GEBZE TECHNICAL UNIVERSITY
COMPUTER ENGINEERING

CSE 443
OBJECT ORIENTED ANALYSIS AND DESIGN
MIDTERM PROJECT REPORT

GÖKHAN HAS
161044067

INSTRUCTOR : PROF.DR. ERCHAN APTOULA

Part 1

While there is only one interface belonging to a single product family in the factory design pattern, there are different interfaces for different product families in the abstract factory. If we think of it as a factory, we can think of Factory DP as a factory where only one product is produced, and Abstract Factory DP as a factory where different products are produced. It is used to abstract the client side with the product family when we have to work with more than one product family. By separating the formation of product families from the client side, it enables us to establish a flexible and developable structure without decision-making conditions.

We learned that there are many classes in Abstract Factory DP. Approximately 35 classes were used here. If it is not fully visible in the UML diagram, it will appear more clearly in png format in the same directory.

Here, 6 items are given as a phone feature. These features are Display, Battery, Cpu and Ram, Storage, Camera and Case, respectively. There are also changing features as a model. There are varying features in different markets for the same model.

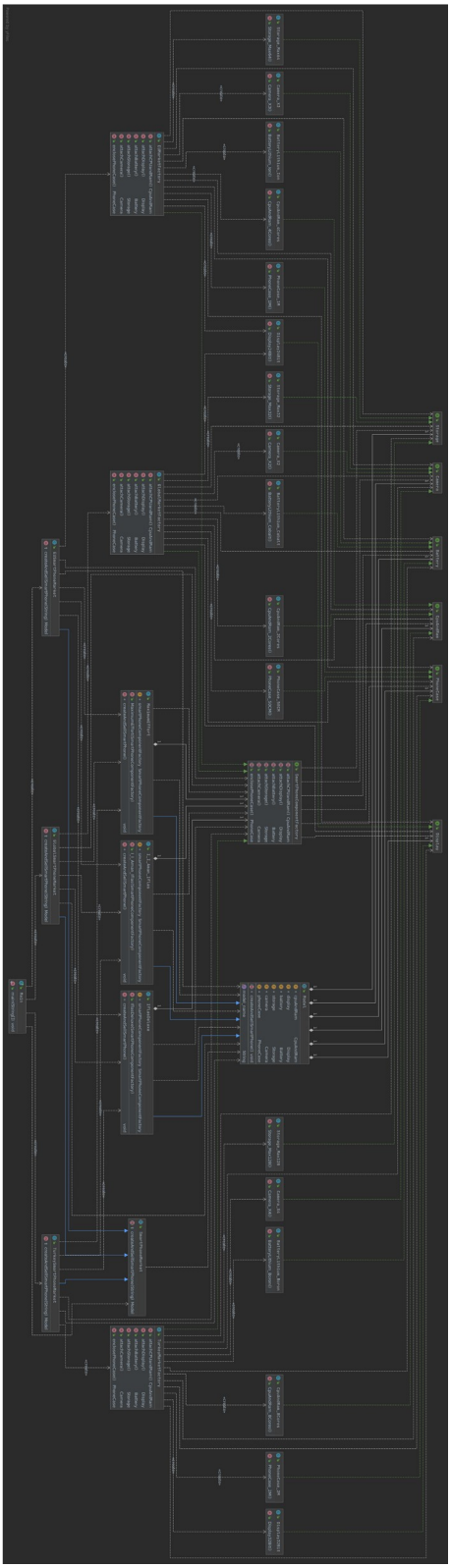
First of all, there are 6 interfaces with the same names for these features. These interfaces do not contain any code snippets. They are intentionally empty. These features will be used for markets. There are 3 markets (Turkey, EU and Global). There are 3 changing models for each market, excluding the display feature. This number is 2 on the display. These classes implement the related interface. There are only constructors in these classes. And among these, the brand-specific feature is printed on the screen with system.out.println.

Next, SmartPhoneComponentFactory interface was written. This interface only contains attach functions that return the 6 attribute references mentioned above. There are 3 classes derived from this interface. These are the factory classes of the relevant markets (Turkey, EU and Global). And these 6 functions return the changed property of the relevant market. For example, for the Turkey market, Display turns 32 bit; 24 bit returns for other grocery stores.

There is an abstract class called SmartPhoneMarket for coding the parts where phones are produced and sold. In this abstract class, there is a method createAndSellSmartPhone and this method is left as abstract. A separate SmartPhoneMarket class is defined for each market. And according to the model information, the class of the relevant model is called.

Likewise, there is a modern abstract class. In this model, there is a createAndSellSmartPhone function in abstract form. And in this abstract class, there are 6 model feature classes for features suitable for each model. There are 3 model classes from this class. And in the abstract function, a phone is produced and made ready for sale.

In diagrams directory, par1UML.png.



```
--------------------------------------------------
### FOR TURKEY MARKET ###
##### CREATING MAXIMUM-EFFORT MODEL #####
CPU AND RAM :   2.8GHZ, 8GB, 8 Cores
DISPLAY     :   5.5 inches, 32 Bit
BATTERY     :   27h, 3600mAh, Lithium-Boron
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


### FOR TURKEY MARKET ###
##### CREATING IFLASDELUXE MODEL #####
CPU AND RAM :   2.2GHZ, 6GB, 8 Cores
DISPLAY     :   5.3 inches, 32 Bit
BATTERY     :   20h, 2800mAh, Lithium-Boron
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


### FOR TURKEY MARKET ###
##### CREATING I-I-AMAN-IFLAS MODEL #####
CPU AND RAM :   2.2GHZ, 4GB, 8 Cores
DISPLAY     :   4.5 inches, 32 Bit
BATTERY     :   16h, 2000mAh, Lithium-Boron
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
```

```
--------------------------------------------------
### FOR EU MARKET ###
##### CREATING MAXIMUM-EFFORT MODEL #####
CPU AND RAM :   2.8GHZ, 8GB, 4 Cores
DISPLAY     :   5.5 inches, 24 Bit
BATTERY     :   27h, 3600mAh, Lithium-Ion
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


### FOR EU MARKET ###
##### CREATING IFLASDELUXE MODEL #####
CPU AND RAM :   2.2GHZ, 6GB, 4 Cores
DISPLAY     :   5.3 inches, 24 Bit
BATTERY     :   20h, 2800mAh, Lithium-Ion
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


### FOR EU MARKET ###
##### CREATING I-I-AMAN-IFLAS MODEL #####
CPU AND RAM :   2.2GHZ, 4GB, 4 Cores
DISPLAY     :   4.5 inches, 24 Bit
BATTERY     :   16h, 2000mAh, Lithium-Ion
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
```

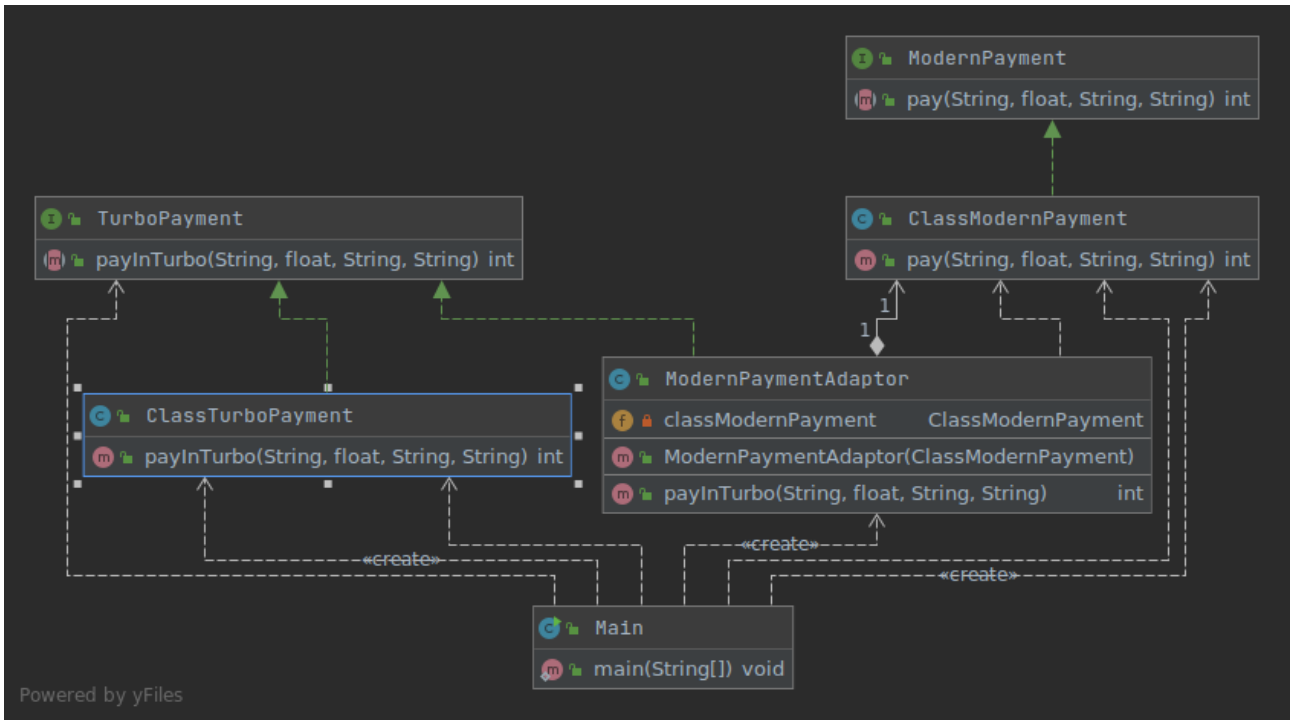
```
--------------------------------------------------
### FOR GLOBAL MARKET ###
##### CREATING MAXIMUM-EFFORT MODEL #####
CPU AND RAM :   2.8GHZ, 8GB, 2 Cores
DISPLAY     :   5.5 inches, 24 Bit
BATTERY     :   27h, 3600mAh, Lithium-Cobalt
STORAGE     :   MicroSD support, 64GB, MAX 32 GB
CAMERA      :   12mp front, 8Mp rear, Opt. zoom x2
CASE        :   151x73x7.7 mm, dustproof, waterproof, aluminum, Waterproof up to 50cm


### FOR GLOBAL MARKET ###
##### CREATING IFLASDELUXE MODEL #####
CPU AND RAM :   2.2GHZ, 6GB, 2 Cores
DISPLAY     :   5.3 inches, 24 Bit
BATTERY     :   20h, 2800mAh, Lithium-Cobalt
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


### FOR GLOBAL MARKET ###
##### CREATING I-I-AMAN-IFLAS MODEL #####
CPU AND RAM :   2.2GHZ, 4GB, 2 Cores
DISPLAY     :   4.5 inches, 24 Bit
BATTERY     :   16h, 2000mAh, Lithium-Cobalt
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

The Adapter Design pattern is applied to reuse an existing class or interface class by adapting it to a different interface class available. Most of the time we would like to reuse an existing class in our system that we think will work. However, the current class may not be exactly what we expected. In this case, we can write an adapter in between and adapt the current class to our system. Thus, we support a similar interface without any code changes in the adapted object. In addition, during the adaptation process, features that are not supported by the adapted object can also be implemented by the adapter.

In addition, during the adaptation process, the features that support the adapted object can be performed on the Adapter side. Transform the interface of a class into the interface that customers want. The adapter allows classes to interoperate otherwise not due to incompatible interfaces. Wraps an existing class with an interface.

There are two classes called ClassModernPayment and ClassTurboPayment. These classes are classes that implement interfaces with the same name. The most important point here is the ModernPaymentAdaptor class. This class implements the TurboPayment interface. However, within the payInTurbo method, the pat method of the object with the ClassModernPayment reference is called to achieve its purpose. A main is written, where each function is tested one by one.




```
Paying with turbo payment method ...
-----------------------------------------------

Paying with modern payment method ...
-----------------------------------------------

It is adapter method, paying with actually modern payment method !
Paying with modern payment method ...
-----------------------------------------------


Process finished with exit code 0
```

Part 3

The command design pattern is based on wrapping the code structure that fulfills user requests and storing them as objects. There may be situations where the definition of the object to be worked on cannot be made. Under these conditions, it is not possible to predict what kind of solutions can be used to interfere with the object, but the operations to be performed are wrapped as an object. This spiral code structure kept as an object creates a solution for the recipient object. As a result of keeping solutions as objects, the command design pattern allows the same code structure to be used over and over again. That's why this design pattern is preferred for this question.

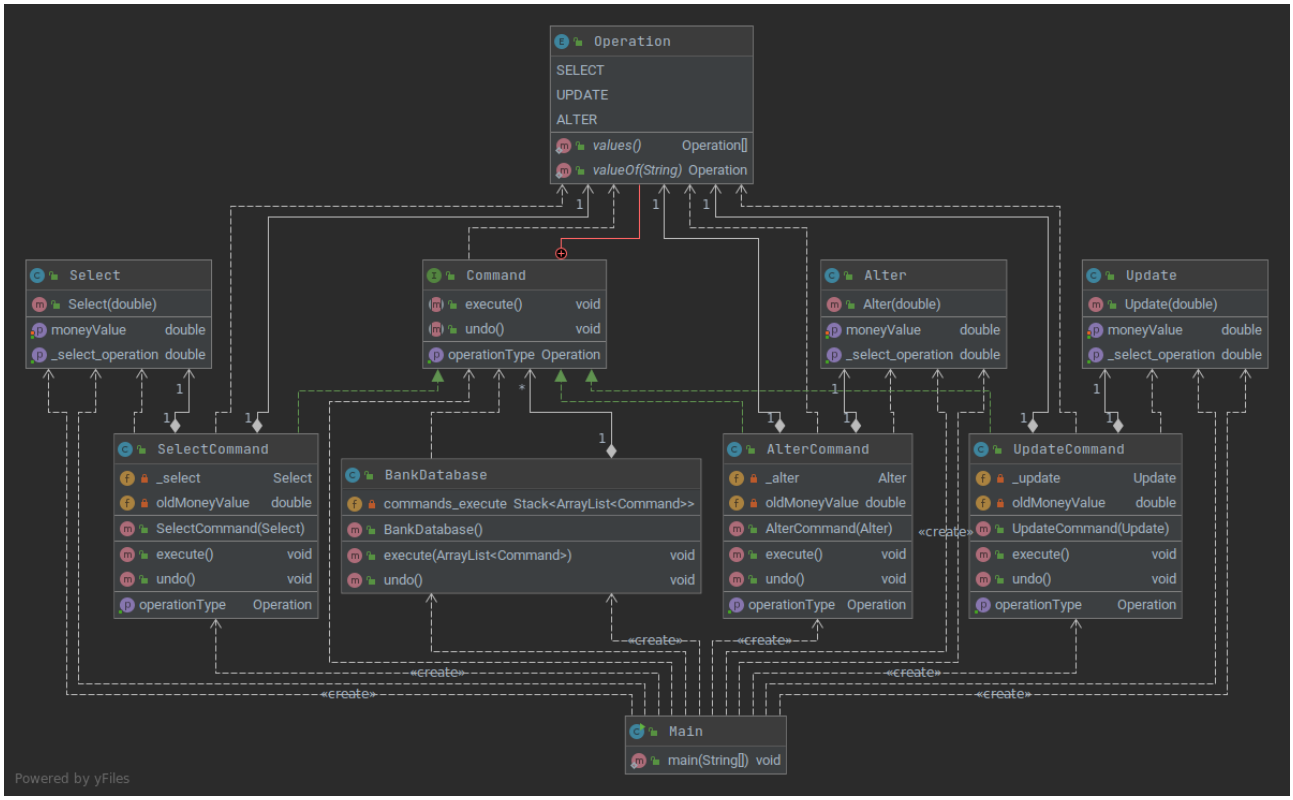
First, I defined an interface called Command. In this interface, I used a structure named enum Operation to indicate which type of command it is. Likewise, I used the functions execute () and undo () to provide undo. As I mentioned above, there is a getOperationType () method that returns a value of Operation enum.

Then there are 3 (Select, Alter, Update) classes because there are 3 types of operations. These classes can be thought of as simulating the database. Within these classes, they keep a variable called double moneyValue, considering that the user will keep the amount of money in the bank. This variable is also available in getter and setter functions. Then there is a get_select_operation () method that returns double. This method behaves as if doing something from the database. In fact, what it does is increase or decrease the user's money by a random value in each class.

Likewise, there are 3 classes (SelectCommand, AlterCommand and UpdateCommand) that derive from the Command interface. Each of these classes privately contains a variable belonging to its respective class. For example, there is a Select _select variable in the SelectCommand class. What remains is the implementation of the execute () and undo () functions. To implement these methods, the double oldMoneyValue variable is kept in these classes. Thus, it is ensured that the operation of the undo () method is performed in particular. Within the execute () method, the oldMoneyValue variable is assigned the value before the operation. If it is undo (), the old value is assigned to that command using set functions. Transactions are printed within these functions.

Finally, I wrote a BankDatabase class that I can simulate these operations. This class has one stack. However, this stack maintains a Command list. Here, rollback transactions are implemented. Likewise, the execute () and undo () methods are written. However, they roam over the Command list returned by the stack and call the execute () or undo () methods of those Commands. As for why I made the stack to keep a list. It was to allow each user to make multiple transactions and only roll back transactions in that transaction. Because, by using various combinations of these 3 transactions, more transactions (for example, a transaction with 6 transactions) could be created. Main and I have always tried 3 pieces, but as I said, I made it possible to do more.

In Main, I simulated the revocation of transactions for 2 users. Select, alter and update methods are called in different order. These methods are then treated as if they were canceled from the last time, and the undo () method is called through the BankDatabase object.




```
--------------- EXAMPLE 1 -------------------------
SELECT COMMAND EXECUTE :    VALUE IS : 8.5
UPDATE COMMAND EXECUTE :    VALUE IS : 18.5
ALTER COMMAND EXECUTE  :    VALUE IS : 16.0
ALTER COMMAND UNDO     :    VALUE IS : 16.0
UPDATE COMMAND UNDO    :    VALUE IS : 18.5
SELECT COMMAND UNDO    :    VALUE IS : 8.5

--------------- EXAMPLE 2 -------------------------
SELECT COMMAND EXECUTE :    VALUE IS : 13.5
UPDATE COMMAND EXECUTE :    VALUE IS : 28.5
ALTER COMMAND EXECUTE  :    VALUE IS : 13.5
ALTER COMMAND UNDO     :    VALUE IS : 13.5
UPDATE COMMAND UNDO    :    VALUE IS : 28.5
SELECT COMMAND UNDO    :    VALUE IS : 13.5
-----------------------------------------
```

Part 4

The template method pattern is in the group of behavioral design patterns. Template method pattern defines the steps required for an operation abstractly and determines how the algorithm will work with a template method. Subclasses implement one or more methods required for the algorithm within their own structure and enable the algorithm used to work according to their own wishes. Thus, by preventing code repetition, reusability of the code and an arrangement to be made in the algorithm framework are provided from a single place. For example, with a method that will be put in the parent class and defined, it can be made to interfere with the flow of the subclass.

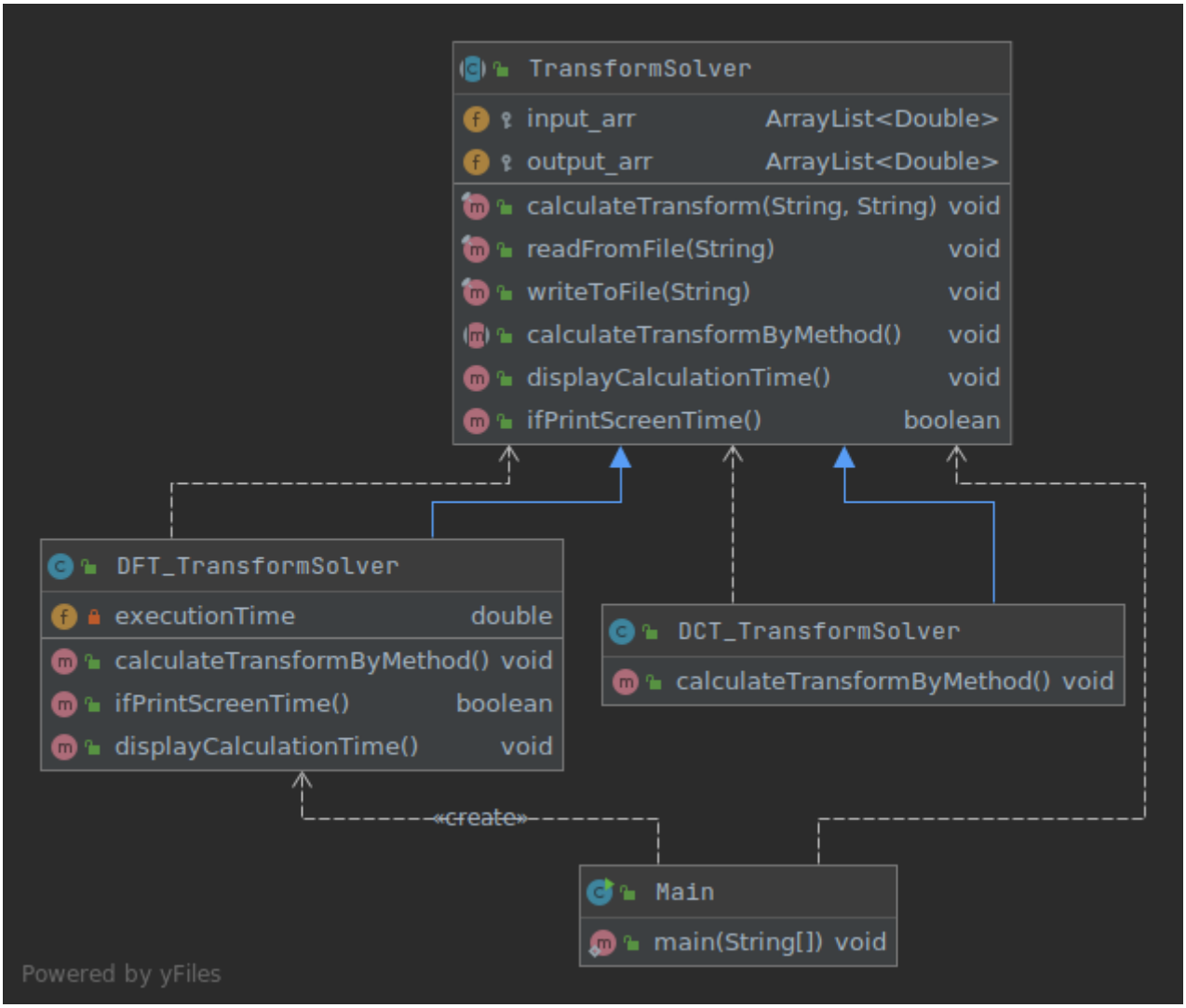
The abstract class here is the TransformSolver method. This class keeps two double ArrayLists as private variables. These lists hold the input and output results. And it is made protected so that subclasses can access and change.

The main final method is the calculateTransform method. In this method, the process of reading from the file, calculating the result, writing the result to the file and pressing the calculation time on the screen, if desired, takes place. Since this is a class final method, it cannot be rewritten in subclasses. Likewise, the functions of reading from file and writing to file have been finalized. These classes are not implemented again in subclasses.

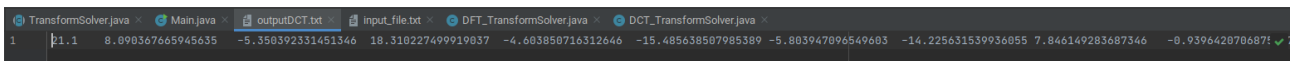
The calculateTransformByMethod () method is an abstract method. DFT_TransformSolver and DCT_TransformSolver classes are implemented according to the related method.

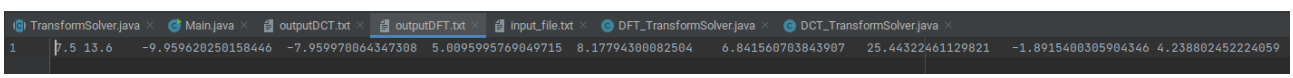
The displayCalculationTime () function was written to have a hook function. This function will be rewritten in the DFT_TransformSolver class. It also has ifPrintScreenTime () function. And by default it returns false. DFT and DCT is coded with the help of the links in the midterm pdf.

NOTE ! In order to make calculations with DFT, the input file has been accepted as follows. For example, say 15 20 -7 9 4 8. Numbers are separated by Tab characters. When calculating the DFT, the first number here is 15 + 20i. It is calculated in pairs. Other numbers are -7 + 9i and 4 + 8i. Here N would be 3. This format is preferred because it is left to us. And if there is no even number, the DFT calculation function will fail. This issue should be considered, as it may change the results. In DCT, every number is considered as a normal real number.




```
Calculating DFT Transform :
Do you want time of the execution printed on screen ? [ y - n ]
y
Time of the execution is : 0.125364 ms
Calculating DCT Transform :

Process finished with exit code 0
```





Screenshots in screenshots directory.