CSE 443 - OBJECT ORİENTED ANALYSİS AND DESİGN

Homework 3 Rapor

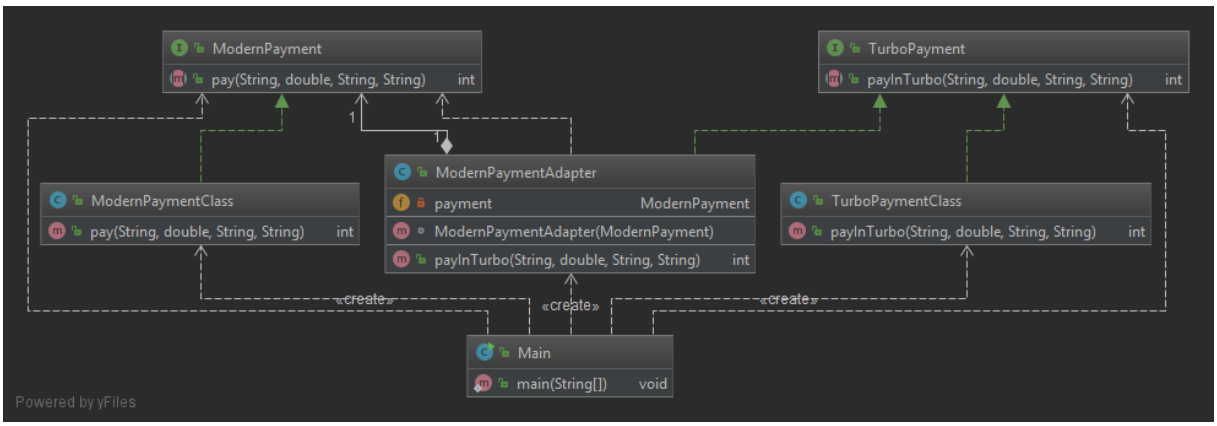
Fürkan YILDIZ
141044031

Part1)

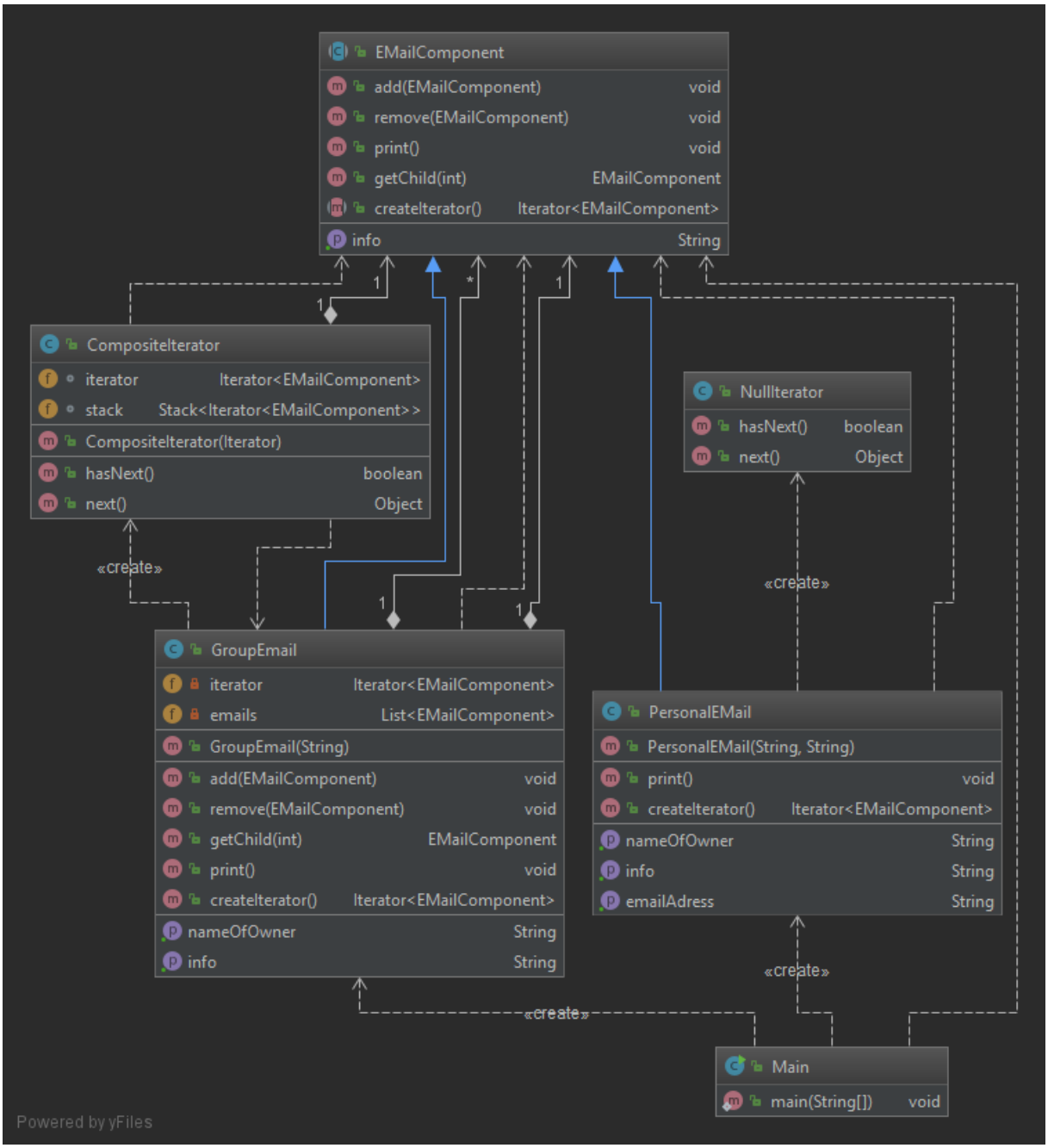
Bu problemde Adapter Design Pattern' inin kullanmamız bize yarar sağlayacak. Böylece yeni ödeme yöntemimiz olan ModernPayment' i, adaptör kullanarak TurboPayment' i eskiden kullandığımız her yerde kullanabileceğiz. Böylece de var olan eski çalıştığını bildiğimiz kodları yenilerinin içinde kullanabileceğiz.

Bunu yapabilmek içinse,

Eski yöntem olan TurboPayment yöntemini kullanmak için, bu interface'yi implement eden bir adaptör oluşturmamız gerekiyor. Bu adaptör' de ise yeni ödeme yöntemimizi constructure' de veriyoruz ki, bu yöntemin fonksiyonlarını, eski yöntemin fonksiyonları içinde çağıralım.



Part2



Concrate classlara göre değilde, interfaceye göre kod yazabilmek için öncelikle “EMailComponent” isminde bir abstract sınıf oluşturuldu

“PersonalEMail” sınıfı tek kişiye ait olan epostaları temsil ediyor. Tek kişilik e postalara add,remove getChild operasyonları yapılamayacağından bu methodllar override edilmedi, abstracttaki gibi exception fırlatılıyor. Yine tek kişiye ait eposta olacağından, bir grup yapısı olmadığından gezilecek bir yapıyı içermiyor, dolayısıyla iterator’e gerek yok. “null” return ederek null kontrolü yapmaktan

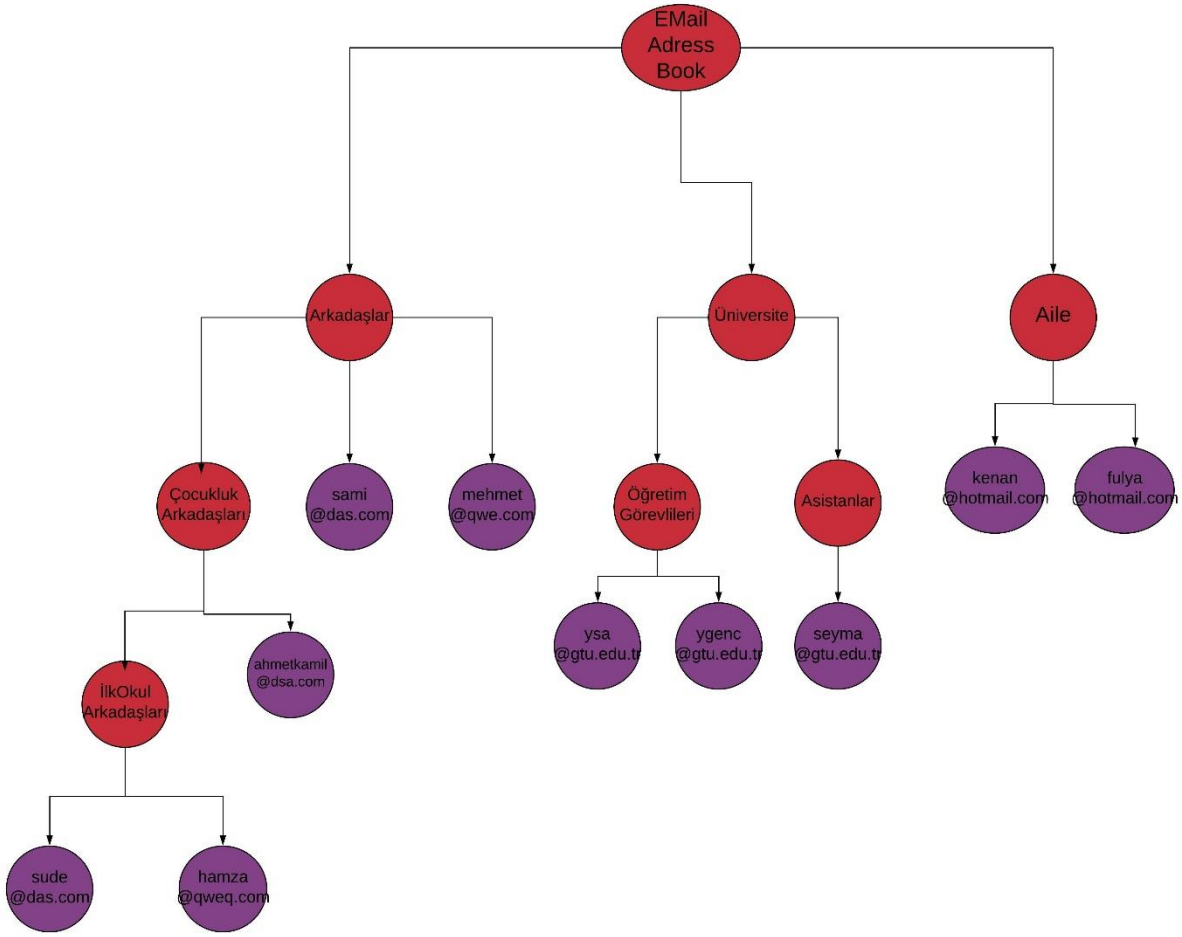
kaçınmak için, iterator methodunda Null Design pattern' i kullanılarak, "NullIterator" sınıfı oluşturuldu, bu sınıf hasNext' i her zaman false dönderiyor, böylece iterate edilemiyor sınıf.

"GroupEmail" sınıfı bir eposta grubunu temsil ediyor. İçerisinde Eposta grubu ya da eposta tutabileceği için "EMailComponent" tipinde bir Liste tutuyor içerisinde. Add, remove ve getChild methodları bu liste üzerinden override edildi. Print methodunda tutulan listedeki tüm elemanlar iteratör ile gezilerek bilgileri ekrana yazılıyor. Her elemanın "print" methodu tekrar çağırıldığı için içerisinde, grupların içerisinde gruplar olsada sırası ile tüm grup ve epostaları yazabiliyor.

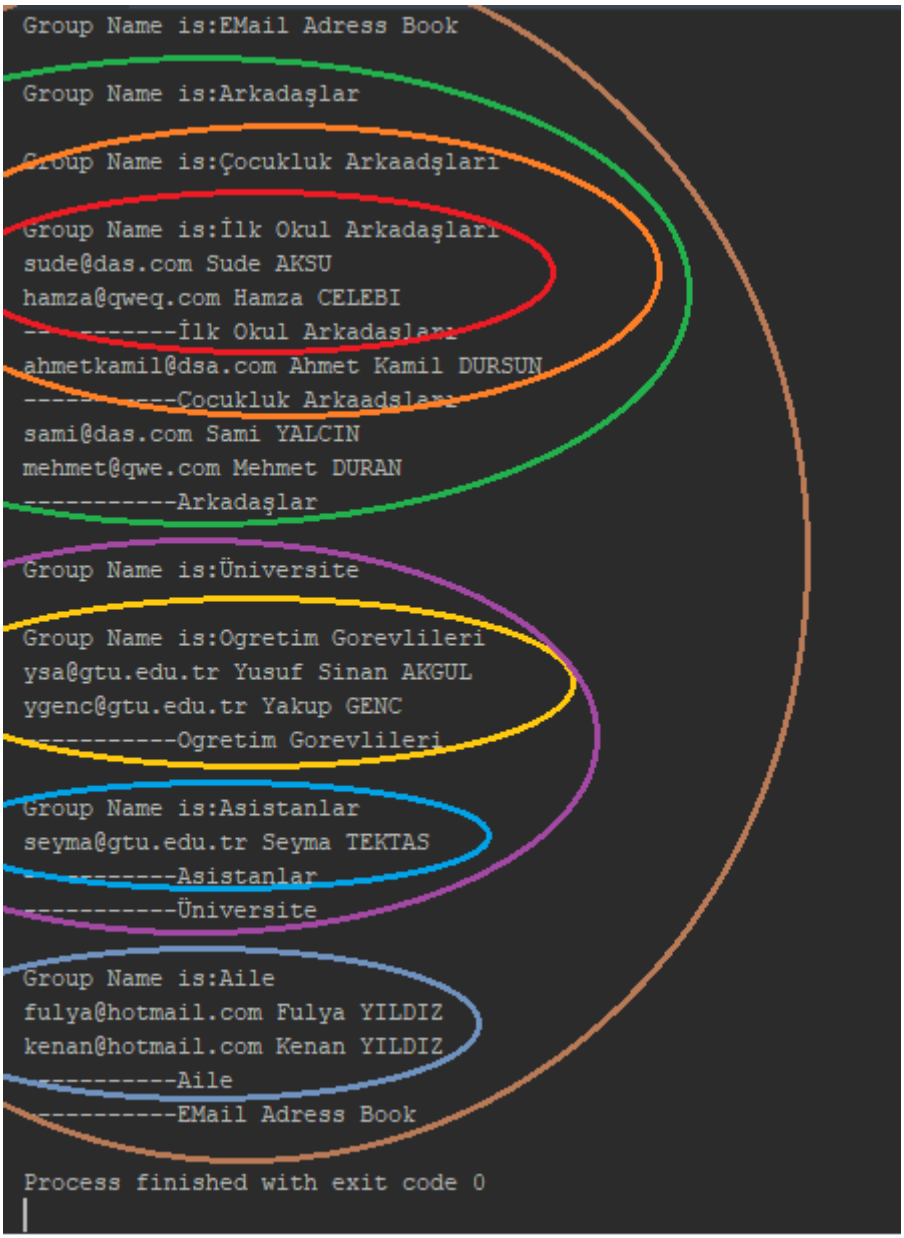
"GroupEmail" sınıfını iterate edebilmek için ise "CompositeIterator" isimli bir sınıf yazıldı. Bu sınıf constructuresinde bir iteratör alıyor, bu iteratör ilk elemanın iteratörü, bu iteratörü, içerisinde tuttuğu stack' e koyarak, stack üzerinden işlem yapıyor. Burada stack tutulmasının sebebi ise grupların içerisinde başka gruplar olabilir bu sebeple recursive bir yapıya ihtiyaç var.

hasNext metodunda, stack' in en üstündeki iteratör, stack'ten çıkartılmadan hasNext' i kontrol ediliyor. hasNext' i varsa, bir grup demektir ve bu grubunda epostaları incelenmelidir dolayısıyla hasNext olduğunda stack'ten iteratör çıkatılmadan, true return return ediliyor. hasNext olmadığında ise eposta sadece tek bir mail dir demektir ve bu mail i iterate edip stack'ten elemanı çıkartmamız gerektiği anlamına gelir, bu sebeple bu durumda stack'ten eleman çıkartılarak hasNext methodu tekrar çağırılır.

Next methodunda ise stack'in en üstündeki eleman çıkartılmadan bu iteratörün next'i çağırılıyor ve return ediliyor ancak return edilmeden önce iteratörün ait olduğu sınıf "GroupEmail" i mi diye kontrol ediliyor ve eğer öyle ise stack'e bu elemanın iteratörü ekleniyor.



Test edilebilmesi için main' de şekildeki gibi bir eposta defteri oluşturdum. Oluşturduğum bu test setinde iç içe birden fazla grubun sorunsuz olarak çalıştığını, grup epostalarıyla, kişisel epostaların beraber çalıştığını göstermek istedim.



Bir grubun epostaları listelenirken, “Group Name:XXX” şeklinde grup listelenmeye başlanıyor, grubun tüm epostaları listelendikten sonra ise grubun bittiğini ifade etmek için “--------XXX” şeklinde (XXX:Grup adı) bir çıktı veriliyor. Çıktının daha iyi anlaşılabilmesi için grupların başlangıç ve bitiş aralıkları, daireler ile gösterildi.

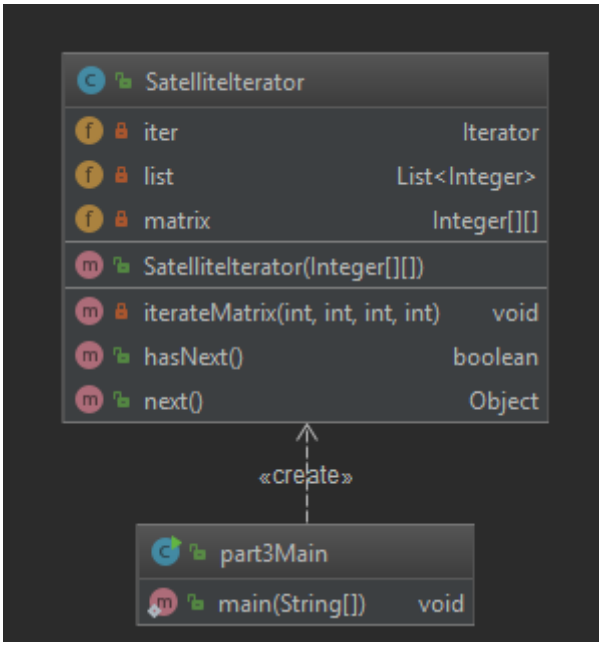
Part 3

Sol üstten, saat yönünde ilerleyen bir iteratör yazdım. 4 tane for döngüsünden oluşuyor ve recursive bir fonksiyon. mxn lik bir matriste, ilk for döngüsüyle sol üstten sağ üstte gitmek için n-1 adım atıyoruz, buradan sağ aşağıya inmek için m-1 adım, sol aşağıya gitmek için n-1 adım ve tekrar sağ üste gitmek için m-1 adım atıyoruz. Böylece tüm çevreyi dolanmış oluyoruz. Bu işlemden sonra başlangıç noktasını (0,0) dan (1,1) e getirdiğimizde ve işlediğimiz kısımları matris ten çıkarttığımızda yine (m-2)x(n-2) lik bir matrix elde etmiş oluyoruz ve bu matrisi de aynı şekilde dönüyoruz(recursive sayesinde). Burada base case ise tek eleman kalması yada hiç eleman kalmaması durumudur.Böylece tüm matrix' i saat yönünde spral olarak dolaşıyoruz ve bir listeye kaydediyoruz.

Bunlara ek olarak, matrisin yatay veya düşey olma durumları var, bu durumlarda 4 for dan sadece 1 ine girmesi gerekiyor, flag tutarak bu işlemide hallediyoruz.

Yazılan bu method "Iterator" u implement eden bir sınıfın içinde yazıldı.

Gezilen tüm elemanlar sırasıyla listeye eklenmişti bu listenin iteratörü, sınıfın iteratörü olarak kullanılıyor. Next ve hasNext methodlarında bu listenin next ve hasNext' i çağırıldı.



Part 4:

Aynı algoritmayı:

1) Read N numbers from a file
2) Transform the numbers into N outputs (DFT or DCT outputs..)
3) Write the N outputs to a new file
4) only in the case of DFT, **sometimes** the user wants additionally the time of execution printed on screen; by default no.

farklı şekilde kullanacağımızdan **Template Method design pattern** 'a ihtiyacımız var.

Bu pattern' a göre …

Discrete Fourier Transform sınıfı için transform methodunu şu formül ile implement ettim.

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

Xn = (a+bi)
0 dan N 'e kadar dönen bir k, for döngüsü. Bu döngünün içerisinde ise yine 0 dan N e kadar dönen bir for döngüsü ve bu ikinci döngünün içerisinde cos sin işlemleri toplamaları yapıldı.

İnput dosyası complex sayılardan oluşur. Her sayı bir satıra yazılır.
Complex sayı = real sayı + imaginary sayi
= double + double i
Şeklindedir. Sayıların ve işaretlerin arasında boşluk olmalıdır. Fourier transform için Vikipedi' deki örnek ve bu örneğin input dosya formatı aşağıda gösterildiği gibidir.

Example

Let $N = 4$ and $\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 - i \\ -i \\ -1 + 2i \end{pmatrix}$

```
1.0 + 0i
2.0 - 1i
0.0 - 1i
-1.0 + 2i
```

Discrete Cosine Transform

DCT-II

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

0 dan N 'e kadar dönen bir k, for döngüsü. Bu döngünün içerisinde ise yine 0 dan N e kadar dönen bir for döngüsü ve bu ikinci döngünün içerisinde cos işlemleri toplamaları yapıldı.

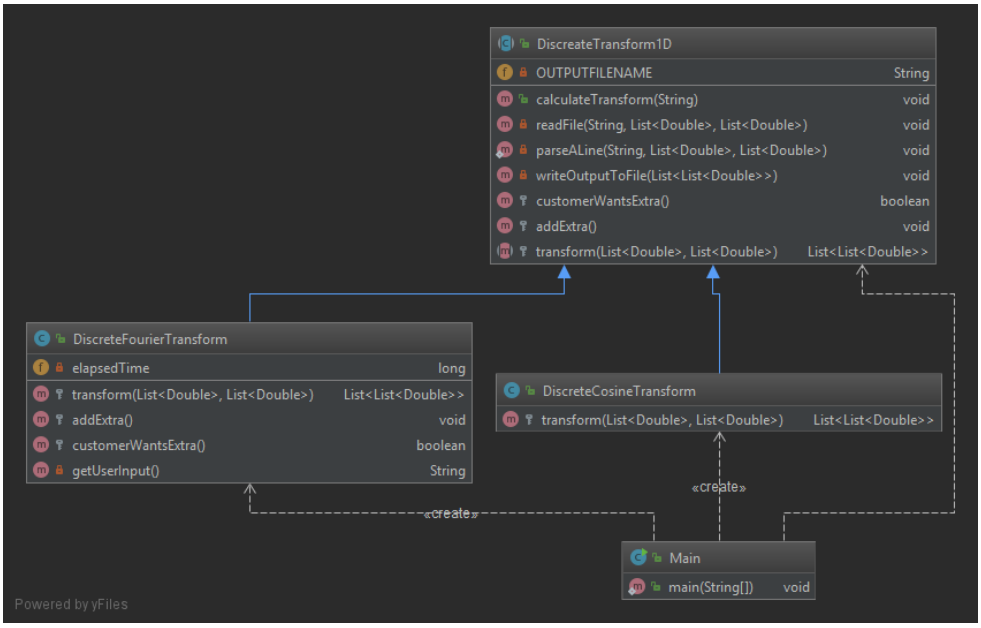
Örnek input dosyası:

```
1.0 + 0i
2.0 + 0i
0.0 + 0i
-1.0 + 0i
```

DCT sadece reel sayılardan oluşan input aldığı için imaginary kısımlar örnekteki gibi 0 olarak girilmeli.

"DiscreateTransform1D" adında bir abstract sınıf oluşturularak bu sınıf için çözüm algoritmasını içeren bir method yazıldı (calculateTransform). Bu algoritmayı gerçeklemek içinse, dosyayı okumak için ve sonuçları dosyaya yazmak için gerekli methodlar bu abstract sınıf içerisinde implement edildi.Transform'u gerçekleştirecek ana method ise abstract method olarak bırakıldı ve subclass ların implement etmesi bekleniyor. DFT 'da transformu gerçekleştirdikten sonra ekstra olarak kullanıcıya soru sormak için algoritmaya (calculateTransform) bir if eklendi bu if'in şartı boolean dönderen customerWantsExtra methodu, bu method abtract sınıfta "false" dönderiyor. Eğer subclass lar herhangi ekstra istiyorlarsa, bu methodu override ederek, ekstralarını if'in içine girdiğinde çağırılacak "addExtra" methodunu override ederek yazabilirler.

DCT için "customerWantsExtra" ve "addExtra" methodlarının override işlemi yapılmadı çünkü herhangibir ekstra istenmiyor. DFT için ise "customerWantsExtra" metodunda kullanıcıya sorarak (transform'un ne kadar sürede gerçekleştiğini görmek isteyip istemediğini) ve "addExtra" methodunda da bu süre ekrana yazdırılarak override edildi.



Mavi oklar "implements"
Beyaz ok ise "uses".