NESNEYE DAYALI KAVRAMLAR VE PROGRAMLAMA
Eylül 2013
Yrd.Doç.Dr. Yunus Emre SELÇUK

GENEL BİLGİLER

BAŞARIM DEĞERLENDİRME

- 1. Ara Sınav: %20,
- 2. Ara Sınav: %20,
- Final Sınavı: %35,
- Proje ödevi: %15,
- Kısa ödevler: %10

- Tarihler ve içerik daha sonra duyurulacaktır.
- Verilemeyen ödev olursa yüzdesi diğer notlandırma araçlarına dağıtılacaktır.

KAYNAKLAR:

- Java Programlama:
  - Java How to Program, Harvey M. Deitel & Paul J. Deitel, Prentice-Hall.
    - 7th ed. veya daha yenisi
  - Core Java 2 Volume I-II, C. S. Horstmann and G. Cornell, Prentice-Hall.
    - 7th ed. veya daha yenisi
- UML:
  - UML Distilled, 3rd ed. (2003), Martin Fowler, Addison-Wesley.

1

GENEL BİLGİLER

DERS İÇERİĞİ

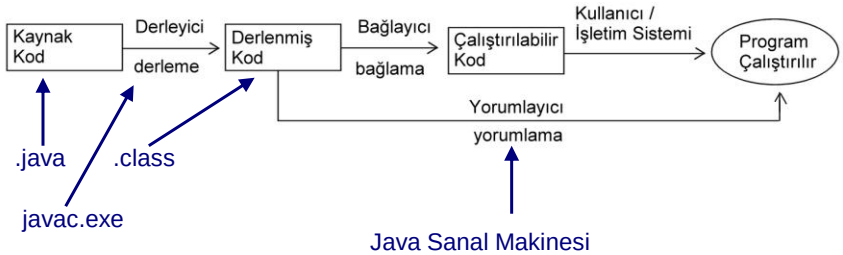
- Java programlama diline genel bakış
- Nesne ve Sınıf Kavramları
- UML Sınıf Şemaları
- Nesne Davranışı ve Metotlar
- Nesne ve Sınıfların Etkileşimleri ve İlişkileri
- UML Etkileşim (Sıralama) Şemaları
- Kalıtım ve Soyut Sınıflar
- Nesne Arayüzleri ve Çoklu Kalıtım
- Çokbiçimlilik, Metotların Yeniden Tanımlanması ve Çoklu Tanımlanması

2

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

JAVA UYGULAMA ORTAMI

- Java yorumlanan bir dildir.



Java Sanal Makinesi
(java.exe, javaw.exe)
(JVM: Java Virtual Machine)
JRE ile gelir
(Java Runtime Environment)

3

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

JAVA UYGULAMA ORTAMI

- Standart Sürüm – Standard Edition:
  - Masaüstü ve sunucu bilgisayarlarda çalışabilecek uygulamalar geliştirmeye yönelik.
- Mikro Sürüm – Micro Edition:
  - Cep telefonu ve avuç içi bilgisayarları gibi taşınabilir cihazlara yönelik.
  - Standart sürümündeki bileşenlerin bir kısmını daha az işlevsellikle içerir.
- Şirket Sürümü – Enterprise Edition:
  - Çok katmanlı uygulamalar ile web hizmetleri uygulamalarını kullanıma açmak için gerekli hizmet yazılımını içerir.
    - Sun Java System Application Server
    - IBM Websphere
    - BEA WebLogic
    - Apache Tomcat
    - …

4

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

JAVA SÜRÜMLERİ

- Eski ve yeni sürümlendirme:

Eski Sürüm (Developer Version)	Yeni Sürüm (Product Version)
Java 1.0	
Java 1.1	
Java 1.2	Java 2 Platform
Java 1.3	Java 2 SE 3 (J2SE3)
Java 1.4	J2SE4
Java 1.5	J2SE5
Java 1.6 {Sun}	Java Platform Standard Edition, version 6 (JSE6)
Java 1.7 {Oracle}	Java Platform Standard Edition, version 7 (JSE7)

5

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

JAVA SÜRÜMLERİ

- Eski sürümlendirme ayrıntıları:
  - JDK 1.7.0.20:
    - Java 2, Version 7.0, update 20.
    - Update:
      - Hata düzeltme, daha iyi başarım ve güvenlik nedenleriyle güncellemeler.
      - Birkaç aylık aralıklarla.
- Nereden indirmeli?
  - Oracle.com/java
  - Dokümantasyonu da ayrıca indirip açınız.

6

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

ÜCRETSİZ JAVA GELİŞTİRME ORTAMLARI

- Eclipse: http://www.eclipse.org
  - Ayrıca indirilir.
  - UML için eUML2 plug-in'i kurulmalı.
  - GUI için ayrı plug-in kurulmalı.
  - Yönetici olarak kurulum gerekmiyor, unzip yetiyor.
- NetBeans:
  - JSE dağıtımı ile birlikte (seçimlik)
  - UML için ayrı plug-in gerek.
    - Kuran bana da isim söylesin.
  - Dahili GUI editörü var.
  - Yönetici olarak kurulum gerektiriyor.

ÜCRETSİZ UML MODELLEME ORTAMLARI

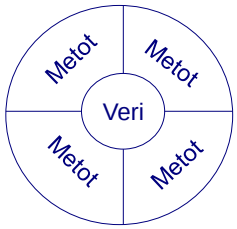
- Violet UML: Hafif sıklet, bizim için yeterli
- Argo UML

7

SINIFLAR, NESNELER VE ÜYELER

NESNE

- Nesne: Nitelikler ve tanımlı eylemler içeren, temel programlama birimi.
  - Nesne ≈ bir gerçek dünya varlığına denk gelir.
    - Nesneler değişkenlere de benzetilebilir
    - Süpermen de sıradan insanlara benzetilebilir!
  - Nesnenin nitelikleri ≈ Nesne ile ilgili veriler.
  - Eylemler ≈ Nesnenin kendi verisi üzerinde(*) yapılan işlemler ≈ Nesnenin kendi verileri ile çalışan metotlar (fonksiyonlar).
    - * Çeşitli kurallar çerçevesinde aksi belirtilmedikçe.
    - Metotlara parametre(ler) de verilebilir.
  - Sarma (Encapsulation): Veri ve eylemlerin birlikteliği.
    - Verilere, eylemler üzerinden erişilir.



8

SINIFLAR, NESNELER VE ÜYELER

SINIF

- Sınıf: Nesneleri tanımlayan şablonlar.
  - Şablon = Program kodu.

```
class myClass {
   /*
       program kodu
   */
}
```

9

SINIFLAR, NESNELER VE ÜYELER

NESNELER VE SINIFLAR

- Örnek nesne: Bir otomobil.
  - Nitelikler: Modeli, plaka numarası, rengi, vb.
    - Niteliklerden birinin tekil tanımlayıcı olması sorgulama işlerimizi kolaylaştıracaktır.
  - Eylemler: Hareket etmek, plaka numarasını öğrenmek, satmak, vb.
- Örnek sınıf: Taşıt aracı.
  - Nitelikleri ve eylemleri tanımlayan program kodu.
- Gerçek dünya benzetimi:
  - Nesne: Bir varlık olarak bir otomobil.
  - Sınıf: Dilbilgisi açısından bir genel isim olarak taşıt aracı.
- Bir nesneye yönelik program içerisinde, istenildiği zaman herhangi bir sınıftan olan bir nesne oluşturulabilir.
- Aynı anda aynı sınıftan birden fazla nesne etkin olabilir.

10

SINIFLAR, NESNELER VE ÜYELER

NESNELER VE SINIFLAR

- Bir nesnenin nitelikleri iki (!) çeşit olabilir:
  - Tamsayı, karakter gibi tek bir birim bilgi içeren 'ilkel' veriler,
  - Aynı veya başka sınıftan olan nesneler.
    - Sonsuz sayıda farklı sınıf oluşturulabileceği için, 'iki çeşit' deyimi çok da doğru değil aslında.
- Nesnenin niteliklerinin bir kısmı ilkel, bir kısmı da başka nesneler olabilir.

11

SINIFLAR, NESNELER VE ÜYELER

TERMİNOLOJİ VE GÖSTERİM

- NYP Terminolojisi:
  - Veri: Üye alan (Member field) = Nitelik (attribute)
  - Durum bilgisi: Nesnenin belli bir andaki niteliklerinin durumları
  - Eylem: Metot (Member method)
  - Nesnenin (Sınıfın) üyeleri = Üye alanlar + üye metotlar
  - Sınıf = tür = tip.
  - S sınıfından oluşturulan n nesnesi = n nesnesi S sınıfının bir örneğidir (instance)
- UML Gösterimi:

Sınıf: Sınıf şemasında

Nesne: Etkileşim şemasında

12

SINIFLAR, NESNELER VE ÜYELER

TERMİNOLOJİ VE GÖSTERİM

- İki tür UML etkileşim şeması (interaction diagram) vardır:
  1. Sıralama şeması (Sequence diagram)
  2. İşbirliği şeması (Collaboration diagrams)
- Bu derste sıralama şemaları çizeceğiz.
  - "Etkileşim" bu tür şemaların özünü çok iyi tarif ediyor. O yüzden "sıralama" ve "etkileşim" terimlerini birbirlerinin yerine kullanabilirim.

13

SINIFLAR, NESNELER VE ÜYELER

HER NESNE FARKLI BİR BİREYDİR!

- Aynı türden nesneler bile birbirinden farklıdır:
  - Aynı tür niteliklere sahip olsalar da, söz konusu nesnelerin nitelikleri birbirinden farklıdır = Durum bilgileri birbirinden farklıdır.
  - Durum bilgileri aynı olsa bile, bilgisayarın belleğinde bu iki nesne farklı nesneler olarak ele alınacaktır.
  - Bu farklılığı sağlamak üzere, her nesne programcının ulaşamadığı bir tekil tanımlayıcıya (UID: unique identifier) sahiptir.
    - Hiçbir nesnenin tanımlayıcısı birbiri ile aynı olmayacaktır.
    - UID'yi JVM kotarır.
- Örnek: Sokaktaki arabalar.
  - Örnek nitelikler: Modeli, rengi.
  - Modelleri ve renkleri farklı olacaktır.
  - Aynı renk ve model iki araba görseniz bile, plakaları farklı olacaktır.
  - Plaka sahteciliği sonucu aynı plakaya sahip olsalar bile, bu iki araba birbirlerinden farklı varlıklardır.

14

SINIFLAR, NESNELER VE ÜYELER

HER NESNE FARKLI BİR BİREYDİR! (devam)

- Aynı tür bile olsa, her nesnenin durum bilgisi farklı olduğu için, aynı türden iki nesne bile aynı mesaja farklı yanıt verebilir.
  - Örnek: Bana adımı sorsanız "Yunus" derim, sizin aranızda kaç Yunus var?
  - Kaldı ki, nesneye mesaj gönderirken farklı parametreler de verebilirsiniz. Aynı nesneye aynı mesaj farklı parametre ile giderse, geri dönen yanıtlar da farklı olacaktır.
- Terminoloji: Aynı türden iki nesne, aynı mesaja farklı yanıtlar verir.

15

SINIFLAR, NESNELER VE ÜYELER

NESNELERE MESAJ GÖNDERME

- Bir nesneye neden mesaj gönderilir?
  - Ona bir iş yaptırmak için
  - Bir üyesine erişmek için.

ÜYELERE ERİŞİM

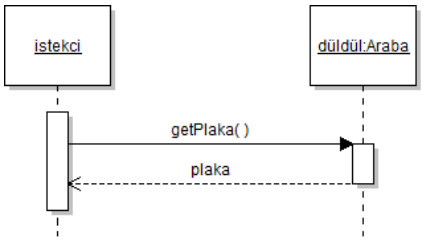
- Üye alana erişim:
  - Üyenin değerini değiştirmek (setting)
  - Üyenin değerini okumak (getting)
    - (Değiştirmeden herhangi bir işlemde kullanmak)
- Üye metoda erişim:
  - Bir eylemler sürecini varsa kendine özgü çalışma parametreleri ile yürütmek.
  - Fonksiyon çağırmak gibi, ama unutmayın: Aksi belirtilmedikçe metot, üyesi olduğu nesnenin üyeleri ile çalışır.
    - Aksinin nasıl belirtileceğini ileride nesneler arasındaki ilişkileri öğrenince göreceksiniz.

16

SINIFLAR, NESNELER VE ÜYELER

TERMİNOLOJİ VE GÖSTERİM

- NYP Terminolojisi:
  - Bir nesnenin diğer bir nesnenin bir üyesine erişmesi, bir nesnenin diğerine bir mesaj göndermesi olarak da tanımlanır.
  - Bir nesneye yönelik program, nesneler arasındaki mesaj akışları şeklinde yürür.
- UML Gösterimi:



- Kod Gösterimi:

```
duldul.plakaniVer();
//Nesne adını Türkçe veremiyoruz.
```

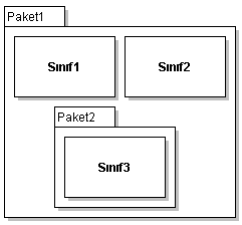
- Anlamı:
  - istekçi adlı bir nesne vardır.
  - istekçi nesnenin sınıfı belli değil.
  - düldül adlı bir nesne vardır.
  - düldül nesnesinin sınıfı Araba'dır.
  - Araba sınıfının plakaniVer adlı bir metodu vardır.
  - istekçi nesne düldül nesnesine plakaniVer mesajı gönderir.
  - düldül nesnesi bu mesaja yanıt olarak kendi plakasını döndürür.

17

SINIFLAR, NESNELER VE ÜYELER

PAKETLER

- Sınıflar paket (package) adı verilen mantıksal kümelere ayrılabilir.
- Amaç: Kendi içerisinde anlam bütünlüğü olan ve belli bir amaca yönelik olarak birlikte kullanılabilecek sınıfları bir araya toplamaktır.



- Bir paketteki sınıfları koda ekleme:

```
import paket1.Sınıf1;
import paket1.*;
import paket1.Paket2.*;
```

  - paket1 eklenince alt paketi olan paket2 içindeki sınıflar eklenmiş olmaz.
- Paketler, aynı adlı sınıfların birbirine karışmamasını da önler:
  - Sınıf adı aynı bile olsa farklı paketlerde bulunan sınıflar, belirsizlik oluşturmaz.

```
java.io.File
com.fileWizard.File
```

- Paket hiyerarşisi, aynı zamanda dosya hiyerarşisidir.

```
com.fileWizard.File -> com\fileWizard\File.java
```

18

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

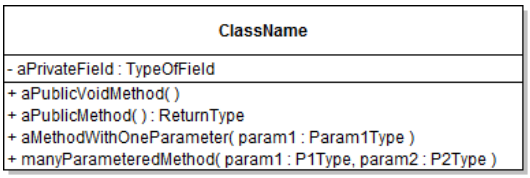
- Bir nesne, kendi sınıfından olan diğer nesnelerin ve bizzat kendisinin bütün üyelerine erişebilir,
- Ancak bir nesnenin üyelerine diğer sınıflardan olan nesnelerin erişmesi engellenebilir.
- Veri Gizleme İlkesi (information hiding):
  - İlke olarak, bir sınıfın içsel çalışması ile ilgili üyeler diğerlerinden gizlenir.
  - Böylece bir nesne diğerini kullanmak için, kullanacağı nesnenin ait olduğu sınıfın iç yapısını bilmek zorunda kalmaz.
- Örnek: TV çalıştırmak için uzaktan kumandalardaki ses ayarı, kanal değiştirme ve güç düğmelerinin evrensel işaretlerini tanımak yeterlidir;
  - Televizyonun içinde katot tüpü adlı bir cihaz olduğunu bilmek gerekmez.
  - Böylece LCD, plazma gibi yeni teknolojiler kullanıcıyı yeniden eğitmeye gerek kalmadan televizyonlarda kullanılabilir.
- Örnek: Arkadaşınız sizden borç para istedi.
  - Borç verirsiniz ya da vermezsiniz.
  - Arkadaşınızın sizin aylık gelirinizi, ATM kartınızın şifresini, vb. bilmesi gerekmez.

19

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

- Genel Görünebilirlik Kuralları (Visibility rules):
  - Public: Bu tip üyelere erişimde hiç bir kısıtlama yoktur.
  - Private: Bu tip üyelere başka sınıflardan nesneler erişemez, yalnız kendisi ve aynı türden olan diğer nesneler erişebilir.
- UML Gösterimi:



- Ayrıca (derste sorumlu değilsiniz):
  - protected: #
    - Kalıtım ile ilgili (paketteki diğer sınıflara ve alt sınıflarına açıktır)
  - package: ~
    - Paketteki diğer sınıflara açıktır
    - Java'da varsayılan kural

20

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

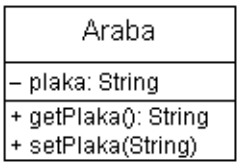
- Veri gizleme ilkesi her zaman için mükemmel olarak uygulanamaz.
  - Bir sınıftaki değişiklik sadece o sınıfı etkilemekle kalmaz, ilişkide bulunduğu başka sınıfları da etkileyebilir.
  - Veri gizleme ilkesine ne kadar sıkı uyulursa, değişikliğin diğer sınıflara yayılması olasılığı veya değişiklikten etkilenen sınıf sayısı da o kadar azalır.
- Veri gizleme ilkesine uyulmasını sağlamak için:
  - Üye alanlar private olarak tanımlanır, ve:
  - Değer atayıcı ve değer okuyucu metotlar kullanılır.
  - Bu ilkeye uymazsanız gitti en az 5 puan!
- Değer atayıcı ve değer okuyucu metotlar (erişim metotları):
  - Değer atayıcı (Setter) metot: Bir nesnenin bir üye alanına değer atamaya yarar.
  - Değer okuyucu (Getter) metot: Bir nesnenin bir üye alanının değerini öğrenmeye yarayan metot.
  - Adlandırma: getUye, setUye

21

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

- Örnek:



- Üyelere erişim kurallarında istenen değişiklikler kolaylıkla yerine getirilebilir.
  - Örneğin plaka içeriğinin okunması serbest olmakla birlikte bu alana değer atanmasının sadece ilgili paket içerisindeki sınıflar tarafından yapılması gerektiğinde, getPlaka metodu public olarak bırakılıp setPlaka metodu paket düzeyi görünebilirliğe alınır

22

SINIFLAR, NESNELER VE ÜYELER

ÜYELERİN ÖZEL DURUMLARI

- Statik üye alanlar:
  - Aynı türden nesnelerin bile durum bilgisi farklıdır (gördük), ancak:
  - Kimi zaman **aynı tipten tüm nesnelerin ortak bir üye alanı paylaşması** istenilebilir.
  - Bu durumda üye alan **`static`** olarak tanımlanır.
  - **SınıfAdı.üyeAdı** şeklinde sınıf üzerinden kullanılırlar, nesneler üzerinden kullanılmazlar.
  - Örnek: Her binek otomobilin 4 tekerleği vardır.
- Statik üye metotlar:
  - Aynı türden iki nesne, aynı mesaja farklı yanıtlar verir (gördük), ancak:
  - Kimi zaman **aynı tipten tüm nesnelerin aynı mesajın aynı şekilde çalışması** istenilebilir.
  - Bu durumda üye metot **`static`** olarak tanımlanır.
  - Statik metot içerisinde yalnız statik üyeler kullanılabilir.
  - Statik üye alana erişim metotları da statik tanımlanır.
  - **SınıfAdı.üyeAdı()** şeklinde kullanılırlar.

23

SINIFLAR, NESNELER VE ÜYELER

ÜYELERİN ÖZEL DURUMLARI

- Final üye alanlar:
  - Bir alanın değerinin sürekli olarak aynı kalması istenebilir.
  - Bu durumda üye alan **`final`** olarak tanımlanır.
  - Final üyelere yalnız bir kez değer atanabilir.
  - Örnek: Bir arabanın şasi numarası o araba fabrikadan çıkar çıkmaz verilir ve bir daha değiştirilemez.
- Final üye metotlar:
  - Sınırlı kullanım alanı: Kalıtım ile aktarılamazlar (ileride).

DİKKAT EDİLECEK NOKTALAR

- Bir üye, hem final hem de static olabilir.
- Tanımları ve adları gereği final ve static kavramları birbiriyle karıştırılabilir:
  - Final: Bir kez değer atama
  - Static: Ortak kullanım. Sözlük karşılığı durağan, ancak siz ortak diye düşünün.

24

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR VE SONLANDIRICILAR

- Kurucu Metot (Constructor):
  - Bir nesne oluşturulacağı zaman sınıfın kurucu adı verilen metodu çalıştırılır.
  - Nesnenin üyelerine ilk değerlerinin atanmasına yarar.
    - Bu yüzden ilklendirici metot olarak da adlandırılırlar.
  - Kurucu metotlara bu derste özel önem gösterilecektir.
- Sonlandırıcı metot:
  - Nesne yok edildiğinde JVM tarafından çalıştırılır.
  - Adı finalize'dır, parametre almaz, geri değer döndürmez.
  - C/C++ aksine, Java programcısının bellek yönetimi ile uğraşmasına gerek yoktur.
    - JVM için ayrılan bellek azalmaya başlamadıkça nesneler yok edilmez.
    - Bu yüzden bir nesnenin finalize metodunu çalıştırmak için çok çabalamanız gerekiyor!
  - Özetle:
    - Bu derste bu konu üzerinde daha fazla durulmayacaktır.

25

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Kurucu Metot kuralları:
  - Public görünülürlüğe sahip olmalıdır.
  - Kurucu metodun adı, sınıfın adı ile aynı olmalıdır.
  - Bir kurucu metodun geriye o sınıftan bir nesne döndürmesine rağmen,
    - Metot imzasında bir geri dönüş tipi belirtilmez,
    - Metot gövdesinde bir sonuç geri döndürme (return) komutu bulunmaz.
  - Final üyelere değer atamak için uygun bir yerdir.
    - Alternatif: Final üyeye tanımlandığı yerde değer atanması
  - Kod içerisinde bir nesne oluşturulacağı zaman ise, kurucu metot **new** anahtar kelimesi ile birlikte kullanılır.

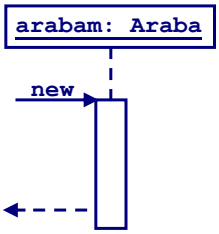
```
arabam = new Araba();
```

26

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Bir nesneyi kullanmak için onu tanımlamak yetmez, kurucusunu da çalıştırmak suretiyle onu ilklendirmek (*initialize, instantiate*) gerekir.
- UML Gösterimi:



- Kod gösterimi 1: Üye alan olarak kullanım

```
public class AClass {
   private Araba arabam;

   someMethod( ) {
       arabam = new Araba();
   }
}
```

- Kod gösterimi 2: Geçici değişken olarak kullanım

```
public class AnotherClass {
   someMethod( ) {
       Araba arabam = new Araba();
   }
}
```

27

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Varsayılan kurucu (default constructor):
  - Parametre almayan kurucudur.
  - Programcı tanımlamazsa, JVM (C++: Derleyici) tanımlar.
- Parametreli kurucular:
  - Üye alanlara parametreler ile alınan ilk değerleri atamak için kullanılır.
  - Bir tane bile parametreli kurucu tanımlanırsa, buna rağmen varsayılan kurucu tanımlanmamışsa, varsayılan kurucu kullanılamaz.
- Bir sınıfta birden fazla kurucu olabilir, ancak varsayılan kurucu bir tanedir.
  - Aynı üye aynı sınıf içinde birden fazla tanımlanamaz.
  - Aynı adı paylaşan ancak imzaları farklı olan birden fazla metot tanımlanabilir.
  - Bu tür metotlara **adaş metotlar**, bu yapılan işe ise adaş metot tanımlama (**overloading**) denir.

28

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

DENETİM AKIŞI

- Denetim akışı: Kodların yürütüldüğü sıra.
  - En alt düzeyde ele alındığı zaman bir bilgisayar programı, çeşitli komutların belli bir sıra ile yürütülmesinden oluşur.
  - Komutların peş peşe çalışması bir nehrin akışına benzetilebilir.
  - Komutların kod içerisinde veriliş sırası ile bu komutların yürütüldüğü sıra aynı olmayabilir.
  - Belli bir komut yürütülmeye başlandığı zaman ise o komut için denetimi ele almış denilebilir.
  - Bu benzetmelerden yola çıkarak, kodların yürütüldüğü sıraya denetim akışı adı verilebilir.

29

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

DENETİM AKIŞININ BAŞLANGICI

- Denetim akışının bir başlangıcının olması gereklidir.
  - Akışın hangi sınıftan başlatılacağını programcı belirler.
  - Java'da denetim akışının başlangıcı: Main komutu.
    - public static void main(String[ ] args)
      - static: Henüz bir nesne türetilmedi!
      - args dizisi: Programa komut satırından ilk parametreleri aktarmak için
  - Main metodunun görevi, gerekli ilk bir/birkaç nesneyi oluşturup programın çalışmasını başlatmaktır.
    - Hatırlayın, bir neseneye yönelik programın nesneler arasındaki mesajlar ile yürüdüğünü söylemiştik.
  - Bir sınıfın main metodunun olması, her zaman o metodun çalışacağı anlamına gelmez.
- Blok: Birden fazla komut içeren kod parçası.
  - Kıvrık parantez çifti içerisinde: { ve }

30

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- UML gösterimi (sınıf şeması)

- Önce UML sınıf şemasını çiz.
- Sonra kodda neresinin şemada nereye denk geldiğini işaretle.
- Pretty printing, camel casing …

- Kaynak kod (gerçekleme)

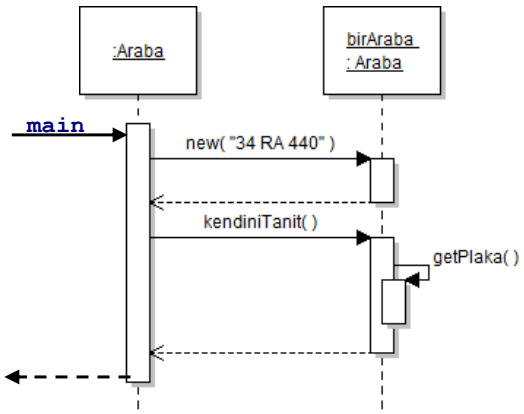
```
public class Araba {
   private String plaka;

   public Araba( String plakaNo ) {
      plaka = plakaNo;
   }
   public String getPlaka( ) {
      return plaka;
   }
   public void setPlaka( String plaka ) {
      this.plaka = plaka;
   }
   public void kendiniTanit( ) {
      System.out.println( "Plakam: " + getPlaka() );
   }

   public static void main( String[] args ) {
      Araba birAraba;
      birAraba = new Araba( "34 RA 440" );
      birAraba.kendiniTanit( );
   }
}
```

31

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

UML ŞEMASI İLE GÖSTERİM

- Örnek koddaki main metodunun, Etkileşim (Interaction) şeması türü olan sıralama şeması (sequence) ile gösterimi.
  - Okların düşeydeki sıralamasına azami dikkat ediniz !

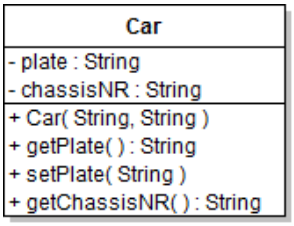


32

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- Araba sınıfının başka bir versiyonu:




```
public class Car {
   private String plate;
   private String chassisNR;
   public Car( String plateNr, String chassisNR ) {
      plate = plateNr;
      this.chassisNR = chassisNR;
   }
   public String getPlate() {
      return plate;
   }
   public void setPlate(String plate) {
      this.plate = plate;
   }
   public String getChassisNR( ) {
      return chassisNR;
   }
}
```

- Araba sınıfının bu versiyonunda bir main metodu yoktur. Bu nedenle doğrudan çalıştırılıp sınanamaz.
- Bu amaçla main metoduna sahip başka bir sınıf kodlamalı ve Car sınıfını oradan test etmeliyiz (ileride gösterilecek).

33

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- Kurucu metotlara özel önem gösterilmelidir:
  - Gerçek dünyada her aracın bir plakası VE bir şasi numarası bulunur.
  - Bu nedenle iki veri de kurucuda ilklendirilmelidir.
  - Böyle bir kurucunun (en az) iki parametresi olacağı barizdir.
  - Buna göre soldaki kod doğru, sağdaki hatalıdır (buggy).

```
public class Car {
   private String plate;
   private String chassisNR;
   public Car( String plateNr,
               String chassisNR ) {
      plate = plateNr;
      this.chassisNR = chassisNR;
   }
   /* Rest of the code */
}
```

```
public class Car {
   private String plate;
   private String chassisNR;
   public Car( String plateNr ) {
      plate = plateNr;
   }
   public Car(String chassisNR ) {
      this.chassisNR = chassisNR;
   }
   /* Rest of the code */
}
```

- Hata türleri:
  - Derleme hatası: Kod derleme aşamasında hata verir (derlenmez). Bu nedenle hiç çalıştırılamaz bile.
  - Bug: Kod derler ve çalışır, ancak hatalı sonuçlar üretir, yanlış davranır, vb.
  - Gerçek hayatta bir aracın şasi numarası asla değişmeyeceğinden, sınıfın bu üyesi için bir setter metodu kodlamak da bir bug olacaktır.

34

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

İLKEL VERİ TİPLERİ

Ad	Anlam	Aralık
int	Tam sayı (4 sekizlik)	– 2.147.483.648 ile + 2.147.483.647 arası (± 2 milyar)
double	Büyük ve hassas ondalıklı sayı	**(± 1,7 E 308)** (Büyük sayılar ve daha hassas işlem için)
float	Küçük ondalıklı sayı	**(± 1,7 E 38)** (Bellek tasarrufu ve daha hızlı işlem için)
boolean	Mantıksal	false – true

- İlkel: Bir birim bilgiyi ifade eden temel veri tipi
- Değişken: Bir ilkeli barındıran saklama alanları
- Nesneler için olduğu gibi; bir ilkeli kullanmadan önce o ilkeli tanımlamak gerekir.
  - int i = 7;
  - İlkeller, ilk değer atanmadan da kullanılabilir.
    - Değer atanmazsa ilk değerleri 0/false olur.
- Sayılarda ondalık ayıracına dikkat!
- boolean: Bayrak değişkeni.

35

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

İLKEL VERİ TİPLERİ

- İlkeller ile işlemler:
  - Aritmetik: + - * / %.
    - İşlem önceliği
    - ++, --, (Bir arttırma ve bir azaltma)
    - ++i ile i++ farkı
    - Atama ve işlem: += –= *= /= %=
    - Anlaşılabilirlik için işi sade tutun, abartmayın
  - Mantıksal: & | ! vb.
- Özetle, önceki programlama derslerinizden öğrendiğiniz gibi.

36

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

STRING SINIFI

- String sınıfı metotları
  - int length()
  - int compareTo(String anotherString)
  - int compareToIgnoreCase(String str)
- System.out.println(String)
  - print / println
- Örnek:

```
package test;
public class StringOps {
   public static void main( String args[] ) {
       String strA, strB;
       strA = "A string!";
       strB = "This is another one.";
       System.out.println(strA.compareTo(strB));
   }
}
```

- Örneğin çıktısı: −19

37

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

KOMUT SATIRI ÜZERİNDEN G/Ç

- System.out nesnesi ile çıktı işlemleri:
  - System sınıfının out üyesi public ve statiktir
    - Bu yüzden out nesnesi doğrudan kullanılabilir.
  - Komut satırına çıktı almak için metotlar:
    - printLn, print: Gördük
    - printf: C kullanıcılarının alıştığı şekilde kullanım.

38

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

KOMUT SATIRI ÜZERİNDEN G/Ç

- java.util.Scanner sınıfı ile giriş işlemleri: JDK 5.0 ile!
  - Oluşturma: Scanner in = new Scanner(System.in);
  - System.in : java.io.InputStream türünden public static üye.
  - Tek tek bilgi girişi için metotlar:
    - String nextLine()
    - int nextInt()
    - float nextFloat()
    - …

```
import java.util.Scanner;
public class ConsoleIOv1 {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.print("What is your name? ");
      String name = in.nextLine();
      System.out.print("How old are you? ");
      int age = in.nextInt();
      System.out.println("Hello, " + name +
      ". Next year, you'll be " + (age + 1) + ".");
   }
}
```

39

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

KOMUT SATIRI ÜZERİNDEN G/Ç

- Scanner sınıfındaki bir hata:
  - nextInt, nextFloat, vb. ilkel okuma komutundan sonra bir karakter katarı okumak için nextLine kullanırsan sorun çıkıyor.
  - Çözmek için ilkel okumadan sonra bir boş nextLine ver.

```
import java.util.Scanner;
public class ConsoleIOv2 {
   public static void main(String[] args) {
      Scanner in = new Scanner(System.in);
      System.out.print("How old are you? ");
      int age = in.nextInt();
      in.nextLine(); //workaround for the bug
      System.out.print("What is your name? ");
      String name = in.nextLine();
      System.out.println("Hello, " + name +
      ". Next year, you'll be " + (age + 1) + ".");
   }
}
```

40

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

KOMUT SATIRI ÜZERİNDEN G/Ç

- Araba sınıfının main metodunu, araç plakasını kullanıcıdan alacak şekilde değiştirelim:


```
package ndk01;
import java.util.Scanner;
public class Araba {
   private String plaka;
   public Araba( String plakaNo ) {
   plaka = plakaNo;
   }
   public String getPlaka( ) {
      return plaka;
   }
   public void setPlaka( String plaka ) {
      this.plaka = plaka;
   }
   public void kendiniTanit( ) {
      System.out.println( "Plakam: " + getPlaka() );
   }

   public static void main( String[] args ) {
      Araba birAraba;
      Scanner input = new Scanner( System.in );
      System.out.print("Enter a license plate: ");
      String plakaNr =  input.nextLine( );
      birAraba = new Araba( plakaNr );
      birAraba.kendiniTanit( );
   }
}
```

41

DENETİM AKIŞI

- Temel programlama dillerinden bildiğiniz yapılar burada da benzer sözdizimi ile geçerli, bu derste artık üzerinde durmayacağız.
- Aşağıda kısa bir özet yer almaktadır.

KARAR VERME İŞLEMLERİ – IF DEYİMİ

```
if (koşul) {...} else if (koşul) {...} ... else (koşul) {...}
```

- Koşul kısmı hakkında:
  - Karşılaştırma: < > <= >= == !=
  - Mantıksal işlemlerde çift işleç kullanılır: && ||

DÖNGÜLER

```
for( baslangicIfadesi; devamIfadesi; artimIfadesi ) { ... }
while( kosul ) { ... }
do { ... } while( kosul );
switch / case ...
```

42

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.

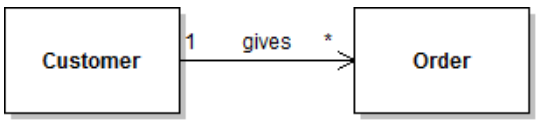
43

NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)

- Bağıntı 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.

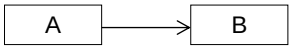
- Örnek: Müşteri ve siparişleri
  - İlişki adları ve nicelikleri de yazılabilir.

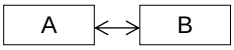


44

NESNELER ARASINDAKİ İLİŞKİLER

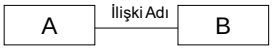
Sahiplik (Association)





Çift yönlü bağımlılık

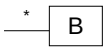
- Gösterim:



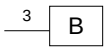
Sahiplik

A, B'ye bağımlı = b nesneleri a nesnelerinden habersiz = A kodunda b'nin public metotları çağrılabilir.

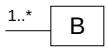
- Okun yönü önemli, kimin kime mesaj gönderebileceğini gösterir.
- Ok yoksa:
  - Ya çift yönlü bağımlılık vardır,
  - Ya da yazılım mimarı henüz bağımlılığın yönünü düşünmemiştir.
- İlişkinin uçlarında sayılar olabilir (cardinality)
  - Çoğulluk ifade eder.
  - İlişkinin o ucunda bulunan nesne sayısını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



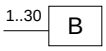
0 veya daha fazla



Tam 3 adet



1 veya daha fazla



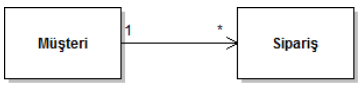
1'den 30'a kadar

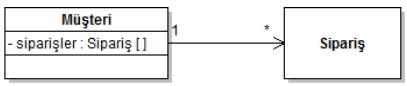
45

NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

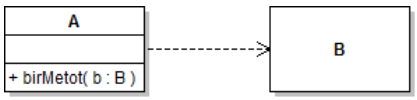
- Sahiplik ilişkisi çizgi ve oklarla gösterilmişse, sınıfların içerisinde ayrıntılı olarak gösterilmek zorunda değildir.
  - Ör: Sol alttaki şekil ile sağ alttaki şekil denktir.
  - Diğer ilişkiler için de aynı şey geçerlidir.





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 içinden b nesnesine mesaj gönderebilir.

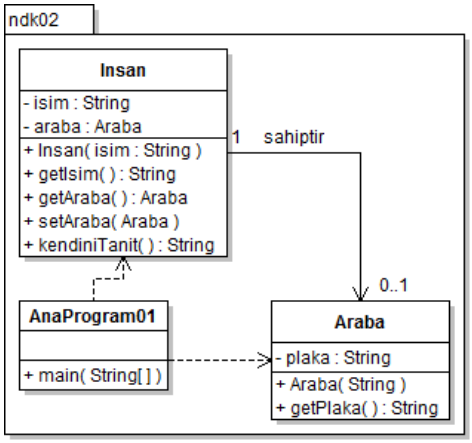
46

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Her insanın bir arabasının olduğu 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



47

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 Insan sınıfına:

48

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Insan sınıfının kaynak kodu:

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

   public Insan( 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 kendiniTanit( ) {
            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?)

49

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- UML sınıf şemamızda Insan - 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 çalıştırılacağının, hatta çalıştırılıp çalıştırılmayacağının 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 sınayalım, ona göre arabasının plakasına ulaşmaya çalışalım.
      - İnsanın arabası yokken, o üye alanın değeri **`null`** olmaktadır.
      - Yani o üye ilklendirilmemiştir.

50

NESNE İLİŞKİLERİNİN KODLANMASI

NESNENİN ETKİNLİĞİNİN SINANMASI

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

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

51

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Nihayet main metodu içeren uygulamamızı yazabiliriz:

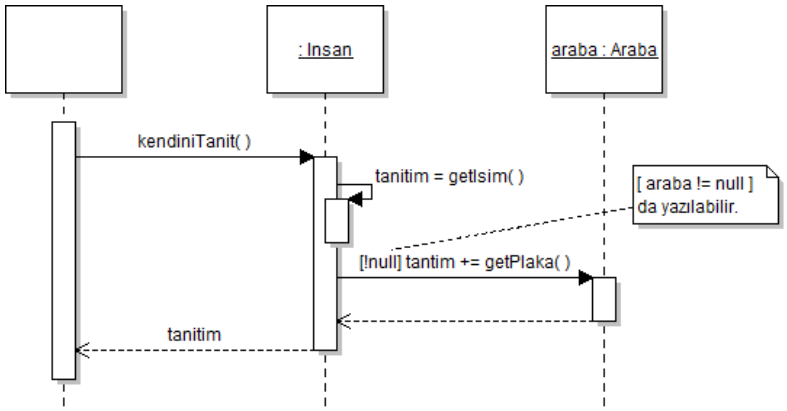
```
package ndk02;

public class AnaProgram01{

   public static void main(String[ ] args) {
            Insan oktay;
            oktay = new Insan("Oktay Sinanoğlu");
            Araba rover;
            rover = new Araba("06 RVR 06");
            oktay.setAraba(rover);
            Insan serkan = new Insan("Serkan Anılır");
            System.out.println( oktay.kendiniTanit() );
            System.out.println( serkan.kendiniTanit() );
   }
}
```

52

NESNE İLİŞKİLERİNİN KODLANMASI

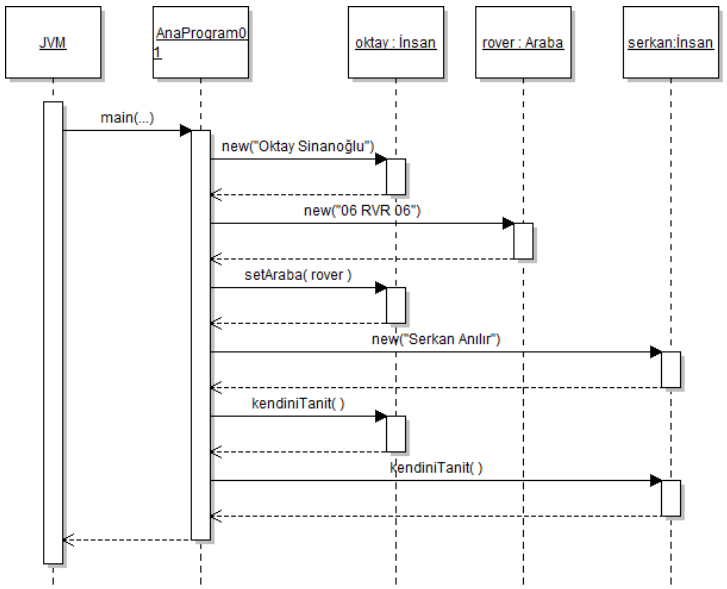
BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Insan sınıfının kendiniTanit metodunun etkileşim şeması:



53

- AnaProgram01 çalıştırılması ile ilgili etkileşim şeması :

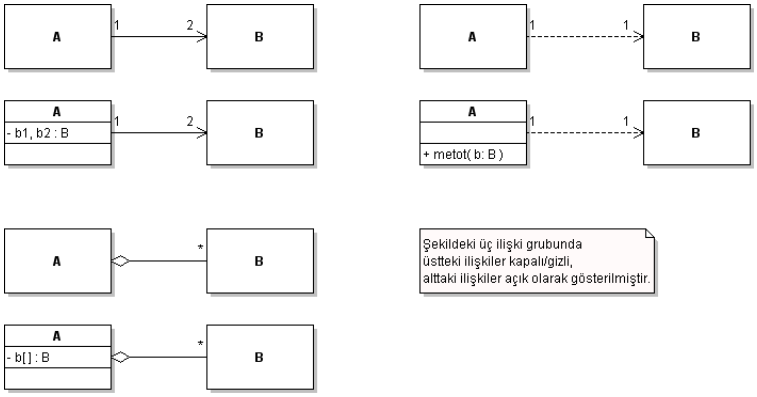


54

NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

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

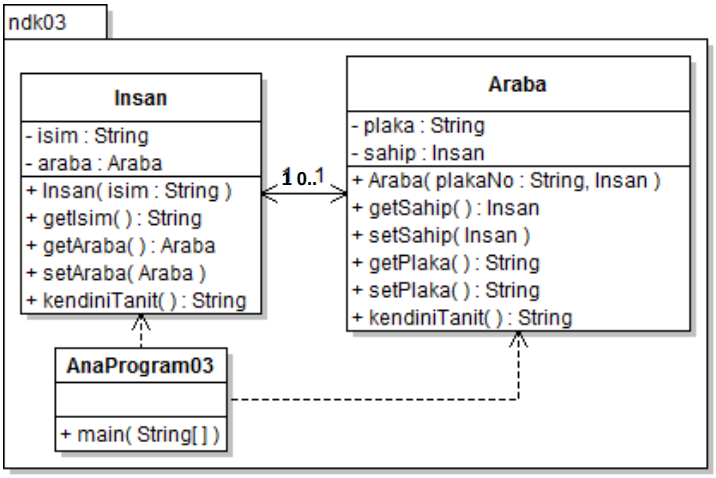


55

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Hem bir insanın arabası olabilir, hem de bir arabanın kimin arabası olduğunu bilmemiz gerekiyorsa, ilişki çift yönlü kurulmalıdır.



56

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Önceki şema ile farkları gördünüz mü?
  - Araba sınıfına sahip üye alanı ve şunlardan en az birini eklemek gerekti:
    - Sahip üyesi için set metodu ve/veya hem plaka hem sahibi alan yapılandırıcı.
    - Sahip üyesi için get metodu
  - Araba sınıfının kaynak kodunu değiştirmemiz gerektiği için paketini de değiştirdik.

57

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Araba sınıfının yeni kaynak kodu:

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

Attention!

58

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

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

59

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              Insan oktay = new Insan("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              Person serkan = new Person("Serkan Anılır");
12              Car honda = new Car("06 JAXA 73");
13              serkan.setCar(honda);
14              honda.setOwner(serkan);
15              System.out.println( serkan.introduceSelf() );
16              System.out.println( honda.introduceSelf() );
17     }
18  }
19
20
```

60

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) – (sinan, honda) ilişkilerini kurarken yanlışlıkla çapraz bağlantı kursak?
  - vb.
- Bu sorunların hepsi, çift yönlü ilişkiyi daha sağlam kurarak ortadan kaldırılabilir.
  - Nereyi değiştirmemiz lazım?

61

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Insan ve Araba sınıflarının değişen kısımları:

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

Dikkat!

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

Dikkat!

62

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

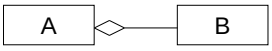
- Sonuç:
  - Çift yönlü bağıntı tek yönlü bağıntıya göre daha esnektir ancak kodlaması daha zordur.
  - Bu nedenle çift yönlü bağıntıya gerçekten ihtiyacınız yoksa kodlamayın.
  - Peki ya sonradan ihtiyaç duyarsak?
    - Şimdiden kodlamaya çalışıp zaman kaybetmeyin. Zaten yetiştirmeniz gereken bir dolu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

63

NESNELER ARASINDAKİ İLİŞKİLER

Toplama (Aggregation)

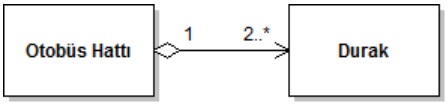
- **Parça-bütün ilişkisini** simgeler.
- Gösterim:



Toplama (aggregation)

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

- Şemada gösterilmese 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.

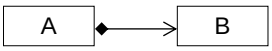


64

NESNELER ARASINDAKİ İLİŞKİLER

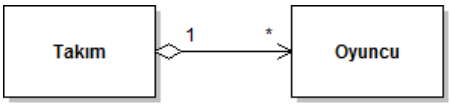
Meydana Gelme (Composition)

- 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:



65

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)

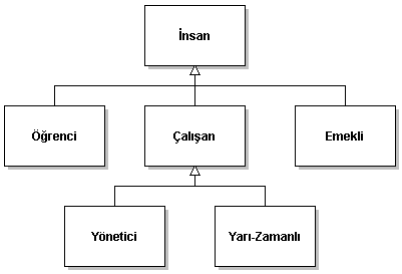
- Kalıtımın işleyişi:
  - Kalıtım yolu ile üst sınıftan alt sınıfa hem üye alanlar hem de üye metotlar aktarılır
    - private üyeler dahil, ancak alt sınıf onlara doğrudan ulaşamaz.
  - Protected üyeler ve kalıtım:
    - Alt sınıflar tarafından erişilir, diğer sınıflar tarafından erişilemez.

66

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 türetilebilir. Oluşan ağaç yapısına kalıtım hiyerarşisi veya kalıtım ağacı denir.



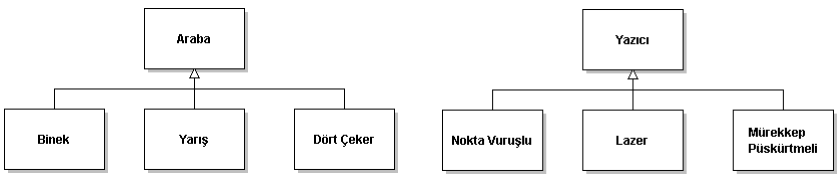
- Kalıtım ağacını çok derin tutmak doğru değildir (Kırılgan üst sınıf sorunu: Bina temelinin çürümesi gibi).

67

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.

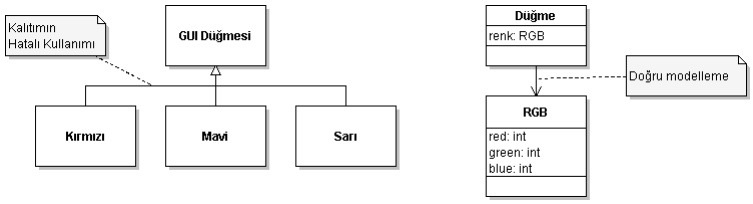


68

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtımın yanlış kullanımı:

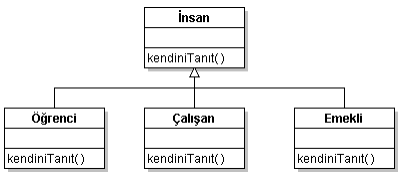


69

KALITIM İLE İLGİLİ ÖZEL KONULAR

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.

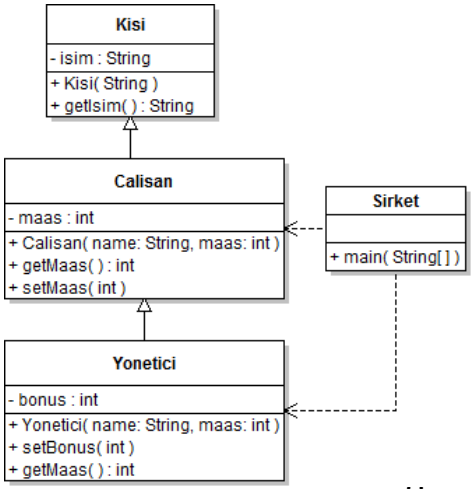
- İnsan türünden bir dizi düşünelim, elemanları İnsan ve alt sınıflarından karışık nesneler olsun. Dizinin tüm elemanlarına kendini tanıt dediğimizde ne olacak?
  - Çalışma anında doğru sınıfın metodu seçilir.
  - Bu çalışma biçimine de çok biçimlilik (polymorphism) denir.
- Peki, üst sınıfın altta yeniden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

70

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:

```
package ders04;
public class Kisi {
        private String isim;
        public Kisi( String name ) { this.isim = name; }
        public String getIsim( ) { return isim; }
}
```

```
package ders04;
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; }
}
```

72

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

```
package ders04;

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;
      }

}
```

DİKKAT: super.super olmaz!!!

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

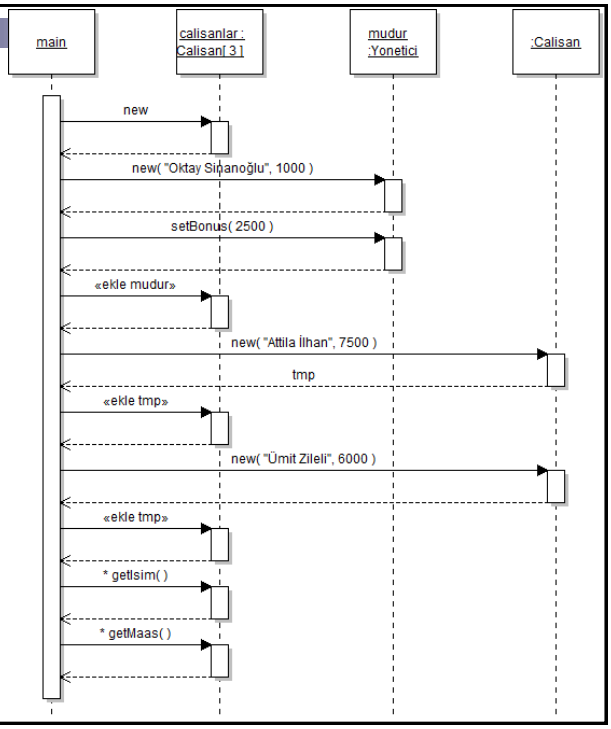
```
package ders04;
public class Sirket {
   public static void main(String[] args) {
            Calisan[] calisanlar = new Calisan[3];;
            Yonetici mudur = new Yonetici( "Oktay Sinanoğlu", 10000 );
            mudur.setBonus( 2500 );
            calisanlar[0] = mudur;
            calisanlar[1] = 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?

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

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

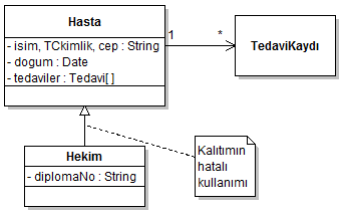
  - Böylece yapılandırıcıya çoklu anlam yüklemiş olduk.
  - Bu kez de bu yapılandırıcıyı kullanacak kişi, maaş ile bonus'u birbirine karıştırmamalı.
- DİKKAT: Çoklu anlam yüklemenin kalıtımla bir ilgisi yoktur. Kalıtım olmadan da adaş metotlar oluşturulabilir, ancak kalıtım olmadan çok biçimlilik ve yeniden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

76

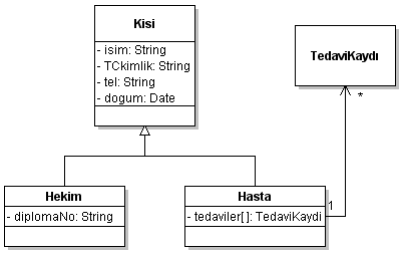
NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Gereksinim:
  - Hastaların isimleri, TC kimlik no.ları, doğum tarihleri ve cep telefonları saklanmalıdır. Bu bilgiler diş 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.

77

NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

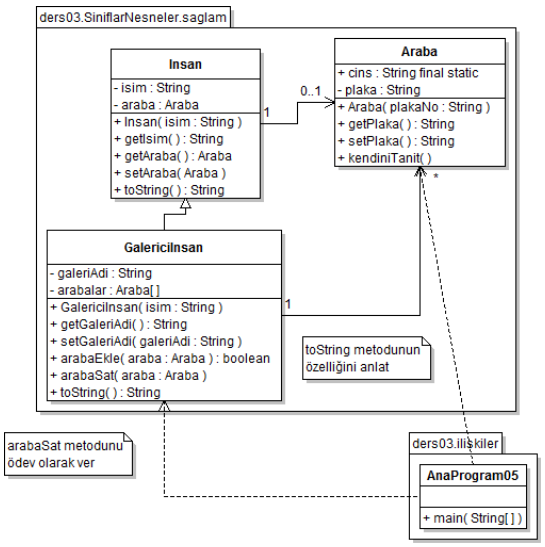
- Örnek: Birden fazla arabası olan araba koleksiyoncusu insanları modellemek için GalericiInsan adlı bir sınıf oluşturalım.
  - Daha önce yazdığımız Araba sınıfını aynen kullanabiliriz.
  - Bir insan galerici de olsa, öğretmen de olsa, öğrenci de olsa bir insandır.
    - Bu nedenle GalericiInsan sınıfını Insan sınıfından kalıtımla türetelim.
  - Bir GalericiInsan nesnesinin birden fazla arabası olabileceğinden toplama veya 1..* sahiplik ilişkisi ile gösterim yapabiliriz.
    - GalericiInsan adı toplama ilişkisine daha uygun.
    - Sınıfın adı ZenginInsan olsaydı 1..* sahiplik ilişkisi şeklinde gösterim daha uygun olurdu.
    - Bu tartışma “Melekler erkek midir, dişi mi?” tartışmasına benzemeden UML şemasına ve koda geçelim.
    - Bu sırada Java’da dizilerin kullanımını ve for döngüsünü de görmüş olacağız.

78

NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

- UML sınıf şeması:



79

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.

80

NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

- GalericiInsan sınıfının kaynak kodu:

```
package ders03.SiniflarNesneler.saglam;
public class GalericiInsan extends Insan {
   private String galeriAdi;
   private Araba[] arabalar;
   public final static int maxAraba = 30;
   private int arabaSayisi;

   public GalericiInsan( String isim ) {
            super( isim );
            arabaSayisi = 0;
            arabalar = new Araba[maxAraba];
   }
   public String getGaleriAdi() { return galeriAdi; }
   public void setGaleriAdi(String galeriAdi) { this.galeriAdi = galeriAdi; }
   public String toString( ) {
            String tanitim = super.toString(); //çokbiçimliliği hatırladınız mı?
            tanitim = "Merhaba, adım: " + getIsim();
            tanitim += "\nÜstelik " + galeriAdi + " adlı bir araba galerim var.";
            tanitim += "\nGalerimde " + arabaSayisi + " adet araba var.";
            return tanitim;
   }
   //devamı var…
```

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

NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

- GalericiInsan sınıfının kaynak kodunun devamı:

```
   public boolean arabaEkle( Araba araba ) {
            if( arabaSayisi < maxAraba ) {
                     arabalar[ arabaSayisi ] = araba;
                     arabaSayisi++;
                     return true;
            }
            else return false;
   }
} //end class
```

Alıştırma: Burada önce araba zaten eklenmiş mi diye, ayrı bir metot yardımı ile bir denetleme yapınız.

- Alıştırma: Peki araba satışı nasıl olacak? arabaSat metodunu kodlayınız.
- Alıştırma: Olmayan bir arabayla işlem yapılmasını önleyecek şekilde koda eklentiler yapınız.
- Alıştırma: İşin içine para meseleleri girseydi ne olacaktı?

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.

83

NYP İLE İLGİLİ ÖZEL KONULAR

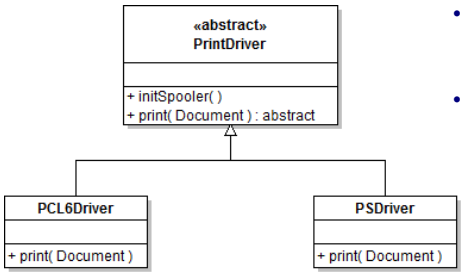
SOYUT SINIFLAR

- 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.

84

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.

85

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT SINIFLAR

- Kaynak kodlar:

```
package ooc06;
public abstract class PrintDriver {
    public void initSpooler( ) {
       /* necessary codes*/
    }
    public abstract void print( Document doc );
}
```

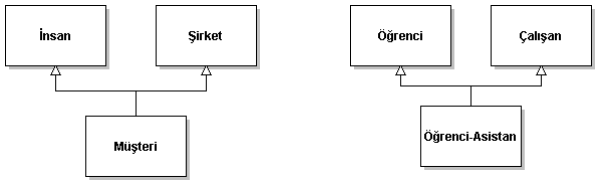
```
package ooc06;
public class PCL6Driver extends PrintDriver {
    public void print(Document doc) {
       //necessary code is inserted here
    }
}
```

86

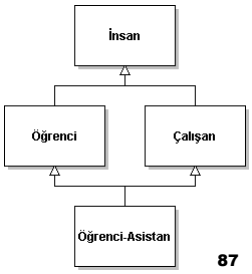
NYP İLE İLGİLİ ÖZEL KONULAR

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 çokbiçimlilik varsa alt düzeyde çokbiçimli metodun hangi sürümü çalışacak?
  - Her dil desteklemez. Ör: Java, C#

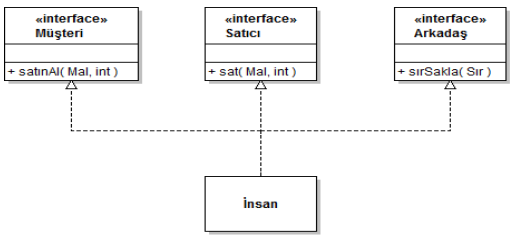


87

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

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



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

- Kodlama:

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

88

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.

89

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

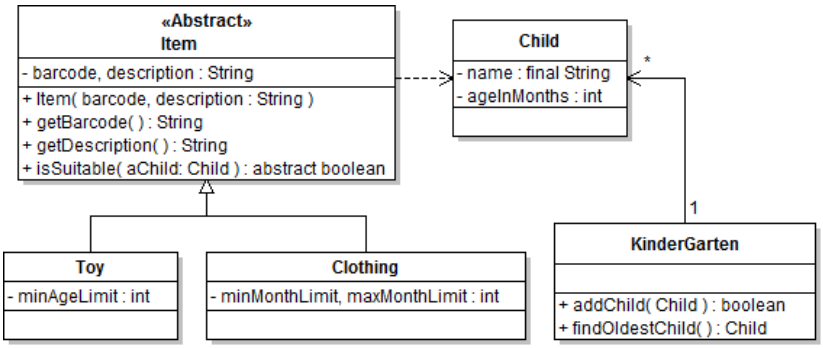
- Arayüzler ile ilgili kurallar:
  - Bir sınıf, gerçeklediği arayüzdeki tanımlı tüm metotların gövdelerini tanımlamak zorundadır.
  - Arayüzlerde normal üye alanlar tanımlanamaz, sadece "public final static" üye alanlar tanımlanabilir.
  - Arayüzlerde sadece public metotlar tanımlanabilir.
  - Arayüzülerin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

90

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?



91

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Item (malzeme) sınıfının kaynak kodu:

```
package ooc06;
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.

92

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Soyut olmayan alt sınıfların kaynak kodları:

```
package ooc06;
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;
    }
}
```

93

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT BİR SINIF TASARLAMAK VE KODLAMAK

- Soyut olmayan alt sınıfların kaynak kodları:

```
package ooc06;
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;
    }
}
```

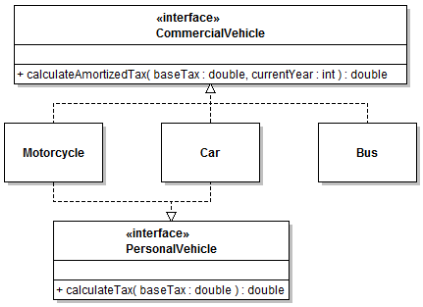
- Kindergarten (AnaOkulu) sınıfının kaynak kodunu UML sınıfında verildiği kadarıyla kodlayıp yapılan tasarımı yeni özelliklerle geliştirme işini alıştırma olarak yapabilirsiniz.

94

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.

95

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Arayüzlerin kodlanması:

```
package ooc07;
public interface CommercialVehicle {
    public double calculateAmortizedTax( double baseTax, int currentYear );
}
```

```
package ooc07;
public interface PersonalVehicle {
    public double calculateTax( double baseTax );
}
```

96

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Araba sınıfının kodlanması

```
package ooc07;
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; }
}
```

97

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Otobüs sınıfının kodlanması

```
package ooc07;
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; }
}
```

98

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.

99