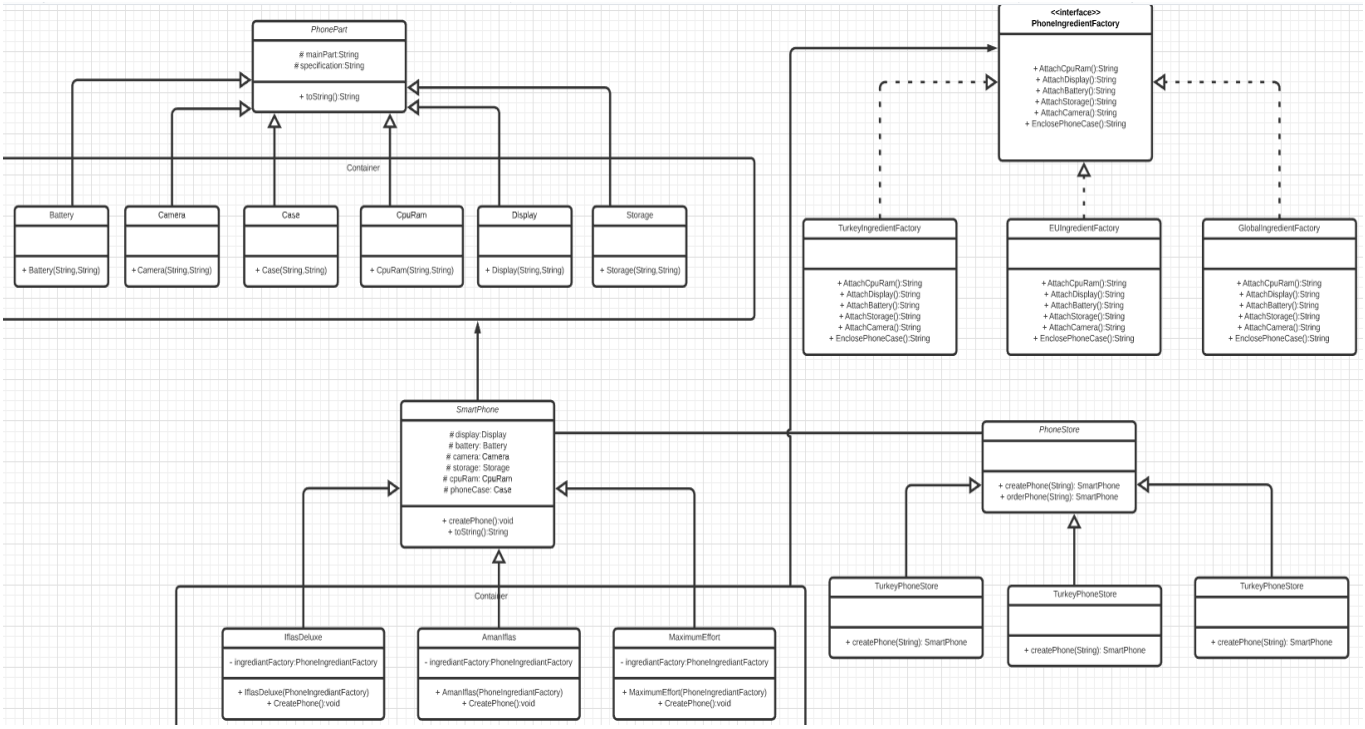
Part 1



My phones consist of 6 part so I create one class for each part. SmartPhone abstract class uses this pieces and 3 different smartPhone model extend this abstract class. Also I have 3 different factory for Turkey,Eu and Global. There is a has-a relation between those factories and smart phones. So depending of the factory that class has it create the phone according to appropriate factory.

For example:

```
public class IflasDeluxe extends SmartPhone{
    PhoneIngredientFactory ingredientFactory;

    public IflasDeluxe(PhoneIngredientFactory ingredientFactory) { this.ingredientFactory = ingredientFactory; }

    @Override
    void CreatePhone() {
        cpuRam = new CpuRam( main: "2.2GHz,6GB ",ingredientFactory.AttachCpuRam());
        display = new Display( main: "5.3 inches " , ingredientFactory.AttachDisplay());
        battery = new Battery( main: "20h,2800mAh " , ingredientFactory.AttachBattery());
        storage = new Storage( main: "MicroSD support,32 GB " , ingredientFactory.AttachStorage());
        camera = new Camera( main: "12Mp front,5mp rear " , ingredientFactory.AttachCamera());
        phoneCase = new Case( main: "149x73x7.7 mm waterproof,aluminum " , ingredientFactory.EnclosePhoneCase());
    }
```

In createPhone method a phone carried out in the following order. Attach CpuRam, attach display, attach battery, attach storage, attach camera and lastly enclose phone case. Also phone has a PhoneFactory as a data field. When creating phone parts using this factory corresponding part methods.

For instance when creating display if our factory is a Turkey factory it will return 32 bit, if its a global factory it will return 24 bit and so on. Other phones create part same as this one. With this design I can add or remove a phone model in my system easly without changing any existing code. Just create a new class for new model and extend it from SmartPhone abstract class thats it. Moreover we can add or remove a factory too without changing anything in existing code.

Also there is Store classes, there classes for client part. In main I create all possible phones with all possible factories and print them.

```
public class TurkeyPhoneStore extends PhoneStore {
    @Override
    protected SmartPhone createPhone(String type) {
        SmartPhone phone = null;
        PhoneIngredientFactory ingredientFactory = new TurkeyIngredientFactory();

        if(type.equals("IflasDeluxe")){
            phone = new IflasDeluxe(ingredientFactory);
            phone.CreatePhone();
        }
        else if(type.equals("AmanIflas")){
            phone = new AmanIflas(ingredientFactory);
            phone.CreatePhone();
        }
        else if(type.equals("MaximumEffort")){
            phone = new MaximumEffort(ingredientFactory);
            phone.CreatePhone();
        }
        return phone;
    }
}
```

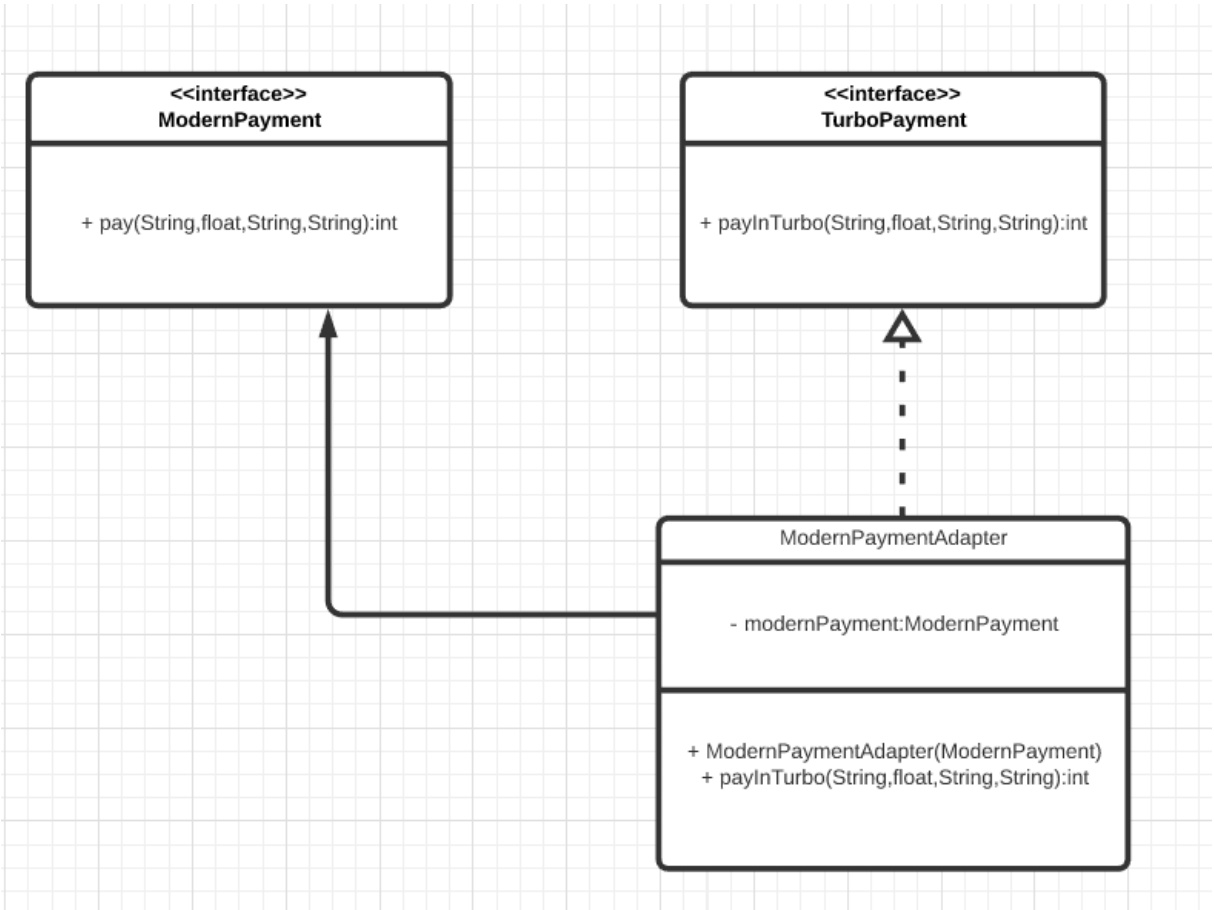
For example for Turkey store it take a String for phone model. If its IflasDeluxe it will create a IflasDeluxe model phone with corresponding turkey factory parts. If its a AmanIflas it will create an AmanIflas model phone in TurkeyFactory and so on. GlobalPhone and EUPhone stores same as this one. They just use their corresponding factory and create phone models.

And in main with using this stores I create all possible phone models with corresponding factory and print screen each step.

```
C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part1>javac Main.java

C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part1>java Main
Turkey-IflasDeluxe
2.2GHz,6GB 8 cores CpuRam created
5.3 inches 32 bit Display created
20h,2800mAh Lithum-Boron Battery created
MicroSD support,32 GB Max 128 GB Storage created
12Mp front,5mp rear Opt. zoom x4 Camera created
149x73x7.7 mm waterproof,aluminum Waterproof up to 2m Case created
display=5.3 inches 32 bit, Battery=20h,2800mAh Lithum-Boron, CpuRam=2.2GHz,6GB 8 cores, Storage=MicroSD support,32 GB Ma
x 128 GB, Camera=12Mp front,5mp rear Opt. zoom x4, Case=149x73x7.7 mm waterproof,aluminum Waterproof up to 2m

Global-IflasDeluxe
2.2GHz,6GB 2 cores CpuRam created
5.3 inches 24 bit Display created
20h,2800mAh Lithum-Cobalt Battery created
MicroSD support,32 GB Max 32 GB Storage created
12Mp front,5mp rear Opt. zoom x2 Camera created
149x73x7.7 mm waterproof,aluminum Waterproof up to 50cm Case created
display=5.3 inches 24 bit, Battery=20h,2800mAh Lithum-Cobalt, CpuRam=2.2GHz,6GB 2 cores, Storage=MicroSD support,32 GB M
ax 32 GB, Camera=12Mp front,5mp rear Opt. zoom x2, Case=149x73x7.7 mm waterproof,aluminum Waterproof up to 50cm
```

Part 2

We dont want to solve this problem by changing our existing code so we can create a class that adapts modernPayment to TurboPayment. We can use adapter design pattern for this. The adapter pattern converts the interface of a class into another interface the clients expect. That means we can convert modernPayment interface to TurboPayment with adapter and can use modernPayment pay method easly.

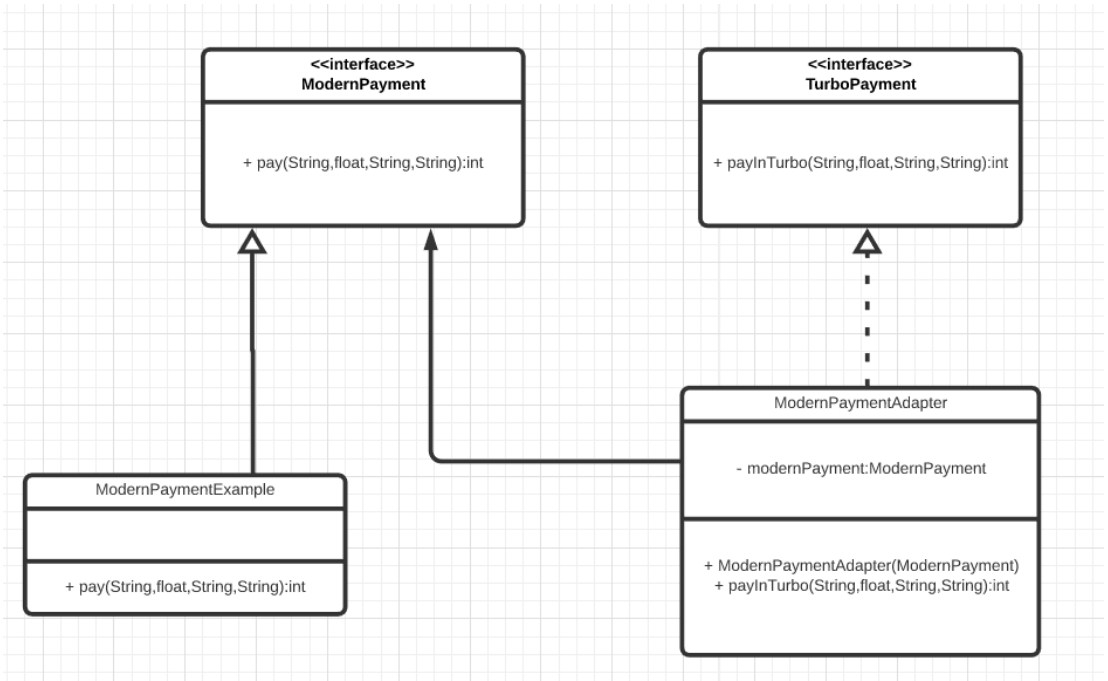


This is my design UML diagram. We can use an adapter that implements TurboPayment interface and has a ModernPayment object as a data field. Now in payInTurbo method we can call modernPayment pay method. This is very good because now we can adapt ModernPayment pay method with TurboPayment pay method without changing any existing code.

```
public ModernPaymentAdapter(ModernPayment modernPayment) { this.modernPayment = modernPayment; }

@Override
public int payInTurbo(String turboCardNo, float turboAmount, String destinationTurboOfCourse, String installmentsButInTurbo)
    return modernPayment.pay(turboCardNo,turboAmount,destinationTurboOfCourse,installmentsButInTurbo);
}
```

Like this, whenever we call ModernPaymentAdapter payInTurbo method we just use ModernPayment pay method.

Also I add ModernPaymentExample concrete class that implements ModernPayment just Show my code works fine.



```
public static void main(String[] args) {
    TurboPayment turboPayment = new ModernPaymentAdapter(new ModernPaymentExample());
    turboPayment.payInTurbo( turboCardNo: "123", turboAmount: 12, destinationTurboOfCourse: "956", installmentsButInTurbo: "hello");
}
```

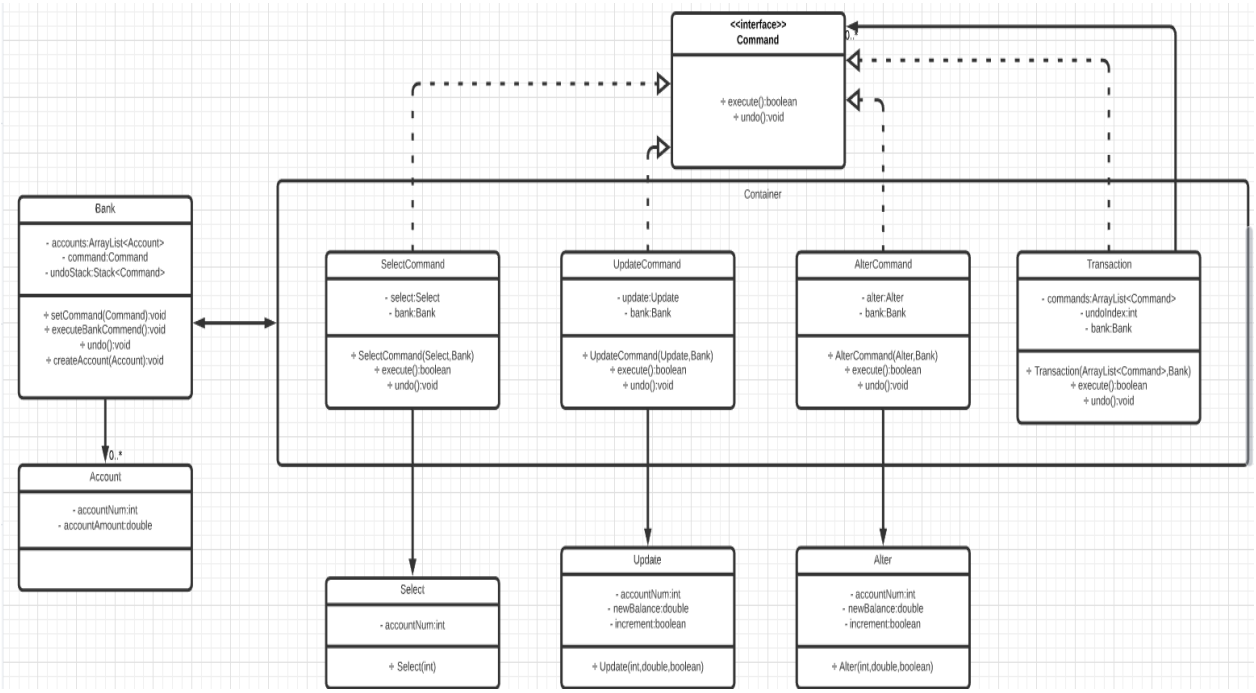
Now thanks to this design, We can call our ModernPayment pay method with using TurboPayment interface without changing any existing code.

```
C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part2>javac Main.java

C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part2>java Main
I am in pay method of ModernPayment
cardNo: 123
amount: 12.0
destination: 956
installments: hello

C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part2>
```

Part 3

a)I use command pattern in this part because there is 3 operations in this question I can encapsulate them as commands and can use execution and undo operations on them. Command pattern supports undo operation. Also thanks to command pattern I can use macro command and combine my Select, Update and Alter commands and write transactions.

So I use command pattern because of its easy to implement an undo operation in this design and with using its macro operation I can create transactions easily.



Here I can implements my Select, Update and Alter operations as Commands. Also Transaction class implement command interface too. With Transaction class I can create combination of my commands like a select followed with update and so on. My commands has a bank reference as data field because I want to access accounts in bank class so I can use this account in commands execution and undo operations.

There are also Select, Update and Alter classes. These classes hold informations about accounts and I use them in their corresponding command classes so that commands know which bank accounts they effect in execution method. Lasty, I create an Account class for holding accounts in Bank class so that I can Show commands can effect bank accounts in driver code.

b) I use a stack that holds command in Bank class for undo operations. So when a command in transaction fails I can undo executed operations by using that stack.

```
public void executeBankCommand(){
    boolean check = command.execute();
    if(!check){
        undo();
    }
}
```

Command execute method return boolean value. If it returns false it means there are some errors when executing command. Then if there are errors because of boolean value false it will execute undo method.

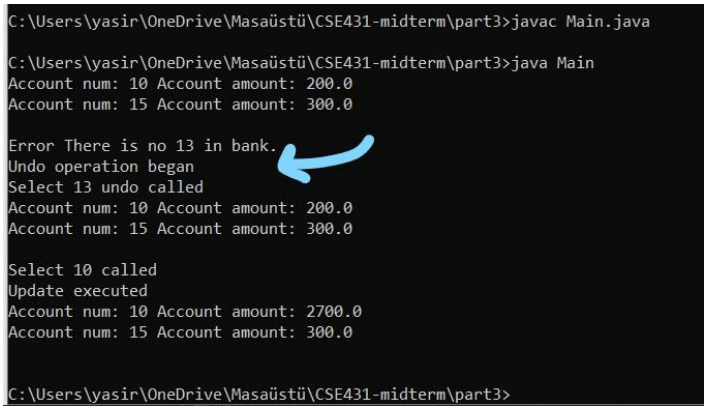
```
public void undo(){
    while(!undoStack.empty()){
        undoStack.pop().undo();
    }
}
```

In undo method we call undo operations of commends which stored in our stack.

```
public static void main(String[] args) {
    Bank bank = new Bank();
    bank.createAccount(new Account( acountNum: 10, acountAmount: 200));
    bank.createAccount(new Account( acountNum: 15, acountAmount: 300));
    System.out.println(bank);


    ArrayList<Command> macroCommand = new ArrayList<>();
    macroCommand.add(new SelectCommand(new Select( accountNum: 13),bank)); //This create an error and trigger undo stack
    macroCommand.add(new UpdateCommand(new Update( accountNum: 10, newBalance: 2500, increment: true),bank));
    Command command = new Transaction(macroCommand,bank);

    bank.setCommand(command);
    bank.executeBankCommand();
    System.out.println(bank);

    ArrayList<Command> macroCommand2 = new ArrayList<>();
    macroCommand2.add(new SelectCommand(new Select( accountNum: 10),bank));
    macroCommand2.add(new UpdateCommand(new Update( accountNum: 10, newBalance: 2500, increment: true),bank));
    Command command2 = new Transaction(macroCommand2,bank);

    bank.setCommand(command2);
    bank.executeBankCommand();
    System.out.println(bank);
}
```

For example, I create 2 accounts for my bank then print bank accounts in main method. Then I create a transaction with using macroCommand which contain a Select followed by Update command. But there is no bank account that has 13 as accountNum in bank so select commend produce an error then immediately undo operation called and update command won't called.

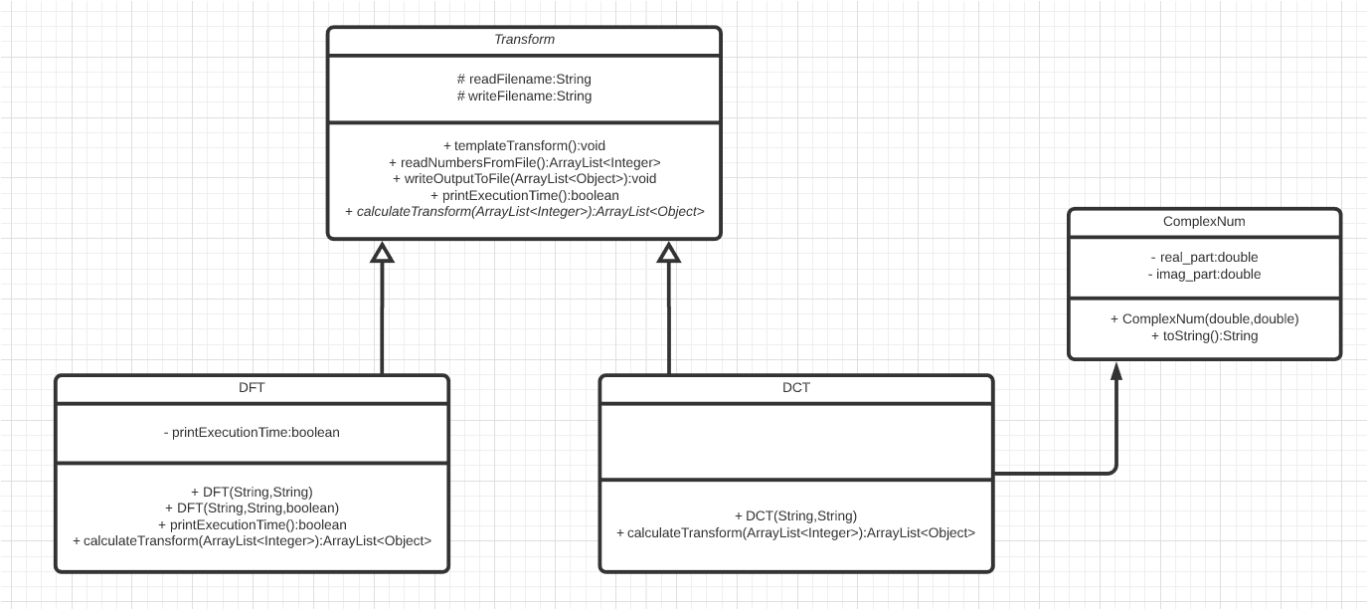
So select operation undo called and bank accounts didn't change because of update method didn't called.



But in second Transaction which consist of a Select command followed by Update will work withour error. So it print Select called and Update executed. After that I print bank account that has 10 as account number to Show its amount change from 200 to 2700.

Part 4

For this part, both DFT and DCT has some similar steps but they also does different things so template method pattern is a good choice in this situation. Both of them read numbers from a file and write transformed numbers to another file but transform operations different. Because of that I create a super class called Transform and implement read file and write file methods in this abstract class and declare calculateTransform abstract method because it's implementation different both in DFT and DCT. DFT and DCT classes have to implement this method as appropriate.



Also there is a method called templateTransform.

```
public final void TemplateTransform() throws IOException {
    ArrayList<Integer> numbers = readNumbersFromFile();
    ArrayList<Object> transformNums = calculateTransform(numbers);
    writeOutputToFile(transformNums);

    if(printExecutionTime()){
        Date date = new Date();
        System.out.println(date.toString());
    }
}
```

This method is my templete method. When called by using other methods it read file calculate transforms then write them corresponding output file. I declare this method as final so sub classes cant override it. Also there is printExecutionTime method. I use this method as a hook. I implement this method in Transform abstract class. It just return false default but I override it in DFT class so if you want, it can return true for some DFT objects. If it return true it just print the execution time on screen. There is 2 constructer for DFT. You can create DFT objects with using 3 arguments constructer with true value as third argument. Then it will trigger printExecutionTime true because of overrided printExecutionTime method and will print execution time on screen. If you create this by using 2 argument constructer or 3 arguments construter

with false value as third argument it will not print execution time on screen just like all other DCT object

```
public static void main(String[] args) throws IOException {
    //DFT dft = new DFT(args[0], "DFToutputFile"); In this case it will not print execution time to screen
    DFT dft = new DFT(args[0], writeFile: "DFToutputFile", printExecutionTime: true); // For this execution time will be printed
    dft.TemplateTransform();

    DCT dct = new DCT(args[0], writeFile: "DCToutputFile");
    dct.TemplateTransform();
}
```

For example, if you create dft object like in the first line it will not print execution time on screen but for second object it will print because it create object with using 3 arguments constructer and third argument true. On the other hand all dct object not print execution time on screen no matter what because I didn't override printExecutionTime method for DCT and printExecutionTime method return false default in Transform abstract class.

In addition, there is one more class called ComplexNum. I use this class for DFT transform calculation because transform numbers to complex numbers.

Because of DFT return complex numbers and DCT return numbers I use ArrayList<Object> for return type in calculateTransform method so in both situation I can use it.

```
C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part4>javac Main.java

C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part4>java Main inputFile
Fri Dec 04 23:34:38 EET 2020

C:\Users\yasir\OneDrive\Masaüstü\CSE431-midterm\part4>
```

After execution of above program, input file numbers convert and printed DFToutputFile for DFT and DCToutputFile for DCT, also I wanted to print execution time for my DFT it will print execution time on screen.