Polymorphism, Encapsulation, super ( ) methodu

Dr.Öğr.Üyesi Çiğdem BAKIR

Encapsulation

Bir nesnenin belirli özellik ve metodlarının saklanması, erişiminin kısıtlanmasıdır. Erişime kısıtladığımız bu özellik ve metodları zaten kullanıcının görmemesi veya kullanmaması gerekir.

Kontrolsüz veri girişini ve kötü amaçlı kullanımı önler.

Erişim Belirleyicileri

- Public: Öğenin kod içerisinde herhangi bir yerden erişilebilir olduğunu belirtir. Herhangi bir kısıtlama vs. bulunmamaktadır.
- Private: Öğenin sadece tanımladığı sınıf içerisinde erişilebilir olduğunu belirtir.
- Protected: Öğenin tanımlandığı sınıf içerisinde ve o sınıftan türeyen sınıflardan erişilebilir olduğunu belirtir.

```
public class Televizyon
{
    private string _marka;
    private int _pixel;
    private int _fiyat;
}
```

```
public int FiyatGoster()
{
    return this._fiyat;
}

public void FiyatGir(int price)
{
    if (price > 0)
    {
        this._fiyat = price;
    }
}
```

İlk örnekte görebileceğiniz gibi _fiyat değişkeni private olarak tanımlandığı için direk olarak erişilemez. Bunun yerine public erişim belirtecine sahip **getter** ve **setter** metotları aracılığı ile değiştirilip kullanılabilir. Bu da bizim veri güncellerken veya kullanırken yaşanabilecek karışıklıkların önüne geçmemizi sağlar.

```
public class Televizyon
{
    public string Marka { get; set; }
    public int Pixel { get; set; }
    public int Fiyat { get; set; }
}
```

Bu örnekte görüldüğü üzere tüm özellikler erişilebilir (get) ve set edilebilir. (Okuma ve yazma)

```
public class Televizyon
{
    private string _marka;
    private int _pixel;
    private int _fiyat;

    public string Marka { get; set; }
    public int Pixel { get; set; }

    public int Fiyat
    {
        get
        {
            return _fiyat; //FiyatGoster()
        }
        set
        {
            if (value > 0)
            {
                this._fiyat = value; //FiyatGir()
            }
        }
    }
}
```

Bu örnekte görebileceğiniz gibi _marka, _pixel, _fiyat değişkeni private olarak tanımlandığı için direk olarak erişilemez. Bunun yerine public erişim belirtecine sahip **getter** ve **setter** metotları aracılığı ile değiştirilip kullanılabilir. Bu da bizim veri güncellerken veya kullanırken yaşanabilecek karışıklıkların önüne geçmemizi sağlar.

super ( ) methodu

- Bir alt-sınıf ne zaman üst-sınıfına erişmek isterse super anahtar sözcüğünü kullanabilir. super ‘in kullanımı iki türlü olur.
- Birincisi, üst-sınıfa ait nesne yaratmak içindir.
- İkincisi, üst-sınıfın öğelerine erişmek içindir.

Üst-sınıfa ait nesne yaratma:

Bir alt-sınıf, super() metodunu kullanarak, üst sınıfının bir nesnesini yaratabilir ve onun değişkenlerine değer atayabilir. **(Bir üst sınıftaki default constructorı çağırır.)**

- AltKutu sınıfında ust-kutuya ait anlık değişkenleri kullanarak nesne yarattık. Üst-sınıfın iç-değişkenleri private olmadığı sürece, bunu yapmak mümkündür. Ama, üst-sınıfın değişkenleri private damgalı olduğunda, alt-sınıftaki kodlar, onlara erişemeyecektir. Çoğunlukla, üst-sınıfın öğelerinin private olmasını isteriz. Böylece üst-sınıfın yapısını diğer sınıflardan saklarız. Buna encapsulation denir. Bu durumlarda, üst-sınıfın öğelerine erişmenin yolu super() metodunu kullanmaktır.

- Bir alt-sınıf içinde üst-sınıfa ait bir nesne yaratmak için sözdizimi şöyledir:

- super(formal_parametreler_listesi);

- Üst-sınıfta overload edilmiş constructor metotları varsa, alt-sınıftan super() metodu ile onların her birisi çağrılabilir. Çağrıda, formal parametreler listesine, üst-sınıfa ait olarak yaratılacak nesneyi belirleyecek gerçek parametreler konur. Bu demektir ki, üst-sınıfta overload edilen constructor'lardan hangisi isteniyorsa, onun parametreleri yerine gerçek değerler konulur. Ancak, super() metodu alt-sınıftaki constructor içine ilk deyim olarak yazılmalıdır; çünkü constructor çağrılınca ilk işi nesneyi yaratmaktır. Nesne yaratılmadan, onunla ilgili hiçbir iş yapılamaz.

AltKutu Sınıfı

```
    /* super() metodunun kullanılışı:
   alt-sınıf, super() metodunu kullanarak
   üst-sınıfın bir nesnesini yaratabilir ve
   onun değişkenlerine değer atayabilir.
*/
class AltKutu extends Kutu {
  double agr;  //kutu'nun ağırlığını tutacak değişken

  /* super() metodu ile en, boy ve yukseklik
     değerlerine atama yapıyor. */
  AltKutu(double e, double b, double y, double a) {
    super(e, b, y); // ust-sınıfa ait constructor'u çağrılıyor
    agr = a;
  }
}
```

- Dikkat edilirse, yukarıdaki AltKutu sınıfı içinde tanımlanan AltKutu() constructoru super(e,b,y)metodunu çağırmaktadır. Bu metot, üst-sınıftaki Kutu(e,b,y) constructor'unun yerine geçmektedir. Bunu daha iyi anlamak için, önce yazdığımız Kutu sınıfını anımsamak yetecektir. Tabii, uygulama programı bu satıra gelmeden önce e,b,y değerlerine gerçek parametre değerlerini atamış olacaktır. Dolayısıyla, constructor, gerçek parametrelerle çağrılıyor. Aksi halde nesneyi yaratamaz, derleme hatası oluşur.

Kutu Sınıfı

```
class Kutu {
double en;
double boy;
double yukseklik;

Kutu(double e, double b, double y) {
en = e;
boy = b;
yukseklik = y;
}
```

AltKutu constructoru gerçek parametrelerle çağrılınca Kutu sınıfına ait bir nesne yaratılmış ve onun en, boy ve yükseklik değerleri nesneye girilmiş olacaktır. AltKutu alt-sınıfı üst-sınıfında olmayan agr (ağırlık) değişkenine sahiptir. Dolayısıyla AltKutu() constructoru nesneyi yaratırken ona da bir değer atamalıdır. Constructor içindeki son deyim olan agr = a; deyimi bu atamayı yapacaktır.

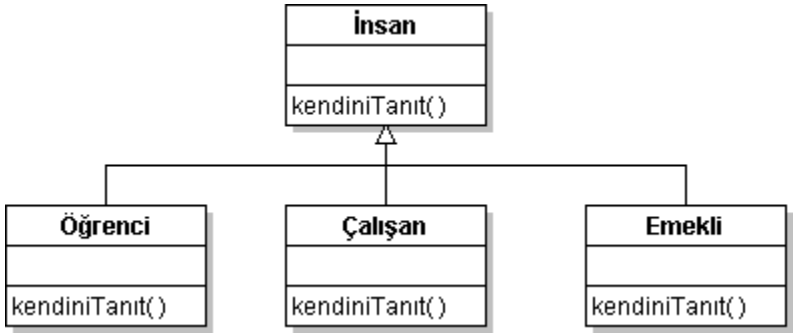
Örneğimizde, super() metodu 3 gerçek parametre ile çağrılmaktadır. Gerektiğinde bu sayı değişebilir. Üst-sınıfta constructor metodu overload edildiği için, aynı adlı farlı constructorları çağırmak mümkündür. Tabii, üst-sınıfta overload edilen bu metotların önceden tanımlanmış olması gerekir.

Polymorphism

- Bir nesnenin birden fazla nesle gibi davranma şeklidir.

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- İstersek kalıtımla geçen metotların gövdesini değiştirebileceğimizi öğrendik.
  - Bu işleme yeniden tanımlama (overriding) adı verildiğini gördük.
- Üst sınıftan bir nesnenin beklendiği her yerde alt sınıftan bir nesneyi de kullanabileceğimizi gördük.
- Bu iki özellik bir araya geldiğinde, ilgi çekici bir çalışma biçimi ortaya çıkar.

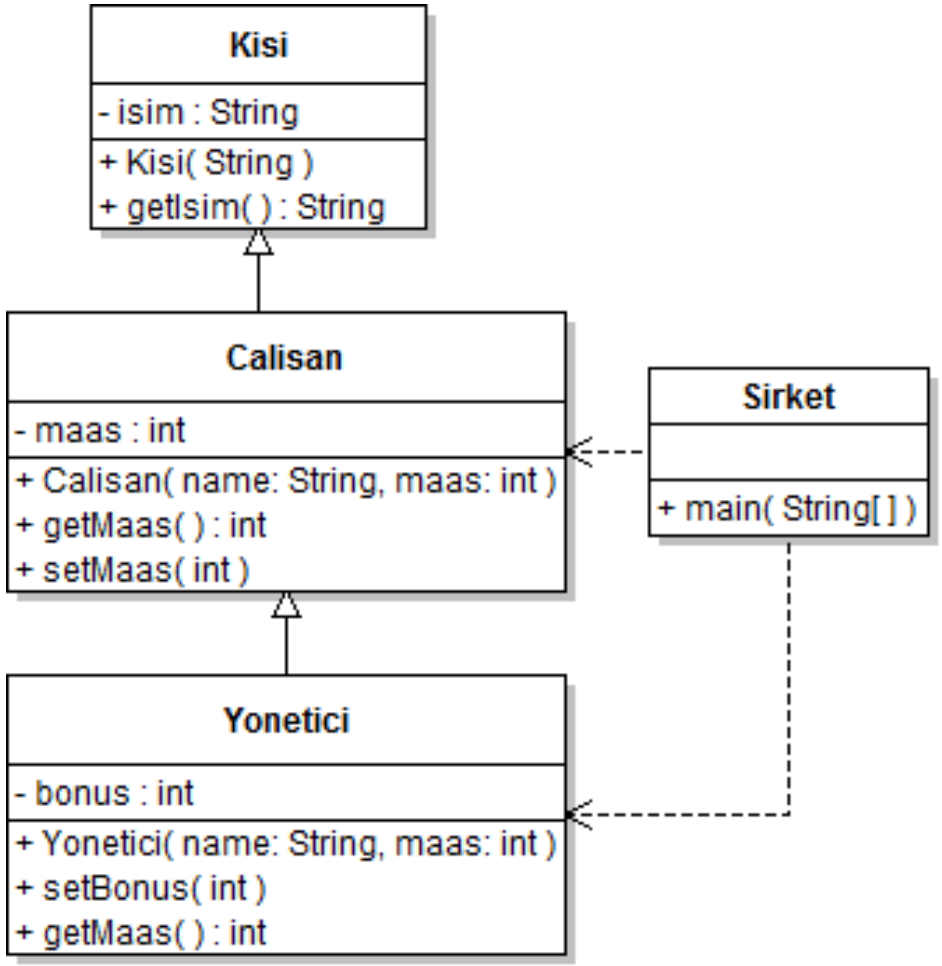


- Örnek alan modeli soldadır.
  - kendiniTanıt() metodu alt sınıflarda yeniden tanımlanmıştır.

- İnsan türünden bir dizi düşünelim, elemanları İnsan ve alt sınıflarından karışık nesneler olsun. Dizinin tüm elemanlarına kendini tanıt dediğimizde ne olacak?
  - Çalışma anında doğru sınıfın metodu seçilir.
  - Bu çalışma biçimine de çok biçimlilik (polymorphism) denir.
- Peki, üst sınıfın altta yeniden tanımladığımız bir metoduna eski yani üst sınıftaki hali ile erişmek istediğimizde ne yapacağız?
  - Bu durumda da **`super`** işaretçisi ile üst sınıfa erişebiliriz!

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Örnek kalıtım ağacı: Kişi – Çalışan – Yönetici
  - Ve bunları kullanan sınıf: Şirket
  - UML sınıf şeması:



```
package ndk06;
public class Kisi {
        private String isim;
        public Kisi( String name ) { this.isim = name; }
        public String getIsim( ) { return isim; }
}
```

```
package ndk06;
public class Calisan extends Kisi {
        private int maas;
        public Calisan( String name, int maas ) {
                super( name );
                this.maas = maas;
        }
        public int getMaas( ) { return maas; }
        public void setMaas( int salary ) { this.maas = salary; }
}
```

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

```
package ndk06;

public class Yonetici extends Calisan {
        private int bonus;

        public Yonetici( String name, int maas ) {
                super( name, maas );
                bonus = 0;
        }

        public void setBonus( int bonus ) {
                this.bonus = bonus;
        }

        public int getMaas( ) {
                return super.getMaas( ) + bonus;
        }

}
```

```
Yerine şöyle
yazılamaz:
super(name)
super(maas)
```

```
Yerine şöyle yazılamaz:
maas + bonus
Üst sınıftan kalıtımla
gelen private üyelere
doğrudan erişilemediğini
hatırlayınız.
```

- **`super`** işaretçisi zincirleme kullanılamaz:
  - Herhangi bir metotta super.super yazılamaz.
- Kurucu metotlarda super sadece bir kez ve ilk komut olarakkullanılır.

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

```
package ndk06;
public class Sirket {
   public static void main(String[] args) {
            Calisan[] calisanlar = new Calisan[3];;
            Yonetici mudur = new Yonetici( "Oktay Sinanoğlu", 10000 );
            mudur.setBonus( 2500 );
            calisanlar[0] = mudur;
            calisanlar[1] = new Calisan( "Attila İlhan", 7500 );
            calisanlar[2] = new Calisan( "Ümit Zileli", 6000 );
            for( Calisan calisan : calisanlar )
                        System.out.println( calisan.getIsim() + " " +
                                    calisan.getMaas( ) );
   }
}
```

- For döngüsü dikkatinizi çekti mi?

ADAŞ METOTLAR / ÇOKLU ANLAM YÜKLEME (OVERLOADING)

- Bir sınıfın aynı adlı ancak farklı imzalı metotlara sahip olabileceğini gördük.
- Böyle metotlara adaş metotlar, bu işleme ise çoklu anlam yükleme (overloading) adı verilir.
- Örnek: Çok biçimlilik konusu örneğindeki Yönetici sınıfına bir yapılandırıcı daha ekleyelim:
  - Yonetici(String name, int maas, int bonus)

```
public Yonetici( String name, int maas, int bonus ){
        super( name, maas );
        this.bonus = bonus;
}
```

  - Böylece yapılandırıcıya çoklu anlam yüklemiş olduk.
  - Bu kez de bu yapılandırıcıyı kullanacak kişi, maaş ile bonus'u birbirine karıştırmamalı.
- DİKKAT: Çoklu anlam yüklemenin kalıtımla bir ilgisi yoktur. Kalıtım olmadan da adaş metotlar oluşturulabilir, ancak kalıtım olmadan çok biçimlilik ve yeniden tanımlama mümkün değildir.

KALITIM VE TÜM SINIFLARIN ÜST SINIFI OLAN OBJECT SINIFI

- java.lang.Object sınıfı, aslında tüm sınıfların üst sınıfıdır.
- Kendi amaçlarınız için bu sınıfın metotlarını yeniden tanımlayabilirsiniz.
  - public String toString( ): Bir nesnenin içeriğini insanlarca kolay anlaşılabilir bir şekilde elde etmek için.
    - Aynı kendiniTanıt metodunda yaptığınız gibi.
  - Böylece bu String'i yazdırmak için doğrudan nesneyi yazdırabilirsiniz.