

PROGRAMLAMA DİLLERİ II

Mart 2013

Yrd.Doç.Dr. Yunus Emre SELÇUK

GENEL BİLGİLER


BAŞARIM DEĞERLENDİRME

- 1. Ara Sınav: %30, 2. Ara Sınav: %30, Final Sınavı: %40,
- Sınav tarihleri daha sonra belirlenecetir.

KAYNAKLAR:

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

1

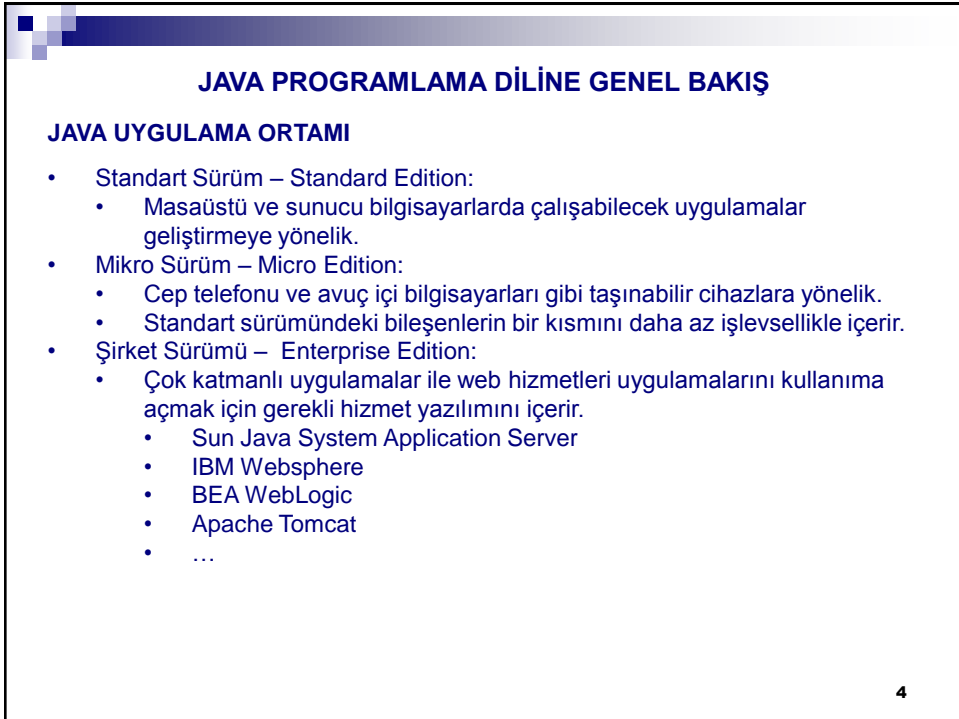
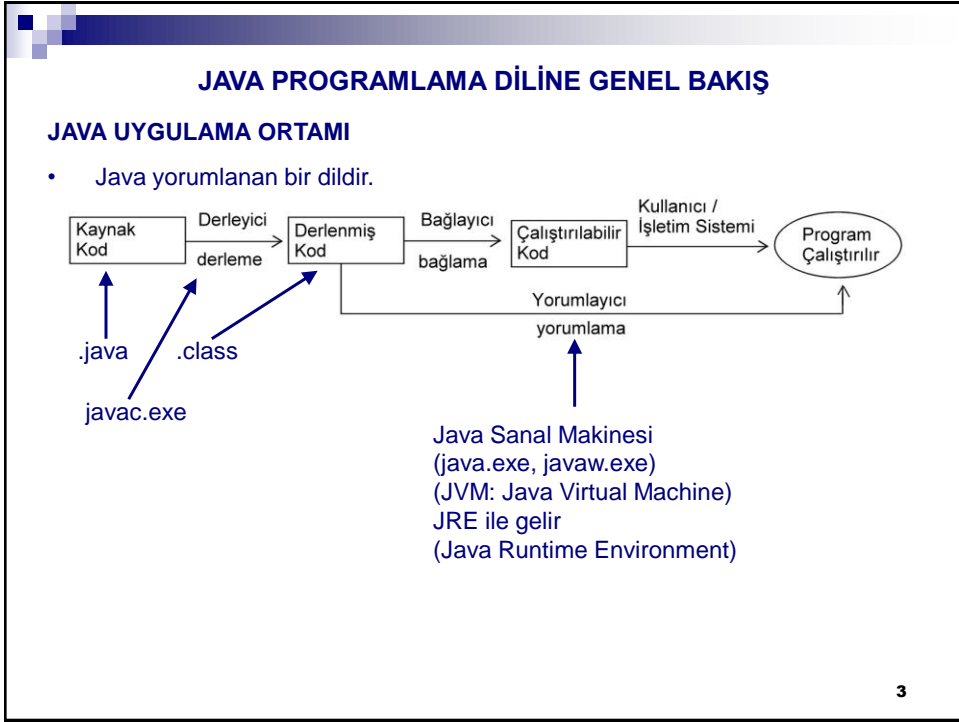


GENEL BİLGİLER

DERS İÇERİĞİ

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

2



JAVA PROGRAMLAMA DİLİNE GENEL BAKIŞ

ÜCRETSİZ JAVA GELİŞTİRME ORTAMLARI

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

ÜCRETSİZ UML MODELLEME ORTAMLARI

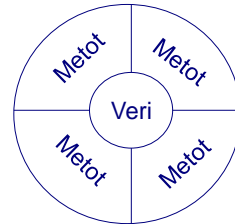
- Violet UML: Hafif sıklet, bizim için yeterli
- Argo UML

7

SINIFLAR, NESNELER VE ÜYELER

NESNE

- Nesne: Nitelikler ve tanımlı eylemler içeren, temel programlama birimi.
 - Nesne = varlık.
 - 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.
 - Sarma (Encapsulation): Veri ve eylemlerin birlikteliği.
 - Verilere, eylemler üzerinden erişilir.



8

SINIFLAR, NESNELER VE ÜYELER

SINIF

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

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

9

SINIFLAR, NESNELER VE ÜYELER

NESNELER VE SINIFLAR

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

10

SINIFLAR, NESNELER VE ÜYELER

NESNELER VE SINIFLAR

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

11

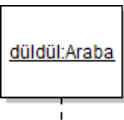
SINIFLAR, NESNELER VE ÜYELER

TERMİNOLOJİ VE GÖSTERİM

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



Sınıf: Sınıf şemasında



Nesne: Etkileşim şemasında

12

SINIFLAR, NESNELER VE ÜYELER

TERMİNOLOJİ VE GÖSTERİM

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

13

SINIFLAR, NESNELER VE ÜYELER

HER NESNE FARKLI BİR BİREYDİR!

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

14

SINIFLAR, NESNELER VE ÜYELER

HER NESNE FARKLIDIR! (devam)

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

15

SINIFLAR, NESNELER VE ÜYELER

NESNELERE MESAJ GÖNDERME

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

ÜYELERE ERİŞİM

- Üye alana erişim:
 - Üyenin değerini değiştirmek
 - Üyenin değerini okumak (Değiştirmeden herhangi bir işlemde kullanmak)
- Üye metoda erişim:
 - Bir eylemler sürecini varsa kendine özgü çalışma parametreleri ile yürütmek.
 - Fonksiyon çağırma gibi, ama unutmayın: Aksi belirtilmedikçe metod, üyesi olduğu nesnenin üyeleri ile çalışır.

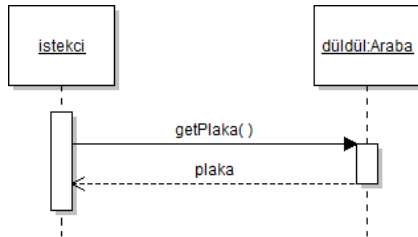
16

SINIFLAR, NESNELER VE ÜYELER

TERMİNOLOJİ VE GÖSTERİM

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

UML Gösterimi:



Kod Gösterimi:

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

Anlamı:

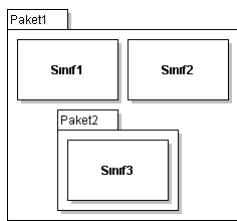
- istekçi adlı bir nesne vardır.
- istekçi nesnenin sınıfı belli değil.
- döldül adlı bir nesne vardır.
- döldül nesnesinin sınıfı Araba'dır.
- Araba sınıfının plakaniVer adlı bir metodu vardır.
- istekçi nesne döldül nesnesine plakaniVer mesajı gönderir.
- döldül nesnesi bu mesaja yanıt olarak kendi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

17

SINIFLAR, NESNELER VE ÜYELER

PAKETLER

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



- Bir paketteki sınıfları koda ekleme:

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

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

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

- Paket hiyerarşisi, aynı zamanda dosya hiyerarşisidir.

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

18

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

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

19

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

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

- UML Gösterimi:

ClassName
- aPrivateField : TypeOfField
+ aPublicVoidMethod()
+ aPublicMethod() : ReturnType
+ aMethodWithOneParameter(param1 : Param1Type)
+ manyParameteredMethod(param1 : P1Type, param2 : P2Type)

- Ayrıca (derste sorumlu değilsiniz):
 - protected: #
 - Kalıtım ile ilgili
 - package: ~
 - Java'da varsayılan kural

20

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

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

21

SINIFLAR, NESNELER VE ÜYELER

GÖRÜNEBİLİRLİK KURALLARI VE VERİ GİZLEME

- Örnek:

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

22

SINIFLAR, NESNELER VE ÜYELER

ÜYELERİN ÖZEL DURUMLARI

- Static 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 kullanılırlar.
 - Örnek: Her binek otomobilin 4 tekerleği vardır.
- Static 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.
 - Static metot içerisinde yalnız static üyeler kullanılabilir.
 - SOR: Static üye alana erişim metotları da static tanımlanır.
 - SınıfAdı.üyeAdı() şeklinde kullanılırlar.

23

SINIFLAR, NESNELER VE ÜYELER

ÜYELERİN ÖZEL DURUMLARI

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

DİKKAT EDİLECEK NOKTALAR

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

24

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR VE SONLANDIRICILAR

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

25

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Kurucu Metot kuralları:
 - Public görünürlüğe 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 komutu ile birlikte kullanılır.

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

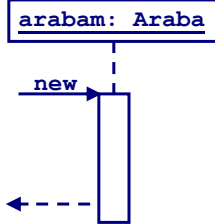
26

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

- Bir nesneyi kullanmak için onu tanımlamak yetmez, kurucusunu da çalıştırmak gerekir.

- UML Gösterimi 1:



- Kod gösterimi 1: Üye alan olarak kullanım

```
public class AClass {
    private Araba arabam;

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

- Kod gösterimi 2: Geçici değişken olarak kullanım

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

27

SINIFLAR, NESNELER VE ÜYELER

KURUCULAR

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

28

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

DENETİM AKIŞI

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

29

BİR NESNEYE DAYALI PROGRAMIN OLUŞTURULMASI

DENETİM AKIŞININ BAŞLANGICI

- Denetim akışının bir başlangıcının olması gereklidir.
 - Akışın hangi sınıftan başlatılacağını programcı belirler.
 - Java'da denetim akışının başlangıcı: Main komutu.
 - `public static void main(String[] args)`
 - `static`: Henüz bir nesne türetilmedi!
 - `args` dizisi: Programa komut satırından ilk parametreleri aktarmak için
 - Main metodunun görevi, gerekli ilk bir/birkaç nesneyi oluşturup programın çalışmasını başlatmaktır.
 - Hatırlayın, bir 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 }`

30

<

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

İLKEL VERİ TIPLERİ

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

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

33

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

İLKEL VERİ TIPLERİ

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

34

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

STRING SINIFI

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

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

- Örneğin çıktısı: -19

35

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.

36

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 ooc00;
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) + ".");
    }
}
```

37

TEMEL VERİ TEMSİLİ VE İŞLEMLERİ

KOMUT SATIRI ÜZERİNDEN G/Ç

- Scanner sorunu (5.0-6.0):
 - nextInt, nextFloat, vb. ilkel okuma komutundan sonra bir karakter katırı okumak için nextLine kullanırsan sorun çıkıyor.
 - Çözmek için ilkel okumadan sonra bir boş nextLine ver.

```
package ooc00;
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) + ".");
    }
}
```

38

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

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

39

DENETİM AKIŞI

- BBG2'den beri bildiğiniz yapılar burada da benzer sözdizimi ile geçerli, bu derste artık üzerinde durmayacağız.
- Aşağıda kısa bir özet yer almaktadır.

KARAR VERME İŞLEMLERİ – IF DEYİMİ

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

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

DÖNGÜLER

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

40

NESNELER ARASINDAKİ İLİŞKİLER

NESNELER ARASINDAKİ İLİŞKİLER

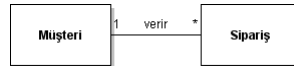
- Bir nesneye yönelik programın, nesneler arasındaki mesaj akışları şeklinde yürüdüğünü gördük.
- Bir nesnenin diğerine bir mesaj gönderebilmesi (yani kullanabilmesi) için, bu iki nesne arasında bir ilişki olmalıdır.
- İlişki çeşitleri:
 - Sahiplik (Association)
 - Kullanma (Dependency)
 - Toplama (Aggregation)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

41

NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)

- Bağlantı ilişkisi için anahtar kelime **sahipliktir**.
- Kullanan nesne, kullanılan nesne türünden bir üyeye sahiptir.
- Sadece ilişki kelimesi geçiyorsa, ilişkinin iki nesne arasındaki sahiplik ilişkisi olduğu anlaşılır.
- SOR: 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.



42

NESNELER ARASINDAKİ İLİŞKİLER

Sahiplik (Association)

Gösterim:

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

Çift yönlü bağımlılık

İlişki Adı

Sahiplik

- 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.

* B 0 veya daha fazla 3 B Tam 3 adet 1..* B 1 veya daha fazla 1..30 B 1'den 30'a kadar

43

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.

Müşteri 1 * Sipariş

Müşteri
- siparişler : Sipariş []
1 * Sipariş

KULLANMA İLİŞKİSİ (DEPENDENCY)

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

A B

+ birMetot(b : B)

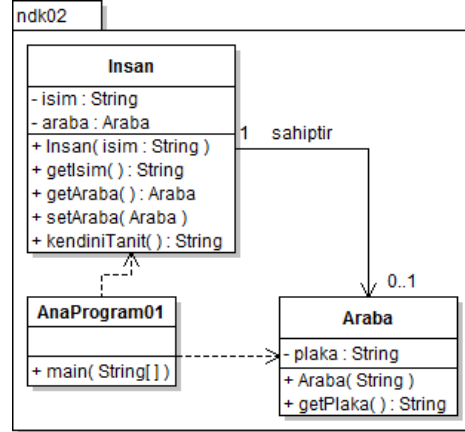
A, B'yi kullanır: A örnekleri birMetot içinden b nesnesine mesaj gönderebilir.

44

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

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



45

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:

46

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- İnsan sınıfının kaynak kodu:

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

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

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

Dikkat! (Bu da nereden çıktı?)

47

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- UML sınıf şemamızda İnsan - Araba ilişkisinin Araba ucunun 0..1 yazdığı dikkatinizi çekti mi?
 - Bu ne anlama geliyor?
 - Her insanın bir arabası olmayabilir anlamına geliyor.
- Ayrıca:
 - Bir sınıfa bir metod eklenince, hangi metodun hangi sırada çalıştırılacağı, hatta çalıştırılıp çalıştırılmayacağına garanti 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 sıyalalım, ona göre arabasının plakasına ulaşmaya çalışalım.
 - İnsanın arabası yokken, o üye alanın değeri null olmaktadır.
 - Yani o üye ilklendirilmemiştir.

48

NESNE İLİŞKİLERİNİN KODLANMASI

NESNENİN ETKİNLİĞİNİN SINANMASI

- Bir nesne iklenirildiğinde artık o nesne için etkindir denilebilir.
- nesne1 işaretçisinin gösterdiği nesnenin iklendirilip iklendirilmediğinin sinanması:

	İfade	Değer
İklenmişse (etkinse)	nesne1 == null	false
	nesne1 != null	true
İklenmemişse (etkin değilse)	nesne1 == null	true
	nesne1 != null	false

49

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- Nihayet main metodu içeren uygulamamızı yazabiliriz:

```
package ndk02;

public class AnaProgram031{

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

50

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – TEK YÖNLÜ

- İnsan sınıfının kendiniTanit metodunun etkileşim şeması:

```
sequenceDiagram
    participant Caller
    participant I as : İnsan
    participant A as araba : Araba
    Caller->>I: kendiniTanit()
    activate I
    I->>I: tanitim = getIsim()
    I->>A: [!null] tanitim += getPlaka()
    activate A
    A-->>I: 
    deactivate A
    I-->>Caller: tanitim
    deactivate I
```

51

- AnaProgram01 çalıştırılması ile ilgili etkileşim şeması :

```
sequenceDiagram
    participant JVM
    participant AP as AnaProgram01
    participant O as oktay : İnsan
    participant R as rover : Araba
    participant S as serkan : İnsan
    JVM->>AP: main(...)
    activate AP
    AP->>O: new("Oktay Sinanoğlu")
    activate O
    O-->>AP: 
    deactivate O
    AP->>R: new("06 RVR 06")
    activate R
    R-->>AP: 
    deactivate R
    AP->>O: setAraba( rover )
    activate O
    O-->>AP: 
    deactivate O
    AP->>S: new("Serkan Anılır")
    activate S
    S-->>AP: 
    deactivate S
    AP->>O: kendiniTanit()
    activate O
    O-->>AP: 
    deactivate O
    AP->>S: kendiniTanit()
    activate S
    S-->>AP: 
    deactivate S
    AP-->>JVM: 
    deactivate AP
```

52

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.

55

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Araba sınıfının yeni kaynak kodu:

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

56

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- Yazdıklarımızı denemek için uygulamayı yazalım.

```
01 package ndk03;
02 public class AnaProgram03 {
03     public static void main(String[] args) {
04         İnsan oktay = new İnsan("Oktay Sinanoğlu");
05         Araba rover = new Araba("06 RVR 06");
06         oktay.setAraba(rover);
07         rover.setSahip(oktay);
08         System.out.println( oktay.kendiniTanıt() );
09         System.out.println( rover.kendiniTanıt() );
10
11         Person serkan = new Person("Serkan Anılır");
12         Car honda = new Car("06 JAXA 73");
13         serkan.setCar(honda);
14         honda.setOwner(serkan);
15         System.out.println( serkan.introduceSelf() );
16         System.out.println( honda.introduceSelf() );
17     }
18 }
19
20
```

57

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

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

58

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

- İnsan ve Araba sınıflarının değişen kısımları:

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

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

59

NESNE İLİŞKİLERİNİN KODLANMASI

BAĞINTI İLİŞKİSİ (ASSOCIATION) – ÇİFT YÖNLÜ

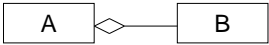
- Sonuç:
 - Çift yönlü bağıntı tek yönlü bağıntıya göre daha esnektir ancak kodlaması daha zordur.
 - Bu nedenle çift yönlü bağıntıya gerçekten ihtiyacınız yoksa kodlamayın.
 - Peki ya sonradan ihtiyaç duyarsak?
 - Şimdiden kodlamaya çalışıp zaman kaybetmeyin. Zaten yetiştirmeniz gereken bir dolu başka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

60

NESNELER ARASINDAKİ İLİŞKİLER

Toplama (Aggregation)

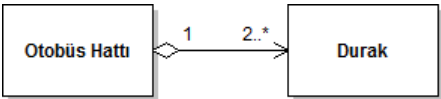
- **Parça-bütün ilişkisini** simgeler.
- Gösterim:



Toplama (aggregation)

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

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



61

NESNELER ARASINDAKİ İLİŞKİLER

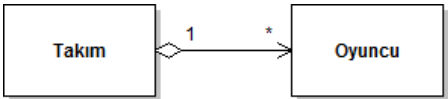
Meydana Gelme (Composition)

- 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:



62

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!

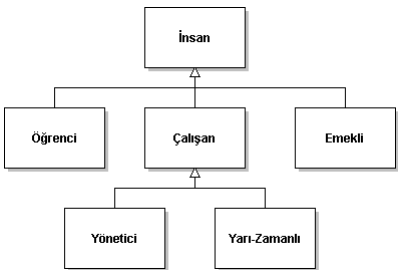
- A:
 - Ebeveyn sınıf (parent)
 - Üst sınıf (super)
 - Temel sınıf (base)
- B:
 - Çocuk sınıf (child)
 - Alt sınıf (sub)
 - Türetilmiş sınıf (derived)
- Kalıtımın işleyişi:
 - Kalıtım yolu ile üst sınıftan alt sınıfa hem üye alanlar hem de üye metotlar aktarılır
 - private üyeler dahil, ancak alt sınıf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

63

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

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



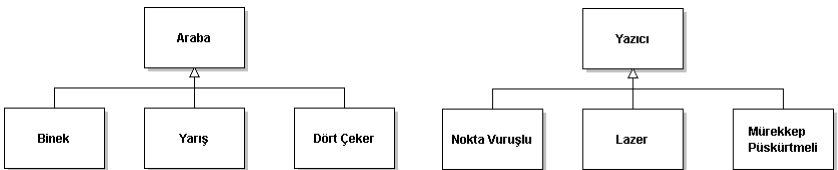
- Kalıtım ağacını çok derin tutmak doğru değildir (Kırılgan üst sınıf sorunu: Bina temelinin çürümesi gibi).

64

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 beklediği herhangi bir bağlamda kullanılabilir.
 - Bu nedenle IS-A ilişkisi olarak da adlandırılır.

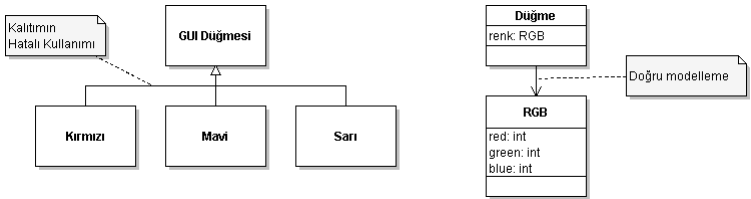


65

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

- Kalıtımın yanlış kullanımı:

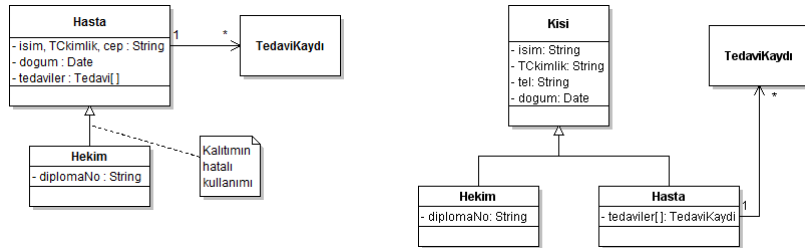


66

NESNELER ARASINDAKİ İLİŞKİLER

KALITIM

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



- Hekimlerin tedavi kaydının tutulması gerekmemektedir. Yanlış kullanımda her hekim aynı zamanda bir hasta olduğu için, tedavi kaydı bilgisini de alır. **67**

NESNE İLİŞKİLERİNİN KODLANMASI

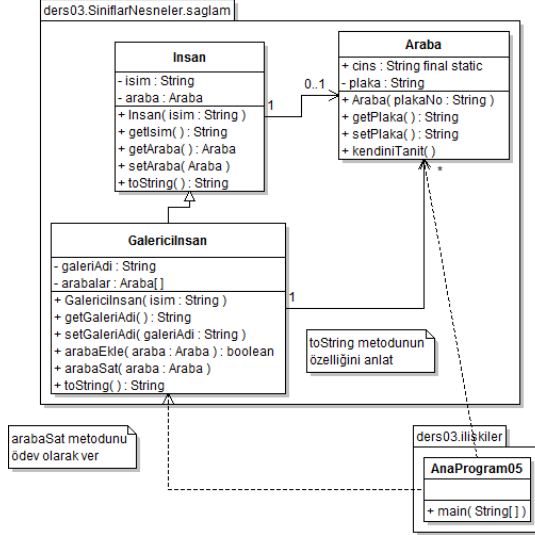
KALITIM VE TOPLAMA İLİŞKİSİ

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

NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

- UML sınıf şeması:
 - Composition yap
 - 1-*



NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

- Galericiİnsan sınıfının kaynak kodu:

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

    public Galericiİnsan( String isim ) {
        super( isim );
        arabaSayisi = 0;
        arabalar = new Araba[maxAraba];
    }

    public String getGaleriAdi() { return galeriAdi; }
    public void setGaleriAdi(String galeriAdi) { this.galeriAdi = galeriAdi; }
    public String toString() {
        String tanitim = super.toString(); //çokbüyümliliği hatırladınız mı?
        tanitim = "Merhaba, adım: " + getIsim();
        tanitim += "\nÜstelik " + galeriAdi + " adlı bir araba galerim var.";
        tanitim += "\nGalerimde " + arabaSayisi + " adet araba var.";
        return tanitim;
    }
}
//devamı var...
```

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

NESNE İLİŞKİLERİNİN KODLANMASI

KALITIM VE TOPLAMA İLİŞKİSİ

- GalericiInsan sınıfının kaynak kodunun devamı:

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

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

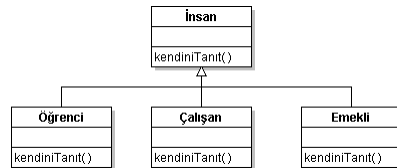
- Alıştırma: Peki araba satışı nasıl olacak? arabaSat metodunu kodlayınız.
- Alıştırma: İşin içine para meseleleri girseydi ne olacaktı?

71

KALITIM İLE İLGİLİ ÖZEL KONULAR

ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

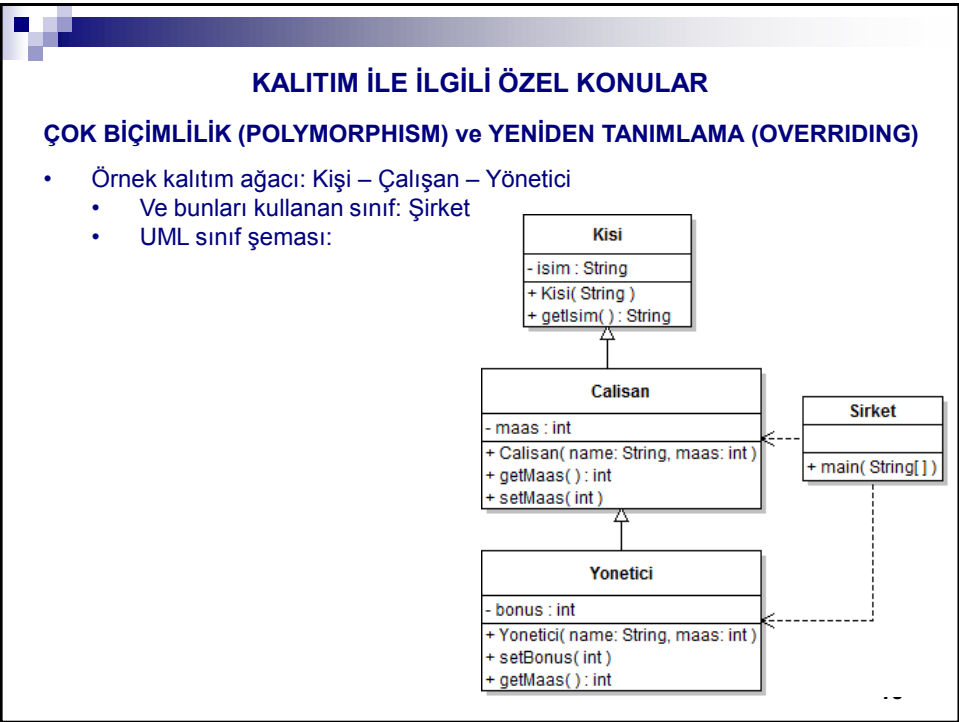
- İstersek kalıtımla geçen metotların gövdesini değiştirebileceğimizi öğrendik.
 - Bu işleme yeniden tanımlama (overriding) adı verildiğini gördük.
- Üst sınıftan bir nesnenin beklendiği her yerde alt sınıftan bir nesneyi de kullanabileceğimizi gördük.
- Bu iki özellik bir araya geldiğinde, ilgi çekici 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 tanıttığımızda ne olacak?
 - Çalışma anında doğru sınıfın metodu seçilir.
 - Bu çalışma biçimine de çok biçimlilik (polymorphism) denir.
- Peki, üst sınıftan altta yeniden tanımladığımız bir metoduna eski yani üst sınıftaki hali ile erişmek istediğimizde ne yapacağız?
 - Bu durumda da super işaretçisi ile üst sınıfa erişebiliriz!

72



KALITIM İLE İLGİLİ ÖZEL KONULAR

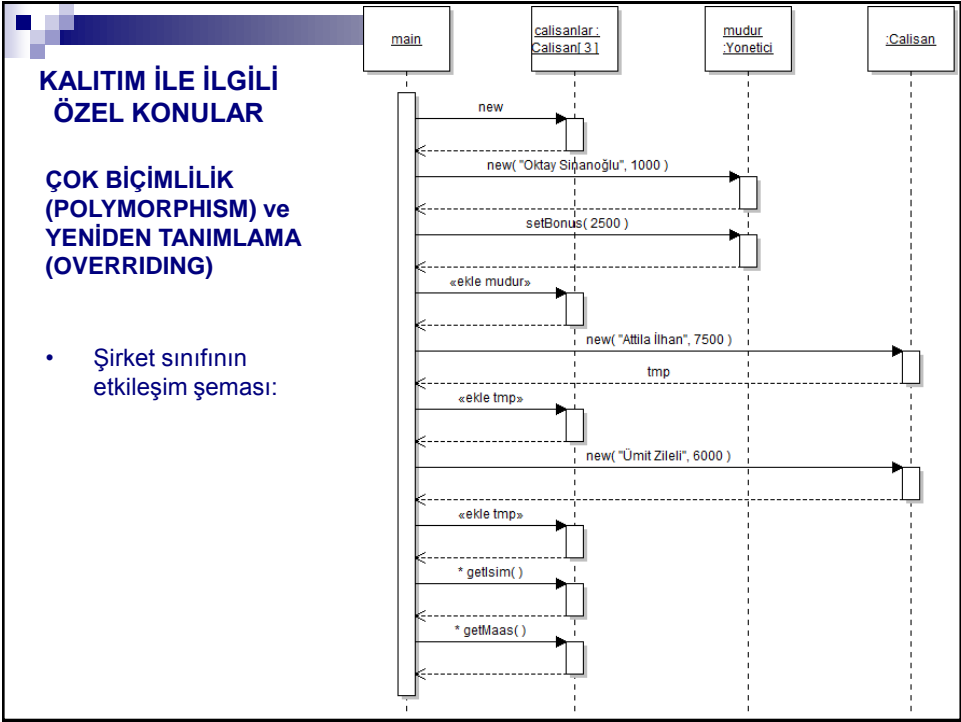
ÇOK BİÇİMLİLİK (POLYMORPHISM) ve YENİDEN TANIMLAMA (OVERRIDING)

- Kaynak kodlar:

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

package ders04;
public class Calisan extends Kisi {
    private int maas;
    public Calisan( String name, int maas ) {
        super( name );
        this.maas = maas;
    }
    public int getMaas() { return maas; }
    public void setMaas( int salary ) { this.maas = salary; }
}
```

74



NYP İLE İLGİLİ ÖZEL KONULAR

ADAŞ METOTLAR / ÇOKLU ANLAM YÜKLEME (OVERLOADING)

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

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

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

78

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.

79

NYP İLE İLGİLİ ÖZEL KONULAR

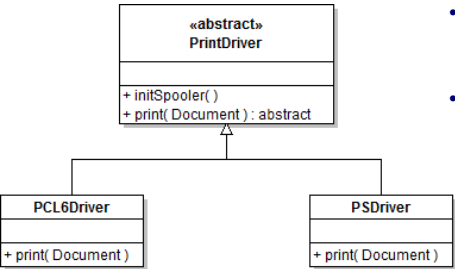
SOYUT SINIFLAR

- 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.

80

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 ilkendirileceğinin 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.

81

NYP İLE İLGİLİ ÖZEL KONULAR

SOYUT SINIFLAR

- Kaynak kodlar:

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

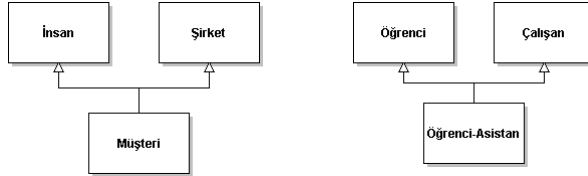
```
package ooc06;
public class PCL6Driver extends PrintDriver {
    public void print(Document doc) {
        //necessary code is inserted here
    }
}
```

82

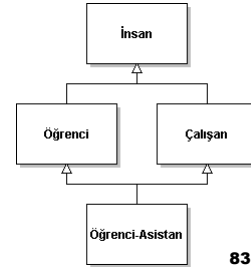
NYP İLE İLGİLİ ÖZEL KONULAR

ÇOKLU KALITIM

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



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

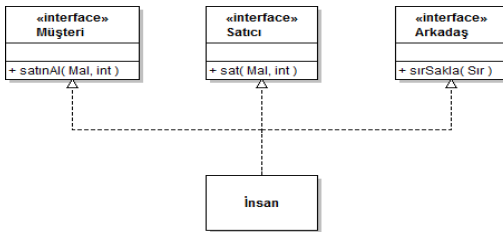


83

SPECIAL TOPICS in OOP

INTERFACES

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




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

Kodlama:

```

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

84




SPECIAL TOPICS in OOP

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.

85



SPECIAL TOPICS in OOP

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.
 - Arayüzlerin kurucusu olmaz.
 - Bir sınıf birden fazla arayüz gerçekleyebilir.

86

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

- Arayüz örneği: Araç – Araba – Uçak
 - UML sınıf şeması:
- Kaynak kodlar:

```
classDiagram
    class Arac {
        <<interface>>
        +getHizSiniri() double
    }
    class Araba {
        -hizSiniri double
        +getHizSiniri() double
        +setHizSiniri(double)
        +kimHizli(Arac) int
    }
    class Ucak {
        -hiz double
        -tavan double
        +Ucak(hiz double, tavan double)
        +getHiz() double
        +getTavan() double
        +getHizSiniri() double
    }
    Arac <|-- Araba
    Arac <|-- Ucak
```

```
package ders04.arayuzOrnegi;
public interface Arac {
    public double getHizSiniri( );
}

package ders04.arayuzOrnegi;
public class Ucak implements Arac {
    private double hiz; //Mach
    private double tavan; //ft.
    public Ucak(double hiz,double tavan){
        this.hiz = hiz;
        this.tavan = tavan;
    }
    public double getHiz( )
    { return hiz; }
    public double getTavan( )
    { return tavan; }
    public double getHizSiniri( )
    { return hiz * 1225; }
}
```

87

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

- Kaynak kodlar (devam):

```
package ders04.arayuzOrnegi;
public class Araba implements Arac {
    private double hizSiniri;

    public double getHizSiniri() {
        return hizSiniri;
    }

    public void setHizSiniri(double hizSiniri) {
        this.hizSiniri = hizSiniri;
    }

    public int kimHizli( Arac baska ) {
        if( hizSiniri < baska.getHizSiniri() )
            return -1;
        else if( hizSiniri == baska.getHizSiniri() )
            return 0;
        else
            return 1;
    }
}
```

88

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

- Kaynak kodlar (devam):

```
package ders03.iliskiler;
import ders04.arayuzOrnegi.*;

public class AnaProgram05 {

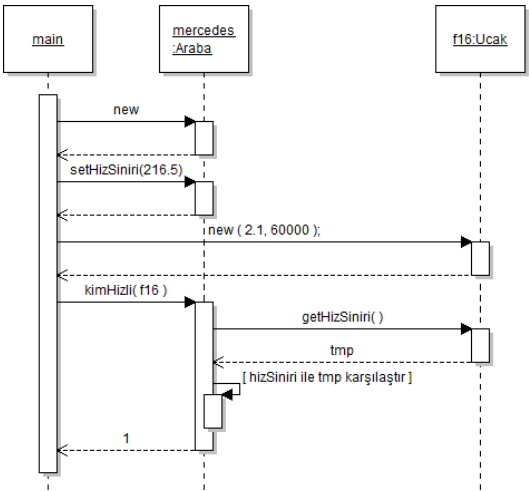
    public static void main(String[] args) {
        Araba mercedes = new Araba();
        mercedes.setHizSiniri(216.5);
        Ucak f16 = new Ucak( 2.1, 60000 );
        int sonuc = mercedes.kimHizli( f16 );
        if( sonuc == -1 )
            System.out.println("F-16 daha hızlı");
        else if( sonuc == 0 )
            System.out.println("İki aracın da hızı aynı");
        else
            System.out.println("Mercedes daha hızlı");
    }
}
```

89

NYP İLE İLGİLİ ÖZEL KONULAR

ARAYÜZLER (INTERFACE)

- Örnek programın sıralama şeması:



90

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

OBJECT SINIFI

- java.lang.Object sınıfı, tüm sınıfların üst sınıfıdır.
 - Siz isterseniz de, istemeseniz de. Yazsanız da, yazmasanız da.
- toString() : String metodunu yeniden tanımlayarak, nesneleri komut satırına daha kolay yazdırabilirsiniz.
- Örnek:

```
package not03a;
public class Calisan extends Kisi {
    //önceki koda ek olarak:
    public String toString() {
        return getIsim() + " " + getMaas() ;
    }
}

public class Sirket {
    //önceki kodda değişen kısım:
    public void calisanlariGoster() {
        for( Calisan calisan : calisanlar )
            if( calisan != null )
                System.out.println( calisan );
    }
}
```

91

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

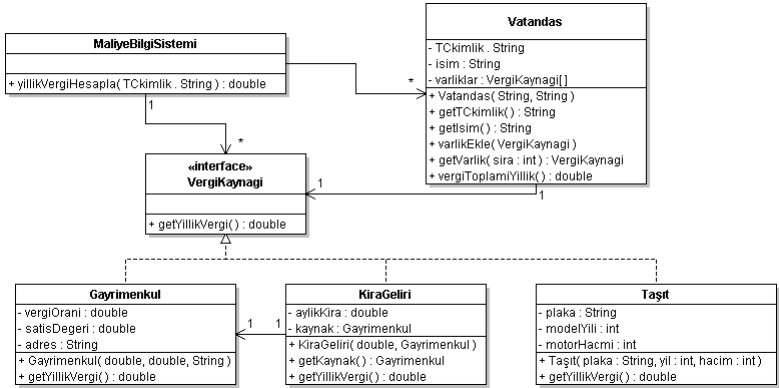
ARAYÜZLER (INTERFACE)

- Benzer biçimde davranan farklı tür varlıkları modellemek için tercih arayüzler tercih edilebilir.
 - Böylece hem kalıtımın “ağır sıklet” yaklaşımından kurtuluruz, hem de çok biçimliliği kullanabiliriz.
- Kurallar:
 - Arayüzlerde normal üye alan tanımlanmaz.
 - Ancak “final static” alan tanımlanır.
 - Arayüzlerde tanımlanan tüm üyeler public olmalıdır.
 - Bir sınıf birden fazla arayüzü gerçekleyebilir.
 - Arayüzler arasında kalıtım ilişkisi kurulabilir.
- Örnek:
 - Ev, araba, kira geliri, vb. varlıklar nedeniyle devlete yıllık vergi veriliyor.
 - Sözü geçen varlıkların tek ortak noktası, birer vergi kaynağı olmalarıdır.
 - Bu varlıkların başka ortak özellikleri yoktur, hepsi çok farklı iç işleyişe sahiptir.

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

ARAYÜZLER (INTERFACE)

- Örnek modelleme:



JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

ARAYÜZLER (INTERFACE)

- Kodlama:

```
package not03b;
public interface VergiKaynagi {
    public double getYillikVergi ( );
}
```

```
package not03b;

public class Gayrimenkul implements VergiKaynagi {
    private double vergiOrani, satisDegeri;
    @SuppressWarnings("unused")
    private String adres;

    public Gayrimenkul( double vergiOrani, double satisDegeri, String adres ) {
        this.vergiOrani = vergiOrani;
        this.satisDegeri = satisDegeri;
        this.adres = adres;
    }

    public double getYillikVergi() {
        return satisDegeri * vergiOrani;
    }
}
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

ARAYÜZLER

- Kodlama:

```
package not03b;
public class Vatandas implements VergiKaynagi {
    private String Tckimlik, isim;
    private VergiKaynagi varliklar[ ];
    public final static int maxVarlik = 20;
    private int varlikSayisi;

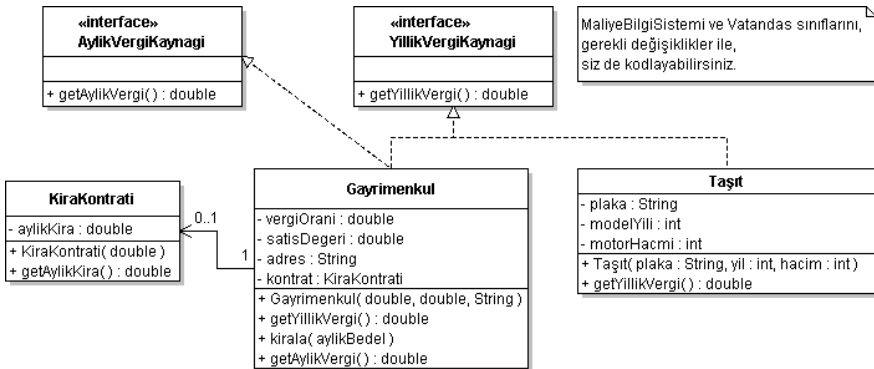
    public Vatandas(String Tckimlik, String isim) {
        this.Tckimlik = Tckimlik;
        this.isim = isim;
        varliklar = new VergiKaynagi[maxVarlik];
        varlikSayisi = 0;
    }
    public String getTckimlik() { return Tckimlik; }
    public String getIsim() { return isim; }
    public void varlikEkle( VergiKaynagi varlik ) {
        if( varlikSayisi < maxVarlik ) {
            varliklar[varlikSayisi] = varlik;
            varlikSayisi++;
        }
    }
    public VergiKaynagi getVarlik( int sıra ) {
        if( sıra < maxVarlik )
            return varliklar[sıra];
        else return null;
    }
    public double getYillikVergi() {
        int toplam = 0;
        for( int i = 0; i < varlikSayisi; i++ )
            toplam += varliklar[i].getYillikVergi( );
        return toplam;
    }
}
```

- Ödev / alıştıırma / lab:
- UML sınıf şemasına göre eksik kalan yerleri siz tamamlayın

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

ARAYÜZLER (INTERFACE)

- Bir sınıf birden fazla arayüzü gerçekleyebilir dedik.
- Örnek tasarımıımızı bu yeteneği göstermek üzere şu şekilde değiştirebiliriz:



- Sizce MaliyeBilgiSistemi ve Vatandaş sınıflarını UML sınıf şemasına nasıl ilişkilendirelim?

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

ARAYÜZLER (INTERFACE)

- Kodlama:

```
package not04a;
public interface YillikVergiKaynagi {
    public double getYillikVergi( );
}
```

```
package not04a;
public interface AylikVergiKaynagi {
    public double getAylikVergi( );
}
```

```
package not04a;
public class KiraKontrati {
    public final static double vergiOrani = 0.18;
    private double aylikKira;

    public KiraKontrati(double aylikKira) {
        this.aylikKira = aylikKira;
    }
    public double getAylikKira() {
        return aylikKira;
    }
}
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

ARAYÜZLER (INTERFACE)

- Kodlama:

```
package not04a;
public class Gayrimenkul implements YillikVergiKaynagi, AylikVergiKaynagi {
    private double vergiOrani, satisDegeri;
    @SuppressWarnings("unused")
    private String adres;
    private KiraKontrati kontrat;

    public Gayrimenkul( double vergiOrani, double satisDegeri, String adres ) {
        this.vergiOrani = vergiOrani;
        this.satisDegeri = satisDegeri;
        this.adres = adres;
    }

    public double getYillikVergi() { return satisDegeri * vergiOrani; }

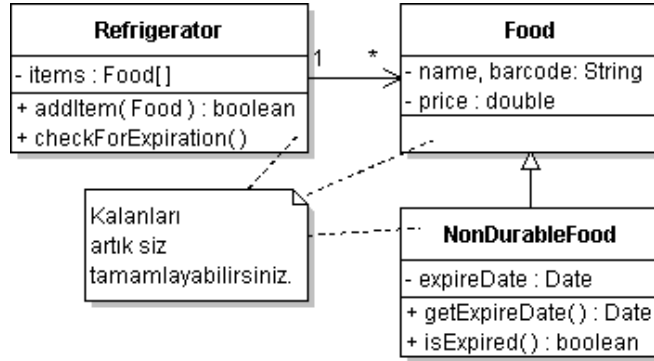
    public void kirala( double bedel ) { kontrat = new KiraKontrati(bedel); }

    @SuppressWarnings("static-access")
    public double getAylikVergi() {
        if( kontrat != null )
            return kontrat.getAylikKira() * kontrat.vergiOrani;
        else return 0;
    }
}
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

TİP DÖNÜŞÜMÜ (TYPE CASTING)

- Eğer şartlar elveriyorsa, bir tipten diğer tipe geçiş yapılabilir.
 - Arayüzden nesne tipine
 - Üst sınıftan alt sınıfa
 - Ancak önce denetim yapılmalıdır.
- Örnek:



JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

TİP DÖNÜŞÜMÜ (TYPE CASTING)

- Kodlar:

```

package not04b;
import java.util.*;

public class NonDurableFood extends Food {
    private Date expireDate;
    public NonDurableFood( Date expireDate, String name,
        double price, String barcode ) {
        super(name, price, barcode);
        this.expireDate = expireDate;
    }
    public Date getExpireDate() { return expireDate; }
    public boolean isExpired( ) {
        Date today = new Date();
        if( expireDate.before(today) )
            return true;
        else return false;
    }
}
  
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

TİP DÖNÜŞÜMÜ (TYPE CASTING)

- Kodlar:

```
package not04b;
import java.util.*;
public class Refrigerator {
    private Food[] items;
    public static final int maxItem = 10;
    private int itemCount = 0;
    public Refrigerator( ) {
        items = new Food[maxItem];
    }
    public boolean addItem( Food item ) {
        if( itemCount >= maxItem )
            return false;
        items[itemCount] = item; itemCount++;
        return true;
    }
}
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

TİP DÖNÜŞÜMÜ (TYPE CASTING)

- Kodlar:

```
public void checkForExpiration( ) {
    boolean hasExpiredItem = false;
    System.out.println("Expired item(s): ");
    for( int i=0; i<itemCount; i++ ) {
        if( items[i] instanceof NonDurableFood )
            if( ((NonDurableFood)items[i]).isExpired() ) {
                hasExpiredItem = true;
                System.out.println(items[i].getName());
            }
    }
    if( hasExpiredItem == false )
        System.out.println("All items are fresh!");
}
```

- Tip uygunluğu denetimi ve ardından tip dönüşümü

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

TİP DÖNÜŞÜMÜ (TYPE CASTING)

- Kodlar:

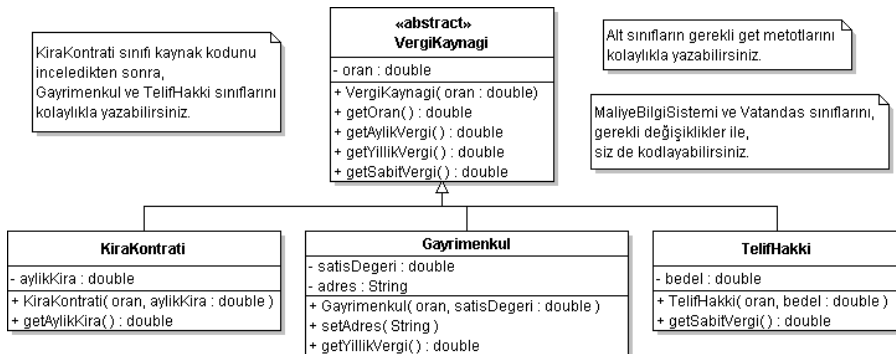
```
public static void main( String[] args ) {
    Refrigerator dolap = new Refrigerator();
    Calendar cal = Calendar.getInstance();
    cal.set(Calendar.YEAR, 2010);
    cal.set(Calendar.MONTH, 0); //Ocak
    cal.set(Calendar.DATE, 12);
    Date tarih = cal.getTime();
    dolap.addItem( new NonDurableFood(
        tarih, "Elma", 2.30, "689659877" ));
    dolap.addItem( new Food("Kola", 2.50, "869123654" ) );
    dolap.addItem( new Food("Sakız", 1.30, "689659877" ) );
    dolap.checkForExpiration();
}
```

- [java.util.Date ve Calendar sınıflarının kullanımı](#)

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

SOYUT (ABSTRACT) SINIFLAR

- Vergi örneğini biraz değiştirelim:



- Sizce MaliyeBilgiSistemi ve Vatandaş sınıflarını UML sınıf şemasına nasıl ilişkilendirelim?

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

SOYUT (ABSTRACT) SINIFLAR

- Kaynak kodlar:

```
package not04c;

public abstract class VergiKaynagi {
    private double oran;

    public VergiKaynagi(double oran) {
        this.oran = oran;
    }
    public double getOran() {
        return oran;
    }
    public abstract double getAylikVergi();
    public abstract double getYillikVergi();
    public abstract double getSabitVergi();
}
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

SOYUT (ABSTRACT) SINIFLAR

- Kaynak kodlar:

```
package not04c;
public class KiraKontrati extends VergiKaynagi {
    private double aylikKira;

    public KiraKontrati(double oran, double aylikKira) {
        super(orán);
        this.aylikKira = aylikKira;
    }
    public double getAylikVergi() {
        return aylikKira * getOran();
    }
    public double getYillikVergi() {
        return getAylikVergi() * 12;
    }
    public double getSabitVergi() {
        return 0;
    }
}
```

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

SOYUT (ABSTRACT) SINIFLAR

- Soyut üst sınıf ile ne kazandık?
 - Gayrimenkul ve TelifHakki sınıflarını kodlarken oranın belirlenmesi ile ilgili kodları kopyala-yapıştır yapmaktan kurtulduk.
 - Başka vergi kaynaklarını kodlayacak farklı programcıların, iş mantığına uymalarını mecbur kıldık.
 - Böylece mantıksal hata yapma olasılıkları azalacak.

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

SIRALAMA (ENUM) SINIFLARI

- İlkel sıralama tanımı aslında bir sınıf tanımıdır.
- Sıralamanın her elemanı o sınıfın bir örneğidir.
- Sıralama sınıfının tanımlananların dışında başka örneği de olamaz.
- Bir sıralama tipine herhangi bir sınıfa yapıldığı gibi üye alanlar ve metotlar eklenebilir.
- Sıralama kurucusu private olmalıdır.
- Örnek:

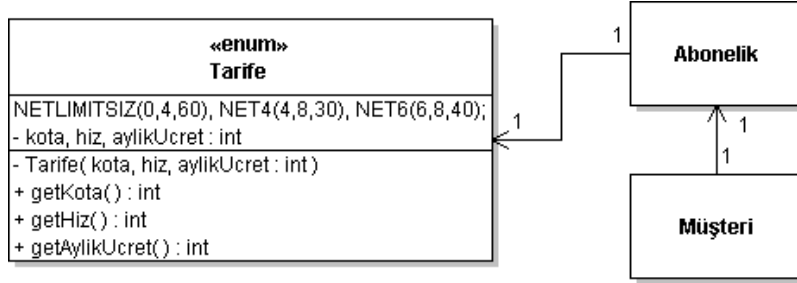
```
package not04d;
public enum Tarife {
    NETLIMITSIZ(0,4,60), NET4(4,8,30), NET6(6,8,40);
    private int kota, hiz, aylıkUcret;
    private Tarife( int kota, int hiz, int ucret ) {
        this.kota = kota; this.hiz = hiz; aylıkUcret = ucret;
    }
    public int getKota() { return kota; }
    public int getHiz() { return hiz; }
    public int getAylıkUcret() { return aylıkUcret; }
}
```

108

JAVA İLE NESNEYE YÖNELİK PROGRAMLAMA

SIRALAMA (ENUM) SINIFLARI

- Örnek:



109

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- Aykırı durum: Exception
- Aykırı durum işlenmesi: Exception Handling
- Çalışma anında ortaya çıkan beklenmedik durumlara denir.
- Aykırı durumlar neden ve nasıl oluşur?
 - JVM'deki hatalar
 - Kullanıcının hatalı bilgi girişi
 - Programcının hataları veya eksiklikleri
 - Dosya ve ağ işlemleri gibi, sürecin tamamının programcının denetiminde olamayacağı durumlarda ortaya çıkan beklenmedik sorunlar
 - Murphy yasaları, vb.

110

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- Denetlenen ve denetlenmeyen aykırı durumlar:
 - Programcının yapabileceği hatalardan oluşan aykırı durumlar denetlenmez (unchecked).
 - Diğer hatalar önceden sezilebilir ve denetlenmek zorundadır (checked).
 - Denetlenebilen aykırı durumların hangi komutların ardından ortaya çıkabileceği bellidir.
 - Bir denetlenebilen aykırı duruma yol açacak komut işleneceği zaman, o aykırı durumun programcı tarafından işlenmesi gerekir.

113

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- Denetlenen aykırı durumların işlenmesi:
 - try – catch blokları içerisinde yapılır.

```
try {  
    komutlar;  
}  
catch( AykırıDurumTipi e ) {  
    /* Aykırı durum oluşursa çalıştırılacak komutlar. */  
}
```
- Denetlenen bir aykırı durum işlenilmeyecekse bu durum bildirilmelidir:
 - Metot düzeyinde yapılır.

```
benimMetodum throws AykırıDurumTipi {  
    /* Aykırı durum oluşturabilecek komut. */  
}
```

114

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- Birden fazla aykırı durumu işlemek:
 - Bir try bloğu içerisinde farklı denetlenen aykırı durum türleri ortaya çıkaracak farklı komutlar olabilir.
 - Bir komut birden fazla türde aykırı durum oluşturabilir.
 - Bu durumlarda her aykırı durum türü için yeni bir catch bloğu eklenir.

```
try {  
    komutlar;  
}  
catch( AykırıDurumTip1 e ) {  
    /* Aykırı durum 1 oluşursa çalıştırılacak komutlar. */  
}  
catch( AykırıDurumTip2 e ) {  
    /* Aykırı durum 2 oluşursa çalıştırılacak komutlar. */  
}
```

115

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- try blokları hakkında:
 - Her yeni açılan try bloğu sisteme ek yük getirir.
 - Bu yüzden farklı aykırı durumlara yol açabilecek komutları tek bir try bloğu içerisinde toplamak yerinde olacaktır.
- catch bloklarında yapılabilecekler:
 - Kullanıcıya beklenmedik bir durum olduğunun bildirilmesi.
 - Komut satırı penceresine aykırı durumu oluşturan koşulların yazdırılması (e.printStackTrace() ile).
 - Bazı kaynakların serbest bırakılması:
 - finally bloğu içerisinde.
 - Bunu gerektiren kaynaklar enderdir.
- Sonraki yansıda kısmi bir örnek vardır.
 - Gerçek dünya örneğini, hem biraz ileride hem de dosya işlemleri sırasında göreceksiniz.

116

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

```
public void dosyadanSekilCiz( String dosyaAdi ) {
    InputStream dosya;
    Graphics g = anaPencere.getGraphics();
    try {
        dosya = dosyaAc( dosyaAdi );
        sekilCiz( dosya, g );
    }
    catch( IOException e ) {
        e.printStackTrace();
        JOptionPane.showMessageDialog( null, "Dosya açarken
            bir hata oluştu:" + e.getMessage(),
            "Dosya hatası", JOptionPane.ERROR_MESSAGE );
    }
    finally {
        dosya.close();
        g.dispose();
    }
}
```

117

AYKIRI DURUMLAR

KENDİ AYKIRI DURUMLARINIZI HAZIRLAMAK

- Uygun bir hazır aykırı durum sınıfından kalıtım ile.
 - Denetlenen yapıda ise: IOException'dan.
 - Denetlenmeyen yapıda ise: RuntimeException'dan.

```
public class FileFormatException extends IOException {
    public FileFormatException( ) {
    }
    public FileFormatException( String hataMesaji ) {
        super( hataMesaji );
        /* Ayrıca yapmak istediğiniz işler */
    }
}
```

118

AYKIRI DURUMLAR

ÇALIŞMA ANINDA AYKIRI DURUM ÜRETMEK

- Başkalarının kullanımına yönelik kod hazırlıyorsanız:
 - Denetlenen aykırı durumları kendiniz işlemeyip, bu işi programınızı kullanana bırakabilirsiniz.
 - Nasıl yapılıyordu? Hatırlayın: throws bildirimi ile.
 - Programınızın içerisinde beklenmedik bir durumla karşılaşırsanız, throw deyimi ile çalışma anında bir aykırı durum üretebilirsiniz.
 - Hazır veya kendinizin hazırladığı bir aykırı durum üretilir.

```
public class BirProgram {
    public void dosyaIsle( ) {
        komutlar();
        if( beklenmedik bir durum )
            throw new FileFormatException("... oldu");
    }
}
```

119

AYKIRI DURUMLAR

AYKIRI DURUMLARLA ÇALIŞMAK

- Şimdiye dek gördüklerimizin tümünü içeren bir örnek verelim.

```
package not06a;
public class Kisi {
    private String isim;
    private int yas;

    public Kisi( String name ) { this.isim = name; }
    public String getIsim( ) { return isim; }
    public int getYas( ) { return yas; }

    public void setYas( int yas ) throws OlanaksizBilgi {
        if( yas < 0 || yas > 150 )
            throw new OlanaksizBilgi( );
        else
            this.yas = yas;
    }
}
```

120

AYKIRI DURUMLAR

AYKIRI DURUMLARLA ÇALIŞMAK

- Kod (devam)

```
package not06a;

import java.io.IOException;

/* Aslında RuntimeException'dan türetmek
 * daha doğru olurdu ama diğer örnek zaten
 * denetlenmeyen yapıda olacağı için bari
 * bu örnek denetlensin dedik.
 */
@SuppressWarnings("serial")
public class OlanaksizBilgi extends IOException {
    public OlanaksizBilgi( ) { }
    public OlanaksizBilgi( String hataMesaji ) {
        super(hataMesaji);
    }
}
```

121

AYKIRI DURUMLAR

AYKIRI DURUMLARLA ÇALIŞMAK

- Kod (devam)
 - Örnekte yanlış bilgi girişinin düzeltilmesi istenmiyor, sadece yanlış yapıldığı bildiriliyor.
 - Kullanıcı doğru giriş yapana kadar ısrar edilmesi için bir do daha gerek.
 - Onu da size alıştıрма olarak bırakıyorum.

```
package not06a;
import java.util.*;

public class Test {
    public static void main(String[] args) {
        Scanner girdi = new Scanner(System.in);
        String isim, devam;
        int yas;
        do {
            System.out.print("Kişinin ismini giriniz: ");
            isim = girdi.nextLine();
            Kisi insan = new Kisi(isim);
            System.out.print("Kişinin yaşını giriniz: ");
```

122

AYKIRI DURUMLAR

AYKIRI DURUMLARLA ÇALIŞMAK

- Kod (devam)

```
try {
    yas = girdi.nextInt( );
    insan.setYas(yas);
    System.out.println(insan.getIsim()+" (" +insan.getYas()+" )");
}
catch( OlanaksizBilgi e ) {
    System.out.println("HATA: Olanaksiz bilgi girişi.");
    e.printStackTrace();
}
catch( InputMismatchException e ) {
    System.out.println("HATA: Yaş için sayı girilmeli.");
    e.printStackTrace();
}
girdi.nextLine( );
System.out.print("Devam (e/h)? ");
devam = girdi.nextLine( );
} while( devam.compareToIgnoreCase("h") != 0 );
}
```

123

DOSYALARLA ÇALIŞMAK

KARŞILAŞILABİLECEK AYKIRI DURUMLAR

- java.io.IOException: Genel G/Ç hatalarını simgeler.
- java.io.EOFException extends IOException: Beklenmedik bir nedenle dosyanın veya akının sonuna gelindiğini gösterir.
- java.io.FileNotFoundException extends IOException: İstenen dosyanın verilen yolda bulunamadığını gösterir.
- java.lang.SecurityException extends java.lang.RuntimeException: İstenen dosya işleminin güvenlik kısıtlamaları nedeniyle yapılamadığını gösterir.

DOSYA İŞLEMLERİ HAKKINDA GENEL BİLGİ

- Java dilinde dosya işlemleri iki ana gruba ayrılmıştır:
 - Dosya yönetim işlemleri: Dizin ve dosyaları oluşturmak, yeniden adlandırmak, silmek, vb.
 - G/Ç işlemleri.
- G/Ç işlemleri sadece dosyalar üzerinde değil, farklı kaynaklar üzerinde de yapılır: TCP soketleri, web sayfaları, komut satırı, vb.
 - Bu nedenle G/Ç işlemleri, dosya işlemlerinden ayrılmıştır ve adı geçen farklı kaynaklar üzerinde de aynen dosyalar üzerinden icra ediliyormuş gibi yapılmaktadır.
- Nesneye yönelik düşünme biçiminin iyi bir uygulaması olan bu yaklaşım, karmaşıklığı arttırmıştır.

124

DOSYALARLA ÇALIŞMAK

DOSYA YÖNETİMİ

- Diskteki dizin ve dosyaları simgeleyen java.io.File sınıfı ile yapılır.
- Bir File nesnesi oluşturmak, diskte fiziksel bir nesne oluşturmak demek DEĞİLDİR!
- Bir dosya nesnesi oluşturmak:
 - File(String dosyaAdi) kurucusu ile.
 - dosyaAdi, dosyanın hem yolunu hem de adını içermelidir.
 - Tam yol veya göreceli yol.
 - Göreceli yol kullanırken dikkat: Geliştirme ortamları kaynak kod ve derlenmiş kodu ayrı dizinlerde tutabilir.
 - Dizin ayırıcı:
 - Windows'ta \, ancak karakter katarı içerisinde \\ kullanımı.
 - Unix'te /
 - Ya taşınabilirliğe ne oldu?
 - public static String File.separator ve public static char File.separatorChar üyeleri.
 - File(String dizin, String isim) ve File(File dizin, String isim) kurucuları ile:
 - Verilen dizinin altındaki verilen isimli bir dosya veya dizini simgeler.
 - File(String) kurucusunun kuralları aynen geçerlidir.

125

DOSYALARLA ÇALIŞMAK

DOSYA YÖNETİMİ

- java.io.File sınıfının diğer metotlarından bazıları:
 - boolean exists(); Dosyanın var olup olmadığı.
 - boolean isFile(); Bu File nesnesi bir dosyayı simgeliyorsa true döner aksi halde dizin demektir ve false döner.
 - File getParentFile(); Dosya ise dosyanın bulunduğu dizini, dizin ise üst dizini, üst dizin dosya yoksa null nesneyi döndürür.
 - String getCanonicalPath() throws IOException; Dosya/dizinin tam yolunu döndürür (dosya adı dahil).
 - boolean canRead(); Çalışan uygulama bu dosyayı okuyabilir mi?
 - boolean canWrite(); Çalışan uygulama bu dosyaya yazabilir mi?
 - boolean createNewFile(); Dosyayı diskte oluşturur,
 - boolean mkdir(); Dizini diskte oluşturur,
 - boolean mkdirs(); Dizini gerekli üst dizinleri ile birlikte diskte oluşturur,
 - boolean renameTo(File yeni); Dizini diskte oluşturur
 - boolean delete(); Dosyayı siler
- boolean dönüşler: İşlem başarılı ise true döner.
- Buradaki metotların ezberlenmesine gerek yoktur.

126

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Herhangi bir G/Ç kaynağı, Java dilinde bir akı (stream) ile temsil edilir.
 - Dosya, bellek, komut satırı, ağ, ... Java'da hepsi birer G/Ç kaynağıdır.
- Okunabilirlik ve başarıma göre temel çalışma biçimleri:
 - İkili (Binary) düzen: Hızlı ancak bir insan tarafından okunamaz.
 - Karakter düzeni: Bir insan tarafından okunabilir ama daha yavaş.
 - Kayıt düzeni: Bileşik kayıtlar şeklinde G/Ç işlemleri (Pascal: record, C:Struct, Java: nesneler)
- Erişim düzenine göre temel çalışma biçimleri:
 - Sıralı/ardışıl erişim (sequential access): Tüm kayıtlar sıra ile okunur.
 - Rastgele (random) erişim: Herhangi bir kayıda doğrudan erişilebilir.
- Java, farklı çalışma biçimleri için farklı akıları, birbirleri ile zincirleyerek kullanır.
- Derste, nesneleri bir bütün olarak işlemeye yarayan kayıt düzeni işlenecektir.
 - Java terminolojisinde bu işleme serileştirme (Serialization) adı verilir.

127

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Serileştirme – Çıktı işlemleri:
 - Nesneleri bir dosyaya yazma işlemleri
 - Serileştirilecek nesneler, java.io.Serializable arayüzünü gerçekleştirmelidir.
 - Programcının arayüzdeki metotları yeniden tanımlamasına gerek yoktur.
 - ObjectOutputStream ve FileOutputStream nesneleri zincirlenerek kullanılır.
 - Aynı akıya birden fazla nesne kaydedilebilir.
 - Zaten birbirleri ile ilişkili nesneler aynı dosyaya kaydedilmelidir, aksi halde 'işaretçi kırılması' olayı yaşanır.

128

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Örnek kayıt: Arkadas sınıfı

```
package not07a;
public class Arkadas implements java.io.Serializable {
    private static final long serialVersionUID = 1L;
    private String isim, telefon, ePosta;
    public Arkadas( String name ) { this.isim = name; }
    public String getIsim( ) { return isim; }
    public String getTelefon( ) { return telefon; }
    public void setTelefon( String telefon ) {
        this.telefon = telefon;
    }
    public String getEPosta( ) { return ePosta; }
    public void setEPosta( String posta ) { ePosta = posta; }
    public String toString( ) {
        return isim + " - " + telefon + " - " + ePosta;
    }
}
```

129

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- @ ile başlayan satırlar hakkında:
 - Şimdiye kadar çeşitli program kodlarında bunları gördük.
 - Bunlara “annotation” adı verilir.
 - “Meta” yani “bilgi hakkında bilgi” düzeyinde komutlardır.
 - IDE’ye, derleyiciye, başka bir programa, vb. bu program hakkında bilgi vermeye yarar.
 - Bu derste “annotation” düzeneğini uyarı mesajlarını (warning) kaldırmaya yönelik olarak kullandık.
 - Aslında uyarılar dikkate alınmalıdır, ancak derste verilen örneklerin belli bir konuya odaklanması ve konunun dağılmaması için bu yola gidilmiştir.
 - Önceki örnekte, uyarı konu ile tam ilgili olduğu için, bu yola gidilmemiştir.

130

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- seri numara üyesi hakkında:
 - `private static final long serialVersionUID = 1L;`
 - 1 yerine kendimiz bu sınıfa vereceğimiz sürüm numarasını (version) kullanabilir veya IDE'nin otomatik bir tekil numara üretmesini sağlayabiliriz.
 - Bu üyenin kodlanmazsa `@SuppressWarnings("serial")` komutu ile ilgili uyarı gizlenebilir.
 - Bu üye ne işe yarar?
 - Bir sınıftan nesneleri bir dosyaya kaydeden ve dosyadan okuyan programlar olacak.
 - Zamanla kaydedilen nesnelerin ait olduğu sınıfının kaynak kodunu değiştirilip yeni üyeler eklenip çıkarılabilir.
 - Zamanla nesneleri okuyan ve kaydeden programlar da değişebilir.
 - Tüm söz konusu sınıf, dosya ve kodların eski ve yeni sürümleri ortalıkta dolaşabilir.
 - Bunların birbirleri ile uyumsuzluklarını önlemek için, seri numara üyesi bize bir fırsat sunar.

131

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Kayıtları dosyaya yazan program:

```
package not07a;
import java.util.*;
import java.io.*;
public class ArkadasOlustur {
    public static void main(String[] args) {
        Integer arkadasSayisi;
        Arkadas[] arkadaslar;
        Scanner giris = new Scanner( System.in );
        System.out.println("Bu program arkadaşlarınızın iletişim " +
            " bilgilerini diskteki bir dosyaya kaydeder.");
        System.out.print("Kaç arkadaşınızın bilgisini gireceksiniz? ");
        arkadasSayisi = giris.nextInt( );
        giris.nextLine( );
        arkadaslar = new Arkadas[arkadasSayisi];
        for( int i = 0; i < arkadasSayisi; i++ ) {
            System.out.print((i+1)+". arkadaşınızın ismi nedir? ");
            arkadaslar[i] = new Arkadas( giris.nextLine() );
            System.out.print("Bu arkadaşınızın telefonu nedir? ");
            arkadaslar[i].setTelefon( giris.nextLine() );
            System.out.print("Bu arkadaşınızın e-posta adresi nedir? ");
            arkadaslar[i].setEPosta( giris.nextLine() );
        }
    }
}
```

132

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Kayıtları dosyaya yazan program:

```
try {
    String dosyaAdi = "arkadaslar.dat";
    ObjectOutputStream yazici = new ObjectOutputStream(
    new FileOutputStream( dosyaAdi ) );
    yazici.writeObject( arkadasSayisi );
    for( Arkadas arkadas : arkadaslar )
        yazici.writeObject( arkadas );
    System.out.println("Girilen bilgiler " + dosyaAdi +
        " adlı dosyaya başarıyla kaydedildi.");
}
catch( IOException e ) {
    System.out.println("Dosyaya kayıt işlemi sırasında"+
        " bir hata oluştu.");
    e.printStackTrace();
}
}
```

133

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Serileştirme – Girdi işlemleri:
 - Nesneleri bir dosyadan okuma işlemleri
 - ObjectInputStream ve FileInputStream nesneleri zincirlenerek kullanılır.
 - Okunan nesneler Object türünde olduğu için, ait oldukları tipe dönüştürülmek zorundadır (typecasting).
 - Okunan nesneler bir dizide saklanacaksa kaç nesne okunacağı bilinmek zorundadır.
 - Dinamik olarak boyutu değişebilen veri yapılarında buna gerek yoktur.

134

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Kayıtları dosyadan okuyan program:

```
package not07a;
import java.io.*;

public class ArkadasGoster {
    public static void main( String[] args ) {
        String dosyaAdi = "arkadaslar.dat";
        try {
            ObjectInputStream okuyucu = new ObjectInputStream(
            new FileInputStream( dosyaAdi ) );
            Integer kayitSayisi = (Integer)okuyucu.readObject();
            for( int i = 0; i < kayitSayisi; i++ ) {
                Arkadas arkadas = (Arkadas) okuyucu.readObject();
                System.out.println(arkadas);
            }
        }
    }
}
```

135

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Kayıtları dosyadan okuyan program:

```
catch( IOException e ) {
    System.out.println("Dosya okuma işlemleri sırasında " +
        " bir hata oluştu.");
    e.printStackTrace();
}
catch( ClassNotFoundException e ) {
    System.out.println("Okunan kayıtları işlerken " +
        "bir hata oluştu.");
    e.printStackTrace();
}
}
```

136