

BLM5504 NESNEYE DAYALI KAVRAMLAR VE PROGRAMLAMA
Yrd. Doç. Dr. Yunus Emre SELÇUK

DERS NOTLARI:
C. NESNEYE YÖNELİK İLERİ KAVRAMLAR

1

NESNELER ARASINDAKİ İLİŞKİLER

NESNELER ARASINDAKİ İLİŞKİLER

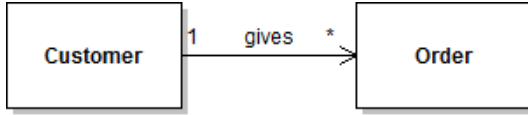
- Bir nesneye yönelik programın, nesneler arasındaki mesaj akışları şeklinde yürüdüğünü gördük.
- Bir nesnenin diğerine bir mesaj gönderebilmesi (yani kullanabilmesi) için, bu iki nesne arasında bir ilişki olmalıdır.
- İlişki çeşitleri:
 - Sahiplik (Association)
 - Kullanma (Dependency)
 - Toplama (Aggregation)
 - Meydana Gelme (Composition)
 - Kalıtım/Miras Alma (Inheritance)
 - Kural koyma (Associative)
- Bu ilişkiler UML sınıf şemalarında gösterilir ancak aslında sınıf örnekleri yani nesneler arasındaki ilişkiler olarak anlaşılmalıdır.

2

NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)

- Bağlantı ilişkisi için anahtar kelime **sahipliktir**.
- Kullanan nesne, kullanılan nesne türünden bir üyeye sahiptir.
- Sadece ilişki kelimesi geçiyorsa, ilişkinin iki nesne arasındaki sahiplik ilişkisi olduğu anlaşılır.
- Bir nesnenin diğerinin yeteneklerini kullanması nasıl olur?
 - Yanıt: Görülebilirlik kuralları çerçevesinde ve metotlar üzerinden.
 - Yani: Mesaj göndererek.
- Örnek: Müşteri ve siparişleri
 - İlişki adları ve nicelikleri de yazılabilir.

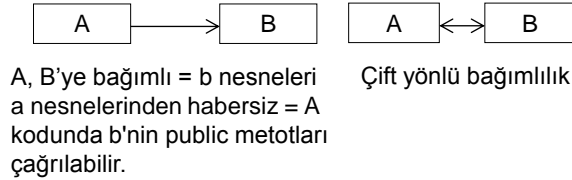


3

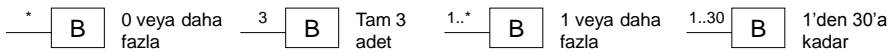
NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)

- Gösterim:



- Okun yönü önemli, kimin kime mesaj gönderebileceğini gösterir.
- Ok yoksa:
 - Ya çift yönlü bağımlılık vardır,
 - Ya da yazılım mimarı henüz bağımlılığın yönünü düşünmemiştir.
- İlişkinin uçlarında sayılar olabilir (cardinality)
 - Çoğulluk ifade eder.
 - İlişkinin o ucunda bulunan nesne sayısını gösterir.



4

NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

- Sahiplik ilişkisi çizgi ve oklarla gösterilmişse, sınıfların içerisinde ayrıntılı olarak gösterilmek zorunda değildir.
 - Ör: Sol alttaki şekil ile sağ alttaki şekil denktir.
 - Diğer ilişkiler için de aynı şey geçerlidir.



KULLANMA İLİŞKİSİ (DEPENDENCY)

- Bir diğerine giden bir mesajın **parametresi** ise veya bir nesne diğerini **sahiplik olmadan kullanıyorsa**.
 - Gösterim:



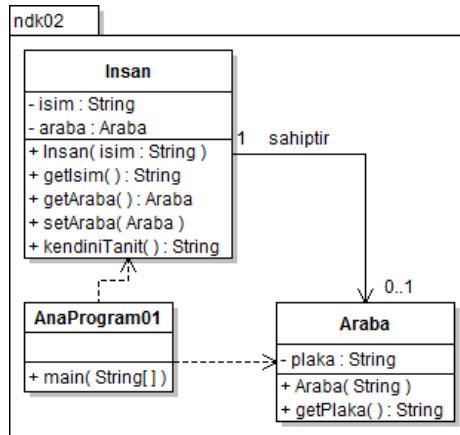
A, B'yi kullanır: A örnekleri birMetot içinde b nesnesine mesaj gönderebilir.

5

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Her insanın bir arabasının olabileceği bir alan modeli oluşturalım.
- Alan modelini kullanan bir de uygulama yazalım (main metodu içeren).
- Karmaşık yazılımlarda alan modeli ile uygulamanın ayrı paketlerde yer alması daha doğru olacaktır.
- UML sınıf şeması yandadır.
- SORU: Sahiplik ilişkisinin Araba ucu neden 0..1?
- Gizli Bilgi: Araba kurucusuna dikkat



6

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Araba sınıfının kaynak kodu:

```
package ndk02;

public class Araba {
    private String plaka;
    public Araba (String plaka) {
        this. plaka = plaka;
    }
    public String getPlaka( ) {
        return plaka;
    }
}
```

- Yukarıdaki koda göre, bir araba nesnesi ilk oluşturulduğunda ona bir plaka atanır ve bu plaka bir daha değiştirilemez.
- Araba sınıfını kodlamak kolaydı, gelelim İnsan sınıfına:

7

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- İnsan sınıfının kaynak kodu:

```
package ndk02;
public class İnsan {
    private String isim;
    private Araba araba;

    public İnsan( String isim ) { this.isim = isim; }

    public String getIsim( ) { return isim; }
    public Araba getAraba( ) { return araba; }
    public void setAraba( Araba araba ) {
        this.araba = araba;
    }
    public String kendiniTanıt( ) {
        String tanitim;
        tanitim = "Merhaba, benim adım " + getIsim()+ ".";
        if( araba != null )
            tanitim += "\n" + araba.getPlaka()+ " plakalı bir arabam var.";
        return tanitim;
    }
}
```

Dikkat! (Bu da nereden çıktı?)

8

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- UML sınıf şemamızda İnsan - Araba ilişkisinin Araba ucunun 0..1 yazdığı dikkatinizi çekti mi?
 - Bu ne anlama geliyor?
 - Her insanın bir arabası olmayabilir anlamına geliyor.
- Ayrıca:
 - Bir sınıfa bir metot eklenince, hangi metodun hangi sırada çalıştırılacağı, hatta çalıştırılıp çalıştırılmayacağına garantisi yoktur.
 - constructor ve finalizer'ın özel kuralları dışında.
- Buna göre bir insan oluşturulabilir ancak ona araba atanmayabilir.
 - İnsanın arabası olmayınca plakasını nasıl öğrenecek?
 - Bu durumda çalışma anında "NullPointerException" hatası ile karşılaşacaksınız.
 - Ancak bizim sorumluluğumuz, sağlam kod üretmektir. Bu nedenle:
 - İnsanın arabasının olup olmadığını sınavalım, ona göre arabasının plakasına ulaşmaya çalışalım.
 - İnsanın arabası yokken, o üye alanın değeri **null** olmaktadır.
 - Yani o üye ilklendirilmemiştir.

9

NESNE İLİŞKİLERİNİN KODLANMASI

NESNENİN ETKİNLİĞİNİN SINANMASI

- Bir nesne ilklendirildiğinde artık o nesne için etkindir denilebilir.
- nesne1 işaretçisinin gösterdiği nesnenin ilklendirilip ilklendirilmediğinin sinanması:

	İfade	Değer
İlklenmişse (etkinse)	nesne1 == null	false
	nesne1 != null	true
İlklenmemişse (etkin değilse)	nesne1 == null	true
	nesne1 != null	false

10

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Nihayet main metodu içeren uygulamamızı yazabiliriz:

```
package ndk02;

public class AnaProgram01{

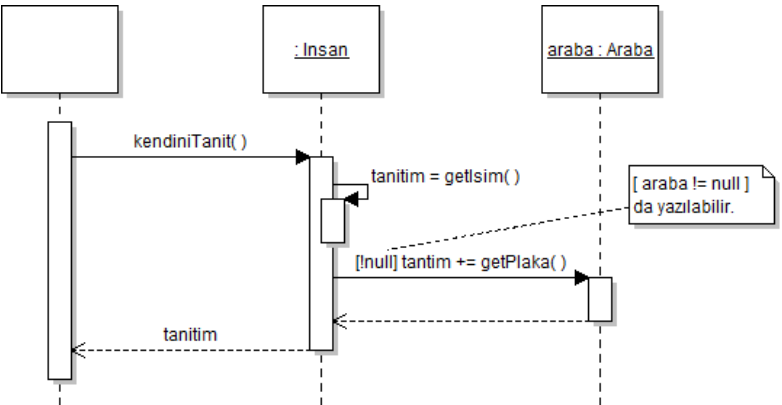
    public static void main(String[ ] args) {
        İnsan oktay;
        oktay = new İnsan("Oktay Sinanoğlu");
        Araba rover;
        rover = new Araba("06 RVR 06");
        oktay.setAraba(rover);
        İnsan aziz = new İnsan("Aziz Sancar");
        System.out.println( oktay.kendiniTanit() );
        System.out.println( aziz.kendiniTanit() );
    }
}
```

11

NESNE İLİŞKİLERİNİN KODLANMASI

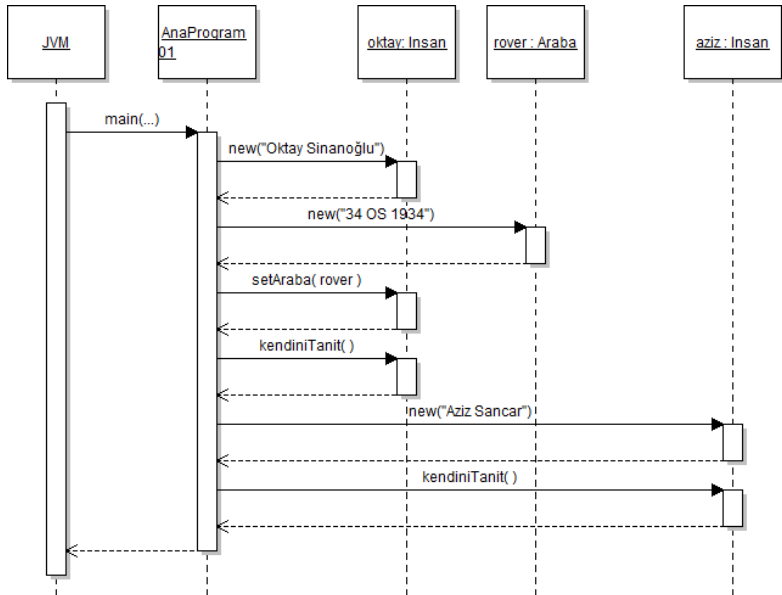
BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- İnsan sınıfının kendiniTanit metodunun etkileşim şeması:



12

- AnaProgram01 çalıştırılması ile ilgili etkileşim şeması :

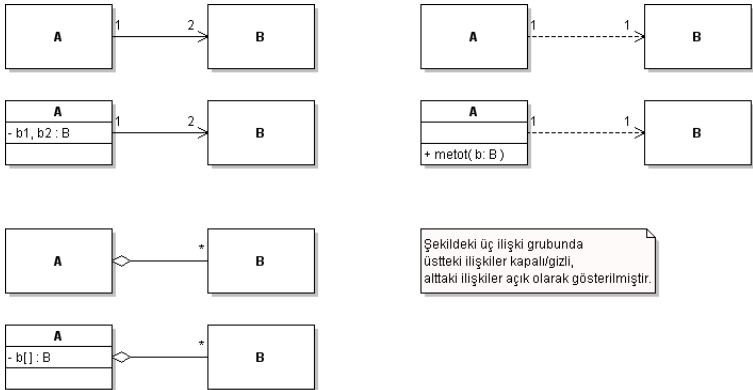


13

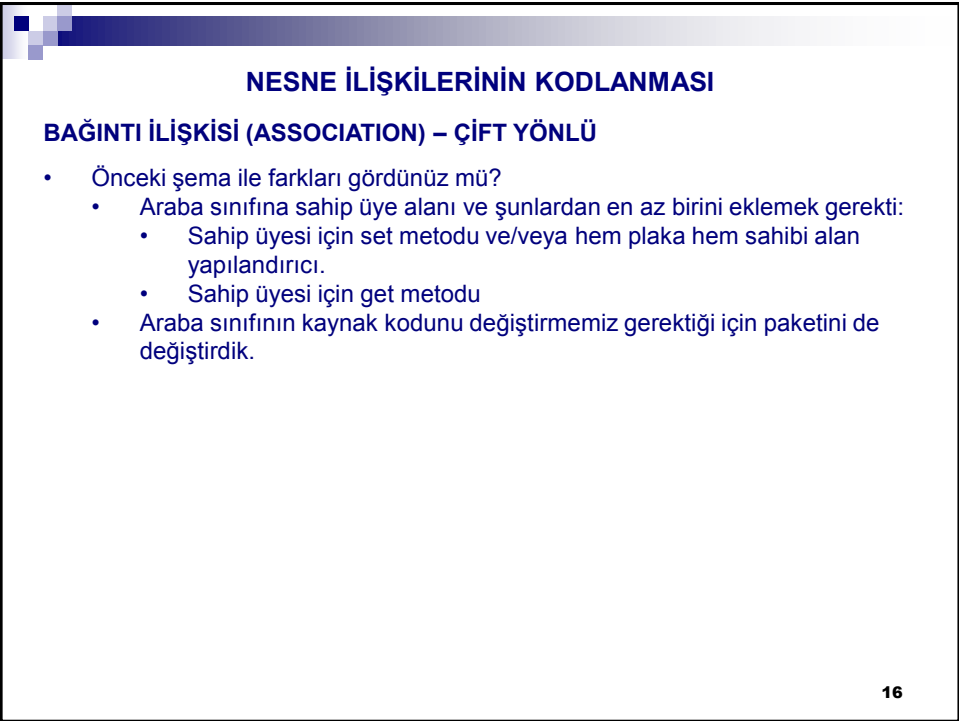
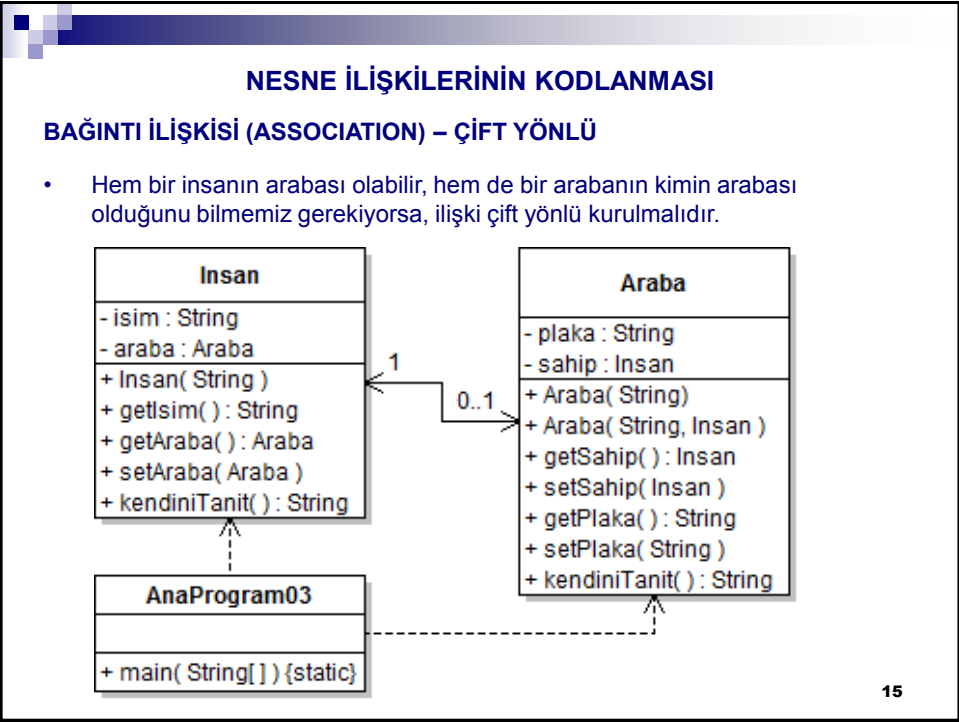
NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

- Sahiplik, kullanma, parça-bütün ilişkileri oklarla gösterilmişse, sınıfların içerisinde ayrıntılı olarak gösterilmek zorunda değildir.



14



NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Araba sınıfının yeni kaynak kodu:

```
package ndk03;
public class Araba {
    private String plaka;
    private İnsan sahip;
    public final static String cins = "Ben bir araba nesnesiyim.";
    public Araba( String plakaNo ) { plaka = plakaNo; }
    public Araba(String plaka, İnsan sahip) {
        this.plaka = plaka;
        this.sahip = sahip;
    }
    public void setSahip( İnsan sahip ) { this.sahip = sahip; }
    public İnsan getSahip() { return sahip; }
    public String getPlaka() { return plaka; }
    public void setPlaka( String plaka ) { this.plaka = plaka; }
    public String kendiniTanıt() {
        String tanitim;
        tanitim = cins + "Plakam: " + getPlaka() + ".";
        if( sahip != null )
            tanitim += "\nSahibimin adı: " + sahip.getIsim();
        return tanitim;
    }
}
```

17

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Neden az önceki kodda if komutuna dikkat çektik?
 - Çünkü birisi Car(String) metodunu çağırıp setOwner metodunu çağırmaı unutabilir.
 - Peki o halde Car(String) kurucusunu silelim mi?
 - Hayır, çünkü gerçek dünyada arabaların fabrikadan çıkar çıkmaz bir sahibi olmaz.

18

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Yazdıklarımızı denemek için uygulamayı yazalım.

```
01 package ndk03;
02 public class AnaProgram03 {
03     public static void main(String[] args) {
04         İnsan oktay = new İnsan("Oktay Sinanoğlu");
05         Araba rover = new Araba("06 RVR 06");
06         oktay.setAraba(rover);
07         rover.setSahip(oktay);
08         System.out.println( oktay.kendiniTanit() );
09         System.out.println( rover.kendiniTanit() );
10
11         İnsan aziz = new İnsan("Aziz Sancar");
12         Araba honda = new Araba("47 AS 1946");
13         aziz.setAraba(honda);
14         honda.setSahip(aziz);
15         System.out.println( aziz.kendiniTanit() );
16         System.out.println( honda.kendiniTanit() );
17     }
18 }
19
20
```

19

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Önceki uygulamada ne gibi sorunlar görüyorsunuz?
 - Niye hem 6. hem de 7. satırları yazmak zorunda kalalım?
 - Ya o satırları yazmayı unutursak?
 - Ya başka (oktay, rover) – (aziz, honda) ilişkilerini kurarken yanlışlıkla çapraz bağlantı kursak?
 - vb.
- Bu sorunların hepsi, çift yönlü ilişkiyi daha sağlam kurarak ortadan kaldırılabilir.
 - Nereyi değiştirmemiz lazım?

20

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- İnsan ve Araba sınıflarının değişen kısımları:

```
package ndk04;
public class İnsan {
    /*eski kodu da ekle*/
    public void setAraba(Araba araba) {
        this.araba = araba;
        if( araba.getSahip() != this )
            this.araba.setSahip(this);
    }
}

package ndk04;
public class Araba {
    /*eski kodu da ekle*/
    public void setSahip(İnsan sahip) {
        this.sahip = sahip;
        if( sahip.getAraba() != this )
            this.sahip.setAraba(this);
    }
}
```

21

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

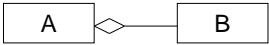
- Sonuç:
 - Çift yönlü bağıntı tek yönlü bağıntıya göre daha esnektir ancak kodlaması daha zordur.
 - Bu nedenle çift yönlü bağıntıya gerçekten ihtiyacınız yoksa kodlamayın.
 - Peki ya sonradan ihtiyaç duyarsak?
 - Şimdiden kodlamaya çalışıp zaman kaybetmeyin. Zaten yetiştirmeniz gereken bir dolu başka işiniz olacak!

22

NESNELER ARASINDAKİ İLİŞKİLER

Toplama (Aggregation)

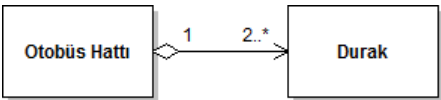
- **Parça-bütün ilişkisini** simgeler.
- Gösterim:



Toplama (aggregation)

- A örneği birden fazla B örneğine sahiptir
- A: Bütün, B: Parça.

- Şemada gösterilme de, toplama ilişkisi şunları ifade eder:
 - Elmas ucunda 1 olur
 - Diğer uçta * ve ok olur.
- Toplama, sahiplik ilişkisinden kavramsal olarak daha güçlüdür.
 - Toplama, sıradan sahiplikten daha güçlü kurallara sahiptir.
 - Örneğin, bir otobüs hattı en az iki durağa sahip olmalıdır ve bir hatta yeni duraklar eklemek için uyulması gereken bazı kurallar vardır.

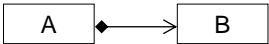


23

NESNELER ARASINDAKİ İLİŞKİLER

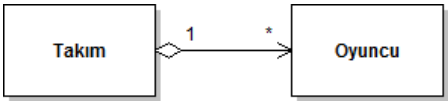
Meydana Gelme (Composition)

- Daha kuvvetli bir parça-bütün ilişkisini simgeler.



Meydana gelme
(composition)

- Meydana gelme ilişkisinde, toplamadan kuvvetli olarak, bir parça aynı anda sadece bir tek bütüne dahil olabilir.
- Örnek:



24

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

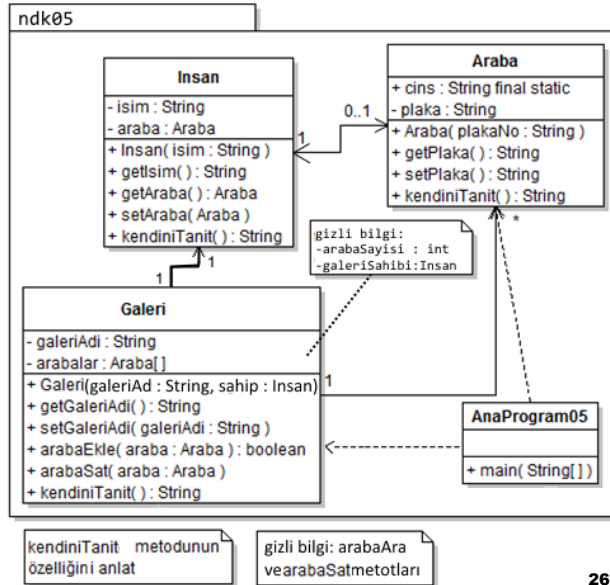
- Örnek: Satılacak birden fazla araba içeren Galeri adlı bir sınıf oluşturalım.
 - Daha önce yazdığımız Araba sınıfını aynen kullanabiliriz.
 - Ek olarak sadece plaka alan bir kurucu yazmak isteyebiliriz. O zaman aynen kullanmamış oluruz.
 - Aynen kullanacaksak:
 - kurucuya sahip üyesini null olarak verebiliriz (NullPointerException?)
 - veya galerinin sahibini verebiliriz (daha emin)
 - Bir Galeri nesnesi birden fazla araba ile ilişkili olabileceğinden toplama veya 1..* sahiplik ilişkisi ile gösterim yapabiliriz.
 - Bu sırada Java'da dizilerin kullanımını ve for döngüsünü de görmüş olacağız.

25

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- UML sınıf şeması:



26

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodu:

```
package ndk05;
public class Galeri {
    private İnsan galeriSahibi;
    private String galeriAdi;
    private Araba[] arabalar;
    private int arabaSayisi;

    public Galeri( String galeriAd, İnsan sahip ) {
        galeriAdi = galeriAd; galeriSahibi = sahip;
        arabaSayisi = 0;
        arabalar = new Araba[30];
    }
    public String getGaleriAdi() { return galeriAdi; }
    public void setGaleriAdi(String galeriAdi) { this.galeriAdi = galeriAdi; }
    public String kendiniTanit( ) {
        String tanitim;
        tanitim = galeriAdi + " adlı galerinin sahibi: " + galeriSahibi.getIsim();
        tanitim += "\nGaleride " + arabaSayisi + " adet araba var.";
        return tanitim;
    }
}
//devamı var...
```

Not: Burada bir kurucu çalışmadı. Sadece dizi için bellekte yer ayrıldı.

27

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodunun devamı:

```
public boolean arabaEkle( Araba araba ) {
    if( arabaAra(araba.getPlaka()) != null )
        return false;
    if( arabaSayisi < arabalar.length ) {
        arabalar[ arabaSayisi ] = araba;
        arabaSayisi++;
        return true;
    }
    else
        return false;
}
public Araba arabaAra( String plaka ) {
    for( int i=0; i<arabaSayisi; i++ )
        if( arabalar[i].getPlaka().equalsIgnoreCase(plaka) )
            return arabalar[i];
    return null;
}
//sınıf kodu devam edecek.
```

Burada önce araba zaten eklenmiş mi diye, ayrı bir metod yardımı ile bir denetleme yapmayı akıl edip kodladık.

28

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodunun devamı:

```
private int arabaBul( String plaka ) {
    for( int i=0; i<arabaSayisi; i++ )
        if( arabalar[i].getPlaka().equalsIgnoreCase(plaka) )
            return i;
    return -1;
}
public boolean arabaSat( String plaka ) {
    int yer = arabaBul(plaka);
    if( yer != -1 ) {
        for( int i=yer; i<arabaSayisi-1; i++ )
            arabalar[i] = arabalar[i+1];
        arabaSayisi--; arabalar[arabaSayisi] = null;
        return true;
    }
    return false;
}
} //end class
```

- Alıştırma: arabaSat metodu daha çok arabaSil metodu olmuş. Satışta mali konular devreye girmelidir. Öylesi bir durum nasıl kodlanmalıdır?

29

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtım benzetmesi: Bir çocuk, ebeveyninden bazı genetik özellikleri alır.
- NYP: Mevcut bir sınıftan yeni bir sınıf türetmenin yoludur.
- Gösterim:



Kalıtım (inheritance)

- Ok yönüne dikkat!

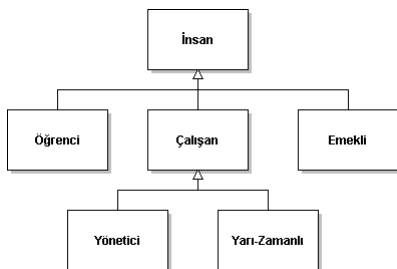
- A:
 - Ebeveyn sınıf (parent)
 - Üst sınıf (super)
 - Temel sınıf (base)
- B:
 - Çocuk sınıf (child)
 - Alt sınıf (sub)
 - Türetilmiş sınıf (derived)
- Kalıtımın işleyişi:
 - Kalıtım yolu ile üst sınıftan alt sınıfa hem üye alanlar hem de üye metotlar aktarılır
 - private üyeler dahil, ancak alt sınıf onlara doğrudan ulaşamaz.
 - Protected üyeler ve kalıtım:
 - Alt sınıflar tarafından erişilir, diğer sınıflar tarafından erişilemez.

30

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtım kuralları:
 - Miras alma adlandırmasının uygunsuzluğu: Alt sınıf herhangi bir üyeyi miras almamayı seçemez.
 - Ancak kalıtımla geçen metotların gövdesi değiştirilebilir.
 - Yeniden tanımlama: Overriding.
 - Final olarak tanımlanan metotlar yeniden tanımlanamaz.
 - Alt sınıfta yeni üye alanlar ve üye metotlar tanımlanabilir.
 - Alt sınıflardan da yeni alt sınıflar türetilir. Oluşan ağaç yapısına kalıtım hiyerarşisi veya kalıtım ağacı denir.



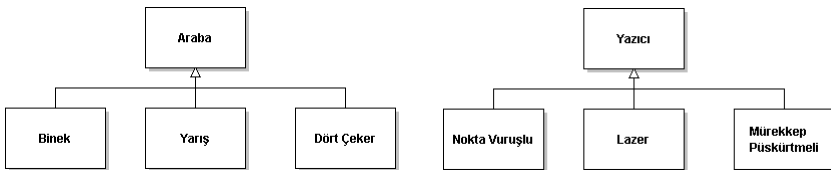
- Kalıtım ağacını çok derin tutmak doğru değildir (Kırılgan üst sınıf sorunu: Bina temelinin çürümesi gibi).

31

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtımın etkileri:
 - Genelleşme – özelleşme ilişkisi (generalization – specialization).
 - Alt sınıf, üst sınıfın daha özelleşmiş, daha yetenekli bir türüdür.
 - Yerine geçebilme ilişkisi (substitutability).
 - Alt sınıftan bir nesne, üst sınıftan bir nesnenin beklendiği herhangi bir bağlamda kullanılabilir.
 - Bu nedenle IS-A ilişkisi olarak da adlandırılır.



32

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtımın yanlış kullanımı:

33

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

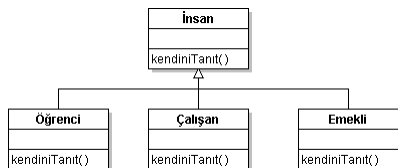
- Gereksinim:
 - Hastaların isimleri, TC kimlik no.ları, doğum tarihleri ve cep telefonları saklanmalıdır. Bu bilgiler dış hekimleri için de saklanmalıdır. Hekimlerin diploma numaralarının saklanması ise kanun gereği zorunludur. Hangi hastanın hangi tarihte hangi hekim tarafından hangi tedaviye tabi tutulduğu sistemden sorgulanabilmelidir.
- Kalıtımın yanlış kullanımı: Kalıtımın doğru kullanımı:

- Hekimlerin tedavi kaydının tutulması gerekmemektedir. Yanlış kullanımda her hekim aynı zamanda bir hasta olduğu için, tedavi kaydı bilgisini de alır. 34

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- İstersek kalıtımla geçen metodların 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.

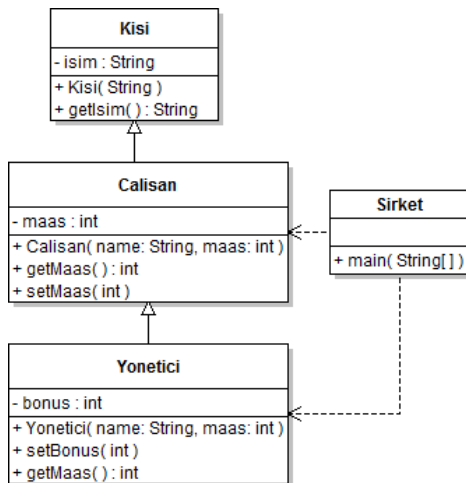
- İnsan türünden bir dizi düşünelim, elemanları İnsan ve alt sınıflarından karışık nesneler olsun. Dizinin tüm elemanlarına kendini tanıt dediğimizde ne olacak?
 - Çalışma anında doğru sınıfın metodu seçilir.
 - Bu çalışma biçimine de çok biçimlilik (polymorphism) denir.
- Peki, üst sınıfın altta yeniden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

35

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

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



KALITIM İLE İLGİLİ ÖZEL KONULAR

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( ) );
    }
}
```

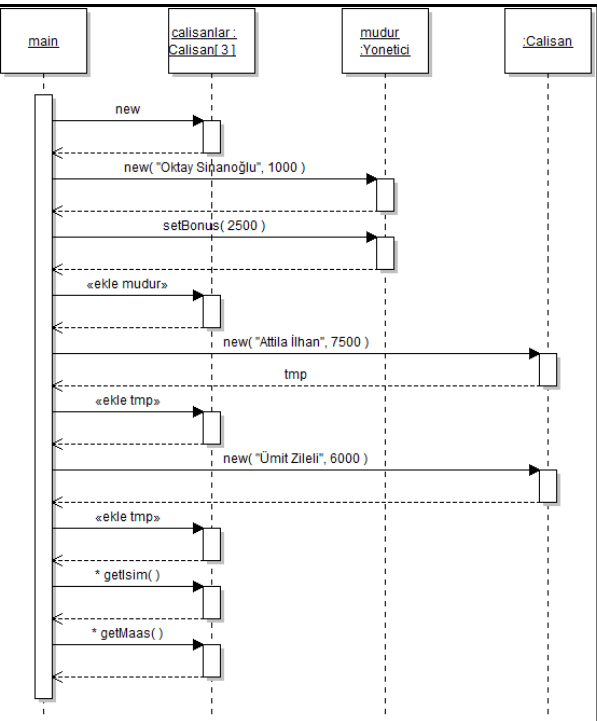
- For döngüsü dikkatinizi çekti mi?

39

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Şirket sınıfının etkileşim şeması:



NYP İLE İLGİLİ ÖZEL KONULAR

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:
 - Yönetici(String name, int maas, int bonus)

```
public Yönetici( 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.

41

NESNE İLİŞKİLERİNİN KODLANMASI

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.

42

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT SINIFLAR

- Soyut sınıflar, kendilerinden kalıtım ile yeni normal alt sınıflar oluşturmak suretiyle kullanılan, bir çeşit şablon niteliğinde olan sınıflardır.
 - Şimdiye kadar kodladığımız normal sınıflara İngilizce *concrete* de denir.
 - Eğer bir sınıfı soyut yapmak istiyorsak, onu **abstract** anahtar kelimesi ile tanımlarız.
- Soyut sınıflardan nesne oluşturulamaz.
- Ancak soyut sınıfın normal alt sınıflarından nesneler oluşturulabilir.
- Soyut sınıflar da normal sınıflar gibi üye alanlar içerebilir.
- Soyut sınıfın metotları soyut veya normal olabilir:
 - Soyut metotların abstract anahtar kelimesi de kullanılarak sadece imzası tanımlanır, gövdeleri tanımlanmaz.
 - Bir soyut sınıfta soyut ve normal metotlar bir arada olabilir.
- Soyut üst sınıflardaki soyut metotların gövdeleri, normal alt sınıflarda mutlaka yeniden tanımlanmalıdır.
 - Aksi halde o alt sınıflar da soyut olarak tanımlanmalıdır.

43

NYP İLE İLGİLİ ÖZEL KONULAR

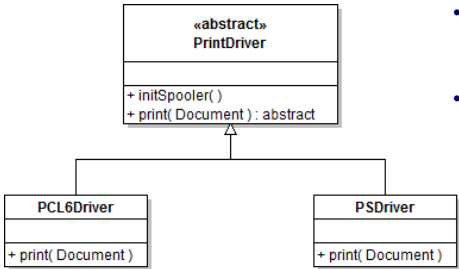
SOYUT SINIFLAR

- Ne zaman soyut sınıflara gereksinim duyulur:
 - Bir sınıf hiyerarşisinde yukarı çıkıldıkça sınıflar genelleşir. Sınıf o kadar genelleşmiş ve kelime anlamıyla soyutlaşmıştır ki, nesnelere o açıdan bakmak gerekmez.
 - Soyut sınıfları bir şablon, bir kalıp gibi kullanabileceğimizden söz açmıştık. Bu durumda:
 - Bir sınıf grubunda bazı metotların mutlaka olmasını şart koşuyorsanız, bu metotları bir soyut üst sınıfta tanımlar ve söz konusu sınıfları ile bu soyut sınıf arasında kalıtım ilişkisi kurarsınız.
- Soyut sınıfların adı sağa yatık olarak yazılır ancak gösterimde sorun çıkarsa <<STEREOTYPE>> gösterimi.
 - <<...>>: Bir sembol anlamı dışında kullanılmışsa.

44

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT SINIFLAR



- Üst sınıfta bir yazdırma işleminin olması gerektiği biliniyor ama bu işin nasıl yapılacağı bilinmiyor.
- Bu nedenle PrintDriver nesneleri bir işimize yaramaz.
 - Sadece yazdırma biriktiricisinin (spooler) nasıl ilklendirileceğinin kodunu yazma yükümlülüğünü üzerimizden alır.

- Yazdırma işlemin nasıl yapılacağı alt sınıflarda tanımlanmıştır.
- Tasarımımız, PCL6 ve PS tiplerinden ve hatta ortaya çıkacak yeni yazdırma tiplerinden birden fazla sürücünün bir bilgisayarda kurulu olmasına ve hepsine ortak bir PrintDriver sınıfı üzerinden erişilmesine izin verir.

45

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT SINIFLAR

- Kaynak kodlar:

```
package ndk07;
public abstract class PrintDriver {
    public void initSpooler( ) {
        /* necessary codes*/
    }
    public abstract void print( Document doc );
}
```

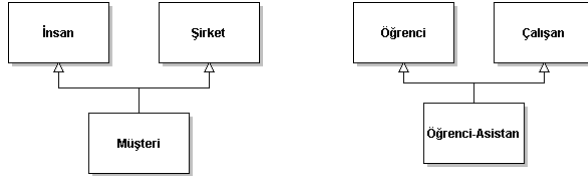
```
package ndk07;
public class PCL6Driver extends PrintDriver {
    public void print(Document doc) {
        //necessary code is inserted here
    }
}
```

46

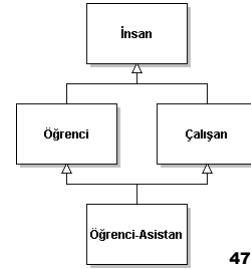
NYP İLE İLGİLİ ÖZEL KONULAR

ÇOKLU KALITIM

- Bir sınıfın birden fazla üst sınıftan kalıtım yolu ile türetilmesi.
- Alt sınıfın, birden fazla üst sınıfın özelliğini taşıması anlamına gelir.



- Çoklu kalıtım ile ilgili sorunlar:
 - Kalıtım çevrimi (Diamond problem): Orta düzeyde çokbüçümlilik varsa alt düzeyde çokbüçümlü metodun hangi sürümü çalışacak?
 - Her dil desteklemez. Ör: Java, C#

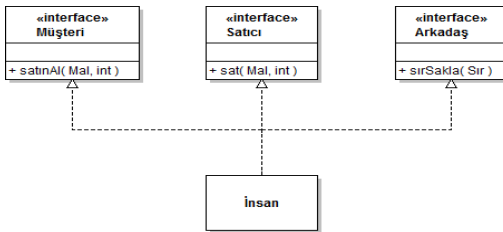


47

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

- Üye alanları olmayan ve tüm metodları soyut olan bir soyut sınıf gibi görülebilir.
 - Eğer isterseniz "public final static" üye alan ekleyebilirsiniz.
- Bir ad altında derlenmiş metodlar topluluğudur.
- Bir örnek üzerinden UML gösterimi:



- Bir sınıf, gerçeklediği arayüzdeki tanımlı tüm metodların gövdelerini tanımlamak zorundadır.

Kodlama:

```

public interface Müşteri {
    public void satınAl( Mal mal, int adet );
}
public class İnsan implements Müşteri,
    Satıcı, Arkadaş {
    public void satınAl( Mal mal, int adet ) {
        //ilgili kodlar
    }
    public void sat ( Mal mal, int adet ) {
        //ilgili kodlar
    }
    public void sırSakla( Sır birSır ) {
        //ilgili kodlar
    }
}

```

48

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

- Arayüzler neye yarayabilir?
 - Nesnenin sorumluluklarını gruplamaya.
 - Nesneye birden fazla bakış açısı kazandırmaya:
 - Farklı tür nesneler aynı nesneyi sadece kendilerini ilgilendiren açılardan ele alabilir.
 - Farklı tür nesneler aynı nesneye farklı yetkilerle ulaşabilir.
 - Kalıtımın yerine kullanılabilme:
 - Çünkü kalıtım "ağır sıklet" bir ilişkidir. Bu yüzden sadece çok gerektiğinde kullanılması önerilir.
 - Çoklu kalıtımın yerine kullanılabilme.

49

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

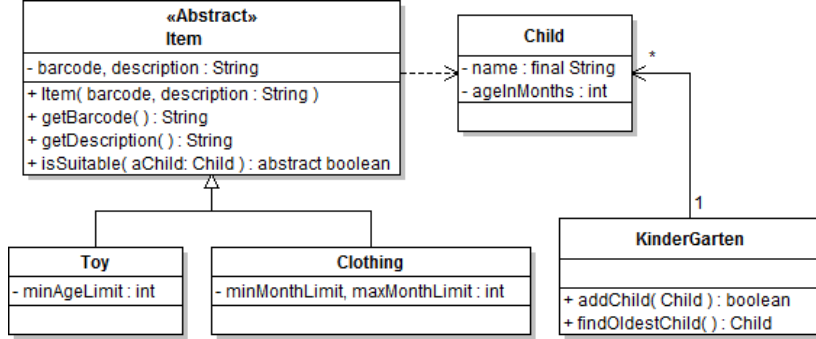
- Arayüzler ile ilgili kurallar:
 - Bir sınıf, gerçeklediği arayüzdeki tanımlı tüm metotların gövdelerini tanımlamak zorundadır.
 - Arayüzlerde normal üye alanlar tanımlanamaz, sadece "public final static" üye alanlar tanımlanabilir.
 - Arayüzlerde sadece public metotlar tanımlanabilir.
 - Arayüzülerin kurucusu olmaz.
 - Bir sınıf birden fazla arayüz gerçekleyebilir.

50

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Çocuk malzemelerini düşünün:
 - Her malzeme her yaştan çocuğa uygun değildir.
 - Oyuncakların ay türünden olmak üzere bir minimum yaş sınırı vardır.
 - Giysilerin ise yıl türünden minimum ve maksimum yaş sınırları vardır.
 - Bu durumu nasıl modellemeli?



51

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Item (malzeme) sınıfının kaynak kodu:

```
package ndk08;
public abstract class Item {
    private String barcode, description;
    public Item(String barcode, String description) {
        this.barcode = barcode;
        this.description = description;
    }
    public String getBarcode() {
        return barcode;
    }
    public String getDescription() {
        return description;
    }
    public abstract boolean isSuitable(Child aChild);
}
```

- Bir malzemenin uygunluğunun belirlenmesi için kullanılması gereken mantık farklı olduğu için, `isSuitable` (uygunMu) metodunu burada soyut tanımladık.
- Ancak her tür malzeme için ortak olan işlemleri bu soyut üst sınıfta kodladık ki bunları alt sınıflarda boş yere aynen tekrarlamak zorunda kalmayalım.

52

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Soyut olmayan alt sınıfların kaynak kodları:

```
package ndk08;
public class Clothing extends Item {
    private int minMonthLimit, maxMonthLimit;

    public Clothing(String barcode, String description,
        int minMonthLimit, int maxMonthLimit ) {
        super(barcode, description);
        this.minMonthLimit = minMonthLimit;
        this.maxMonthLimit = maxMonthLimit;
    }
    public boolean isSuitable(Child aChild) {
        if( aChild.getAgeInMonths() >= minMonthLimit
            && aChild.getAgeInMonths() <= maxMonthLimit )
            return true;
        return false;
    }
}
```

53

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Soyut olmayan alt sınıfların kaynak kodları:

```
package ndk08;
public class Toy extends Item {
    private int minAgeLimit;

    public Toy(String barcode, String description, int minAgeLimit) {
        super(barcode, description);
        this.minAgeLimit = minAgeLimit;
    }
    public boolean isSuitable(Child aChild) {
        if( aChild.getAgeInMonths()/12 >= minAgeLimit )
            return true;
        return false;
    }
}
```

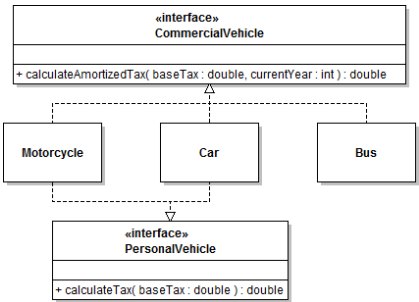
- Kindergarten (AnaOkulu) sınıfının kaynak kodunu UML sınıfında verildiği kadarıyla kodlayıp yapılan tasarımı yeni özelliklerle geliştirme işini alıştırma olarak yapabilirsiniz.

54

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Araçların vergilendirilmesi ile ilgili olarak şu gereksinimler verilmiştir:
 - Ticari ve şahsi araçlar farklı şekilde vergilendirilir.
 - Motosikletler, arabalar ve otobüsler ticari araç olarak kayıt edilebilir.
 - Sadece motosikletler ve arabalar şahsi araç olarak kayıt edilebilir.
 - Sadece ticari araçların vergilerinden amortisman düşülebilir.
 - Ticari veya şahsi olmalarından bağımsız olarak farklı tür araçların vergilendirilmesi farklıdır.
- Bu gereksinimleri nasıl modelleyebiliriz?



- Not: Eğer farklı tür araçların vergilendirilmesi benzer olsaydı, arayüz yerine önceki örnekteki gibi soyut üst sınıf kullanımı daha doğru olurdu.

55

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Arayüzlerin kodlanması:

```
package ndk08;
public interface CommercialVehicle {
    public double calculateAmortizedTax( double baseTax, int currentYear );
}
```

```
package ndk08;
public interface PersonalVehicle {
    public double calculateTax( double baseTax );
}
```

56

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Araba sınıfının kodlanması

```
package ndk08;
public class Car implements CommercialVehicle, PersonalVehicle {
    private int modelYear;
    private double engineVolume;
    public Car(int modelYear, double engineVolume) {
        this.modelYear = modelYear;
        this.engineVolume = engineVolume;
    }
    public double calculateTax( double baseTax ) {
        return baseTax * engineVolume;
    }
    public double calculateAmortizedTax( double baseTax, int currentYear ) {
        //Tax can be reduced %10 for each year as amortization
        int age = currentYear - modelYear;
        if( age < 10 )
            return baseTax * engineVolume * (1-age*0.10);
        return baseTax * engineVolume * 0.10;
    }
    public int getModelYear() { return modelYear; }
    public double getEngineVolume() { return engineVolume; }
}
```

57


NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Otobüs sınıfının kodlanması

```
package ndk08;
public class Bus implements CommercialVehicle {
    private int modelYear;
    private double tonnage;
    public Bus(int modelYear, double tonnage) {
        this.modelYear = modelYear; this.tonnage = tonnage;
    }
    public double calculateAmortizedTax( double baseTax, int currentYear ) {
        double ratioT, ratioA;
        if( tonnage < 1.0 )
            ratioT = 1.0;
        else if( tonnage < 5.0 )
            ratioT = 1.2;
        else if( tonnage < 10.0 )
            ratioT = 1.4;
        else
            ratioT = 1.6;
        ratioA = (currentYear - modelYear) * 0.05;
        if( ratioA > 2.0 )
            ratioA = 2.0;
        return baseTax * ratioT * ratioA;
    }
    public int getModelYear() { return modelYear; }
    public double getEngineVolume() { return tonnage; }
}
```

58



NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER İLE SOYUT SINIFLAR ARASINDA TERCİH YAPMAK

- Eğer farklı tür araçların vergilendirilmesi benzer olsaydı, yani aynı formülde farklı katsayılar kullanılarak hesaplanabilseydi (parametrize edilebilseydi) arayüzler yerine iki soyut üst sınıf kullanımı daha doğru olurdu.
- Benzer şekilde, ticari ve şahsi araçların vergilendirilmesi parametrize edilebilseydi, sadece bir soyut üst sınıf tanımlayıp uygun metot parametrelerinin seçimi daha doğru olurdu.
- Bu durumlar alıştırma olarak sizlere bırakılmıştır.

59