Encapsulation (Kapsülleme)

Sarmalama ilkesi, bir sınıfa ait değişkenlerin veya niteliklerin ancak o sınıfa ait metotlar tarafından değiştirilebilmesi ve okunabilmesi ilkesidir. Bu ilke sayesinde nesnelerde oluşacak anlamsızlıkların önüne geçilebilir.

Ayrıca değişkenlere sınıfların dışından erişim olmaması ve bir sınıf içindeki değişkenlerin nasıl ve ne kadar olacağının da başka kodlardan saklanmış olması anlamına gelir. Böylelikle biz değişkenlerimizi sarmalayarak istenmeyen durumlardan korunacak bir filtre haline dönüştürebiliriz. Bunu bir örnek ile anlamaya çalışalım.

Encapsuliaton Örneği

Kitap adında bir sınıfımız olsun ve bu sınıfımıza ait 3 adet değişkenimiz olsun bunlar ; kitapAdi, sayfaSayisi ve yazar. Bu değişkenlerin erişim belirleyicileri public olsun ve her sınıftan erişilsin. Kitap sınıfından oluşturacağımız bir nesne bu niteliklerin hepsini taşısın. Bu yüzden oluşturacağımız Constructor (kurucu) metodunu bu şekilde oluşturalım.

```
public class Kitap {
    public int sayfaSayisi;
    public String kitapAdi, yazar;
    Kitap(String kitapAdi, int sayfaSayisi, String yazar) {
        this.kitapAdi = kitapAdi;
        this.sayfaSayisi = sayfaSayisi;
        this.yazar = yazar;
    }
}
```

Görüldüğü üzere normal bir sınıfımız ve kurucu metodumuz var. Kitap sınıfından bir nesne oluşturalım.

```
Kitap book = new Kitap("Harry Potter", 500, "JK Rowling");
```

Kitap sınıfından **book** adlı bir nesne oluşturduk ve bu nesnemizin niteliklerini belirttik. Peki biz bu kurucu metotta sayfa sayısını negatif bir değer girseydik ne olurdu ? Hiç bir kitabın sayfa sayısı negatif bir değer olamayacağı için, nesnemizde bir **anlamsızlık** olacaktı. Biz bu sorunu **constructor** (kurucu) metotumuza yazacağımız bir **if** kontrolü ile çözebiliriz.

```
public class Kitap {
    public int sayfaSayisi;
    public String kitapAdi, yazar;
    Kitap(String kitapAdi, int sayfaSayisi, String yazar) {
        this.kitapAdi = kitapAdi;
        this.yazar = yazar;
        if (sayfaSayisi < 1) {
            this.sayfaSayisi = 10;
        } else {
            this.sayfaSayisi = sayfaSayisi;
        }
    }
}
```

Constructor metodu görüldüğü gibi modifiye ettik ve nesne oluşturulurken anlamız verilerin olmasını engelledik. Ama sorunlarımız hala bitmedi , biz nesneye ait niteliklere hala dışarıdan erişebiliyoruz ve book.sayfaSayisi = -10 dersek , nesneye ait sayfa sayısını yine anlamsızlaştırmış oluruz. Bu sorunu çözmek için sınıfa ait değişkenlere dışarıdan erişimi kapatmamız gerekir ve oluşturduğumuz değişkenlerin erişim belirleyicilerini **(Access Modifiers)** değiştirmemiz gerekli. Tüm **public'leri private** olarak değiştiriyoruz.

Sınıfımızın son hali

```
public class Kitap {
    private int sayfaSayisi;
    private String kitapAdi, yazar;
    Kitap(String kitapAdi, int sayfaSayisi, String yazar) {
        this.kitapAdi = kitapAdi;
        this.yazar = yazar;
        if (sayfaSayisi < 1) {
            this.sayfaSayisi = 10;
        } else {
            this.sayfaSayisi = sayfaSayisi;
        }
    }
}
```

Sınıfa ait değişkenlerimin izinlerini **private** yaparak bu sorunu çözdük ama, biz book nesnesine ait değişkenlere erişimi tamamen kısıtladık. Yani biz oluşturduğumuz nesneye ait sayfa sayısını ekrana bastıramayız çünkü değişken **private** olarak tanımlandı. Ya da sayfa sayısı yanlış girilmiş bir nesneyi daha sonrasında düzenleyemeyiz. Bu sorunu çözmek için sınıfa ait değişkenlerimizi **sarmalayarak**, sınıf içerisinde ki **metotlar** yardımı ile değişkenlerimizi **koruma** altına alıyoruz ve kullanıma sunuyoruz. Bu metotlara sonrasında ismini çok duyacağımız **Getter ve Setter** metotları diyoruz.

Getter ve Setter Metotları

Getter

Sınıfımıza ait **private** değişkenler mevcut. Bu değişkenlere dışarıdan erişebilmek için **her bir değişkenimiz** için **Getter** metodu yazmalıyız. Nesneye ait bu metot çağrıldığında geriye bir değer döndürmeli ve bu değer bizim istediğimiz **private** değişken olmalı. **sayfaSayisi** değişkeni için **getter metodu** tanımlayalım,

```
public class Kitap {
        private int sayfaSayisi;
        private String kitapAdi, yazar;


        Kitap(String kitapAdi, int sayfaSayisi, String yazar) {
                this.kitapAdi = kitapAdi;
                this.yazar = yazar;
                if (sayfaSayisi < 1) {
                        this.sayfaSayisi = 10;
```

```
                } else {
                        this.sayfaSayisi = sayfaSayisi;
                }
        }

        public int getSayfaSayisi() {
                return this.sayfaSayisi;
        }
}
```

Görüldüğü gibi basit bir metot yardımı ile sınıfa ait **private** değişkenimize ulaşabildik. Burada dikkat edilmesi gereken noktalar **getter** metotları **geri dönüşü** olan metot tipindedir ve isimlendirilmesi ise **get** ile başlayıp sonra değişken ismi yazılmalıdır. İsimlendirmeyi bu şekilde yapmasak da çalışacaktır lakin kodun okunabilirliği adına bu kurala uyulması **gerekir.**

Setter

Biz **getter** metodu ile **private** olan değişkenimize ulaştık.Peki bu değişkenin değerini değiştirmek istediğimizde ne yapmalıyız ? Bir sınıfa ait **private** bir değişkenin değerini değiştirmek için, **setter** metodu yazılmalıdır. sayfaSayisi değişkeni için setter metodu yazalım.

```
public class Kitap {
        private int sayfaSayisi;
        private String kitapAdi, yazar;


        Kitap(String kitapAdi, int sayfaSayisi, String yazar) {
                this.kitapAdi = kitapAdi;
this.yazar = yazar;
                if (sayfaSayisi < 1) {
                        this.sayfaSayisi = 10;
                } else {
                        this.sayfaSayisi = sayfaSayisi;
                }
        }

        public int getSayfaSayisi() {
                return this.sayfaSayisi;
        }

        public void setSayfaSayisi(int sayfaSayisi) {
                this.sayfaSayisi = sayfaSayisi;
        }
}
```

Görüldüğü üzere **setter** metodu sadece değiştirme işlemi yapacağı için **void** olarak tanımlandı ve bir adet parametre aldı. Bu parametre bizim yeni değerimi taşıyor olup, sınıfa ait değişkene aktarılmıştır. Ama burada hala bir sorun söz konusudur, bizler **setter** metodunu kullanarak sayfa sayısını negatif girebiliriz. Bu sorunu aşmak için **constructor** (kurucu) metotta yaptığımız gibi bir **if** koşulu ile bu sorunu çözebiliriz.

```
public class Kitap {
        private int sayfaSayisi;
```

```
        private String kitapAdi, yazar;


        Kitap(String kitapAdi, int sayfaSayisi, String yazar) {
                this.kitapAdi = kitapAdi;
                this.yazar = yazar;
                if (sayfaSayisi < 1) {
                        this.sayfaSayisi = 10;
                } else {
                        this.sayfaSayisi = sayfaSayisi;
                }
        }


        public int getSayfaSayisi() {
                return this.sayfaSayisi;
        }


        public void setSayfaSayisi(int sayfaSayisi) {
                if (sayfaSayisi < 1) {
                        this.sayfaSayisi = 10;
                } else {
                        this.sayfaSayisi = sayfaSayisi;
                }
        }
}
```

Setter metodunu modifiye ederek nesnemiz için anlamsız olan durumu ortadan kaldırmış olduk. **Setter** metodunun genel özellikleri ise , geriye bir değer döndürmeyen metot olması ve isimlendirme yaparken başlangıç olarak set yazıp sonrasında değişken ismini yazmaktır.

Bu örnekteki **sayfaSayisi** değişkenini koruma ve anlamsızlaşmasını önlemek için **Nesne Yönelimli Programlamanın** ilkesi olan **Encapsulation** (Sarmalama) ilkesinden yararlandık. Bir sınıfa ait değişkenlerimizi **Getter** ve **Setter** metotları yardımı ile sarmaladık ve istenilen şartlara göre oluşmasını sağladık.