

**EGE ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**NESNEYE DAYALI PROGRAMLAMA**

# PROJE RAPORU

**HAZIRLAYANLAR**

**05180000107 - SÜLEYMAN AKTAŞ**

**05190000875 - MEHMET NURAYDIN**

**05190000871 - MEHMET AKDUMAN**

**TESLİM TARİHİ**

08/01/22

### GİRİŞ

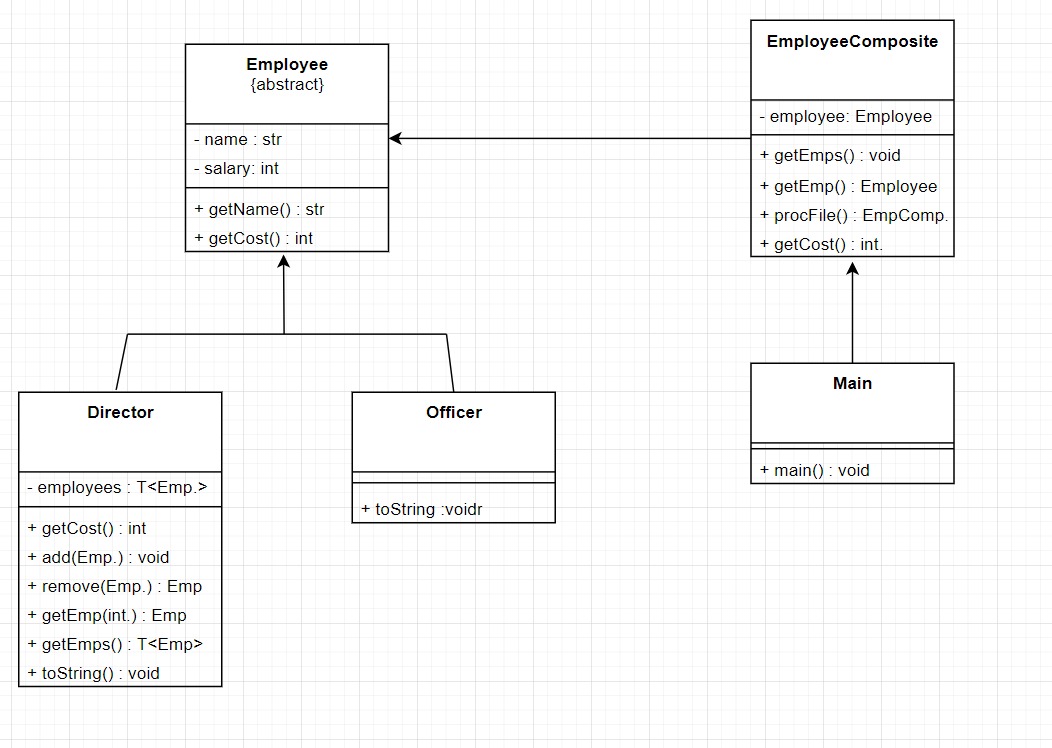
Proje kapsamındaki temel gereksinimler:

1. Bir firmanın çalışanlarına ilişkin hiyerarşik yapının, uygun bir tasarım deseni kullanılarak   
   temsil edilmesi.
2. Hiyerarşik yapının dolaşılması için Iterator deseni kullanılması.
3. Bir Deneme sınıfı yazarak, bu sınıfın main metodu içinde, verilen (girdi.txt) dosyayı okuyarak verilmiş olan hiyerarşik yapıyı ilgili nesnelerle oluşturulması.
4. Geliştirilen uygulamaya ilişkin bir işlevselliği (veya bir senaryoyu) gerçekleştirirken,   
   JUnit kullanarak birim testlere dayalı geliştirilmesi.

Temel gereksinimleri belirtilen projede bu gereksinimlerin;

* İlk maddesi olan hiyerarşik yapı için kullandığımız tasarım deseni COMPOSITE tasarım desenidir. Bu deseni kullanmak için öncelikle firmanın hiyerarşik yapısında bulunan direktör ve memur çalışanlarının ortak özelliklerini ve metotlarını içeren bir EMPLOYEE adında abstract class ı oluşturuldu. Ardından bu EMPLOYEE sınıfından kalıtım alan DIRECTOR ve OFFICER sınıfları oluşturuldu. Ve bu iki sınıfımızdan girdi.text dosyasında verilen verilere göre sırayla dosya okuma yapılarak direktör veya memur olduğunu belirleyip gerekli nesneler oluşturulup ve bu firmada olan hiyerarşik yapının ağaç şeklinde oluşturulması için EmployeeComposite sınıfı oluşturuldu.
* Temel gereksinimlerin ikinci maddesinde bu hiyerarşik yapının dolaşılması ve ihtiyaç olunan koleksiyonun oluşturulmasında iterator dosyasında LinkedList sınıfı oluşturulmuştur.
* Ardında üçüncü maddede deneme sınıfı yazılıp main metodu içinde dosya okuma işlemi gerçekleştirilip nesneler oluşturulmuştur.
* Son olarak bu uygulama geliştirilirken test dosyasında oluşturulan oop1test sınıfında junit kullanılarak birim teste dayalı bir şekilde çalışmaya tabi tutulmuştur.

Şekil 1’ de de verilen projenin composite tasarım desenine uygun olarak çizilmiş UML sınıf diyagramında sınıflar arası ilişki gösterilmektedir.



**Şekil 1.** Proje için kullanılan composite tasarım deseninin UML sınıf diyagramı.

### PROGRAMIN ÇALIŞMASI VE ÖZELLİKLERİ

EMPLOYEE Sınıfı

public abstract class Employee {

    private *String* nameSurname;

    private *int* salary;

    public Employee(*String* *nameSurname*, *int* *salary*) {

*this*.nameSurname = nameSurname;

*this*.salary = salary;

    }

    public *String* getNameSurname() {

        return *this*.nameSurname;

    }

    public *String* getName() {

        return *this*.nameSurname.split(" ")[0].trim();

    }

    public *int* getCost() {

        return *this*.salary;

    }

    public *int* getSalary() {

        return *this*.salary;

    }

    public *void* add(*Employee* *employee*) {

        throw **new** UnsupportedOperationException();

    }

    public *void* remove(*Employee* *employee*) {

        throw **new** UnsupportedOperationException();

    }

    public *Employee* getEmployee(*int* *employeeIndex*) {

        throw **new** UnsupportedOperationException();

    }

    public *void* displayEmployeeInfo(*boolean* *isCost*) {

        throw **new** UnsupportedOperationException();

    }

}

EMPLOYEE sınıfı firmadaki hiyerarşik yapıda bulunan direktör ve memur çalışanların ortak olan isim ve maaş bilgilerini tutan sınıf olarak oluşturulmuştur.

OFFICER Sınıfı

public class Officer extends *Employee* {

    public Officer(*String* *nameSurname*, *int* *salary*) {

*super*(nameSurname, salary);

    }

    // memurların bilgilerini yazdıran metod

    public *void* displayEmployeeInfo(*boolean* *isCost*) {

        System.out.print("\t");

        System.out.print("M " + getNameSurname() + " " + getSalary() + "\n");

    }

}

Officer sınıfı Employee sınıfından kalıtım alarak bu sınıftan oluşturulan nesnelerin memur olduğunu gösteren sınıftır.

DIRECTOR Sınıfı

public class Director extends *Employee* {

    private *CustomLinkedList*<Employee> employees = **new** *CustomLinkedList*<Employee>();;

    public Director(*String* *nameSurname*, *int* *salary*) {

*super*(nameSurname, salary);

    }

    public *CustomLinkedList*<Employee> getEmployee() {

        return employees;

    }

    public *int* getCost() {

*int* \_total = 0;

        for (*int* i = 0; i < employees.size(); i++) {

            \_total += employees.get(i).getCost();

        }

        \_total += *super*.getCost();

        return \_total;

    }

    public *void* add(*Employee* *employee*) {

        employees.add(employee);

    }

    public *void* remove(*Employee* *employee*) {

        employee.remove(employee);

    }

    public *Employee* getEmployee(*int* *employeeIndex*) {

        return employees.get(employeeIndex);

    }

    public *CustomLinkedList*<Employee> getChildren() {

        return employees;

    }

    public *boolean* contains(*Employee* *emp*) {

        for (*int* i = 0; i < employees.size(); i++) {

            if (employees.get(i).getNameSurname().trim().equalsIgnoreCase(emp.getNameSurname().trim())) {

                return true;

            }

        }

        return false;

    }

    private *void* recursiveMetod(*CustomLinkedList*<Employee> *list*, *int* *depth*, *boolean* *isCost*) {

        for (*int* i = 0; i < list.size(); i++) {

            for (*int* j = 0; j < depth; j++) {

                System.out.print("\t");

            }

*Employee* element = list.get(i);

            System.out.print("D " + element.getNameSurname() + " "

                    + ((isCost && element.getClass() == Director.class) ? element.getCost() : element.getSalary())

                    + "\n");

            if (element.getClass() == Director.class) {

*Director* el2 = (Director) element;

                if (el2.getChildren().size() > 0) {

                    recursiveMetod(el2.getChildren(), depth + 1, isCost);

                }

            }

        }

    }

    public *void* displayEmployeeInfo(*boolean* *isCost*) {

*int* depth = 1;

        System.out.println(isCost ? "\nToplam maaşlar\n" : "\nHam maaşlar\n");

        System.out.println("D " + getNameSurname() + " " + (isCost ? getCost() : getSalary()));

        recursiveMetod(employees, depth, isCost);

    }

}

Director sınıfı Employee sınıfında kalıtım alarak bu sınıftan oluşturulan nesnelerin memurlara veya diğer direktörlerin yöneticisi konumunda olan direktörlerin oluşturulmasını sağlar.

EmployeeComposite Sınıfı

public class EmployeeComposite {

*Employee* employeeList;

    public EmployeeComposite(*Employee* *newEmployeeList*) {

        employeeList = newEmployeeList;

    }

    public *void* getEmployeeList() {

        employeeList.displayEmployeeInfo(false);

    }

    public *Employee* getEmployee() {

        return employeeList;

    }

    public static *EmployeeComposite* processTxtFile(*String* *txtList*) {

*Director* rootDirector = null;

*String*[] list = txtList.split("/");

*String*[] theRoot = null;

        for (*String* eleman : list) {

*String*[] parameters = eleman.split(",");

            if (parameters[3].trim().equalsIgnoreCase("root")) {

*Director* \_director = **new** Director(parameters[1], Integer.parseInt(parameters[2]));

                rootDirector = \_director;

                theRoot = parameters;

            }

            else {

                if (theRoot == null) {

                    theRoot = parameters;

                }

                if (parameters[3].trim().equalsIgnoreCase(theRoot[1].split(" ")[0].trim())) {

                    if (parameters[0].trim().equalsIgnoreCase("d")) {

*Director* \_director = **new** Director(parameters[1], Integer.parseInt(parameters[2]));

                        rootDirector.add(\_director);

                    } else {

*Officer* \_officer = **new** Officer(parameters[1], Integer.parseInt(parameters[2]));

                        rootDirector.add(\_officer);

                    }

                } else {

                    break;

                }

            }

        }

        getChildren(rootDirector, list);

        return **new** EmployeeComposite(rootDirector);

    }

    private static *void* getChildren(*Director* *rootDirector*, *String*[] *list*) {

*int* size = rootDirector.getChildren().size();

        for (*int* i = 0; i < size; i++) {

            if (rootDirector.getChildren().get(i).getClass() == Director.class) {

*Director* element = (Director) rootDirector.getChildren().get(i);

                for (*String* parameter : list) {

*String*[] parameters = parameter.split(",");

                    if (element.getName().equals(parameters[3].trim())) {

                        if (parameters[0].toLowerCase().equalsIgnoreCase("d")) {

*Director* director = **new** Director(parameters[1], Integer.parseInt(parameters[2]));

                            element.add(director);

                            if (!element.getChildren().isEmpty()) {

                                getChildren(element, list);

                            }

                        } else {

*Officer* officer = **new** Officer(parameters[1], Integer.parseInt(parameters[2]));

                            if (element.getChildren().size() == 0) {

                                element.add(officer);

                            } else {

                                if (!element.contains(officer)) {

                                    element.add(officer);

                                }

                            }

                        }

                    }

                }

            }

        }

    }

    public *void* getCost() {

        employeeList.displayEmployeeInfo(true);

    }

}

EmployeeComposite sınıfı verilen girdileri ilk önce root yani ana direktörü (yöneticiyi) bularak sonrasında altında çalışanları aramaya başlar.

* Eğer altında çalışan varsa ve bu direktörse onu listeye ekler ardından son bulunan direktörün altında çalışan olup olmadığını kontrol eder
  + Fakat çalışan bulamaz ise son kaldığı direktörün bir üst yönetici konumundaki direktöre dönüp başka çalışanları olup olmadığını kontrol eder.
* Eğer altında çalışan memur ise çalışan tüm memurları ekledikten sonra bir üst yöneticiye dönerek tarama işlemine devam eder.

LinkedList Interface

public interface LinkedList<T> {

    public *void* addNode(*T* *data*);

    public *void* add(*int* *index*, *T* *element*);

    public *boolean* isEmpty();

    public *boolean* add(*T* *data*);

    public *void* clear();

    public *T* get(*int* *index*);

    public *int* size();

}

Linked list için gerekli metotları implemente etmek için oluşturulan interface.

CustomLinkedList Sınıfı

public class CustomLinkedList<T> implements *LinkedList*<T> {

    class Node<T> {

        private *T* data;

        private *Node*<T> next;

        public Node(*T* *data*) {

*this*.data = data;

*this*.next = null;

        }

        public Node(*T* *data*, *Node*<T> *next*) {

*this*.data = data;

*this*.next = next;

        }

        public *T* getData() {

            return data;

        }

        public *void* setData(*T* *newData*) {

            data = newData;

        }

        public *Node*<T> getNext() {

            return next;

        }

        public *void* setNext(*Node*<T> *newNext*) {

            next = newNext;

        }

    }

    private *Node*<T> head = null;

    private *Node*<T> tail = null;

    private *int* currentSize;

    public CustomLinkedList() {

        head = null;

        tail = null;

        currentSize = 0;

    }

    public *void* addNode(*T* *data*) {

*Node*<T> newNode = **new** *Node*<T>(data);

        if (head == null) {

            head = newNode;

            tail = newNode;

        } else {

            tail.next = newNode;

            tail = newNode;

        }

    }

    @*Override*

    public *boolean* isEmpty() {

        return size() == 0;

    }

    @*Override*

    public *boolean* add(*T* *e*) {

        if (isEmpty()) {

            head = **new** *Node*<T>(e, null);

            tail = head;

        } else {

            tail.setNext(**new** *Node*<T>(e, null));

            tail = tail.getNext();

        }

        currentSize++;

        return true;

    }

    @*Override*

    public *void* clear() {

        head = null;

        tail = null;

        currentSize = 0;

    }

    @*Override*

    public *T* get(*int* *index*) {

        if (index < 0 || index >= size()) {

            throw **new** IndexOutOfBoundsException();

        }

        if (index < size() - 1) {

*Node*<T> current = head;

            for (*int* i = 0; i < index; i++) {

                current = current.getNext();

            }

            return current.getData();

        }

        return tail.getData();

    }

    @*Override*

    public *void* add(*int* *index*, *T* *element*) {

        if (index < 0 || index > size()) {

            throw **new** IndexOutOfBoundsException();

        }

        if (index == size()) {

            add(element);

            return;

        }

        if (index == 0) {

            head = **new** *Node*<T>(element, head);

        } else {

*Node*<T> current = head;

            for (*int* i = 0; i < index - 1; i++) {

                current = current.getNext();

            }

            current.setNext(**new** *Node*<T>(element, current.getNext()));

        }

        currentSize++;

    }

    @*Override*

    public *int* size() {

        return currentSize;

    }

}

CustomLinkedList sınıfı LinkedList interface’ sinden implement edilerek oluşturulmuş bir iterator desenidir. Oluşturulan hiyerarşik yapıyı dolaşmak için oluşturulmuştur. Ve bu iterator deseni çerçevesinde proje için gerekli olan koleksiyon oluşturuldu.

Deneme Sınıfı

public class Deneme {

  public static *void* main(*String*[] *args*) {

*String* txtList = "";

    try {

*File* myObj = **new** File("../oop1/src/main/java/com/mycompany/oop1/girdi.txt");

*Scanner* myReader = **new** Scanner(myObj);

        while (myReader.hasNextLine()) {

*String* data = myReader.nextLine();

            txtList += data + "/";

        }

        myReader.close();

    } catch (*FileNotFoundException* *e*) {

        System.out.println("Böyle bir dosya bulunamadı");

    }

*EmployeeComposite* emp = EmployeeComposite.processTxtFile(txtList);

    emp.getEmployeeList();

    emp.getCost();

  }

}

Deneme sınıfı main metodu altında oluşturulan yapının girdi.text dosyasını okuyarak ve gerekli dönüşümleri sağlayan metotlar çağırılarak projede istenilen şekilde çalışması sağlanmaktadır.

Opp1Test Sınıfı

public class oop1test {

    @*BeforeAll*

    public static *void* setUpClass() {

    }

    @*AfterAll*

    public static *void* tearDownClass() {

    }

    @*BeforeEach*

    public *void* setUp() {

    }

    @*AfterEach*

    public *void* tearDown() {

    }

    @*Test*

    public *void* displayEmployee() {

*Employee* employee = **new** Officer("", 80);

        assertNotNull(employee, "Employee nesnesi null !..");

    }

    @*Test*

    public *void* calculateSalary() {

*Director* director = **new** Director("Ugur Guclu", 2000);

        director.add(**new** Officer("Emre Kosar", 700));

        director.add(**new** Officer("Ahmet Egeli", 700));

        assertEquals(3400, director.getCost(), "Beklenilen maaş fiyatı sağlanamadı.");

    }

    @*Test*

    public *void* checkEmpty() {

*CustomLinkedList*<Employee> customLinkedList = **new** *CustomLinkedList*<Employee>();

        assertNotEquals(0,customLinkedList.size(), "Bu liste boş");

        customLinkedList.add(**new** Officer("", 80));

        assertEquals(0,customLinkedList.size(), "Bu liste boş değil");

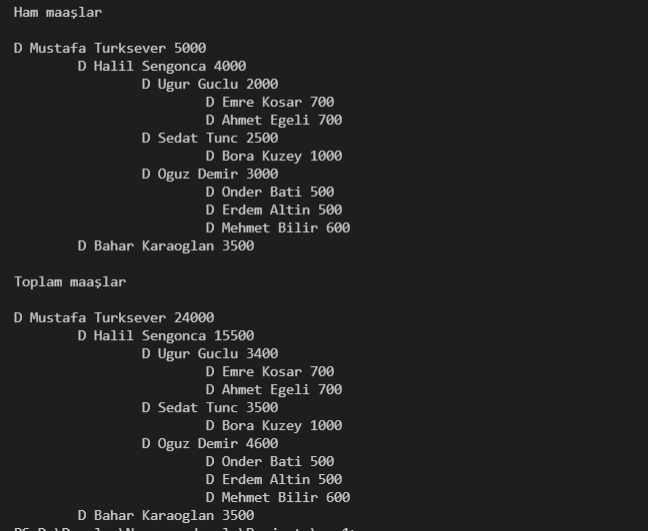
    }

}

Opp1test sınıfı junit kullanılarak bütün proje birim teste tabi tutulmuştur.

* DisplayEmployee metodu employee nesnelerinin null olup olmadığını kontrol eder.
* CalculateSalary metotdu girilen maaş tutarlarının beklenen maaş fiyatı olup olmadığını kontrol eder.
* checkEmpty metodu ise oluşturulan linkedListin boş olup olmadığını kontrol eder.

### PROGRAM ÇIKTISI



Proje kapsamında programın çalıştırılmasının sonucunda firmanın hiyerarşik yapısına uygun olarak composite tasarım kalıbı kullanılarak elde edilen program çıktısı üstteki ekran alıntısında gösterilmiştir.