

Görsel Programlama

DERS 02

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Nesneye yönelik programlamanın diğer programlama paradigmalarına karşı bir avantajı kodun yeniden kullanılabilirliğidir (code reuseability).

Yazılan bir sınıf ile faydalı bir program birimi oluşturulmuş olur.

Bu sınıf tekrar tekrar kullanılabilir.

Bu sınıfı temel alarak, yeni kod parçaları ekleyerek yeni yeni sınıflar oluşturulabilir.

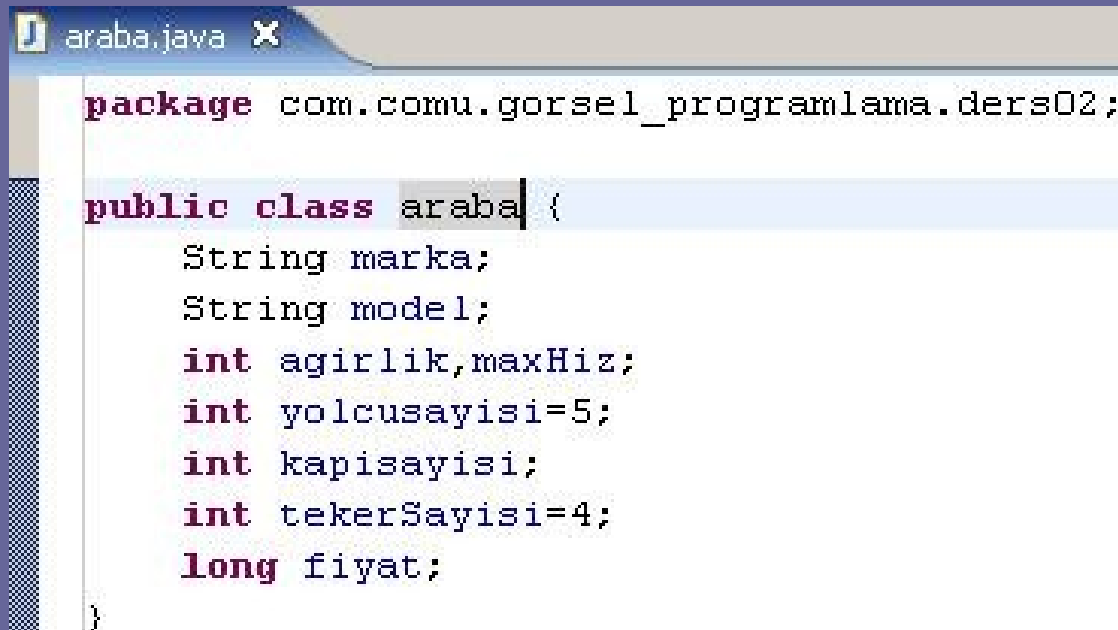
Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Bir sınıfın başka bir sınıf içerisinde bir örneği oluşturulup bir değişken olarak kullanılabilir. Bu kullanıma **kompozisyon (composition)** denilir.

Diğer bir kod tekrar kullanım yöntemi kalıtım(inheritance) dır. Bir sınıf temelinde başka bir sınıf oluşturulur. Yeni oluşturulan sınıf kalıtılmış olduğu ana sınıfın tüm özelliklerini (alanlarını ve metotlarını) miras alır ve kendisine yeni özellikler katabilir.

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Kalıtımın Java ' da uygulanması:



```
araba.java X
package com.comu.gorsel_programlama.ders02;

public class araba {
    String marka;
    String model;
    int agirlik,maxHiz;
    int yolcusayisi=5;
    int kapisayisi;
    int tekerSayisi=4;
    long fiyat;
}
```

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Motorsiklet nesnesini modellemek istersek:

A screenshot of a Java code editor window. The title bar shows 'motorsiklet.java' with a close button. The code is as follows:

```
package com.comu.gorsel_programlama.ders02;

public class motorsiklet {
    String marka;
    String model;
    int agirlik,maxHiz;
    int yolcusayisi=2;
    int tekersayisi=2;
    long fiyat;
}
```

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Dikkat edilirse iki sınıfta ortak özellikleri vardır:

```
araba.java x
package com.comu.gorsel_programlama.ders02;

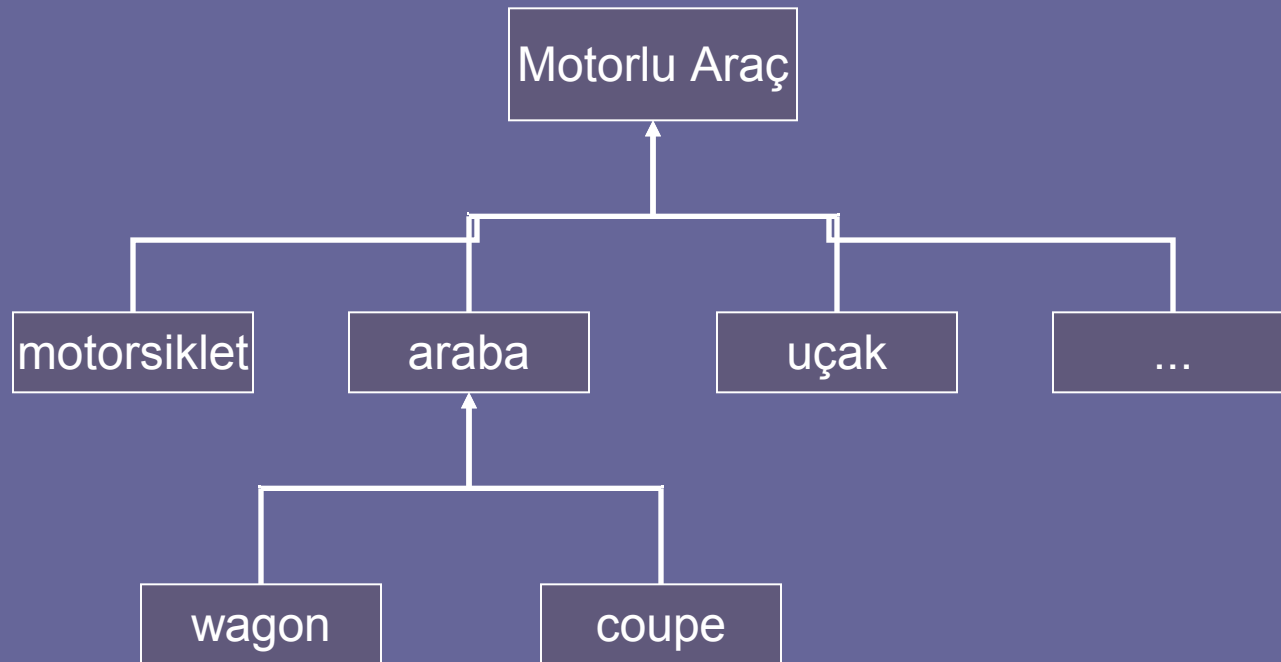
public class araba {
    String marka;
    String model;
    int agirlik,maxHiz;
    int yolcusayisi=5;
    int kapisayisi;
    int tekerSayisi=4;
    long fiyat;
}
```

```
motorsiklet.java x
package com.comu.gorsel_programlama.ders02;

public class motorsiklet {
    String marka;
    String model;
    int agirlik,maxHiz;
    int yolcusayisi=2;
    int tekersayisi=2;
    long fiyat;
}
```

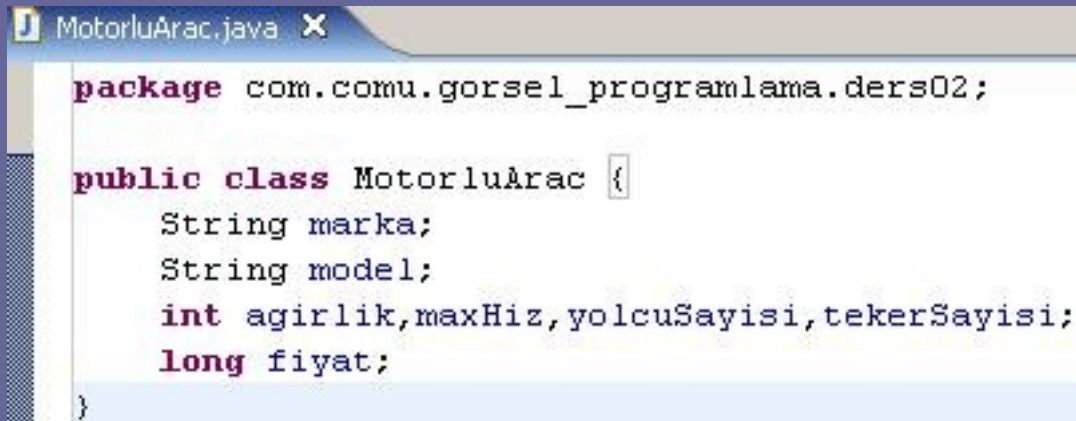
Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Kalıtımın (inheritance) amacı soyutlama (abstraction) sağlamaktır; yani her sınıfın içinde mümkün olduğunca o sınıfa özel alan ve metotların tanımlanmasıdır.



Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Sınıf hiyerarşisindeki sınıfların Java kodları:

A screenshot of a Java code editor window titled "MotorluArac.java". The code defines a package and a public class with several attributes.

```
package com.comu.gorsel_programlama.ders02;

public class MotorluArac {
    String marka;
    String model;
    int agirlik,maxHiz,yolcuSayisi,tekerSayisi;
    long fiyat;
}
```


Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

```
araba2.java X
package com.comu.gorsel_programlama.ders02;

public class araba2 extends MotorluArac {
    int kapisayisi;
    public araba2() {
        tekerSayisi=4;
        yolcuSayisi=5;
    }
}
```

```
motorsiklet2.java X
package com.comu.gorsel_programlama.ders02;

public class motorsiklet2 extends MotorluArac {
    public motorsiklet2() {
        tekerSayisi=2;
        yolcuSayisi=2;
    }
}
```

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

araba2 ve motorsiklet2 MotorluArac sınıfının alt sınıflarıdır(sub class).

Üst sınıflarından tüm değişkenleri(marka,model,yolcuSayisi,tekerSayisi,...) miras yoluyla alırlar ve kullanabilirler.

ÜST SINIFLARDA **private** erişim denetleyicileri ile tanımlanan alanlar ve metotlar alt sınıflarca miras olarak alınmalarına rağmen alt sınıfın metotları tarafından görülemez ve kullanılamazlar; erişilemezler.

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

Kalıtım ile oluşturulmuş sınıftan yeni sınıflar kalıtım yoluyla oluşturulabilirler.

Bazen bir sınıftan kalıtım yolu ile yeni sınıfların türetilmesini istemeyiz. Bu durumda sınıfımızı son sınıf (final class) olarak tanımlarız ve artık bu sınıftan yeni sınıfların kalıtım yolu ile türetilmesini engellemiş oluruz.

Kodun Tekrar Kullanımı ve Kalıtım(Inheritance)

```
wagon.java x
package com.comu.gorsel_programlama.ders02;

final public class wagon extends araba2 {
    public wagon() {
        kapisayisi=5;
    }
}
```

```
wagon2.java x
package com.comu.gorsel_programlama.ders02;

public class wagon2 extends wagon {
    |
}
```

```
wagon2.java x
package com.comu.gorsel_programlama.ders02;

The type wagon2 cannot subclass the final class wagon{
    |
}
```

wagon sınıfı final olduğu için bu sınıftan kalıtım yoluyla yeni sınıf oluşturulamaz.

Geçersiz Kılma (Overriding)

Alt sınıflar üst sınıfın tüm alan ve metotlarını kalıtım ile alır, fakat istenilirse bu metotların davranışlarını (yaptıkları işleri) değiştirebilir.

Bu işleme metotların **geçersiz kılınması(overriding)** denilir.

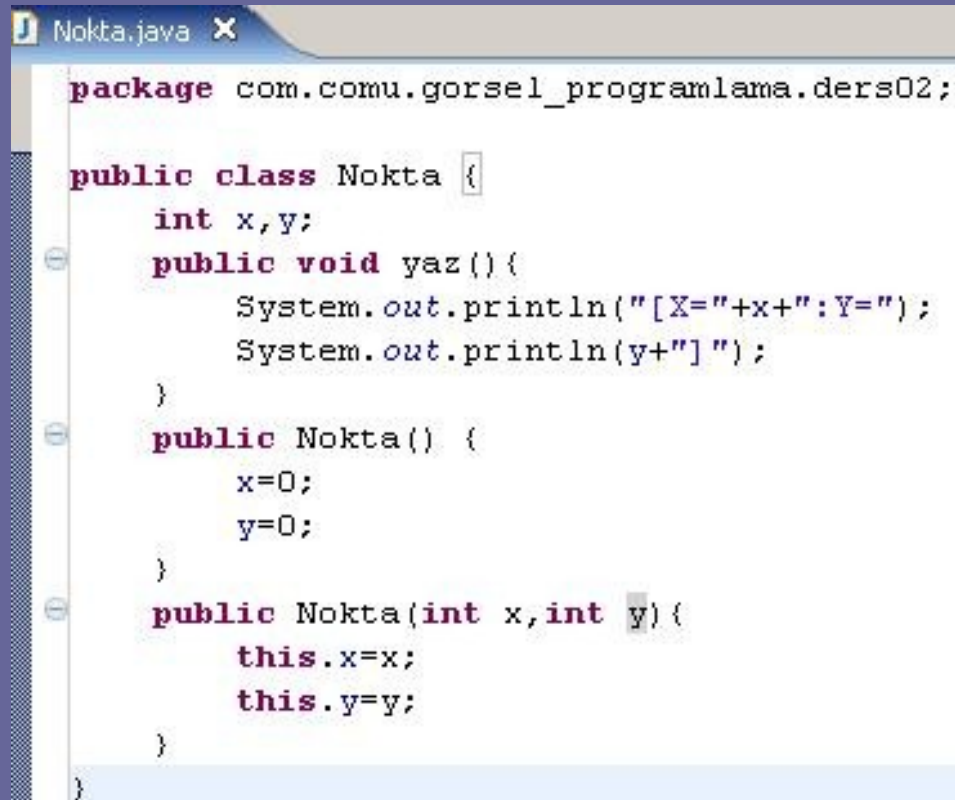
Geçersiz Kılma (Overriding)

Bu işlemi gerçekleştirmek için şunlar yapılır:

4. Alt sınıf içinde üst sınıfın geçersiz kılınacak metot aynı isim ve parametreler ile yeniden yazılır.
5. Yazılan metot içeriği değiştirilerek yeni yapması istenilen iş kodlanır.

Geçersiz Kılma (Overriding)

Bu işlemi gerçekleştirmek için şunlar yapılır:



```
package com.comu.gorsel_programlama.ders02;

public class Nokta {
    int x,y;
    public void yaz(){
        System.out.println("[X="+x+":Y=");
        System.out.println(y+"]");
    }
    public Nokta() {
        x=0;
        y=0;
    }
    public Nokta(int x,int y){
        this.x=x;
        this.y=y;
    }
}
```

Geçersiz Kılma (Overriding)

```
RenkliNokta.java X
package com.comu.gorsel_programlama.ders02;

public class RenkliNokta extends Nokta {
    String renk;
    //geçersiz kılınan metot (overriding)
    public void yaz(){
        System.out.println("[X="+x+":Y=");
        System.out.println(y+"] Renk="+renk);
    }
    public RenkliNokta(int x,int y, String renk) {
        this.x=x;
        this.y=y;
        this.renk=renk;
    }
}
```


Geçersiz Kılma (Overriding)

Sınıfta yazmış olduğumuz bir metodun geçersiz kılınmasını (overriding) engellemek istersek, bu metodu son metot (final method) olarak tanımlarız.

Örn:

```
final public void yaz()
```

Bu şekilde tanımlanan metot geçersiz kılınmaya çalışılırsa hata verecektir.

Geçersiz Kılma (Overriding)

```
Nokta.java x RenkliNokta.java
package com.comu.gorsel_programlama.ders02;

public class Nokta {
    int x,y;
    final public void yaz(){
        System.out.println("[X="+x+":Y=");
        System.out.println(y+"]");
    }
    public Nokta() {
        x=0;
        y=0;
    }
    public Nokta(int x,int y){
        this.x=x;
        this.y=y;
    }
}
```

```
Nokta.java RenkliNokta.java x
package com.comu.gorsel_programlama.ders02;

public class RenkliNokta extends Nokta {
    String renk;
    //geçersiz kılınan metot (overriding)
    public void yaz(){
        System.out.
        System.out.
    }
    public RenkliNokta(int x,int y, String renk) {
        this.x=x;
        this.y=y;
        this.renk=renk;
    }
}
```

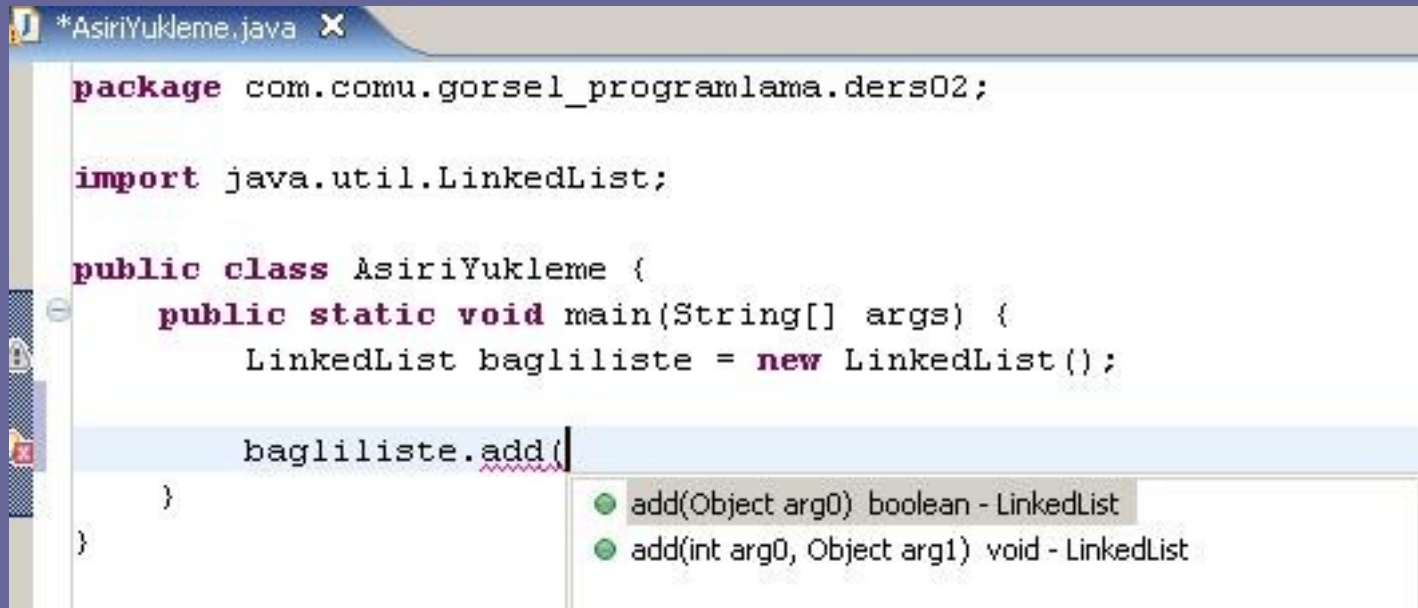
Cannot override the final method from Nokta
Press 'F2' for focus.

Metotların Üzerine Yükleme – Aşırı yükleme(overloading)

Bir metodun aşırı yüklenmesi; bir metodun farklı parametre sayıları yada tipleri ile aynı isim altında tanımlanması ve kullanılması demektir.

Metotların Üzerine Yükleme – Aşırı yükleme(overloading)

Aynı isimli iki **add** metodu vardır. İlki verilen nesneyi listenin sonuna ekler; ikincisi verilen nesneyi istenilen indis yerine ekler. **add** metodu **aşırı yüklenmiş** bir metottur.



```
*AsiriYukleme.java X
package com.comu.gorsel_programlama.ders02;

import java.util.LinkedList;

public class AsiriYukleme {
    public static void main(String[] args) {
        LinkedList bagliliste = new LinkedList();

        bagliliste.add(
    }
}
```

add(Object arg0) boolean - LinkedList
add(int arg0, Object arg1) void - LinkedList

Metotların Üzerine Yükleme – Aşırı yükleme(overloading)

Nokta sınıfının yapıcılarıda aşırı yükleme ile oluşturulmuştur. Yeni bir Nokta sınıfından nesne oluştururken iki şekilde oluşturabiliriz. Yorumlayıcı hangisini kullanmak istediğimizi parametrelere bakarak karar verir.

```
Nokta.java X
package com.comu.gorsel_programlama.ders02;

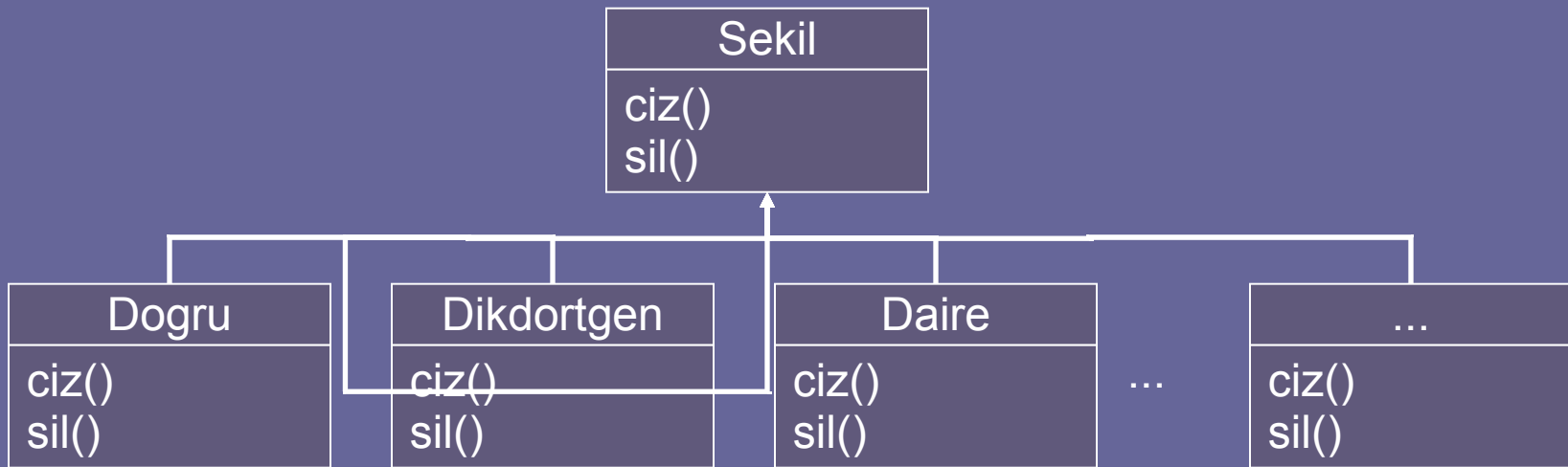
public class Nokta {
    int x,y;
    public void yaz(){
        System.out.println("[X="+x+":Y=");
        System.out.println(y+"] ");
    }
    public Nokta() {
        x=0;
        y=0;
    }
    public Nokta(int x,int y){
        this.x=x;
        this.y=y;
    }
}
```

```
Nokta nokta1 = new Nokta();
Nokta nokta2 = new Nokta(10,20);
```

Çok Şekillilik (Polimorphism)

Kalıtım yolu ile oluşturulan sınıflar bir sınıf hiyerarşisi oluştururlar ve alt sınıfların hepsinde üst sınıfın arayüzü ortaktır.

Her alt sınıf aynı zamanda birer üst sınıftır ve üst sınıf olarak kullanılabilirler.



Örnek bir geometrik şekil sınıf hiyerarşisi

Çok Şekillilik (Polimorphism)

Burada “Dogru bir şekildir”, “Daire bir şekildir” diyebiliriz.

```
Sekil.java X
package com.comu.gorsel_programlama.ders02;

public class Sekil {
    public void ciz(){}
    public void sil(){}
}
```

```
Dogru.java X
package com.comu.gorsel_programlama.ders02;

public class Dogru extends Sekil {

    public void ciz() {
        System.out.println("Dogru ciz");
    }

    public void sil() {
        System.out.println("Dogru sil");
    }
}
```

Çok Şekillilik (Polimorphism)

```
Dikdortgen.java X
package com.comu.gorsel_programlama.ders02;

public class Dikdortgen extends Sekil {

    public void ciz() {
        System.out.println("Dikdortgen ciz");
    }

    public void sil() {
        System.out.println("Dikdortgen sil");
    }

}
```

```
Daire.java X
package com.comu.gorsel_programlama.ders02;

public class Daire extends Sekil {

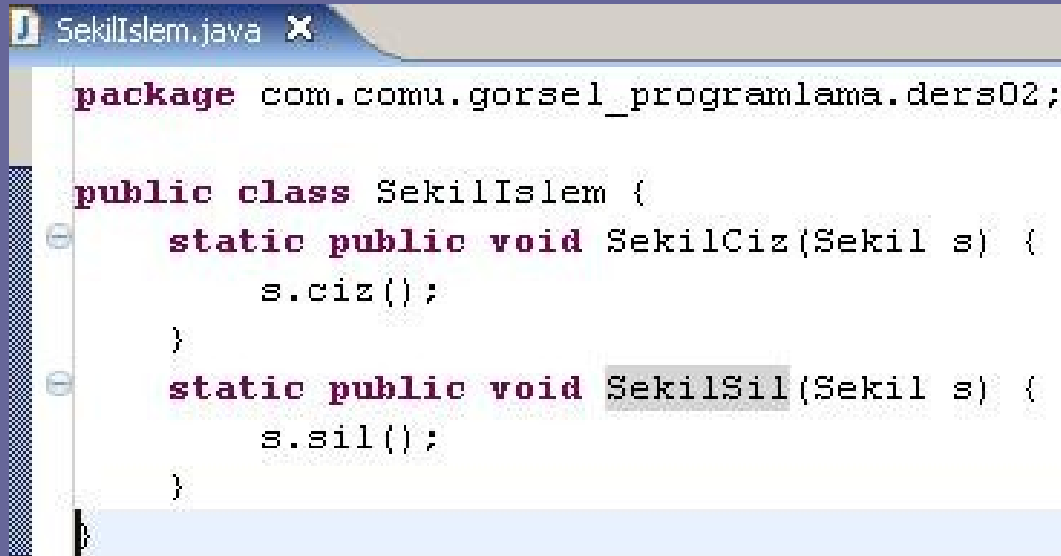
    public void ciz() {
        System.out.println("Daire ciz");
    }

    public void sil() {
        System.out.println("Daire sil");
    }

}
```


Çok Şekillilik (Polimorphism)

Aşağıda tanımlanan sınıfın metotlarına istenilen Şekil nesnesini(Dogru, Dikdortgen, Daire,...) gönderebiliriz. Hangisini çalışacağına Java program çalışırken karar verir ve çalıştırır.

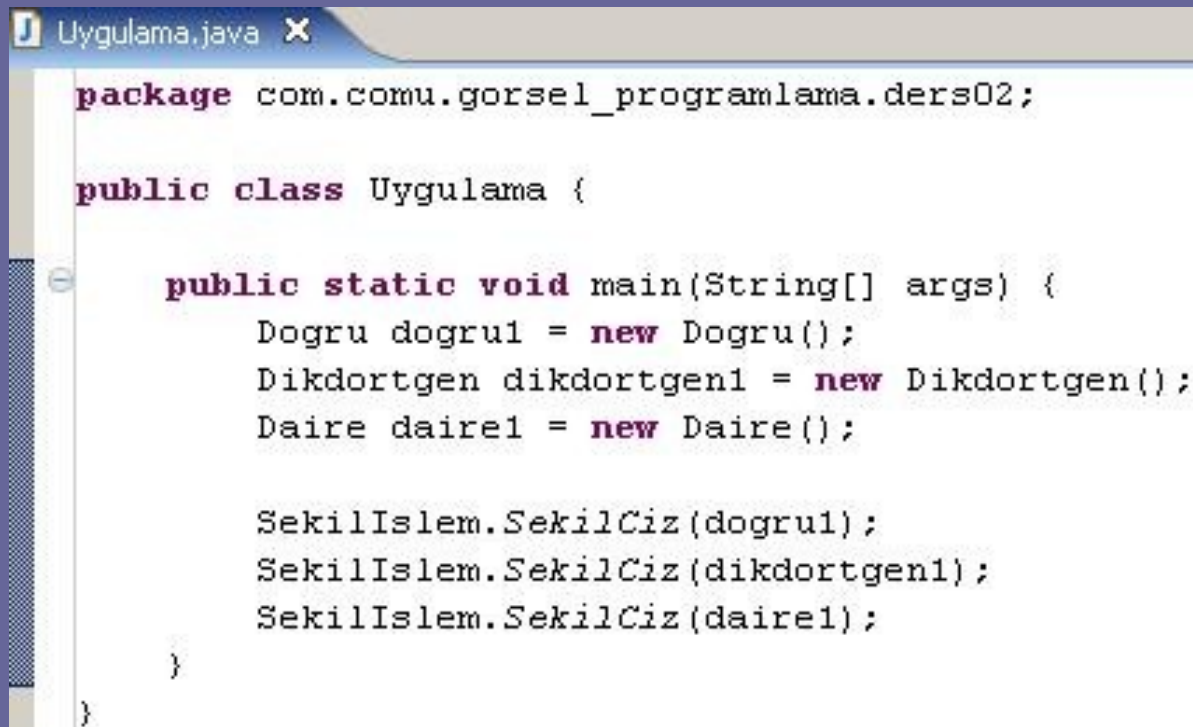


```
package com.comu.gorsel_programlama.ders02;

public class SekilIslem {
    static public void SekilCiz(Sekil s) {
        s.ciz();
    }
    static public void SekilSil(Sekil s) {
        s.sil();
    }
}
```

Çok Şekillilik (Polimorphism)

Uygulama:



```
Uygulama.java x
package com.comu.gorsel_programlama.ders02;

public class Uygulama {

    public static void main(String[] args) {
        Dogru dogru1 = new Dogru();
        Dikdortgen dikdortgen1 = new Dikdortgen();
        Daire daire1 = new Daire();

        SekilIslem.SekilCiz(dogru1);
        SekilIslem.SekilCiz(dikdortgen1);
        SekilIslem.SekilCiz(daire1);
    }
}
```

Bu şekilde gelen soyut sınıflardan hangisinin metodunun çalıştırılacağına karar verme işlemine **dinamik bağlama (dynamic binding)** denilir.

Soyut Sınıflar (Abstract Class) ve Soyut Metotlar (Abstract Methods)

Soyut (abstract) kelimesi ile oluşturulurlar.

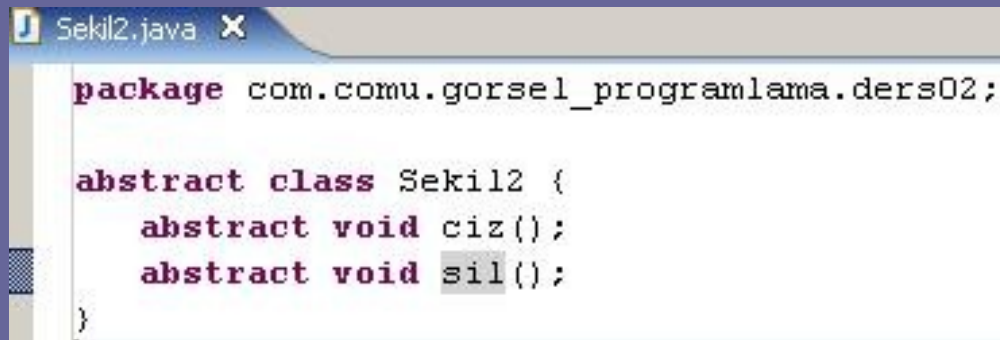
Bu şekilde tanımlanan metotlar sadece tanımlanırlar, içlerine kod yazılmaz.

Bu metotların kodları kalıtım ile oluşturulmuş olan alt sınıflarda yazılır.

Soyut sınıf kullanılarak nesne oluşturulamaz. Bu sınıflar sadece bir soyutlama sağlayarak alt sınıflar için bir ortak arayüz sağlama amacıyla geliştirilirler.

Soyut Sınıflar (Abstract Class) ve Soyut Metotlar (Abstract Methods)

Örnek;

A screenshot of a Java code editor window titled 'Sekil2.java'. The code defines an abstract class 'Sekil2' with two abstract methods: 'ciz()' and 'sil()'. The package is 'com.comu.gorsel_programlama.ders02'.

```
package com.comu.gorsel_programlama.ders02;  
  
abstract class Sekil2 {  
    abstract void ciz();  
    abstract void sil();  
}
```

Sekil2 sekil1= new Sekil2(); **X YAPILAMAZ!!!**

Arayüzler (Interfaces)

Arayüzler veri soyutlamanın gelişmiş şeklidir. Özellikleri :

- 1-Soyut sınıflar gibi metotlar gerçekleştirilmemiştir.
- 2-Bütün alanlar belirtilmese de public,static ve final dır.
- 3-Bütün alanlar ve metotlar (public) olmalıdır.
- 4-Bütün metotlar abstract ve public tir.

Arayüzler (Interfaces)

Java dilinde **çoklu kalıtım (multi inheritance)** yoktur. Her sınıf sadece tek bir sınıftan kalıtım yolu ile üretilebilir. Gerçek hayatta farklı nesnelerin ortak özellikleri olabilir yani farklı farklı sınıflarla ortak özellikleri olabilir.

Bu durumda arayüzler (interface) kullanılarak çoklu kalıtım sağlanabilir. Arayüzlerin tanımlanmasında sınıflara benzer fakat arayüzler birden fazla arayüzden kalıtım yolu ile çoğaltılabilirler.

Arayüzler (Interfaces)

```
public interface Arayüzismi [extends Ust Arayuzler]{  
    //Arayuz Govdesi  
}
```

Arayüzler (Interfaces)

```
Dovusebilir.java X
package com.comu.gorsel_programlama.ders02;

public interface Dovusebilir {
    void dovus();
}
```

```
Yuzebilir.java X
package com.comu.gorsel_programlama.ders02;

public interface Yuzebilir {
    void yuz();
}
```

```
Ucabilir.java X
package com.comu.gorsel_programlama.ders02;

public interface Ucabilir {
    void uc();
}
```


Arayüzler (Interfaces)

```
FilmKarakteri.java X
package com.comu.gorsel_programlama.ders02;

public class FilmKarakteri {
    public void savas() {
        System.out.println("Savaş");
    }
}

Kahraman.java X
package com.comu.gorsel_programlama.ders02;

public class Kahraman extends FilmKarakteri
    implements Dovusebilir, Yuzebilir, Ucabilir {
    public void dovus() {
        System.out.println("Dövüş");
    }

    public void yuz() {
        System.out.println("Yüz");
    }

    public void uc() {
        System.out.println("Uç");
    }
}
```

Arayüzler (Interfaces)

```
Macera.java x
package com.comu.gorsel_programlama.ders02;

public class Macera {
    public static void Savas(FilmKarakter f){
        f.savas();
    }
    public static void Yuz(Yuzebilir y){
        y.yuz();
    }
    public static void Dovus(Dovusebilir d){
        d.dovus();
    }
    public static void Uc(Ucabilir u){
        u.uc();
    }
    public static void main(String[] args) {
        Kahraman superman = new Kahraman();
        Savas(superman);
        Yuz(superman);
        Dovus(superman);
        Uc(superman);
    }
}
```

Görsel Programlama

DERS 02