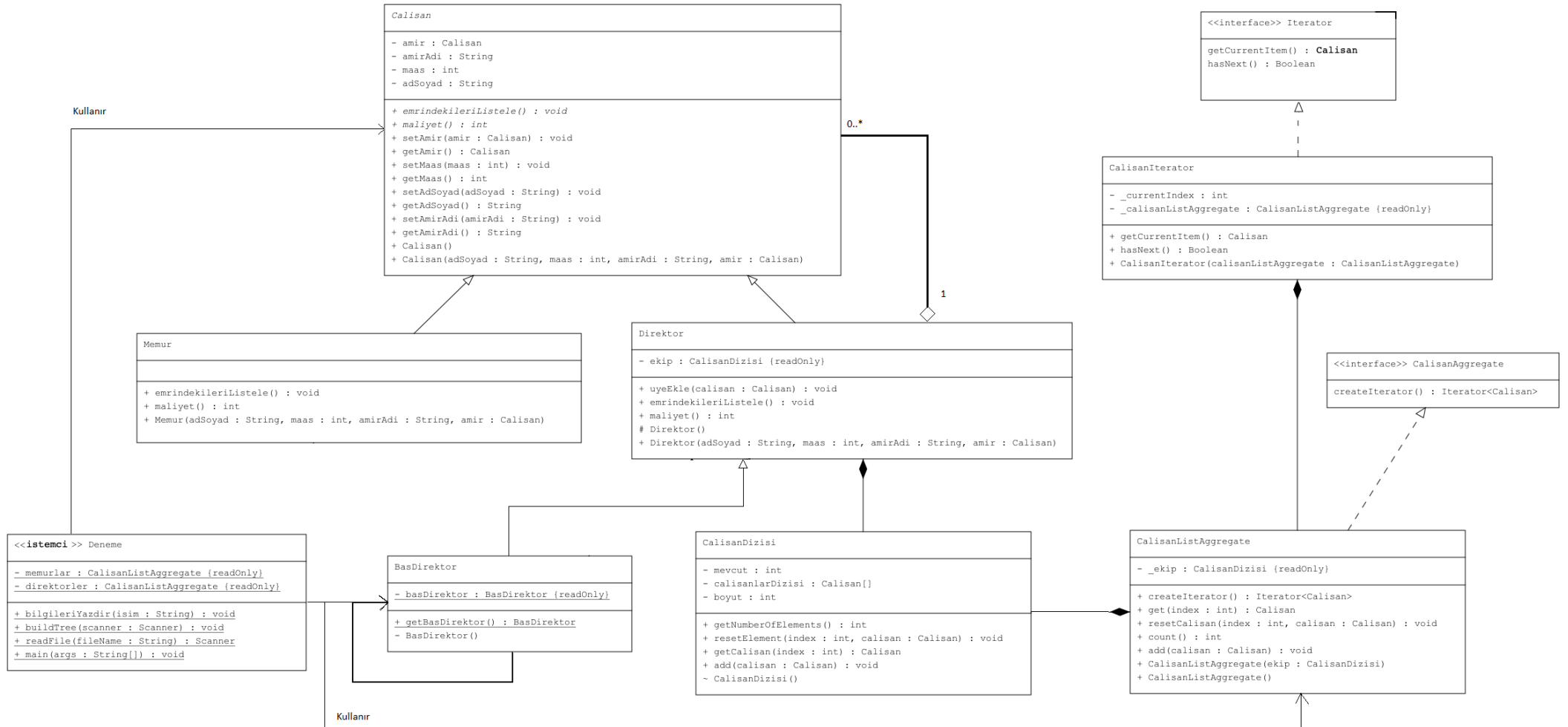


İçindekiler

1. UML Sınıf Diyagramı	2
2. Sınıfların Açıklamaları	3
3. Kodların Çalıştırılması ve Sonuç	8
4. Birim Testler ve Sonuç.....	8

1. UML Sınıf Diyagramı



2. Sınıfların Açıklamaları

Bu projede Composite, Singleton ve Iterator tasarım desenlerini kullandım.



Şekil 1 : Composite tasarım deseni

Calisan Sınıfı : Hiyerarşik ağaç yapısının temel bileşeni olan bu soyut sınıfta, abstract metot kontratlarıyla birlikte, alt sınıflarda (Memur ve Direktor) ortak olan tüm sahaları ve metotları (getter ve setter'lar) tanımladım. Böylece bu sınıftan türetilen Memur ve Direktor sınıflarında sadelik sağladım.

2 adet abstract metot tanımladım. Birincisi, `int` tipinde değer döndürecek olan `maliyet()` metodudur. Bu metot, herhangi bir çalışanın firmaya maliyetinin hesaplanması için Memur ve Direktor sınıflarında içi doldurulacak olan abstract metottur. İkincisi, `void` tipinde, bir çalışanın hiyerarşik olarak altında çalışanların listelenmesi için Memur ve Direktor sınıflarında içi doldurulacak abstract metot olan `emrindekileriListele()` 'dir.

Dosyadan okunan verilere göre, sırasıyla ad soyad, maaş, amir adı ve amir nesnesi değerlerini argüman olarak alan constructor ile Memur ve Direktor alt sınıflarından kendi tiplerine göre nesneler oluşturulacak.

Parametresiz constructor yer almasının sebebi, Singleton tasarım deseni ile oluşturulan BasDirektor sınıfının "adSoyad", "maas", "amirAdi" ve "amir" sahaları setter metotlarıyla belirlenecek olup Singleton tasarım deseni gereği BasDirektor sınıfında private parametresiz constructor yazma gereksinimidir.

Memur Sınıfı : `Calisan` sınıfından miras yoluyla türetilen, hiyerarşinin en alt aşamasını (leaf) temsil eden somut sınıftır. `Memur` sınıfının constructor'ındaki parametreler `super` belirteci ile doğrudan üst classı olan `Calisan` sınıfının constructor'ına gönderilip sadelik sağlanmıştır.

`Calisan` sınıfından miras alınan `maliyet()` ve `emrindekileriListele()` abstract metotları override edilerek gerekli şekilde doldurulmuştur. Bir memura bağlı kimse olmayacağı için, o memurun maliyeti sadece kendi maaşıdır ve o da `super` class'ın `getMaas()` metodu ile döndürülür. Benzer şekilde, bir memurun emrinde kimse olmayacağı için, `emrindekileriListele()` void metodunda sadece kendi bilgilerini yazdırdım.

Direktor Sınıfı : `Calisan` sınıfından miras yoluyla türetilen, ağaç yapısında hiyerarşik olarak memura göre üstte olan ve kendisine bağlı çalışanları olan somut sınıftır. Composite tasarım deseni gereği bir direktöre bağlı olan çalışanları tutan, esneklik sağlanması için özel olarak yazılan `CalisanDizisi` sınıfından oluşturulan diziye sahiptir. Bu diziye `uyeEkle()` metodu ile Composite tasarım deseni gereği direktöre bağlı çalışanları ekliyorum. Memura bağlı başka çalışan olmadığı için, `CalisanDizisi` dizisi ile `uyeEkle()` metodu `Direktor` sınıfına özgüdür.

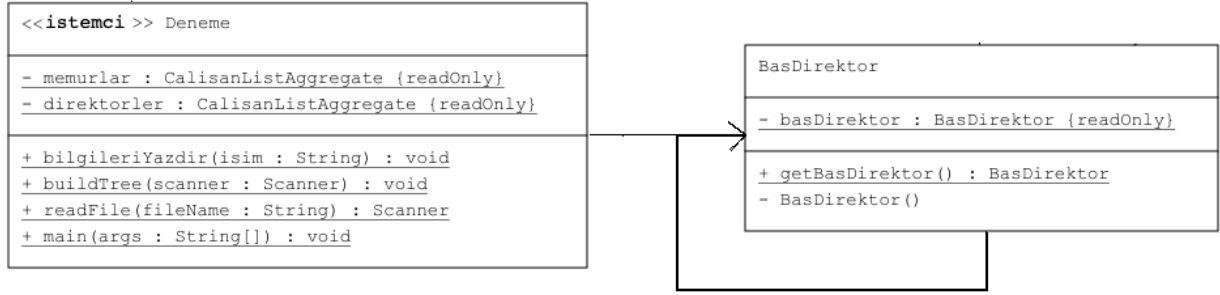
`Calisan` sınıfından miras alınan `maliyet()` ve `emrindekileriListele()` abstract metotları override edilerek gerekli şekilde doldurulmuştur. `Direktor` nesnesi üzerinden maliyeti hesaplamak için `maliyet()` metodunda `Iterator` tasarım desenini kullandım. Benzer şekilde, direktor nesnesi üzerinden hiyerarşiyi yani kendisine bağlı olan çalışanları yazdırmak için bu metotta da `Iterator` tasarım deseni kullandım.

`Direktor` sınıfı ile `Calisan` sınıfı arasında 1'e N ilişki vardır. Yani bir direktöre birden fazla çalışan bağlı olabilir ama bir çalışan sadece bir direktöre bağlı olabilir.

CalisanDizisi Sınıfı : Dosyadan okuma işleminde esneklik istendiği için bu sınıfı yarattım. Bu sınıfın sahaları, `Calisan` tipindeki nesneleri tutan bir dizi, bu dizinin ilk sabit boyutu ve dizinin en son dolu indisidir. Parametresiz constructor ile bu sınıftan bir nesne yaratılır. Default olarak dizinin boyutunu 12 olarak belirledim. Diziye `Calisan` tipindeki nesneleri ekleyecek olan `add()` metodunda ilk başta dizinin dolu olup olmadığı kontrol edilir. Eğer dizi doluysa iki katı büyüklükte yeni bir dizi oluşturularak, dolu dizi yeni yaratılan diziye kopyalanır ve yeni diziyle devam edilir. Dizide boş yer varsa, yeni eleman dizinin sonuna eklenir.

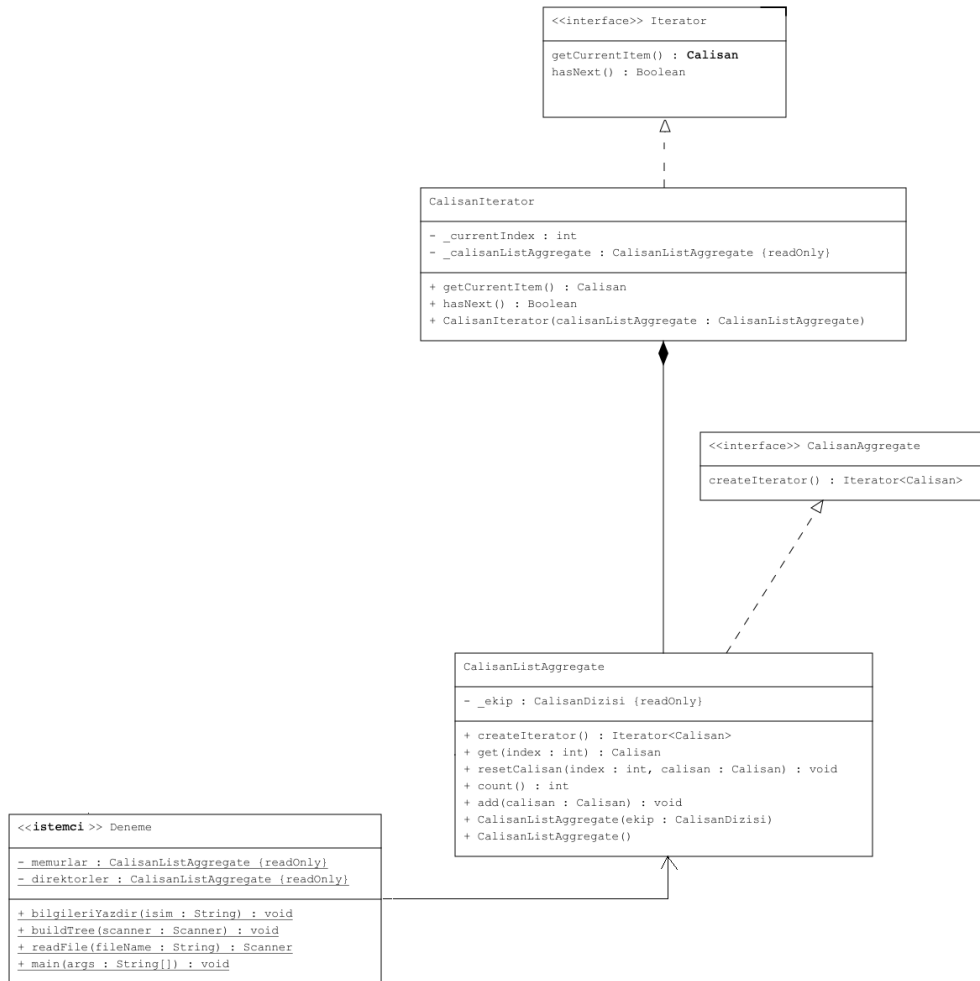
`getCalisan()` metodu, `indexi` belirtilen çalışanı döndüren metottur. Dosyadan okuma esnasında kullanıyorum.

`resetElement()` metodu, dosyadan okuma işlemlerinde ilk turda direktörleri diziye attıktan sonra, bu dizi üzerinde ikinci kez dolaşırken `amir` adı sahasında "Root" yazan direktörü bulup baş direktör olarak değiştirmeye yarıyor.



Şekil 2 : Singleton tasarım deseni

BasDirektor Sınıfı : Baş direktör de bir direktör olduğu için, Direktor sınıfından miras yoluyla BasDirektor sınıfını yazdım. Yalnız burada dikkat edilmesi gereken nokta, bu sınıftan sadece bir tane baş direktör nesnesi yaratılacak ve nereden çağrılırsa çağrılсын aynı nesne döndürülecektir; çünkü sadece bir baş direktör vardır. Bu yüzden Singleton tasarım deseni kullandım. private ve parametresiz constructor ile sınıfın kendi içinde bir adet nesne yaratıp bu nesneyi static getBasDirektor() metodu ile döndürüyorum. Söz konusu nesnenin sahaları üst sınıfının da bir üst sınıfı olan Calisan sınıfındaki setter metotları (setAdSoyad(), setMaas(), setAmirAdi(), setAmir()) ile ayarlanır.



Şekil 3 : Iterator tasarım deseni

Iterator Arayüzü: Iterator tasarım deseninin temel sınıfıdır. Koleksiyon üzerinde dolaşmak için operasyonların tanımlandığı arayüzdür. Generic bir yapı ile birçok yerde kullanılmasını sağladım.

`hasNext()` metodu koleksiyonda bir sonraki elemanın var olup olmadığını döndüren boolean tipinde bir metottur.

`getCurrentItem()`, koleksiyonda dolaşma esnasında o an geçerli elemanı döndüren metottur.

CalisanAggregate Arayüzü : Bu arayüz içerisinde `Iterator` arayüzünü uygulayan sınıfların örneklerini elde etmeye yarar.

CalisanIterator Sınıfı : `Iterator` arayüzünü ve koleksiyonun üzerinde dolaşmak için gerekli metotları uygular. Veriler üzerinde dolaşma işlemleri burada gerçekleştiriliyor. Bunun için `CalisanListAggregate` sınıfı ile birlikte çalışıyor.

CalisanListAggregate Sınıfı : `CalisanAggregate` arayüzünü uygular ve `Iterator` arayüzünü uygulayan sınıfın örneğini üretir. Iterator tasarım deseninde bu sınıf sayesinde iteratör nesneleri yaratarak koleksiyon üzerinde dolaşma işlemini gerçekleştiriyoruz.

Deneme Sınıfı : Özetle, `main()` metodu içinde `girdi.txt` dosyasını okuyarak verilmiş olan hiyerarşik yapıyı oluşturdum ve kişilerle ilgili istenen bilgileri yazdırdım.

En başta, Iterator deseni için gerekli olan, `CalisanListAggregate` sınıfından, direktör nesnelerini tutan diziyi içeren “direktorler” nesnesini ve memur nesnelerini tutan diziyi içeren “memurlar” nesnesini yarattım.

`readFile()` statik metodu ile dosyadan okuma işlemini yaptırıp `Scanner` nesnesini döndürür.

`buildTree()` statik metodunda, az önce oluşturduğum `Scanner` nesnesini kullanarak Composite tasarım deseni ile hiyerarşik yapıyı kurdum. Bu metottaki `while`’ın her bir döngüsünde, dosyayı satır satır okuma, satırın içindeki değişkenleri virgöl kısımlarından ayırma, ayırma sonucunda meydana gelen dizinin ilk elemanının “D” harfi olmasına göre `direktorler` nesnesi içindeki diziyi, “M” harfi olmasına göre `memurlar` nesnesi içindeki diziyi ekleme işlemlerini yaptırardım. Dosya ilk kez okunurken, memur ve direktör nesnelerini oluştururken amir sahalarını “null” olarak bıraktım. Çünkü bu adımda çalışanları sadece tiplerine göre ayırdım. Yani henüz Composite yapı kurulmadı.

`while`'dan sonraki `for` döngüsü esas olarak Composite yapıyı kurduğum kısımdır. `for` döngüsünü `direktorler` nesnesi üzerinde dolaşmak için yaptım. En baştaki `if` bloğunda, amir adı "Root" olan çalışan aranıyor. Bu çalışan baş direktördür. Baş direktör bulunduktan sonra `direktorler` nesnesi içindeki dizinin o anki `Direktor` tipli elemanının yerine `resetCalisan()` metodu ile `BasDirektor` tipli baş direktör elemanı yazılıyor. Bu işlem için yazdığım `if` bloğu baş direktör bulunduktan sonra bir daha koşul gerçekleşmeyeceği için tekrar çalışmayacaktır. (`buildTree()` metodunu, okunan dosyada baş direktörün yer almadığı veya baş direktörün ilk sırada değil daha sonra yazıldığı durumlarda dahi düzgün bir yapı kurulacak şekilde yazdım.)

`if` bloğundan sonraki `for` döngüsünde ise `memurlar` nesnesi üzerinde dolaşılıyor. Dıştaki `for` döngüsündeki o anki direktörün ad soyadı, içteki `for` döngüsündeki memurun amir adı sahasını içeriyorsa (amir adı sahası sadece ilk isimleriyle verildiği için ve karşılaştırma yapacağımız direktör ise hem isim hem de soyisimden oluştuğu için `java`'nın `String` sınıfında hazır bulunan `contains()` metodunu kullandım) o andaki direktör o andaki memurun `setAmir()` metodu ile amiri olarak atanıyor. Aynı zamanda o andaki memur da o andaki direktörün ekibine `uyeEkle()` metodu ile ekleniyor.

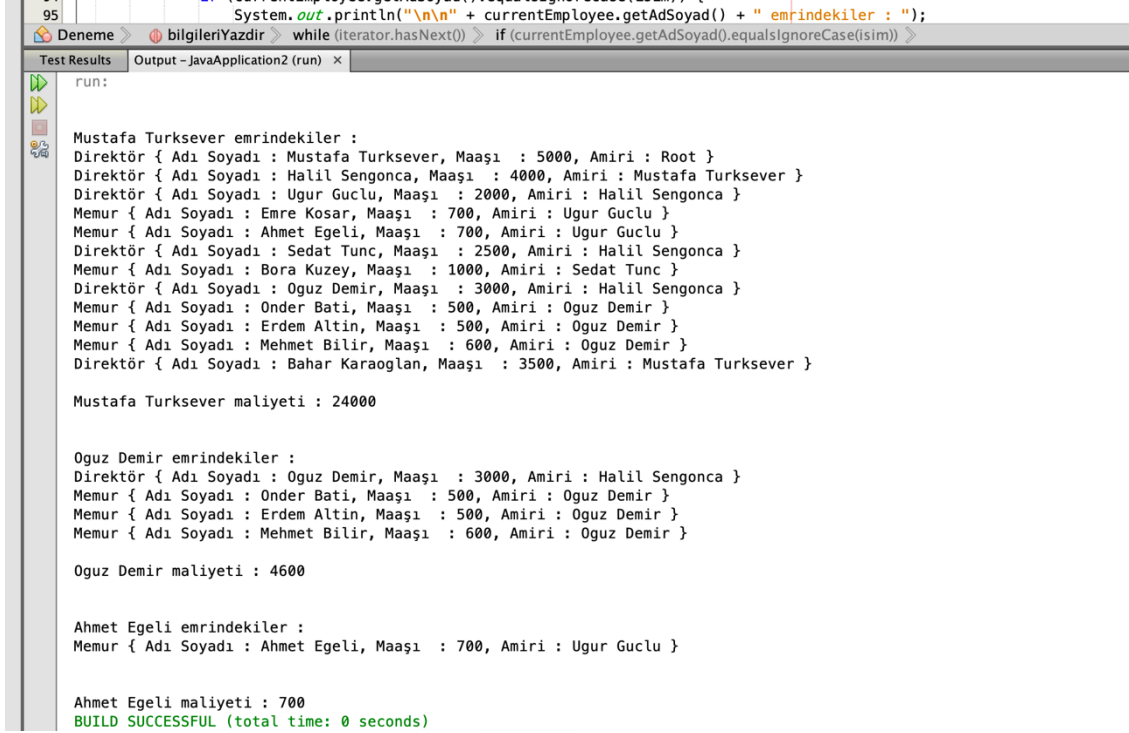
İçteki ikinci `for` döngüsünde, aynı mantığı bu sefer direktörler arasında uyguladım.

`main()` metodu içinde son olarak kullandığım metot, İteratör tasarım desenine ait nesneler sayesinde yazdırma işlemleri yaptırdığım statik `bilgileriYazdir()` metodudur. Bu metot bilgileri yazdırılmak istenen çalışanların ad soyadını parametre olarak alıyor. İlk olarak `direktorler` nesnesi üzerinden bir iteratör yaratıp bu iteratör sayesinde dolaşma işlemini yaptırdım. Yaratılan iteratör nesnesinin `hasNext()` boolean metodu sayesinde `while` koşulu sağlandıkça döngü devam eder. Döngünün içinde iteratör nesnesinin `getCurrentItem()` metodu ile döndürülen `Calisan` tipli nesne `Direktor` sınıfına downcast edilmelidir. Çünkü döndürdüğü nesne tipi `Calisan` tipindedir. Elde edilen direktör nesnesinin ad soyadı, metodun parametresiyle uyuşuyorsa `emrindekileriListele()` metodu ile kendisine bağlı çalışanlar yazdırılıyor, `maliyet()` metoduyla kendisi ve kendisine bağlı tüm çalışanların maaşlarının toplamı yazdırılıyor ve `break` ile döngüden çıkılıyor.

Aynı mantığı memurlar için tekrar uyguladım.

3. Kodların Çalıştırılması ve Sonuç:

Kodlar başarılı bir şekilde çalışıyor ve bu proje için istenenleri eksiksiz bir şekilde listeliyor. Kodları çalıştırdığımızda oluşan çıktının ekran resmi aşağıdaki gibidir.



```
95 System.out.println("\n\n" + currentEmployee.getAdSoyad() + " emrindekiler : ");
Deneme > bilgileriYazdir > while (iterator.hasNext()) > if (currentEmployee.getAdSoyad().equalsIgnoreCase(isim)) >
Test Results Output - JavaApplication2 (run) x
run:
Mustafa Turksever emrindekiler :
Direktör { Adı Soyadı : Mustafa Turksever, Maaşı : 5000, Amiri : Root }
Direktör { Adı Soyadı : Halil Sengonca, Maaşı : 4000, Amiri : Mustafa Turksever }
Direktör { Adı Soyadı : Ugur Guclu, Maaşı : 2000, Amiri : Halil Sengonca }
Memur { Adı Soyadı : Emre Kosar, Maaşı : 700, Amiri : Ugur Guclu }
Memur { Adı Soyadı : Ahmet Egeli, Maaşı : 700, Amiri : Ugur Guclu }
Direktör { Adı Soyadı : Sedat Tunc, Maaşı : 2500, Amiri : Halil Sengonca }
Memur { Adı Soyadı : Bora Kuzey, Maaşı : 1000, Amiri : Sedat Tunc }
Direktör { Adı Soyadı : Oguz Demir, Maaşı : 3000, Amiri : Halil Sengonca }
Memur { Adı Soyadı : Onder Bati, Maaşı : 500, Amiri : Oguz Demir }
Memur { Adı Soyadı : Erdem Altin, Maaşı : 500, Amiri : Oguz Demir }
Memur { Adı Soyadı : Mehmet Bilir, Maaşı : 600, Amiri : Oguz Demir }
Direktör { Adı Soyadı : Bahar Karaoglan, Maaşı : 3500, Amiri : Mustafa Turksever }

Mustafa Turksever maliyeti : 24000

Oguz Demir emrindekiler :
Direktör { Adı Soyadı : Oguz Demir, Maaşı : 3000, Amiri : Halil Sengonca }
Memur { Adı Soyadı : Onder Bati, Maaşı : 500, Amiri : Oguz Demir }
Memur { Adı Soyadı : Erdem Altin, Maaşı : 500, Amiri : Oguz Demir }
Memur { Adı Soyadı : Mehmet Bilir, Maaşı : 600, Amiri : Oguz Demir }

Oguz Demir maliyeti : 4600

Ahmet Egeli emrindekiler :
Memur { Adı Soyadı : Ahmet Egeli, Maaşı : 700, Amiri : Ugur Guclu }

Ahmet Egeli maliyeti : 700
BUILD SUCCESSFUL (total time: 0 seconds)
```

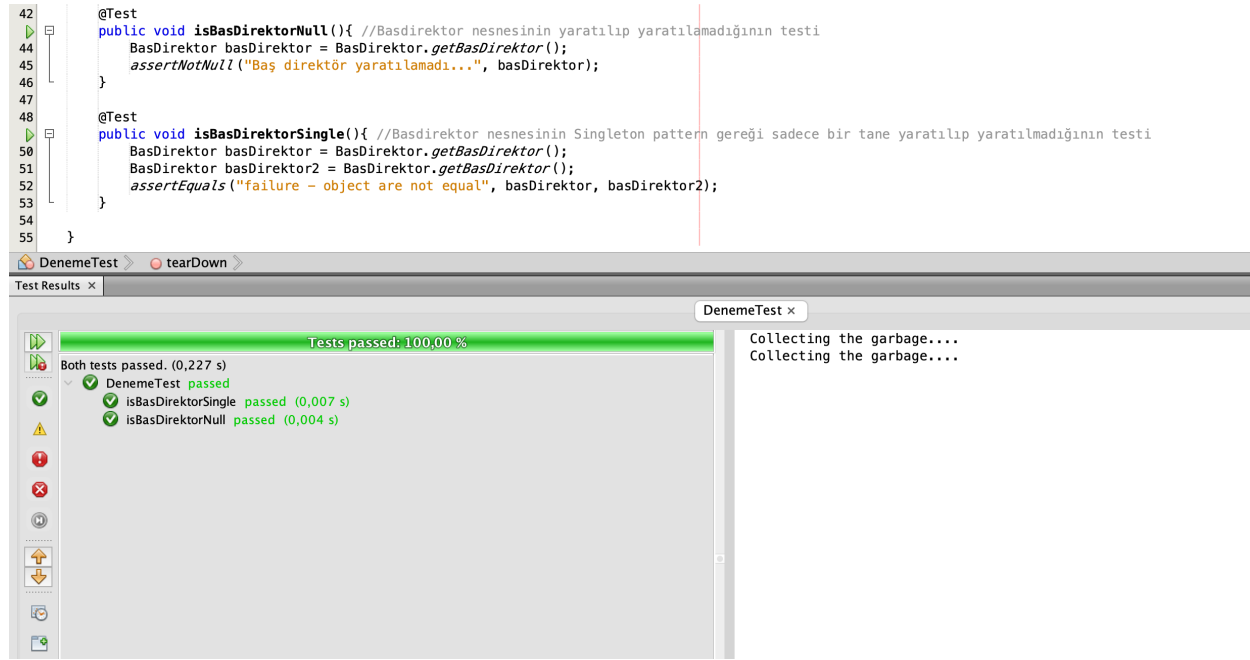
Şekil 4 :Programın çalışması başarılı.

4. Birim Testler

Birim testler için JUnit 4 kullandım.

Deneme sınıfı için DenemeTest adında bir test sınıfı yarattım. Bu test sınıfında Basdirektor nesnesinin yaratılıp yaratılmadığının test eden isBasDirektorNull() metodu ve Basdirektor nesnesinin Singleton tasarım deseni gereği sadece bir tane yaratılıp yaratılmadığının test eden isBasDirektorSingle() metotlarını yazdım.

Test çalıştırıldığındaki ekran görüntüsü aşağıdaki gibidir.



Şekil 5 : Test sonuçları başarılı

Yazdığım birim testler başarılı oldu.

`isBasDirektorNull()` metodu içindeki `assertNotNull()` assertion'ına göre `BasDirektor` nesnesi null değil, yani yaratılabildi.

`isBasDirektorSingle()` metodu içindeki `assertEquals()` assertion'ına göre ayrı ayrı yaratılan `BasDirektor` nesneleri aynı olduğu için Singleton tasarım deseni gereği `BasDirektor` sınıfından sadece bir tane nesne yaratıldığı anlaşılmaktadır.