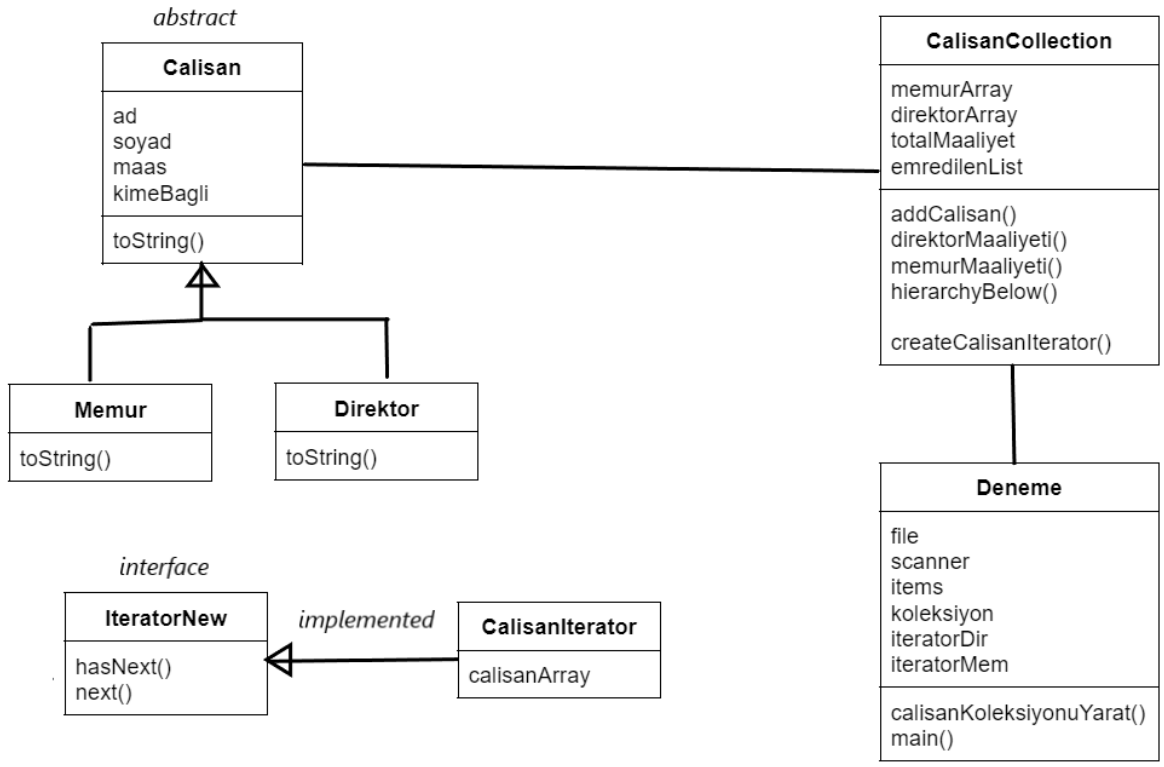
Nesne Yönelimli Programlama Proje Raporu

UML Diyagramı:



Sınıfların Açıklamaları:

Öncelikler **Calisan** adinda **abstract** bir ana sınıf oluşturdum. Her çalışanın ad, soyad, maaş ve kime bağlı olduğunu belirten parametrelere sahip olacağını kabul ettim ve bunları yazdıracak **toString()** metodunu **override** ettim.

girdi.txt dosyasından gelen veriler (çalışanlar) iki türde (memur/direktör) olacağından **Calisan** sınıfından **Memur** ve **Direktor** alt sınıflarını türettim. Bu sınıflar da farklı parametre bulunmadan üst sınıfının parametrelerini kullanıyor. **Memur** veya **Direktor** nesnelerinin yazdırılabilmesi için ayrı ayrı **toString()** metodlarını **override** ettim. **getter** metodlarını oluşturdum.

Iterator pattern'ı uygulayabilmek için **IteratorNew** arayüzünü oluşturdum. Arayüze **hasNext()** ve **next()** metodlarını ekledim. Farklı çalışan türleri için **IteratorNew**'i **implement** edebilen **CalisanIterator** sınıfını oluşturdum. Bu sınıfın içinde **hasNext()** ve **next()** metodlarını uyguladım. Bu sınıfta üzerinde **iterate** edilecek **calisanArray** dizisini oluşturdum.

Fonksiyonel uygulamaları içermesi için **CalisanCollection()** sınıfını oluşturdum. Burada **memurArray** ve **direktorArray** ile dizilerini oluşturdum. **addCalisan()** metoduyla **input**'tan gelecek çalışanları ayırıp özelliklerine göre nesne dizilerine yükledim. Seçilen bir direktörün altındaki çalışanları tutması için **emredilenler** listesini oluşturdum. **direktorMaliyeti()** metodu ile hem yukarıdan aşağı çalışanları tarayıp gerekiyorsa **emredilenler**'e ekledim hem de kümülatif maliyeti hesapladım. **hierarchyBelow()** fonksiyonu direktör maliyetini hesaplamada yardımcı olarak çalışır. **memurMaliyeti()** sadece memurun kendi maaşını döndürür ve yazdırır. Son olarak **createCalisanIterator()** ile dizileri gezebilecek bir **iterator** oluşturulur.

Deneme sınıfını ile oluşturduğum tasarımı **input** örneğiyle birlikte çalıştırmak için oluşturdum. Önce **.txt** dosyasını hata tespiti ile çağırdım. Sonra tüm **.txt** içeriğini bir **string**'e yükledim. **string**'i boşluklara göre parçalayarak bir **items** listesi oluşturdum. Deneme sınıfında **CalisanCollection**'daki fonksiyonlardan yararlanmak için bir **koleksiyon** nesnesi oluşturmam gerekiyordu. Her yeni **koleksiyon** oluşturmayı **calisanKoleksiyonuYarat()** metodunda gerçekleştirdim. Bu metodun parametre olarak aldığı listedeki elemanları bir memur veya direktör nesnesine dönüştüren döngü yazdım. Oluşturduğum koleksiyonlardan **createCalisanIterator()** fonksiyonu ile **IteratorDirektor** ve **IteratorMemur** iteratörlerini oluşturdum. **while** döngüleriyle iteratörlerin sahip olduğu diziyi dolaşarak memurları ve direktörleri ham bir şekilde yazdırdım. Son olarak istenilen isimler için direktör maliyetini ve emrinde çalışanları veya memur maliyetini yazdırdım.

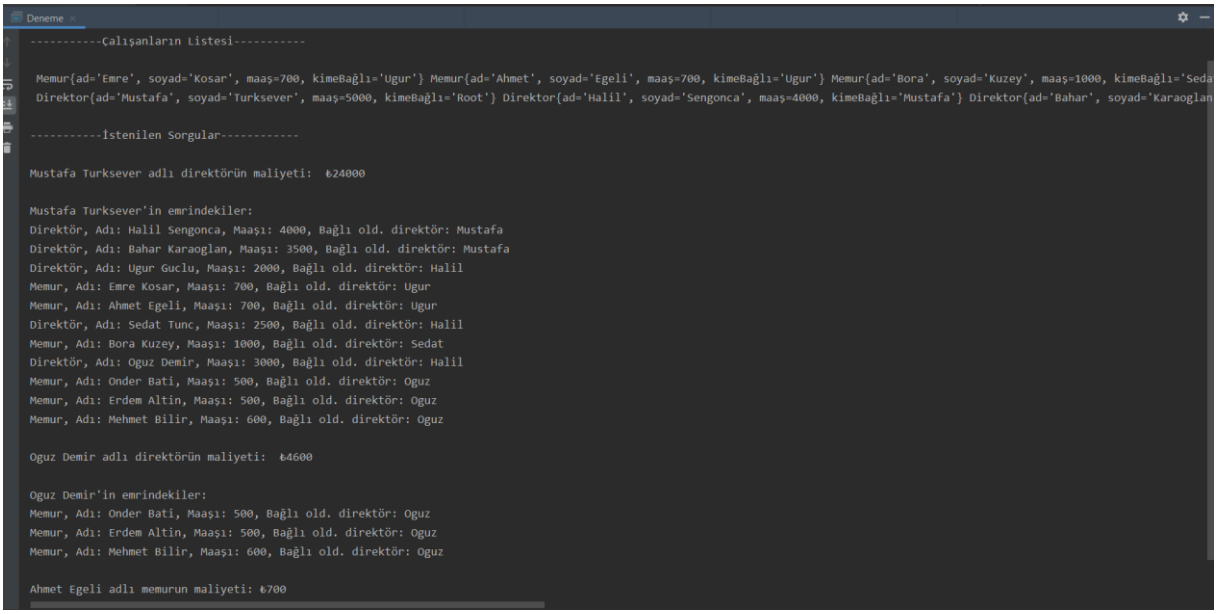
DenemeTest sınıfı ile birim test oluşturdum. Burada dosyanın bulunup bulunmamasını test ettim. Ayrıca oluşturulan iteratörlerin (**IteratorDirektor/ IteratorMemur**) null olup olmadığını test etmek için **assertNotNull()** metodunu kullandım.

Tasarım Desenleri:

Bu yapıda tek bir sınıftan türetilen birden fazla türe sahibiz. Ayrıca bu iki tür kendi aralarında hiyerarşik yapıda olmalarından dolayı **Composite Pattern**'ı uygun buldum ve bu tasarımı uygulamaya çalıştım.

Iterator Pattern'ı hiyerarşik yapıda dolaşmak için kullandım. Bunu da yeni bir **Iterator** arayüzü ve uygulamaları ile tasarladım.

Deneme.java Output:



DenemeTest.java Output:

IteratorNew iteratorMem = null; *olarak atandığı taktirde:*