EGE UNIVERSITY

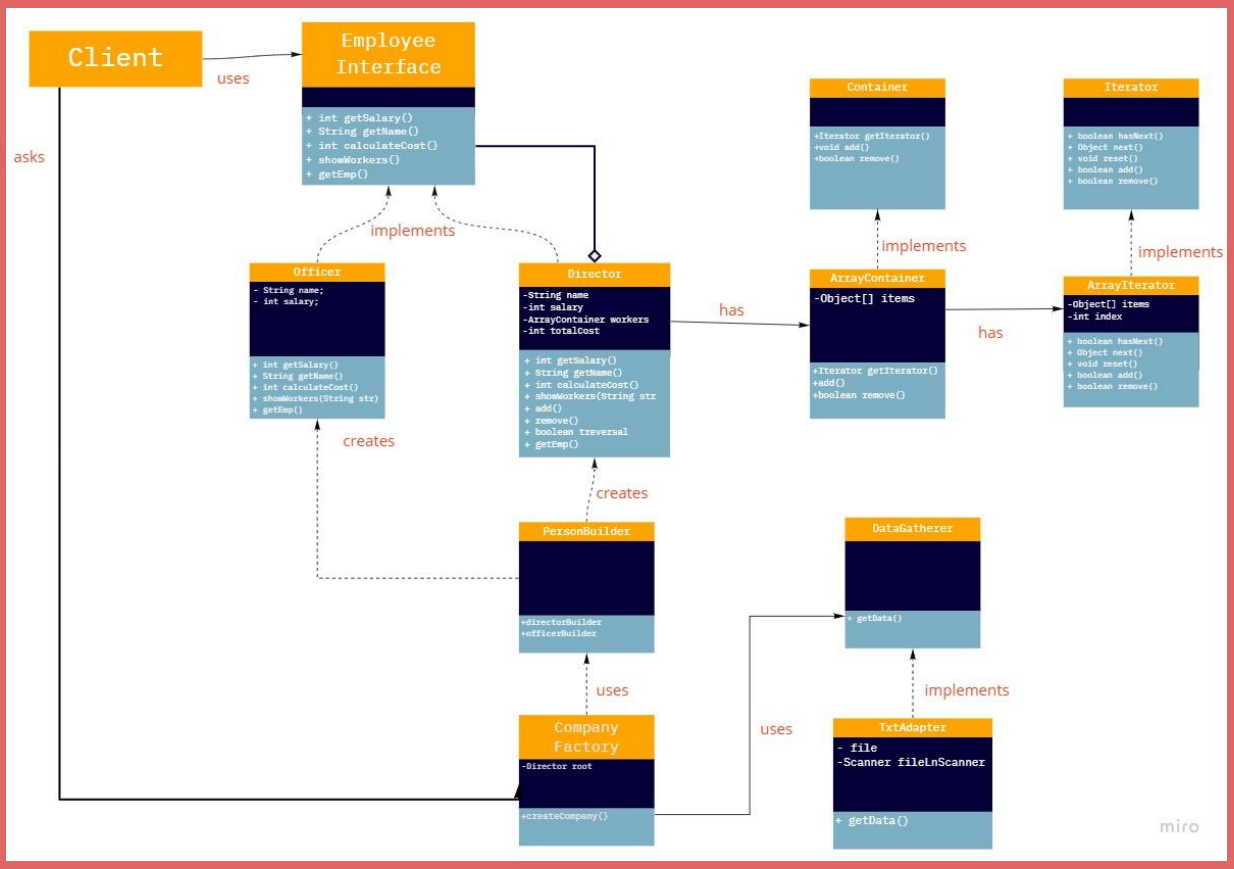
FACULTY OF ENGINEERING

COMPUTER ENGINEERING DEPARTMENT

Object oriented programming report

05180000037 - Sinan DÖŞEYİCİ
05190000061 - Oktay KALOĞLU

UML CLASS DIAGRAM



Not: Fotoğraf raporda net olmadığı için .jpg uzantılı uml diyagramını ekte de sunduk.

Definition of Classes

EmployeeInterface

Composite yapıda nesnelerin tutulabilmesi için alt nesnelerin bulundurması gereken ortak fonksiyonları belirten interface.

Officer

Composite Yapının leaf nodelarını ifade eder altında çalışanlar bulunmadığı için recursive fonksiyonlarda kendi değerlerini döndürür.

Director

Composite yapıda yöneticiyi ve çalışan collectionlarını bulundurur. Gerekli fonksiyonlar recursive yapıdadır.

Iterator

Collection ların kullanabileceği ortak metotları barındıran interface.

(add,remove,hasNext,next gibi)

ArrayIterator

Composite yapıda kullanılan array için özel metotları barındırır. Array yapısında index gibi bir tam sayı ile itemlere ulaşılabildiği için bu yapıya özel metotları barındırır.

Container

Composite yapıda çalışanların depolanabilmesi için gerek depolama yöntemleri için ortak metotları barındırır. En önemli metodu getIterator() dür. her bir container için o containerın iterator unu döndürür.

Array Container

Composite yapının collection larını arraylerle oluşturur. Ekleme sırasında diziler dolabilir, dolan diziler için deep copy yapar. getIterator() bu array container için yeni array iterator üretir.

CompanyFactory

Factory kendisine verilen dosya yolunda ki dosyadan adapter yardımı ile tek tek çalışan üreticek veriyi çeker ve composite yapıyı oluşturur.

PersonBuilder

Company factory için oluşturulması gereken nesneleri buildler.

DataGatherer

Farklı dosya türleri için Company factorynin kullandığı metodun her bir adaptee de gerçekleştirilmesini sağlar. getData() ile employee oluşturabilecek sadece bir tane string döndürür.

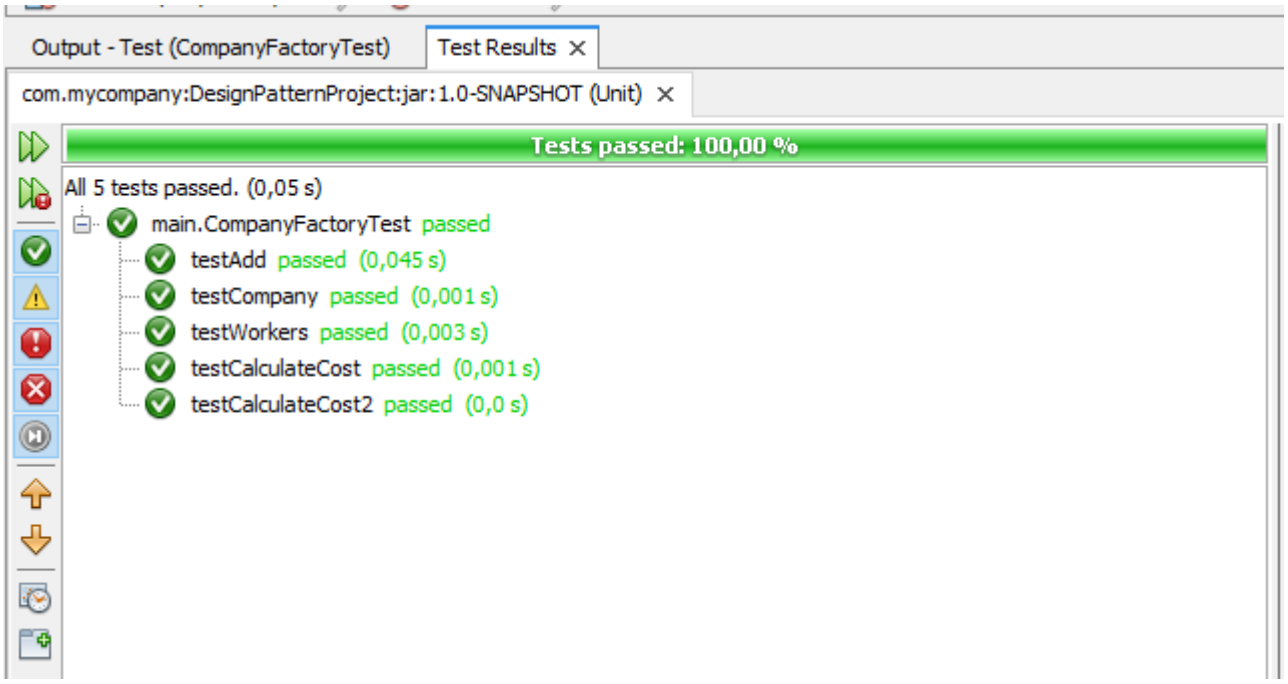
TxtDataGatherer

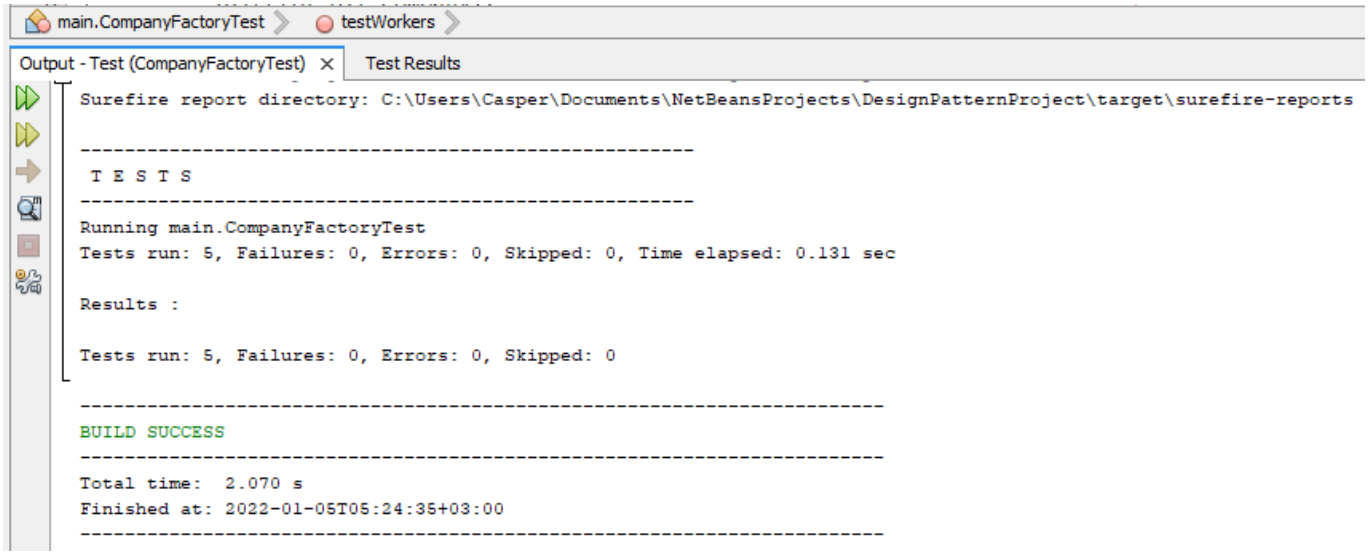
Uzantısı verilen dosyadan dataları satır satır çekip döndürür.

Junit Tests' Documentation

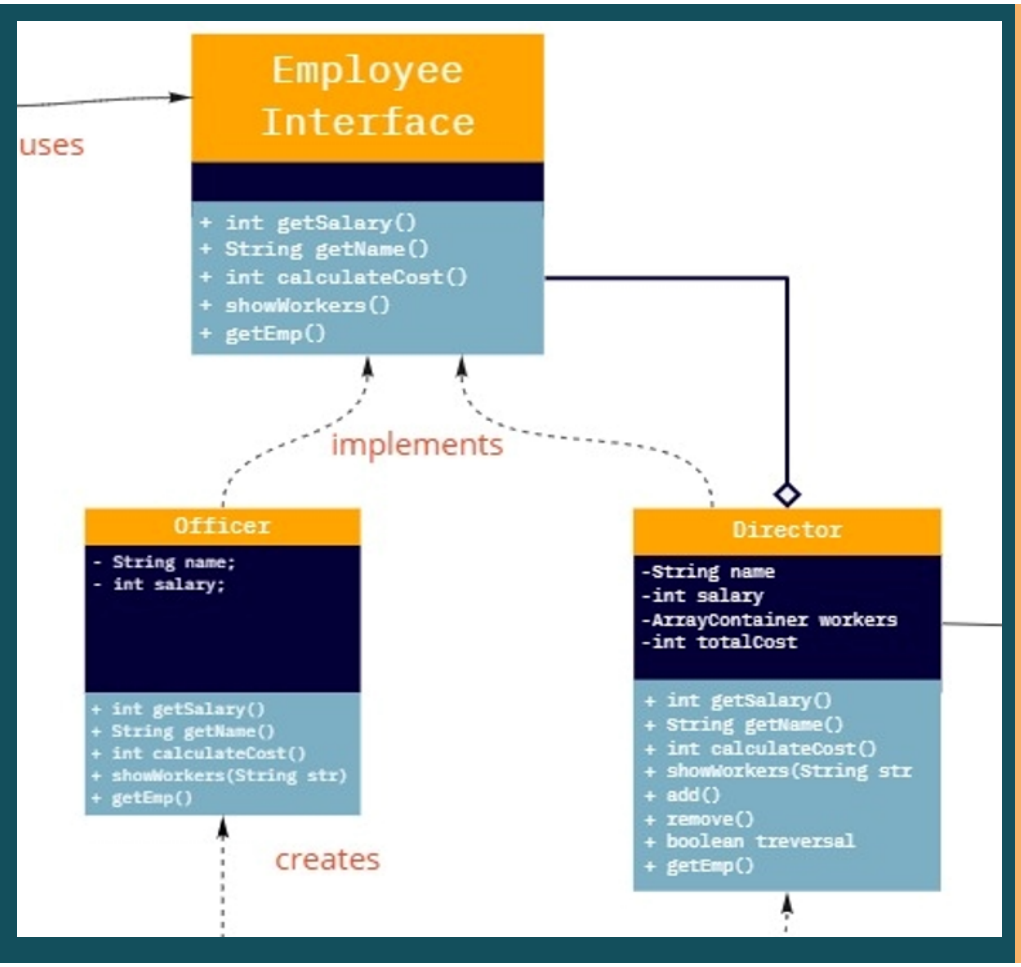
```
Director Company(){
    CompanyFactory company = new CompanyFactory("girdi.txt");
    return company.getDirector();
}

@Test
public void testCompany(){
    Director dir = Company();
    assertNotNull(dir,"Company is created");

}
@Test
public void testWorkers(){

    Director dir=Company();
    assertEquals(dir.getName(),"Mustafa Turksever","Doğru root");
    assertNotNull(dir.getEmp("Mustafa" ),"Mustafa var");
    assertNotNull(dir.getEmp("Oguz" ),"Oguz var");
    assertNotNull(dir.getEmp("Sedat" ),"Sedat var");
    assertNotNull(dir.getEmp("Ugur"  ),"Ugur var");
    assertNotNull(dir.getEmp("Halil" ),"Halil var");
    assertNotNull(dir.getEmp("Bahar" ),"Bahar var");
    assertNull(dir.getEmp("herhangi" ),"herhangi aslında yok");

}

@Test
public void testCalculateCost(){
    Director dir=Company();
    assertEquals(dir.calculateCost(),24000,"Doğru masraf");
}
@Test
public void testCalculateCost2(){
    Director dir=Company();
    assertEquals(dir.getEmp("Oguz").calculateCost(),4600,"Doğru masraf");
}
@Test
public void testAdd(){
    Director dir=Company();
    dir= (Director) dir.getEmp("Bahar");
    Director emp = new Director("ss",1);
    for (int i=0;i<15;i++){
        dir.add(emp);
    }

    assertEquals(dir.calculateCost(),3515,"Doğru çalışan sayısı");
}
```

Burada en aşağıda yazdığımız testAdd() testi ile kontrol etmek istediğimiz şey aslında çalışan sayısı ve bahar directorunun collectionunun limitinden fazla çalışan eklediğimizde yeni bir collection oluşturup oluşturmadığının kontrolü. "Bahar" isimli çalışanın altında kimsenin çalışmadığını bildiğimiz için eklenen employee sayısı kadar 1 birim fiyattan çalışan tanımladığımızda ve bunu calculateCost ile çağırdığımızda "Bahar" isimli çalışanın maaşı + çalışanların maaşları toplanmış olacak. Ve bu sayede "Bahar" isimli çalışanın altında kimse çalışmadığından, maaşı + çalışan sayısından bulmuş olacağız bu değer de 3500 + 15 'ten 3515 olmuş olacak. 15 kişi hatasız eklenmiştir.


Output - Test (CompanyFactoryTest)
Test Results
com.mycompany:DesignPatternProject:jar:1.0-SNAPSHOT (Unit)
Tests passed: 100,00 %
All 5 tests passed. (0,05 s)
main.CompanyFactoryTest passed
testAdd passed (0,045 s)
testCompany passed (0,001 s)
testWorkers passed (0,003 s)
testCalculateCost passed (0,001 s)
testCalculateCost2 passed (0,0 s)


main.CompanyFactoryTest
testWorkers
Output - Test (CompanyFactoryTest)
Test Results
Surefire report directory: C:\Users\Casper\Documents\NetBeansProjects\DesignPatternProject\target\surefire-reports
T E S T S
Running main.CompanyFactoryTest
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.131 sec
Results :
Tests run: 5, Failures: 0, Errors: 0, Skipped: 0
BUILD SUCCESS
Total time: 2.070 s
Finished at: 2022-01-05T05:24:35+03:00

How we used these design patterns !

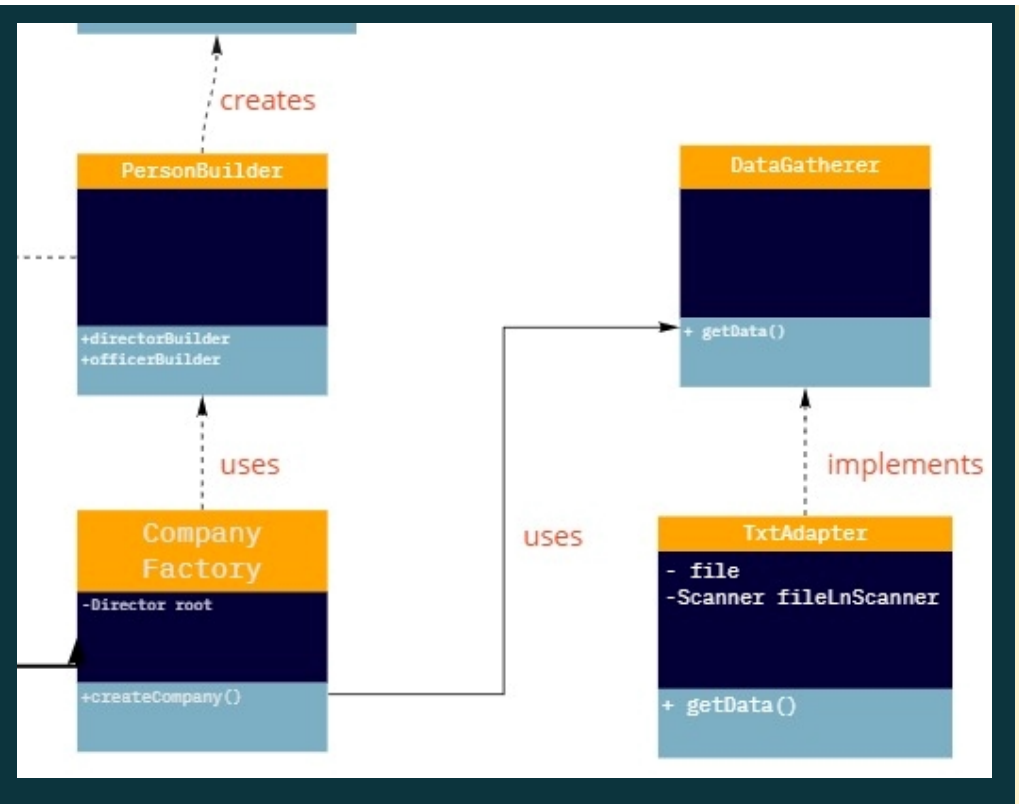


Composite Pattern

Bu tasarımda benzer özelliklerde nesneler kullanıldığı ve kullanılacağı için ve kullanıcının da collectionları yada leafleri birbirinden ayırt etmediği için tercih ettik. Employee componentini kullanarak geri kalan bütün compositeler ve leafler için aynı metotları kullanır. Bu nedenle her bir class için özel kontrolleri en aza indirdiği için kod karmaşıklığını azaltır.
Officer ı leaf olarak kullandık altında daha fazla çalışan olamayacağı için. Directoru ise composite olarak kullandık her bir director direk başka directorlere ve officerlara sahip olabileceği için

Iterator Design Pattern

Her bir director bir array içerisinde altında çalışan insanları tutar. Bu arrayler ileride daha farklı depolama metotlarını kullanabilecekleri için container implement ettik ki ileride ki farklılıklarda yazılan metodların değiştirilmeden kullanılabilmesi için. her bir container kendisine özel iterator tutmaktadır. Bu iteratorler her bir container ın kullanacağı ortak metodları implemente ederek değişen container yapısına uyum sağlar.



Factory - Builder - Adapter Design Patterns

Company factory ile kullanıcı girdiği dosyadan composite yapıca şirket oluşturmasını sağlar. Şirketi oluştururken person builderı kullanarak gerekli nesneleri yaratır. Nesneleri yaratmak için gerek dataları Data Gatherer adaptörü ile girilen dosyanın türü fark etmeksizin her bir nesne üretmek için gereken bilgileri string olarak dosyadan çeker. Adaptor ise gerekli dosyadan türüne uygun metotları içeren adaptörle gereken bilgileri getirir.