BMÜ-112 ALGORİTMA VE PROGRAMLAMA-II
SOYUT SINIFLAR VE ARAYÜZLER

Doç.Dr. İlhan AYDIN

Soyut sınıflar
(Abstract Classes)

Bazı durumlarda, yapılacak işlere uyan somut bir üst-sınıf ve ona ait somut metotlar tanımlamak mümkün olmayabilir. Böyle durumlarda, soyut bir üst-sınıf ve ona ait soyut metotlar tanımlamak sorunu kolayca çözebilir.

Soyut metot (abstract method) adı ve parametreleri olduğu halde gövdesi olmayan bir metottur. Dolayısıyla belirli bir iş yapmaz. O, alt-sınıflarda örtülür (overriding). Sözdizimi şöyledir:

```
abstract [erişim_belirtkesi] veri_tipi metot_adı(parametre_listesi);
```

Soyut sınıf (abstract class) tanımlamak için abstract nitelemi eklenir. Her soyut sınıfın en az bir tane soyut metodu olmalıdır. Bu nedenle, soyut bir sınıfa ait nesne yaratılamaz (instantiate edilemez). Tersine olarak, içinde soyut bir metot olan her sınıf soyut bir sınıf olur. Soyut sınıfın gövdesinde constructor metodu, soyut metot(lar) ve gerekiyorsa somut metotlar yer alabilir. Sözdizimi şöyledir:

```
[Erişim_belirtkesi] abstract class sınıf_adı {

    // soyut sınıf içinde en az bir soyut bir metot yer almalıdır
    // soyut sınıf içinde somut metot olabilir
        // constructor metodu tanımlanabilir
}
```

Soyut bir sınıfın alt-sınıfları tanımlanabilir. Alt-sınıflarda, üstteki soyut metotlar örtülür (overriding). Eğer bir alt sınıfta örtülmemiş metot kalırsa, o alt-sınıf da soyut bir sınıf olur.

Soyut sınıfın referans değişkeni (işaretçi, pointer) tanımlanabilir, ama o işaretçi, soyut sınıfa ait bir nesne işaret edemez, çünkü öyle nesne oluşturulamaz. Ancak, alt-sınıflara ait nesneleri işaret edebilir. Asıl kullanılma yeri de budur.

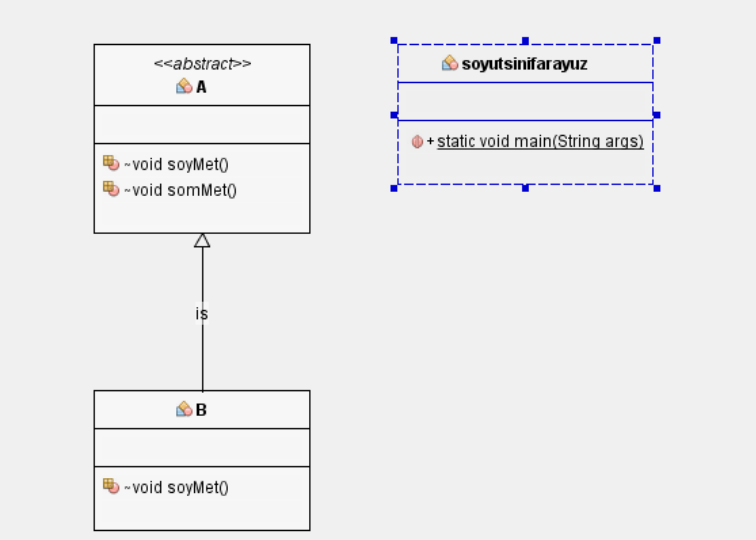
Liste 1:

Aşağıdaki örnek, soyut sınıf ve soyut metotların tanımlanışını göstermektedir.

Doç.Dr. İlhan AYDIN

```
1   package soyutsinifarayuz;
2   // soyut sınıf
3   abstract class A {
4           // soyut sınıf içinde en az bir soyut bir metot yer almalıdır
5     abstract void soyMet();
6         // soyut sınıf içinde somut metot olabilir
7     void somMet() {
8       System.out.println("Ben somut bir metodum.");
9     }
10  }
11  class B extends A {
12    void soyMet() {
13      System.out.println("Sen soyut bir metotsun.");
14    }
15  }
16  public class soyutsinifarayuz {
17    public static void main(String args[]) {
18      B b = new B();
19      b.soyMet();
20      b.somMet();
21    }
```

```
utput - soyutsinifarayuz (run)
run:
Sen soyut bir metotsun.
Ben somut bir metodum.
BUILD SUCCESSFUL (total time: 0 seconds)
```



Liste 2:

Aşağıdaki örnek, daha önce düzlemsel şekillerin alanlarını bulmak için yazdığımız Sekil sınıfının işlevini görecek soyut bir sınıf tanımlamaktadır.

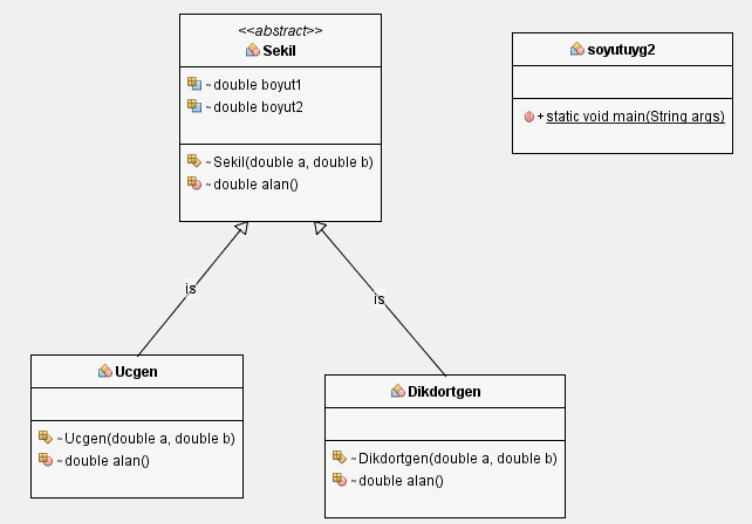
BMÜ-112 ALGORİTMA VE PROGRAMLAMA-II
SOYUT SINIFLAR VE ARAYÜZLER

Doç.Dr. İlhan AYDIN

```
1  package soyutsinifarayuz;
2  // soyut sınıf
3  abstract class Sekil {
4    double boyut1;
5    double boyut2;
6    Sekil(double a, double b) {
7      boyut1 = a;
8      boyut2 = b;  }
9          // soyut metot
10   abstract double alan();}
11 class Dikdortgen extends Sekil {
12   Dikdortgen(double a, double b) {    super(a, b);  }
13         // Dikdörtgen alanı için override
14   double alan() {
15     System.out.println("Dikdortgenin alanı .");
16     return boyut1 * boyut2;  }
17 }
18 class Ucgen extends Sekil {
19   Ucgen(double a, double b) {
20     super(a, b);
21   }
22         // Üçgen alanı için override
23   double alan() {
24     System.out.println("Ucgenin alanı .");
25     return boyut1 * boyut2 / 2;
26   }
27 }
```

```
29 public class soyutuyg2 {
30   public static void main(String args[]) {
31   // Sekil f = new Sekil(10, 10);          // deyim geçersizdir
32                                            // soyut sınıfa ait nesne yaratılamaz
33     Dikdortgen r = new Dikdortgen(9, 5);
34     Ucgen t = new Ucgen(10, 8);
35     Sekil ref; // referans değişkeni tanımlıyor; nesne işaret etmiyor
36                                    // deyim geçerlidir
37     ref = r;   // alt-sınıfa ait bir nesneyi işaret ediyor; deyim geçerlidir
38     System.out.println("Alan = " + ref.alan());
39     ref = t;   // alt-sınıfa ait bir nesneyi işaret ediyor; deyim geçerlidir
40     System.out.println("Alan = " + ref.alan());
41   }
42 }
```

```
Output - soyutsinifarayuz (run)
run:
Dikdortgenin alanı .
Alan = 45.0
Ucgenin alanı .
Alan = 40.0
BUILD SUCCESSFUL (total time: 0 seconds)
```



BMÜ-112 ALGORİTMA VE PROGRAMLAMA-II
SOYUT SINIFLAR VE ARAYÜZLER

Doç.Dr. İlhan AYDIN

Arayüz
(Interface)

Java'da arayüz soyut sınıf yerine kullanılır, ama soyut sınıftan farklı ve daha kullanışlıdır. Arayüz kullanarak, bir sınıfın neler yapacağını belirlerken, onları nasıl yapacağını gizleyebiliriz. Arayüzün yapısı sınıfın yapısına benzerse de aralarında önemli farklar vardır.

- Arayüz ,interface anahtar sözcüğü ile tanımlanır. Arayüz abstract metotlar içerir.
- Arayüz, anlık (instance) değişkenler içeremez. Ancak, erişim belirteci konmamış olsa bile, arayüz içindeki değişkenler final ve static olur. Bu demektir ki, arayüzde tanımlanan değişkenler, onu çağıran sınıflar tarafından değiştirilemez.
- Arayüz, yalnızca public ve ön-tanımlı (dafault) erişim belirtkesi alabilir, başka erişim belirteci alamaz.
- public damgalı arayüz, public damgalı class gibidir. Her kod ona erişebilir.
- Erişim damgasız arayüz, erişim damgasız class gibidir. Bu durumda, arayüze, ait olduğu paket içindeki bütün kodlar ona erişebilir. Paket dışındaki kodlar erişemez.
- Arayüz, public erişim belirtkesi ile nitelenmişse, içindeki bütün metotlar ve değişkenler otomatik olarak public nitelemesine sahip olur.
- Bir sınıf birden çok arayüze çağırabilir (implement).
- Aynı arayüzü birden çok sınıf çağırabilir.
- Sınıftaki metotlar tam olarak tanımlıdır, ama arayüzde metotların gövdeleri yoktur. Onlar abstract metotlardır. Metodun tipi, adı, parametreleri vardır, ama gövde tanımı yoktur; yani yaptığı iş belirli değildir. Metotların gövdesi, o arayüzü çağıran sınıf içinde yapılır. Böylece bir metot, farklı sınıflarda farklı tanımlanabilir. Bu özelik, Java'da polymorphismi olanaklı kılan önemli bir niteliktir.

Arayüzün Yapısı

Arayüzün genel yapısı aşağıdaki gibidir. Erişim-belirtkesi ya public olur ya da hiç konmaz, o zaman default erişim etkin olur.

```
[erişim_belirteci] interface  adı {
      tip metot_adı_01 (parametre_listesi);
      tip metot_adı_02 (parametre_listesi);
      tip final değişken_adı_01 = değer_01;
      tip final değişken_adı_02 = değer_01;
      // …
      tip metot_adı_N (parametre_listesi);
      tip final değişken_adı_N = değer_N;
}
```

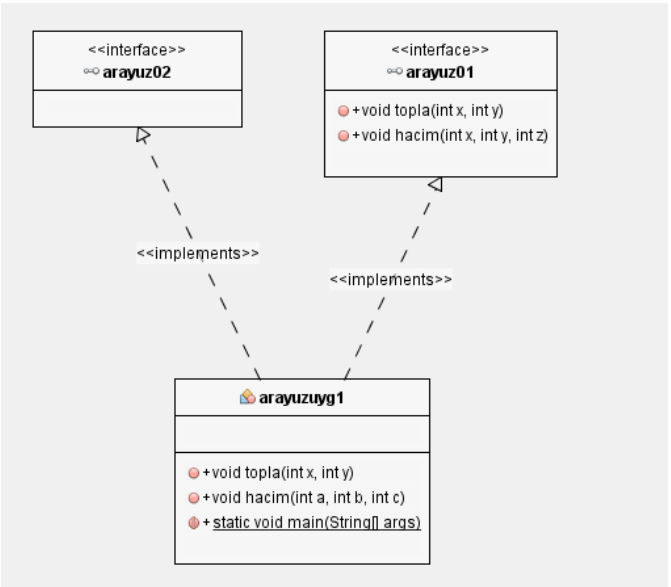
Örnek 1: Aşağıdaki sınıf, yukarıdaki iki arayüzü var etmektedir. Sınıf içinde, arayüzün metotlarının serbestçe tanımladığına dikkat ediniz. Aynı arayüzü var edecek başka sınıflar, bu metotları başka başka tanımlayabilirler. Bu, bir metodun farklı işler görmesini sağlar ve polymorphism diye bilinir.

BMÜ-112 ALGORİTMA VE PROGRAMLAMA-II
SOYUT SINIFLAR VE ARAYÜZLER

Doç.Dr. İlhan AYDIN

```
1   package soyutsinifarayuz;
    interface arayuz01{
          public void topla(int x, int y);
          public void hacim(int x, int y, int z);
5   }
    interface arayuz02{
          public static final double fiyat = 1250.00;
          public static final int sayac = 5;
9   }
10  class arayuzuyg1 implements arayuz01, arayuz02{
          public void topla(int x, int y){
12              System.out.println("x+y = " + (x+y));
13        }
          public void hacim(int a, int b, int c){
15              System.out.println("Hacim = " + (a*b*c));
16        }
17
18        public static void main(String[] args){
19              arayuzuyg1 d01 = new arayuzuyg1();
20              d01.topla(12,15);
21              d01.hacim(3,5,7);
                System.out.println(d01.fiyat);
                System.out.println(d01.sayac);
24        }
25  }
```

```
Output - soyutsinifarayuz (run)
run:
x+y = 27
Hacim = 105
1250.0
5
BUILD SUCCESSFUL (total time: 0 seconds)
```



Örnek 2:

Doç.Dr. İlhan AYDIN

```
1  package soyutsinifarayuz;
2  interface Aryz {  void aryzMet(int param);}
3  class Deneme02 implements Aryz {
4          // Implement Aryz's interface
5    public void aryzMet(int p) {
6      System.out.println("aryzMet metodu " + p + " ile çağrılıyor. " );}
7  }
8  class Deneme05 implements Aryz {
9          // Implement Aryz's interface
10   public void aryzMet(int p) {
11     System.out.println("Başka bir aryzMet");
12     System.out.println("p nin karesi = " + (p*p));
13   }
14 }
15 abstract class EksikClass implements Aryz {
16   int a, b;
17   void show() {
18     System.out.println(a + " " + b);
19   }
20 }
21 public class arayuz3 {
22     public static void main(String args[]) {
23    Aryz c = new Deneme02();
24     Deneme05 d05 = new Deneme05();
25     c.aryzMet(23);
26     c = d05; // c şimdi Deneme05'in bir nesnesini işaret ediyor
27     c.aryzMet(23);
28   }
29 }
```

```
Output - soyutsinifarayuz (run)
run:
aryzMet metodu 23 ile çağrılıyor.
Başka bir aryzMet
p nin karesi = 529
```

Örnek 3: Aşağıdaki örnek, interface kullanarak bir stack oluşturmaktadır.

BMÜ-112 ALGORİTMA VE PROGRAMLAMA-II
SOYUT SINIFLAR VE ARAYÜZLER

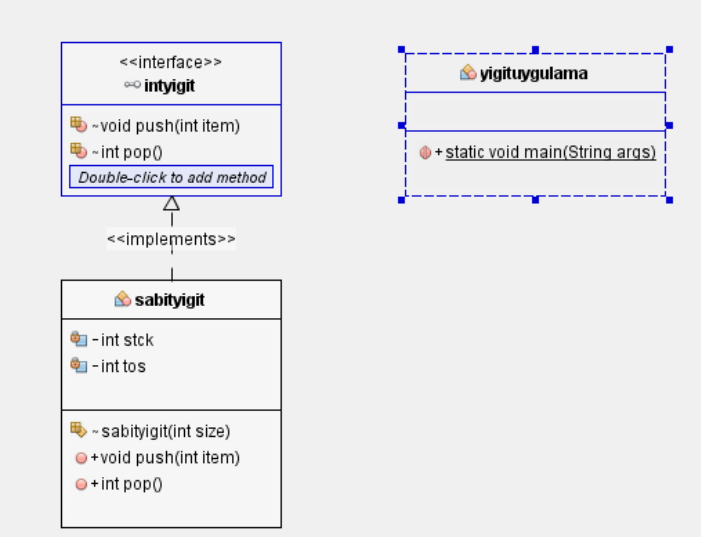
Doç.Dr. İlhan AYDIN

```
package soyutsinifarayuz;
interface intyigit {
  void push(int item); // Eleman ekle
  int pop(); // eleman cikar
}
class sabityigit implements intyigit {
  private int stck[];
  private int tos;
  // allocate and initialize stack
  sabityigit(int size) {
    stck = new int[size];
    tos = -1;
  }
  // Eleman ekleme
  public void push(int item) {
    if(tos==stck.length-1)
      System.out.println("Stack is full.");
    else
      stck[++tos] = item;
  }
  public int pop() {
    if(tos < 0) { System.out.println("Stack underflow.");
      return 0;}
    else      return stck[tos--];
  }
}
```

```
27    class yigituygulama {
28      public static void main(String args[]) {
29        sabityigit mystack1 = new sabityigit(5);
30        sabityigit mystack2 = new sabityigit(8);
31        // yigitlara eleman ekleme
32        for(int i=0; i<5; i++) mystack1.push(i);
33        for(int i=0; i<8; i++) mystack2.push(i);
34        // eleman cikarma
35        System.out.println("mystack1'deki elemanlar:");
36        for(int i=0; i<5; i++)
37          System.out.print(mystack1.pop()+" ");
38
39        System.out.println("\nmystack2'deki elemanlar:");
40        for(int i=0; i<8; i++)
41          System.out.print(mystack2.pop()+" ");
42      }
43    }
44
```

```
Output - soyutsinifarayuz (run)
run:
mystack1'deki elemanlar:
4 3 2 1 0
mystack2'deki elemanlar:
7 6 5 4 3 2 1 0 BUILD SUCCESSFUL (total time: 0 seconds)
```

Doç.Dr. İlhan AYDIN

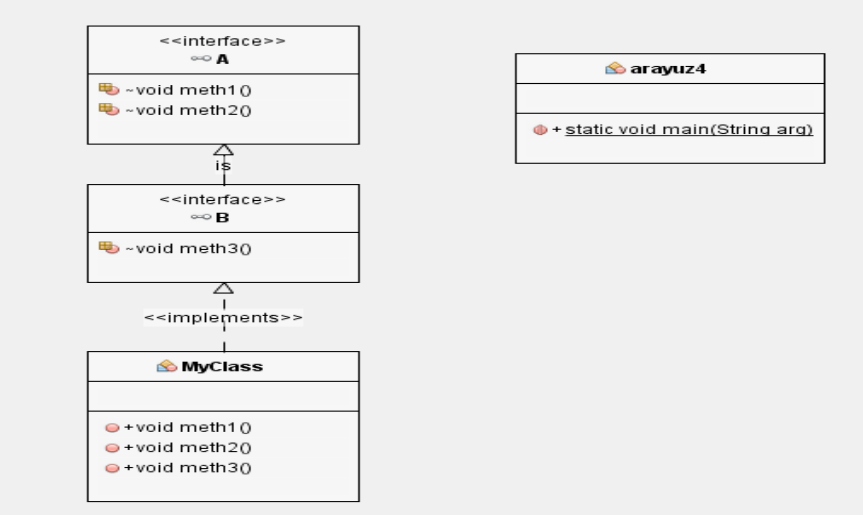


Örnek 4: Aşağıdaki örnek, sınıflarda olduğu gibi, arayüzlerin de alt-arayüzlerinin yaratılabileceğini göstermektedir.

```
// Bir arayuz diğerini extends eder.
interface A {
  void meth1();
  void meth2();
}
// B şu anda meth1() ve meth2()'yi ekler -- ayrıca meth3() e sahiptir.
interface B extends A {
  void meth3();
}
// Bu sınıf A ve B'nin tum metotlarını implement etmeli
class MyClass implements B {
  public void meth1() {     System.out.println("Implement meth1().");   }
  public void meth2() {     System.out.println("Implement meth2().");  }
  public void meth3() {     System.out.println("Implement meth3().");  }
}
 public class arayuz4 {
  public static void main(String arg[]) {
    MyClass ob = new MyClass();
    ob.meth1();
    ob.meth2();
    ob.meth3();
  }
}
```

Doç.Dr. İlhan AYDIN

```
Output - soyutsinifarayuz (run)
run:
Implement meth1().
Implement meth2().
Implement meth3().
BUILD SUCCESSFUL (total time: 0 seconds)
```



Örnek-5:

```
abstract class Shape{
    String objectName = " ";
    Shape(String name)    {          this.objectName = name;      }
     public void moveTo(int x, int y) {
        System.out.println(this.objectName + " " + " x = " + x + " ve y =" + y+
                " noktasına taşındı"); }
    abstract public double area();
    abstract public void draw(); }
class Rectangle extends Shape{
    int length, width;
    Rectangle(int length, int width, String name) {
        super(name);
        this.length = length;
        this.width = width;}
    @Override
    public void draw(){ System.out.println("Rectangle has been drawn ");  }
    @Override
    public double area()     {
        return (double)(length*width);   }
}
class Circle extends Shape {
    double pi = 3.14;
    int radius;
    //constructor
    Circle(int radius, String name){
        super(name);
        this.radius = radius;}
    @Override
    public void draw() {  System.out.println("Circle has been drawn ");  }
    @Override
    public double area()    {          return (double)((pi*radius*radius)/2);  }
}
```

Doç.Dr. İlhan AYDIN

```
public class arayuz5 {
    public static void main (String[] args)      {
        Shape rect = new Rectangle(2,3, "Dikdortgen");
        System.out.println("Dikdortgen alanı: " + rect.area());
        rect.moveTo(1,2);
        System.out.println(" ");
        // creating the Objects of circle class
        Shape circle = new Circle(2, "Cember");
        System.out.println("Cember alanı: " + circle.area());
        circle.moveTo(2,4);

    }
}
```

```
- soyutsinifarayuz (run)
run:
Dikdortgen alanı: 6.0
Dikdortgen  x = 1 ve y =2 noktasına taşındı

Cember alanı: 6.28
Cember  x = 2 ve y =4 noktasına taşındı
BUILD SUCCESSFUL (total time: 0 seconds)
```

Örnek-6: Aynı işi interface ile yapalım.

```
interface sekil {
    void ciz();
    double alan(); }
class dikdortgen implements sekil{
    int length, width;
    // constructor
    dikdortgen(int length, int width)     {
        this.length = length;
        this.width = width; }
    @Override
    public void ciz()       {
        System.out.println("Rectangle has been drawn ");
    }
    @Override
    public double alan() {
        return (double)(length*width);      }
}
 class cember implements sekil{
    double pi = 3.14;
    int radius;
    cember(int radius)    {          this.radius = radius;      }
    @Override
    public void ciz()       {
        System.out.println("Circle has been drawn ");     }
    @Override
    public double alan() {         return (double)((pi*radius*radius)/2);   }
}
```

Doç.Dr. İlhan AYDIN

```
28      public class uygulama6{
29        public static void main (String[] args)
30        {
31            sekil rect = new dikdortgen(2,3);
32            System.out.println("Dikdortgen alanı: " + rect.alan());
33            sekil c = new cember(2);
34            System.out.println("cember alanı: " + c.alan());
35        }
36    }
```

```
Output - soyutsinifarayuz (run)
run:
Dikdortgen alanı: 6.0
cember alanı: 6.28
BUILD SUCCESSFUL (total time: 0 seconds)
```

Abstract class ve Interface Farkı

- **Yöntem türü:** Interface yalnızca soyut yöntemlere sahip olabilir. Soyut sınıf soyut ve soyut olmayan yöntemlere sahip olabilir. Java 8'den, varsayılan ve statik yöntemlere de sahip olabilir.
- **Final Değişkenler:** Java Arayuzlerinde bildirilen değişkenler varsayılan olarak final'dır. Soyut bir sınıf, final olmayan değişkenler içerebilir.
- **Değişken türleri:** Soyut sınıf, final, final olmayan, statik ve statik olmayan değişkenlere sahip olabilir. Interface yalnızca statik ve son değişkenleri vardır.
- **Kalıtım ve Soyutlama:** Java arabirimi "implements" anahtar sözcüğü kullanılarak uygulanabilir ve soyut sınıf, "extends" anahtar sözcüğü kullanılarak genişletilebilir.
- **Çoklu uygulama:** Bir ınterface yalnızca başka bir Java interface'ini genişletebilir, soyut bir sınıf başka bir Java sınıfını genişletebilir ve birden fazla Java interface'ini uygulayabilir.
- **abstract class A extends B implements D,E {}**
- **Veri Üyelerinin Erişilebilirliği:** Bir Java interface üyeleri varsayılan olarak herkese açıktır. Java soyut sınıfında özel, korumalı vb. Sınıf üyeleri bulunabilir.

Ne zaman kullanılır?

Bu ifadelerden herhangi biri durumunuz için geçerliyse soyut sınıfları kullanmayı düşünün:

- Java uygulamasında, bazı kod satırlarını paylaşmanız gereken bazı ilgili sınıflar vardır, o zaman bu kod satırlarını soyut sınıf içine koyabilirsiniz ve bu soyut sınıf, ilgili tüm sınıflar tarafından genişletilmelidir.
- Soyut sınıfta statik olmayan veya final olmayan alanları tanımlayabilirsiniz, böylece bir yöntemle ait oldukları Nesnenin durumuna erişebilir ve bunları değiştirebilirsiniz.
- Soyut bir sınıfı genişleten sınıfların birçok ortak yöntem veya alana sahip olmasını veya genel (erişim ve koruma gibi) dışında erişim değiştiricileri gerektirmesini bekleyebilirsiniz.

Bu ifadelerden herhangi biri durumunuz için geçerliyse arayüzleri kullanmayı düşünün:

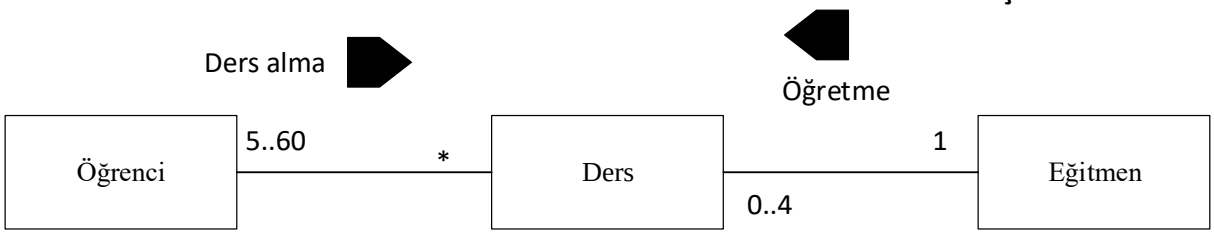
- Tam bir soyutlamadır, Bir arabirim içinde bildirilen tüm yöntemler, bu arabirimi uygulayan sınıf (lar) tarafından uygulanmalıdır.
- Bir sınıf birden fazla arabirim uygulayabilir. Buna çoklu kalıtım denir.

İlişkilendirme(Assocation)

İlişkilendirme, iki sınıf arasındaki bir etkinliği tanımlayan genel bir ikili ilişkidir. Örneğin, bir dersi alan bir öğrenci Öğrenci sınıfı ve Ders sınıfı ve verdiğim ders öğretim üyesi arasında bir ilişki Fakültesi sınıfı ve Ders sınıfın arasında bir ilişki olmasıdır. Bu ilişkilendirmeler, Şekil'de gösterildiği gibi UML grafik gösteriminde temsil edilebilir.

BMÜ-112 ALGORİTMA VE PROGRAMLAMA-II
SOYUT SINIFLAR VE ARAYÜZLER

Doç.Dr. İlhan AYDIN




```
class ogrenci {
    int no;
    String adsoyad;
    int derssay;
    int i;
    private ders[] dersliste;
    public ogrenci(int no, String adsoyad, int derssay) {
        this.no = no;
        this.adsoyad = adsoyad;
        this.derssay = derssay;
        dersliste=new ders[derssay];
        i=0;
    }
    public void eklederss(ders c){
        if(i<derssay){
        dersliste[i]=c;
        i++;
        }
    }
}
```

```
class ders{
    int kod;
    String dersad;
    private ogrenci[] sinifliste;
    private egitmen f;
    int ogrsay,i;

    public ders(int kod, String dersad, egitmen f) {
        this.kod = kod;
        this.dersad = dersad;
        sinifliste=new ogrenci[ogrsay];
        this.f = f;
        this.ogrsay = 20;
        i=0;
    }
   void ekleogrenci(ogrenci e){
       if(i<ogrsay){
        sinifliste[i]=e;
        i++;
        }
   }
   void setegitmen(egitmen f){        this.f=f;
   }
}
```

```
class egitmen{
    String egtadi;
    int derssay=20,i;
private ders[] dersliste;
    public egitmen(String egtadi) {
        this.egtadi = egtadi;
        dersliste=new ders[derssay];
        i=0;
    }
    void eklederss(ders c){


    }
}
```

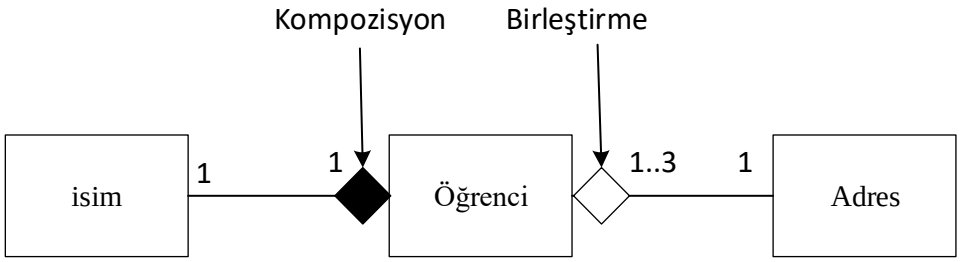
Bir ilişki, ilişkiyi tanımlayan isteğe bağlı bir etiketle iki sınıf arasındaki düz bir çizgi ile gösterilmiştir. UML Şeklinde etiketler ders alma ve öğretme'dir. Her ilişkinin, ilişkinin yönünü gösteren isteğe bağlı küçük bir siyah üçgeni olabilir. Bu şekilde, N bir öğrencinin bir dersi aldığını gösterir (öğrenciyi alan bir dersin aksine). İlişkide yer alan her sınıf, ilişkide oynadığı rolü tanımlayan bir rol adına sahip olabilir.

Birleştirme ve Kompozisyon

Birleştirme, iki nesne arasındaki sahiplik ilişkisini temsil eden özel bir ilişki biçimidir. Birleştirme modellerinin ilişkileri vardır. Owner nesnesine toplama nesnesi ve sınıfına birleştirme sınıfı denir. Konu nesnesine birleştirilmiş nesne denir ve sınıfına birleştirilmiş sınıf denir.

Birleştirilen nesnenin varlığı birleşen nesneye bağlıysa, iki nesne arasındaki toplama işlemini kompozisyon olarak adlandırırız. Başka bir deyişle, bir ilişki kompozisyonsa, birleştirilmiş nesne kendi başına var olamaz. Örneğin, “bir öğrencinin adı vardır”, Öğrenci sınıfı ile Ad sınıfı arasındaki bir

kompozisyon ilişkisidir, çünkü Ad Öğrenci'ye bağlıdır, "bir öğrencinin bir adresi vardır", Öğrenci sınıfı ile Adres sınıfı arasında bir birleşme ilişkisidir. Bir adres kendi başına var olabilir. Kompozisyon münhasır sahiplik anlamına gelir. Bir nesnenin başka bir nesnesi vardır. Sahip nesne yok edildiğinde, bağımlı nesne de yok edilir. UML'de, doldurulmuş bir elmas birleştirilmiş bir sınıfla (Ad) kompozisyon ilişkisini belirtmek için bir birleştirilmiş sınıfına (bu durumda Öğrenci) eklenir ve birleştirme ilişkisini göstermek için bir toplama sınıfına (Öğrenci) birleştirilmiş sınıfla (Adres) boş bir elmas eklenir.



her öğrencinin yalnızca bir çokluk adresi vardır ve her adres en fazla 3 öğrenci tarafından paylaşılabilir. Her öğrencinin bir adı vardır ve adı her öğrenci için benzersizdir.

public class Isim {
...
}
	public class ogrenci {
	private Isim ad;
	private Adress adr;
	...
	}
		public class Adres {
		...
		}

Birleştirilen sınıf	**Birleştiren sınıf**	**Birleştirilen sınıf**