

# Midterm Project Report

**Elif Akgün**

1801042251

## 1. Part 1

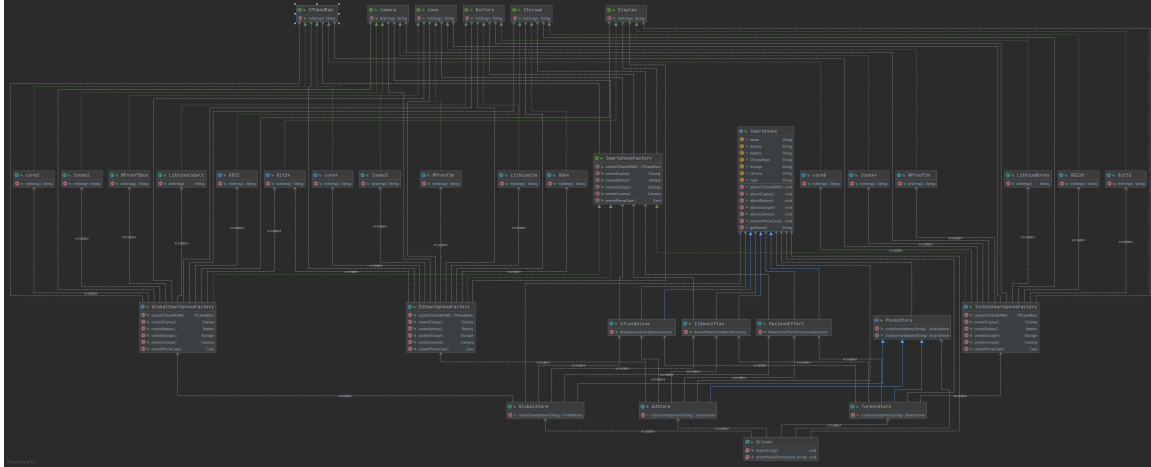
### 1.1 Design Explanation

In this part, we have got the same product families (display, battery, CPU & Ram, storage, camera and case) but different implementations. For example, Turkey uses one set of ingredients and EU another. All market's smartphones made from the same components, but each market has a different implementation of those components. We need to create ingredient families not ingredients so I used Abstract Factory design pattern.

I created an interface for each component (Display, Battery etc.). Then I created concrete classes of those interfaces. For example, I created LithiumBoron, LithiumCobalt and LithiumIon classes that implement Battery interface. Also I created a factory to create our ingredients which is SmartphoneFactory; the factory is responsible for creating each ingredient in the ingredient family. For each ingredient I defined a create method in the interface. Then I created concrete factory classes for each market that implement SmartphoneFactory which are TurkeySmartphoneFactory, EUSmartphoneFactory, and GlobalSmartphoneFactory. These classes create own ingredients. I created an abstract class which is Smartphone. It holds a set of ingredients that are used in production. Also it has some methods to attach cpu ram to the board, attach display, attach battery, attach storage, attach camera and enclose the phone case. There are three subclasses of Smartphone which are MaximumEffort, IffasDeluxe, and IIAmanIffas. They have a SmartphoneFactory object and a constructor with SmartphoneFactory parameter. In constructor, it sets data field of classes with appropriate ingredients. Thanks to polymorphism it goes proper factory and creates proper ingredient at run time. Also there is another abstract class PhoneStore. It has two methods: orderSmartphone and an abstract method createSmartphone. orderSmartphone method has a string type parameter. First, it creates asked Smartphone object at run time. Then it produces smartphone step by step. Lastly, there are three classes that extends PhoneStore class: TurkeyStore, EUStore, and GlobalStore. They have only createSmartphone method and they create proper smartphone object according to parameter. For example if parameter(item) equals MaximumEffort, it creates MaximumEffort object.

## 1.2 Class Diagram

Figure 1: Class Diagram of Part 1



## 1.3 Tests

Figure 2: Main Method

```
public class Driver {
    public static void main(String[] args) {
        PhoneStore turkeyStore = new TurkeyStore();
        PhoneStore euStore = new EUStore();
        PhoneStore globalStore = new GlobalStore();

        System.out.println("\n-----Production in Turkey-----");
        orderPhone(turkeyStore, type: "MaximumEffort");
        orderPhone(turkeyStore, type: "IflasDeluxe");
        orderPhone(turkeyStore, type: "I-I-Aman-Iflas");

        System.out.println("\n-----Production in EU-----");
        orderPhone(euStore, type: "MaximumEffort");
        orderPhone(euStore, type: "IflasDeluxe");
        orderPhone(euStore, type: "I-I-Aman-Iflas");

        System.out.println("\n-----Production in Global-----");
        orderPhone(globalStore, type: "MaximumEffort");
        orderPhone(globalStore, type: "IflasDeluxe");
        orderPhone(globalStore, type: "I-I-Aman-Iflas");
    }

    private static void orderPhone(PhoneStore mdl, String type){
        Smartphone model = mdl.orderSmartphone(type);
        System.out.println(model.getName() + " was produced.\n");
    }
}
```

Figure 3: Output<sub>1</sub>

```

-----Production in Turkey-----
Attaching CPU & Ram (2.8GHz, 8GB, 8 cores)
Attaching display (5.5 inches, 32 bit)
Attaching battery (27h, 3600mAh, Lithium-Boron)
Attaching storage (MicroSD support, 64GB, Max 128 GB)
Attaching camera (12Mp front, 8Mp rear, Opt. zoom x4)
Enclosing the phone case (151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 2m)
MaximumEffort was produced.

Attaching CPU & Ram (2.2GHz, 6GB, 8 cores)
Attaching display (5.3 inches, 32 bit)
Attaching battery (20h, 2800mAh, Lithium-Boron)
Attaching storage (MicroSD support, 32GB, Max 128 GB)
Attaching camera (12Mp front, 5Mp rear, Opt. zoom x4)
Enclosing the phone case (149x73x7.7 mm waterproof, aluminum, Waterproof up to 2m)
IfilasDeluxe was produced.

Attaching CPU & Ram (2.2GHz, 4GB, 8 cores)
Attaching display (4.5 inches, 32 bit)
Attaching battery (16h, 2000mAh, Lithium-Boron)
Attaching storage (MicroSD support, 16GB, Max 128 GB)
Attaching camera (8Mp front, 5Mp rear, Opt. zoom x4)
Enclosing the phone case (143x69x7.3 mm waterproof, plastic, Waterproof up to 2m)
IIAmanIfilas was produced.

```

Figure 4: Output<sub>2</sub>

```

-----Production in EU-----
Attaching CPU & Ram (2.8GHz, 8GB, 4 cores)
Attaching display (5.5 inches, 24 bit)
Attaching battery (27h, 3600mAh, Lithium-Ion)
Attaching storage (MicroSD support, 64GB, Max 64 GB)
Attaching camera (12Mp front, 8Mp rear, Opt. zoom x3)
Enclosing the phone case (151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 1m)
MaximumEffort was produced.

Attaching CPU & Ram (2.2GHz, 6GB, 4 cores)
Attaching display (5.3 inches, 24 bit)
Attaching battery (20h, 2800mAh, Lithium-Ion)
Attaching storage (MicroSD support, 32GB, Max 64 GB)
Attaching camera (12Mp front, 5Mp rear, Opt. zoom x3)
Enclosing the phone case (149x73x7.7 mm waterproof, aluminum, Waterproof up to 1m)
IfilasDeluxe was produced.

Attaching CPU & Ram (2.2GHz, 4GB, 4 cores)
Attaching display (4.5 inches, 24 bit)
Attaching battery (16h, 2000mAh, Lithium-Ion)
Attaching storage (MicroSD support, 16GB, Max 64 GB)
Attaching camera (8Mp front, 5Mp rear, Opt. zoom x3)
Enclosing the phone case (143x69x7.3 mm waterproof, plastic, Waterproof up to 1m)
IIAmanIfilas was produced.

```

Figure 5: Output<sub>3</sub>

```

-----Production in Global-----
Attaching CPU & Ram (2.8GHz, 8GB, 2 cores)
Attaching display (5.5 inches, 24 bit)
Attaching battery (27h, 3600mAh, Lithium-Cobalt)
Attaching storage (MicroSD support, 64GB, Max 32 GB)
Attaching camera (12Mp front, 8Mp rear, Opt. zoom x2)
Enclosing the phone case (151x73x7.7 mm dustproof, waterproof, aluminum, Waterproof up to 50cm)
MaximumEffort was produced.

Attaching CPU & Ram (2.2GHz, 6GB, 2 cores)
Attaching display (5.3 inches, 24 bit)
Attaching battery (20h, 2800mAh, Lithium-Cobalt)
Attaching storage (MicroSD support, 32GB, Max 32 GB)
Attaching camera (12Mp front, 5Mp rear, Opt. zoom x2)
Enclosing the phone case (149x73x7.7 mm waterproof, aluminum, Waterproof up to 50cm)
IfLasDeLuxe was produced.

Attaching CPU & Ram (2.2GHz, 4GB, 2 cores)
Attaching display (4.5 inches, 24 bit)
Attaching battery (16h, 2000mAh, Lithium-Cobalt)
Attaching storage (MicroSD support, 16GB, Max 32 GB)
Attaching camera (8Mp front, 5Mp rear, Opt. zoom x2)
Enclosing the phone case (143x69x7.3 mm waterproof, plastic, Waterproof up to 50cm)
IIAmanIfLas was produced.

```

## 2. Part 2

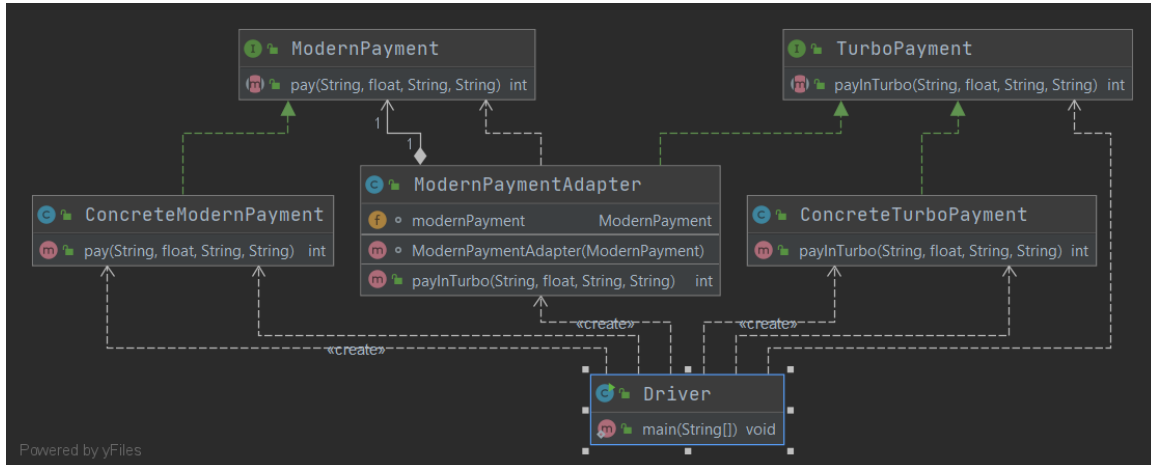
### 2.1 Design Explanation

In this part, there are two interfaces: ModernPayment and TurboPayment. TurboPayment is extensively an old binary library from the 1990s and we cannot modify this interface. Although ModernPayment has pay method, TurboPayment has payInTurbo where all the method parameters have the same meaning and role as in ModernPayment. We should be able to continue using all classes that implement the TurboPayment interface with the new ModernPayment interface. I used Adapter Pattern because it 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.

I created two interfaces: ModernPayment and TurboPayment. Also I created concrete class of these interfaces which are ConcreteModernPayment and ConcreteTurboPayment. They implement pay and payInTurbo methods respectively. I created an adapter class ModernPaymentAdapter that implements TurboPayment interface. It has a ModernPayment object, a constructor with ModernPayment object parameter and also payInTurbo method. In payInTurbo method, it goes to ModernPayment's pay with modernPayment object.

## 2.2 Class Diagram

Figure 6: Class Diagram of Part 2



## 2.3 Tests

Figure 7: Main Method

```

public static void main(String[] args) {
    ConcreteModernPayment concreteModernPayment = new ConcreteModernPayment();
    ConcreteTurboPayment concreteTurboPayment = new ConcreteTurboPayment();
    TurboPayment turboPaymentAdapter = new ModernPaymentAdapter(concreteModernPayment);

    System.out.println("-----Modern Payment Behaviour-----");
    concreteModernPayment.pay(cardNo: "0000000000", amount: 1000, destination: "1111111111", installments: "Yes");
    System.out.println();

    System.out.println("-----Turbo Payment Behaviour-----");
    concreteTurboPayment.payInTurbo(turboCardNo: "2222222222", turboAmount: 2000, destinationTurboOfCourse: "3333333333", installmentsButInTurbo: "Yes");
    System.out.println();

    System.out.println("-----Turbo Payment Adapter Behaviour-----");
    turboPaymentAdapter.payInTurbo(turboCardNo: "4444444444", turboAmount: 3000, destinationTurboOfCourse: "5555555555", installmentsButInTurbo: "Yes");
    System.out.println();
}
  
```

Figure 8: Output

```

-----Modern Payment Behaviour-----
You Are in ConcreteModernPayment Class
Card No: 0000000000
Amount: 1000.0
Destination: 1111111111
Installments: Yes

-----Turbo Payment Behaviour-----
You Are in ConcreteTurboPayment Class
Card No: 2222222222
Amount: 2000.0
Destination: 3333333333
Installments: Yes

-----Turbo Payment Adapter Behaviour-----
You Are in ConcreteModernPayment Class
Card No: 4444444444
Amount: 3000.0
Destination: 5555555555
Installments: Yes

```

### 3. Part 3

#### 3.1 Design Explanation

In this part, there are some database operations. 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). We need to execute and undo the operations. I used Command Pattern because should one of the operations fail, all others must be reversed or discarded. The Command Pattern encapsulates a request as an object, thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.

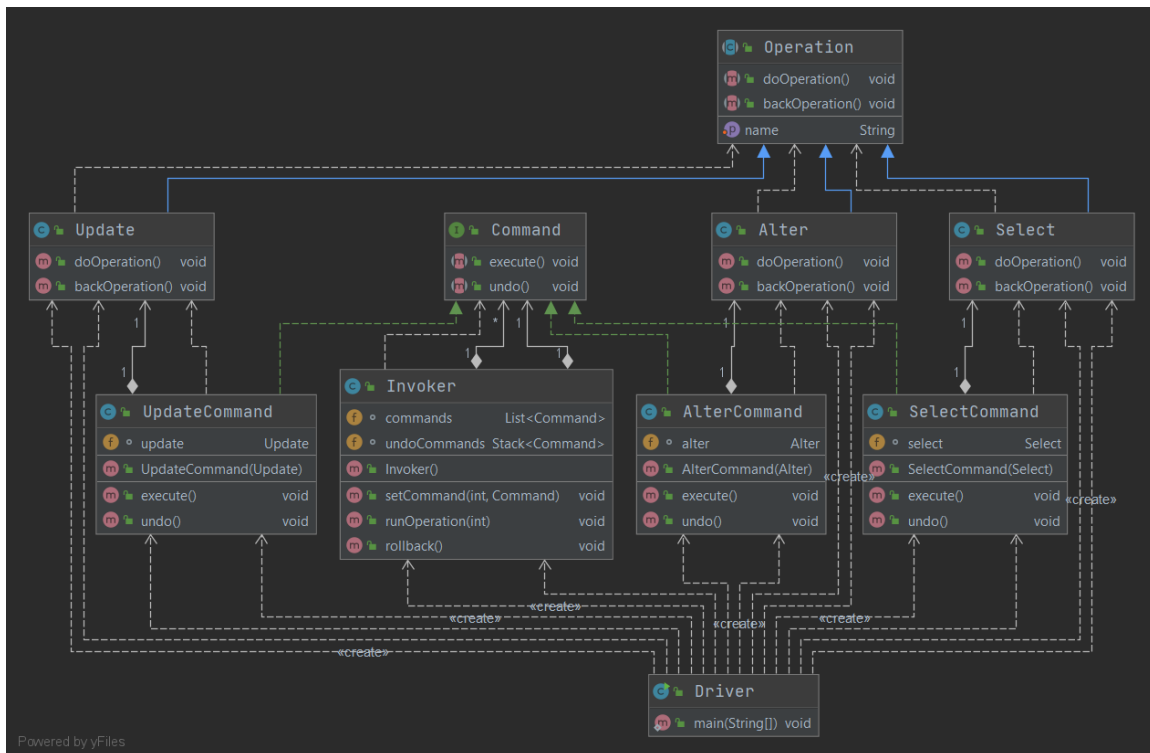
I created Command interface and it has two methods: execute and undo. Execute method executes operations and undo method takes back the operations. Then, I create an abstract class Operation and its subclasses Select, Update and Alter. It has doOperation and backOperation methods. doOperation method does proper operation and backOperation method takes back this operation. Also there are three concrete classes that implement Command interface: SelectCommand, UpdateCommand and AlterCommand. They has Select object, Update object and Alter object respectively. With this objects, they execute operations with proper doOperation method and undo operations with proper backOperation method. I created Invoker class. It has commands list and a stack to keep executed

operations. setCommand method adds operation to list that comes as parameter, runOperation method executes operations that invoked, and rollback method discard all methods if one of the operations fail.

In Invoker class, when executing the operations, if one of the operations fail it throws an exception. Then, in catch block rollback method is invoked and all executed methods in stack are discharged. To simulate error occurrences, I generated a random number in runOperation method. If the mode of this number with two is 1, it means an error occurred.

### 3.2 Class Diagram

Figure 9: Class Diagram of Part 3



### 3.3 Tests

Figure 10: Main Method

```
public static void main(String[] args) {
    Invoker invoker= new Invoker();
    Select select = new Select();
    Update update = new Update();
    Alter alter = new Alter();

    select.setName("FLOWER");
    update.setName("FLOWER");
    alter.setName("FLOWER");

    //create operations
    SelectCommand selectCommand = new SelectCommand(select);
    UpdateCommand updateCommand = new UpdateCommand(update);
    AlterCommand alterCommand = new AlterCommand(alter);

    //add operations to invoker
    invoker.setCommand( slot: 0, selectCommand);
    invoker.setCommand( slot: 1, updateCommand);
    invoker.setCommand( slot: 2, alterCommand);

    //TRANSACTION
    try{
        invoker.runOperation( slot: 0);
        invoker.runOperation( slot: 1);
        invoker.runOperation( slot: 2);
        System.out.println("Transaction completed successfully.");
    } catch (Exception e) {
        System.err.println("Transaction failed.");
        invoker.rollback();
    }
}
```

Figure 11: Output when transaction completed successfully

```
Select operation is done in table FLOWER
Update operation is done in table FLOWER
Alter operation is done in table FLOWER
Transaction completed successfully.
```



Figure 12: Output when transaction was not completed successfully

```

Select operation is done in table FLOWER
Update operation is done in table FLOWER
Transaction was not completed successfully.
Update operation is taken back in table FLOWER
Select operation is taken back in table FLOWER
Transaction failed.

```

## 4. Part 4

### 4.1 Design Explanation

I used Template Method design pattern in this part because both the 1D Discrete Fourier Transform and the 1D Discrete Cosine Transform do the same thing and follow same steps. They read N tab separated numbers from a file provided as a command line argument, transform the numbers into N outputs, write the N outputs to a new file and only in the case of DFT, the time of execution printed on screen if the user wants.

I created an abstract class Transform. It has some methods and lists. numbers list keeps read numbers from file, realPart list keeps real part of transformed numbers, and imagPart list keeps imaginary part of transformed numbers. It also has a startTime data field for DFT. Transform has procedure method. This method can not change by subclasses because it is final. procedure method invokes some methods which are readFile, transform, writeFile and printScreen. readFile method reads the input file and adds numbers to numbers list, transform method transforms numbers with proper method, writeFile method writes transformed numbers to output file, and printScreen method returns true if user enters 'y'. Also it has printTime method prints execution time on the screen. transform method and printTime method are abstract so each transform types implement own method properly. There are two subclasses of Transform class: DiscreteCosineTransform and DiscreteFourierTransform. They implement transform method and transform numbers with DCT and DFT respectively.

In case of DFT, it gets user input from console as 'y' or 'n' with getUserInput. This method returns the user answer. Then, printScreen method return true if user input is 'y'.

In my design, both method read same input file. DCT method work with only real numbers so the input file should includes only real numbers.

Figure 13: DFT formula

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

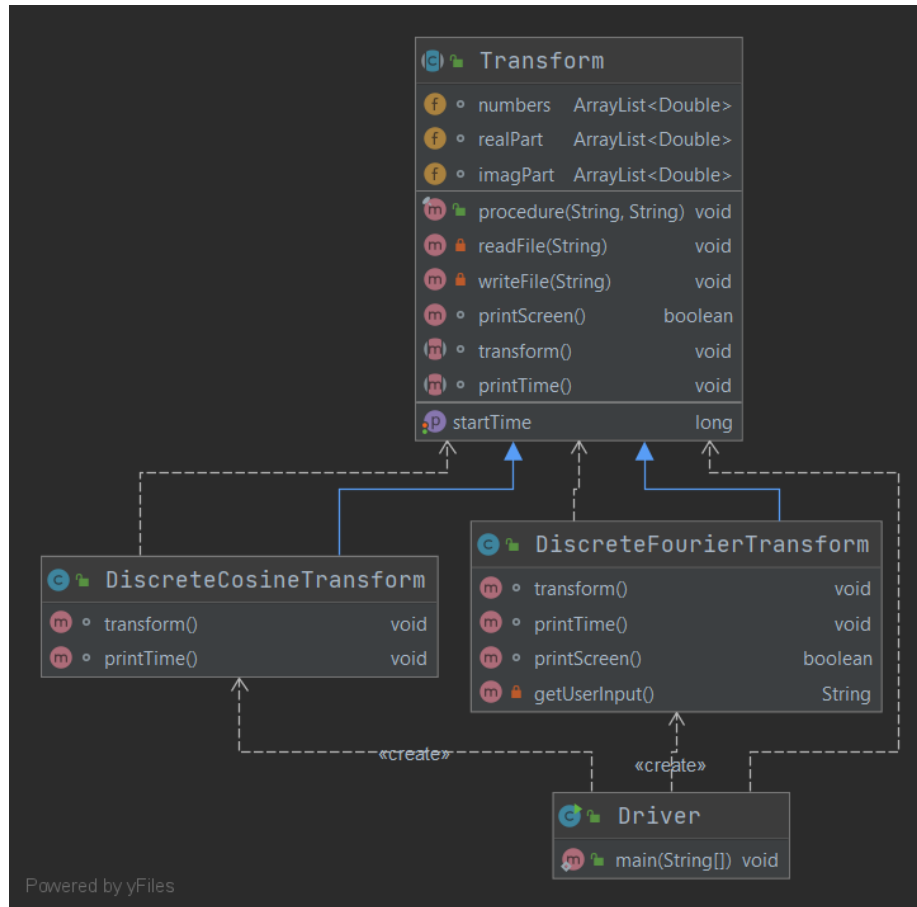
Figure 14: DCT formula

**DCT-II** [edit]

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

## 4.2 Class Diagram

Figure 15: Class Diagram of Part 4



### 4.3 Tests

Figure 16: Main Method

```
public static void main(String[] args) throws IOException {
    System.out.println("\n-----Discrete Fourier Transform-----");
    Transform dft = new DiscreteFourierTransform();
    dft.procedure(args[0], output: "output_dft.txt");

    System.out.println("\n-----Discrete Cosine Transform-----");
    Transform dct = new DiscreteCosineTransform();
    dct.procedure(args[0], output: "output_dct.txt");
}
```

Figure 17: Output-1

```
-----Discrete Fourier Transform-----
Do you want print the time of execution on screen (y/n)? y
Execution Time: 3.484sec

-----Discrete Cosine Transform-----
```

Figure 18: input.txt

input.txt - Not Defteri					
Dosya	Düzen	Biçim	Görünüm	Yardım	
3	8.7	6	-5.4	12	-3

Figure 19: output\_dft.txt

```

21,300 +0,000j
2,250 -4,936j
-14,250 -15,329j
20,700 +0,000j
-14,250 +15,329j
2,250 +4,936j
    
```

Figure 20: output\_dct.txt

```

21,300
6,413
-0,520
-1,485
-20,400
14,898
    
```

Figure 21: Output-2

```

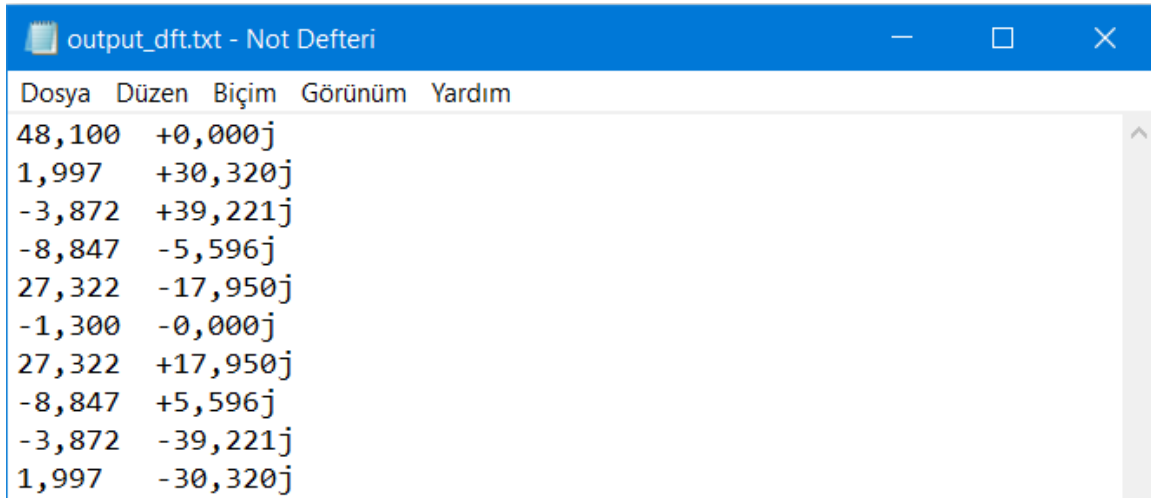
-----Discrete Fourier Transform-----
Do you want print the time of execution on screen (y/n)? n
-----Discrete Cosine Transform-----
    
```

Figure 22: input.txt

```

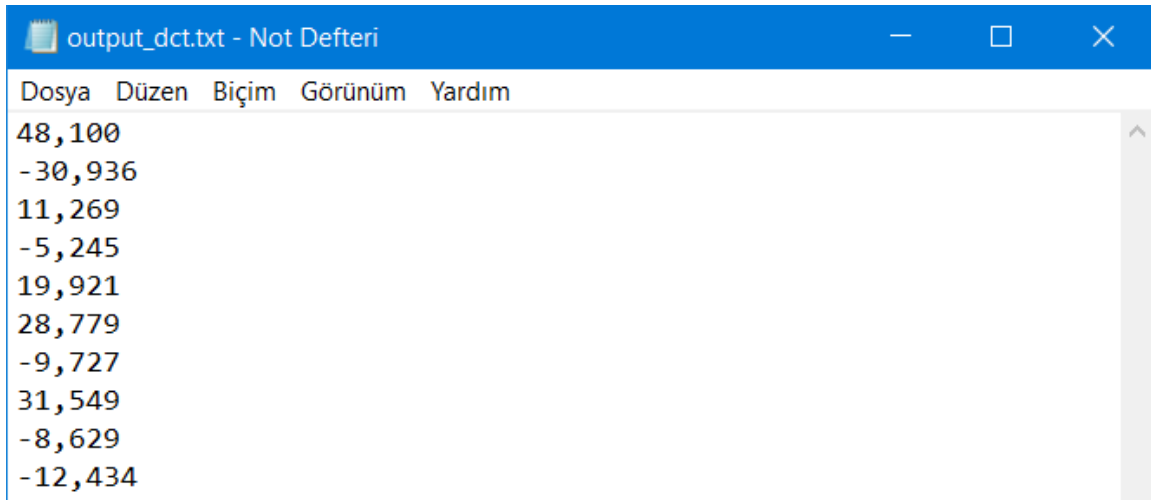
8 -6.7 -5.9 7.3 2 11 -3.7 4.1 23 9
    
```

Figure 23: output\_dft.txt



Dosya	Düzen	Biçim	Görünüm	Yardım
48,100	+0,000j			
1,997	+30,320j			
-3,872	+39,221j			
-8,847	-5,596j			
27,322	-17,950j			
-1,300	-0,000j			
27,322	+17,950j			
-8,847	+5,596j			
-3,872	-39,221j			
1,997	-30,320j			

Figure 24: output\_dct.txt



Dosya	Düzen	Biçim	Görünüm	Yardım
48,100				
-30,936				
11,269				
-5,245				
19,921				
28,779				
-9,727				
31,549				
-8,629				
-12,434				