

# NESNE YÖNELİMLİ PROGRAMLAMA 2(Object Oriented Programming 2/OOP)

**Öğr. Gör. Celil ÖZTÜRK**

Marmara Üniversitesi

Teknik Bilimler Meslek Yüksekokulu

# İçerik

- ✓ Arayüzler(Interface)
- ✓ Overriding,Overloading
- ✓ Static
- ✓ Abstract
- ✓ Klavye Olayları
- ✓ Buton Sınıfı
- ✓ Mouse Event

# Overriding

- Bir alt-sınıfta tanımlanan bir metodun veri-tipi (değer bölgesi), adı ve parametrelerinin sayısı ve tipleri, üst-sınıfında tanımlı bir başka metodun veri-tipi (değer bölgesi), adı ve parametrelerinin sayısı ve tipleri ile, karşılıklı olarak, aynı ise, alt-sınıftaki metod üst-sınıftakini örtüyor (**overriding**) denilir.




# Overriding

```
class A
{
    int i,j;
    public A(int a,int b)
    {
        i=a;
        j=b;
    }
    //i ile j'yi yazar.
    void show()
    {
        System.out.println("i and j:"+i+" "+j);
    }
}

class B extends A{
    int k;
    public B(int a, int b, int c)
    {
        super(a,b);
        k=c;
    }
    //k'yı göster - A içindeki show() metodunu örter.
    void show() {
        System.out.println("k:"+k);
    }
}
```

# Overriding

```
public class InterfaceOverriding {  
    public static void main(String[] args) {  
        B subOb=new B(1, 2, 3);  
        subOb.show();  
    }  
}
```

Output - InterfaceOverriding (run) X	
	run:
	i:3
	BUILD SUCCESSFUL (total time: 0 seconds)

# Overriding

Üst-sınıfta örtülen metodu çağırmak istersek, **super** anahtar sözcüğünü kullanabiliriz. Aşağıdaki program bunun nasıl yapılacağını göstermektedir

```
class B extends A{
    int k;
    public B(int a, int b, int c)
    {
        super(a,b);
        k=c;
    }
    //k'yı göster - A içindeki show() metodunu örter.
    void show(){
        System.out.println("k:"+k);
    }
}
```

# Overloading ve Overriding Farkı

- Bir sınıf içindeki iki metodun adları aynı olduğu halde formal parametreleri farklı ise bu iki metod aşkındır (overloaded). Aşkın metotlardan biri alt-sınıfta öteki üst sınıfta da olabilir.
- Overridden ve overloaded metotlar java'nın asıl gücünü ortaya koyan özelliklerdir.
  - Overloded metotların çözümlenmesi derlenme aşamasında,
  - Overridden metotların çözümlenmesi ise koşturma (runTime) aşamasında olur.
  - Bir tek metod adıyla, birden çok farklı metotları koşturma özeliğine, java'da çok biçimlilik (**polymorphism**) denir.

# Static

- Bir sınıfa ait anlık değişken tanımlandığında, o sınıfa ait her bir nesne içinde söz konusu değişkenin bir kopyası oluşur.
- Dolayısıyla, farklı nesnelerde o değişkene farklı değerler atanabilir. Aynı anda, o değişkenin yaratılan nesne sayısı kadar farklı değeri olabilir.
- Bazı durumlarda değişkenlerin veya metotların, onların sınıfına ait olarak yaratılacak nesnelerden bağımsız ve biricik olmalarını, farklı nesnelerde farklı işlevlere sahip kopyalarının yaratılmamasını isteriz.
  - Bunun için **static** kelimesi kullanılır.
  - Static damgalı değişkenlere ana bellekte ancak birer tane yer ayrılır, her nesne için ayrı yer ayrılmaz.
  - Bunun yerine, onu kullanacak her nesne, ona ayrılan tek adrese ulaşır.
  - Böylece, o sınıfa ait bir çok nesne yaratılsa bile, static değişkenin değerini ortak olarak kullanırlar.



# Static

- static nitelmesi metotlara ve değişkenlere verilebilir; sınıflara verilemez.
- static damgalı metot ancak static damgalı verilere (değişken, sabit) erişebilir.
- static damgalı metot, ancak static damgalı metotları çağırabilir.
- this ve super anahtar sözcüklerini kullanamazlar.

```
static int m= 5;  
static double b ;
```

```
static void renkSec (Color color) {  
    ....  
}
```

# Abstract

- Abstract belirtkisi soyut class ve/veya soyut metot yaratır.
- Soyut metotların adı, değişkenleri ve değer-tipi (return type) belirtilir, ama metot gövdesi yazılamaz.
- Abstract class en az bir tane abstract metoda sahip olmalıdır.
- Abstract class'tan nesne yaratılamaz (instantiate); çünkü içerdiği abstract metod çalıştırılamaz.
- Bir abstract class'ta bir metot varsa, o metot otomatik olarak abstract sayılır.
- Ama birden çok metot içerdiğinde, Bazıları abstract olmayabilir.

```
abstract void renk (Color color);
```

# Abstract

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);
```

# Abstract

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 metod olan her sınıf soyut bir sınıf olur. Soyut sınıfın gövdesinde constructor metodu, soyut metod(lar) ve gerekiyorsa somut metodlar yer alabilir. Sözdizimi şöyledir:

```
[Erişim_belirtkisi] abstract class sınıf_adı {  
  
    // soyut sınıf içinde en az bir soyut bir metod yer alınmalıdır  
    // soyut sınıf içinde somut metod olabilir  
    // constructor metodu tanımlanabilir  
}
```

# Abstract

```
// soyut sınıf
abstract class A {

    // soyut sınıf içinde en az bir soyut bir metod yer almalıdır
    abstract void soyMet();

    // soyut sınıf içinde somut metod olabilir
    void somMet() {
        System.out.println("Ben somut bir metodum.");
    }
}

class B extends A {
    void soyMet() {
        System.out.println("Sen soyut bir metotsun.");
    }
}

class UygulamaPrg {
    public static void main(String args[]) {
        B b = new B();

        b.soyMet();
        b.somMet();
    }
}
```

# Interface(Arayüz)

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 benzese de aralarında **önemli farklar** vardır.

# Interface(Arayüz)

- Arayüz ,interface anahtar sözcüğü ile tanımlanır. Arayüz abstract metotlar içerir.
- Arayüz, anlık (instance) değişkenler içeremez. Ancak, belirtkeleri 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 belirtkisi alabilir, başka erişim belirtkisi alamaz.
- public damgalı arayüz, public damgalı class gibidir. Her kod ona erişebilir.

# Interface(Arayüz)

- 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 belirtkisi 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.



# Interface(Arayüz)

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 özellik, Java'da çok biçimliliği olanaklı kılan önemli bir niteliktir.

# Niçin Interface kullanılır?

- Bir arayüzden bir sınıf uygulanırken, soyut (abstract ) metotların hepsini uygulamak için zorlarız.
- Arayüz, bir sınıf ne yapabilir ve aslında onu nasıl yapabilir ayırımına zorlar.
- Bir programcının, bir sınıfın fonksiyonlarını değiştirmeksizin, herhangi bir noktada yapılmış bir şeyi nasıl değiştirebileceğidir.
- Aslında çok biçimlilik özelliğinin uygulanmasıdır.

# Interface(Arayüz)

- Bir sınıf kalıtım özelliğini aldığı bir super sınıfa ait olabilir. Bir sınıf birden çok interface'i implement edebilir. Bir sınıf bir arayüzün kalıtımını alması için implements anahtar sözcüğü kullanılır.

Anahtar sözcük

```
public class className implements interfaceName{  
...class implementation...}
```

- Örneğin, SavingAccount sınıfı, Account ve Transactionlog arayüzlerini implement eder.

```
public class SavingsAccount implements Account, Transactionlog{  
...class implementation...}
```

# Interface(Arayüz)

```
[erişim_belirtkisi] 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 arayüz iki tane metot içermektedir.

```
public interface arayuz01{  
    public void topla(int x, int y);  
    public void hacim(int x, int y, int z);  
}
```

**Örnek 2:** Aşağıdaki arayüz iki tane sabit değişken içermektedir.

```
public interface arayuz02{  
    public static final double fiyat = 1250.00;  
    public static final int sayac = 5;  
}
```

# Interface(Arayüz)

Bir arayüzü deklare etmek için kullanılan anahtar sözcük class yerine interface'dir.

Class yerine kullanılan  
anahtar sözcük

```
public interface InterfaceBankAccount
{
    public final String bank= "JavaBank";
    public void deposit(int amt);
    public void withdraw(int amt);
    public int getbalance();
}
```

# Interface(Arayüz)

```
public interface InterfaceBankAccount
{
    public final String bank= "JavaBank";
    public void deposit(int amt);
    public void withdraw(int amt);
    public int getbalance();
}
```

InterfaceBankAccount arayüzünden genişletilen (extend) edilen sınıflar burada tanımlanan metodlar için çalışan metodları sağlamak zorunda olacak.

BankAccount arabirimini uygulayan sınıf

```
public class Account implements InterfaceBankAccount{
    private String bankname;
    public Account() {
        this.bankname = InterfaceBankAccount.bank; }
    public void deposit(int amt) { /* deposit code */ }
    public void withdraw(int amt) { /* withdraw code */ }
    public int getbalance() { /* getBalance code */ }
```

Bir arabirim uygulama sınıfları interface sabitlerine erişebilir.

# Interface(Arayüz)

**Final anahtar kelimesinin anlamı, bank değişkeni interface içinde bir sabittir çünkü siz sabitleri ve metodları sadece burada tanımlayabilirsiniz.**

```
this.bankname = InterfaceBankAccount.bank
```

**Başka bir değişkene bir statik değişkeni atarken kullandığınız aynı sözdizimi, bir arayüzden sabit atanırken kullanır.**



# Interface(Arayüz)

Interface, deposit, withdraw ve getbalance diye 3 metod tanımlar. Bunlar bizim sınıfımızda implement edilmelidir.

```
public class Account implements InterfaceBankAccount{  
    private String bankname;  
    int balance;  
    public Account() {  
        this.bankname = InterfaceBankAccount.bank; }  
    public void deposit(int amt) {  
        balance = balance + amt;  
    }  
    public void withdraw(int amt) {  
        balance = balance - amt  
    }  
    public int getbalance() {  
        return balance  
    }  
}
```



# GRAFİK SINIFLARI

## Klavye Hareketleri Olayları

### KeyEvent Sınıfının Yöntemleri

Yöntem	Açıklama
getKeyChar( )	Olaya sebep olan klavyenin tuşunu char tipinden verir.
getKeyCode( )	Olaya sebep olan klavyenin tuşunun integer olarak kodunu verir.

Klavyenin tuşları KeyEvent sınıfında birer sabit olarak tanımlanmıştır.

VK\_ENTER, VK\_UP, VK\_CONTROL

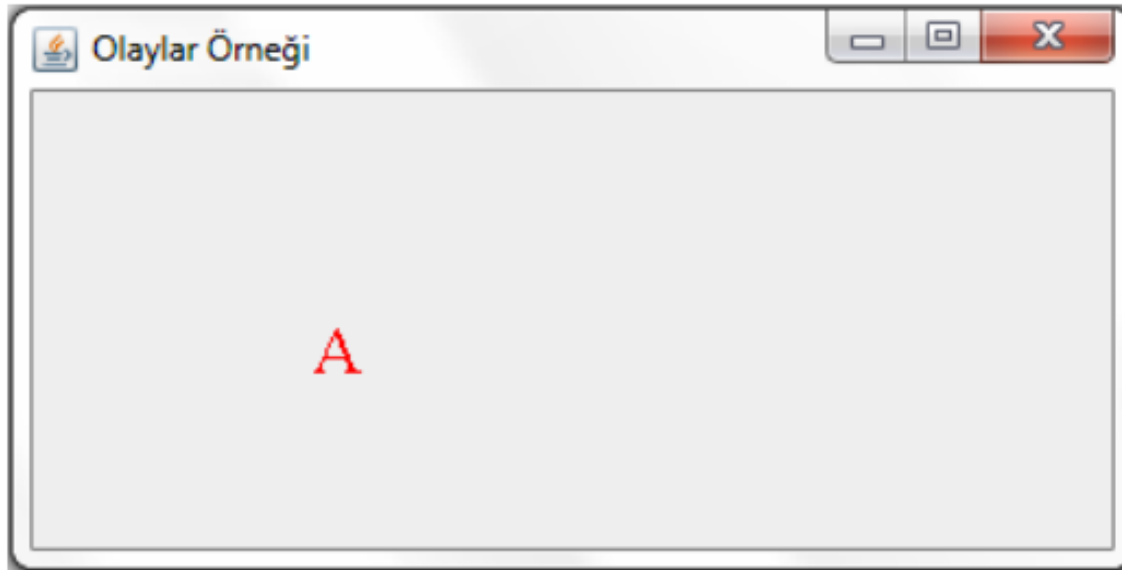
InputEvent sınıfındaki yöntemler **KeyEvent**'te de kullanılabilir.

# GRAFİK SINIFLARI

## KeyListener Arabiriminin Yöntemleri

Yöntem	Açıklama
keyPressed( )	Kaynak komponent üzerinde iken herhangi bir tuşa basıldığında çalışır.
keyReleased( )	Kaynak komponent üzerinde iken herhangi bir tuş serbest bırakıldığında çalışır.
keyTyped( )	Kaynak komponent üzerinde iken herhangi bir tuşa basılıp serbest bırakıldığında çalışır.

**Örnek:** Ekrana bir yazı yazılacak ve klavyenin yön tuşlarına basılarak yazı hareket edecektir.



# GRAFİK SINIFLARI(Klavye Olayları)

## KlavyePaneli.java

```
package olaylarprojesi;
import java.awt.*;
import java.awt.event.*;
import javax.swing.JPanel;
public class KlavyePaneli extends JPanel{
    private int x=100;
    private int y=100;
    private char harf='A';
    public KlavyePaneli() {
        KlavyeTusuOlayDinleyici dinleyici=new KlavyeTusuOlayDinleyici();
        addKeyListener(dinleyici);
    }
```

# GRAFİK SINIFLARI(Klavye Olayları)

```
protected void paintComponent(Graphics g){
    super.paintComponent(g);
    g.setColor(Color.RED);
    g.setFont(new Font("Times New Roman",Font.PLAIN,24));
    g.drawString(String.valueOf(harf),x,y);
}
private class KlavyeTusuOlayDinleyici extends KeyAdapter{
    public void keyPressed(KeyEvent e){
        switch(e.getKeyCode()){
            case KeyEvent.VK_DOWN: y+=10;break;
            case KeyEvent.VK_UP: y-=10;break;
            case KeyEvent.VK_LEFT: x-=10;break;
            case KeyEvent.VK_RIGHT: x+=10;break;
            default:harf=e.getKeyChar();}
        repaint();
    }
}
```

# GRAFİK SINIFLARI(Klavye Olayları)

## OlayPenceresi5.java

```
package olaylarprojesi;
import java.awt.BorderLayout;
import javax.swing.*.*;
public class OlayPenceresi5 extends JFrame {
    KlavyePaneli panel=new KlavyePaneli();
    public OlayPenceresi5() {
        setLayout(new BorderLayout());
        add(panel,BorderLayout.CENTER);
        panel.setFocusable(true);
    }
}
```

**panel.setFocusable(true);** Panelin pencere içinde odaklanmasını sağlar. Odaklanmamış komponentlerde klavye ile ilgili yöntemler çalışmaz.

# GRAFİK SINIFLARI(Klavye Olayları)

## Main.java

```
package olaylarprojesi;
import javax.swing.JFrame;
public class Main {
    public static void main(String[] args) {
        OlayPenceresi5 pencere= new OlayPenceresi5();
        pencere.setSize(400,200);
        pencere.setTitle("Olaylar Örneği");
        pencere.setLocationRelativeTo(null);
        pencere.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pencere.setVisible(true);
    }
}
```

# GRAFİK SINIFLARI(Buton Sınıfı)

## Button Sınıfının Özellikleri

Özellik	Açıklama
actionCommand	Butonun aksiyon komutudur.
text	Buton üzerinde görünen yazıdır.
icon	Buton üzerinde görünen ikondur.
pressedIcon	Butona basıldığında görünen ikondur.
rolloverIcon	Fare butonun üzerinde olduğunda görünen ikondur.
Mnemonic	Butonun klavyedeki mnemonic tuşunun değeridir. Alt tuşuna ve mnemonic tuşa basılarak buton seçili duruma getirilir.
horizontalAlignment	Butonun üzerindeki yazının veya ikonun yataydaki hizasıdır. Default'u center'dır.
horizontalTextPosition	Butonun üzerindeki yazının ikona göre yataydaki pozisyonudur. Default'u right'dır.



# GRAFİK SINIFLARI(Buton Sınıfı)

## Button Sınıfının Özellikleri

Özellik	Açıklama
verticalAlignment	Butonun üzerindeki yazının veya ikonun dikeydeki hizasıdır. Default'u center'dır.
verticalTextPosition	Butonun üzerindeki yazının ikona göre dikeydeki pozisyonudur. Default'u center'dır.
borderPainted	Butonun kenarlarının çizili olup olmadığını tutan boolean data alanıdır.
iconTextGap	Butonun üzerindeki yazı ve ikon arasındaki boşluğun pixel olarak değeridir.
selected	RadioButton ve CheckBox'ın seçili olup olmadığı bilgisini tutan boolean bir data alanıdır.

# GRAFİK SINIFLARI(Buton Sınıfı)

## Button Yapıcıları

Yapıcı
JButton( )
JButton(Icon)
JButton(String)
JButton(String, Icon)

# Fare Hareketleri Olayları

## MouseEvent Sınıfının Yöntemleri

Yöntem	Açıklama
getButton( )	Farenin hangi butonuna tıklandığı bilgisini verir.
getClickCount( )	Fare tıklamalarının sayısını verir.
getPoint( )	Farenin bulunduğu noktanın x ve y koordinatlarını Point nesnesi şeklinde verir.
getX( )	Farenin bulunduğu noktanın x koordinatını verir.
getY( )	Farenin bulunduğu noktanın y koordinatını verir.

MouseEvent türediği InputEvent sınıfının yöntemlerini de miras alır.

### InputEvent Sınıfının Yöntemleri

Yöntem	Açıklama
getWhen( )	Olayın meydana geldiği zamanı verir.
isAltDown( )	Olay sırasında klavyedeki Alt tuşuna basılı ise True değeri döner.
isControlDown( )	Olay sırasında klavyedeki Control tuşuna basılı ise True değeri döner.
isMetaDown( )	Olay sırasında klavyedeki Meta tuşuna basılı ise True değeri döner.
isShiftDown( )	Olay sırasında klavyedeki Shift tuşuna basılı ise True değeri döner.

## MouseListener Arabiriminin Yöntemleri

Yöntem	Açıklama
mousePressed( )	Kaynak komponent üzerinde fareye basıldığında çalıştırılır.
mouseReleased( )	Kaynak komponent üzerinde fare serbest bırakıldığında çalıştırılır.
mouseClicked( )	Kaynak komponent üzerinde fareye tıklandığında çalıştırılır.
mouseEntered( )	Fare kaynak komponente girdiğinde çalıştırılır.
mouseExited( )	Fare kaynak komponentten çıktıktan sonra çalıştırılır.

# Kaynaklar

- Java ve Java Teknolojileri, *Tevfik KIZILÖREN* – Kodlab Yayınları
- Yazılım Mühendisliği CBU-Dr. Öğr. Üyesi Deniz Kılınç Yazılım Mimarisi ve Tasarımı Ders Notları
- Dr. Öğr. Gör. Aysun Zehra ALTIKARDEŞ, Marmara Üniversitesi Bilg. Prog. NESNE YÖNELİMLİ PROGRAMLAMA 2 DERSİ NOTLARI