Java

Tugberk Kocatekin
T.C. Istanbul Arel University
Spring 2023

Yeni sınıf

- Eclipse üzerinde src'yi bulup sağ tıklayarak yeni Class oluştur diyoruz.
- Bu aşamada istersek `main` ekle diyebiliriz. Demeyelim şimdilik.
- Yeni bir sınıf oluşturalım.
  - Bunu oluşturduğumuzda `public class` yazacak.
  - Bu, o sınıfa diğer sınıflar tarafından erişilebileceğini gösteriyor.

Metod

- Şimdi yeni yarattığımız sınıf içerisinde bir metod yaratalım.
  - Metod dediğimiz şey aslında bir **fonksiyon**.
- Yarattığımız sınıfın adı Elma olsun. Onun altına da bir metod yaratalım.

```
public class Elma {
    public void selamVer() {
        System.out.println("elma");
    }
}
```

- Bu kodu çalıştırırsak ne olur?

- Bu kodu çalıştırdığımızda hiçbir şey göremeyiz, çünkü **Main** yok.
- O zaman bu **sınıfı** biz Main içerisinde çağıralım?
- Ana programımız neyse, ona gidip onun Main metodu içerisinde çağıracağız.
  - Bu metodu çağırmak için önce o sınıfa ait bir obje yaratmamız gerekiyor. Onu şu şekilde yapıyoruz: `Elma yeniElma = new Elma();`
  - Ardından, `yeniObje` yazıp noktaya bastığımızda, gelen metodlar arasında diğer sınıftaki metodu da görebiliriz ve onu çağırabiliriz!

- Burada dikkat edilmesi gereken şey isimler.
- Çoğu zaman sınıf adı ile obje adını aynı kullanırlar, bu kafa karıştırabilir. Ona uygun bir şey seçmek gerekir.
  - Hatırlarsanız sınıf isimleri büyük harfle başlamalı demiştik.
  - Yukarıdaki Elma olsa, o zaman şöyle yapabiliriz:
    - `Elma elma = new Elma();`

- Burada mümkün mertebe bu tarz sınıfları birbirinden ayırıyoruz.
- Bu sayede herkes başka fonksiyonalite üzerinde rahatlıkla çalışabiliyor. Bunları kullanacağımız zaman da gerekli objeleri yaratarak çağırıyoruz.
  - Java kullanırken bunu çok yapacağız.
  - Reuse düşünerek program yazmalıyız.
- Aynı metod isimlerini farklı sınıflarda kullanabiliriz, çünkü obje tanımlarken zaten sınıfa göre tanımlıyoruz. O yüzden çağırırken sorun olmaz.

Örnekler

- Elma, Armut, Ayva isminde sınıflar yaratalım. Bunların hepsini Main içerisinde oluşturup hepsine aynı komutu verelim.

Parametreli metodlar

- Armut sınıfına gidip yeni bir metod yaratalım.

```
public void yeniSelam(String isim) {
    System.out.println("Selam " + isim);
}
```

- Main içerisinde zaten tanımlamıştık, o tanımladığımız objeden çağıralım.
- `armut.yeniSelam("ahmet")`

- Peki, yeniSelam yerine, bir önceden tanımladığımız `selamVer` metoduna ekleme yapsak, aynısını altına ancak bu sefer parametreler yazsak nasıl olur?
- Yani:

```
public void selamVer() {
    System.out.println("ben armut");
}

public void selamVer(String isim){
    System.out.println("ben armut " + isim);
}
```

- İkisi de çalışıyor!
    - Method Overloading

Public ve Private

- OOP'nın avantajlarından bir tanesi **Encapsulation**.
  - Buna aynı zamanda *bilgi saklama* da deniyor (information hiding).
- Çünkü aslında birçok şeyi bir arada bir demet gibi tutabiliyoruz. Bu şekilde, bir sınıfa ait bir obje yalnızca o sınıf tarafından erişilebiliyor. Dışarıdan bir değişime uğramıyor.
  - Dolayısıyla *encapsulation* sayesinde onu dış değişimlerden koruyoruz.
- O yüzden, Java programlama dilini kullanıyorsak eğer, bir Class yarattığımızda, içerisinde tanımladığımız _attribute_ları **private** olarak tanımlamalıyız.
- Böyle yaptığımızda, ona ancak *getter* ve *setter* fonksiyonları erişebilir.
  - Public yaparsak herkes erişir. Bunu istemiyoruz.
- Örneğin Python programlama dilinde tam olarak `private` mantığı yok, o yüzden tam anlamıyla encapsulation yok.

Private

- Bu sefer, public yerine **private** kullanacağız ve değişkenler tanımlayacağız.
  - Public dediğimizde diğer sınıflar da kullanabiliyordu.
- Ancak private dediğimizde, başka sınıflar bu değişkeni kullanamaz.
  - Yalnızca o sınıfın metodları ulaşabilir.
- Örneğin, `Armut` sınıfı içerisinde iki tane değişken oluşturalım.
  - Biri `public String ad;` diğeri de `private String isim;` olsun.
  - Main içerisinde bir obje tanımlayalım: `Armut armut = new Armut();`
  - ad ve isim parametrelerine `armut.ad` şeklinde eriştiğimizde birine izin verecek, diğerine vermeyecek.

Setters ve getters

- Private olanlara başka sınıflardan erişemiyoruz. Ancak kendi sınıfındaki metodlar erişebiliyor demiştik.
- O yüzden bu **private** olanları değiştirebilmek için bunlara metodlar ile erişeceğiz.
  - Bu sefer diğer sınıftan da değişkene **direkt** olarak erişmek yerine bu metodları kullanarak erişeceğiz.
  - private olan değişkenleri değiştirmek için kullandığımız metodlara **setter** diyoruz.
    - Okumak için kullandıklarımıza da **getter** diyeceğiz.

Setter ve getter

- armut sınıfı içerisinde önce setter oluşturalım.

```
private String isim;
public void setName(String name) {
    isim = name;
}
```

- Şimdi, bu metodu Main sınıfından çağırırsak eğer, parametre olarak verdiğimiz neyse, isim ona eşit olacak.
- Getter da ekleyelim (parametre yok ve return a dikkat)

```
public String getName() {
    return isim;
}
```

Örnek

- Yepyeni sınıf yaratalım. Adı Renk olsun.
- Bu sınıfta renkAdi diye bir private değişken olsun.
- Main class'da kullanıcıdan input alalım ve bunu renkAdi değeri yapalım, ekrana yazdıralım.
    - Yazdırmak için ayrı bir sınıf yaratabiliriz!

Constructor

- Sınıflar yarattık, ardından objeler yaptık. Ancak biz artık şunu istiyoruz:
  - Bir obje yarattığımızda, o objeye otomatik olarak bazı değişkenler, metodlar atansın.
  - Bunun için **constructor** kullanacağız.
- Örneğin bir önceki örneği düşünürsek, bir isim aldık ve `konus()` metodunu çağırdık.
  - Peki ya isim almadan çağırırsak?
    - **null** dönecek.

- Peki aslında bu objeyi yarattığımızda bazı değişkenleri otomatik olarak yaratmasını sağlayamaz mıyız?
- Yine metod yaratacağız aslında, tek fark metod adı **sınıf adı** ile aynı olacak.

```
public Renk(String name) {
    renkAdi = name;
}
```

- Hatırlarsanız, biz bir obje yaratırken şunu yapıyorduk:
  - `Renk renk = new Renk();` şimdi ise burada en sonda bir parametre ekleyeceğiz:
  - `Renk renk = new Renk("sari");`

Örnek

- Main içerisinde yine tanımlama yapalım.

```
Renk renkObje = new Renk("sari");
renkObje.konus();
```

- Bu çalışıyor, ancak bu sefer içini boş vermek isterseniz, hata verecek.
- Constructor'ı içinde argümanlarla tanımladığımız için, boş constructor yok.
  - Buna dikkat etmek gerekiyor.
  - Eğer boş tanımlanabilsin de istiyorsanız iki tane constructor yaratabilirsiniz.
    - Overload yeteneği var unutmayalım.

- Bu şekilde birden fazla obje yarattığımızda, her obje için yeni değişkenler kullanılacak.
    - Ve bu objeler birbirlerinin değişkenlerinden haberdar değil. Yani değişkenler, sadece kendi objeleri tarafından biliniyor.

Random

- `import java.util.Random;`
- Random sınıfından yeni bir obje yaratarak zar atalım.

```
Random zar = new Random();
int sayi;

for(int i=0;i<10;i++) {
    sayi = zar.nextInt(6);
    System.out.println(sayi);
}
```

- 6 dedik ancak 0,1,2,3,4,5 geldi. Bu durumda bunun başına `1+` ekleyeceğiz. Eğer sıfır gelirse 1 ekleyecek, 5 gelirse de 6 ya çevirecek. Bunun gibi düşünülebilir Random fonksiyonunun kullanımı.

Örnek

- nextInt() dedik, çünkü dönüş olarak bir **sayı** bekliyoruz.
- Örneğin bir Dizi içerisindeki elemanlar içerisinden rastgele seçim yapmak istersek nasıl yapacağız?

```
String[] dizi = {"Sarı","Yeşil","Lacivert","Siyah"};
Random rand = new Random();
int sayi;
sayi = rand.nextInt(dizi.length);
System.out.println(dizi[sayi]);
```

- Peki ya böyle bir fonksiyonumuz olsun istersek?
- Yani bir methoda diziyi verelim, bize rastgele bir eleman versin.

- O zaman yeni bir metod yaratırız. Return değeri String olacak.

```
public static String randomEleman(String[] dizi) {
    Random rnd = new Random();
    int sayi;
    sayi = rnd.nextInt(dizi.length);
    return dizi[sayi];
}
```

- Bunu ekrana yazdırmak için de fonksiyonu Main içinde çağırdıktan sonra bir değişkene atayacağız ve yazdıracağız.

Örnek

- Aynı zamanda 10 elemanlı bir dizi oluşturup, 10 tane random zar atışı yaptıktan sonra bunları diziye yazan bir program da yapabiliriz.

```java
int[] atışlar = new int[10];
Random rnd = new Random();
for(int i=0;i<10;i++) {
    atışlar[i] = 1 + rnd.nextInt(6);
}

for(int i=0;i<10;i++) {
    System.out.print(atışlar[i]);
}
```

Ödev

Ödev olarak bu zarların çiftli atılmış halini saklamaya çalışın. Tavladaki gibi, her atış 1 tane değil 2 tane random değer üretsin. Bunları dizi içine koyalım.