# Structural Pattern

## 1. Decorator Design Pattern

- Decorator Design Pattern, bir nesneye dinamik olarak işlevsellik ve davranış eklenmesini sağlar.
    - Aynı sınıftaki diğer nesnelerin davranışlarını etkilemez.
    - İşlevselliği kalıtım ile değil, composition ile sağlar.
- Decorator ile çalışma zamanında bir nesneye (sınıfa değil) işlevsellik eklenebilir.
    - Gereksinim ve seçime göre bu özelleştirilmiş işlevsellik, tek bir nesneye uygulanabilir.

### *1.1. Senaryo*

Örnek olarak `writeData()` ve `readData()` metotlarını içeren bir `DataSource` arayüzünü düşünelim.

- Daha sonra bu arayüzü implement eden farklı veri tiplerine sahip sınıflar olsun.
    - İlgili veri kaynakları, **daha spesifik** özelliklere sahiptir.
- Oluşturulacak veri kaynaklarını bir adım daha özelliştirmeye çalışalım.
    - Veriyi önce **şifreleyelim**, daha sonra da **sıkıştırmak** isteyelim.
    - İlgili projede daha **fazla ekstra özelliştirme** yapılması gerektiğini varsayalım.

### *1.2. Problem*

- Nesne davranışını değiştirmek için ilk akla gelen, nesnenin daha büyük bir sınıftan miras alması gerektiğidir.
    - Kalıtım **statiktir.** Bir nesnenin çalışma zamanında davranışı değiştirilemez.
    - Kalıtım ile alt sınıflar, yalnızca bir sınıftan miras alabilir.
        - Her programlama dili, çoklu kalıtımı desteklemez.

- Miras alınacak sınıf sayısının az olması, **nesnenin özelleştirilebilmesi** noktasında bir engeldir.

## *1.3. Çözüm*

- Miras yerine composition yapısı kullanılır.
- Pattern, decorator sınıfları oluşturur.
  - Sınıf metotlarının imzasını değiştirmeden ek işlevsellik sağlar.

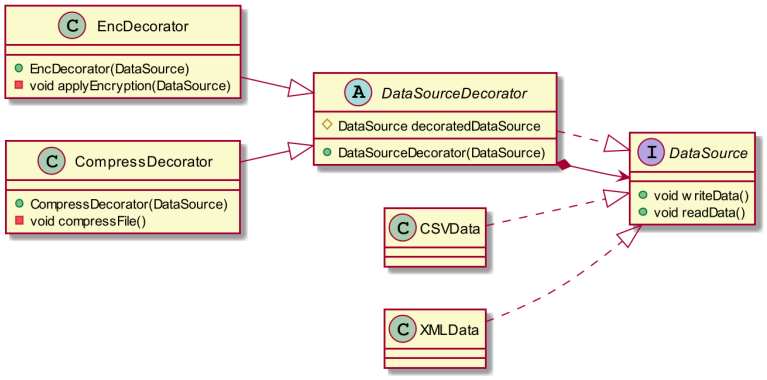


**Figure 1. Decorator tasarım kalıbı UML şeması**

**DataSource.java.**

```
public interface DataSource { ❶
 public void writeData();

 public void readData();
}
```

❶ İlgili arayüzün iki adet metodu bulunmakta.

**CSVData.java.**

```
public class CSVData implements DataSource { ❶
    // implemented methods...
}
```

❶ Veri kaynağı daha spesifik oldu, CSV ve XML tipindeki veri kaynaklarımız mevcut.

**DataSourceDecorator.java.**

```
public abstract class DataSourceDecorator implements DataSource { ❶
 protected DataSource decoratedDataSource;

 public DataSourceDecorator(DataSource decoratedDataSource) {
  super();
  this.decoratedDataSource = decoratedDataSource;
 }
}
```

❶ Veri şifreleme, veri sıkıştırma için daha spesifik özellikler için, DatSource'yi implement eden bir **Decorator** sınıfı oluşturuldu.

**EncDecorator.java.**

```
public class EncDecorator extends DataSourceDecorator {

 public EncDecorator(DataSource decoratedDataSource) {
  super(decoratedDataSource);
 }

 @Override
 public void writeData() {
  decoratedDataSource.writeData();
  applyEncryption(decoratedDataSource); ❶
 }

 private void applyEncryption(DataSource decoratedDataSource) {
  System.out.println("Data encrypted!");
 }

 // other implemented method.

}
```

❶ Görüldüğü üzere, `writeData()` metodunu icra etmekle birlikte, ekstra olarak veri şifreleme işlemleri de gerçekleştirilmekte.

**EncDecorator.java.**

```
public class CompressDecorator extends DataSourceDecorator {

 public CompressDecorator(DataSource decoratedDataSource) {
  super(decoratedDataSource);
 }
```

```
 @Override
 public void writeData() {
  decoratedDataSource.writeData();
  compressFile(); ❶

 }

 private void compressFile() {
  System.out.println("File successfully compressed!");
 }

 // other implemented method.

}
```

❶ Burada da, `writeData()` metodu ile birlikte veri sıkıştırma işlemi uygulanmıştır.

**DecoratorDemo.java.**

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

 System.out.println("Creating decorated (encrypt and compress) source
 CSV file...");
 DataSource decCsv = new EncDecorator(new CompressDecorator(new
 CSVData())); ❶

 decCsv.writeData();
 }
```

Kodun çıktısı aşağıda verilmiştir.

```
Creating decorated (encrypt and compress) source CSV file...
Writing used CSV data source.
File successfully compressed!
Data encrypted!
```

- Görüldüğü üzere, `DatSource`, `CSVData` ve `XMLData` çekirdek sınıflarında bir değişiklikte bulunulmadı.
- Decorator sınıfları oluşturularak, bu üç sınıfın **davranışları özelleştirildi.**