YMH112
ALGORİTMA ve PROGRAMLAMA II
LAB. KISMI
5. CANLI DERS
GECE (B)GRUBU

26 MAYIS 2021

SAAT =16.00-17:00

ARŞ. GÖR. ALEV KAYA

POLYMORPHİSM(ÇOK BİÇİMLİLİK)

- **Çok biçimlilik(Polymorphism), kalıtım(Inheritance)** ile iç içe geçmiştir.
- En genel anlamda**, oluşturduğumuz nesnelerin gerektiğinde kılıktan kılığa girip başka bir nesneymiş gibi davranabilmesine polimorfizm diyebiliriz (**Peki ne demek şimdi bu cümle?)
- Olaya ilk önce nesne kalıtımından kısaca söz ederek başlamak istiyorum. **OOP(Object Oriented Programming)** diyorsak nesnelerden bahsediyoruz demektir bu.
- **Her şey nesne** olarak düşünülebilir.
- Nesneler ise birbirlerinden türeyebilmektedir.

- **Örneğin bir ana sınıf düşünün** aklınızda, **bir de bu ana sınıfın yavrucuklarını** düşünün.

- Yavrucukları diyorum ama, bunu **yavru sınıflar daha küçüktür, daha az öğe içerir gibi düşünmeyin** sakın.

- **Tam tersine bu yavrucuklar daha gelişkin olabilirler.**

- Annelerinin tüm özelliklerine sahip olurlar da, hatta bir de annelerinden farklı başka özellikler de içerebilirler.

- Bu yavrucukların da yavrucukları olabilir.

- Bu böyle devam eder gider...

- Sağ tarftaki **UML diagramında** bir kalıtım yapısı bulunmakta.

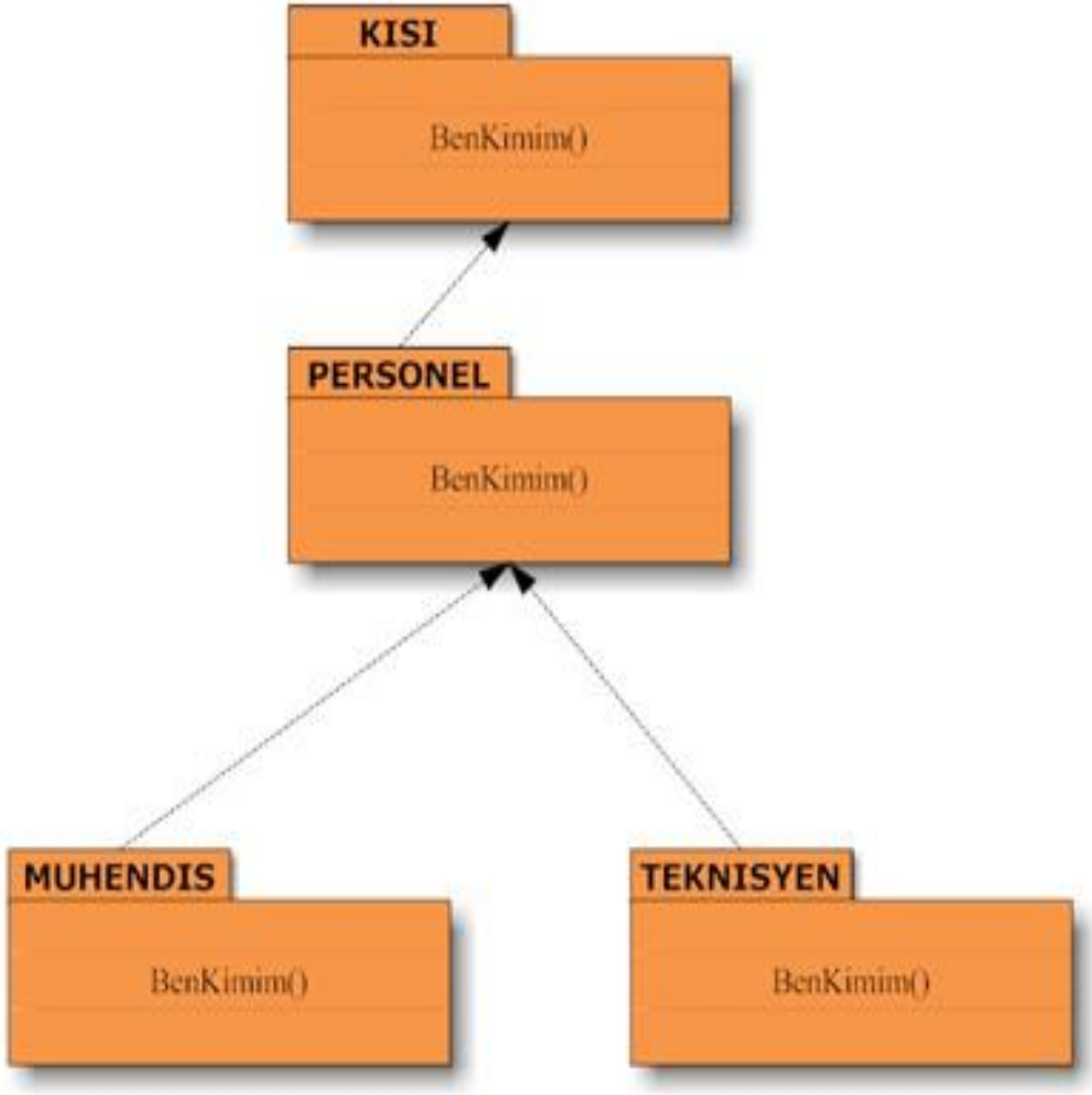
- **KISI** ana sınıfından **PERSONEL** isimli yeni bir sınıf ve **PERSONEL** sınıfından da da **MUHENDIS** ve **TEKNISYEN** olmak üzere iki farklı sınıf türetilmiştir.

- **MUHENDIS** ve **TEKNISYEN** bir **PERSONEL**'dir.
- **PERSONEL** ise bir **KISI**'dir türünde ifadelerin kurulabildiği her yerde kalıtımdan söz etmek mümkündür. Böylece bu nesnelerin arasında artık kalıtımı ilişkisi kurulmuş olur.

- **PERSONEL** sınıfı **KISI** sınıfında yer alan ***BenKimim*** isimli fonksiyonu **override** etmiş yani geçersiz kılmıştır.

- Bu fonksiyon için kendisi bir içerik oluşturmuştur. Aynı şekilde **MUHENDIS** ve **TEKNISYEN** sınıfları da kendi içeriklerini ***BenKimim*** fonksiyonu içerisine yerleştirmiştir.

- Sınıfların tanımlamalarını da aşağıda verilmekte.



Main.java

Source | History

```java
package kisi;

public class Main {

    static public class KISI {

        public void BenKimim() {
            System.out.println("Ben herhangi bir kisiyim.");
        }
    }

    static public class PERSONEL extends KISI {

        @Override
        public void BenKimim() {
            System.out.println("Ben bir personelim.");
        }
    }

    static public class MUHENDIS extends PERSONEL {

        @Override
        public void BenKimim() {
            System.out.println("Ben bir mühendisim.");
        }
    }

    static public class TEKNISYEN extends PERSONEL {

        @Override
        public void BenKimim() {
            System.out.println("Ben bir teknisyenim.");
        }
    }

    static public void KimimOnuYaz(KISI Birisi) {
        Birisi.BenKimim();
    }

```

```
27
28      static public class TEKNISYEN extends PERSONEL {
29
30          @Override
            public void BenKimim () {
32              System.out.println("Ben bir teknisyenim.");
33          }
34      }
35
36      static public void KimimOnuYaz(KISI Birisi) {
37          Birisi.BenKimim();
38      }
39
40      public static void main(String[] args) {
41          KISI Insan1 = new KISI();
42          PERSONEL Insan2 = new PERSONEL();
43          MUHENDIS Insan3 = new MUHENDIS();
44          TEKNISYEN Insan4 = new TEKNISYEN();
45          KimimOnuYaz(Insan1);
46          KimimOnuYaz(Insan2);
47          KimimOnuYaz(Insan3);
48          KimimOnuYaz(Insan4);
49      }
50  }
51
```

```
compile:
run:
Ben herhangi bir kisiyim.
Ben bir personelim.
Ben bir mühendisim.
Ben bir teknisyenim.
BUILD SUCCESSFUL (total time: 0 seconds)
```

- Kodu incelediğimizde görürüz ki olanlar aslında **polimorfizmin** ta kendisidir.

- Şöyle ki; ***KimimOnuYaz*** fonksiyonu parametre olarak yalnızca **KISI** sınıfı türünde değişkenleri kabul etmektedir.

- Fakat biz bu fonksiyon içerisine her dört sınıfın örneklerini de (**KISI**, **PERSONEL**, **MUHENDIS**, **TEKNISYEN**) gönderebildik ve herhangi bir hata ile de karşılaşmadık. Herhangi bir hata ile karşılaşmadığımız gibi üstüne üstlük çok güzel bir durum ile de karşılaştık. Peki bu iş nasıl oldu? Şimdi bu güzel durumu incelemeye çalışalım.

- Main fonksiyonu içerisinde dört sınıfa ait nesne örneklerimizi oluşturduk, sonrasında ilk önce **KISI** türünden ***Insan1*** **referansını** ***KimimOnuYaz*** isimli fonksiyona gönderdik.

- Burada bir tuhaflık yok, çünkü ***KimimOnuYaz*** fonksiyonu zaten giriş parametresi olarak **KISI** nesnesini seviyor.

- Dolayısı ile ***KimimOnuYaz*** içerisindeki ***Birisi*** nesne örneğinin ***BenKimim*** fonksiyonu çağırılmış oldu.

- *Birisi* referansının türü **KISI** olduğu için **KISI** sınıfının ***BenKimim*** fonksiyonu çağırıldı.

- İkinci olarak ise ***Insan2*** **referansı** ***KimimOnuYaz*** fonksiyonu tarafından giriş olarak alındı. İşte burada dikkat etmemiz gereken önemli bir nokta var.

- ***KimimOnuYaz*** fonksiyonu yalnızca **KISI** sınıfına ait nesne örneklerini kabul ettiği için tam bu satırda bir tür dönüşümü işlemi yapıldı. Bu işleme **UpCasting** adı verilmektedir.

- Yani tam olarak şu oldu aslında, ***KimimOnuYaz*** fonksiyonu içerisindeki **KISI** türünden ***Birisi*** nesnesi birçok kılığa birden bürünüverdi bir anda. Yeri geldi **KISI** **nesne türünden bir nesne örneği oldu**, yeri geldi **TEKNİSYEN** nesne türünden bir nesne örneği oluverdi bir anda.

- ***Birisi*** ismindeki nesne örneği kendi sınıfı olan **KISI** ve **KISI** sınıfından türeyen diğer nesne örneklerine bağlanabildi.

- Sonuç olarak ise *Birisi* referansının nesne türü ne ise o nesneye ait ***BenKimim*** fonksiyonu çağırıldı. **İşte size Polimorfizm!!!**

- Bu kod satırlarının arkasında saklı olan bir diğer kavram ise **Geç Bağlama(Late Binding)'dır.**

- **Geç bağlamayı temel olarak bir nesne örneğinin hangi nesneye bağlandığı ve hangi nesneye ait olduğunun çalışma zamanında(Run Time) belli olması olarak tanımlanabilir.**

- **Eğer siz bir nesne örneği oluşturur ve bu referansın herhangi bir öğesine ulaşmak isterseniz bu derleme zamanında(Compile Time) ortaya çıkan bir durumdur.**

- Yani referansa ve bu referansın bağlandığı nesneye ait her şey gayet açıktır. Fakat yukarıdaki örneğimizdeki gibi bir yöntem uygulayacak olursanız bu sefer referans nesnenin bağlandığı nesne belirli değildir ve **run time** sırasında belli olmaktadır.

- Tıpkı *Birisi* referansının yeri geldiğinde **KISI** sınıfına bağlanması yeri geldiğinde ise **PERSONEL** sınıfına bağlanması gibi.

- Peki bu örnek ile çok biçimliliği kanıtladık ama bu **çok biçimlilik ne işe yarar ki? Nerelerde ihtiyaç duyacağız ki biz böyle bir şeye?**

- Doğal olarak kullanım alanı olmayan bir şeyin gerekliliği de tartışılır. Yukarıdaki örnek aslında gözümüze çok hoş göründüğü için sanki olağan bir şey miş gibi karşılıyor olabilirsiniz.

- Zaten yukarıdaki programcıktaki gibi bir işi yaptırabilmek için böyle bir kod yazmak gerekirdi diyebilirsiniz ama hiç de öyle değil.

- Eğer çok biçimlilik bizim hizmetimize sunulmamış olsaydı aşağıdaki gibi kalabalık ve hiç de hoş görünmeyen son derede adaptif olmayan bir yönteme baş vurmak zorunda kalacaktık.

- Bu internette yaptığım aramalarda çok sıklıkla karşıma çıkan ve olayı çok güzel özetleyen bir örnek olacak.

Main.java

Source | History

```java
package kısı2;

public class Main {

    static public class KISI2 {

        public void BenKimim() {
            System.out.println("Ben herhangi bir kisiyim.");
        }
    }

    static public class PERSONEL extends KISI2 {

        @Override
        public void BenKimim() {
            System.out.println("Ben bir personelim.");
        }
    }

    static public class MUHENDIS extends PERSONEL {

        @Override
        public void BenKimim() {
            System.out.println("Ben bir mühendisim.");
        }
    }

    static public class TEKNISYEN extends PERSONEL {

        @Override
        public void BenKimim() {
            System.out.println("Ben bir teknisyenim.");
        }
    }

```

```
35
36      static public void KimimOnuYaz(KISI2 Birisi) {
37          //Birisi.BenKimim();
38          if (Birisi instanceof KISI2) {
39              KISI2 Obje = (KISI2) Birisi;
40              Obje.BenKimim();
41          } else if (Birisi instanceof PERSONEL) {
42              PERSONEL Obje = (PERSONEL) Birisi;
43              Obje.BenKimim();
44          } else if (Birisi instanceof MUHENDIS) {
45              MUHENDIS Obje = (MUHENDIS) Birisi;
46              Obje.BenKimim();
47          } else if (Birisi instanceof TEKNISYEN) {
48              TEKNISYEN Obje = (TEKNISYEN) Birisi;
49              Obje.BenKimim();
50          }
51      }
52
53      public static void main(String[] args) {
            KISI2 Insan1 = new KISI2();
55          PERSONEL Insan2 = new PERSONEL();
56          MUHENDIS Insan3 = new MUHENDIS();
57          TEKNISYEN Insan4 = new TEKNISYEN();
58          KimimOnuYaz(Insan1);
59          KimimOnuYaz(Insan2);
60          KimimOnuYaz(Insan3);
61          KimimOnuYaz(Insan4);
62      }
63  }
64
```

```
compile:

run:

Ben herhangi bir kisiyim.

Ben bir personelim.

Ben bir mühendisim.

Ben bir teknisyenim.

BUILD SUCCESSFUL (total time: 0 seconds)
```

- Az önce tek bir satırlık kod ile kolay ve çok güzel bir şekilde hallettiğimiz işimizi artık satırlarca kod yazarak ancak görebiliyoruz.

- Evet iki programcık da aynı işi yapıyor ve hiç bir farkları yok birbirlerinden. Aslında program çıktısı olarak değil de performans olarak ufak bir farkları var, Bu konuya yazının sonuna doğru geleceğim.

- İkinci yöntemin çok da hoş olmadığı açık. ***KimimOnuYaz*** fonksiyonuna yine temel sınıf ve bu sınıftan türeyen nesne örneklerini gönderiyoruz fakat çok biçimliliği uygulayamadığımız için gelen nesne örneklerinin hangi sınıftan türediğini bilme ihtiyacı hissediyoruz.

- Ayrıca gelen referansları da ait oldukları nesne türüne tür dönüşümü yaparak çeviriyoruz. Çünkü hangi nesneye ait ***BenKimim*** fonksiyonunun çağırılacağının belirlenmesi gerekmektedir.

- Kalıtım nedeni ile ana sınıfta olup da **yavru sınıflarda override edilmeyen birçok metod olabilir.** Bu durumda ana sınıf türünden bir değişkene yavru sınıfa ait bir referansı kolaylıkla bağlayablirsiniz, bir sorun ile de karşılaşmazsınız. Metodu da rahatlıkla çağırabilirsiniz.

- Fakat kalıtım yolu ile yavru sınıflara iletilmeyen yeni bir metoda sahip yavru sınıfa ait referansı, ana sınıfa ait bir değişkene bağlayıp, yavru sınıfta yeni oluşturulan metodu çağırmaya kalktığınızda olanlar olacaktır. **Bu bir derleme zamanı hatası oluşturur.**

- Çünkü bu metod ana sınıfta yer almamaktadır**. Polimorfizmin olduğu her yerde ana ve yavru sınıflara ait metodlar aynı olmalıdır ve her iki sınıf içerisinde de yer alıyor olmalıdır. Bu zaten polimorfizm mantığının bir gereğidir.**

- Çok biçimliliğin bu güzelliklerinin bir bedeli de var tabi ki az önce bahsettiğim gibi. O da performans konusunda bazı olumsuzluklara yol açması olarak gösterilebilir. Bunun nedeni ise **JVM'in çalışma anında geç bağlama oluşup oluşmadığına karar verme çabası ve referansların hangi nesne türüne bağlanacağının kontrol edilmesi ve o nesne türüne bir bağlama yapılmasıdır.**

- Normalde bir dizi oluşturduğumuzda bu dizinin tüm elemanları oluşturulduğu nesne türünden elemanlar olmalıdır. Yani **int** türünde bir dizi değişken tanımlayıp bu diziye **string** türünden bir elemanı gönderemezsiniz. Gönderirseniz zaten hatalar karşınıza çıkacaktır.

- **Fakat polimorfizm ile farklı türden değişkenleri bir diziye atmamız bir derece mümkün olmaktadır, eğer aralarında bir kalıtımdan söz edilebiliyor ve upcasting yapılabiliyor ise. Yani ortada bir polimorfizm var ise.**

- ...