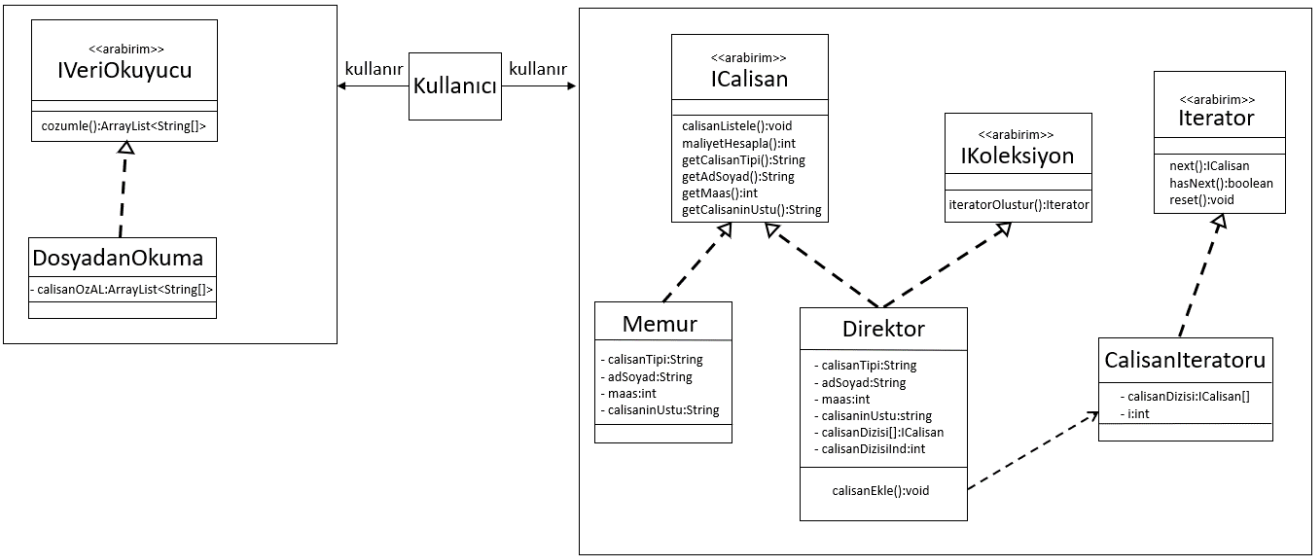
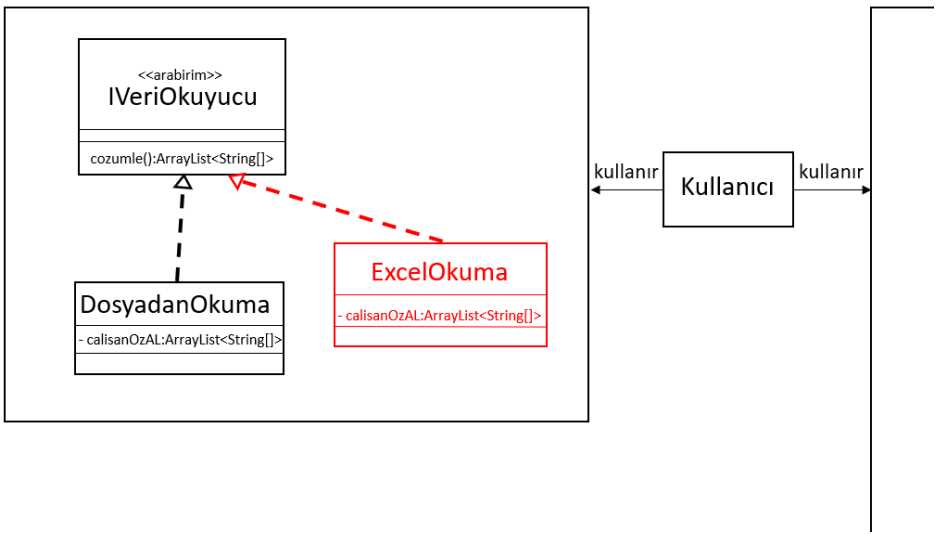
1. UML SINIF DİYAGRAMI

Projede hiyerarşik bir yapı bulunduğu için Composite pattern kullanmak uygun bulundu. Direktör'ün içindeki dizinin dönülmesi için Iterator pattern kullanıldı.



Projede istenen son gereksinim gereğince Strategy pattern kullanıldı. Girdinin ileride farklı bir ortamlardan okunması durumunda geliştirici, ilgili ortamdan okumanın gerçekleştirileceği bir sınıf oluşturup bu sınıfı IveriOkuyucu sınıfına arayüz ilişkisiyle bağlamalıdır.

Ör. Excel ortamından okuma yapmak istenildiğinde UML diyagramı aşağı şekilde değişecektir.



2. SINIFLARIN AÇIKLAMALARI

2.a. Deneme:

Main'in içerisinde bulunduğu sınıftır. Okunan girdiye göre Memur ve Direktör nesneleri oluşturulup calisanAL adlı ArrayList'e eklenir. Bu ArrayList iç içe iki for döngüsüyle dönülerek Direktör nesnelerine astları eklenir. Pdf'de istenen çalışanların bilgileri yazdırılır.

2.b. ICalisan:

Direktör ve Memur nesnelerinin calisanListele() ve maliyetHesapla() metodlarını gerçekleştirmelerini zorunlu tutar. getCalisanTipi(), getAdSoyad(), getMaas() ve getCalisaninUstu() metodlarıysa; Direktör ve Memur sınıflarında ortak bulunmaları sebebiyle bu sınıfta yer almaktadır.

```
public interface ICalisan
{
    void calisanListele();
    int maliyetHesapla();

    String getCalisanTipi();
    String getAdSoyad();
    int getMaas();
    String getCalisaninUstu();
}
```

2.c. Memur:

(String) calisanTipi, (String) adSoyad, (int) maas ve (String) calisaninUstu değişkenleri bulunur. Get'leri mevcuttur. calisanListele() metodu, memurun bilgilerini yazdırır. maliyetHesapla() metodunda maas'in değeri döndürülür.

2.ç. Direktör:

Memur sınıfındaki değişkenlerin üzerine en fazla 100 nesne tutabilen (ICalisan[]) calisanDizisi ve bu dizinin indeksi için int tipinde değişken bulunur. Bu indeks sadece, diziye ICalisan nesnesi eklemek için kullanılır. calisanDizisi'nin boyutu, bu projeye yeterli olacağı düşüncesiyle 100 olarak belirlenmiştir. calisanListele() ve maliyetHesapla() metodlarında dizinin dönülebilmesi için (Iterator) cIterator gerçekleştirilir. (int) tab, çalışan listelenirkenki dallanmayı sağlar.

```
private String calisanTipi;
private String adSoyad;
private int maas;
private String calisaninUstu;
private ICalisan[] calisanDizisi = new ICalisan[100];
private int calisanDizisiInd = 0;

Iterator cIterator = iteratorOlustur();
static int tab = 0;
```

calisanListele() gerçekleştirilirken while(cIterator.hasNext()) ile direktör'ün astları, cIterator.next() kullanılarak çağrılıp calisanListele() metodları kullandırtılır. Eğer ast, direktör ise özyinelemeli durum gerçekleşecektir; değilse konsola bilgiler direkt olarak yazdırılacaktır. tab, hiyerarşik yazdırım için static şeklindedir. Her direktör görüldüğünde yazım bir miktar sağdan gerçekleştirilmek üzere ayarlanacaktır. direktör'ün astlarının yazdırımı bittiğinde tab bir azaltılarak yazım aynı miktarda sola getirilir.

```
public void calisanListele()
{
    cIterator.reset();
    tab++;

    System.out.println(calisanTipi + ", " + adSoyad + ", " + maas

    while (cIterator.hasNext())
    {
        for (int i = 0; i < tab; i++)
            System.out.print("     ");

        ICalisan c = (ICalisan) cIterator.next();
        c.calisanListele();
    }

    tab--;
}
```

maliyetHesapla() gerçekleştirilirken yine calisanListele() metodundaki mantık hakimdir. Farklı olarak (int) maliyet değişkenine kendinin, astlarının ve astlarının astları… olmak üzere özyinelemeli şekilde emrinde çalışan herkesin maaşı toplanıp döndürülür.

```
public int maliyetHesapla()
{
    cIterator.reset();

    int maliyet = 0;
    maliyet += maas;

    while (cIterator.hasNext())
        maliyet += cIterator.next().maliyetHesapla();

    return maliyet;
}
```

İki metod için de aynı iterator (cIterator) kullanıldığı için metodların başlangıçlarında cIterator.reset() bulundurularak olası karışıklık önlenir.

2.d. IKoleksiyon:

Direktör sınıfının içindeki calisanDizisi, iterator ile dönüleceği için Direktör bu sınıfı implement eder. iteratorOlustur() metodunun içinde calisanDizisi parametresi kullanılarak CalisanIteratoru oluşturulur.

```
public interface IKoleksiyon
{
    public Iterator iteratorOlustur();
}
```

2.e. Iterator:

CalisanIteratoru sınıfı tarafından implement edilir. Bulunmasındaki amaç, dizi yerine başka bir veri yapısının gezilmesi gerektiğinde de hasNext(), next() ve reset() metodlarına ihtiyaç duyulacak olmasıdır.

```
public interface Iterator
{
    boolean hasNext();
    ICalisan next();
    void reset();
}
```

2.f. CalisanIteratoru:

(ICalisan[]) calisanDizisi ve int tipinde indeksine sahiptir. Yapıcısı ICalisan nesnesi tutan bir dizi alır ve calisanDizisi'ne bu dizinin içeriğini aktarır.

hasNext() için calisanDizisi boş mu veya dolu mu diye bakılır, true veya false döndürülür.

next() için indeks bir artırılarak dizinin ilgili değeri döndürülür.

reset() için indeks değeri 0 yapılır.

2.g. IVeriOkuyucu:

Başka bir ortamdan okuma yapılacağı zaman ilgili ortam için sınıf yazılır ve bu sınıf IVeriOkuyucu'yu implement eder.

```
public interface IVeriOkuyucu
{
    ArrayList<String[]> cozumle();
}
```

Yeni sınıf, yaptığı okuma sonrasında yine DosyadanOkuma sınıfında bulunduğu gibi her bir nesnenin calisanTipi, calisanAdiSoyadi, maas ve calisaninUstu bilgileri elde edilip bir String dizisine yerleştirir. Ve bu String dizileri ArrayList'e eklenerek ArrayList döndürülür.

2.ğ. DosyadanOkuma:

Scanner ile okuma yapılır. Satır satır okuma gerçekleştirilir. Her satır içindeki virgüllerden ayrılarak bir String dizisi oluşturur. Bu diziler sırayla bir ArrayList'e eklenir ve ArrayList döndürülür.

```
// D, Mustafa Türksever, 5000, Root
// calisanTipi, calisanAdiSoyadi, maas, calisaninUstu
String[] calisanOzArr = { satir_arr[0], satir_arr[1], satir_arr[2], satir_arr[3] };

calisanOzAL.add(calisanOzArr);
```

3. BİRİM TESTİN GERÇEKLEŞTİRİMİ

D(Mustafa Turksever,5000) *D:DİREKTÖR, M:MEMUR anlamındadır.*
D(Halil Sengonca,4000)
D(Ugur Guclu,2000)
M(Emre Kosar,700) M(Ahmet Egeli,700)
D(Sedat Tunc,2500)
M(Bora Kuzey,1000)
D(Oguz Demir,3000)
M(Önder Bati,500) M(Erdem Altin,500) M(Mehmet Bilir,600)
D(Bahar Karaoglan,3500)

Birim testte maliyetHesapla() metodunun çalışışı kontrol edilmiştir. Kırmızı dikdörtgenin içerisine alınmış direktörlerin nesnelerinin adları yukarıdan aşağıya D1, D2, D3; memurlarınkiyse soldan sağa M1 ve M2 şeklindedir.

İlk testte D1'in hiçbir astı olmadığı için sadece kendi maaşını döndürüp döndürmediği kontrol edilir.

```
@Test
void Test1_CalisansizDurum()
{
    assertEquals(5000, d1.maliyetHesapla());
}
```

İkinci testte D1'e D2 eklenip D1'in maliyeti kontrol edilir. Bu sayede eklenen direktörlerin maaşlarının, üstlerinin toplam maliyetine eklendikleri kanıtlanır.

```
@Test
void Test2 DirektorEklenmesi()
{
    d1.calisanEkle(d2);
    assertEquals(9000, d1.maliyetHesapla());
}
```

Üçüncü testte M1, D3'e eklenerek D3'ün maliyeti kontrol edilir. Böylelikle direktörlere eklenen memurların maaşlarının, üstlerinin toplam maliyetlerine eklendikleri kanıtlanır.

```
@Test
void Test3_MemurEklenmesi()
{
    d3.calisanEkle(m1);
    assertEquals(4200, d3.maliyetHesapla());
}
```

Dördüncü ve son testte D3, D1'e ve M2, D3'e eklenir. Kırmızı alandaki tüm çalışanların toplam maliyetleri D1'in maliyetinin hesaplanması aracılığıyla kontrol edilir.

```
@Test
void Test4_ToplamMaliyetinHesaplanması()
{
    d1.calisanEkle(d3);
    d3.calisanEkle(m2);
    assertEquals(13900, d1.maliyetHesapla());
}
```