BLM5219 NESNEYE DAYALI KAVRAMLAR VE PROGRAMLAMA
Ekim 2024
Dr.Öğr.Üyesi Yunus Emre SELÇUK

GENEL BİLGİLER

BAŞARIM DEĞERLENDİRME

- 1. Ara Sınav: %30, 22 Kasım 2024 Cuma, klasik
- 2. Ara Sınav: %30, 20 Aralık 2024 Cuma ???
- Ara sınav telafisi: 10 Ocak 2024 Cuma ???
- Final Sınavı: %40, Final programı sonradan duyurulacak
- Vize haftaları diğer derslerinizin durumlarına göre değiştirilebilir.
- Sınavlar yüz yüze yapılacaktır.

KAYNAKLAR:

- Java Programlama (Nesne Yönelimli):
  - Java How to Program, Deitel & Deitel, Prentice-Hall. (≥ 9th ed. Early Objects Version)
  - Core Java 2 Vol. I, Horstmann & Cornell, Prentice-Hall. (≥ 7th ed.)
- Java Programlama (Yapısal):
  - Algoritma Geliştirme ve Programlamaya Giriş, Fahri Vatansever, Seçkin Y.
- UML:
  - UML Distilled, 3rd ed. (2003), Martin Fowler, Addison-Wesley.

1

Bu yansı ders notlarının düzeni için boş bırakılmıştır.

2

GENEL BİLGİLER

DERS İÇERİĞİ

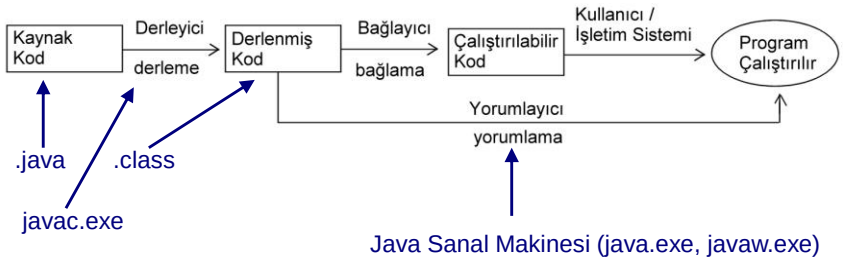
- Temel içerik:
  - Java programlama diline genel bakış
  - Nesne ve Sınıf Kavramları
  - UML Sınıf Şemaları
  - Nesne Davranışı ve Metotlar, Metotların Çoklu Tanımlanması
  - Nesne ve Sınıfların Etkileşimleri ve İlişkileri
  - UML Etkileşim (Sıralama) Şemaları
  - Kalıtım ve Soyut Sınıflar
  - Çokbiçimlilik, Metotların Yeniden Tanımlanması
  - Nesne Arayüzleri
- Öğrenciler ders notlarında yer alan sıralama şemalarından sınavlarda sorumlu olmayacaklardır. Görsel şemaların öğrenimi kolaylaştırması nedeniyle sıralama şemaları ile ilgili içerik ders notlarından çıkarılmamıştır.

3

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)

- JDK: Java Development Toolkit: Java kodu yazabilmek için gerekli araçlar topluluğu
- JRE: Java Runtime Environment: Derlenmiş Java kodunu kullanıcıların çalıştırabilmesi için gerekli araçlar topluluğu.
- JDK ⊃ JRE

4

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.
    - Eski: Sun Java System Application Server, Yeni: Java Application Server
    - IBM Websphere
    - BEA WebLogic
    - Apache Tomcat
    - …

5

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)
Java 1.8	Java Platform Standard Edition, version 8 (JSE8)

6

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

JAVA SÜRÜMLERİ

- Mevcut durum:

Yeni Sürüm	Açıklama
Java SE 8	Son halka açık güncelleme (32 bit eski sistemler için)
Java SE 9+	Ara sürümler, desteklenmeyebilir.
Java SE 21	LTS sürümü (Çıkış: Eylül 2023)
Java SE 23	Güncel sürüm (Çıkış: Eylül 2024)

7

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

JAVA SÜRÜMLERİ

- Sürümlendirme ayrıntıları:
  - JDK 11.0.9:
    - Java Version 11.0, update 9.
    - Version: Ana sürüm, 9.0'dan itibaren Eylül – Mart döngüsü ile yeni ana sürüm çıkıyor.
    - Update:
      - Hata düzeltme, daha iyi başarım ve güvenlik nedenleriyle güncellemeler.
      - Birkaç aylık aralıklarla.

- Nereden indirmeli?
  - Oracle.com/java
  - Dokümantasyonu da ayrıca indirip açınız.
  - JSE ⊃ JDK ⊃ JRE

8

JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

ÜCRETSİZ JAVA GELİŞTİRME ORTAMLARI

- Eclipse: http://www.eclipse.org
  - UML için eUML2 plug-in'i kurulmalı.
  - GUI için ayrı plug-in kurulmalı.
  - Yönetici olarak kurulum gerekmiyor, unzip yetiyor.
  - Yeni sürümlerinin sadece 64-bit desteği vardır, 32-bit eski sistemler için Eclipse IDE 2018-09 M3 kullanılabilir.
  - Kurulu JDK sürümünüzle projelerde kullanılan varsayılan sürümün aynı olmasına dikkat edin.
- NetBeans:
  - UML için ayrı plug-in gerek (Kuran bana da isim söylesin).
  - Dahili GUI editörü var.
  - Yönetici olarak kurulum gerektiriyor.
- Diğer: JCreator, IntelliJ, vb.

ÜCRETSİZ UML MODELLEME ORTAMLARI

- Violet UML (0.21.1): Hafif sıklet, bizim için yeterli (avesis.yildiz.edu.tr/yselcuk)
- Argo UML

9

BMH112 PROGRAMLAMA DİLLERİ
Dr.Öğr.Üyesi Yunus Emre SELÇUK

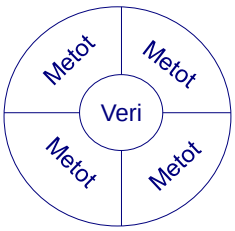
DERS NOTLARI:
A. NESNEYE YÖNELİMİN TEMELLERİ

10

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.



11

SINIFLAR, NESNELER VE ÜYELER

SINIF

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

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

12

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.

13

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.

14

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

15

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.

16

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 (UUID: Universally unique identifier) sahiptir.
    - Hiçbir nesnenin tanımlayıcısı birbiri ile aynı olmayacaktır.
    - UUID'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.

17

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.

18

SINIFLAR, NESNELER VE ÜYELER

NESNELERE MESAJ GÖNDERME

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

ÜYELERE ERİŞİM

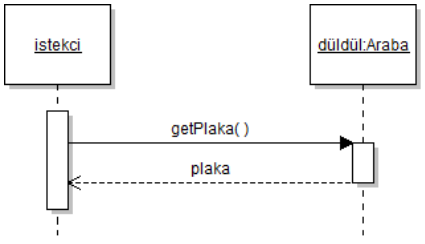
- Üye alana erişim:
  - Üyenin değerini değiştirmek (setting)
  - Üyenin değerini okumak (getting)
    - (Değiştirmeden herhangi bir işlemde kullanmak)
- Üye metoda erişim:
  - Bir eylemler sürecini varsa kendine özgü çalışma parametreleri ile yürütmek.
  - Fonksiyon çağırmak gibi, ama unutmayın: Aksi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

19

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.getPlaka();
//Nesne adını Türkçe veremiyoruz.
```

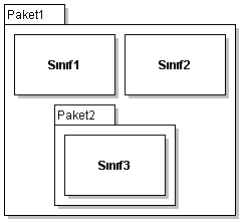
- Anlamı:
  - istekçi adlı bir nesne vardır.
  - istekçi nesnenin sınıfı belli değil.
  - düldül adlı bir nesne vardır.
  - düldül nesnesinin sınıfı Araba'dır.
  - Araba sınıfının getPlaka adlı bir metodu vardır.
  - istekçi nesne düldül nesnesine getPlaka mesajı gönderir.
  - düldül nesnesi bu mesaja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

20

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 karışmasını 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
```

21

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

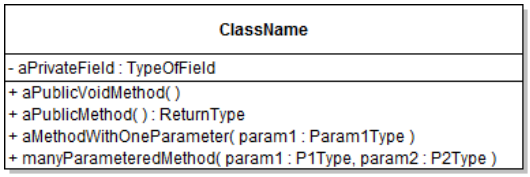
- Bir nesne, kendi sınıfından olan diğer nesnelerin ve bizzat kendisinin bütün üyelerine erişebilir,
- Ancak bir nesnenin üyelerine diğer sınıflardan olan nesnelerin erişmesi engellenebilir.
- Veri Gizleme İlkesi (information hiding):
  - İlke olarak, bir sınıfın içsel çalışması ile ilgili üyeler diğerlerinden 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.

22

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:



+ : public
- : private

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

23

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

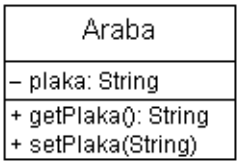
- Veri gizleme ilkesi her zaman için mükemmel olarak uygulanamaz.
  - Bir sınıftaki değişiklik sadece o sınıfı etkilemekle kalmaz, ilişkide bulunduğu başka sınıfları da etkileyebilir.
  - Veri gizleme ilkesine ne kadar sıkı uyulursa, değişikliğin diğer sınıflara yayılması olasılığı veya değişiklikten 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

24

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

25

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.

26

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 yeniden tanımlanamazlar (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.

27

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.

28

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.

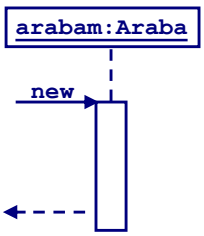
```
arabam = new Araba();
```

29

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();
   }
}
```

30

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 (imza = ad + parametre listesi) farklı olan birden fazla metot tanımlanabilir.
  - Bu tür metotlara **adaş metotlar (overloaded methods)**, bu yapılan işe ise adaş metot tanımlama (**overloading**) denir.

31

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.

32

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 metodu.
    - 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 nesneye 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 }

33

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)

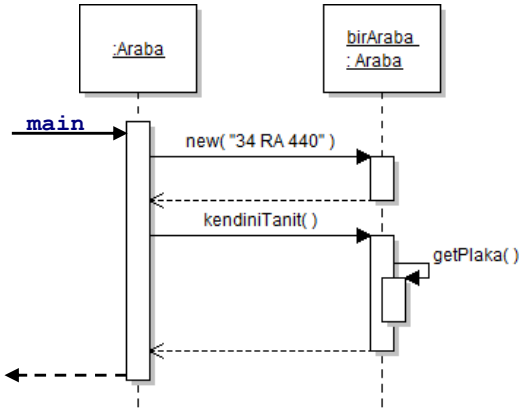
```
package ndk01;
public class Araba {
   private String plaka;

   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( );
   }
}
```

34

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 !



35

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

KENDİ SINIFLARINIZI OLUŞTURMAK VE KENDİ NESNELERİNİZİ TÜRETMEK

- Araba sınıfının başka bir versiyonu:

Car
- plate : String - chassisNR : String
+ Car(String, String) + getPlate( ) : String + setPlate(String) + getChassisNR( ) : String

```
package ndk01;
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).

36

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 doğrudur. Sağdaki hem derlemez, hem de hatalıdır (metot imzaları çakışıyor).

```
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.

37

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)
long	Tam sayı (8 sekizlik)	**(± 9,22 E 16)** (sonu L ile bitmeli)
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
char	Tek bir karakter	'ç' gibi.

- 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.

38

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.

39

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

STRING SINIFI

- String sınıfı metotları
  - int length()
  - int compareTo(String anotherString) //not: 'A' < 'a'
  - int compareToIgnoreCase(String str)
- System.out.println(String)
  - print / println
- Örnek:

```
package ndk01;
public class StringOps01 {
   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

40

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

STRING SINIFI (devam)

- String sınıfı metotları (devam)
  - boolean contains(String anotherString)
  - String toUpperCase( )
  - String toLowerCase( )
  - Dikkat: toUpper/LowerCase metotları çağırılan nesnenin kendisini değiştirmiyor.
  - Bu bilgi doğrultusunda, sonraki yansıdaki kodun çıktısı acaba ne olacaktır?

41

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

STRING SINIFI (devam)

```
package ndk01;
public class StringOps02 {
   public static void main( String args[] ) {
       String strA = "İstanbul", strB = "Yıldız";
       System.out.println(strA.contains(strB));
       strB = "tan";
       System.out.println(strA.contains(strB));
       strB.toUpperCase();
       System.out.println(strB);
       System.out.println(strA.contains(strB));
       strB = strB.toUpperCase();
       System.out.println(strB);
       System.out.println(strA.contains(strB));
   }
}
```

- Örneğin çıktısı: ???

42

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.

43

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()
    - …

```
package ndk01;
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) + ".");
   }
}
```

44

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.

```
package ndk01;
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) + ".");
      in.close();
   }
}
```

- Scanner nesnesini programın sonunda kapatmayı unutmayınız! (close)

45

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

KOMUT SATIRI ÜZERİNDEN G/Ç

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

Araba
- plaka : String
+ Araba(plakaNo : String) + getPlaka( ) : String + setPlaka(String) + kendiniTanit( ) + main(String[ ])

```
package ndk01;
import java.util.Scanner;
public class ArabaV2 {
   private String plaka;
   public ArabaV2( 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 ) {
      ArabaV2 birAraba;
      Scanner input = new Scanner( System.in );
      System.out.print("Enter a license plate: ");
      String plakaNr =  input.nextLine( );
      birAraba = new ArabaV2( plakaNr );
      birAraba.kendiniTanit( );
      input.close( );
   }
}
```

46

DENETİM AKIŞI

- Yapısal programlamadan bildiğiniz kurallar Java dili için de aynen geçerlidir, çünkü nesneye yönelim yapısal programlamanın üst kümesidir.
  - Yapısal programlamada eksik olanlar için notların sonraki bölümü eklenmiştir.
  - Ek alıştırmalar için bkz. Fahri Vatansever, “Algoritma Geliştirme ve Programlamaya Giriş”, Seçkin Yayıncılık.

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 ...
```

47

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – SWITCH DEYİMİ

- Anahtar tamsayı, tek karakter veya enum olabilir:

```
public class SwitchCase {
   enum Yon { YUKARI, ASAGI, SAG, SOL };
   void deneme( ) {
```

```
switch (anahtar1) {
   case 1:
               break;
   default:
               break;
}
```

```
switch (anahtar2) {
   case YUKARI:
               break;
   default:
               break;
}
```

```
switch (anahtar3) {
   case 'Y':
               break;
   default:
               break;
}
```

```
   }
}
```

48

Bu yansı ders notlarının düzeni için boş bırakılmıştır.

49

BMH112 PROGRAMLAMA DİLLERİ
Dr.Öğr.Üyesi Yunus Emre SELÇUK

DERS NOTLARI:
B. DENETİM AKIŞI AYRINTILARI

- Önemli açıklama:
  - B bölümünde anlatılan konular yapısal programlama temellerini hatırlatmak içindir.
  - Sınav içeriğinin odağını A ve C bölümleri oluşturacaktır.
  - Ancak nesneye yönelim yapısal programlamanın üst kümesi olduğundan, B bölümü içeriğini de unutmamak gereklidir.

50

DENETİM AKIŞI

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.

51

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Karar verme işlemleri:
  - Bir koşulu sınayarak ne yapılacağına karar vermek için if komutu kullanılır.

```
if( ifade )
      ifadenin sonucu doğruysa çalışacak tek bir komut;
if( koşul ) {
      //komutlar
}
```

- İfade/koşul çeşitleri:
  - Karşılaştırma (ilişkisel işleçler(operator)): < > <= >= == !=
  - Mantıksal işlemler: Boole cebiri. & | !
- İfadeler ve/veya mantıksal işlemleri ile birleştirilebilir.
  - Çift işleç kullanılır: && ||
- Karmaşık ifadeler işlem önceliği ve okunabilirliği arttırmak için parantezlenebilir.

52

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Karşılaştırma:

```
if( yas >= 18 )
    System.out.println("Tebrikler, siz bir yetişkinsiniz.");
if( x != y )
    System.out.println("x ile y'nin değerleri farklıdır.");
if( x == y )
    System.out.println("x ile y'nin değerleri aynıdır.");
```

53

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Mantıksal işlemler:

```
boolean dogru = true, yanlis = false, mantikli = true;
if( !dogru )
   System.out.println("Söylenen yalan.");
if( !dogru & mantikli )
   System.out.println("Söylenen yalan ama mantıklı!");
```

- Birleştirme:

```
if( dogru && mantikli )
   System.out.println("Söylenen hem doğru hem mantıklı.");
```

- Sınamalarda işlem yerine birleştirme tercih edin (& yerine &&, | yerine ||)

54

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Karar verme işlemleri:
  - Bir koşulu hem doğruluk hem de yanlışlık açısından değerlendirmek için if komutu else anahtar kelimesi ile birlikte kullanılır.

```
if( koşul ) {
      //koşul doğru ise çalışacak komutlar
}
else {
      //koşul yanlış ise çalışacak komutlar
}
```

- Örnek:

```
if( yas >= 18 ) {
   System.out.println("Tebrikler, siz bir yetişkinsiniz.");
}
else {
   System.out.println("Henüz bir yetişkin değilsiniz.");
}
```

55

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Örnek program ve akış şeması:

```
package temeller;
import java.util.*;
public class KararVerme01 {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Yetişkinlik yaşı sınırını girin: ");
    int sinir = in.nextInt();
    System.out.print("Kişinin yaşını girin: ");
    int yas = in.nextInt();
    if( yas >= sinir )
       System.out.println("Kişi kanunen bir yetişkindir.");
    else
       System.out.println("Kişi henüz reşit değildir.");
    in.close();
  }
}
```

56

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Zincirleme karar verme işlemleri:

```
if( koşul1 ) {
      //koşul1 doğru ise çalışacak komutlar
}
else if( koşul2 ){
      //koşul1 yanlış ve koşul2 doğru ise çalışacak komutlar
}
else if( koşul3 ){
      //koşul1,2 yanlış; koşul3 doğru ise çalışacak komutlar
}
...
else {
      /*tüm koşullar yanlış ise çalışacak komutlar.
        seçimliktir, bulunması şart değildir.
        Birden fazla if varsa, else en yakın if'e aittir,
        ama siz yine de kıvrık parantezleri kullanın.*/
}
```

57

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Zincirleme karar verme işlemleri örneği:

```
if( sicaklik > enSicakGun ) {
    System.out.println("Bugün olağanüstü sıcak bir gün.");
}
else if( sicaklik < enSogukGun ){
    System.out.println("Bugün olağanüstü soğuk bir gün.");
}
else {
    System.out.println("Bugün sıradan bir gün.");
}
```

- Akış şemasını da çiz.

58

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- if blokları iç içe alınabilir :

```
if( koşul1 ) {
      //koşul1 doğru ise çalışacak komutlar
      if( koşul2 ){
          //koşul2 de doğru ise çalışacak komutlar
      }
}
else {
      //koşul1 yanlış ise çalışacak komutlar
      if( koşul3 ){
          //koşul1 yanlış ve koşul3 doğru ise çalışır.
      }
}
```

- Akış şemasını da çiz.

59

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

- Örnek: Basit bir piyango çekilişi
  - Bilgisayar iki basamaklı rastgele bir sayı üretir.
  - Kullanıcı bir tahmin girer.
  - Kullanıcı sayıyı doğru tahmin ederse 10,000 TL kazanır.
  - Kullanıcı sayının basamaklarını ters sırada tahmin ederse 3,000TL kazanır.
  - Eğer kullanıcı tek bir basamağı tahmin ederse 1,000TL kazanır.

60

DENETİM AKIŞI

KARAR VERME İŞLEMLERİ – IF DEYİMİ

```
package temeller;
import java.util.*;
public class KararVerme02 {
  public static void main(String[] args) {
   Scanner in = new Scanner(System.in);
   int piyango = (int)(Math.random() * 100);
   System.out.print("Tahmininizi girin: ");
   int tahmin = in.nextInt();
   System.out.println("Çekilen sayı: " + piyango);
   if (tahmin == piyango)
     System.out.println("Doğru bildiniz: Ödülünüz 10,000TL");
   else if (tahmin % 10 == piyango / 10 && tahmin / 10 == piyango % 10)
     System.out.println("İki rakam bildiniz: Ödülünüz 3,000TL");
   else if (tahmin % 10 == piyango / 10 || tahmin % 10 == piyango % 10
        || tahmin / 10 == piyango / 10 || tahmin / 10 == piyango % 10)
     System.out.println("Tek rakam bildiniz: Ödülünüz 1,000TL");
   else
     System.out.println("Üzgünüm, kaybettiniz.");
   in.close();
  }
}
```

61

DENETİM AKIŞI

DÖNGÜLER – FOR DEYİMİ

- Şimdiye kadar verdiğimiz komutlar yalnız bir kez işlendi veya koşula bağlı olarak hiç işlenmedi.
- Aynı komutu birden fazla kez işletmek istediğimizde çevrim (döngü:loop) ifadeleri kullanırız.
- for ifadesi ile döngü: Bir komutu belli bir sayıda yinelemek için.

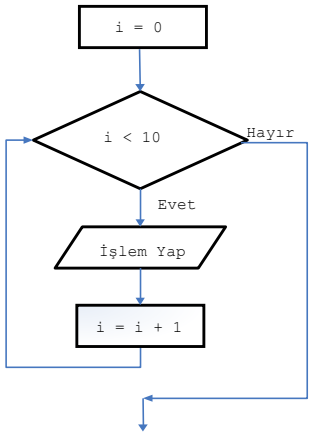
```
for( baslangicIfadesi; devamIfadesi; artimIfadesi ) {
    komutlar;
}
```

- Başlangıç ifadesi:
  - Döngüyü yineleme sayısını, bir sayaca bağlı olarak belirleriz.
  - Başlangıç ifadesinde sayaca ilk değer ataması yapılır.
  - Sayaç, başlangıç ifadesi içerisinde de tanımlanabilir.
    - int i = 0;

62

DENETİM AKIŞI

DÖNGÜLER – FOR DEYİMİ



- İleri doğru sayım:

```
for( i = 0; i < 10; i++ ) {
    System.out.print(i+" ");
}
```

  - Çıktısı: 0 1 2 3 4 5 6 7 8 9
- Geriye doğru sayım:

```
for( i = 10; i >= 0; i = i - 2 ) {
    System.out.print(i+" ");
}
```

  - Çıktısı: 10 8 6 4 2 0
- Sınır değerlere dikkat.
- Devam ifadesinde != yerine < > <= >= kullan.
  - Özellikle ondalıklı sayılarda.
- Döngünün tamamlanmasını beklemeden sonlandırmak istiyorsak **break** komutunu uygun bir biçimde kullanabiliriz.
- Verilen kurallara aykırı işler yapabilirsiniz ancak döngü aşırı karmaşıklaşır, tavsiye etmiyoruz.

63

DÖNGÜLER – FOR DEYİMİ

- Örnek program: Verilen bir N sayısının asal olup olmadığının tespiti.
  - Algoritma: N'in 2-N/2 aralığındaki sayılarla tam bölünüp bölünemediğine bakarız. Bir tane bile tam bölen bulursak N asal değildir.

```
package temeller;
import java.util.*;
public class Donguler01For {
  public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    System.out.print("Asallığı sınanacak sayıyı girin: ");
    int N = in.nextInt();
    boolean asal = true;
    for( int i=2; i<=N/2; i++ ){
      if( N % i == 0 ) {
        asal = false;
        break;
      }
    }
    if( asal )
       System.out.println("Girilen sayı asaldır.");
    else
       System.out.println("Girilen sayı asal değildir.");
    in.close();
  }
}
```

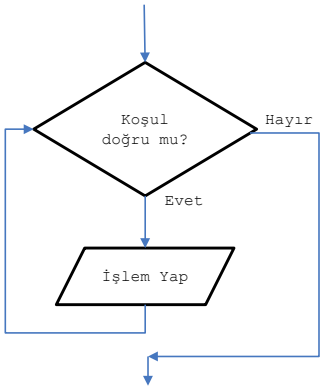
64

DENETİM AKIŞI

DÖNGÜLER – WHILE DEYİMİ

- while ifadesi ile döngü: Bir komutu belli bir koşul geçerli olduğu sürece yinelemek için.

```
while( kosul ) {
    komutlar;
}
```



- Döngüye girmeden önce döngüye girmeyi kesinleştirmek için koşulu doğrulamak gerekir (bkz. örnek kod).
- Çevrim, koşulun geçersiz olduğu anda değil, koşul geçersiz olduktan sonraki ilk kontrolde biter (bkz. örnek kod çıktısı).

65

DENETİM AKIŞI

DÖNGÜLER – WHILE DEYİMİ

- Örnek kod: Kullanıcı çıkmak isteyene kadar girdiği sayının karekökünü hesaplayan program

```
package temeller;
import java.util.*;
public class Donguler02While {
   public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        System.out.println("Girilen sayıların kareköklerini hesaplayan");
        System.out.println("bu programdan çıkmak için 0 girebilirsiniz.");
        double deger = 7.6;
        while( deger != 0 ) {
            System.out.print("Bir sayı girin: ");
            deger = in.nextDouble();
            if( deger > 0 )
                System.out.println("Karekökü: " + Math.sqrt(deger) );
        }
        in.close();
   }
}
```

66

DENETİM AKIŞI

DÖNGÜLER – WHILE DEYİMİ

- Örnek çıktı:

```
Girilen sayıların kareköklerini hesaplayan
bu programdan çıkmak için 0 girebilirsiniz.
Bir sayı girin: 9
Karekökü: 3.0
Bir sayı girin: 42
Karekökü: 6.48074069840786
Bir sayı girin: -5
Bir sayı girin: 92
Karekökü: 9.591663046625438
Bir sayı girin: 0
```

67

DENETİM AKIŞI

DÖNGÜLER – WHILE DEYİMİ

- Örnek çıktı:

```
Girilen sayıların kareköklerini hesaplayan
bu programdan çıkmak için 0 girebilirsiniz.
Bir sayı girin: 5.5
Exception in thread "main" java.util.InputMismatchException
at java.util.Scanner.throwFor(Unknown Source)
at java.util.Scanner.next(Unknown Source)
at java.util.Scanner.nextDouble(Unknown Source)
at temeller.Donguler02While.main(Donguler02While.java:11)
```

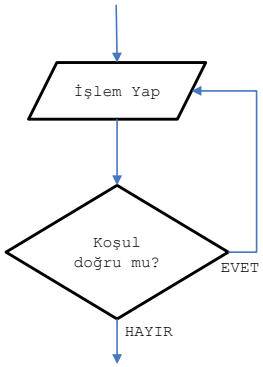
68

DENETİM AKIŞI

DÖNGÜLER – DO/WHILE DEYİMİ

```
do {
    komutlar;
} while( kosul );
```

- Farkı:
  - Kontrolün sonda yapılması.
    - Böylece ilk değer atama sorunu kalkar.
  - Noktalı virgül sonda.



69

DENETİM AKIŞI

İÇ İÇE DÖNGÜLER

- Döngü içinde döngü kullanabiliriz.
- For içinde for, for içinde while, … her kombinasyon mümkündür.
- Dış çevrim daha seyrek, iç çevrim daha sık döner.
- Toplam tekrar sayısı = dış çevrim sayısı * iç çevrim sayısı.

```
System.out.println("(i,j)");
for( int i = 1; i < 5; i++ ) {
   for( int j = 1; j < 5; j++ ) {
       System.out.println("("+i+","+j+")");
   }
}
```

```
(i,j)
(1,1)
(1,2)
(1,3)
(1,4)
(2,1)
(2,2)
(2,3)
(2,4)
(3,1)
(3,2)
(3,3)
(3,4)
(4,1)
(4,2)
(4,3)
(4,4)
```

70

DENETİM AKIŞI

DÖNGÜLERDEN ÇIKMA: BREAK DEYİMİ

- break komutu ile.
- Sadece break içeren döngüden çıkılır, iç içe döngüler varsa bunların tümünden çıkılmaz.

```
System.out.println("(i,j)");
for( int i = 1; i < 5; i++ ) {
   for( int j = 1; j < 5; j++ ) {
       if( i == 2 )
              break;
       System.out.println("("+i+","+j+")");
   }
}
```

```
(i,j)
(1,1)
(1,2)
(1,3)
(1,4)
(3,1)
(3,2)
(3,3)
(3,4)
(4,1)
(4,2)
(4,3)
(4,4)
```

71

DİZİLER

DİZİLER

- Birçok programlama probleminin çözümü için, birbiri ile ilişkili ve aynı tipten verileri ayrı ayrı değişkenlerde tutmak yerine, bunları birlikte saklamak daha uygundur.
- Birlikte saklanan veriler tek bir birleşik "veri yapısı" içerisinde yer alır.
- Bu derste dizi (array) adlı veri yapısını inceleyeceğiz.
- Dizi tanımlama:

```
TipAdi[] diziAdi;
diziAdi = new TipAdi[elemanSayisi];
```

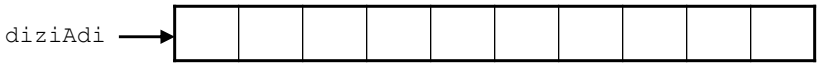
- Tek adımda tanımlama:

```
TipAdi[] diziAdi = new TipAdi[elemanSayisi];
```

- Alternatif tanımlama:

```
TipAdi diziAdi[]; //C usulü
```

- Java usulü, değişkenin bir dizi olduğunu daha iyi vurguluyor.
- Bellekteki durum: Herbiri bir eleman alabilecek, ardışıl konumlanmış hücreler.



72

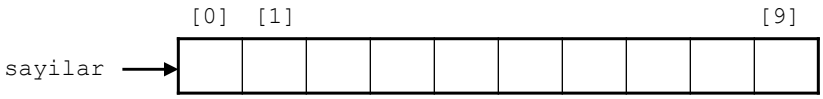
DİZİLER

DİZİLER

- Örnek tanımlama:

```
int[] sayilar = new int[10];
```

- Bellekte oluşan durum:



- DİKKAT: Dizi elemanlarının dizin numarası (indeksi) sıfırdan başlar, eleman sayısı – 1'de biter.
- Diziler genelde çevrim ifadeleri ile kullanılır.
  - Dizilerin tüm elemanlarına sıralı olarak ulaşmak için for kullanılır.
  - Yaz: Üstteki diziye 10,20,…,100 atayan kod.
- Değer atayarak tanımlama:

```
int[] sayilar = { 10, 20, 30, … ,100 };
```

73

DİZİLER

DİZİLER

- Dizideki eleman sayısını (dizinin kapasitesini) öğrenmek için:

```
int[] a = new int[100];
int j = a.length;
```

- İstenilen boyutta bir dizi oluşturmak için:

```
int kapasite = 100;
int[] a = new int[ kapasite ];
```

74

DİZİLER

DİZİLER ve FOR EACH DÖNGÜSÜ

- Dizilerin elemanlarına sıralı erişim amaçlı özel bir for çevrimi:

```
for( TurAdi degiskenAdi : diziAdi ) {
    komutlar;
}
```

- Örnek:

```
for( int sayi : sayilar ) {
    System.out.println(sayi);
}
```

- Değişken, dizi elemanları ile aynı tipte olmalıdır.
- Dizilerden başka veri yapıları ile de for-each çevrimi kullanılabilir (ileride anlatılacak).
- Avantajlar:
  - Sınır değerleri ile uğraşmak yok.
  - Daha basit gösterim.
- Dezavantajlar:
  - İndeks değeri kullanmak istiyorsak bir anlamı olmaz (Ör: diziye 10,20,… atamak).

75

DİZİLER

DİZİLER İLE İLGİLİ PROBLEMLER

- Dizideki en küçük veya (en büyük) elemanı bulmak.
  - Değerin kaç olduğunu bulmak yetmez, en küçük elemanın dizideki yerini de bulmak gerekir.
  - Sorunu çözmek için elemanın indeksini saklamak yeter.
  - Algoritmayı yaz:
    - Dizinin ilk elemanını en küçük farzet.
    - Bunu sırayla dizinin diğer elemanları ile karşılaştır.
    - Karşılaştırdığın eleman baktığından küçükse, artık o elemanı en küçük farzet.

76

DİZİLER

DİZİDEKİ EN KÜÇÜK VE EN BÜYÜK ELEMANI BULMAK

```
package temeller;
import java.util.Scanner;
public class DiziOrnekleri01 {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int diziKapasitesi;
        System.out.print( "Dizi kaç eleman içersin? " );
        diziKapasitesi = in.nextInt( );
        int[] dizi = new int[ diziKapasitesi ];
        for( int i = 0; i < diziKapasitesi; i++ ) {
            System.out.print( (i+1) + ". elemanı girin: " );
            dizi[i] = in.nextInt( );
        }
        int enKucuk = 0, enBuyuk = 0;
        for( int i = enKucuk+1; i < dizi.length; i++ ) {
            if( dizi[enKucuk] > dizi[i] )
                enKucuk = i;
            if( dizi[enBuyuk] < dizi[i] )
                enBuyuk = i;
        }
        System.out.println( "Dizinin en küçük elemanı: " + dizi[enKucuk] );
        System.out.println( "Dizinin en büyük elemanı: " + dizi[enBuyuk] );
        in.close();
    }
}
```

77

DİZİLER

DİZİLER İLE İLGİLİ PROBLEMLER

- Diziyi sıralamak: Artan veya azalan sırada.
  - Algoritma: Selection Sort
    - Dizideki en küçük elemanı önceki şekilde bul.
    - Bulduğun en küçük eleman ile dizinin başındaki elemanın yerini değiştir: İlk eleman yerine oturdu.
    - Dizinin 2. en küçük elemanını bul. Dizinin başından başlama, çünkü orada 1. en küçük eleman var. Dolayısıyla dizinin 2. elemanından başla.
    - Bulduğun 2. en küçük eleman ile dizinin başındaki elemanın yerini değiştir: 2. eleman yerine oturdu.
    - Bu şekilde tüm elemanları yerine oturt.
    - İşlem:
      - 1. elemanı 2., 3., … elemanlarla karşılaştır.
      - 2. elemanı 3., 4., … elemanlarla karşılaştır.
      - … =>Yukarıdan aşağıya 1,2, … soldan sağa sütun başı +1'den başlayan bir seri var. Her seri bir döngü: İç içe iki döngü.

78

DİZİYİ ARTAN SIRADA SIRALAMA

```
package temeller;
import java.util.Scanner;
public class DiziOrnekleri02 {
    public static void main(String[] args) {
          Scanner in = new Scanner(System.in);
          int diziKapasitesi;
          System.out.print( "Dizi kaç eleman içersin? " );
          diziKapasitesi = in.nextInt( );
          int[] dizi = new int[ diziKapasitesi ];
          for( int i = 0; i < diziKapasitesi; i++ ) {
              System.out.print( (i+1) + ". elemanı girin: " );
              dizi[i] = in.nextInt( );
          }
          int i, j;
          for( i = 0; i < dizi.length-1; i++ ) {
              int minIndex = i;
              for( j = i+1; j < dizi.length; j++ ) {
                    if( dizi[j] < dizi[minIndex] ) {
                        minIndex = j;
                    }
              }
              int temp = dizi[i];
              dizi[i] = dizi[minIndex];
              dizi[minIndex] = temp;
          }
          System.out.print( "Sıralanmış dizi: " );
          for( int eleman: dizi )
              System.out.print( eleman + " " );
          in.close();
    }
}
```

79

DİZİLER

DİZİYİ ARTAN SIRADA SIRALAMA

- İç içe 2 döngü: $O(n^2)$ karmaşıklık.
  - Özyinelemeli algoritmalarla daha düşük karmaşıklık.
  - Ör: Quicksort
  - Hazır kullanım: java.util.Arrays.sort metodu.

```
for( i = 0; i < dizi.length-1; i++ ) {
    int minIndex = i;
    for( j = i+1; j < dizi.length; j++ ) {
          if( dizi[j] < dizi[minIndex] ) {
              minIndex = j;
          }
    }
    int temp = dizi[i];
    dizi[i] = dizi[minIndex];
    dizi[minIndex] = temp;
}
```

→

```
Arrays.sort( dizi );
```

80

DİZİLER

DİZİLER İLE İLGİLİ PROBLEMLER

- Sıralı dizide en hızlı eleman arama algoritması: İkili arama (binary search)
  - Aranan değeri dizinin ortasındaki elemanla karşılaştır.
    - bas: dizinin başını, son: dizinin sonunu gösteren değişken.
    - Başlangıçta bas = 0, son = uzunluk-1, orta := (bas + son) / 2
  - Eşitlerse, bulunmuştur.
  - Dizinin ortasındaki eleman aranan elemandan küçükse, dizinin sadece sağ tarafında ara (bas = orta + 1).
  - Dizinin ortasındaki eleman aranan elemandan büyükse, dizinin sadece sol tarafında ara (son = orta – 1).
  - bas > son ise dizide arayacak yer kalmamış demektir.
  - Sonuç: Aranan eleman dizide mevcutsa elemanın indeksi
    - Aranan eleman dizide yoksa -1 sonucu verilsin
  - Gerçekleme ayrıntıları: Dizi bir kez girilsin, sonra kullanıcı 0 girene dek girdiği her değer dizide aransın.

81

DİZİLER

İKİLİ ARAMA

- Program kodu:

```
package temeller;
import java.util.*;
public class DiziOrnekleri03 {
    public static void main(String[] args) {
          Scanner in = new Scanner(System.in);
          int diziBoyutu;
          System.out.print( "Dizi kaç eleman içersin? " );
          diziBoyutu = in.nextInt( );
          int[] dizi = new int[ diziBoyutu ];
          for( int i = 0; i < diziBoyutu; i++ ) {
              System.out.print( (i+1) + ". elemanı girin: " );
              dizi[i] = in.nextInt( );
          }
          Arrays.sort(dizi);
          System.out.print( "Sıralanmış dizi: " );
          for( int eleman: dizi )
          System.out.print( eleman + " " );
          System.out.println();
```

82

- Program kodu (devam):

```
        char c; int aranan, bas, son, orta, yer;
        do {
            System.out.print("Aramak istediğiniz elemanı girin: ");
            aranan = in.nextInt();
            bas = 0; son = dizi.length-1; yer = -1;
            while( bas <= son ) {
                  orta = ( bas + son ) / 2;
                  System.out.print(" bas: " + bas);
                  System.out.print(" son: " + son);
                  System.out.print(" orta: " + orta);
                  System.out.println(" dizi[orta]: " + dizi[orta]);
                  if( dizi[orta] == aranan ) {
                      yer = orta; break;
                  }
                  else {
                      if( dizi[orta] < aranan )
                            bas = orta + 1;
                      else
                            son = orta - 1;
                  }
            }
            if( yer != -1 )
                  System.out.println("Aranan değer " + (yer+1) + ". sırada bulundu.");
            else
                  System.out.println("Aranan değer bulunamadı.");
            System.out.print("Devam etmek istiyor musunuz (e/h)? ");
            in.nextLine();
            c = in.nextLine().charAt(0);
        } while( c == 'E' || c == 'e');
        in.close();
    }
}
```


DİZİLER

İKİLİ ARAMA

- Program çıktısı (kısmi):

```
Sıralanmış dizi: 13 19 27 28 45 57 58 76 83 99
Aramak istediğiniz elemanı girin: 27
 bas: 0 son: 9 orta: 4 dizi[orta]: 45
 bas: 0 son: 3 orta: 1 dizi[orta]: 19
 bas: 2 son: 3 orta: 2 dizi[orta]: 27
Aranan değer 3. sırada bulundu.
Devam etmek istiyor musunuz (e/h)? e
Aramak istediğiniz elemanı girin: 88
 bas: 0 son: 9 orta: 4 dizi[orta]: 45
 bas: 5 son: 9 orta: 7 dizi[orta]: 76
 bas: 8 son: 9 orta: 8 dizi[orta]: 83
 bas: 9 son: 9 orta: 9 dizi[orta]: 99
Aranan değer bulunamadı.
Devam etmek istiyor musunuz (e/h)? h
```

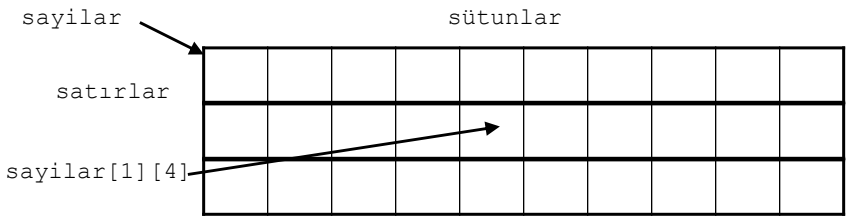
- Algoritmanın karmaşıklığı: O(ln N)
  - N = 2^k eleman için en fazla k adımda aranan elemana ulaşılır.

DİZİLER

ÇOK BOYUTLU DİZİLER

- Şimdiye dek gördüğümüz diziler doğrusal bir yapıydı = tek boyutluydu.
- Bir değişkeni 0 boyutlu bir varlık, yani nokta gibi düşünebilirsiniz. Dizileri ise yan yana dizilmiş noktalardan oluşan bir çizgi gibi, yani bir boyutlu olarak düşünebilirsiniz.
- Doğal olarak düşünceniz 2 ve 3 boyutlu dizilere genişleyecektir.
- 2 boyutlu dizi: Bir satranç tahtası gibi, satırlar ve sütunlardan oluşmuş, dizilerin dizisi.
- 2 boyutlu dizi tanımlama: [satır][sütun], dikkat: önce satır, sonra sütun.

```
int[][] sayilar = new int[3][10];
```



85

DİZİLER

ÇOK BOYUTLU DİZİLER

- Dersin nesneye yönelim odağını oluşturan konuları, çok boyutlu dizi kullanımına ihtiyaç duymadığı için, öğrenciler ders kapsamında bu konudan sorumlu değillerdir.

86

Bu yansı ders notlarının düzeni için boş bırakılmıştır.

BMH112 PROGRAMLAMA DİLLERİ
Dr.Öğr.Üyesi Yunus Emre SELÇUK

DERS NOTLARI:
C. NESNEYE YÖNELİK İLERİ KAVRAMLAR

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 (kalıtım hariç).

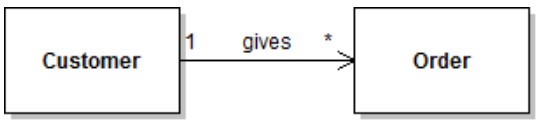
89

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.

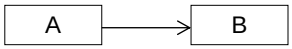
- Örnek: Müşteri ve siparişleri
  - İlişki adları ve nicelikleri de yazılabilir.

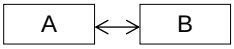


90

NESNELER ARASINDAKİ İLİŞKİLER

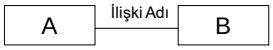
Sahiplik (Association)





Çift yönlü bağımlılık

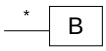
- Gösterim:



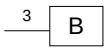
Sahiplik

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.

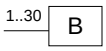


0 veya daha fazla



Tam 3 adet

1 veya daha fazla



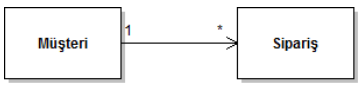
1'den 30'a kadar

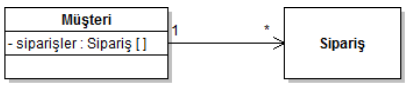
91

NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

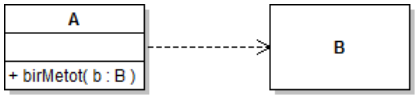
- Sahiplik ilişkisi çizgi ve oklarla gösterilmişse, sınıfların içerisinde ayrıntılı olarak gösterilmek zorunda değildir.
  - Ör: Sol alttaki şekil ile sağ alttaki şekil denktir.
  - Diğer ilişkiler için de aynı şey geçerlidir.





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.

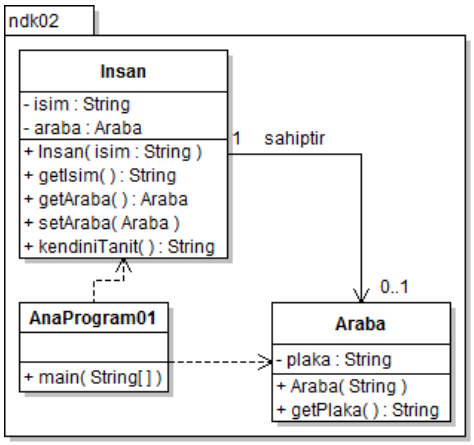
92

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



93

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:

94

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?)

95

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.

96

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`

97

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Nihayet main metodu içeren uygulamamızı yazabiliriz:

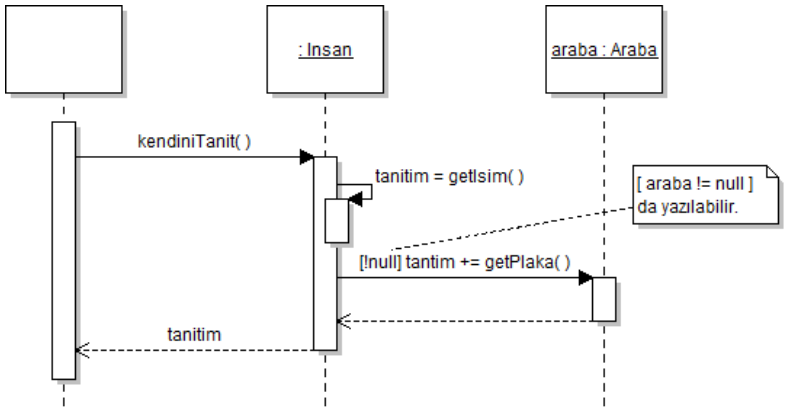
```
package ndk02;

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 aziz = new Insan("Aziz Sancar");
            System.out.println( oktay.kendiniTanit() );
            System.out.println( aziz.kendiniTanit() );
   }
}
```

98

NESNE İLİŞKİLERİNİN KODLANMASI

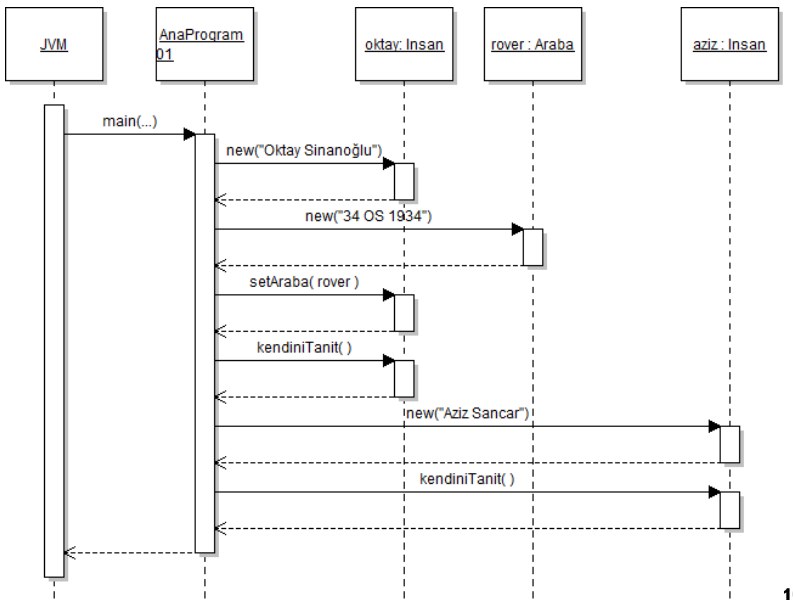
BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Insan sınıfının kendiniTanit metodunun etkileşim şeması:



99

- AnaProgram01 çalıştırılması ile ilgili etkileşim şeması :

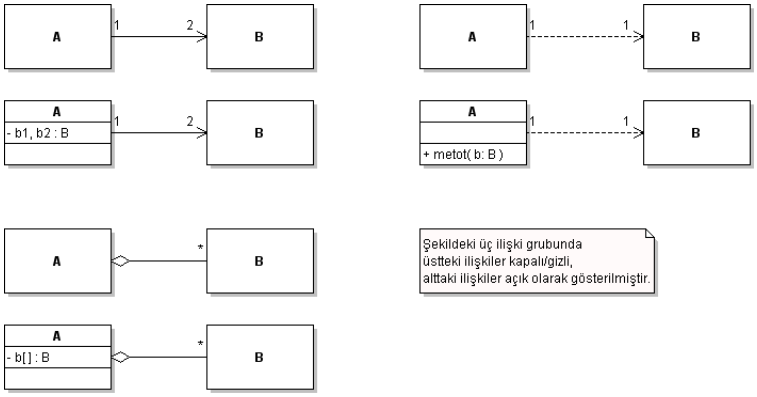


100

NESNELER ARASINDAKİ İLİŞKİLER

GİZLİ İFADELER

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

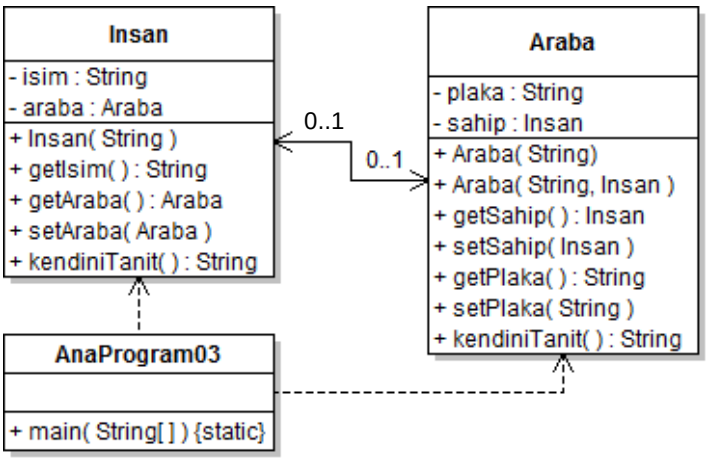


101

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.



102

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.

103

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 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 = "[ARABA] Plakam: " + getPlaka();
       if( sahip != null )
            tanitim += " Sahibimin adı: " + sahip.getIsim();
       return tanitim;
   }
}
```

Attention!

104

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.

105

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              Insan aziz = new Insan("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
```

106

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?

107

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!

108

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Main metodu içeren sınıf:

```
package ndk04c;
public class AnaProgram04a {
        public static void main(String[] args) {
                Insan oktay = new Insan("Oktay Sinanoğlu");
                Araba rover = new Araba("06 OS 1934");
                //oktay.setAraba(rover);
                rover.setSahip(oktay);
                System.out.println( oktay.kendiniTanit() );
                System.out.println( rover.kendiniTanit() );

                Insan aziz = new Insan("Aziz Sancar");
                Araba honda = new Araba("34 AZ 34");
                aziz.setAraba(honda);
                //honda.setSahip(aziz);
                System.out.println( aziz.kendiniTanit() );
                System.out.println( honda.kendiniTanit() );
        }
}
```

109

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Bir eksiğimiz daha kaldı, almamız gereken 2. önlem var:
  - Car(String, Insan) kurucumuz da var.

```
package ndk04a;
public class Araba {
   private String plaka;
   private Insan sahip;
   public Araba( String plakaNo ) { plaka = plakaNo; }
   public Araba(String plaka, Insan sahip) {
           this.plaka = plaka;
           setSahip(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 = "[ARABA] Plakam: " + getPlaka();
       if( sahip != null )
           tanitim += " Sahibimin adı: " + sahip.getIsim();
       return tanitim;
   }
}
```

2. önlem!

110

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- 2. önlemi almazsak, aşağıdaki gibi bir main ile ilişkinin iki yönü de sağlam kurulmaz:

```
package ndk04a;
public class AnaProgram04 {
        public static void main(String[] args) {
                Insan oktay = new Insan("Oktay Sinanoğlu");
                Araba rover = new Araba("06 OS 1934", oktay);
                System.out.println( oktay.kendiniTanit() );
                System.out.println( rover.kendiniTanit() );
        }
}
```

111

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

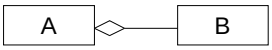
- Sonuç:
  - Çift yönlü bağıntı tek yönlü bağıntıya göre daha esnektir ancak kodlaması daha zordur.
  - Bu nedenle çift yönlü bağıntıya gerçekten ihtiyacınız yoksa kodlamayın.
  - Peki ya sonradan ihtiyaç duyarsak?
    - Şimdiden kodlamaya çalışıp zaman kaybetmeyin. Zaten yetiştirmeniz gereken bir dolu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

112

NESNELER ARASINDAKİ İLİŞKİLER

Toplama (Aggregation)

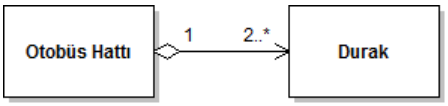
- **Parça-bütün ilişkisini** simgeler.
- Gösterim:



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.



113

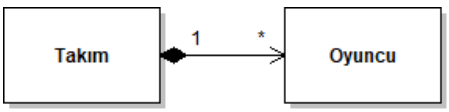
NESNELER ARASINDAKİ İLİŞKİLER

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:



114

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

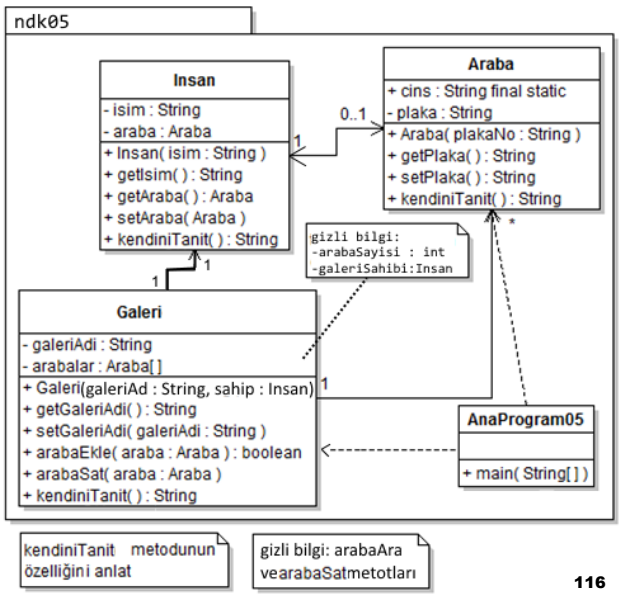
- Örnek: Satılacak birden fazla araba içeren Galeri adlı bir sınıf oluşturalım.
  - Daha önce yazdığımız Araba sınıfını aynen kullanabiliriz.
    - Ek olarak sadece plaka alan bir kurucu yazmak isteyebiliriz. O zaman aynen kullanmamış oluruz.
    - Aynen kullanacaksak:
      - kurucuya sahip üyesini 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.

115

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- UML sınıf şeması:



- Not: Galeri.arabaSat (plaka:String) daha uygun olacak, öyle kodlayalım.

116

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodu:

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

   public Galeri( String galeriAd, Insan 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 ayırıldı.

117

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 metot yardımı ile bir denetleme yapmayı akıl edip kodladık.

118

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Galeri sınıfının kaynak kodunun devamı:

```
private int arabaBul( Araba araba ) {
    for( int i=0; i<arabaSayisi; i++ )
        if( arabalar[i] == araba ) )
            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?

119

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- Bir main metodu içeren sınıfla yazdıklarımızı deneyelim:

```
package ndk05;
public class AnaProgram05 {
   public static void main(String[] args) {
      Insan insan = new Insan("Yunus Zengin");
      Galeri galeri = new Galeri("Yunus Oto", insan);
      Araba honda = new Araba("34 GH 001");
      boolean sonuc = galeri.arabaEkle(honda);
      if( sonuc )
         System.out.println("Test1 OK");
      else
         System.out.println("Test1 FAIL");
      sonuc = galeri.arabaEkle(honda);
      if( !sonuc )
         System.out.println("Test2 OK");
      else
         System.out.println("Test2 FAIL");
```

120

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- main metodu devamı:

```
galeri.arabaEkle( new Araba("34 GA 002") );
galeri.arabaEkle( new Araba("34 GA 003") );
galeri.arabaEkle( new Araba("34 GA 004") );
if( galeri.arabaAra("34 GA 003") != null )
   System.out.println("Test3 OK");
else
   System.out.println("Test3 FAIL");
if( galeri.arabaSat("34 GA 002") )
   System.out.println("Test4 OK");
else
   System.out.println("Test4 FAIL");
if( galeri.arabaAra("34 GA 002") == null )
   System.out.println("Test5 OK");
else
   System.out.println("Test5 FAIL");
if( galeri.arabaAra("34 GA 003") != null )
   System.out.println("Test6 OK");
else
   System.out.println("Test6 FAIL");
if( galeri.arabaAra("34 GA 004") != null )
   System.out.println("Test7 OK");
else
   System.out.println("Test7 FAIL");
```

121

NESNE İLİŞKİLERİNİN KODLANMASI

TOPLAMA İLİŞKİSİ

- main metodu devamı:

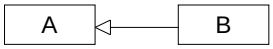
```
if( !galeri.arabaSat("35 GA 002") )
   System.out.println("Test8 OK");
else
   System.out.println("Test8 FAIL");

if( galeri.getArabaSayisi() == 3 )
   System.out.println("Test9 OK");
else
   System.out.println("Test9 FAIL");
  }
}
```

122

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)

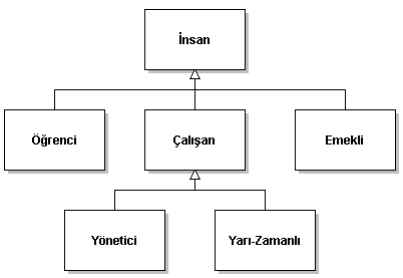
- Kalıtımın işleyişi:
  - Kalıtım yolu ile üst sınıftan alt sınıfa hem üye alanlar hem de üye metotlar aktarılır
    - private üyeler dahil, ancak alt sınıf onlara doğrudan ulaşamaz.
  - Protected üyeler ve kalıtım:
    - Alt sınıflar tarafından erişilir, diğer sınıflar tarafından erişilemez.

123

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.



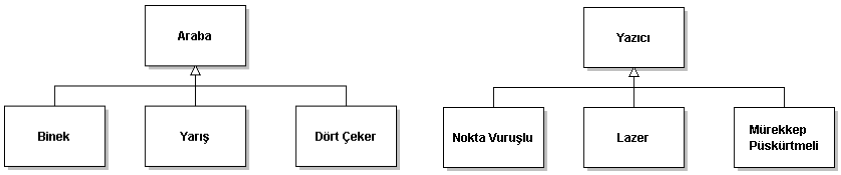
- Kalıtım ağacını çok derin tutmak doğru değildir (Kırılgan üst sınıf sorunu: Bina temelinin çürümesi gibi).

124

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.

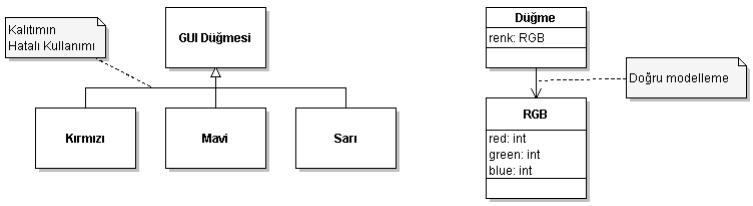


125

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtımın yanlış kullanımı:

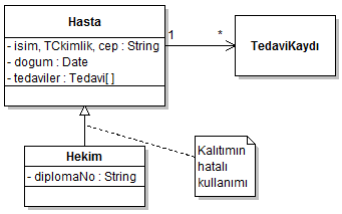


126

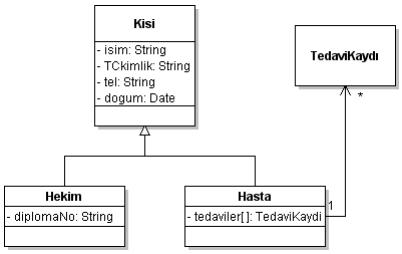
NESNELER ARASINDAKİ İLİŞKİLER

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:



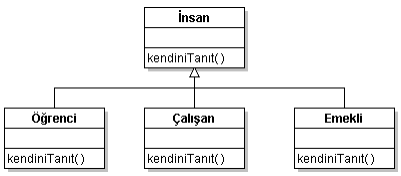
- Hekimlerin tedavi kaydının tutulması gerekmemektedir. Yanlış kullanımda her hekim aynı zamanda bir hasta olduğu için, tedavi kaydı bilgisini de alır

127

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.

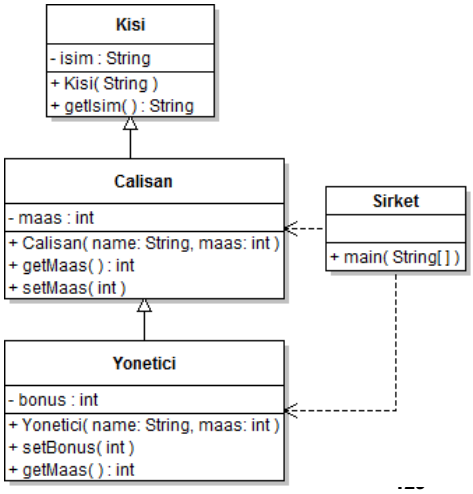
- İnsan türünden bir dizi düşünelim, elemanları İnsan ve alt sınıflarından karışık nesneler olsun. Dizinin tüm elemanlarına kendini tanıt dediğimizde ne olacak?
  - Çalışma anında doğru sınıfın metodu seçilir.
  - Bu çalışma biçimine de çok biçimlilik (polymorphism) denir.
- Peki, üst sınıfın altta yeniden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

128

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

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

130

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar (devam):

```
package ndk06;

public class Yonetici extends Calisan {
        private int bonus;

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

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

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

}
```

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

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

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

131

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( ) );
   }
}
```

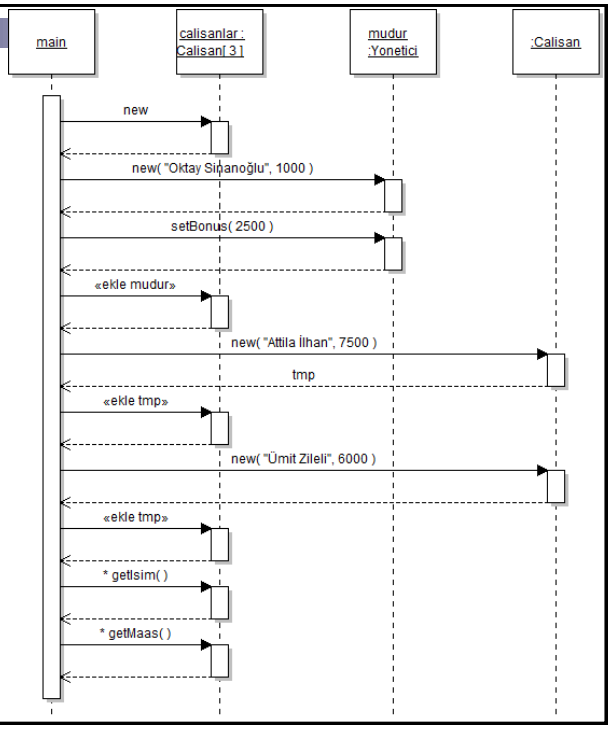
- For döngüsü dikkatinizi çekti mi?

132

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.

134

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.

135

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.

136

NYP İLE İLGİLİ ÖZEL KONULAR

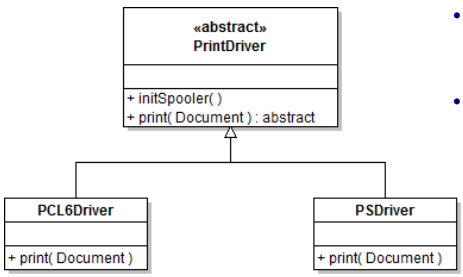
SOYUT SINIFLAR

- 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.

137

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.

138

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 );
}
```

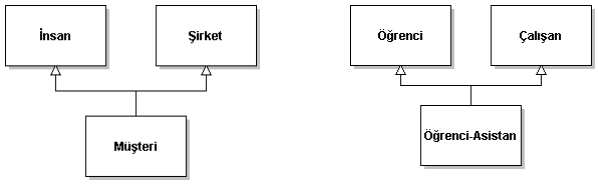
```
package ndk07;
public class PCL6Driver extends PrintDriver {
    public void print(Document doc) {
       //necessary code is inserted here
    }
}
```

139

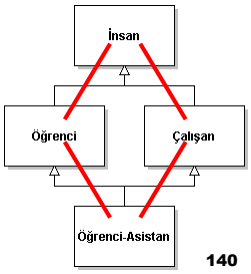
NYP İLE İLGİLİ ÖZEL KONULAR

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#

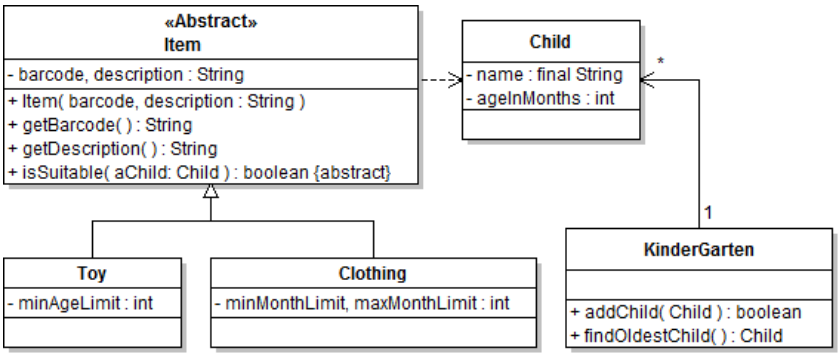


140

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?



141

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.

142

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;
    }
}
```

143

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;
    }
}
```

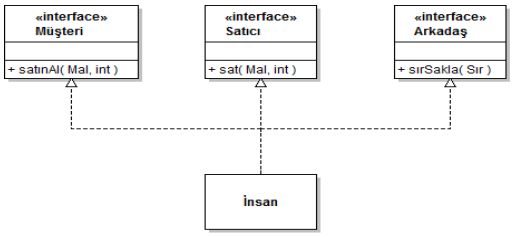
- Kindergarten (AnaOkulu) sınıfının kaynak kodunu UML sınıfında verildiği kadarıyla kodlayıp yapılan tasarımı yeni özelliklerle geliştirme işini alıştırma olarak yapabilirsiniz.

144

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
   }
}
```

145

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.

146

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

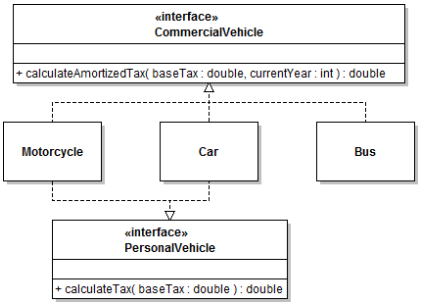
- Arayüzler ile ilgili kurallar:
  - Bir sınıf, gerçeklediği arayüzdeki tanımlı tüm metotların gövdelerini tanımlamak zorundadır.
  - Arayüzlerde normal üye alanlar tanımlanamaz, sadece "public final static" üye alanlar tanımlanabilir.
  - Arayüzlerde sadece public metotlar tanımlanabilir.
  - Arayüzülerin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

147

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.

148

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Arayüzlerin kodlanması:

```
package ndk09;
public interface CommercialVehicle {
    public double calculateAmortizedTax( double baseTax, int currentYear );
}
```

```
package ndk09;
public interface PersonalVehicle {
    public double calculateTax( double baseTax );
}
```

149

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Araba sınıfının kodlanması

```
package ndk09;
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; }
}
```

150

NYP İLE İLGİLİ ÖZEL KONULAR

BİR ARAYÜZ TASARLAMAK VE KODLAMAK

- Otobüs sınıfının kodlanması

```
package ndk09;
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;
           //Tax can be reduced %20 for each year as amortization for heavy vehicles
           /* This rule invalidates CommercialVehicle.yearLimit
            * therefore a suitable but unavoidable more complex design can be done */
           ratioA = (currentYear - modelYear) * 0.2;
           if( ratioA <= 0.0 )
               ratioA = 0.2;
           return baseTax * ratioT * ratioA;
    }
    public int getModelYear() { return modelYear; }
    public double getEngineVolume() { return tonnage; }
}
```

151

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.

152