Object Oriented Analysis and Design

MIDTERM REPORT

Ahmet Yuşa Telli
151044092 | AYTELLİ@GTU.EDU.TR

Part 1:

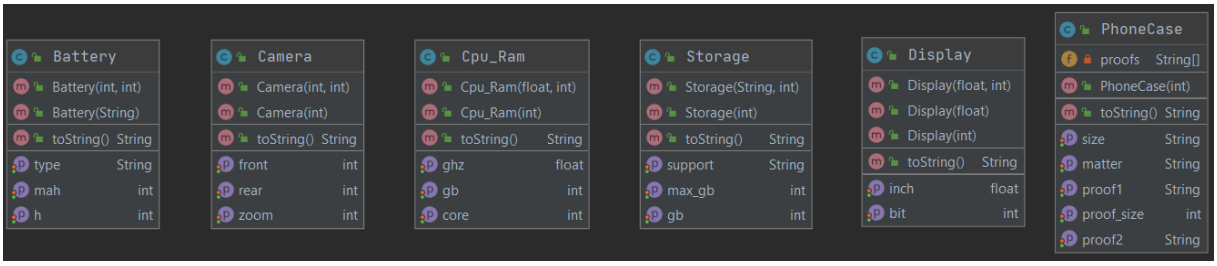
The first part we implement “Abstract Factory Design Pattern”. We have a company which name is Iflas Technologies Ltd. This company sells smartphones. There are 3 different types of smart phones: MaximumEffort, IflasDeluxe, I-I-Aman-Iflas. Each one has a display, a battery, a cpu-ram board, a storage, a camera board and a case.

This company have three factories in three different regions: Turkey, EU, Global, and have three different market for these regions. These factories have different types of displays, batteries, cpu-ram boards, storages, camera boards and cases. For instance, MaximumEffort smartphone has a 5.5 inches display but if it creates in Turkey Factory it is 32bits, if it creates in EU or Global it is 24 bits. Factories have differences like this.

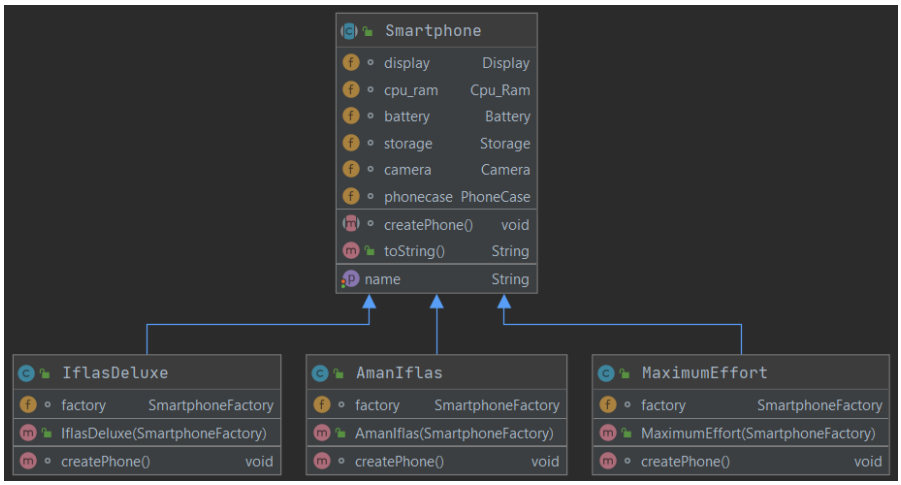
What we did?

We have some classes of smartphone objects. They are cpu_ram, display, battery, storage, camera, phone_case. They have their information.

- Cpu_Ram class has ghz, gb and core numbers. Ghz is float number, gb and core are integer numbers.
- Display class has inch and bit numbers. Inch is a float number; bit is an integer number.
- Battery class has H, mah numbers and a type. Type is a string and H and mah are integers.
- Storage class has type, gb and max_gb. Type is string, gb and max_gb are integers.
- Phone case class has size, proofs, matter and proof_size. Proof_size is integer and others are strings.

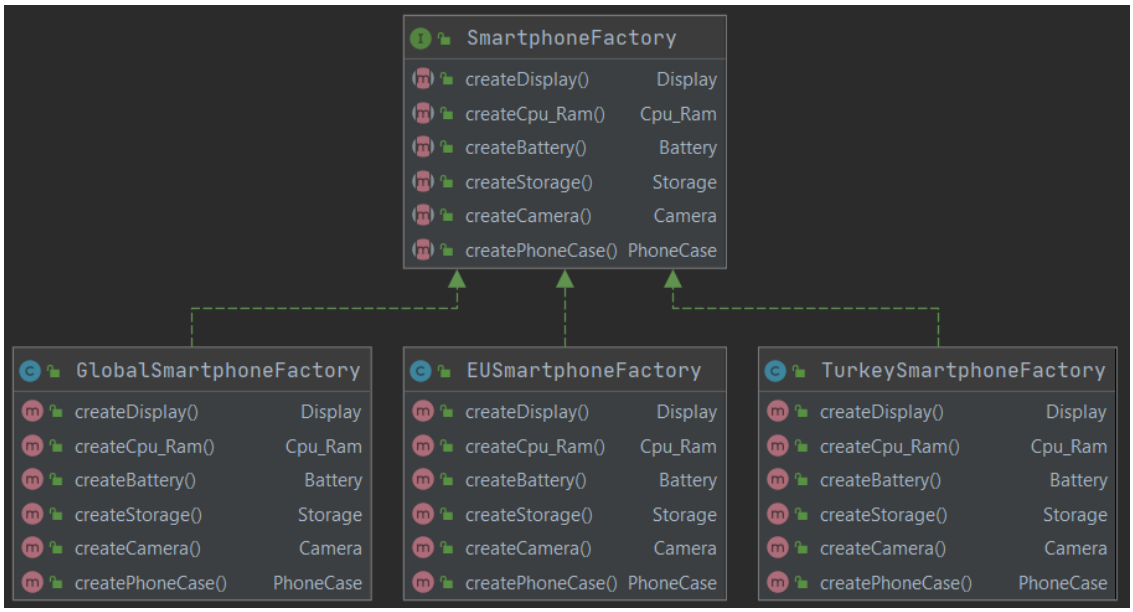


We have one abstract class which name is Smartphone. The types of smartphones are extending from this class. This abstract class includes objects of smartphones: display, battery, cpu_ram, storage, camera, phone_case. It has one important abstract method. createPhone() method is abstract and the types of smartphones classes override this method by themselves.



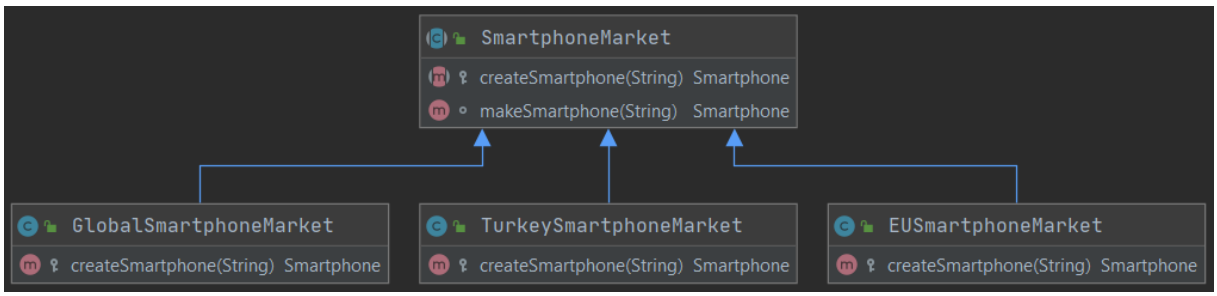
We have 3 classes for types of smartphones: IflasDeluxe, MaximumEffort, AmanIflas. These classes have a factory type (SmartphoneFactory) in their fields and their constructor takes a type of factory object. For create a smartphone, we override the abstract method which name is createPhone() in Smartphone abstract class. When we create a smartphone, we call factory's methods and we add our specific features to smartphone's objects. For instance, when we create an IflasDeluxe model in Turkey Factory, we take a TurkeySmartphoneFactory object and call its methods.

We have a SmartphoneFactory interface. It has smartphone's objects: display, cpu_ram, battery etc. We have three different regions: Turkey, EU, Global and we implement their classes from this interface. These classes have their own special features. For instance, TurkeySmartphoneFactory implements from this interface and in TurkeySmartphoneFactory class, we add special features to smartphone objects, like display object has 32 bits, cpu_ram object has 8 cores etc.



Apart from these we have an abstract class SmartphoneMarket. This class has an abstract method (createSmartphone()) and one normal method (makeSmartphone()). The abstract method returns a Smartphone object, we implement this abstract method into the market classes of the regions. They choose the type of smartphone and give to SmartphoneMarket class in the other method.

The other method is makeSmartphone(), it calls the abstract method and take a smartphone object (IflasDeluxe or MaximumEffort or AmanIflas). Then, the SmartphoneMarket class calls this smartphone object's method (createPhone()), this method is in Smartphone abstract class and implemented in smartphone models classes.



In main, we create a region market object. Using this market object, we create a smartphone object with model name.

Main Output:

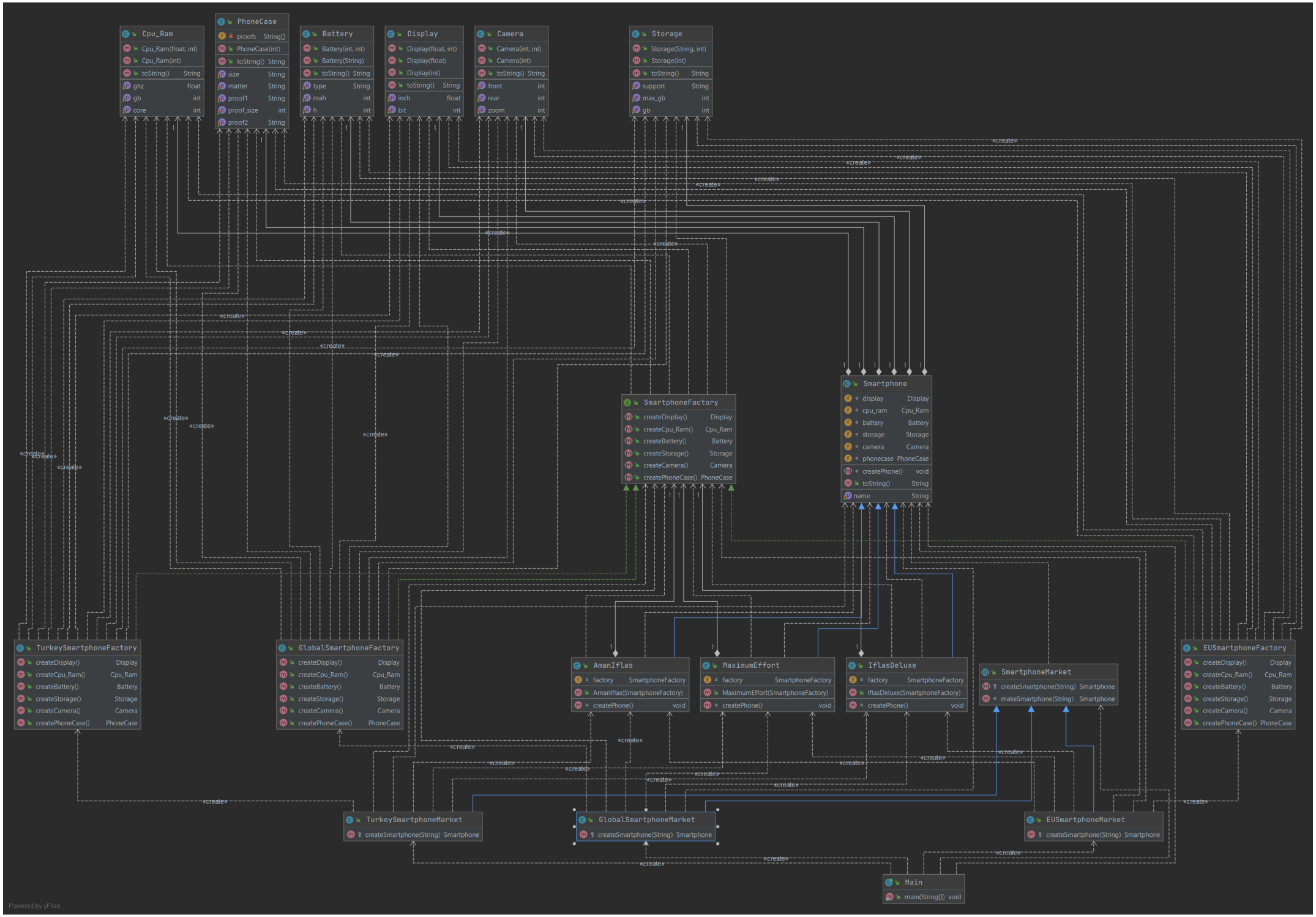
```
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
Iflas Delux in Turkey Factory
CPU RAM Board: 2.2 Ghz, 6 GB, 8 Cores.
Display 5.3 inch 32 bit.
Battery: 20 H, 2800 mAh, Lithium-Boron type.
Storage: MicroSD Support, 32 GB,128 Max GB.
Camera: 12 MP Front, 5 MP Rear, 4x Opt. Zoom.
Phone Case: 149x73x7.7 mm, waterproof, aluminum matter.
Waterproof up to 200 cm.


I-I-Aman Iflas in EU Factory
CPU RAM Board: 2.2 Ghz, 4 GB, 4 Cores.
Display 4.5 inch 24 bit.
Battery: 16 H, 2000 mAh, Lithium-Ion type.
Storage: MicroSD Support, 16 GB,64 Max GB.
Camera: 8 MP Front, 5 MP Rear, 3x Opt. Zoom.
Phone Case: 143x69x7.3 mm, waterproof, plastic matter.
Waterproof up to 100 cm.


Maximum Effort in Global Factory
CPU RAM Board: 2.8 Ghz, 8 GB, 2 Cores.
Display 5.5 inch 24 bit.
Battery: 27 H, 3600 mAh, Lithium-Cobalt type.
Storage: MicroSD Support, 64 GB,32 Max GB.
Camera: 12 MP Front, 8 MP Rear, 2x Opt. Zoom.
Phone Case: 151x73x7.7 mm, dustproof, waterproof,
 aluminum matter.
Waterproof up to 50 cm.
```

In Main:

We create a TurkeySmartphoneMarket object. This object calls its makeSmartphone() method and gives IflasDeluxe model. This method returns a smartphone object and print its features.

Second, we create a EUSmartphoneMarket object. This object calls its makeSmartphone() method and gives Aman-Iflas model. This method returns a smartphone object and print its features.

The last example is created GlobalSmartphoneMarket object. This object calls its makeSmartphone() method and gives MaximumEffort model. This method returns a smartphone object and print its features.


Cpu_Ram
Cpu_Ram(float, int)
Cpu_Ram(int)
toString() String
ghz float
gb int
core int
PhoneCase
proofs String[]
PhoneCase(int)
toString() String
size String
matter String
proof1 String
proof_size int
proof2 String
Battery
Battery(int, int)
Battery(String)
toString() String
type String
mah int
h int
Display
Display(float, int)
Display(float)
Display(int)
toString() String
inch float
bit int
Camera
Camera(int, int)
Camera(int)
toString() String
front int
rear int
zoom int
Storage
Storage(String, int)
Storage(int)
toString() String
support String
max_gb int
gb int
«create»
SmartphoneFactory
createDisplay() Display
createCpu_Ram() Cpu_Ram
createBattery() Battery
createStorage() Storage
createCamera() Camera
createPhoneCase() PhoneCase
Smartphone
display Display
cpu_ram Cpu_Ram
battery Battery
storage Storage
camera Camera
phonecase PhoneCase
createPhone() void
toString() String
name String
TurkeySmartphoneFactory
createDisplay() Display
createCpu_Ram() Cpu_Ram
createBattery() Battery
createStorage() Storage
createCamera() Camera
createPhoneCase() PhoneCase
GlobalSmartphoneFactory
createDisplay() Display
createCpu_Ram() Cpu_Ram
createBattery() Battery
createStorage() Storage
createCamera() Camera
createPhoneCase() PhoneCase
AmanIflas
factory SmartphoneFactory
AmanIflas(SmartphoneFactory)
createPhone() void
MaximumEffort
factory SmartphoneFactory
MaximumEffort(SmartphoneFactory)
createPhone() void
IflasDeluxe
factory SmartphoneFactory
IflasDeluxe(SmartphoneFactory)
createPhone() void
SmartphoneMarket
createSmartphone(String) Smartphone
makeSmartphone(String) Smartphone
EUSmartphoneFactory
createDisplay() Display
createCpu_Ram() Cpu_Ram
createBattery() Battery
createStorage() Storage
createCamera() Camera
createPhoneCase() PhoneCase
TurkeySmartphoneMarket
createSmartphone(String) Smartphone
GlobalSmartphoneMarket
createSmartphone(String) Smartphone
EUSmartphoneMarket
createSmartphone(String) Smartphone
Main
main(String[]) void
Powered by yFiles

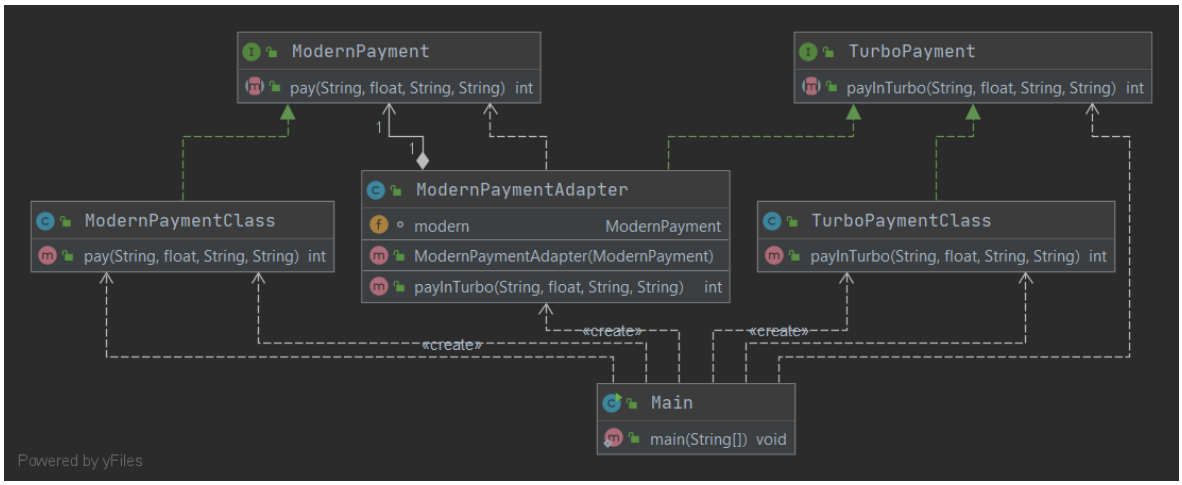
Part 2:

In the second part, we implement a “Adapter Design Pattern”. We have an old interface which name is “TurboPayment” and it has an old method which name is payInTurbo. We want to use this method using a ModernPayment interface’s pay method. We need an adapter class between TurboPayment and ModernPayment.

First of all, we create two classes from these interfaces and implement their methods. ModernPaymentClass implemented Pay method and TurboPaymentClass implemented PayInTurbo.

We need an adapter class. This class implements from TurboPayment interface and in field we have a ModernPayment interface object. We override PayInTurbo methods but in this method, we call the ModernPayment’s Pay method.

In test, we create an adapter class object and this object calls patInTurbo method, but it uses ModernPayment Pay method in background.



Main Output:

```
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
Modern Payment Pay:
CardNo: 123
Amount: 5.5
destination ModernDest.
installments: ModernInstall

Turbo Payment PayInTurbo: TurboCardNo: 1111
turboAmount: 11.1
destinationTurboOfCourse: TurboDest
installmentsButInTurbo: TurboInstall

In adapter class, Call Modern Payment Pay method.
Modern Payment Pay:
CardNo: 1111
Amount: 11.1
destination TurboDest
installments: TurboInstall

Process finished with exit code 0
```

Run ModernPaymentClass’s Pay method.

Run TurboPaymentClass’s PayInTurbo method.

Run ModernPaymentAdapter Class’s PayInTurbo method.

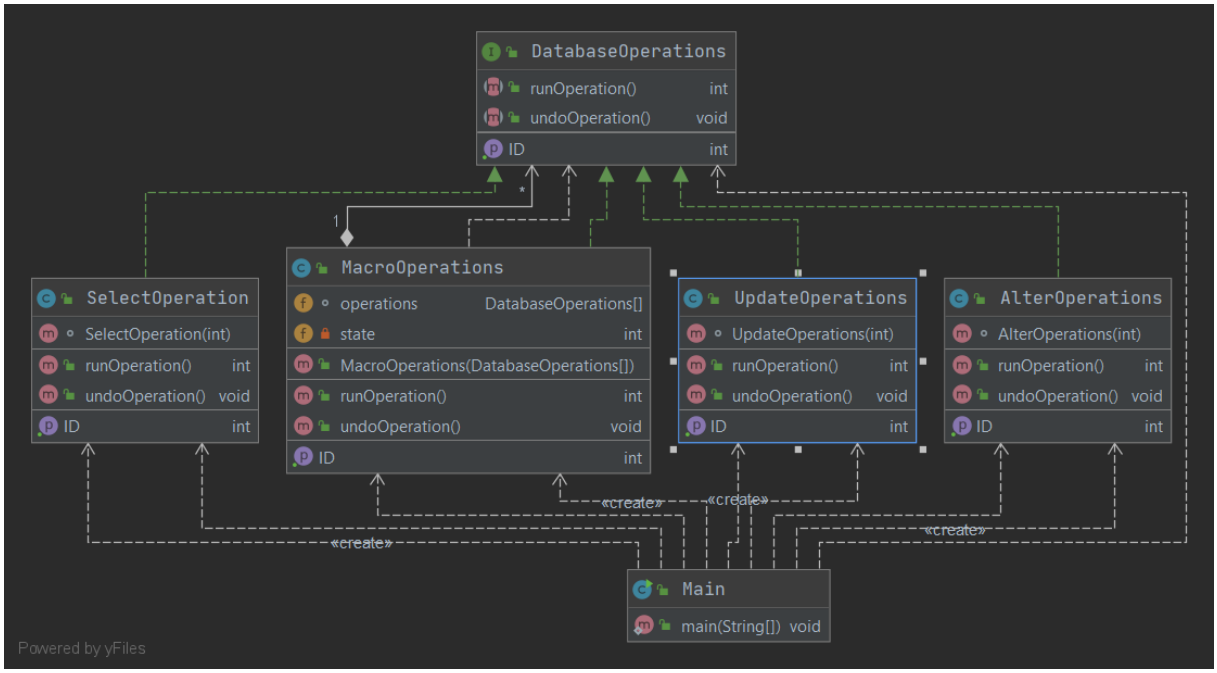
Part 3:

We implement a “Command Design Pattern” for the solution.

We have a database and we need to do operations in a series. Select, alter and update operations implements from DatabaseOperations interface. Each operation has an ID, a run method and an undo method.

The run operations generate a random integer between 1 and 0. If a run method returns 1, we understand it is success, but if it returns 0, we understand it is failed. If the run method is failed, we should undo previous operations.

We have a MacroOperation class and implement from DatabaseOperations. Transactions must be unbreakable. For this reason, in this class we have an operations array. Its constructor takes an operations array. Then, run all operations’ run methods. While running operations, if one operation is failed, it does not continue to run the others and call previous operation’s undo methods.



Main Output:

```
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
SELECT 1 run.
ALTER 1 run.
ALTER 2 run.
One operation is failed. Undo all operations.
ALTER 2 undo.
ALTER 1 undo.
SELECT 1 undo.

SELECT 1 run.
SELECT 2 run.
ALTER 1 run.
SELECT 3 run.
ALTER 2 run.
SELECT 3 run.

Process finished with exit code 0
```

First example MacroOperation is run and Alter2 operation is failed. Undo previous operations.

Second example all transactions completed successfully.

Part 4:

In the last part, we implement DFT and DCT calculation for 1D array. We use “Template Method Design Pattern” for this part.

We have an abstract DiscreteTransform class. It has 5 methods and 4 fields. One abstract method is tranformNumbers(), this method will override in DFT and DCT classes. The calculateTransform() method, we design the calculation steps. First read_file(), tranformNumbers(), write_file() and hook().

First, we read input from file. We do not want it to be changeable. For this reason, we made it final method. The input file should include complex numbers like:

1-1i 2-1i 6-1i -1+2i 5+1i 2+5i -4-2i 3+3i 6-6i 4+5i

We split real and imaginary parts and save our arraylist which type is ComplexNumber.

Then, we call transformNumbers() method. This method runs different way for different types of transforms. DFT and DCT override this method by themselves.

After calculation, we write to file the result. We do not want it to be changeable. For this reason, we made it final method and implemented in DiscreteTransform class. Each transform classes have their own output file. The DiscreteTransform class write the result their own files.

At the end, we have a hook method and this method made to free other classes. In DFT class, we calculate an execution time and ask the user to show it.

Only calculation methods implemented in DCT and DFT. Other methods implemented in DiscreteTransform class. But DFT has a hook method for ask to user.

At the beginning, you should enter the input file’s name on terminal.

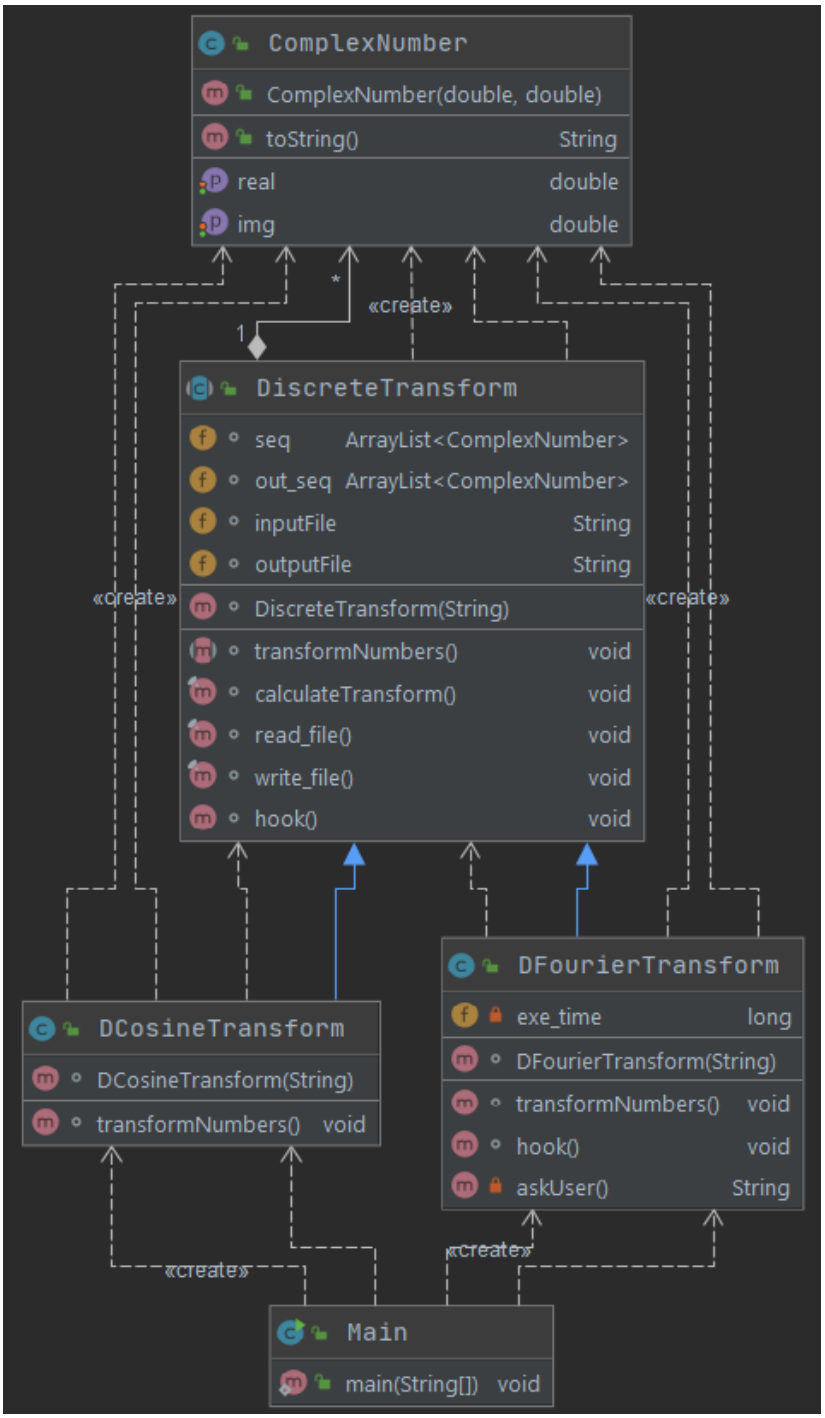
Main Output:

```
"C:\Program Files\Java\jdk1.8.0_271\bin\java.exe" ...
Enter file name:
input.txt
Read File in Abstract Class: ./res/files/input.txt
Transform in Fourier.
Write File in Abstract Class:./res/files/fourier.txt
Do you want to know time of execution for DFT?
y
Time of Execution: 1774800
Read File in Abstract Class: ./res/files/input.txt
Transform in Cosine
Write File in Abstract Class:./res/files/cosine.txt

Process finished with exit code 0
```

Enter a file name:

Firstly, run DFT and ask to the user to show the execution time.

Then, DCT run.



Ahmet Yuşa Telli

151044092