Structural - Composite Design Pattern

Bir kurumda çalışanların hiyerarşik yapısını tasarlayabilmek için yani ağaç yapılarında sıklıkla kullanılan bir patterndir. Compose design pattern'in görevi, nesneleri bir ağaç yapısında birleştirip uygulamanın genelindeki parça bütün ilişkisini yeniden düzenleyip şekillendirmektir

- Component
    - Bileşikler için temel soyut tanımlamalardır.
    - Bileşik işlemi için nesnelerin arayüzünü oluşturur.
    - Tüm sınıfların arayüzündeki varsayılan davranışı gerçekleştirir.
    - Yavru bileşenlere ulaşmamızı ve onları kontrol etmemizi sağlamak için bir arayüz tanımlar.
- Leaf
    - Bileşik işleminde yavru nesneleri temsil eder.
    - Tüm bileşen metodları yapraklar tarafından tamamlanır.
- Composite
    - Yaprakları olan bileşenleri temsil eder.
    - Çocuklarını yönlendiren metodları gerçekler.
    - Genelde çocuklarını görevlendirerek bileşik metodlarını gerçekler.

Composite design pattern için bilmemiz gereken iki temel tanım vardır. Bunlar;

Component: Alt bileşenleri olmayan bileşen. (Çalışanlar örnek olarak)

Composite: Alt bileşenleri mevcut olan bileşendir. (Manager'lar örnek olarak)

- Component içerisinde tüm nesneler için ortak olan method(lar) bulunur.Abstract bir class ya da Interface olarak tasarlanabilir

```
interface Department{
   void printDepartmentName();
}
```

- Leaf diğer nesnelere bir referans içermez ve Component'i implemente eder.

```
//Leaf
class FinancialDepartment implements Department{
   private Integer id;
   private String name;

   //default constructor, getter and setter

   @Override
   public void printDepartmentName() {
      System.out.println("Department name : " + getClass().getSimpleName());
   }
}

//Leaf
class SalesDepartment implements Department{
   private Integer id;
   private String name;

   //default constructor, getter and setter

   @Override
   public void printDepartmentName() {
      System.out.println("Department name : " + getClass().getSimpleName());
   }
```

- Composite Leaf elemanlarına sahiptir.İçerisinde Component'lere ait listeyi tutar

```
//Composite
class HeadDepartment implements Department{
    private Integer id;
    private String name;
    List<Department> childDepartments;
    public HeadDepartment(Integer id, String name) {
        this.id = id;
        this.name = name;
        this.childDepartments = new ArrayList<>();
    }
    @Override
    public void printDepartmentName() {
        childDepartments.forEach(Department::printDepartmentName);
    }
    public void addDepartment(Department department){
        childDepartments.add(department);
    }
    public void removeDepartment(Department department){
        childDepartments.remove(department);
    }
}
```

- Main

```
public static void main(String[] args) {
    Department salesDepartment = new SalesDepartment(1,"Sales");
    salesDepartment.printDepartmentName();
    Department financialDepartment = new FinancialDepartment(1, "Financial");
    financialDepartment.printDepartmentName();

    HeadDepartment headDepartment = new HeadDepartment(1,"Head Department");
    headDepartment.addDepartment(salesDepartment);
    headDepartment.addDepartment(financialDepartment);
    headDepartment.printDepartmentName();
}
```

Head Department iki departmanı da altında tutmaktadır.Ast üst ilişkisi,daha doğru tabir ile tree yapısı