

# CENG 217 NESNE TABANLI DİZAYN

ders 4

**Doç. Dr. Halûk Gümüşkaya**

haluk@gumuskaya.com / haluk@fatih.edu.tr

<http://www.gumuskaya.com>

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**  
**Fatih Üniversitesi**



Thursday, November 24, 2005

# ARABİRİM VE ÇOK ŞEKİLLİLİK



Object-Oriented  
Design & Patterns

SECOND EDITION

CAY HORSTMANN

# Ders taslađı

## 1. Icon arabirim tipi

2. Çok biçimlilik

3. Karşılaştırılabilir( Comparable) arabirim tipi

4. Karşılaştırıcı(Comparator) arabirim tipi

5. Anonim sınıflar

6. Çerçevesel ve kullanıcı arabirim tamamlayıcısı

7. Kullanıcı Arabirim Hareketleri

8. Zamanlayıcılar (Timers)

9. Şekille çizme

10. Bir arabirim tipini dizayn etme

# Bir resmi gösterme

- ◆ Mesajı göstermek için `JOptionPane`'i kullanılır:

```
JOptionPane.showMessageDialog(null, "Hello, World!");
```

- ◆ İcon'un notu sol tarafta



# Bir resmi gösterme

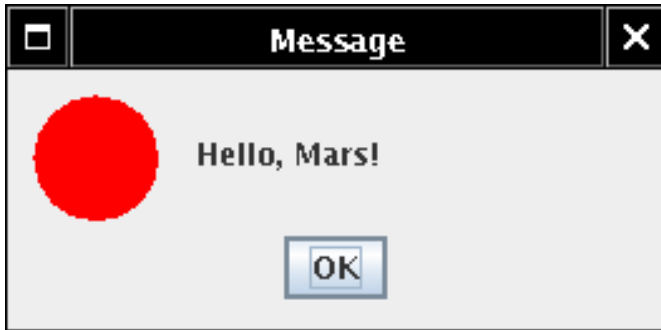
- ◆ Keyfi bir resim dosyası belirlenebilir

```
JOptionPane.showMessageDialog( null,  
    "Hello, World!",  
    "Mesaj",  
    JOptionPane.INFORMATION_MESSAGE,  
    new ImageIcon("globe.gif"));
```



# Bir resmi gösterme

- ◆ Bir resim (image)dosyası oluşturmak istemezsen ne olur?
- ◆ interface(arabirim) tipini implements eden herhangi bir sınıf kullanılabilir.
- ◆ **ImageIcon** bu sınıflardan biridir
- ◆ Kendi sınıfını oluşturmak kolaydır



# Icon arabirim tipi

- ◆ Icon arabirim tipinin tanımı:

```
public interface Icon
{
    int getIconWidth();
    int getIconHeight();
    void paintIcon(Component c, Graphics g, int x, int y)
}
```

## Arabirim tipler

- ◆ Bir arabirim tipinin bir uygulaması yoktur
- ◆ Arabirim tipindeki methodlar public olarak ifade edilmez—arabirim tipinin tümü otomatik olarak public'tir
- ◆ inherit eden sınıf bütün methodların uygulamasını bulundurmak zorundadır
- ◆ Ch4/icon2/MarsIcon.java
- ◆ `showMessageDialog` Icon objesini bekler(parametre olarak alır.)
- ◆ Ok to pass `MarsIcon`
- ◆ Ch4/icon2/IconTester.java

# MarsIcon.java ve IconTester.java

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * mars gezegeninin şekline sahip olan icon
 */
public class MarsIcon implements Icon
{
    /**
     * Constructs a Mars icon of a given size.
     * @param aSize the size of the icon
     */
    public MarsIcon(int aSize) { size = aSize; }

    public int getIconWidth() { return size; }

    public int getIconHeight() { return size; }

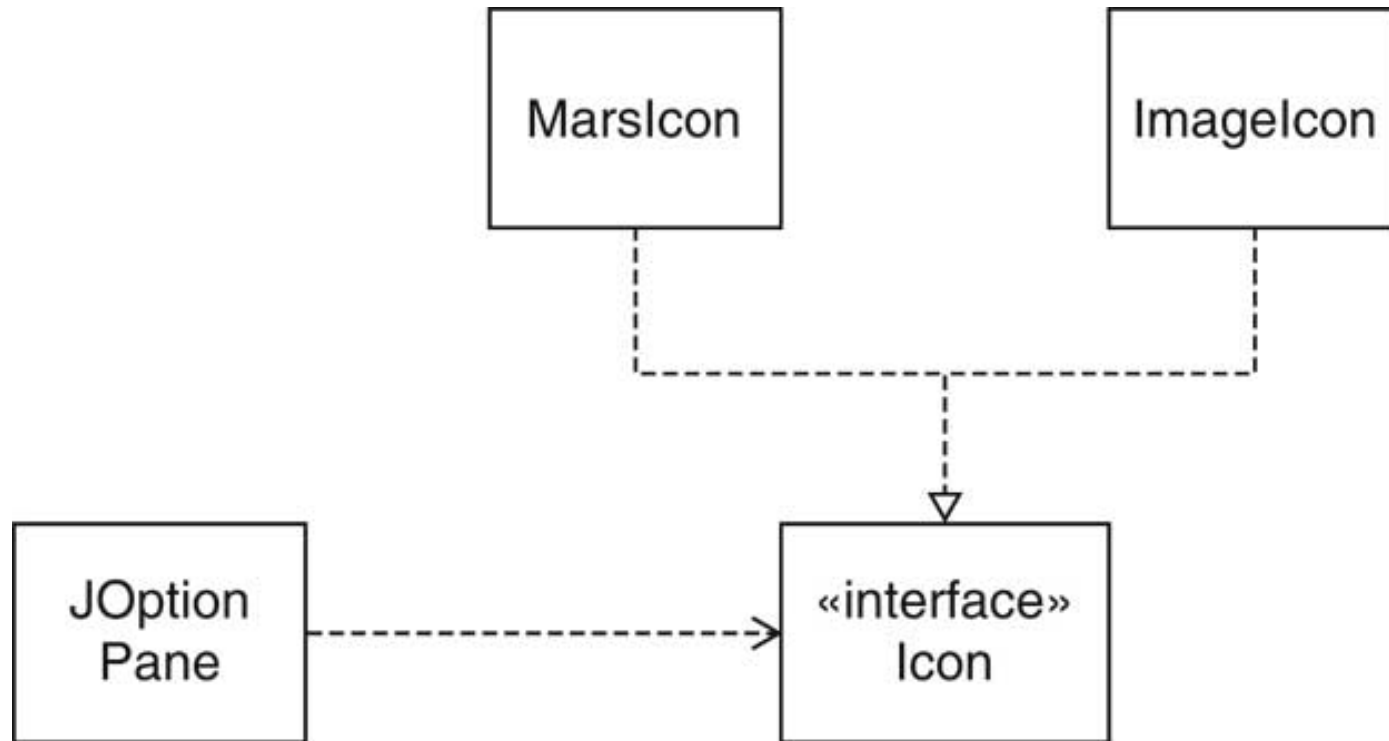
    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        Graphics2D g2 = (Graphics2D) g;
        Ellipse2D.Double planet = new Ellipse2D.Double(x, y,
            size, size);
        g2.setColor(Color.RED);
        g2.fill(planet);
    }

    private int size;
}
```

```
import javax.swing.*;

public class IconTester
{
    public static void main(String[] args)
    {
        JOptionPane.showMessageDialog(
            null,
            "Hello, Mars!",
            "Message",
            JOptionPane.INFORMATION_MESSAGE,
            new MarsIcon(50));
        System.exit(0);
    }
}
```

# Icon arabirim tipi ve sınıflara uygulama



# Ders taslağı

---

1. Icon arabirim tipi
- 2. Çok biçimlilik**
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
6. Çerçeveler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

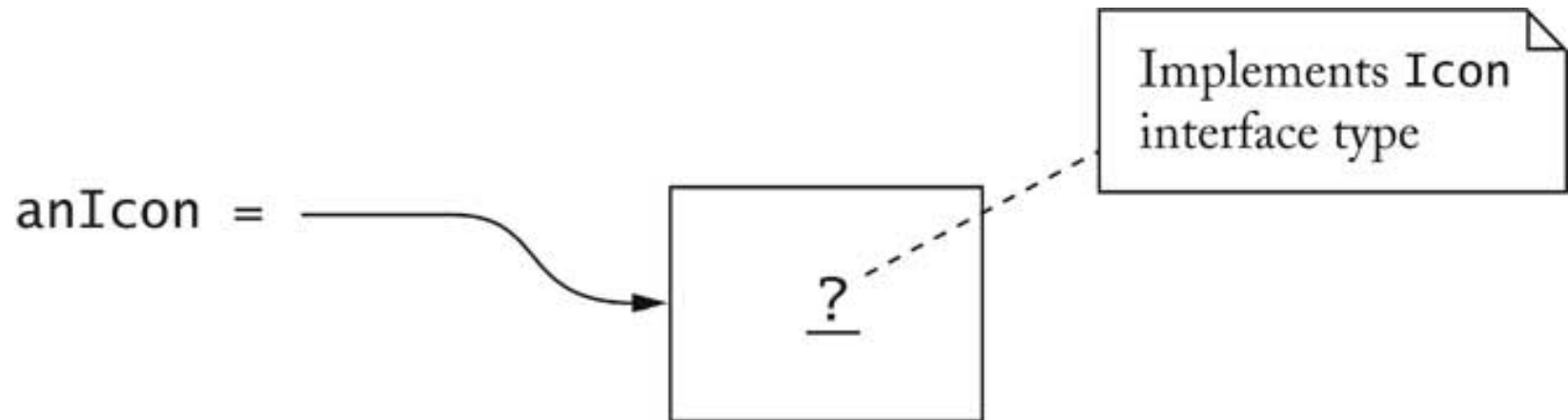
# Çok biçimlilik

- ◆ `public static void showMessageDialog(...Icon anIcon)`
- ◆ `showMessageDialog` aşağıdakileri gösterir
  - icon
  - mesaj
  - OK butonu
- ◆ `showMessageDialog` dialogun boyutunu hesaplamalı
- ◆  $\text{genişlik} = \text{icon genişlik} + \text{mesaj boyutu} + \text{boşluk boyutu}$
- ◆ Bir icon'un genişliğini nasıl bilebiliriz?
- ◆ `int genişlik(width) = anIcon.getIconWidth();`

# Çok biçimlilik

- ◆ `showMessageDialog` hangi `icon`'un kullanıldığını bilemez
  - `ImageIcon`?
  - `MarsIcon`?
  - ...?
- ◆ `anIcon`'un tipi `Icon` değil
- ◆ `Icon` tipinde bir nesne yoktur
- ◆ `anIcon` bir sınıftır ve bu sınıf `Icon` dan implement eder.
- ◆ bu sınıf `getIconWidth` methodunu tanımlar

# Arabirim(interface) tipinin bir değişkeni



# Çok biçimlilik

hangi getWidth methodu çağırılır?

- ◆ bunlar olabilir
  - `MarsIcon.getWidth`
  - `ImageIcon.getWidth`
  - .....
- ◆ anIcon referansının hangi nesneye işaret ettiğine bağlıdır, e.g. `showMessageDialog(..., new MarsIcon(50))`
- ◆ **Çok biçimlilik(polimorphism)**: asıl nesne tipine göre farklı methodlar seç
- ◆ “**polymorphic**” terimi “çok şekilli” demektir

# Çok biçimlilik

---

## ◆ Loose coupling – gevşek bağımlılık(coupling)

- `showMessageDialog ImageIcon'` dan eklenir
- image processing hakkında bilgiye gerek yok

## ◆ Genişletilebilir

- Client(müşteri) yeni icon'lara ihtiyaç duyabilir

# Ders taslađı

---

1. Icon arabirim tipi
2. Çok biçimlilik
- 3. Karşılaştırılabilir( Comparable) arabirim tipi**
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
6. Çerçeveseler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

# Comparable arabirim tipi

- ◆ Koleksiyonlar (collections) çeşitli gruplama methodlarına sahiptir:

```
ArrayList<E> a = . . .  
Collections.sort(a);
```

- ◆ Listedeki nesnelere **Comparable** arabirimini implement etmek zorundadır.

```
public interface Comparable<T>  
{  
    int compareTo(T other);  
}
```

- ◆ Arabirim parametrelendirilir ( **ArrayList** gibi)
- ◆ Parametre tipi other'in tipidir

# Comparable arabirim tipi

- ◆ `object1.compareTo(object2)`
  - Object1 object2'den az ise negatif sayıya
  - Objectler aynı ise 0'a
  - Object1 object2'den büyük ise pozitif sayıya return eder
- ◆ `sort` methodu elementleri karşılaştırır ve tekrar düzenler
- ◆ `if (object1.compareTo(object2) > 0) . . .`
- ◆ `String` sınıfı `Comparable<String>` interface tipini implement eder: lexicographic (sözlük) sırasını
- ◆ **Example: Country (ülke)** sınıfı: ülkeleri alanla karşılaştırır
- ◆ `Ch4/sort1/Country.java`
- ◆ `Ch4/sort1/CountrySortTester.java`

# Country.java and CountrySortTester.java

```
/**
 * A country with a name and area.
 */
public class Country implements Comparable<Country>
{
    /**
     * Constructs a country.
     * @param aName the name of the country
     * @param anArea the area of the country
     */
    public Country(String aName, double anArea)
    { name = aName;
      area = anArea;
    }

    public String getName() { return name; }
    public double getArea() { return area; }

    /**
     * Compares two countries by area.
     * @param other the other country
     * @return a negative number if this country
     *         has a smaller area than otherCountry,
     *         0 if the areas are the same, a positive number otherwise
     */
    public int compareTo(Country other)
    {
        if (area < other.area) return -1;
        if (area > other.area) return 1;
        return 0;
    }
    private String name;
    private double area;
}
```

```
import java.util.*;

public class CountrySortTester
{
    public static void main(String[] args)
    {
        ArrayList<Country> countries = new ArrayList<Country>();
        countries.add(new Country("Uruguay", 176220));
        countries.add(new Country("Thailand", 514000));
        countries.add(new Country("Belgium", 30510));

        Collections.sort(countries);
        // Now the array list is sorted by area
        for (Country c : countries)
            System.out.println(c.getName() + " " + c.getArea());
    }
}
```

# Ders taslađı

---

1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
- 4. Karşılaştırıcı(Comparator) arabirim tipi**
5. Anonim sınıflar
6. Çerçeveseler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

# The Comparator interface type

- ◆ Ülkeleri isimlerine göre nasıl gruplandırırız?
- ◆ `Comparable` iki kez implement edilmez!
- ◆ `Comparator` interface(arabirim) tipi fazladan esneklik verir

```
public interface Comparator<T>
{
    int compare(T obj1, T obj2);
}
```

- ◆ Gruplamak için comparator nesnesini geçir:  
`Collections.sort(list, comp);`

# Comparator interface tipi

---

- ◆ Ch4/sort2/CountryComparatorByName.java
- ◆ Ch4/sort2/ComparatorTester.java
  
- ◆ Comparator nesnesi bir fonksiyon nesnesidir
- ◆ Bu belirli comparator nesnesi state'e(durum) sahip değildir
- ◆ State(durum) kullanışlı olabilir.küçükten büyüğe ve büyükten küçüğe gruplama

# CountryComparatorByName.java and ComparatorTester.java

```
import java.util.*;

public class CountryComparatorByName implements Comparator<Country>
{
    public int compare(Country country1, Country country2)
    {
        return country1.getName().compareTo(country2.getName());
    }
}
```

```
import java.util.*;

public class ComparatorTester
{
    public static void main(String[] args)
    {
        ArrayList<Country> countries = new ArrayList<Country>();

        countries.add(new Country("Uruguay", 176220));
        countries.add(new Country("Thailand", 514000));
        countries.add(new Country("Belgium", 30510));

        Comparator<Country> comp = new CountryComparatorByName();
        Collections.sort(countries, comp);

        // Now the array list is sorted by area
        for (Country c : countries)
            System.out.println(c.getName() + " " + c.getArea());
    }
}
```

# Ders taslağı

---

1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
- 5. Anonim sınıflar**
6. Çerçeveler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

# Anonim sınıflar

- ◆ Sadece bir kez kullanılan nesnelere isim vermeye gerek yoktur

```
Collections.sort(countries,  
                new CountryComparatorByName());
```

- ◆ Sadece bir kez kullanılan sınıflara isim vermeye gerek yoktur

```
Comparator<Country> comp = new  
    Comparator<Country>() // Make object of anonymous class  
    {  
        public int compare(Country country1, Country country2)  
        {  
            return country1.getName().compareTo(country2.getName());  
        }  
    }  
};
```

- ◆ anonim **new** açıklaması(expression):

- Comparator'u implement eden anonim sınıfı tanımlar
- Bu sınıfın compare methodunu tanımlar
- Bu sınıfın bir nesnesini yapar

# Anonim sınıfları öğrenme

- ◆ Kodu tekrar yazma ve açık olarak bir sınıf ismini tanıtma:

```
class MyComparator implements Comparator<Country> // give a name to the class
{
    public int compare(Country country1, Country country2)
    {
        return country1.getName().compareTo(country2.getName());
    }
}
Comparator<Country> comp = new MyComparator();
```

- ◆ Tecrube kazandıktan sonra, onlar senin için oldukça doğal olacak ve sen kodları tekrar yazmak zorunda olmadığını keşfedeceksin

# Anonim sınıflar

- ◆ Ortak olarak kullanılan methodlar:

```
public static Comparator<Country> comparatorByName()  
{  
    return new Comparator<Country>()  
    {  
        public int compare(Country country1, Country country2)  
        {  
            ...  
        }  
    };  
}
```

- ◆ `Collections.sort(a, Country.comparatorByName())` ;
- ◆ Çok comparator anlamı gelirse (by name, by area...) düzenle

# Ders taslağı

---

1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
- 6. Çerçevesel ve kullanıcı arabirim tamamlayıcısı**
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

# çerçeveler

---

- ◆ Çerçeve penceresi dacrasyonlara sahiptir
  - ◆ Başlık kalıbı
  - ◆ Kapama kutusu
  - ◆ Sistemi pencereleyerek sağlanan
- ◆ 

```
JFrame frame = new JFrame();  
frame.pack();  
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
frame.setVisible(true);
```
- ◆ ...

# Birleşen ekleme

- ◆ Birleşenleri oluşturma

```
 JButton helloButton = new JButton("merhaba de ");
```

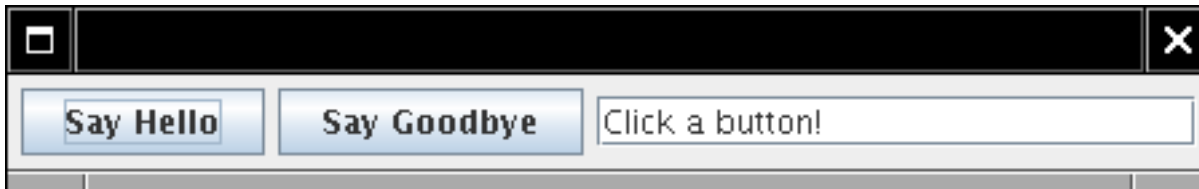
- ◆ Çerçevenin kalıbını oluştur

```
 frame.setLayout(new FlowLayout());
```

- ◆ Çerçeveye bileşen ekle

```
 frame.add(helloButton);
```

- ◆ Ch4/frame/FrameTester.java



# FrameTester.java

```
import java.awt.*;
import javax.swing.*;

public class FrameTester
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        JButton helloButton = new JButton("Say Hello");
        JButton goodbyeButton = new JButton("Say Goodbye");

        final int FIELD_WIDTH = 20;
        JTextField textField = new JTextField(FIELD_WIDTH);
        textField.setText("Click a button!");

        frame.setLayout(new FlowLayout());

        frame.add(helloButton);
        frame.add(goodbyeButton);
        frame.add(textField);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```

# Ders taslağı

---

1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
6. Çerçeveler ve kullanıcı arabirim tamamlayıcısı
- 7. Kullanıcı Arabirim Hareketleri**
8. Zamanlayıcılar (Timers)
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

# Kullanıcı arabirim faaliyetleri

- ◆ Previous program's buttons don't have any effect
- ◆ düğmeye *listener object(s)* 'i ekle
- ◆ **ActionListener** interface(arabirim) tipini implement eden sınıfa aittir.

```
public interface ActionListener
{
    int actionPerformed(ActionEvent event) ;
}
```

- ◆ Düğmeye basıldığı zaman listener bildirilir

# Kullanıcı arabirim faaliyetleri

- ◆ **actionPerformed** methodunun içine faaliyet kodu ekle
- ◆ Rutin kodun üzerinde açıkla

```
helloButton.addActionListener(new  
    ActionListener()  
    {  
        public void actionPerformed(ActionEvent event)  
        { textField.setText("Hello, World");  
        }  
    });
```

- ◆ Düğmeye basıldığında , text field(yazı alanı) oluşturulur

# Enclosing Scope'dan değişkenlere ulaşma

- ◆ dikkat: daha içteki sınıf enclosing scope'dan değişkenlere ulaşabilir.örneğin text field
- ◆ enclosing instance fields, local değişkenlere ulaşılabilir
- ◆ If daha içteki bir sınıf enclosing scope'dan değişkenlere ulaşabiliyorsa,bu değişken **Final** olarak declare edilmiş olmalı  
`final JTextField textField = ...;`
- ◆ **final** bir değişkenin yaşadığı sürece aynı nesneyi gösterdiği gerçeğini belirtir

# Kullanıcı arbirim faaliyetleri

---

- ◆ Constructor listener'ı ekler:

```
helloButton.addActionListener(listener);
```

- ◆ Düğmeler tüm listener'leri hatırlar

- ◆ Düğmeye basıldığında ,düğme listener'a bildirir

```
listener.actionPerformed(event);
```

- ◆ Listener textfield'in yazısını(text) değiştirir

```
textField.setText("Hello, World!");
```

- ◆ Ch4/action1/ActionTester.java

# ActionTester.java (1)

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ActionTester
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        final int FIELD_WIDTH = 20;
        final JTextField textField = new JTextField(FIELD_WIDTH);
        textField.setText("Click a button!");

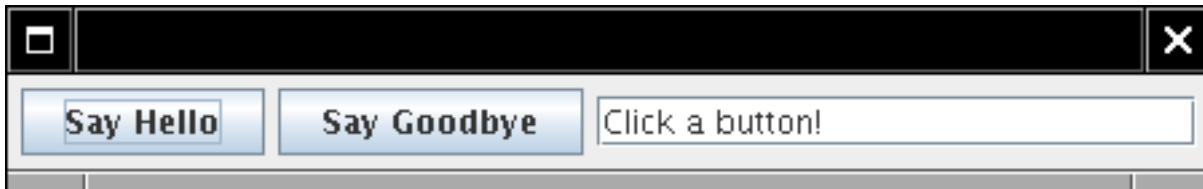
        JButton helloButton = new JButton("Say Hello");

        helloButton.addActionListener(new
            ActionListener()
            {
                public void actionPerformed(ActionEvent event)
                {
                    textField.setText("Hello, World!");
                }
            }
        );

        .....
    }
}
```

# ActionTester.java (2)

```
    JButton goodbyeButton = new JButton("Say Goodbye");  
    goodbyeButton.addActionListener(new  
        ActionListener()  
        {  
            public void actionPerformed(ActionEvent event)  
            {  
                textField.setText("Goodbye, World!");  
            }  
        }  
    );  
    frame.setLayout(new FlowLayout());  
  
    frame.add(helloButton);  
    frame.add(goodbyeButton);  
    frame.add(textField);  
  
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
    frame.pack();  
    frame.setVisible(true);  
}
```



# İlgili faaliyetleri oluşturma

- ◆ When benzer faaliyetli çeşitli action listener olduğunda, nesneleri oluşturan yardım edici method oluştur.
- ◆ Parametre olarak değişken bilgilerini geçir
- ◆ Parametreleri final olarak declare et.

```
public static ActionListener createGreetingButtonListener( final String message)
{   return new
    ActionListener()
    {   public void actionPerformed(ActionEvent event)
        {
            textField.setText(message);
        }
    };
}
```

# ActionTester2.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class ActionTester2
{
    public static void main(String[] args)
    {
        .....
        final int FIELD_WIDTH = 20;
        final JTextField textField = new JTextField(FIELD_WIDTH);

        helloButton.addActionListener( createGreetingButtonListener("Hello World!"));
        goodbyeButton.addActionListener( createGreetingButtonListener("Goodbye World!"));

        public static ActionListener createGreetingButtonListener( final String message)
        {
            return new
                ActionListener()
                {
                    public void actionPerformed(ActionEvent event)
                    {
                        textField.setText(message);
                    }
                };
        }

        private static JTextField textField;
    }

    .....
}
```

# Ders taslađı

---

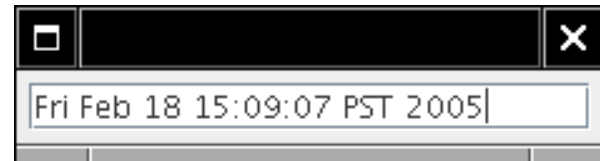
1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
6. Çerçeveseler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
- 8. Zamanlayıcılar (Timers)**
9. Şekille çizme
10. Bir arabirim tipini dizayn etme

# Timers(zamanlayıcı)

- ◆ `javax.swing` 'deki **Timer** sınıfı eşit zaman aralıklarında oluşan faaliyet olaylarının sırasını bildirir ve dizayn edilmiş action listener'ı bildirir.
- ◆ Timer'ı oluşturmak için olaylar arasındaki delay (erteleme) ve action listenerı temin et

```
ActionListener listener = ...;  
final int DELAY = 1000; // 1000 millisec = 1 sec  
Timer t = new Timer(DELAY, listener);  
t.start();
```

- ◆ Delay geçerken action listener çağırılır
- ◆ Ch4/timer/TimerTester.java



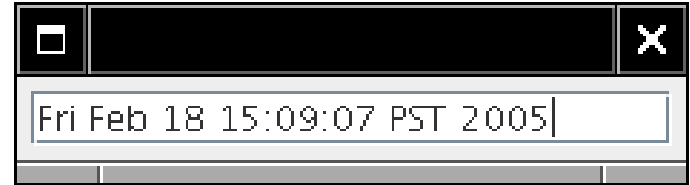
# TimerTesters.java

```
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.swing.*;
import javax.swing.Timer;

// This program shows a clock that is updated once per second.
public class TimerTester
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        final int FIELD_WIDTH = 20;
        final JTextField textField = new JTextField(FIELD_WIDTH);
        frame.setLayout(new FlowLayout());
        frame.add(textField);

        ActionListener listener = new
            ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                Date now = new Date();
                textField.setText(now.toString());
            }
        };

        final int DELAY = 1000; // Milliseconds between timer ticks
        Timer t = new Timer(DELAY, listener);
        t.start();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}
```



# Ders taslađı

---

1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
6. Çerçeveseler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
- 9. Şekille çizme**
10. Bir arabirim tipini dizayn etme

# Şekilleri çizme

- ◆ `paintIcon` methodu `Graphics` tipinde ***graphics context'ini alır***
- ◆ Actually a `Graphics2D` object in modern Java versions

```
public void paintIcon(Component c, Graphics g, int x, int y)
{
    Graphics2D g2 = (Graphics2D) g;
    ...
}
```

- ◆ `Shape` interface'ini implement eden herhangi bir nesne çizebiliriz

```
Shape s = ...;
g2.draw(s);
```

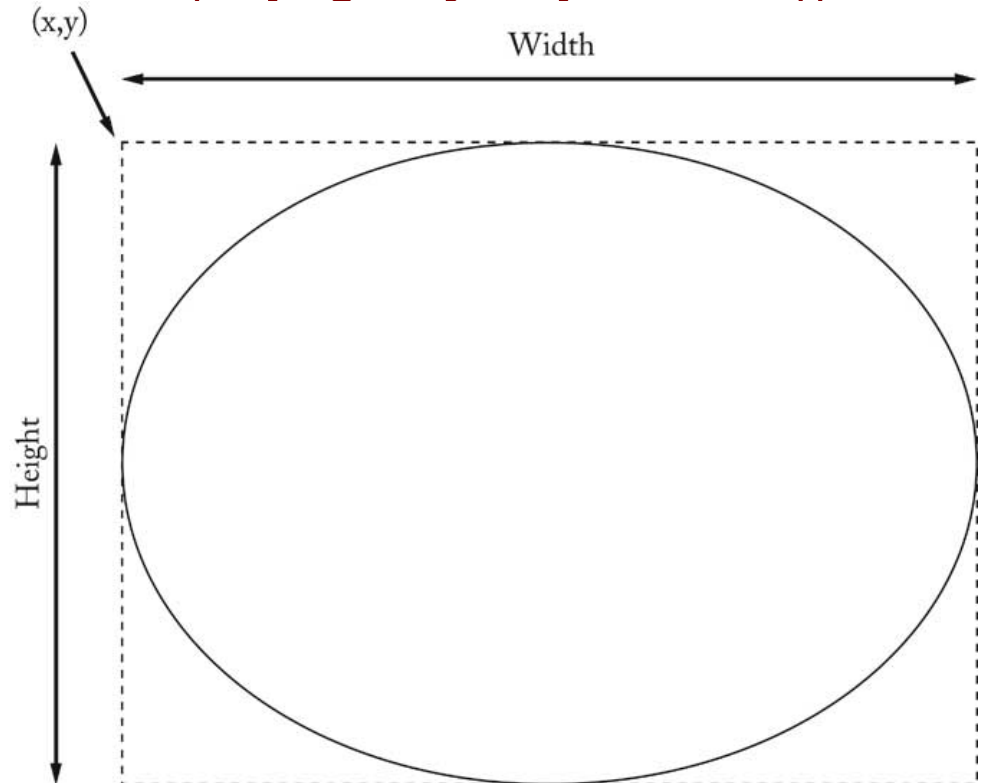
# Elips ve dikdörtgen çizme

## ◆ `Rectangle2D.Double`

- Üst sol köşe
- genişlik
- yükseklik oluşturulur

## ◆ `g2.draw(new Rectangle2D.Double(x, y, genişlik, yükseklik));`

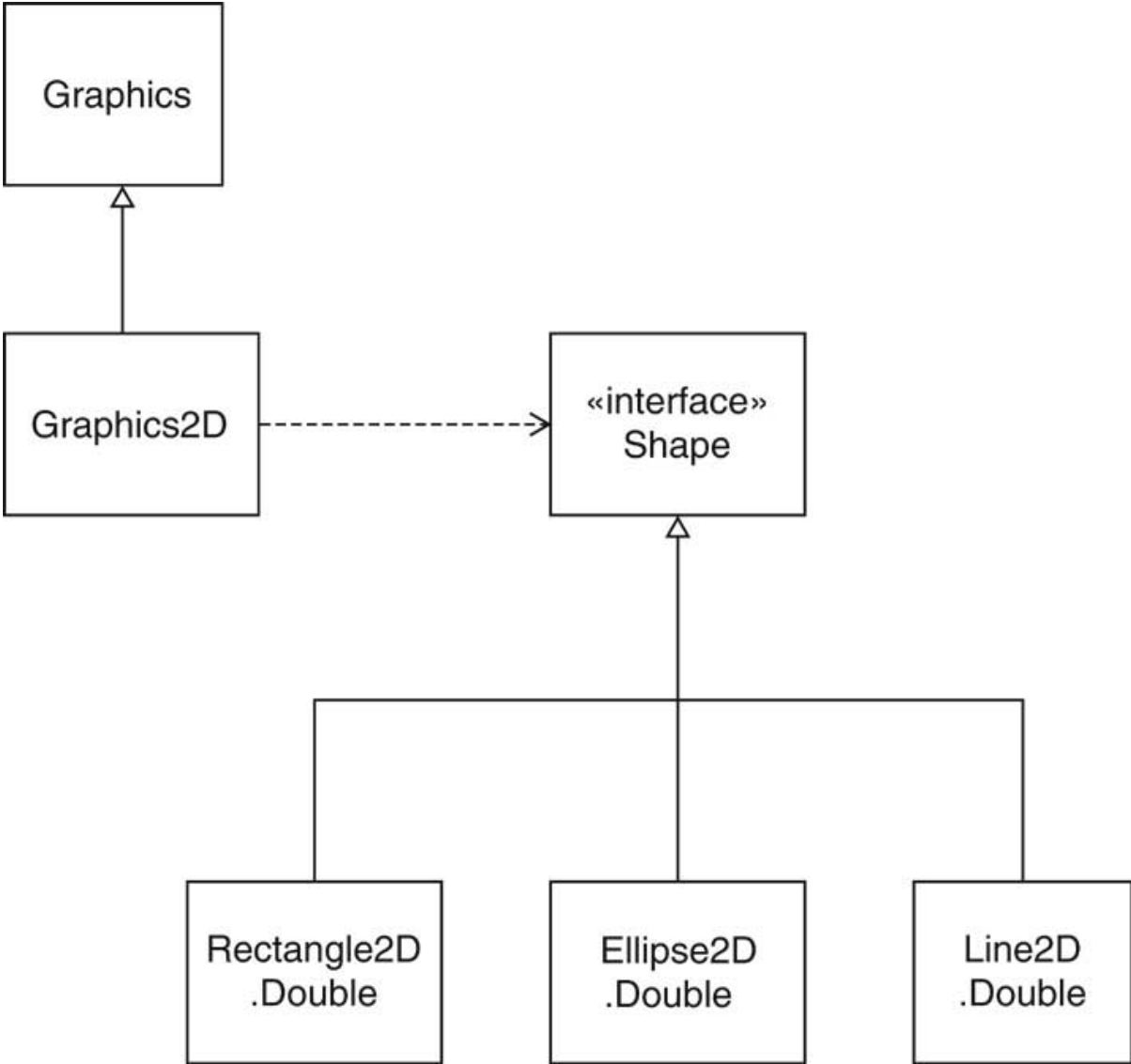
## ◆ `Ellipse2D.Double`, in kutu sınırlarını belirle



# Çizgi çizme

- ◆ **Point2D.Double** uzayda bir noktadır
- ◆ **Line2D.Double** noktaları birleştirir
  
- ◆ `Point2D.Double baş = new Point2D.Double(x1, y1);`  
`Point2D.Double son = new Point2D.Double(x2, y2);`  
`Shape bölüm = new Line2D.Double(baş, son);`  
`g2.draw(bölüm);`

# Shape(şekil) sınıfları arasındaki ilişki



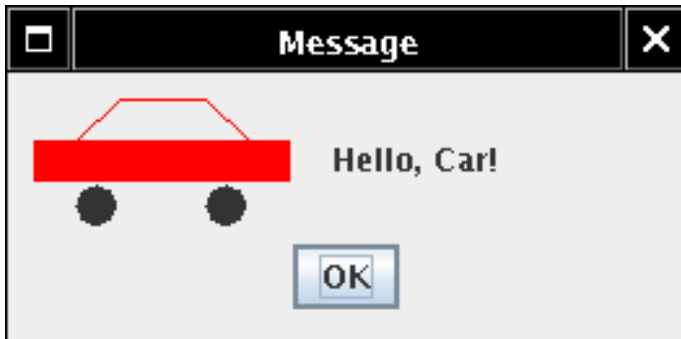
# Yazı çizme

- ◆ `g2.drawString(text, x, y);`
- ◆ `x, y` temel nokta koordinatlarıdır



# Şekilleri doldurma

- ◆ Şekillerin içini doldurur  
`g2.fill(shape) ;`
- ◆ Renkleri değiştirir:  
`g2.setColor(Color.red) ;`
- ◆ Araba çizen program
- ◆ Ch4/icon3/Carlcon.java



# Carlcon.java (1)

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.*;

/**
 * An icon that has the shape of a car.
 */
public class Carlcon implements Icon
{
    /**
     * Constructs a car of a given width.
     * @param width the width of the car
     */
    public Carlcon(int aWidth)
    {
        width = aWidth;
    }

    public int getIconWidth()
    {
        return width;
    }

    public int getIconHeight()
    {
        return width / 2;
    }

    .....
}
```

# Carlcon.java (2)

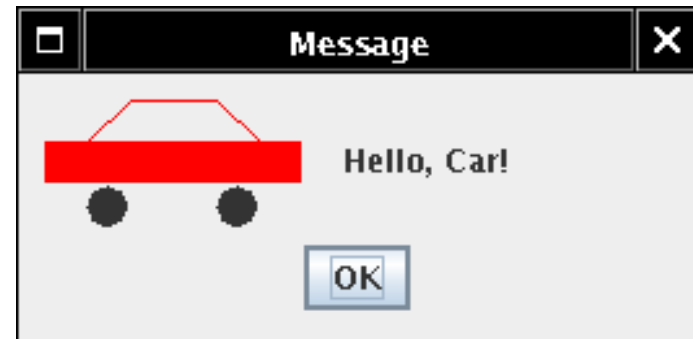
```
public void paintIcon(Component c, Graphics g, int x, int y)
{
    Graphics2D g2 = (Graphics2D) g;
    Rectangle2D.Double body = new Rectangle2D.Double(x, y + width / 6, width - 1, width / 6);
    Ellipse2D.Double frontTire = new Ellipse2D.Double(x + width / 6, y + width / 3, width / 6, width / 6);
    Ellipse2D.Double rearTire = new Ellipse2D.Double(x + width * 2 / 3, y + width / 3, width / 6, width / 6);

    // The bottom of the front windshield
    Point2D.Double r1 = new Point2D.Double(x + width / 6, y + width / 6);
    // The front of the roof
    Point2D.Double r2 = new Point2D.Double(x + width / 3, y);
    // The rear of the roof
    Point2D.Double r3 = new Point2D.Double(x + width * 2 / 3, y);
    // The bottom of the rear windshield
    Point2D.Double r4 = new Point2D.Double(x + width * 5 / 6, y + width / 6);

    Line2D.Double frontWindshield = new Line2D.Double(r1, r2);
    Line2D.Double roofTop = new Line2D.Double(r2, r3);
    Line2D.Double rearWindshield = new Line2D.Double(r3, r4);

    g2.fill(frontTire);
    g2.fill(rearTire);
    g2.setColor(Color.red);
    g2.fill(body);
    g2.draw(frontWindshield);
    g2.draw(roofTop);
    g2.draw(rearWindshield);
}

private int width;
}
```



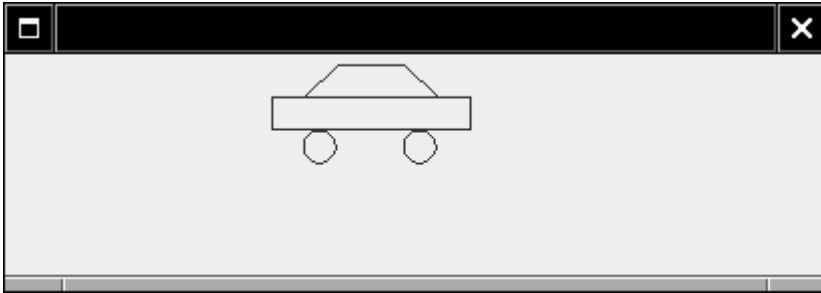
# Ders taslağı

---

1. Icon arabirim tipi
2. Çok biçimlilik
3. Karşılaştırılabilir( Comparable) arabirim tipi
4. Karşılaştırıcı(Comparator) arabirim tipi
5. Anonim sınıflar
6. Çerçeveler ve kullanıcı arabirim tamamlayıcısı
7. Kullanıcı Arabirim Hareketleri
8. Zamanlayıcılar (Timers)
9. Şekille çizme
- 10. Bir arabirim tipini dizayn etme**

# Yeni bir interface tipini tanımlama

- ◆ Arabayı hareket ettirmek için timer'ı kullan
- ◆ `CarShape` ile araba çiz
- ◆ İki sorumluluk:
  - . Şekli çiz
  - Şekli hareket ettir
- ◆ `MoveableShape` tipinde yeni bir interface tanımla



# MoveableShape Interface Tipi için CRC Card

MoveableShape
<i>paint the shape</i>
<i>move the shape</i>

# Yeni bir interface tipi tanımlama

- ◆ Standart kütüphaneye bildirmek için methodları adlandır

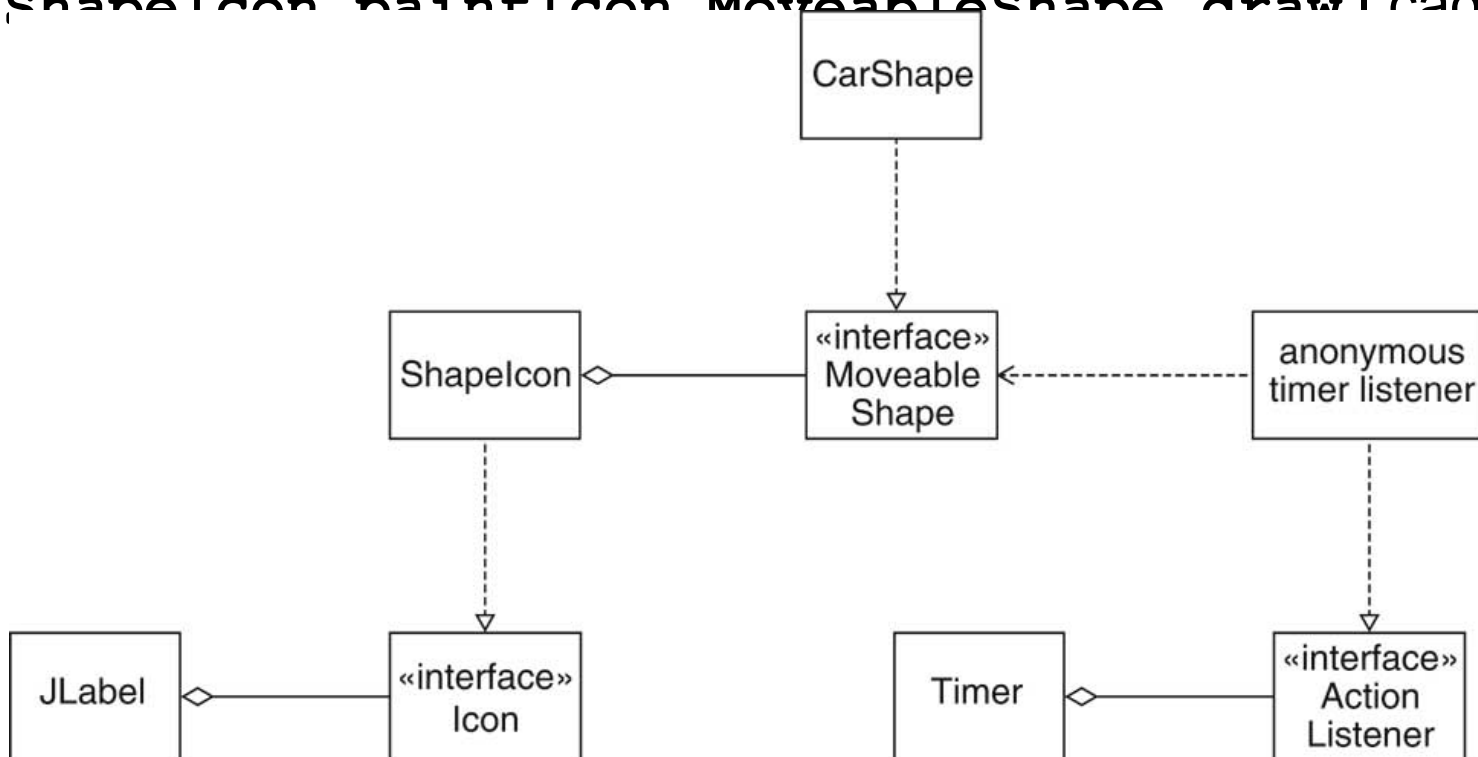
```
public interface MoveableShape
{
    void draw(Graphics2D g2) ;
    void translate(int dx, int dy) ;
}
```

- ◆ CarShape sınıfı MoveableShape implement eder

```
public class CarShape implements MoveableShape
{
    public void translate(int dx, int dy)
    { x += dx; y += dy; }
    . . .
}
```

# Animasyon implement etme

- ◆ Label şekilleri içeren icon'u içerir
- ◆ Timer action şekli hareket ettirir, repaint methodunu çağırır
- ◆ Label Icon'a ihtiyaç duyar, biz MoveableShape'e sahibiz
- ◆ ShapeIcon adapter sınıfı var
- ◆ ShapeIcon paintIcon MoveableShape draw'ı çağırır



# Animasyon implement etme

---

- ◆ Ch4/animation/MoveableShape.java
- ◆ Ch4/animation/Shapelcon.java
- ◆ Ch4/animation/AnimationTester.java
- ◆ Ch4/animation/CarShape.java

# MoveableShape.java

```
import java.awt.*;

/**
 * A shape that can be moved around.
 */
public interface MoveableShape
{
    /**
     * Draws the shape.
     * @param g2 the graphics context
     */
    void draw(Graphics2D g2);

    /**
     * Moves the shape by a given amount.
     * @param dx the amount to translate in x-direction
     * @param dy the amount to translate in y-direction
     */
    void translate(int dx, int dy);
}
```

# Shapelcon.java

```
import java.awt.*;
import java.util.*;
import javax.swing.*;

/**
 * An icon that contains a moveable shape.
 */
public class Shapelcon implements Icon
{
    public Shapelcon(MoveableShape shape, int width, int height)
    {
        this.shape = shape;
        this.width = width;
        this.height = height;
    }

    public int getIconWidth()
    { return width; }

    public int getIconHeight()
    { return height; }

    public void paintIcon(Component c, Graphics g, int x, int y)
    {
        Graphics2D g2 = (Graphics2D) g;
        shape.draw(g2);
    }

    private int width;
    private int height;
    private MoveableShape shape;
}
```

# CarShape.java (1)

```
import java.awt.*;
import java.awt.geom.*;
import java.util.*;

/**
 * A car that can be moved around.
 */
public class CarShape implements MoveableShape
{
    /**
     * Constructs a car item.
     * @param x the left of the bounding rectangle
     * @param y the top of the bounding rectangle
     * @param width the width of the bounding rectangle
     */
    public CarShape(int x, int y, int width)
    {
        this.x = x;
        this.y = y;
        this.width = width;
    }

    public void translate(int dx, int dy)
    {
        x += dx;
        y += dy;
    }

    ....
}
```

# CarShape.java (2)

```
public void draw(Graphics2D g2)
{
    Rectangle2D.Double body = new Rectangle2D.Double(x, y + width / 6, width - 1, width / 6);
    Ellipse2D.Double frontTire = new Ellipse2D.Double(x + width / 6, y + width / 3, width / 6, width / 6);
    Ellipse2D.Double rearTire = new Ellipse2D.Double(x + width * 2 / 3, y + width / 3, width / 6, width / 6);

    // The bottom of the front windshield
    Point2D.Double r1 = new Point2D.Double(x + width / 6, y + width / 6);
    // The front of the roof
    Point2D.Double r2 = new Point2D.Double(x + width / 3, y);
    // The rear of the roof
    Point2D.Double r3 = new Point2D.Double(x + width * 2 / 3, y);
    // The bottom of the rear windshield
    Point2D.Double r4 = new Point2D.Double(x + width * 5 / 6, y + width / 6);
    Line2D.Double frontWindshield = new Line2D.Double(r1, r2);
    Line2D.Double roofTop = new Line2D.Double(r2, r3);
    Line2D.Double rearWindshield = new Line2D.Double(r3, r4);

    g2.draw(body);
    g2.draw(frontTire);
    g2.draw(rearTire);
    g2.draw(frontWindshield);
    g2.draw(roofTop);
    g2.draw(rearWindshield);
}

private int x;
private int y;
private int width;
}
```

# AnimationTester.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

// This program implements an animation that moves a car shape.
public class AnimationTester
{ public static void main(String[] args)
  {
    JFrame frame = new JFrame();
    final MoveableShape shape = new CarShape(0, 0, CAR_WIDTH);
    Shapelcon icon = new Shapelcon(shape, ICON_WIDTH, ICON_HEIGHT);
    final JLabel label = new JLabel(icon);

    frame.setLayout(new FlowLayout());
    frame.add(label);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.pack();
    frame.setVisible(true);

    final int DELAY = 100; // Milliseconds between timer ticks
    Timer t = new Timer(DELAY, new
        ActionListener()
        {
            public void actionPerformed(ActionEvent event)
            {
                shape.translate(1, 0);
                label.repaint();
            }
        });
    t.start();
  }
  private static final int ICON_WIDTH = 400;
  private static final int ICON_HEIGHT = 100;
  private static final int CAR_WIDTH = 100;
}
```

# Referanslar

---

- Bu ders object oriented design and patterns[1]kitabına bağlıdır
- Çeşitli kaynaklardan sağlamış slaytlar ve benim kendi slaytlarımvar

## **Ana kaynaklar:**

1. Object Oriented Design and Patterns (OODP), 2nd Edition, Cay Horstmann, John Wiley, ISBN: 0-471-74487-5, 2005.