Mp rear, Opt. zoom x2 Camera in Global Component Factory
    * attach 143x69x7.3 mm waterproof, plastic, Waterproof up to 50cm Case in Global Component Factory

After processes, Result:
---- Global - I-I-Aman-Iflas Model ----
DISPLAY: 4.5 inches, 24 bit
BATTERY: 16h, 2000mAh, Lithium-Cobalt
CPU&RAM: 2.2GHz, 4GB, 2 cores
STORAGE: MicroSD support, 16GB, Max 32 GB
CAMERA: 8Mp front, 5Mp rear, Opt. zoom x2
CASE: 143x69x7.3 mm waterproof, plastic, Waterproof up to 50cm

-------------------------------------------------
```

PART 2:

In the Payment project, A new java interface called "ModernPayment" will be created for credit card payments.

However company is still using extensively an old binary library from the 1990s called “TurboPayment” for card payments that it cannot afford to replace.

Where all the method parameters have the same meaning and role as in ModernPayment.

To continue using all the classes implementing the TurboPayment interface with your new ModernPayment interface, a design model will be implemented in Java.

The old library is binary, so cannot be modified the TurboPayment interface or the classes that implement it.

Thus, I designed Payment project using the **Adapter Design Pattern**.

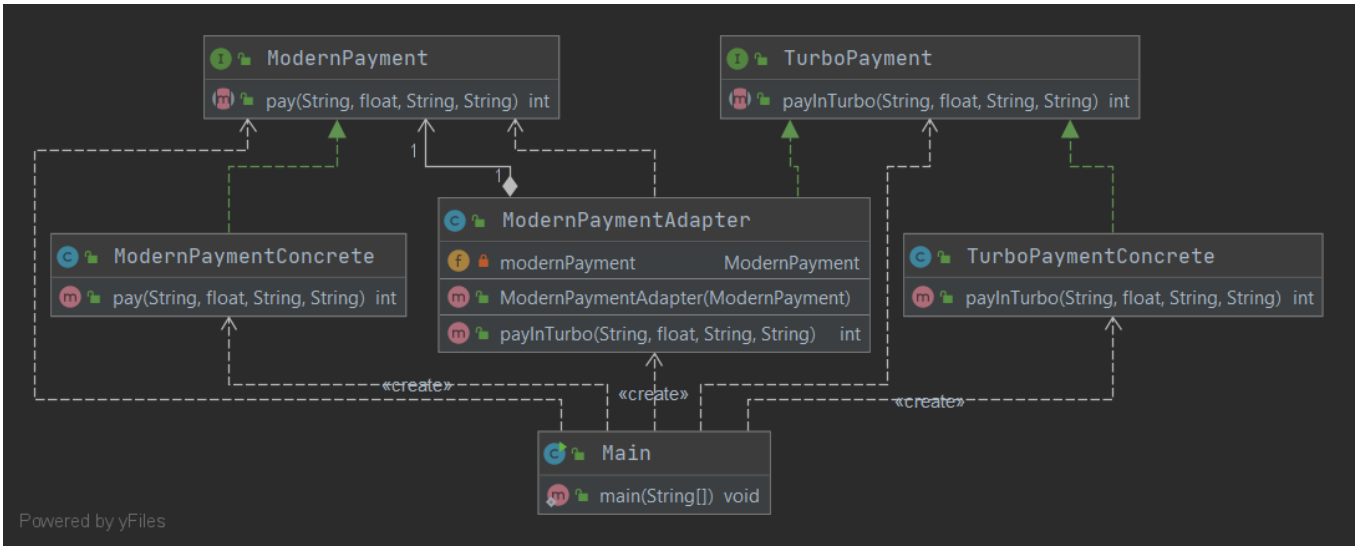
The Adapter Design Pattern converts the interface of a class into another interface the clients expect. Adapter lets classes work together that couldn’t otherwise because of incompatible interfaces.

A class called ModernPaymentAdapter is created. This class acts as an adapter between the old TurboPayment class and the new ModernPayment class.

The ModernPaymentAdapter class implements the older TurboPayment interface. Thus, the new interface can be used without changing the payInTurbo() method in TurboPayment. ModernPaymentAdapter keeps the new ModernPayment class as field. Calls ModerPayment in the pay () method, overriding the payInTurbo () method.

To be used in the test, 2 concrete classes have been created to create objects from both interfaces.

Class Diagram:



Output:

```
### Turbo Payment Object ###
    -> Real Type: TurboPaymentConcrete  -> Declared Type: TurboPayment

### Paying with Turbo Payment ###
    -> Card No: 4829-4828-3492
    -> Amount: 2500.79
    -> Destination: Credit Card 1
    -> Installments: 9
### Paying is done! ###


-------------------------------------
```

```
### Modern Payment Object ###
    -> Real Type: ModernPaymentConcrete     -> Declared Type: ModernPaymentConcrete

### Paying with Modern Payment ###
    -> Card No: 5713-3390-4512
    -> Amount: 1037.8
    -> Destination: Credit Card 2
    -> Installments: 4
### Paying is done! ###


-------------------------------------
```

```
### Adaptor Object for paying Modern In Turbo ###
    -> Real Type: ModernPaymentConcrete     -> Declared Type: TurboPayment

### Paying with Modern Payment ###
    -> Card No: 6783-9901-3867
    -> Amount: 637.06
    -> Destination: Credit Card 3
    -> Installments: 2
### Paying is done! ###

-------------------------------------
```

PART 3:

In the DatabaseBank project, a design solution must be designed personally to model transactions and database transactions.

A database operation can be a SELECT, an UPDATE or an ALTER.

A transaction is a series of operations (a SELECT followed by an ALTER, or an UPDATE followed by ALTER followed by SELECT, and so on).

If one of the transactions fails, the others will be reversed or canceled.

a) Explain your design solution for this problem and your motivation for your decisions, which design pattern you use (if any) and how (draw the class diagram):

I designed DatabaseBank project using the **Command Design Pattern**.

The Command Design Pattern encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.

I chose this design pattern because it consists of commands such as Select, Update, Alter, these commands can be called one after the other, and these series of operations are reversible.

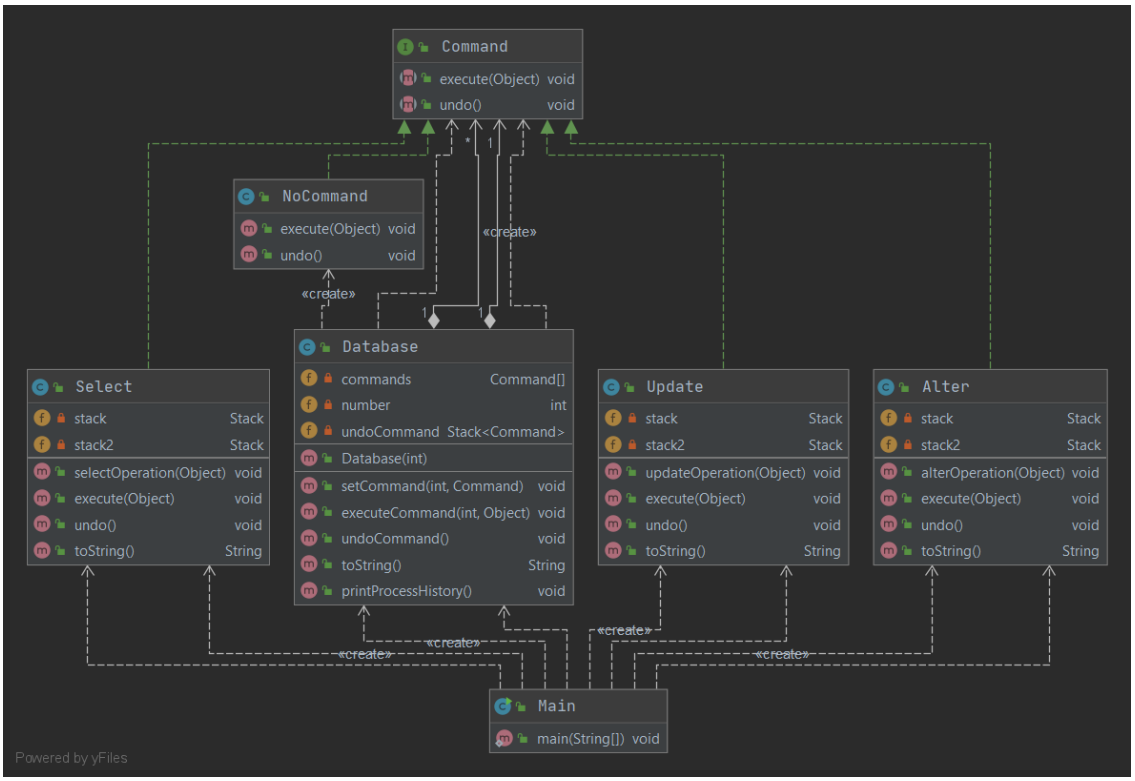
There is an interface called Command. This class contains the execute () and undo () methods.

The command classes Select, Update, Alter implement this interface. Override the execute () and undo () methods themselves.

Database stores these commands in an array. Adds a new command to array with the setCommand () method. Allows any command to run with the executeCommand () method. With the undoCommand () method, it undo the operation by calling the undo () method of the last running command. The printProcessHistory () method prints commands that have worked so far and have not been undone.

Commands can return to the previous state by keeping a stack inside them. The database stores the commands called in itself in a stack. It can undo such commands in the order they were called.

Class Diagram:



b) Code your solution's significant classes; and explain in detail how you would implement the rollback of transactions:

Codes are available in ./src.

Command Interface includes execute() and undo() methods.

Select / Update / Alter classes implement Command interface, and override execute () and undo () methods.

The Database class stores these commands in an array. And as commands are called, they are push into the stack. Thus, when the undoCommand () function is called, the undo () method of the last command in the stack is called. Thus, the last operation is undone.

Database undoCommand():

```
public void undoCommand(){
    if(!undoCommand.empty()){
        Command lastCommand = undoCommand.pop();
        lastCommand.undo();
    }
    else{
        System.out.println("There is no operation to undo.\n");
    }

}
```

Select undo():

```
public void undo(){
    System.out.println("--- Undo select command is started. ---");
    if(!stack.empty()){
        Object item = stack.pop();
        System.out.println("--- Undo select command is finished. Object: " + item + " ---\n");
    }
    else{
        System.out.println("--- Undo select command is finished. ---\n");
    }
}
```

Output:

```
Database database = new Database( number: 3);

Select selectCommand = new Select();
database.setCommand( slot: 0, selectCommand);

Update updateCommand = new Update();
database.setCommand( slot: 1, updateCommand);

Alter alterCommand = new Alter();
database.setCommand( slot: 2, alterCommand);

System.out.println(database);
```

```
---------- DATABASE ----------
[slot 0] Select
[slot 1] Update
[slot 2] Alter
```

```
database.executeCommand( slot: 0,  item: "Select-1 Command");
database.executeCommand( slot: 1,  item: "Update-1 Command");
database.executeCommand( slot: 2,  item: "Alter-1 Command");

database.executeCommand( slot: 0,  item: "Select-2 Command");
database.executeCommand( slot: 2,  item: "Alter-2 Command");
database.undoCommand();
```

```
--- Select execute command is started. Object: Select-1 Command ---
--- Select operation process completed. ---
--- Select execute command is finished. ---

--- Update execute command is started. Object: Update-1 Command ---
--- Update operation process completed. ---
--- Update execute command is finished.

--- Alter execute command is started. Object: Alter-1 Command ---
--- Alter operation process completed. ---
--- Alter execute command is finished.

--- Select execute command is started. Object: Select-2 Command ---
--- Select operation process completed. ---
--- Select execute command is finished. ---

--- Alter execute command is started. Object: Alter-2 Command ---
--- Alter operation process completed. ---
--- Alter execute command is finished.

--- Undo alter command is started. ---
--- Undo alter command is finished. Object: Alter-2 Command ---
```

```
database.executeCommand( slot: 1,  item: "Update-3 Command");
database.executeCommand( slot: 2,  item: "Alter-3 Command");
database.executeCommand( slot: 0,  item: "Select-3 Command");
database.undoCommand();
database.undoCommand();

database.printProcessHistory();
```

```
--- Update execute command is started. Object: Update-3 Command ---
--- Update operation process completed. ---
--- Update execute command is finished.

--- Alter execute command is started. Object: Alter-3 Command ---
--- Alter operation process completed. ---
--- Alter execute command is finished.

--- Select execute command is started. Object: Select-3 Command ---
--- Select operation process completed. ---
--- Select execute command is finished. ---

--- Undo select command is started. ---
--- Undo select command is finished. Object: Select-3 Command ---

--- Undo alter command is started. ---
--- Undo alter command is finished. Object: Alter-3 Command ---
```

```
---------- Process History ----------

Process #5 Update-3 Command

Process #4 Select-2 Command

Process #3 Alter-1 Command

Process #2 Update-1 Command

Process #1 Select-1 Command

--------------------
```

```
database.executeCommand( slot: 1,  item: "Update-4 Command");
database.executeCommand( slot: 2,  item: "Alter-4 Command");
database.executeCommand( slot: 1,  item: "Update-5 Command");
database.executeCommand( slot: 0,  item: "Select-4 Command");
database.executeCommand( slot: 0,  item: "Select-5 Command");
database.executeCommand( slot: 1,  item: "Update-6 Command");
database.executeCommand( slot: 2,  item: "Alter-5 Command");
database.executeCommand( slot: 0,  item: "Select-6 Command");
database.executeCommand( slot: 2,  item: "Alter-6 Command");
database.undoCommand();
database.executeCommand( slot: 2,  item: "Alter-7 Command");
database.executeCommand( slot: 1,  item: "Update-7 Command");
database.undoCommand();
database.undoCommand();
database.executeCommand( slot: 0,  item: "Select-7 Command");
database.executeCommand( slot: 0,  item: "Select-8 Command");
database.undoCommand();
database.executeCommand( slot: 1,  item: "Update-8 Command");

database.printProcessHistory();
```

```
---------- Process History ----------

Process #15 Update-8 Command

Process #14 Select-7 Command

Process #13 Select-6 Command

Process #12 Alter-5 Command

Process #11 Update-6 Command

Process #10 Select-5 Command

Process #9 Select-4 Command

Process #8 Update-5 Command

Process #7 Alter-4 Command

Process #6 Update-4 Command

Process #5 Update-3 Command

Process #4 Select-2 Command

Process #3 Alter-1 Command

Process #2 Update-1 Command

Process #1 Select-1 Command
```

PART 4:

DiscreteTransform project is a project that will allow it to realize 1D discrete transforms.

1D discrete transform has 2 types. These; 1D Discrete Fourier Transform and 1D Discrete Cosine Transform.

Both the 1D Discrete Fourier Transform and the 1D Discrete Cosine Transform do the same thing. Given a finite sequence of numbers, they express them us a sum of basis functions; their difference being that DFT uses complex exponentials and DCT uses real-valued cosine functions. They are both very powerful transformations with numerous actual applications; such as jpeg compression, antenna technologies, etc.

When given an array of numbers of length N, DFT and DCT both produce a new array of numbers also of length N. DFT produces complex numbers, DCT produces real numbers.

Both transforms follow quite much the same main procedure:

- Read N tab separated numbers from a file provided as a command line argument.
- Transform the numbers into N outputs (DFT or DCT outputs..).
- Write the N outputs to a new file.
- Only in the case of DFT, sometimes the user wants additionally the time of execution printed on screen; by default NO.

Thus, I designed the DiscreteTransform project using the **Template Method Design Pattern**.

The Template Method Pattern defines the skeleton of an algorithm in a method, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure.

We have an abstract class called DiscreteTransform. In this class, input file and output file are kept as String field. The numbers to be read from the input file are stored in an ArrayList named inputNumber.

The transform () method is the method to perform the transform. This method will be common to all sub classes and cannot be overridden since it is a final keyword. It performs the following stages respectively:

```
public final void transform() throws IOException {
    readFile(inputPath);
    transfromNumbers();
    writeFile(outputPath);
    hook();
}
```

The readFile () method reads the tab separated numbers from the input file. Common to all subclasses, but can override if a specialized class is added later.

The transformNumbers () method is an abstract method. Subclasses that extend the DiscreteTransform class have to implement this function. In this function, they perform the transform operation by using their own special mathematical formula or expression.

The writeFile () method writes the string that includes output numbers to the output file. Common to all subclasses, but can override if a specialized class is added later.

The outputToWriteFile() method is an abstract method. Subclasses that extend the DiscreteTransform class have to implement this function. In this function, output numbers are strings in their own special format and returns this string to be printed.

The writeFile () method writes the string that includes output numbers to the output file. Common to all subclasses, but can override if a specialized class is added later.

The hook () method is a method written so that it can be done after the transform when there are additional operations. The default state does nothing. It can be optionally overridden by subclasses. This project will optionally print time of execution information for DFT. Therefore, DFT will override the hook () method to do this. The RequestTime () method does this and is called by the hook () in DFT.

The ComplexNumber class was created to be used for DFT. Because the output of DFT is expressed with a complex number.

Formulas:

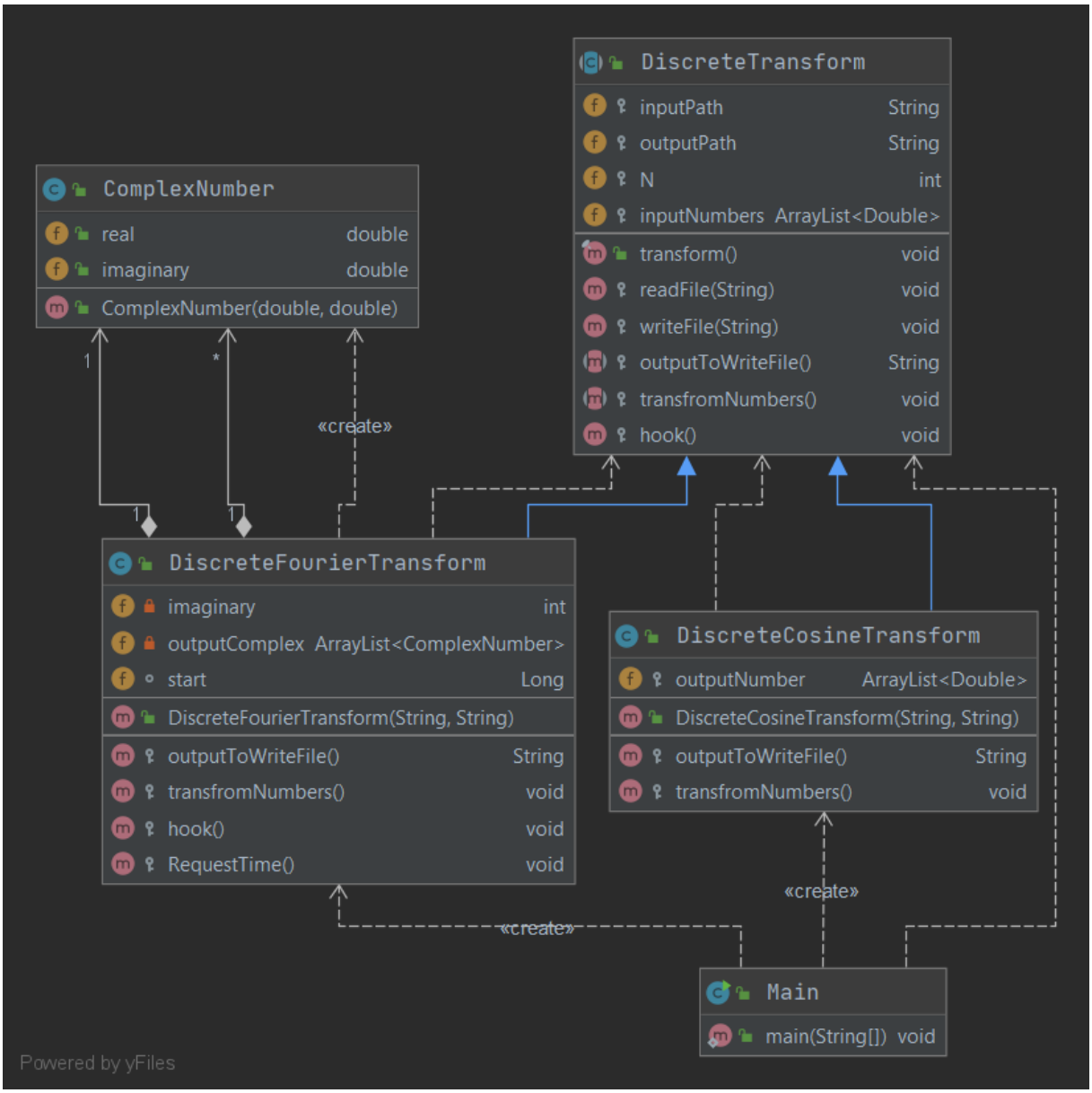
DFT:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{i2\pi}{N} kn} \\ = \sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N} kn\right) - i \cdot \sin\left(\frac{2\pi}{N} kn\right)\right], \quad \text{(Eq.1)}$$

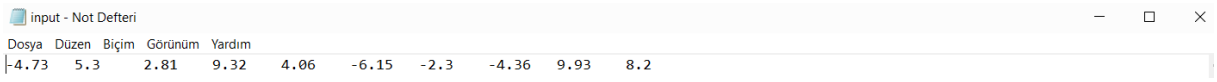
DCT:

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

Diagram:

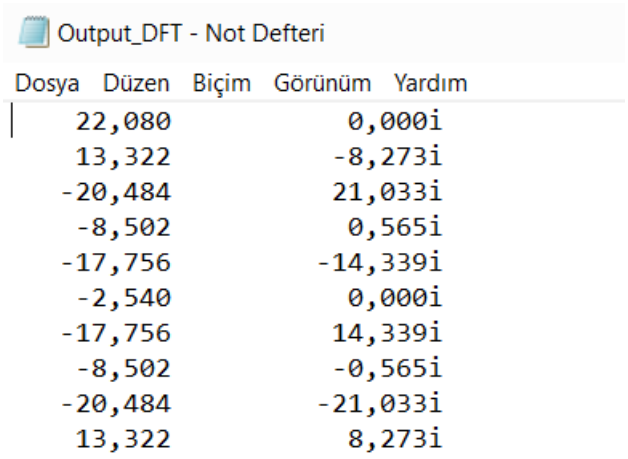

DiscreteTransform
inputPath String
outputPath String
N int
inputNumbers ArrayList<Double>
transform() void
readFile(String) void
writeFile(String) void
outputToWriteFile() String
transfromNumbers() void
hook() void
ComplexNumber
real double
imaginary double
ComplexNumber(double, double)
1
*
«create»
1
1
DiscreteFourierTransform
imaginary int
outputComplex ArrayList<ComplexNumber>
start Long
DiscreteFourierTransform(String, String)
outputToWriteFile() String
transfromNumbers() void
hook() void
RequestTime() void
DiscreteCosineTransform
outputNumber ArrayList<Double>
DiscreteCosineTransform(String, String)
outputToWriteFile() String
transfromNumbers() void
«create»
«create»
Main
main(String[]) void
Powered by yFiles

Input File:



Output Files:

DFT:



DCT:

Output Terminal:

DFT:

```
### Discrete Fourier Transform ###
    22,080       0,000i

    13,322      -8,273i

   -20,484      21,033i

    -8,502       0,565i

   -17,756     -14,339i

    -2,540       0,000i

   -17,756      14,339i

    -8,502      -0,565i

   -20,484     -21,033i

    13,322       8,273i

Do you want to know time of execution? (For yes press 'y' , or press any key)
y

Time of execution of Discrete Fourier Transform: 91ms


------------------------------------------------------------
```

DCT:

```
### Discrete Cosine Transform ###
    22,080

    -4,954

    10,114

   -33,427

    -4,209

     4,497

    -4,540

    -3,507

   -19,124

     4,880
```