Class (Sınıf)

Object Oriented Programming(OOP) yani Nesne Tabanlı Programlama'nın temel yapısıdır. Class'lar, günlük hayatımızda Nesne olarak tanımladığımız kavramları, programlama yapılarına modelleme yapabilmemiz için kullandığımız yapılardır. Bu yapı sayesinde somut ve ya soyut kavramların özellikleri(Properties ya da Field) ve yaptıkları işleri / eylemleri (OOP'de Metod adıyla geçmektedir Fonksiyon kelimesi bu programalama yaklaşımında kullanılmamaktadır ama yaptığı iş bakımından fark yoktur.) programalama dünyasına aktarımı yapılmış olur.

```java
/**
 *
 * @author cemdirman
 */
public class Araba {
    String marka;
    String model;
    String renk;
    int beygirGücü;
    int üretimYili;
    int hiz;
    boolean sagSinyal = false;
    boolean solSinyal = false;
    void hizlan(){
        hiz += 10;
    }

    void dur(){
        hiz = 0;
    }

    void sagaSinyalVer(){
        sagSinyal = true;
    }

}
```

Instance (Örnek)

Bir Class'ın yani sınıfın örneğine bu Class'ın instance'ı denilir. Yani instance'ını oluşturacağımız Class'dan 'new' keyword'u ile kendi tipinde

```
/**
 *
 * @author cemdirman
 */
public class Test {
    public static void main(String[] args) {
        Araba a1 = new Araba();
        Araba bmw = new Araba();
    }

}
```

oluşturduğumuz bir değişken elde etmiş oluruz.

Local Değişken

Local ve Global olmak üzere iki farklı değişken tanımlama yöntemi bulunmaktadır. Local değişkenler sadece belirli scope(kapsam) içerisinde tanımlanmış ve sadece tanımlandığı scope içerisinde görülebilir(kullanılabilir).

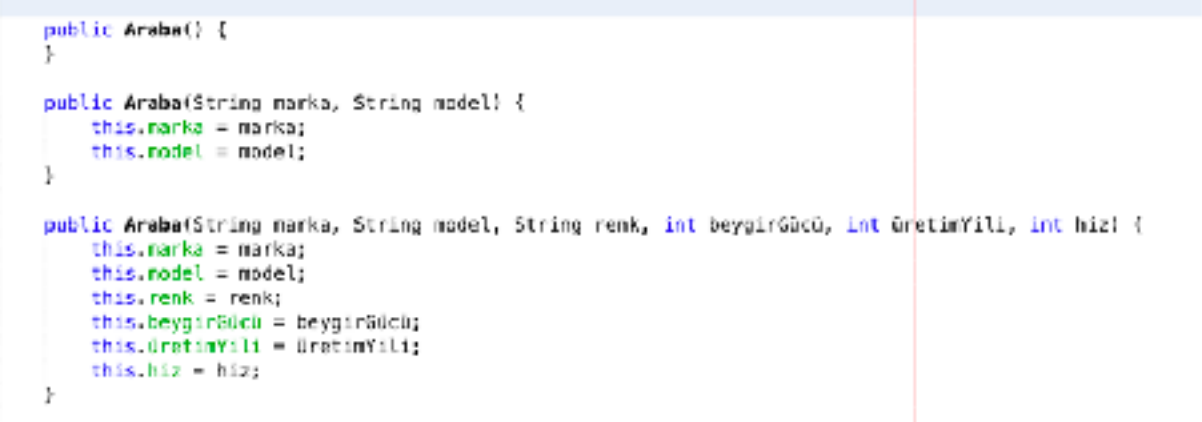
```
void hizlan(){
    int artisHizi = 10;
    hiz += artisHizi;
}
```

Not: cope, süslü parantezler arasında kalan alan.

Constructor Method(Yapıcı Metod)

Bir Class'dan 'new' keyword'u ile instance'ını oluşturulurken çağırdığımız metoddur. Bu method sayesinde Class'ın özelliklerine ilk değer atama işlemi yapılabilir. Her Class default olarak boş constructor'a sahiptir fakat parametre alan bir constructor oluşturduğumuzda bu default olarak bulunan metod kaybolur. Yani eğer boş constructor ile nesne oluşturulacak ise ve de parametre alan bir constructor oluşturulduysa, boş constructor oluşturmak gerekir.

Aşağıdaki görselde sırasıyla; boş, iki ve altı parametre alan üç farklı



constructor metod oluşturulmuştur.

Burda ise yukarıda oluşturulan constructor'lara göre üç farklı Araba Nesne'sinin instance'ı oluşturulmuştur.

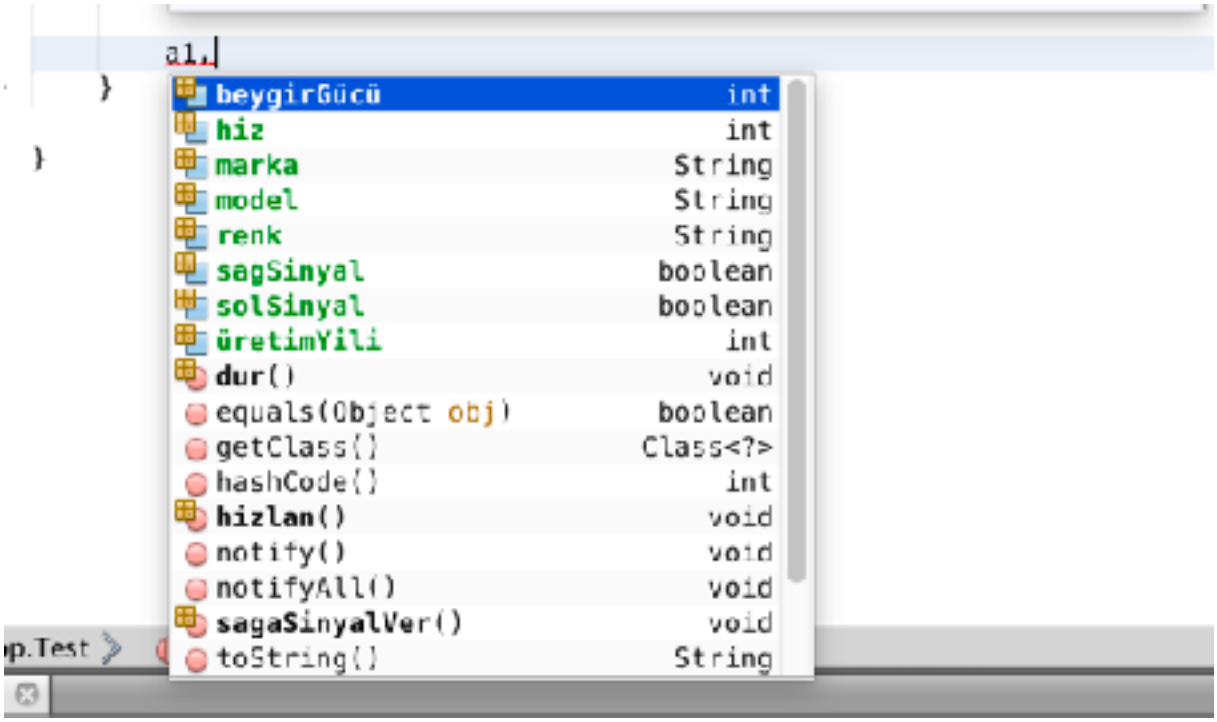
```
 *
 * @author cemdirman
 */
public class Test {
    public static void main(String[] args) {
        Araba a1 = new Araba();
        Araba a2 = new Araba("BMW","520D");
        Araba a3 = new Araba("Audi","A5","Siyah",500,2018,260);
    }

}
```

Nesne Üzerinden Method Çağırma

Nesne'nin sahip olduğu metodlara *dot notation(nokta notasyonu)* ile oluşturulan değişken ismi üzerinden metod çağrılır.

Dikkat edilmesi gerekenler noktalar;

- Methodların dönüş tipleri
- Methodların aldığı parametreler



Get / Set Metodları

Nesnelerin özelliklerine direkt olarak erişilebileceği gibi get/set metodlarıyla da ulaşılabilir. Buradaki amaç özelliklere direkt olarak erişim yerine bu metodlar üzerinden erişim sağlanarak set ile özelliğe değer atama, get ile değeri alma işlemi kontrollü hale getirilir.

- Get, nesnenin özelliğine almak için kullanılır, metodun dönüş tipi özelliğin dönüş tipiyle aynıdır.
- Set, nesnenin özelliğine değer atamak için kullanılır, metodun dönüş tipi void'tir ve özelliğin tipiyle aynı tipde bir parametre alır. Aynı zamanda özellikle verilen değeri kontrol etmek amacıyla kullanılır. Örneğin bir depodaki bir malın adeti eksi değer olamayacağından, değer atama sırasında kontrol yapılarak yanlış değerlerin atanması engellenmiş olunur.

Static Keyword'ü

Static keyword'u Class memory'e(RAM) yüklendiğinde oluşur ve o nesne yaşadığı sürece var olur. Bir class'dan üretilen bütün instance'larda tek ve aynı değere sahiptir. Yani bu özellik class'ın nesnesi yerine class'a aittir. Oluşturulan bütün nesneler tek bir değere ulaşmış olurlar. Değişkenler ya da metodlar static olabilir.

- Statik değişken, tüm nesnelerin ortak özelliğini (her nesne için benzersiz olmayan) ifade etmek için kullanılabilir. çalışanların şirket adı, öğrenci okul adı gibi.
- Static metodları kullanmak için, class'dan nesne üretmemize gerek yoktur. Bunun mantığı ise belirtildiği gibi static değişken ya da metodlar üretilen nesneye ait değil, class'a aittir.

```
 * @author cemdirman
 */
public class Araba {
    String marka;
    String model;
    String renk;
    int beygirGücü;
    int üretimYili;
    boolean sagSinyal = false;
    boolean solSinyal = false;

    static int hiz;
```

```
 *
 * @author cendirman
 */
public class Test {
    public static void main(String[] args) {
        Araba a1 = new Araba();
        Araba a2 = new Araba("BMW","520D");
        Araba a3 = new Araba("Audi","A5","Siyah",580,2018,260);

        System.out.println("Hiz:" + Araba.hiz);
        Araba.hiz = 50;
        a1.hizlan();//hiz değişkenini 10 arttırır
        System.out.println("Hiz:" + Araba.hiz);
    }

}
```

```
run:
Hiz:260
Hiz:60
BUILD SUCCESSFUL (total time: 0 seconds)
```

Encapsulation

OOP'in en önemli kavramlarından biridir. Kelime anlamı olarak kapsülleme, yani saklamak olarak düşünülebilir.Buradaki amaç sadece Class'ın özelliklerini ya da metodlarını, ilgili birimlerin görebilmesidir. Bu görülebilmeyi ise şu keyword'ler sayesinde sağlıyoruz;

- public: bütün class'lara açık.
- protected: sadece içinde bulunduğu paketteki class ulaşabilir.
- private: sadece kendi class'ı üzerinden oluşturulan nesneler ulaşabilir.

Access Levels

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Tablo: Access Modifiers

Bu keywordler ile kapsüllediğimiz özellikleri ise daha önce belirtilen get/ set metodları sayesinde kullanırız.

UYGULAMA

1- Öğrenci class'ı oluşturulup isim, soyisim, yas, okul numarası ve sınıf özelliklerini encapsulate edilir.(özellikler private olmalı ve bu özelliklere get/set metodları üzerinden erişim olmalıdır.)

2- Farklı parametreler alan constructor tanımlanır.

```
/*
 * @author cendirman
 */
public class Ogrenci {
    private String isim;
    private String soyisim;
    private int yas;
    private int okulNo;
    private String sinif;

    public Ogrenci() {
    }

    public Ogrenci(String isim, String soyisim, int yas) {
        this.isim = isim;
        this.soyisim = soyisim;
        this.yas = yas;
    }

    public Ogrenci(String isim, String soyisim, int yas, int okulNo, String sinif) {
        this.isim = isim;
        this.soyisim = soyisim;
        this.yas = yas;
        this.okulNo = okulNo;
        this.sinif = sinif;
    }

    public String getIsim() {
        return isim;
    }

    public void setIsim(String isim) {
        this.isim = isim;
    }

    public String getSoyisim() {
        return soyisim;
    }

    public void setSoyisim(String soyisim) {
        this.soyisim = soyisim;
    }

    public int getYas() {
        return yas;
    }
```

3- Oluşturulan Öğrenci class'ından üç farklı constructor ile nesne üretilir.

```
public static void main(String[] args) {
    Ogrenci ogrenci1 = new Ogrenci();
    ogrenci1.setIsim("Cem");
    String soyisim = "Dırman";
    ogrenci1.setSoyisim(soyisim);
    ogrenci1.setOkulNo(12345);

    Ogrenci ogrenci2 = new Ogrenci("Ömer", "Koçbil", 23);
    String isim = ogrenci2.getIsim();

    Ogrenci ogrenci3 = new Ogrenci("Ahmet", "Demirşen", 23, 1, "12-B");
}
```