

**EGE UNIVERSITY**

**FACULTY OF ENGINEERING**

**COMPUTER ENGINEERING DEPARTMENT**

**OBJECT ORIENTED PROGRAMMING**

**2021–2022 FALL SEMESTER**

**PROJECT REPORT**

**DELIVERY DATE**

05/01/2022

**PREPARED BY**

05180000035, Ece TEK

05180000043, Eda GÜNEŞ

05180000047, Elifnaz ACUN

**Sınıf Listesi**

* **Employee:** Bu sınıfta composite design pattern’a uygun olarak alt bileşenlere ulaşılmasını ve kontrol edilmesini sağlamak için bir abstract class oluşturuldu. Alt sınıflara inherite edilecek değişken tipleri tanımlandı ve istenen işlemleri gerçekleştirecek abstract metotlar oluşturuldu. Bu sınıf composite design’a göre component sınıfıdır.
* **Director:** Bu sınıf Employee class’ını extend ederek tüm değişkenleri inherite etmiş oldu.Oluşturulacak her nesne için employeeCollection sınıfından türettiğimiz bir Employee tipli dizi atandı. Bu dizide addEmployee() metodu ile ilgili direktörün altında çalışan memur veya diğer direktörler tutuldu. cost() metodu ile tutulan dizide oluşturulan iterator dolaşarak direktörün maliyetini hesapladı. makeList() metodunu recursive olarak kullanarak ilgili direktörün altındaki tüm çalışanlar ekrana yazdırıldı. Bu sınıf composite design’a göre composite sınıfıdır.
* **Officer:** Bu sınıf Employee class’ını extend ederek tüm değişkenleri inherite etmiş oldu.cost() metodu ile, ilgili memurun yalnızca kendi maaşını döndürecek şekilde yazıldı. makeList() metodunda getName() ile memurun kendi adının dönmesi sağlandı. Bu sınıf composite desing’a göre leaf sınıfıdır.
* **Collection:** Bu interface’in içinde createIterator() metodu ile iterator oluşturulması sağlandı.
* **employeeCollection:** Collection interface’ini implement eden bu sınıfta Employee tipli bir dizi oluşturuldu.addEmployee() metodu ile bu diziye eleman ekleme metodu, employeeNumber() ile dizinin toplam eleman sayısını döndüren metot ve createIterator() ile de arrayIterator aracılığıyla iteratorun metodu oluşturuldu. Bu metotlar kullandığımız dizinin arrayList gibi etkin kullanılabilmesi amacıyla oluşturuldu.
* **Iterator:** Iterator interface’i oluşturduğumuz bu sınıfta iteratorun dizi içerisinde istediğimiz şekilde ilerleyebilmesi için gerekli metotlar yazıldı. hasNext() ile dizinin sonuna kadar dolaşılması sağlandı. Employee tipindeki next() metodu ile de iterator ilerletildi.
* **arrayIterator:** Iterator interface’ini implement eden bu sınıfta iterator metotlarının gerçekleştirimi yapıldı. Next() ile iteratorun dolaştığı dizide nesneler arasında ilerleme sağlandı. hasNext() metodu içinde dizinin boş olmaması ve uzunluğunun geçilmemesi şartıyla iterator dolaşımının devamlılığı sağlandı.
* **Context:** Context constructor’ına belirlenen strateji gönderildi. Sonrasında executeStrategy() metotu içinde bu stratejinin metotunun çağrımı gerçekleştirildi.
* **ReadTxt:** Verilen txt dosyasından okuma yaparak Director ve Officer nesnelerini oluşturuldu ve bunların ayrı arrayListler içerisinde tutulması sağlandı. Burada arrayList kullanımının amacı, hiyerarşik düzen için kontrollerin arrayList metotlarıyla yapılıp diziye hazır biçimde atamaktır. Böylece verimlilik arttırılır.
* **Strategy:** Bu interface ile farklı veritabanlarından veri okunmasını sağlayacak olan metot tanımlandı.
* **Main sınıfı:** Context nesnesini oluşturarak, belirlenen strateji gerçekleştirildi. Sonrasında Employee c nesnesine root ataması gerçekleştirildi. Bunun temel amacı findElement() metodunda inputla alınan çalışan adının arama işlemini daha kolay yapmasıdır. Çünkü root içerisinde tüm çalışanlar hiyerarşik düzende sıralanmıştır, ağaç içinde dolaşır gibi düşünülebilir. Başka veri girişi olacaksa aynı işlemler tekrarlanır. elemanInput() metotu ile de input alma kodu tekrarı engellenmiş oldu.

***JUNIT KULLANIMI***

Director ve Officer sınıfları için ayrı JUnit test class’ı açıldı. DirectorTest class’ı içinde Director class’ında bulunan addEmployee() metodu kullanılarak çalışan ekleyip iterator’ın gezinip gezinmediği assertTrue ile kontrol edildi. Aynı test içinde findElement() metodu içinde üretilen direktörün adının bulunabildiği assertFalse ile test edildi.

OfficerTest class’ı içinde Officer class’ında bulunan cost() metodu kullanılarak üretilen memurun maliyetinin doğru hesaplandığı assertEquals ile test edildi.

***DESIGN PATTERNS***

**Composite Design:** Aynı superclass’dan türeyen farklı tip nesneler için hiyerarşik bir yapı oluşturulması gerektiğinden ve yine bu nesneler arasında ayrım yapılmaksızın aynı işlem yapılmak istendiğinden karmaşadan kurtulmak amacıyla bu dizayn tercih edildi. Employee abstract sınıfı Component, Director sınıfı Composite ve Officer sınıfı da Leaf görevi gördü. Bu pattern Client’ı nesneler arası seçim yapmaya zorlamaz. Ayrıca recursive yapının kullanımını da destekler.s

**Iterator Design:**

**Strategy Design:** Farklı veri tabanlarından okuma işlemi yapılabilmesi için stratejiler oluşturmaya yarar. Seçilen stratejiye göre ilgili class kullanımı ile okuma işlemi gerçekleştirilir.