Gebze Technical University
Computer Engineering

Object Oriented Analysis and Design
CSE443 – 2018 Autumn

HOMEWORK 3 REPORT

Yunus ÇEVİK
141044080

Course Teacher: Erchan Aptoula

Course Assistant: Muhammet Ali Dede

Answer – 1

1 METHOD

Kredi kartı ödemeleri için “Payment” projesini tasarlarken Adapter Tasarım Desenini kullanarak tasarladım. Bu tasarım deseni, structural tasarım desenlerinden biridir. Bu tasarım deseni, birbiriyle ilişkili olmayan interface'lerin birlikte çalışmasını sağlar. Bu işlemi ise, bir sınıfın interface'ini diğer bir interface'e dönüştürerek yapar.

Adapter tasarım deseni ismini gerçek hayattaki adaptörlerden almıştır. Örneğin, telefon şarz cihazı bir adaptördür. 220V'luk gerilimi 5V'a dönüştürür. Mobil şarz cihazı, mobil şarz soketi ile duvar soketi arasında bir adaptör görevi görür.

Var olan sistemin interface'i, target interface olarak adlandırılır. Bu interface'i implement edecek bir Adapter sınıfı yaratılır. Adapter sınıfında, Adaptee interface türünden bir sınıf değişkeni bulunur. Son olarak client sınıfı Adapter sınıfı nesnesi ve Adaptee nesnesini yaratır.

Adapter Tasarım Deseni’nin Faydaları:

1. Birbiriyle ilişkili olmayan interface'lerin birlikte çalışmasını sağlar.
2. Kodların yeniden yazılması engeller.
3. Var olan modül(ler) değiştirilmeden sisteme yeni modüller eklenebilir.

2 If the company had wanted “Bi-Directional Adapter” from us.

Şayet şirket ModernPaymet yapısına geçtikten sonra yeri geldiğinde de TurboPayment yapısına geri dönmek isteseydi. Bu durumda ModernPayment olan ödeme tipi eski olan TurboPayment’ a geri dönmek isteyecekti. Bunu yapmamızda mümkündür bunun için Adapter tasarım desenini çift yönlü yapmamız yeterli olacaktır.

Payment Adapter from TurboPayment to ModernPayment

Adapter (TurboPayment ➔ ModernPayment)

Payment Adapter from ModernPayment to TurboPayment

Adapter (TurboPayment ➔ ModernPayment)

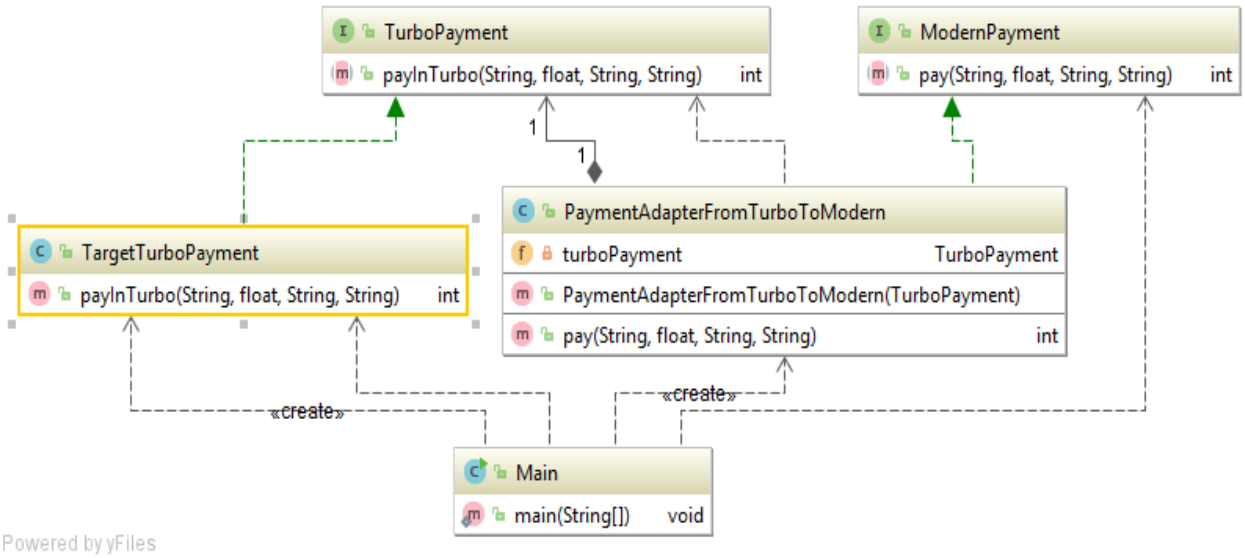
Aşağıda belirtilen Java Class’ ını yazarak TurboPayment’ ı adaptör ile ModernPayment’ a dönüştürdüğümüz yapıyı ModernPayment’ tan başka bir adaptör ile TurboPayment’ a dönüştürmüş oluruz.

```java
/* ModernPayment' tan TurboPayment' a dönüşüm yapmak için elimizde
ModernPayment'tan implements edilmiş TargetModernPaymentımızın olması gerekmek-
tedir. Çünkü dönüşüm yapacak adaptöre bu sınıfın objesi verilmektedir. */
public class TargetModernPayment implements ModernPayment {

    @Override
    public int pay(String cardNo, float amount, String destination, String
installments) {
        return (int) amount;
    }
}

/* ModernPayment' tan TurboPayment' a dönüşüm yapmak için oluşturulmuş
TurboPayment interface yapısından implements edilmiş adatör sınıfı. */
public class PaymentAdapterFromModernToTurbo implements TurboPayment {

    private  ModernPayment modernPayment;

    public PaymentAdapterFromModernToTurbo(ModernPayment modernPayment){
        this.modernPayment = modernPayment;
    }

    @Override
    public int payInTurbo(String turboCardNo, float turboAmount,
                          String destinationTurboOfCourse, String
installmentsButInTurbo) {
        System.out.println("Adapter (TurboPayment <- ModernPayment)");
        System.out.println("---- Turbo Payment ----");
        System.out.println("Turbo Card No :" + turboCardNo + "\nDestination Tur-
bo Of Course :" + destinationTurboOfCourse + "\nInstallments But In Turbo :" +
installmentsButInTurbo);
        return modernPayment.pay(turboCardNo, turboAmount,
destinationTurboOfCourse, installmentsButInTurbo);
    }
}
```

Yukarıda belirtilen sınıfları oluşturarak çift yönlü dönüşüm işlemi yapmamız mümkündür.

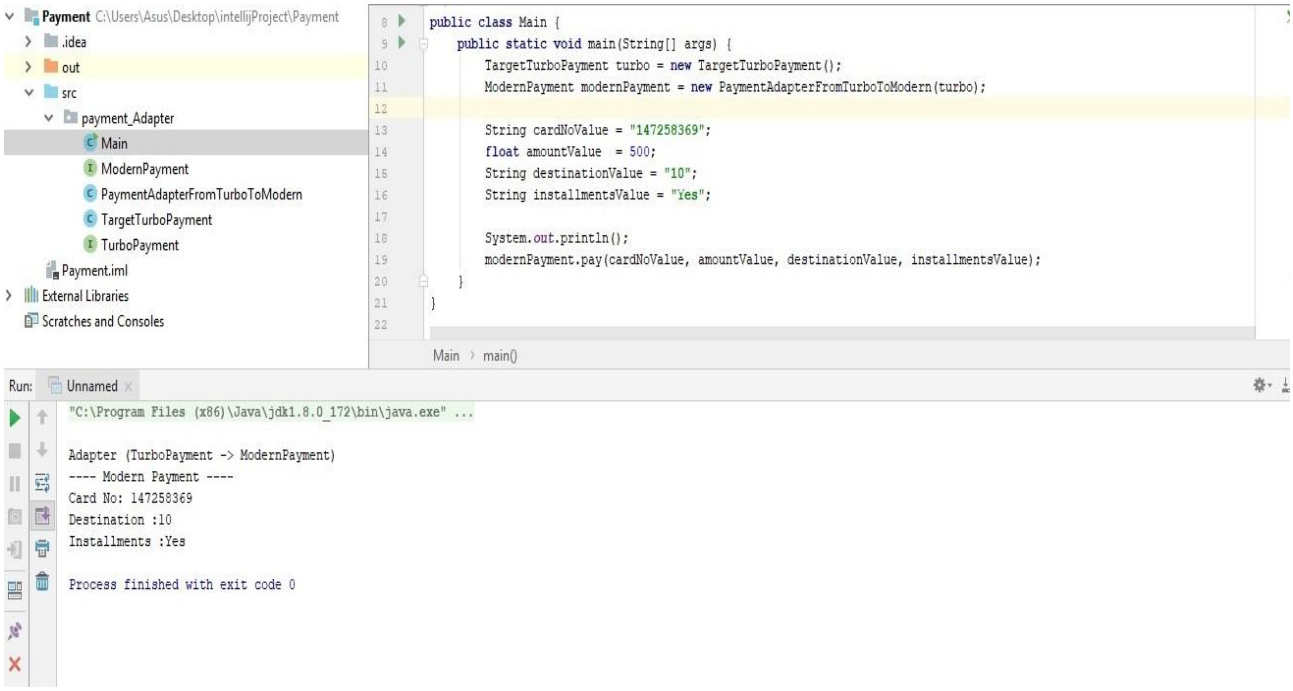
Result kısmında ek olarak çift yönlü dönüşümün de sonucu da gösterilecektir.

Class Diagrams



3 Result

Şireketin belirlemiş olduğu eskiden kullanılan “TurboPayment” yapısından “ModernPayment” yapısına bir adaptör yardımı ile dönüşümünün sonucu aşağıda yer almaktadır.



Ayrıca yukarıda belirttiğim “Bi – Directional Adapter (Çift Yönlü Adaptör)” yapısını şirket bizden istemiş olsaydı aşağıdaki gibi olacaktı.

```
"C:\Program Files (x86)\Java\jdk1.8.0_172\bin\java.exe" ...

Adapter (TurboPayment -> ModernPayment)
---- Modern Payment ----
Card No: 147258369
Destination :10
Installments :Yes

------------------------------

Adapter (TurboPayment <- ModernPayment)
---- Turbo Payment ----
Turbo Card No :147258369
Destination Turbo Of Course :10
Installments But In Turbo :Yes

Process finished with exit code 0
```

```
public class Main {
    public static void main(String[] args) {
        TargetTurboPayment turbo = new TargetTurboPayment();
        ModernPayment modernPayment = new PaymentAdapterFromTurboToModern(turbo);

        TargetModernPayment modern = new TargetModernPayment();
        TurboPayment turboPayment = new PaymentAdapterFromModernToTurbo(modern);

        String cardNoValue = "147258369";
        float amountValue  = 500;
        String destinationValue = "10";
        String installmentsValue = "Yes";

        System.out.println();
        modernPayment.pay(cardNoValue, amountValue, destinationValue, installmentsValue);
        System.out.println("\n------------------------------\n");
        turboPayment.payInTurbo(cardNoValue, amountValue, destinationValue, installmentsValue);

    }
}
```

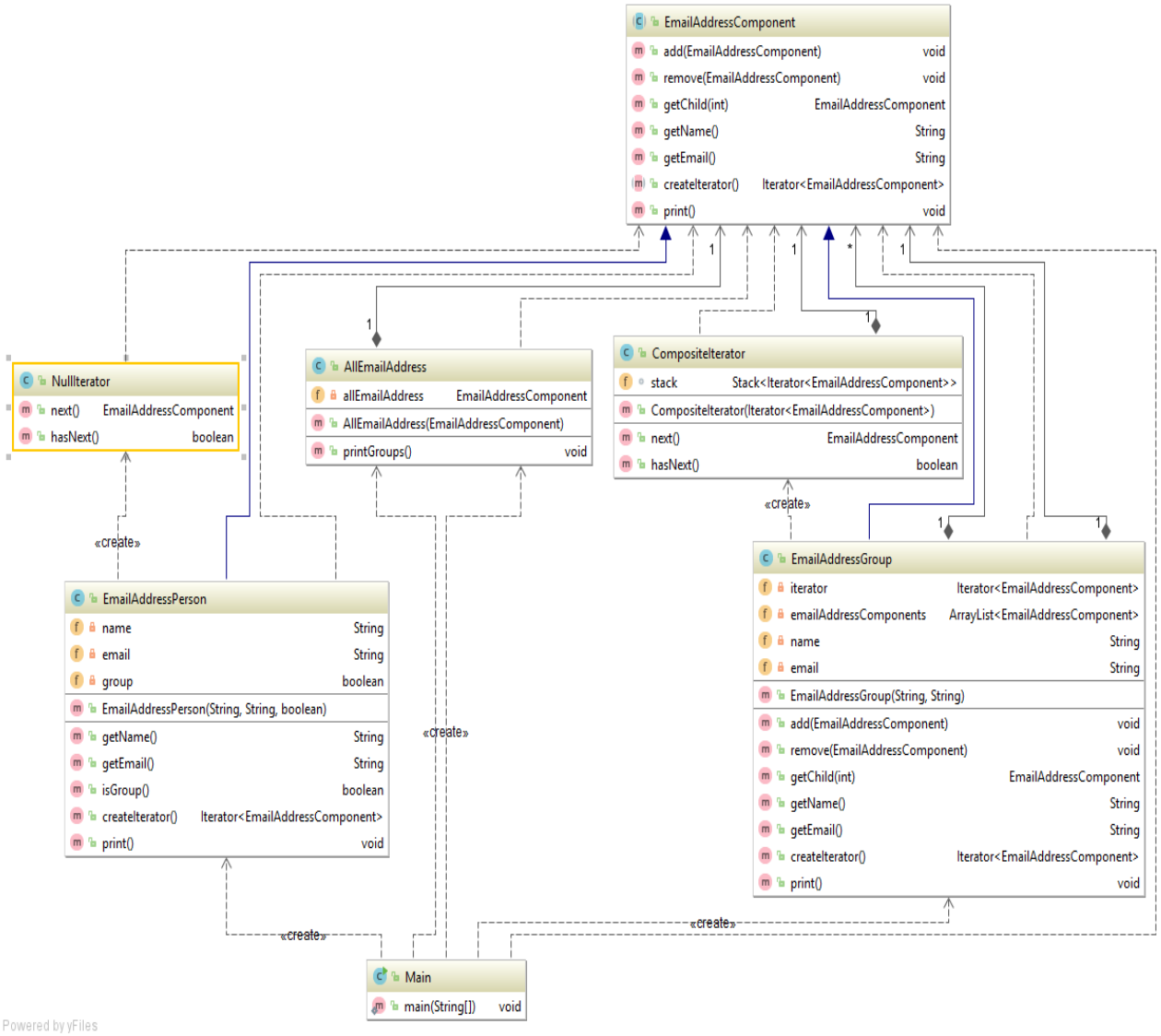
"Bi - Directional Adapter" istenmiş olsaydı main içerisinde şu şekilde kullanmamız yeterli olacaktı.

Answer – 2

1 METHOD

Email Address Book projesini tasarlarken Composite Tasarım Desenini kullanarak tasarladım. Composite Tasarım Deseni, structual grubuna ait; ağaç yapısında ki nesne kalıplarının hiyerarşik olarak iç içe kullanılmasını düzenlemektir. Amacı nesneleri ağaç yapısına göre düzenleyerek, ağaç yapısındaki alt üst ilişkisini kurmaktır. Bu tasarım desenine göre, ağaç yapısındaki üst ve alt nesneler aynı arayüz sınıfından türeyerek, birbirlerine benzerler. Yani istemci, yaprak nesneye de üst nesneye de aynı şekilde davranır. Böylece hiyerarşiyi ifade etmek ve hiyerarşi üzerinde işlem yapmak kolaylaşır. Hiyerarşiye yeni nesneler eklemek de kolay hale gelir.

Class Diagrams

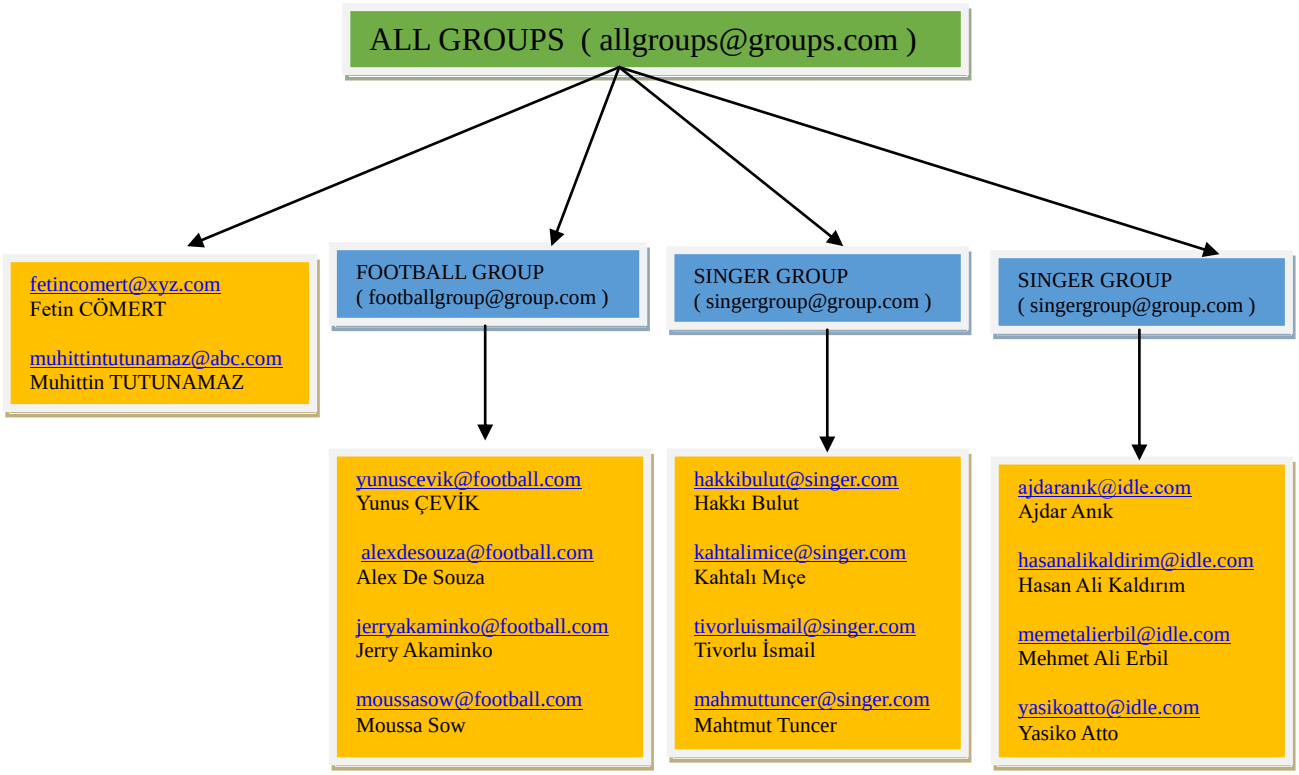


2 Result

Composite Tasarım Deseni ile yapmış olduğumuz bu projenin ekran çıktısı aşağıdaki gibidir. “ALL GROUPS” yazan kısım ağaç yapısının “Root” u olarak görünmektedir. Diğer gruplar ve gruba dahil olmayanlar ise “Root” un çocukları. “FOOTBALL”, “SINGER” ve “IDLE” grupları ise ağacın “Leaf” leridir.

```
"C:\Program Files (x86)\Java\jdk1.8.0_172\bin\java.exe" ...

ALL GROUPS  ( allgroups@groups.com )
---------------------------------------

(Group: ✗)    fetincomert@xyz.com  -> Fetin CÖMERT

FOOTBALL GROUP  ( footballgroup@group.com )
---------------------------------------
(Group: ✓)    yunuscevik@football.com  -> Yunus ÇEVİK
(Group: ✓)    alexdesouza@football.com  -> Alex De Souza
(Group: ✓)    jerryakaminko@football.com  -> Jerry Akaminko
(Group: ✓)    moussasow@football.com  -> Moussa Sow

SINGER GROUP  ( singergroup@group.com )
---------------------------------------
(Group: ✓)    hakkibulut@singer.com  -> Hakkı Bulut
(Group: ✓)    kahtalimice@singer.com  -> Kahtalı Mıçe
(Group: ✓)    tivorluismail@singer.com  -> Tivorlu İsmail
(Group: ✓)    mahmuttuncer@singer.com  -> Mahtmut Tuncer

IDLE GROUP  ( idlegroup@group.com )
---------------------------------------
(Group: ✓)    ajdaranık@idle.com  -> Ajdar Anık
(Group: ✓)    hasanalikaldirim@idle.com  -> Hasan Ali Kaldırım
(Group: ✓)    memetalierbil@idle.com  -> Mehmet Ali Erbil
(Group: ✓)    yasikoatto@idle.com  -> Yasiko Atto

(Group: ✗)    muhittintutunamaz@abc.com  -> Muhittin TUTUNAMAZ

Process finished with exit code 0
```

Yukarda gösterilen ekran çıktısının ağaç yapısında çizimi aşağıdaki gibidir.

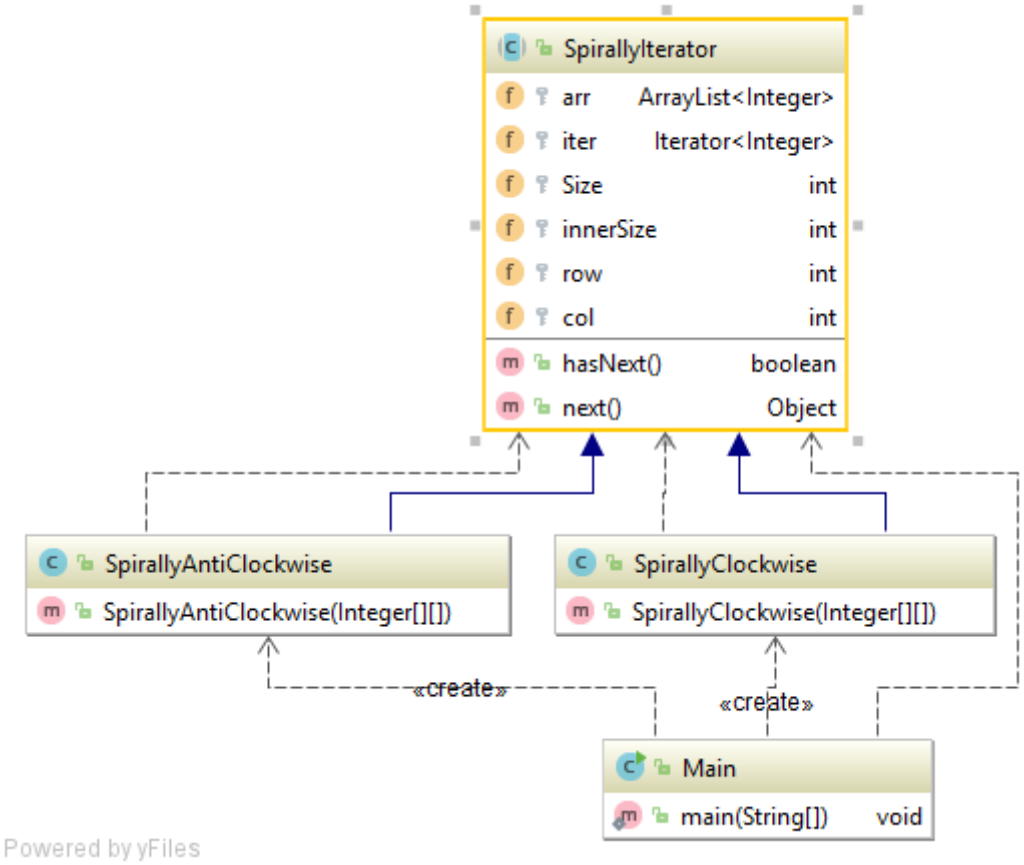


Answer – 3

1 METHOD

Göktürk uydusunun iki boyutlu tamsayı verilerini dışa aktarmak için iki çeşit yineleme algoritması yazabiliriz. Bunlardan biri “Saat Yönünde” diğeri ise “Saat Yönünün Tersi” şeklinde yinelemeli olarak dışa aktaran algoritmalardır. Bu algoritmaları yazarken iki boyutlu tamsayı dizisinin en üst sol tarafında bulunan (data[0][0]] kısmından başlayarak yineleme yapmamız gerekmektedir.

Class Diagrams



2 Result

Göktürk uydusunun iki boyutlu tamsayı verilerinin iki türlü yineleme algoritması ile dışa aktarılma şekili aşağıdaki gibidir.

```
"C:\Program Files (x86)\Java\jdk1.8.0_172\bin\java.exe" ...
Data :
1   2   3   4
5   6   7   8
9   10  11  12
13  14  15  16

The first iterator: 1 2 3 4 8 12 16 15 14 13 9 5 6 7 11 10
The second iterator: 1 5 9 13 14 15 16 12 8 4 3 2 6 10 11 7
Process finished with exit code 0
```

Not:

Saat Yönünde Yineleme ➔ “The first iterator”

Saat Yönünün Tesi Yineleme ➔ “The second iterator”

Answer – 4

1 METHOD

Transforms projesini tasarlarken Template Metot Tasarım Desenini kullanarak tasarladım. Template Metot Tasarım Deseni, davranışsal (behavioral) tasarım desenlerinden biridir. Bir algoritmanın adımlarını tanımlamayı sağlar ve alt sınıfların bir veya daha fazla adımların implementasyonunu yapmasını olanak tanır. Örneğin, bir araba yapımını düşünelim. Sırası ile arabanın iskeleti oluşturulur, motoru takılır, tekerlikleri takılır daha sonra camları takılır. Burada dikkat edilecek olan husus, adımların sırayla yapılmasının zorunlu olmasıdır. Template metot tasarımı ile bu adımların sırasıyla yapılması zorunlu hale getirilmiştir. Benzer adımlardan oluşan sınıfları tasarlarken, aynı kodları her sınıfta tekrar tekrar kullanmayı engellemek için Template Metot Tasarım Deseni kullanılır. Örneğin, kahve ve çay hazırlamakla ilgili sınıflar ortak özelliklere sahiptir. Her ikisinde de sırasıyla şu işlemler yapılır: Su kaynatma, demleme, bardağa dökme ve isteğe göre çeşni ekleme.

Türü abstract olan bir sınıf yaratılır. Bu sınıf, içerisinde aynı adımlara sahip olan işlemleri temsil eder. Her işlem için bir metod eklenir. İşlemlerin sırasıyla yapılmasını zorunlu yapabilmek için ise template metodu (Projedeki Translate metodu gibi) eklenir. Template metodu işlem adımlarının zorunlu yapılması için final türüne sahip olur. Bu sayede alt sınıflar template metoduoverride edemezler. Template metod içerisinde, işlem adımları için oluşturulmuş metodlar sırasıyla çağrılır.

```
public final void translate(String inFileName, String outFileName) throws
IOException {
    ReadFile(inFileName);
    Execution();
    WriteFile(outFileName);
    Hook();
}
```

Hook ?

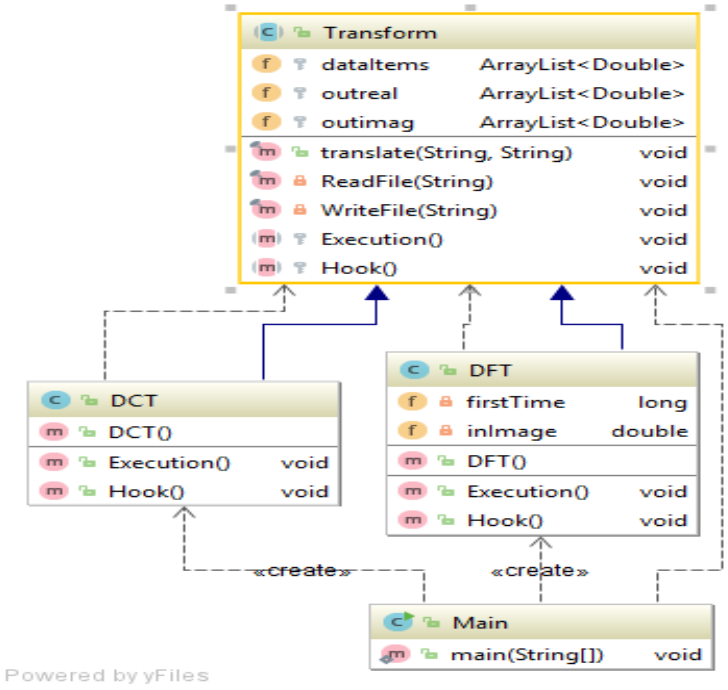
Hook(Kanca), abstract sınıf içinde tanımlanmış, içi boş veya default implementasyona sahip bir metodtur. Bu metod alt sınıflar tarafından override edilebilir.

Hook kullanmanın faydaları şunlardır:

1. Algoritmanın işleyişini alt sınıfın değiştirebilmesi sağlanır.
2. Algoritmanın opsiyonel parçasının olması sağlanır.
3. Alt sınıfa, gerçekleşmiş veya gerçekleşek bazı adımlara tepki verme şansı verir.

Sonuç olarak Template Metot Tasarım Deseni bizi kod tekrarından kurtarır ve “**final”** keyword’ u sayesinde standart bir tasarım sağlar.

Class Diagrams



2 FORMULAS

DFT

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N} kn}$$
$$= \sum_{n=0}^{N-1} x_n \cdot [\cos(2\pi kn/N) - i \cdot \sin(2\pi kn/N)],$$

DCT-II

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

3 Result

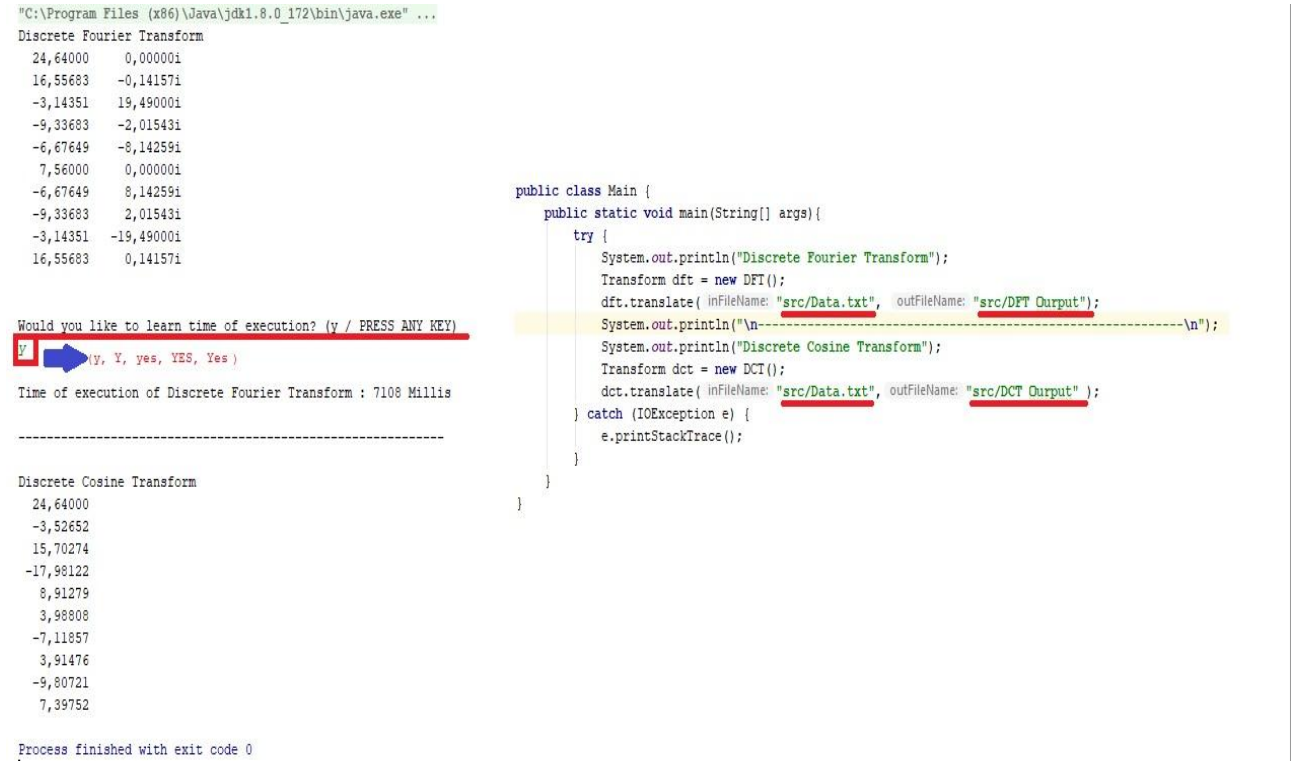
Template Metot Tasarım Deseni ile yapmış olduğumuz bu projenin sonuç kısımda "Data.txt" dosyası içerisinde bulunan "Real Sayılara" DFT (Discrete Fourier Transform) ve DCT (Discrete Cosine Transform) algoritmalarını uygulayarak sonuçları "DFT_Output.txt" ve "DCT_Output.txt" dosyalarını yazmış bulumaktayım. Aşağıda belirtilen çıktılar için

Input değerler olarak "Real Sayılar" ➔ Output Değerler ➔ Comlex Sayılar
Output Değerler ➔ Real Sayılar bulunmuştur.

Not: DFT için formül gereği comlex sayı verilmesi gerekmektedir. Ancak benim "Data.txt" dosyam içerisinde "Real Sayılar" olduğu için Comlex sayının "Real" kısmına dosya içerisindeki sayılar, imaginary kısmına ise her sayı bir Complex sayı olduğu için ve Real

sayıların, imaginary kısımları olmadığı için “SIFIR (0)” kabul edilmektedir.
Aşağıdakı kod bloğu içerisinde yer alan formül de imaginary kısmı “SIFIR (0)” kabul edilerek hesaplamalar yapılmıştır.

```
private double inImag = 0;

int N = dataItems.size();
for (int k = 0; k < N; k++) {
    double sumreal = 0;
    double sumimag = 0;
    for (int t = 0; t < N; t++) {
        double angle = 2 * Math.PI * t * k / N;
        sumreal += dataItems.get(t) * Math.cos(angle) + inImag *
Math.sin(angle);
        sumimag += -dataItems.get(t) * Math.sin(angle) + inImag *
Math.cos(angle);
    }
    outreal.add(sumreal);
    outimag.add(sumimag);
}
```



Data.txt Real Numbers	DFT_Output.txt Real – Imaginary Numbers		DCT_Output.txt Real Numbers
2.7	24,64000	0,00000i	24,64000
3.5	16,55683	-0,14157i	-3,52652
1.8	-3,14351	19,49000i	15,70274
2.9	-9,33683	-2,01543i	-17,98122
4.0	-6,67649	-8,14259i	8,91279
-1.7	7,56000	0,00000i	3,98808
-2.3	-6,67649	8,14259i	-7,11857
-4.36	-9,33683	2,01543i	3,91476
9.9	-3,14351	-19,49000i	-9,80721
8.2	16,55683	0,14157i	7,39752