011 2562 NESNEYE DAYALI PROGRAMLAMA – Şubat 2014
Yrd.Doç.Dr. Yunus Emre SELÇUK
GENEL BİLGİLER

BAŞARIM DEĞERLENDİRME

- Puanlama (değişebilir):
  - 1. Vize: %20, 2. Vize: %20, Final: %30, Proje: %15, Lab: %15
- Sınav ve proje tarihleri: Daha sonra duyurulacak
  - Nesneye Dayalı Kavramlar dersinde öğrendiklerinizin aksine yapacağınız hatalardan puan kırılacaktır.

KAYNAKLAR:

- Core Java 2 Volume I – Fundamentals, 8th ed. (2007), C. S. Horstmann and G. Cornell, Prentice-Hall
- Core Java 2 Volume II – Advanced Features, 8th ed. (2008), C. S. Horstmann and G. Cornell, Prentice-Hall
  - Core Java mevcut son basımlarında dosya işlemleri 2. cilde taşınmıştır. 7th ed.'de ise sadece Vol.I yeterlidir.
  - 8th ed. Vol.I ve II, Merkez Kütüphane'mizde (Davutpaşa) 3 kopya olarak mevcuttur.
  - Öğrenciler başka Java kitaplarından da yararlanabilir.

1

GENEL BİLGİLER

DERS İÇERİĞİ

- Java ile Nesneye Yönelik Programlama:
  - İlkeller ve sarmalayıcılar
  - Özel durumların denetimi / Aykırı durum işleme (exception handling)
  - Generic sınıflarla temel veri yapıları
  - Tip dönüşümü
  - Dosya işlemleri (serialization)
  - Enum yapıları
  - İç sınıflar
  - Çok izlekli çalışma (multithreading)
  - GUI programlamaya giriş

2

İLKELLER VE SARMALAYICILAR

İLKEL VERİ TİPLERİ

- İlkel: Tek bir birim bilgi taşıyan değişken.
- Sarmalayıcı: Üye alan olarak tek bir ilkel taşıyan ve bu ilkel ile ilgili yararlı metotlar içeren sınıf.
- Java'da doğal (ondalıksız) sayı ilkelleri ve sarmalayıcıları:

Ad	Anlam	Aralık	Sarmalayıcı
int	Tam sayı (4 sekizlik)	– 2.147.483.648 ile + 2.147.483.647 arası (± 2 milyar)	Integer
long	Büyük tam sayı (8 sekizlik)	(± 9,22 x 10^{18})	Long
short	Küçük tam sayı (2 sekizlik)	– 32.768 ile + 32.767 arası	Short
byte	Sekizlik	– 128 ile + 127 arası	Byte

3

İLKELLER VE SARMALAYICILAR

İLKEL VERİ TİPLERİ

- Java'da gerçek (ondalıklı) sayı ilkelleri ve sarmalayıcıları:

Ad	Anlam	Aralık	Sarmalayıcı
double	Büyük ondalıklı sayı	**(± 1,79 E 308)**	Double
float	Küçük ondalıklı sayı	**(± 3,4 E38)**	Float

- Java'daki diğer ilkeller ve sarmalayıcıları:

Ad	Anlam	Aralık	Sarmalayıcı
char	Karakter	'A'-'Z', 'a'-'z', vb. (UTF-16 kodlama)	Char
boolean	Mantıksal	false – true	Boolean

- İlkel olmayan String sınıfını da burada hatırlatmak isterim.

4

İLKELLER VE SARMALAYICILAR

İLKEL VERİ TİPLERİ

- İlkeller ile işlemler:
  - Aritmetik: + - * / %.
    - İşlem önceliği
    - ++, --,
    - ++i ile i++ farkı: y * ++z, y * z++
    - Atama ve işlem: += –= *= /= %=
    - Anlaşılabilirlik için işi sade tutun, abartmayın
    - java.lang.Math sınıfının static metotları: pow, abs, round, …
  - İkili (Binary)
    - Boole cebiri: & | ~ ^ (and or not xor)
    - Basamak kaydırma: << >>
    - Ör: n sayısının sağdan 4. biti: (n&8)/8 veya (n&(1<<3))>>3

5

İLKELLER VE SARMALAYICILAR

- Bir ilkelin bir nesne gibi kullanılması gerektiği durumlar ortaya çıktığında, ilgili sarmalayıcı (wrapper) sınıf kullanılır.
  - Örneğin serileştirmede ve Map yapısı anahtarı olarak (ileride anlatılacak).
- Sarmalayıcılar java.lang paketinde yer alır (import'a gerek yok).
- Örnek: Integer sınıfının bazı metotları (daha fazlası için javadoc'a bakınız):
  - int compareTo(Integer anotherInteger)
  - int intValue()
  - static int parseInt(String s)
  - String toString()
  - static String toString(int i)
  - static Integer valueOf(String s)

```
int ilkel1 = 5, ilkel2 = 7;
Integer sarma1, sarma2;
sarma1 = new Integer( ilkel1 );
sarma2 = new Integer( ilkel2 );
System.out.println("Sonuç1:"+sarma1.compareTo(sarma2)); //-1
```

6

İLKELLER VE SARMALAYICILAR

SARMALAYICI SINIFLAR (WRAPPERS)

- Örnek: String sınıfının bazı metotları:
  - int compareTo(String anotherString)
  - int compareToIgnoreCase(String str)
  - String concat(String str) //+ işlemi!
  - int indexOf (String str) //-1, 0, …
    - "Selçuk". indexOf ("ç") = 3
  - int length()
  - String substring(int m, int n) //m ile n-1. (çünkü indexOf sıfırdan başlıyor)
    - "Selçuk".subString(0,3) = "Sel"
  - String toLowerCase()
  - String toUpperCase()
  - String trim()
- System.out.println(String)
  - print / println
- Not: Bu metotların nasıl çalıştığını anlamakta zorluk çekiyorsanız Nesneye Dayalı Kavramlar dersinin temellerini iyi anlayamamış olabilirsiniz!

7

İLKELLER VE SARMALAYICILAR

İLKEL SIRALAMA (ENUM: ENUMERATION)

- Sınırlandırılmış bir değerler kümesi tanımlamaya yarar.
- Alacağı sıralı değerler açık olarak verilmiş veri türüdür.

```
public enum Yon {
    YUKARI, ASAGI, SAG, SOL
};
```

- Yukarıdaki örnek, Yon türünden nesneler sadece YUKARI, ASAGI, SAG ve SOL değeri içerebilir anlamına geliyor.

```
Yon yon = Yon.YUKARI; //Doğru
Yon olmaz = Yon.KUZEY; //Derleme hatası
Yon hata = Yon.Sag; //Derleme hatası
```

- Tanımlama işlemi sınıf düzeyinde yapılmalıdır (metot içerisinde yapılamaz).
- Enum sınıfları da tanımlanabilir (ileride anlatılacak).

8

PRIMITIVES AND METHOD PARAMETERS

METOT PARAMETRELERİ

- İlkeller değer ile, nesneler işaretçilerinin değeri ile aktarılır.
  - Bu söz C dilindeki “Call-by-value” ve “Call-by-reference” gibi gelebilir ama değildir.
    - Çünkü yukarıda ne yazıyor? “Call-by-value-of-references”.
    - Call-by-value: Değer ile aktarım – kalıcı olmayan aktarım – geçici aktarım – vb.
    - Call-by-reference: İşaretçi ile aktarım – kalıcı aktarım – pointer ile aktarım – adres aktarımı – vb.
  - Kelimelerle anlatılması zor bir kavramdır, ama yine de deneyelim:
    - Değer ile aktarılan parametrelerde, metot içerinde yapılan değişiklik kalıcı olmaz.
    - Sarmalayıcılarda operatör kullanımı ile yapılan değişiklikler de bu şekildedir.
    - Nesnelere setter/getter metotları ile yapılan değişiklikler kalıcıdır.
    - Ayrı swap metodu mümkün değil, ancak setter/getter metotları ile veya “yerinde (inline)” olarak mümkündür.

9

DENETİM AKIŞI

METOT PARAMETRELERİ

- Anlaşılmadı mı? Örneklerle görelim.
  - Kalıcı olmayan örnekler:

```
package not02b;
public class Gecici {
          public void ilkelDuzenle( int x ) { x++; }
          public void sarmalayiciDuzenle( Integer x ) { x++; }
          public void ilkelDegistir( int x, int y ) {
                    int temp;
                    temp = x;
                    x = y;
                    y = temp;
          }
          public void sarmalayiciDegistir( Integer x, Integer y ) {
                    Integer temp; temp = x; x = y; y = temp;
          }
          public void sarmalayiciDegistirAlt(Integer x, Integer y) {
                    Integer temp;
                    temp = new Integer(x);
                    x = new Integer(y);
                    y = new Integer(temp);
          }
          public static void main(String[] args) {
                    //Kendi kendisini deneyen sınıf!
                    Gecici nesne = new Gecici( );
                    nesne.geciciDenemeler( );
          }
```

10

DENETİM AKIŞI

METOT PARAMETRELERİ

- Kalıcı olmayan örneğin devamı:

```
        public void geciciDenemeler( ) {
                int sayi = 3;
                System.out.println("Önce : " + sayi );
                this.ilkelDuzenle(sayi);
                System.out.println("Sonra: " + sayi );

                Integer sarma = new Integer( 5 );
                System.out.println("Önce : " + sarma );
                this.sarmalayiciDuzenle(sarma);
                System.out.println("Sonra: " + sarma );

                int sayi1 = 1, sayi2 = 2;
                System.out.println("Önce : " + sayi1 + ", " + sayi2 );
                this.ilkelDegistir(sayi1, sayi2);
                System.out.println("Sonra: " + sayi1 + ", " + sayi2 );

                Integer sarma1 = new Integer( 1 );
                Integer sarma2 = new Integer( 2 );
                System.out.println("Önce : " + sarma1 + ", " + sarma2 );
                this.sarmalayiciDegistir(sarma1, sarma2);
                System.out.println("Sonra: " + sarma1 + ", " + sarma2 );

                System.out.println("Önce : " + sarma1 + ", " + sarma2 );
                this.sarmalayiciDegistirAlt(sarma1, sarma2);
                System.out.println("Sonra: " + sarma1 + ", " + sarma2 );
        }
}
```


DENETİM AKIŞI

METOT PARAMETRELERİ

- Kalıcı olmayan örneğin çıktısı:

```
Önce : 3
Sonra: 3
Önce : 5
Sonra: 5
Önce : 1, 2
Sonra: 1, 2
Önce : 1, 2
Sonra: 1, 2
Önce : 1, 2
Sonra: 1, 2
```

DENETİM AKIŞI

METOT PARAMETRELERİ

- Kalıcı olan örnek:

```
package not02b;
public class Kalici {
          public static void main(String[] args) {
                    Kalici nesne = new Kalici( );
                    nesne.kaliciDenemeler( );
          }
          public void kaliciDenemeler( ) {
                    int x = 1, y = 2;
                    System.out.println("Önce : " + x + ", " + y );
                    int temp;
                    temp = x;
                    x = y;
                    y = temp;
                    System.out.println("Sonra: " + x + ", " + y );

                    Integer sarma1 = new Integer( 3 );
                    Integer sarma2 = new Integer( 5 );
                    System.out.println("Önce : " + sarma1 + ", " + sarma2 );
                    Integer gecici = sarma1;
                    sarma1 = sarma2;
                    sarma2 = gecici;
                    System.out.println("Sonra: " + sarma1 + ", " + sarma2 );
          }
}
```

13

DENETİM AKIŞI

METOT PARAMETRELERİ

- Kalıcı örneğin çıktısı:

```
Önce : 1, 2
Sonra: 2, 1
Önce : 3, 5
Sonra: 5, 3
```

14

DENETİM AKIŞI

METOT PARAMETRELERİ

- Bir başka kalıcı örnek ve çıktısı:

```
package not02b;
public class TestIt {
          public static void main(String[] args) {
                    int[] dizi = { 1, 2, 3, 4, 5 };
                    LowHighSwap.doIt( dizi );
                    for( int j = 0; j < dizi.length; j++ )
                              System.out.print( dizi[j] + " " );
          }
}
class LowHighSwap {
          static void doIt( int[] z ) {
                    int temp = z[ z.length - 1 ];
                    z[ z.length - 1 ] = z[ 0 ];
                    z[ 0 ] = temp;
          }
}
/* Bu örneğin tümünde static olmanın
 * kalıcılıkla hiçbir ilgisi yoktur. */
```

```
5 2 3 4 1
```

15

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

AYKIRI DURUM İŞLEME (EXCEPTION HANDLING)

16

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

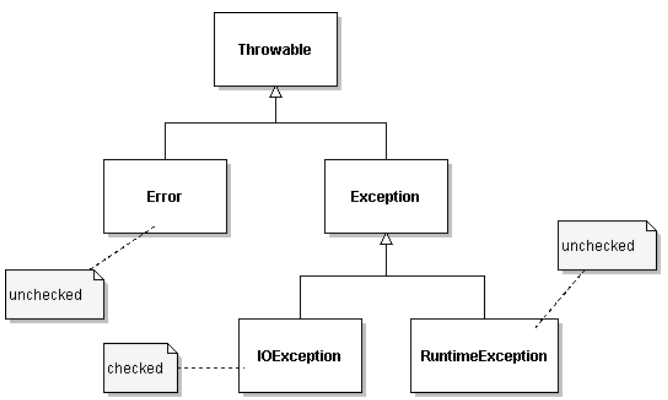
- Aykırı durum: Exception
- Aykırı durum işlenmesi: Exception Handling
- Çalışma anında ortaya çıkan beklenmedik durumlara denir.
- Aykırı durumlar neden ve nasıl oluşur?
  - JVM'deki hatalar
  - Kullanıcının hatalı bilgi girişi
  - Programcının hataları veya eksiklikleri
  - Dosya ve ağ işlemleri gibi,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

17

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- Java'da her aykırı durum bir sınıf ile modellenmiştir:



18

AYKIRI DURUMLAR

AYKIRI DURUMLARIN YAKALANMASI VE İŞLENMESİ

- java.lang.Error:
  - Sistem kaynaklarının tükenmesi veya JVM'nin içsel hataları sonucu oluşur.
  - Ender ortaya çıkar.
- java.lang.RuntimeException:
  - Programcının yaptığı bir programlama hatası nedeniyle ortaya çıkar.
  - Örneğin:
    - Tanımlanmış ama oluşturulmamış bir nesneye erişim (java.lang.NullPointerException)
    - Dizinin boyutundan daha büyük bir dizin numarasına erişim (java.lang.IndexOutOfBoundsException)
- java.io.IOException:
  - Dosya veya ağ işlemlerinde bir şeylerin ters gitmesi sonucu oluşur.
  - Programcının hatası değildir.

19

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.

20

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. */
}
```

21

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( AykırıDurumTipi1 e ) {
    /* Aykırı durum 1 oluşursa çalıştırılacak komutlar. */
}
catch( AykırıDurumTipi2 e ) {
    /* Aykırı durum 2 oluşursa çalıştırılacak komutlar. */
}
```

22

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.

23

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();
    }
}
```

24

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 */
    }
}
```

25

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 üretilebilir.

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

26

AYKIRI DURUMLAR

AYKIRI DURUMLARLA ÇALIŞMAK

- Şimdiye dek gördüklerimizin tümünü içeren bir örnek verelim.

```
package not06a;

import java.io.IOException;

/* Eğer RuntimeException'dan türetsek
 * tüm kodlarımız nasıl olacaktı araştırınız.
 */
@SuppressWarnings("serial")
public class OlanaksizBilgi extends IOException {
    public OlanaksizBilgi( ) { }
    public OlanaksizBilgi( String hataMesaji ) {
        super(hataMesaji);
    }
}
```

27

AYKIRI DURUMLAR

AYKIRI DURUMLARLA ÇALIŞMAK

- Kod (devam)

```
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;
    }
}
```

28

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 alıştırma (!) 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: ");
```


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: Olanaksız 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 );
  }
}
```

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

31

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

GENEL BİLGİLER

- Jenerik programlama, farklı tür nesneler için yeniden kullanılabilir kod üretmenin yollarından biridir.
  - Java ve C# gibi güncel diller jenerik sınıfları desteklemektedir.
  - Jenerik sınıflar, C++'taki şablon sınıflarına (template class) benzetilebilir.
- Bu bölümün amacı sizleri jenerik sınıfların kullanımına alıştırmaktır.
  - Bu amaca ulaşmak için Java dili ile gelen hazır jenerik sınıflardan bazılarının kullanımı gösterilecektir.
  - Derste kullanılacak örnekler olarak java.util kütüphanesindeki temel veri yapıları seçilmiştir.
  - Bu nedenle bu veri yapıları hakkında özet bilgi de verilecektir.
- Bu bölümün amaçları arasında aşağıdakiler yer almamaktadır:
  - Kendi jenerik sınıflarınızı oluşturmak
  - Veri yapılarının öğretilmesi

32

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

VERİ YAPILARI

- Verinin bilgisayar belleğinde simgelenmesi ve düzenlenmesinin farklı biçimlerine veri yapıları adı verilir.
- Çok genel bakıldığında, verinin herhangi bir sunumu bir veri yapısı olarak algılanabilir.
  - Ör. int ilkeli, Integer sarmalayıcısı, vb.
- Veri yapıları kavramı ile asıl anlatılmak istenen, **bir veri koleksiyonunun organizasyonu için düşünülmüş farklı düzenlerdir**.
- Verinin nasıl organize edildiği bir programın farklı işlerdeki başarımını doğrudan etkiler.
- Kullanılacak algoritmanın ve veri yapılarının doğru veya yanlış seçilmesi, bir programın çalışmasının birkaç saniye veya birkaç saat sürmesi arasındaki farkı belirler!
- Literatürde en sık kullanılan veri yapıları arasında diziler, listeler, tablolar (harita/map), yığınlar, kuyruklar, öbekler (heap), ağaçlar, graflar sayılabilir.

33

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAĞLI LİSTE VERİ YAPISI

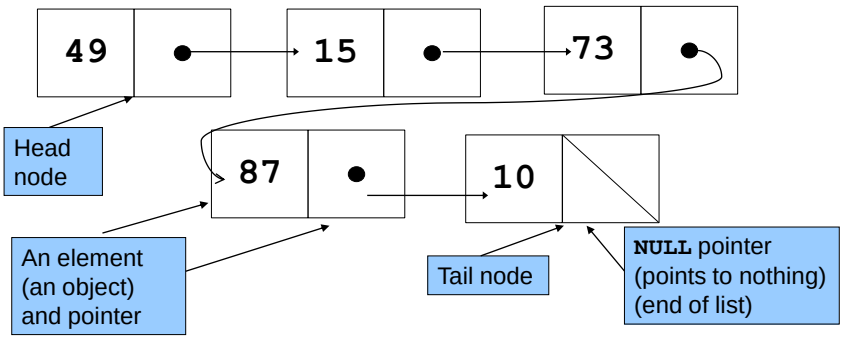
- Düğüm olarak adlandırılan aynı türden nesnelerin doğrusal bir sıra oluşturacak biçimde düzenlendiği veri yapısıdır.
- Her bir düğüm bir sonraki düğüme bağlıdır.
  - Bu bağların kurulması, düzenlenmesi, vb. eylemlerden hazır yapıları kullanan Java programcısı sorumlu değildir.
- Listenin başındaki ve sonundaki düğümlere birer nesne işaretçisi ile erişilir.
- Düğümler arasında dolaşmak için veri yapısından bir dolaşım nesnesi (iterator object) istenir.
  - Veri yapısının sunduğu dolaşım olanaklarını kullanacak şekilde kendi dolaşım kodunuzu yazabilirsiniz.
- Bir listeyi tümüyle dolaşmak for-each döngüsü kullanılarak kolayca kodlanabilir.

34

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAĞLI LİSTE VERİ YAPISI

- Integer nesneleri tutan örnek bir liste:



35

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAĞLI LİSTE VERİ YAPISI

- Bağlı listelerin dizilerden üstünlükleri:
  - Bir listeyi genişletmenin maliyeti çok azdır (çok hızlıdır).
  - Herhangi bir konuma eleman eklemek veya o konumdan eleman çıkarmak çok hızlıdır.
  - Sıralama algoritmaları bağlı listeler üzerinde daha hızlı çalışır.
- Dizilerin bağlı listelerden üstünlükleri:
  - Dizinin herhangi bir n. elemanına erişim çok hızlıyken listelerde ardışıl (en yavaş) erişim söz konusudur.
- Bağlı liste türleri:
  - Tek bağlı liste: Sadece tek yönde dolaşılabilir.
  - Çift bağlı liste: Her iki yönde (ileri ve geri) dolaşılabilir.
- Bir liste aynı zamanda çevrimsel olabilir.
  - Bu durumda son düğüm ilk düğümü gösterir (tespih gibi).

36

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAĞLI LİSTE VERİ YAPISI

- Boyutları çalışma anında ve veri kaybı olmadan genişletilebilir dizi yapısı: java.util.ArrayList sınıfı.
  - Bir bağlı liste (Linked list) veri yapısı gerçeklemesidir.
- Tanımlama ve oluşturma:

```
ArrayList liste = new ArrayList();
```

- Bu tür tanımlamada, dizinin elemanları Object türünden (ortak üst sınıf) nesnelerdir.
  - Yerine geçebilirlik ilkesi sayesinde bu tür kullanım mümkündür.
  - Ancak nesneleri gerçek tipleri üzerinden kullanmak için tip dönüşümü yapmak gereklidir.
  - Bu gerekliliği ortadan kaldırmak için JSE 5.0 ile birlikte 'genel sınıflar' desteklenmeye başlanmıştır.
- Genel sınıf: Herhangi bir türden nesne topluluğu ile birlikte çalışabilecek, genel amaçlı kullanıma yönelik sınıflar.
  - Örnek: Insan örneklerinin saklanacağı bir ArrayList sınıfı oluşturmak:

```
ArrayList<Insan> liste = new ArrayList<Insan>();
```

37

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAĞLI LİSTE VERİ YAPISI

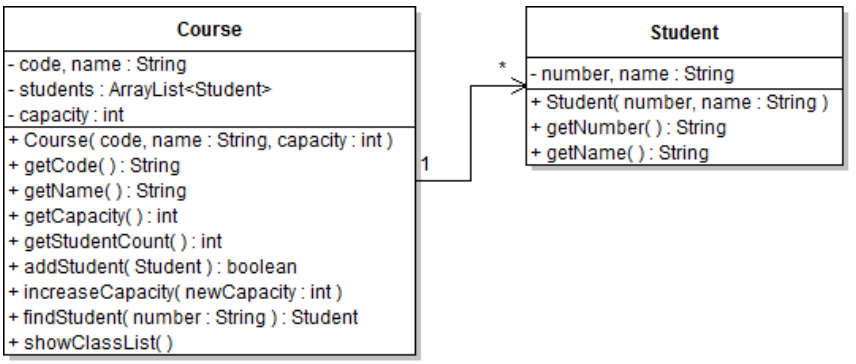
- ArrayList sınıfının temel metotları:
  - add(<T> nesne): Dizinin sonuna bir eleman eklemek için.
  - <T> get(int i): i. sıradaki elemanı döndürür.
    - for-each çevrimi de kullanılabilir.
  - int size( ): Dizideki eleman sayısını döndürür.
  - for-each döngüsü ile kolaylıkla gezinilebilir.
- ArrayList sınıfının diğer metotlarından seçmeler:
  - ensureCapacity(int genişlik): Dizinin genişliğini belirler.
    - Genişlik çalışma anında, veri kaybı olmadan arttırılabilir.
  - trimToSize( ): Kırpma. Dizinin genişliğini içerdiği eleman sayısına eşitler.
  - set(int i, <T> nesne): nesneyi i. sıraya yerleştirir.
  - remove(int i): i. sıradaki nesneyi listeden çıkartır.
- Listenin genişliği i'den küçükse çalışma anı hatası oluşur :
  - IndexOutOfBoundsException, unchecked

38

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAĞLI LİSTE VERİ YAPISI

- Bir çoklu sahiplik örneğini, bu kez de genel sınıflarla gerçekleyelim:



GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

```
package oop02a;
import java.util.*;

public class Course {
    private String code; private String name; private int capacity;
    private ArrayList<Student> students;

    public Course(String code, String name, int capacity) {
        this.code = code; this.name = name; this.capacity = capacity;
        students = new ArrayList<Student>();
    }
    public String getCode() { return code; }
    public String getName() { return name; }
    public int getCapacity() { return capacity; }
    public int getStudentCount() {
        return students.size();
    }
    public boolean addStudent( Student aStudent ) {
        if( getStudentCount() == capacity ||
                        findStudent(aStudent.getNumber()) != null )
                return false;
        students.add(aStudent);
        return true;
    }
```

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

```
    public Student findStudent( String number ) {
        for( Student aStudent : students )
                if( aStudent.getNumber().compareTo(number) == 0 )
                        return aStudent;
        return null;
    }
    public void increaseCapacity( int newCapacity ) {
        if( newCapacity <= capacity )
                return;
        capacity = newCapacity;
    }
    public void showClassList( ) {
        System.out.println("Class List of "+code+" "+name);
        System.out.println("Student#  Name, Surname");
        System.out.println("--------  ------------------------------");
        for( Student aStudent : students )
                System.out.println(aStudent.getNumber()+
                " " + aStudent.getName());
    }
}
```

- Böyle bir örneği bir de diziler kullanarak kodlayarak generic sınıfların ve veri yapılarının avantajlarını kendiniz de görebilirsiniz.
  - Bu örnekte capacity alanına liste veri yapısının gereksinimleri nedeniyle değil, iş mantığının gereksinimleri nedeni ile ihtiyaç duyulmuştur.

41

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

TABLO (EŞLEME/MAP) VERİ YAPISI

- Tablo/eşleme yapısı: java.util.HashMap sınıfı
  - Hashtable veri yapısının bir gerçeklemesi.
  - Verilen ayırt edici alana göre hızlı arama yapar. Böylece listelerde olduğu gibi sıralı arama yapmaya gerek kalmaz.
- Tanımı: java.util.HashMap<K,V>
  - K: Key, V: Value
- Temel metotları:
  - public V get(Object key);
    - Verilen anahtara sahip olan nesneyi (kayıdı/değeri) döndürür.
  - public V put(K key, V value);
    - Verilen nesneyi (kayıdı/değeri), verilen anahtarla tabloya kaydeder.
    - Anahtarın, nesneden elde edilmesi önerilir.
    - Tabloda zaten K anahtarlı değer varsa onu döndürür
      - Eski nesne silinir, yenisi yazılır.
  - public Collection<V> values( );
    - Tabloda saklanan tüm değerlerin bir listesini döndürür.
    - Collection generic sınıfı hakkında bu aşamada üzülmenize gerek yok, for-each içinde kullanabileceğinizi bilmeniz yeter.

42

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

- Aynı örneği bu kez HashMap ile yapalım:

```
package oop02b;
import java.util.*;
public class Course {
    private String code; private String name; private int capacity;
    private HashMap<String,Student> students;

    public Course(String code, String name, int capacity) {
        this.code = code; this.name = name; this.capacity = capacity;
        students = new HashMap<String,Student>();
    }
    public String getCode() { return code; }
    public String getName() { return name; }
    public int getCapacity() { return capacity; }
    public int getStudentCount() {
        return students.size();
    }
    public boolean addStudent( Student aStudent ) {
        if( getStudentCount() == capacity ||
                        findStudent(aStudent.getNumber()) != null )
                return false;
        students.put(aStudent.getNumber(), aStudent);
        return true;
    }
```

43

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

TABLO (EŞLEME/MAP) VERİ YAPISI

```
    public Student findStudent( String number ) {
        return students.get(number);
    }
    public void increaseCapacity( int newCapacity ) {
        if( newCapacity <= capacity )
                return;
        capacity = newCapacity;
    }
    public void showClassList( ) {
        System.out.println("Class List of "+code+" "+name);
        System.out.println("Student#  Name, Surname");
        System.out.println("--------  ------------------------------");
        for( Student aStudent : students.values() )
                System.out.println(aStudent.getNumber()+
                " " + aStudent.getName());
    }
}
```

- Bu örneği bir de diziler kullanarak yazabileceğiniz kodla ve önceki liste veri yapısını kullanarak yazılan kodla karşılaştırıp avantaj/dezavantajları kendiniz de görebilirsiniz.

44

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

SINAMA

- Örnekleri yaptık ama sınamadık, belki bir bug vardır?

```
public class USIS {
    public static void main(String[] args) {
        Course oop = new Course("0112562", "Obj. Or. Prog.", 3);
        Student yasar = new Student("09011034","Yaşar Nuri Öztürk");
        if( !oop.addStudent(yasar) )
                System.out.println("Problem #1");
        if( oop.addStudent(yasar) )
                System.out.println("Problem #2");
        Student yunus = new Student("09011045","Yunus Emre Selçuk");
        oop.addStudent(yunus);
        Student fatih = new Student("09011046","Fatih Çıtlak");
        oop.addStudent(fatih);
        Student cemalnur = new Student("09011047","Cemalnur Sargut");
        if( oop.addStudent(cemalnur) )
                System.out.println("Problem #3");
        if( oop.findStudent("09011046") != fatih )
                System.out.println("Problem #4");
        System.out.println("End of test");
    }
}
```

Niye paket adı yok? Yeni USIS sınıfını belgelendirmeye (UML sınıf şemasına) nasıl ekleriz?

45

GENERIC SINIFLARLA TEMEL VERİ YAPILARINA GİRİŞ

BAZI VERİ YAPILARININ JAVA GERÇEKLEMELERİ HAKKINDA ÖZET BİLGİ

- java.util.LinkedList<E> implements List<E>
  - Elemanların eklenme sırası önemli.
  - Araya eleman ekleme ile aradan eleman çıkarma hızlı.
  - Rastgele erişim yavaş.
  - Çift yönlü listedir, ListIterator alıp geri gidiş yapılabilir (bu derste işlenilmeyecek).
- java.util.ArrayList<E> implements List<E>
  - Araya eleman ekleme ile aradan eleman çıkarma yavaş, rastgele erişim hızlı.
- java.util.Vector<E> implements List<E>
  - ArrayList gibi ama synchronized; çok görevcikli kullanım için uygun,
  - Ancak tek görevcikli kullanımda başarımı düşer.
- java.util.HashMap<K,V> implements Map<K,V>
  - Anahtara göre hızlı eleman aramaya yarar.
- java.util.Hashtable<K,V> implements Map<K,V>
  - HashMap gibi ama synchronized; çok görevcikli kullanım için uygun,
  - Ancak tek görevcikli kullanımda başarımı düşer.
  - Dikkat: Hashtable'ın t'si küçük.

46

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

TİP DÖNÜŞÜMÜ (TYPECASTING)

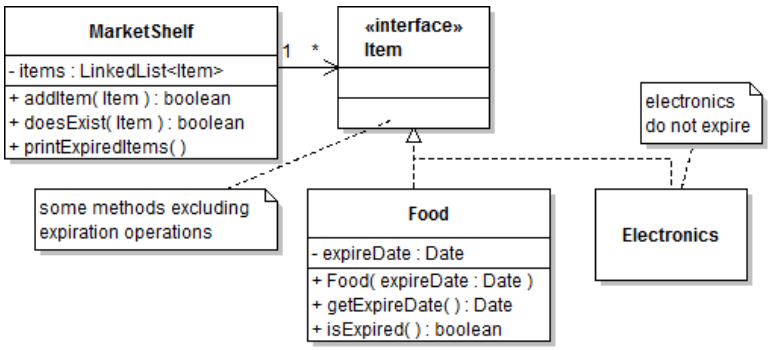
47

TİP DÖNÜŞÜMÜ (TYPECASTING)

- Kalıtım ile ilgili şu kuralı hatırlayınız:
  - Alt sınıftan olan bir nesne her zaman üst sınıftan bir nesne yerine kullanılabilir.
- Bu, güvenli ve dolayısıyla otomatik olarak yapılabilecek bir dönüşümdür.
- Eğer şartlar elveriyorsa, bir tipten diğer bir tipe geçiş yapılabilir.
  - Arayüz tipinden nesne tipine
  - Üst sınıftan alt sınıfa
- Ancak bu tür dönüşümler güvensiz olarak kabul edilir.
  - Bu dönüşümlerden önce bir sınama yapılmalı ve dönüşüm için özel bir sözdizimi kullanılmalıdır.
  - Sınama yapılmazsa denetlenmeyen türden bir aykırı durum ortaya çıkar.
- Söz konusu dönüşüm işlemine tip dönüşümü adı verilir.

TİP DÖNÜŞÜMÜ (TYPECASTING)

- Örnek:



TİP DÖNÜŞÜMÜ (TYPECASTING)

- MarketShelf sınıfını kodlama:

```
import java.util.*;
public class MarketShelf {
    private LinkedList<Item> items;
    public MarketShelf() {
        items = new LinkedList<Item>();
    }
    public boolean doesExist( Item anItem ) {
        for( Item item : items )
        if( item == anItem )
                return true;
        return false;
    }
    public boolean addItem( Item anItem ) {
        if( doesExist(anItem) )
                return false;
        items.add(anItem);
        return true;
    }
```

TİP DÖNÜŞÜMÜ (TYPECASTING)

- MarketShelf sınıfını kodlama:

```
public void checkForExpiration( ) {
    boolean hasExpiredItem = false;
    System.out.println("Expired item(s): ");
    for( Item item : items ) {
       if( item instanceof Food )
           if( ((Food)item).isExpired() ) {
               hasExpiredItem = true;
               System.out.println(item);
           }
    }
    if( hasExpiredItem == false )
           System.out.println("All items are fresh!");
}
```

- Sınama ve tip dönüşümü

TİP DÖNÜŞÜMÜ (TYPECASTING)

- MarketShelf sınıfını kodlama:

```
public static void main( String[] args ) {
    MarketShelf fridge = new MarketShelf();
    Calendar cal = Calendar.getInstance();
    cal.add(Calendar.DAY_OF_MONTH,1);
    Date future = cal.getTime();
    cal.set(Calendar.YEAR, 2010);
    cal.set(Calendar.MONTH, 0); //0: January
    cal.set(Calendar.DATE, 12);
    Date past = cal.getTime();
    fridge.addItem( new Food(past) );
    fridge.addItem( new Food(future) );
    fridge.addItem( new Electronics() );
    fridge.checkForExpiration();
}
}
```

- Kaşla göz arasında **java.util.Date ve Calendar sınıflarının kullanımını** da anlattık!
  - Sonraki yansıda da devam edecek

TİP DÖNÜŞÜMÜ (TYPECASTING)

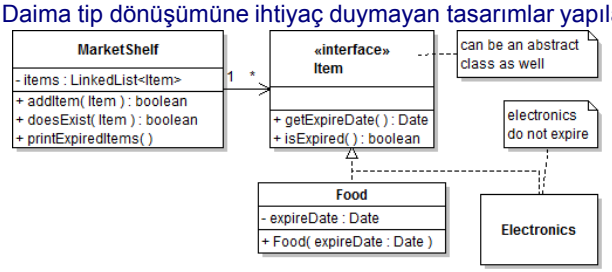
- Food sınıfını kodlama:

```
package oop03b;
import java.util.Date;
public class Food implements Item {
    private Date expireDate;

    public Food(Date expireDate) {
        this.expireDate = expireDate;
    }
    public Date getExpireDate() { return expireDate; }
    public boolean isExpired( ) {
        Date today = new Date();
        if( expireDate.before(today) )
                return true;
        return false;
    }
    public String toString() {
        return "A food expiring at " + expireDate;
    }
}
```

- **java.util.Date sınıfının kullanımı**

TİP DÖNÜŞÜMÜ (TYPECASTING)

- Tip dönüşümü ile ilgili eleştiriler:
  - Tip dönüşümüne bağımlı kod, genelde iyi bir tasarım sayılmaz. Bu nedenle sadece zorunlu olduğunuzda kullanınız.
  - Java'ya jenerik sınıf desteği kazandırılmadan önce tüm veri yapıları Object türünden üyelerle çalışıyordu ve biz de tip dönüşümü yapmak zorunda kalıyorduk.
  - Artık tip dönüşümüne sadece serileştirilmiş nesneleri toparlarken ihtiyaç duyuluyor (sonraki konu).
- Daima tip dönüşümüne ihtiyaç duymayan tasarımlar yapılabilir. Örneğin:



- Diğer durumlarda soyut sınıflar ve çokbiçimliliğe dayalı daha güzel tasarımlar yapılabilir.
  - Bu durumda alt sınıflara bağımlılık kurmak önerilmez.

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

DOSYALARLA ÇALIŞMAK

55

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.

56

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 ayıracı:
      - 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.

57

DOSYALARLA ÇALIŞMAK

DOSYA YÖNETİMİ

- java.io.File sınıfının bazı metotları:
  - boolean exists( ); dosya/dizinin fiziksel mevcudiyet durumunu döndürür
  - boolean isFile( ); nesnenin bir dosya olup olmadığını döndürür
  - File getParentFile( ); bu dosya/dizinin üst dizinini döndürür
  - String getCanonicalPath( ) throws IOException; nesnenin kendi adı dahil olmak üzere dosya/dizinin tam yolunu döndürür
  - boolean canRead( ); nesneye okuma hakkı var mı?
  - boolean canWrite( ); nesneye yazma hakkı var mı?
  - boolean createNewFile( ); Dosyayı fiziksel olarak da oluşturur
  - boolean mkdir( ); Dizini fiziksel olarak da oluşturur
  - boolean mkdirs( ); Dizini gerekli üst dizinlerle birlikte fiziksel olarak da oluşturur
  - boolean renameTo(File newName); nesneyi yeniden adlandırır.
  - boolean delete( ); nesneyi siler.
- Notlar:
  - Tüm bu metotları ezberlemek zorunda değilsiniz.
  - Fiziksel: Dosya/dizinin diskte gerçekten yer alması

58

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.

59

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Serileştirme – Çıktı işlemleri:
  - Nesneleri bir dosyaya yazma işlemleri
  - Serileştirilecek nesneler, java.io.Serializable arayüzünü gerçeklemelidir.
  - 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.

60

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Örnek kayıt: Arkadas sınıfı

```
package not04a;
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;
    }
}
```

61

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.

62

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.

63

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Kayıtları dosyaya yazan program:

```
package not04a;
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() );
        }
```

64

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.");

        }
        giris.close();
        catch( IOException e ) {
                System.out.println("Dosyaya kayıt işlemi sırasında"+
                        " bir hata oluştu.");
                e.printStackTrace();
        }
    }
}
```


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.

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);
        }
    }
```

67

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();
    }
  }

}
```

68

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ – DÜZELTİLMEMİŞ HATALAR

- Bir akının sonuna gelip gelmediğimizi sınamanın doğru çalışan bir yolu halen bulunmamakta. Bu nedenle aşağıdaki gibi bir kod **yazamıyoruz**:

```
try {
    ObjectInputStream okuyucu = new ObjectInputStream(
                    new FileInputStream( dosyaAdi ) );
    Arkadas arkadas = (Arkadas) okuyucu.readObject();
    while okuyucu.hasNext() {
            System.out.println(arkadas);
            arkadas = (Arkadas) okuyucu.readObject();
    }
    okuyucu.close();
}
```

Çalışmaz!!!

69

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ – DÜZELTİLMEMİŞ HATALAR

- `int ObjectInputStream.available( )`, metodu var ama hatalı
  - http://www.coderanch.com/t/378141/java/java/EOF-ObjectInputStream
- Üstelik, readObject() metodu dosya sonuna gelinince null döndüremiyor.
  - http://stackoverflow.com/questions/2626163/java-fileinputstream-objectinputstream-reaches-end-of-file-eof
- Bir aykırı durum olmasına izin verip catch bloğunda döngüyü sonlandırma yoluna gidilebilir ama tercih edilmez.
  - Aykırı durum işleme, denetim akışını düzenleme amacıyla icat edilmemiştir
- Daha iyi bir çözüm, saklanması gereken tüm nesneleri bir veri yapısında tutmak ve o veri yapısıyla serileştirme işlemlerini yapmaktır.
  - Sonraki yansıda gösterilecektir.
  - Veri yapısındaki nesneler `java.io.Serializable` arayüzünü gerçeklemeyi sürdürmelidirler.
  - A→B ilişkisi varsa her iki sınıf da `java.io.Serializable` arayüzünü gerçeklemelidir.

70

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Veri yapıları ile serileştirme (diske yazma) işlemleri:

```
package oop04b;
import java.util.*;
import java.io.*;
@SuppressWarnings("resource")
public class ArkadasOlustur {
  public static void main(String[] args) {
     LinkedList<Arkadas> arkadaslar = new LinkedList<Arkadas>();
     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? ");
     int arkadasSayisi = giris.nextInt( );
     giris.nextLine( );
     for( int i = 0; i < arkadasSayisi; i++ ) {
          System.out.print((i+1)+". arkadaşınızın ismi nedir? ");
          Arkadas arkadas = new Arkadas( giris.nextLine() );
          System.out.print("Bu arkadaşınızın telefonu nedir? ");
          arkadas.setTelefon( giris.nextLine() );
          System.out.print("Bu arkadaşınızın e-posta adresi nedir? ");
          arkadas.setEPosta( giris.nextLine() );
          arkadaslar.add(arkadas);
     }
```

71

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Veri yapıları ile serileştirme (diske yazma) işlemleri (devam):

```
     try {
          String dosyaAdi = "arkadaslarAlt.dat";
          ObjectOutputStream yazici = new ObjectOutputStream(
                    new FileOutputStream( dosyaAdi )  );
          yazici.writeObject( arkadaslar );
          yazici.close();
          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();
     }
  }
}
```

72

DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Veri yapıları ile serileştirme (diskten okuma) işlemleri:

```
package oop04b;
import java.io.*;
import java.util.*;
public class ArkadasGoster {
  public static void main( String[] args ) {
    String dosyaAdi = "arkadaslarAlt.dat";
    try {
        ObjectInputStream okuyucu = new ObjectInputStream(
                new FileInputStream( dosyaAdi ) );
        @SuppressWarnings("unchecked")
        LinkedList<Arkadas> arkadaslar =
                (LinkedList<Arkadas>)okuyucu.readObject();
        for( Arkadas arkadas : arkadaslar ) {
                System.out.println(arkadas);
        }
        okuyucu.close();
    }
```


DOSYALARLA ÇALIŞMAK

AKILAR (STREAM) İLE DOSYA İŞLEMLERİ

- Veri yapıları ile serileştirme (diskten okuma) işlemleri (devam):

```
    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();
    }
  }
}
```

- Metin dosyaları ile çalışma konusu için Core Java Vol.II (8th ed) veya bir başka Java kitabına başvurabilirsiniz.

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

ENUM SINIFLARI

ENUM SINIFLARI

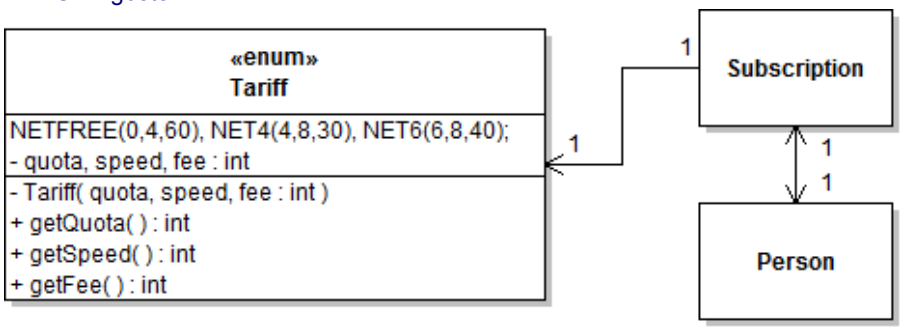
- İlkel sıralama tanımı aslında bir sınıf tanımıdır.
- Sıralamanın her elemanı o sınıfın bir örneğidir.
- Sıralama sınıfının tanımlananların dışında başka örneği de olamaz.
- Tanımlama gövdenin ilk komutu olmalıdır.
- Bir sıralama tipine herhangi bir sınıfa yapıldığı gibi üye alanlar ve metotlar eklenebilir.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

Tanım ilk satırda yer almalıdır.

```
package oop05a;
public enum Tariff {
   NETFREE(0,4,60), NET4(4,8,30), NET6(6,8,40);
   private int quota, speed, fee;
   private Tariff( int quota, int speed, int fee ) {
       this.quota = quota; this.speed = speed; this.fee = fee;
   }
   public int getQuota() { return quota; }
   public int getSpeed() { return speed; }
   public int getFee() { return fee; }
}
```

ENUM SINIFLARI

- UML gösterimi:



77

ENUM SINIFLARI

- Enum nesnesini tanımlamak ve kullanmak:

```
public class Test {

   public static void main(String[] args) {
      Tariff tariff4 = Tariff.NET4;
      Person yunus = new Person("Yunus Emre");
      yunus.subscribeTo(tariff4);
      Person berkin = new Person("Berkin Gülay");
      berkin.subscribeTo(Tariff.NETFREE);
      System.out.println(yunus);
      System.out.println(berkin);
   }

}
```

78

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

İÇ SINIFLAR

79

İÇ SINIFLAR

- Bir sınıfın gövdesi içerisinde başka bir sınıf tanımlamak mümkündür.
  - İç sınıf, bir normal (dış) sınıfın içinde kodlanan sınıftır.
- İç sınıfın yapabilecekleri:
  - Dış sınıfın private dahil tüm üyelerine erişebilir.
  - Eğer kendisi private olarak tanımlanırsa aynı paketteki sınıflardan bile gizlenebilir.
  - GUI programlamada anonim (isimsiz) iç sınıflar şeklinde sıkça kullanılırlar.
- İç sınıfın yapamayacakları:
  - Bir iç sınıfta static metot tanımlanamaz.
- Örnek: Kişi ve çalışan sınıfları

80

İÇ SINIFLAR

```
package oop05b;
public class Person {
    private String name;
    public Person( String name ) { this.name = name; }
    @SuppressWarnings("unused")
    private class Employee { //begin inner class
        private int salary;
        public Employee( int salary ) { this.salary = salary; }
        public int getSalary() { return salary; }
        public void setSalary(int salary) { this.salary = salary; }
        public String toString( ) { return name + " " + salary; }
    } //end inner class
    public static void main( String[] args ) {
        Employee[] staff = new Employee[3];
        Person kisi;
        kisi = new Person("Osman Pamukoğlu");
        staff[0] = kisi.new Employee( 10000 );
        kisi = new Person("Nihat Genç");
        staff[1] = kisi.new Employee( 7500 );
        kisi = new Person("Barış Müstecaplıoğlu");
        staff[2] = kisi.new Employee( 6000 );
        for( Employee eleman: staff )
               System.out.println( eleman );
    }
}
```

81

İÇ SINIFLAR

- Önceki örnekte gördüklerimiz:
  - Bir iç sınıf tanımlamak
  - Dış nesneye iç nesneden ulaşmak
  - İç nesneye dış nesneden ulaşmak
- Önceki örnekte iç sınıf private olarak tanımlanmıştı
  - Böylece iç sınıfa aynı pakettekiler dahil olmak üzere hiçbir başka sınıftan ulaşılamayacak.
  - Bu da demektir ki Person.main metodu başka sınıfa taşınamaz.
- Sonraki örnek bir public iç sınıfa diğer sınıflardan nasıl erişilebileceğini gösterecek:

82

İÇ SINIFLAR

```
package oop05c;

public class Person {
    private String name;
    public Person( String name ) { this.name = name; }

    public class Employee {
        private int salary;
        public Employee( int salary ) { this.salary = salary; }
        public int getSalary() { return salary; }
        public void setSalary(int salary) { this.salary = salary; }
        public String toString( ) { return name + " " + salary; }
    }
}
```


İÇ SINIFLAR

```
package oop05c;

//this import is absolutely necessary
import oop04c.Person.Employee;

public class TestInnerClassDirectly {
    public static void main( String[] args ) {
        Employee[] staff = new Employee[3];
        Person kisi;
        kisi = new Person("Osman Pamukoğlu");
        staff[0] = kisi.new Employee( 10000 );
        kisi = new Person("Nihat Genç");
        staff[1] = kisi.new Employee( 7500 );
        kisi = new Person("Barış Müstecaplıoğlu");
        staff[2] = kisi.new Employee( 6000 );
        for( Employee eleman: staff )
                System.out.println( eleman );
    }
}
```

- Not: import komutu yerine her gerektiğinde Person.Employee da yazabilirsiniz ama import daha rahat.

İÇ SINIFLAR

- Bu örnek private iç sınıfı herhangi bir farklı sınıfın kullanımına nasıl açabileceğimizi gösteriyor
  - Bu işin yolu dış sınıfa bu işi yapacak bir public metot eklemektir

```
package oop05d;
public class Person {
    private String name;
    private Employee employee;
    public Person(String name) { this.name = name; }
    public void enlist( int salary ) {
        employee = new Employee( salary ); }
    public String toString( ) {
        String mesaj = name;
        if( employee != null )
                mesaj += " " + employee.getSalary( );
        return mesaj;
    }
    @SuppressWarnings("unused")
    private class Employee {
        private int salary;
        public Employee( int salary ) { this.salary = salary; }
        public int getSalary() { return salary; }
        public void setSalary(int salary) { this.salary = salary; }
    }
}
```

85

İÇ SINIFLAR

```
package oop05d;
public class TestInnerClassViaOuterClass {
    public static void main(String[] args) {
        Person[] staff = new Person[3];
        staff[0] = new Person( "Polat Alemdar" );
        staff[0].enlist( 10000 );
        staff[1] = new Person( "Memati Baş" );
        staff[1].enlist( 7000 );
        staff[2] = new Person( "Abdülhey Çoban" );
        staff[2].enlist( 5000 );
        for( Person insan: staff )
                System.out.println( insan );
    }
}
```

86

011 2562 NESNEYE DAYALI PROGRAMLAMA DERS NOTLARI
Yrd. Doç. Dr. Yunus Emre SELÇUK

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

87

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

İZLEK (TREAD)

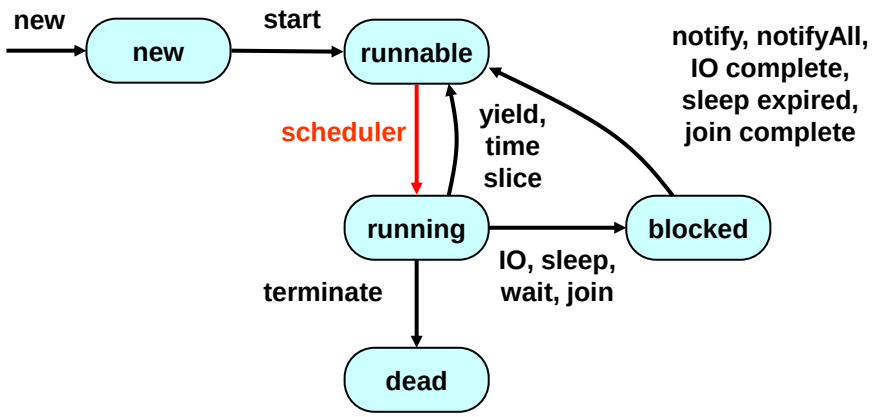
- Bilişim sözlüğü:
  - Bir süreçte tek bir denetim akışı (izlek teriminin kökeni)
  - Bir bilgi işleme sürecinde gerçekleştirilebilecek en küçük işlem birimi (görevcik teriminin kökeni)
  - Programda ana denetim akışına ek olarak aynı anda birden fazla başka görevler daha çalıştırılıyorsa, bu görevlerin her birine izlek denir.
- Çoklu izlekle çalışmanın yararları ve önemi:
  - Bir iş paralel olarak çalışabilecek birden fazla göreve ayrılabiliyorsa, işlem daha kısa sürede tamamlanacaktır.
  - Günümüzde çok çekirdekli işlemciler iyice yaygınlaşmıştır.
  - Tek çekirdekli işlemcilerde bile “Hyperthreading” gibi teknolojiler kullanılarak işlemcinin birden fazla görevle ilgilenmesi sağlanmaktadır.
  - Böylece her bir izlek ayrı çekirdekte/işlem biriminde çalıştırılabilir.

88

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

İZLEK TEMELLERİ

- Bir izleğin içinde bulunabileceği durumlar ve durumlar arası olası geçişler:



ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

İZLEK TEMELLERİ

- İzleğin beklemesi:
  - Bir izlek belli bir süre bir işlem yapmayacaksa, o izleği bekletilebilir.
  - Bekleme için boş yere döngü çalıştırmak gibi işlemler yapılmamalıdır.
    - 100,000 kez dönecek boş bir for döngüsü kurmak gibi boş işlemler (Busy waiting) yapılmamalıdır.
  - Bunun yerine izlek, belli bir süre uykuya alınabilir.
    - Blocked durumu
    - Bu süre dolduktan sonra, sistem kaynaklarının elverdiği kadar kısa bir süre içerisinde, izlek tekrar çalışır.
      - Yani Runnable durumuna geçer.
    - Uykuya alınan izlek sistem kaynaklarını boşa harcamaz.
  - İzleğin uykudan uyanamaması **denetlenen** bir aykırı durumdur.
    - java.lang.InterruptedException

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

İZLEK TEMELLERİ

- Bir görevi ayrı bir izlekte çalıştırmak için yapılması gerekenler:
  1. İzlekte yapılacak iş, Runnable arayüzünü gerçekleyen bir sınıfın public void run( ) metoduna kodlanır.
  2. Bu sınıftan bir nesne türet
  3. Bu nesneyi bir Thread nesnesinin kurucusuna parametre olarak ver
  4. Thread nesnesinin public void start( ) metodunu çağır (run metodu doğrudan çağrılmamalıdır).

- Örnek uygulama:
  - Takımlar ve bunlar için tezahurat yapan amigolar olsun.
  - Bir amigo iki tezahurat arasında 0-1000ms. arasında bir süre beklesin.

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

```
package oop06a;
import java.util.Random;

public class SoccerFan implements Runnable {
    public final static int STEPS = 10;
    public final static int DELAY = 1000;
    private String teamName, shoutPhrase;

    public SoccerFan( String teamName, String shoutPhrase ) {
        this.teamName = teamName;
        this.shoutPhrase = shoutPhrase;
    }

    public void run() {
        Random generator = new Random();
        try {
            for( int i = 0; i < STEPS; i++ ) {
                System.out.println( teamName + " " + shoutPhrase );
                Thread.sleep( generator.nextInt(DELAY) );
            }
        }
        catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

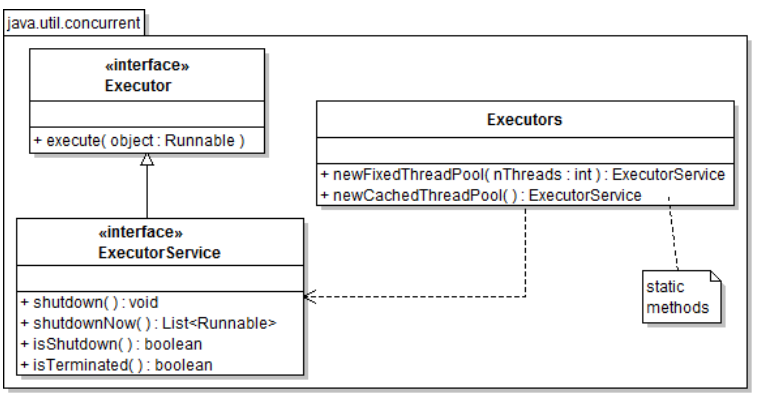
```
package oop06a;

public class Match {
    public static void main(String[] args) {
        Thread aThread;
        aThread = new Thread( new SoccerFan("G.S.", "Rulez!") );
        aThread.start( );
        aThread = new Thread( new SoccerFan("G.S.", "is the champ!") );
        aThread.start( );
        aThread = new Thread( new SoccerFan("F.B.", "is no.1!") );
        aThread.start( );
        aThread = new Thread( new SoccerFan("F.B.", "is the best!") );
        aThread.start( );
    }

}
```

93

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

İZLEK HAVUZLARI

- Az sayıda basit görevin hepsini ayrı birer izlekte çalıştırmakta sakınca yoktur.
- Ancak bir işlemcideki fiziksel olarak ayrı çalışan çekirdeklerin sayısı kısıtlıdır. Bu yüzden çalıştırılan her yeni izlek sisteme ayrı bir ek yük getirir.
- Sonuçta çok sayıda izlek çalıştıracaksanız bir izlek havuzu kullanınız.
- Java'da izlek havuzu gerçeklemesi şu şekilde yapılmıştır:



94

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

- java.util.concurrent.Executor:
  - public void execute(Runnable object): Verilen görevi çalıştırır.
- java.util.concurrent.ExecutorService:
  - public void shutdown( ) :
    - Çalıştırıcıyı kapatır. Havuza yeni görev kabul edilmez ancak mevcut görevlerin bitmesi beklenir.
  - public List<Runnable> shutdownNow( )
    - Havuzdaki görevleri anında sonlandırır ve havuzu kapatır. Henüz bitmemiş görevleri içeren bir liste geriye döndürür.
  - public boolean isShutdown( ):
    - Havuzun kapatılıp kapatılmadığını döndürür.
  - public boolean isTerminated( ):
    - Havuzdaki görevlerin tümünün bitip bitmediğini geri döndürür.
    - Görevlerin bitmesini beklemek amacıyla bir main metodu içerisinde kullanılabilir.
- java.util.concurrent.Executors:
  - public static ExecutorService newFixedThreadPool(nThreads : int)
    - Sabit sayıda izlek kullanan bir havuz oluşturur.
  - public static ExecutorService newCachedThreadPool( )
    - Ucu açık bir havuz oluşturur. Sonlanan görevciklerin izlekleri yeniden kullanılır.

95

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

İZLEK HAVUZLARI

- Önceki örneğimizi havuz kullanacak şekilde değiştirelim:
  - SoccerFan kodu değişmeyecek.
  - Farklı boyutlarda sabit havuz kullanarak deneylerde bulunmayı size bırakıyoruz.

```
package oop06a;
import java.util.concurrent.*;
public class MatchWithPool {
    public static void main(String[] args) {
        ExecutorService pool = Executors.newCachedThreadPool( );
        pool.execute( new SoccerFan("G.S.", "Rulez!") );
        pool.execute( new SoccerFan("G.S.", "is the champ!") );
        pool.execute( new SoccerFan("F.B.", "is no.1!") );
        pool.execute( new SoccerFan("F.B.", "is the best!") );
        pool.shutdown( );
    }
}
```

96

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

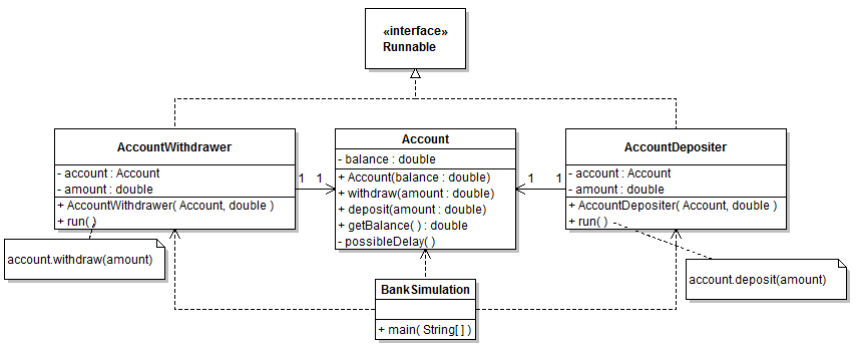
YARIŞ DURUMU

- Çok izlekli gerçek uygulamalarda iki veya daha fazla görevciğin aynı veriye ulaşması gerekecektir.
- Peki ya iki izlek aynı anda aynı nesneye ulaşmaya çalışıp bu nesnenin durumunu değiştirmeye çalışırsa ne olacak?
  - Bu durum potansiyel veri tutarsızlıklarına yol açacaktır.
  - Bu duruma yarış durumu adı verilir.

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

GÖREVCİKLERİN EŞGÜDÜMLÜ KULLANILMASI

- Yarış durumlarını engellemek için görevcikler eşgüdümlü çalıştırılmalıdır.
  - Aşağıdaki örneği ele alalım:



- Sınıf şeması yeterince bilgi veriyor ancak yine de kodu bir yazıp çalıştıralım:

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

```
package oop06b;
public class Account {
    private double balance;
    public Account(double balance) { this.balance = balance; }
    public double getBalance() { return balance; }
    public void withdraw( double amt ){
        double curBal = getBalance();
        possibleDelay( );
        balance = curBal - amt;
    }
    public void deposit( double amt ){
        double curBal = getBalance();
        possibleDelay( );
        balance = curBal + amt;
    }
    private void possibleDelay( ) {
        try { Thread.sleep(5); }
        catch (InterruptedException e) { e.printStackTrace(); }
    }
}
```

99

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

```
package oop06b;
public class AccountDepositer implements Runnable {
    private Account account;
    private double amount;
    public AccountDepositer(Account account, double amount) {
        this.account = account; this.amount = amount;
    }
    public void run() {
        account.deposit(amount);
    }
}

package oop06b;
public class AccountWithdrawer implements Runnable {
    private Account account;
    private double amount;
    public AccountWithdrawer(Account account, double amount) {
        this.account = account; this.amount = amount;
    }
    public void run() {
        account.withdraw(amount);
    }
}
```

100

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

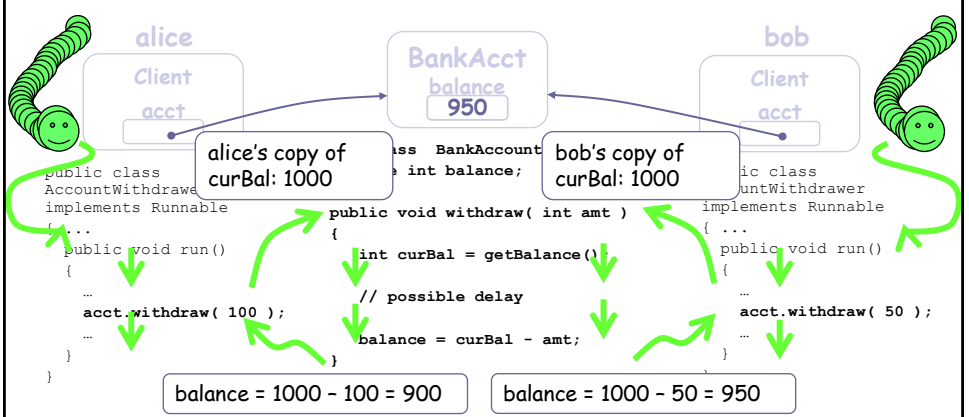
```
package oop06b;
import java.util.concurrent.*;
public class BankSimulation {
    public static void main(String[] args) {
        Account anAccount = new Account(0);
        System.out.println("Before: "+anAccount.getBalance());
        ExecutorService executor = Executors.newCachedThreadPool( );
        for( int i = 0; i < 100; i++ ) {
            AccountDepositer task=new AccountDepositer(anAccount,1);
            executor.execute(task);
        }
        for( int i = 0; i < 50; i++ ) {
            AccountWithdrawer task=new AccountWithdrawer(anAccount,1);
            executor.execute(task);
        }
        executor.shutdown();
        while( !executor.isTerminated() );
                System.out.println("After: "+anAccount.getBalance());
    }
}
```

- Sonucun nasıl olmasını bekliyordunuz? Nasıl bir sonuçla karşılaştınız?

101

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

- Aşağıdaki gibi bir durumu ele alalım:
  - Her izlek kendi yerel değişkenlerine ve parametrelerine sahiptir ancak üye alanlar izlekler arasında ortaklaşa kullanılır.
  - Bu zorunluluk yarış durumunun temel kaynağıdır.



102

ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

- Peki, bir yarış durumu nasıl engellenebilir?
  - Yarışa yol açabilecek metotları belirleriz ve bu metotları synchronized anahtar kelimesi ile işaretleriz.
  - Bir senkronize metoda aynı anda sadece bir izlek erişebilir, diğerleri otomatik olarak bekler.

```
package oop06c;
public class Account {
    private double balance;
    public Account(double balance) { this.balance = balance; }
    public synchronized void withdraw( double amt )  {
        double curBal = getBalance();
        possibleDelay( );
        balance = curBal - amt;
    }
    public synchronized void deposit( double amt )   {
        double curBal = getBalance();
        possibleDelay( );
        balance = curBal + amt;
    }
    public double getBalance() { return balance; }
    public void possibleDelay( ) { /*same as the previous one */ }
}
```

103

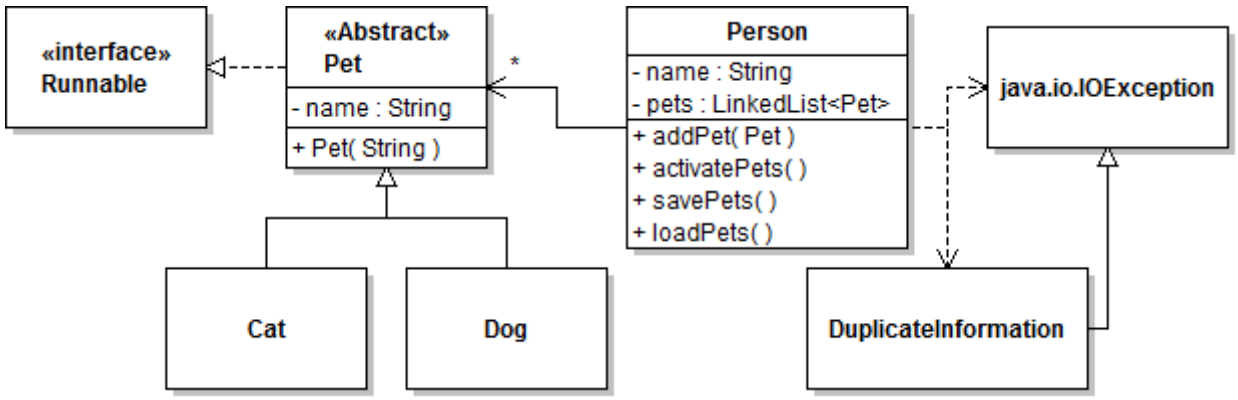
ÇOK İZLEKLİ PROGRAMLAMAYA GİRİŞ

- Diğer sınıfların kodlarını değiştirmeye gerek yoktur.
- Çıktı:

```
Before: 0.0
After: 50.0
```

104

0112562 OBJECT ORIENTED PROG., Section 1, CLASSROOM EXERCISE



Soruları yukarıdaki UML sınıf şemasına göre yanıtlayınız. Soruları yanıtlarken UML şemasında açıkça gösterilmemiş olası gizli bilgileri göz önünde bulundurarak kod yazmanız gerekebilir.

Soru 1: Pet sınıfının kaynak kodunu yazınız.

Soru 2: Cat veya Dog sınıfının kaynak kodunu yazınız. Bir evcil hayvan bir izleğe verildiğinde ekrana şunun gibi bir yazı yazdırır: Meaow! My name is Garfield.

Soru 3: DuplicateInformation sınıfının kaynak kodunu yazınız.

Soru 4: Person sınıfının kaynak kodunu yazınız. Bu sınıfın metotları hakkındaki ayrıntılar şunlardır:

- addPet(Pet): Bu metot kişinin evcil hayvanları arasına yeni birtanesini ekler. Aynı hayvan iki kez eklenmemelidir ve aynı kişinin aynı ada sahip iki farklı hayvanı olmamalıdır.
- activatePets( ): Bu metot hayvanları bir izlek havuzu içerisinde çalıştırır.
- savePets( ): Bu metot hayvanları bir dosyaya kaydeder. Eğer hayvanlar bir izlekte çalışıyorlarsa önce izleklerin sona ermesi beklenmelidir.
- loadPets( ): Bu metot hayvanları dosyadan geri yükler. Eğer hayvanlar bir izlekte çalışıyorlarsa önce izleklerin sona ermesi beklenmelidir.

Soru 5: Şimdiye kadar yazdığınız kodu kullanan bir main metodu yazınız.

```
public abstract class Pet implements Runnable, java.io.Serializable {
	private static final long serialVersionUID = 1L;
	private String name;
	public Pet(String name) { this.name = name;  }
	public String getName() { return name; }
}
public class Cat extends Pet {
	private static final long serialVersionUID = 1L;
	public Cat(String name) { super(name); }
	public void run() {
		System.out.println("(" + getName() + "): Meaow!");
	}
}
import java.io.*;
public class DuplicateInformation extends IOException {
	public DuplicateInformation( String msg ) {
		super(msg);
	}
}
```

```
import java.util.*;
import java.util.concurrent.*;
import java.io.*;
public class Person {
	private String name;
	private LinkedList<Pet> pets;
	private ExecutorService pool;
	public Person(String name) {
		this.name = name;
		pets = new LinkedList<Pet>();
		pool = Executors.newCachedThreadPool( );
	}
	public Pet searchPet( String petName ) {
		for( Pet pet : pets )
			if( pet.getName().equalsIgnoreCase(petName) )
				return pet;
		return null;
	}
	public boolean searchPet( Pet aPet ) {
		for( Pet pet : pets )
			if( pet == aPet )
				return true;
		return false;
	}
	public void addPet( Pet aPet ) throws DuplicateInformation {
		if( searchPet(aPet) == true )
			throw new DuplicateInformation("Pet already exists!");
		if( searchPet(aPet.getName() ) != null )
			throw new DuplicateInformation("Pet with the name " +
					aPet.getName() + "already exists!");
		pets.add(aPet); //no else is necessary. See Main.
	}
	public void activatePets( ) {
		for( Pet pet : pets ) {
			pool.execute(pet);
		}
	}
	public String getName() { return name; }

	public void savePets( ) {
		while( !pool.isTerminated() );
		try {
			ObjectOutputStream stream = new ObjectOutputStream(
					new FileOutputStream(name+"sPets.dat") );
			stream.writeObject(pets);
			stream.close();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
	}
	public void loadPets( ) {
		while( !pool.isTerminated() );
		try {
			ObjectInputStream stream = new ObjectInputStream(
					new FileInputStream(name+"sPets.dat") );
			pets = (LinkedList<Pet>) stream.readObject( );
			stream.close();
		}
		catch (IOException e) {
			e.printStackTrace();
		}
		catch (ClassNotFoundException e) {
			e.printStackTrace();
		}
	}
}
```

```java
public class Main {

	public static void main(String[] args) {
		Person jon = new Person("Jon Arbuckle");
		Cat garfield = new Cat("Garfield");
		try {
			jon.addPet(garfield);
			jon.addPet(garfield);
		}
		catch (DuplicateInformation e) {
			e.printStackTrace();
		}
		System.out.println("What happens after exception?");
		jon.activatePets();

	}

}
```