Bölüm 9: Grafikler ve Kullanıcı Arayüzleri

Java ile Nesne Yönelimli Programlama

Grafik programlama | JFrame ve JPanel | Graphics ile çizim | Point, Color, Font | Yerleşim yöneticileri | Fare olayları | Düğme ve metin bileşenleri | Örnek uygulamalar

Sercan KÜLCÜ · Giresun Üniversitesi · Elektrik-Elektronik Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Pencere kurmak

`JFrame` ve `JPanel` ile pencere ve panel oluşturmak, bileşen eklemek, pencereyi göstermek ve kapatmak.

Çizmek

`paintComponent` metodunu geçersiz kılıp `Graphics` ile şekil ve metin çizmek; ekran koordinatlarını kullanmak.

Yerleştirmek

`BorderLayout`, `FlowLayout` ve `GridLayout` ile bileşenleri düzenlemek.

Olaylara yanıt vermek

Dinleyici arayüzleri, adaptör sınıfları ve lambda ile fare ve düğme olaylarını işlemek.

Güvenli başlatmak

Arayüzü `SwingUtilities.invokeLater` ile olay iş parçacığında kurmanın nedenini açıklamak.

Uygulama yazmak

Çizim programı, metin dönüştürücü ve para üstü hesaplayıcıyı okuyup değiştirebilmek.

Grafik Programlama

- Java, **grafik kullanıcı arayüzü** (GUI, Graphical User Interface) geliştirmek için güçlü ve kapsamlı bir araç seti sunar: pencereler, düğmeler, metin alanları, çizimler.
- Böylece **etkileşimli ve görsel** uygulamalar yazılabilir. Bunun için üç kavramı anlamak gerekir:

Bileşenler (components)

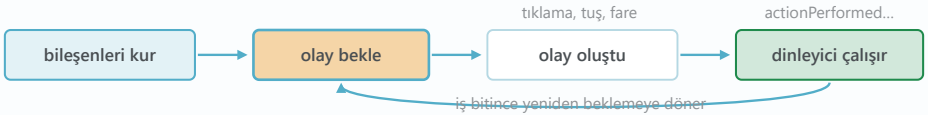
Pencere, düğme, etiket, metin alanı gibi arayüzün **yapı taşları**. Her biri bir sınıfın nesnesidir.

Olay işleme (event handling)

Fare tıklaması, tuşa basma gibi kullanıcı **eylemlerini yakalar** ve bunlara yanıt veren metotları çağırır.

Görsel oluşturma (painting)

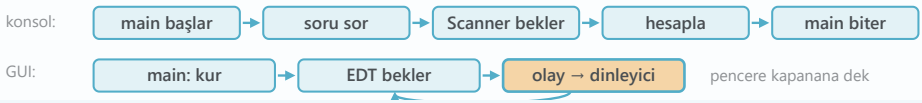
Şekilleri çizer, ekranı günceller ve bileşenlerin **görüntülenmesini** sağlar.



Benzetme: bir GUI programı, müşteriyi bekleyen bir dükkân gibidir. Vitrin (bileşenler) hazırlanır, sonra program **bekler**; müşteri ne zaman neye dokunursa (olay) ilgili görevli (dinleyici metot) çalışır.

Konsol ve GUI Programı

	Konsol programı	GUI programı
Akış	Sıralı: `main` baştan sona çalışır, soruyu program sorar	**Olay güdümlü** (event-driven): program kullanıcının hamlesini bekler
Girdi	`Scanner` ile klavyeden satır okunur	Düğme, metin alanı, fare; dinleyici metotlar çağrılır
Çıktı	`System.out.println`	Bileşenlerin metni değişir veya panel yeniden çizilir
Bitiş	`main` bitince program biter	`main` bitse de pencere kapanana kadar çalışır
Kütüphane	`java.util` , `java.io`	`java.awt` , `javax.swing` , `java.awt.event`



Kural: GUI programında “sonra ne olacak?” sorusunun cevabı koddaki sıra değil, **kullanıcının sıradaki eylemidir**. Kodun büyük kısmı “şu olursa şunu yap” biçimindeki dinleyicilerden oluşur.

GRAFİK PROGRAMLAMA

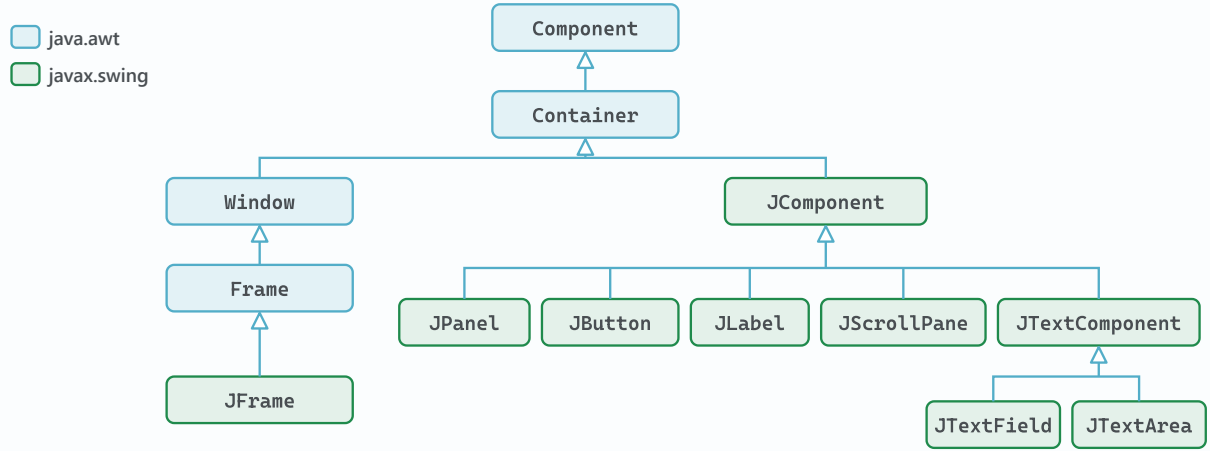
AWT ve Swing

AWT (Abstract Window Toolkit), `java.awt`

Java'nın ilk grafik kütüphanesi. Bugün en çok ortak sınıfları için kullanılır: `Graphics`, `Color`, `Font`, `Point`, yerleşim yöneticileri ve olay sınıfları (`java.awt.event`).

Swing, `javax.swing`

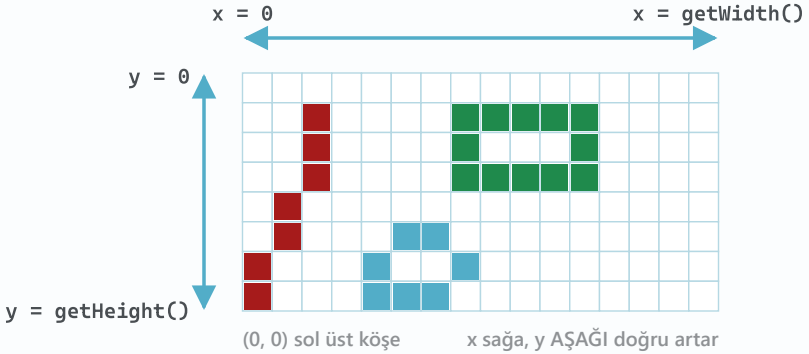
AWT'nin üzerine kurulmuş, bileşenleri Java'nın kendisi çizen kütüphane. Sınıf adları **J** ile başlar: `JFrame`, `JPanel`, `JButton`, `JLabel`...



İçi boş üçgen: "türetilmiştir" (kalıtım). `JFrame` bir AWT penceresidir; diğer Swing bileşenleri `JComponent`'ten türer (`JButton` arada `AbstractButton` üzerinden).

Uygulama Adımları

1. Java'nın grafik sınıfları kullanılarak arayüze **bileşenler eklenir**.
2. Kullanıcının eylemlerine yanıt veren **olay işleme** metotları tanımlanır.
3. `paint()` veya `paintComponent()` gibi metotlarla **görsel oluşturulur**.



Ekran bir **piksel ızgarasıdır**. Koordinatlar tam sayıdır ve bileşenin **sol üst köşesinden** ölçülür. Matematikteki eksenin tersine, **y aşağı doğru büyür**: `(0, 0)` sol üst, `(getWidth() - 1, getHeight() - 1)` sağ alt pikseldir.

GRAFİK PROGRAMLAMA

Temel Grafik Kavramları

Tuval (canvas)

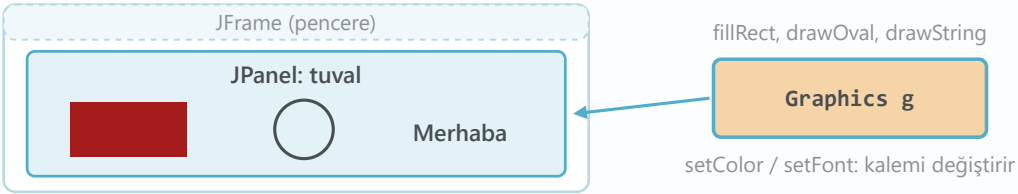
Üzerine resim çizilebilen yüzeydir. Swing'de genellikle `JPanel` (AWT'de `Canvas`) kullanılır: sınıfı `JPanel`'den türetir, çizimi onun içine yaparız.

`Graphics` sınıfı

Çizim işlemlerini yapan "kalem"dir. Metotları: `drawLine()`, `drawRect()`, `fillOval()`, `drawString()`, `drawImage()` ...

`Color` ve `Font`

Kalemin rengini `Color`, yazı tipini `Font` belirler: `g.setColor(Color.RED)`, `g.setFont(new Font("Serif", Font.BOLD, 24))`.

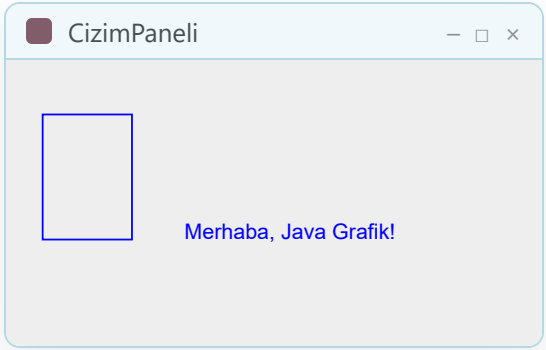


Kalem benzetmesi: `Graphics` nesnesi programcının elindeki kalemdir. `setColor` kalemi değiştirir; değiştirene kadar bütün çizimler **aynı renkle** yapılır.

paintComponent ile Çizim

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

class CizimPaneli extends JPanel {
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);     // zemini temizle
        g.setColor(Color.BLUE);
        g.drawRect(20, 30, 50, 70);  // x, y, en, boy
        g.drawString("Merhaba, Java Grafik!", 100, 100);
    }
}
```

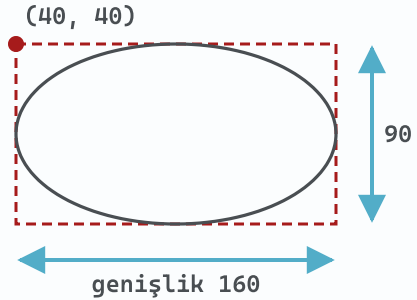


Panel 300 × 160 iken sonuç. `drawString`'in y değeri yazının **taban çizgisidir** (harflerin altı).

- `paintComponent`'i **biz çağırmayız**: pencere ilk açıldığında, boyutu değiştiğinde veya `repaint()` istendiğinde Swing çağırır ve hazır bir `Graphics g` verir.
- `panel.getGraphics()` ile de bir grafik nesnesi alınabilir, ama bununla yapılan çizim pencere bir sonraki kez yeniden çizildiğinde **silinir**. Kalıcı çizim için yol `paintComponent`'tir.

Graphics Çizim Metotları

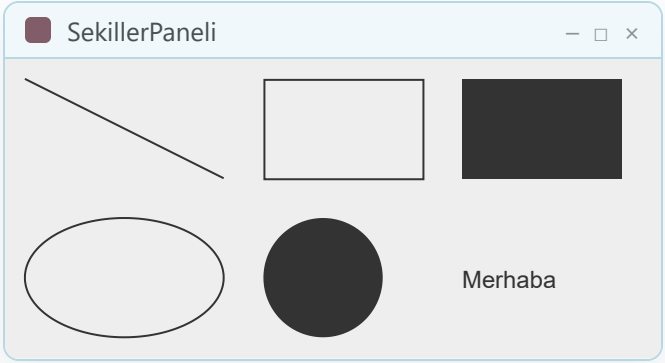
Metot	Ne yapar?
`drawLine(x1, y1, x2, y2)`	İki nokta arasında çizgi
`drawRect(x, y, en, boy)`	Dikdörtgen çerçevesi; (x, y) sol üst köşe
`fillRect(x, y, en, boy)`	İçi dolu dikdörtgen
`drawOval(x, y, en, boy)`	Bu dikdörtgene sığan elips (en = boy ise çember)
`fillOval(x, y, en, boy)`	İçi dolu elips / daire
`drawString(s, x, y)`	Metin; y taban çizgisi
`setColor(c)` / `setFont(f)`	Sonraki çizimlerin rengi / yazı tipi
`drawImage(resim, x, y, null)`	Bir resmi (x, y) noktasına çizer



Elips ve daireler **sınırlayıcı dikdörtgenleri** ile verilir: merkez değil, sol üst köşe. Merkezi (cx, cy), yarıçapı r olan daire: `fillOval(cx - r, cy - r, 2r, 2r)`.

Şekiller: Kod ve Sonuç

```
class SekillerPaneli extends JPanel {
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawLine(10, 10, 110, 60);
        g.drawRect(130, 10, 80, 50);
        g.fillRect(230, 10, 80, 50);
        g.drawOval(10, 80, 100, 60);
        g.fillOval(130, 80, 60, 60);
        g.drawString("Merhaba", 230, 115);
    }
}
```

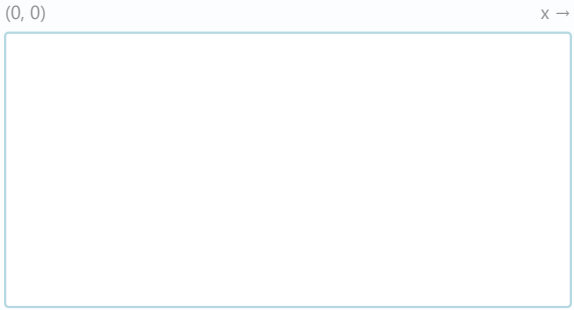


Renk verilmediğinde kalem, panelin **ön plan rengi** (Swing'de koyu gri) ile çizer. `draw…` çerçeve, `fill…` içi dolu şekil çizer.

Çizimler **üst üste** boyanır; sonradan çizilen öncekini örter.

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
▶          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
11     }
12 }
```



Swing bu metodu kendisi çağırır. `super.paintComponent(g)` tuvali arka plan rengiyle **temizler**.

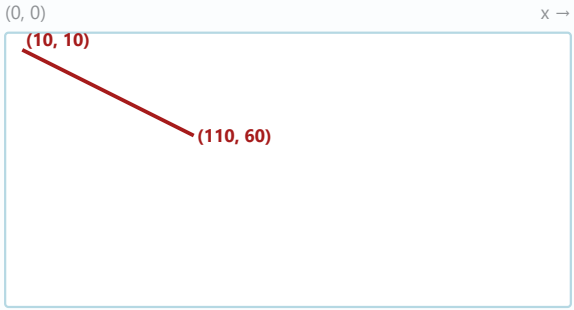
Adım 1 / 8

DEĞİŞKENLER

`g = Graphics (koyu gri kalem)`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
▶          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
11     }
12 }
```



İki nokta arasına çizgi. y aşağı doğru büyüdüğü için çizgi sağa ve **aşağı** iner.

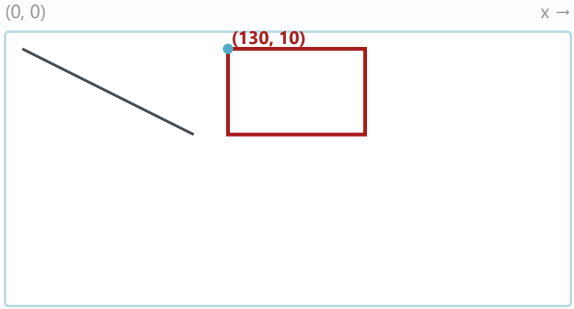
Adım 2 / 8

DEĞİŞKENLER

`x1 = 10` `y1 = 10` `x2 = 110` `y2 = 60`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
▶          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
11     }
12 }
```



`drawRect` : (x, y) **sol üst köşe**; yalnız çerçeve çizilir, içi boş kalır.

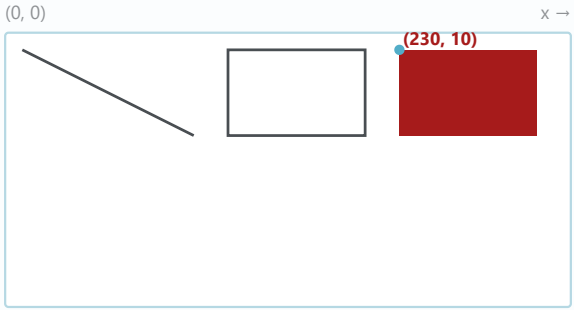
Adım 3 / 8

DEĞİŞKENLER

`x = 130` `y = 10` `en = 80` `boy = 50`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
▶          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
11     }
12 }
```



`fillRect` : aynı boyutta, içi kalemin rengiyle dolu dikdörtgen.

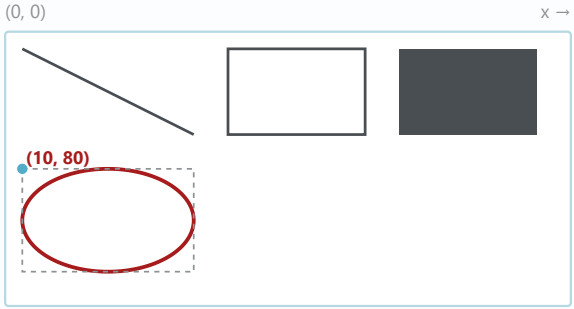
Adım 4 / 8

DEĞİŞKENLER

`x = 230` `y = 10` `en = 80` `boy = 50`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
▶          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
11     }
12 }
```



`drawOval` : kesikli kutuya **sığan** elips. (10, 80) elipsin değil kutunun köşesi.

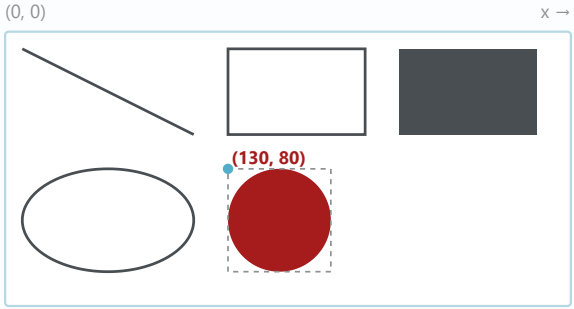
Adım 5 / 8

DEĞİŞKENLER

`x = 10` `y = 80` `en = 100` `boy = 60`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
▶          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
11     }
12 }
```



En = boy olduğundan dolu bir **daire**: merkez (160, 110), yarıçap 30.

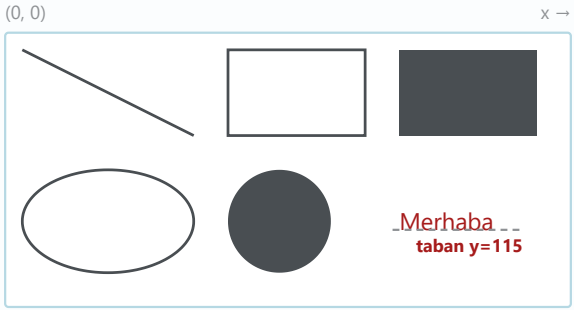
Adım 6 / 8

DEĞİŞKENLER

`x = 130` `y = 80` `en = 60` `boy = 60`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
▶          g.drawString("Merhaba", 230, 115);
11     }
12 }
```



`drawString`'de y, metnin **taban çizgisidir**: harfler 115'in üstüne yazılır.

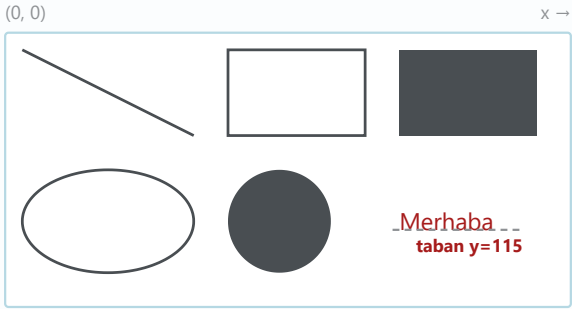
Adım 7 / 8

DEĞİŞKENLER

`s = "Merhaba"` `x = 230` `y = 115`

Şekiller: Çizim Sırası Adım Adım

```
1  class SekillerPaneli extends JPanel {
2      @Override
3      public void paintComponent(Graphics g) {
4          super.paintComponent(g);
5          g.drawLine(10, 10, 110, 60);
6          g.drawRect(130, 10, 80, 50);
7          g.fillRect(230, 10, 80, 50);
8          g.drawOval(10, 80, 100, 60);
9          g.fillOval(130, 80, 60, 60);
10         g.drawString("Merhaba", 230, 115);
▶      }
12 }
```



Metot biter; pencere her yeniden boyandığında (boyut değişimi, `repaint()`) aynı sırayla **baştan** çizilir.

Adım 8 / 8

DEĞİŞKENLER

—

PENCERE VE PANEL

JFrame Sınıfı

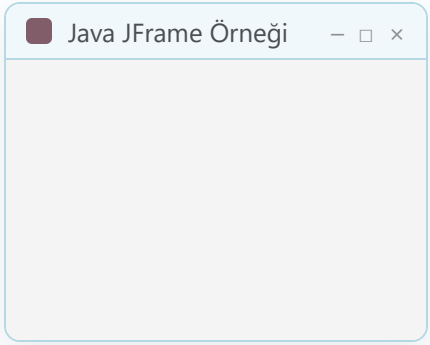
Ekranda başlık çubuğu, simge durumuna küçültme ve kapatma düğmeleri olan **üst düzey pencere** (top-level window) oluşturur. Diğer bütün bileşenler bir pencerenin içinde yaşar.

Kod	Görevi
`JFrame pencere = new JFrame("Başlık");`	Başlığı verilen pencereyi oluşturur (henüz görünmez)
`pencere.add(bilesen);`	Pencereye (içerik bölmesine) bileşen ekler
`pencere.setSize(400, 300);`	Genişlik ve yükseklik (piksel, kenarlıklar dahil)
`pencere.pack();`	Boyutu içerikteki bileşenlerin tercih ettiği boyuta göre ayarlar
`pencere.setLocation(x, y);`	Pencerenin ekrandaki konumu; `setLocationRelativeTo(null)` ortalar
`pencere.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);`	Kapatılınca programı sonlandırır
`pencere.setVisible(true);`	Pencereyi görünür yapar (genellikle en son çağrılır)

İlk Pencere

```java
import javax.swing.JFrame;

public class JFrameOrnegi {
    public static void main(String[] args) {
        JFrame pencere = new JFrame("Java JFrame Örneği");
        pencere.setSize(400, 300);
        pencere.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pencere.setVisible(true);
    }
}
```

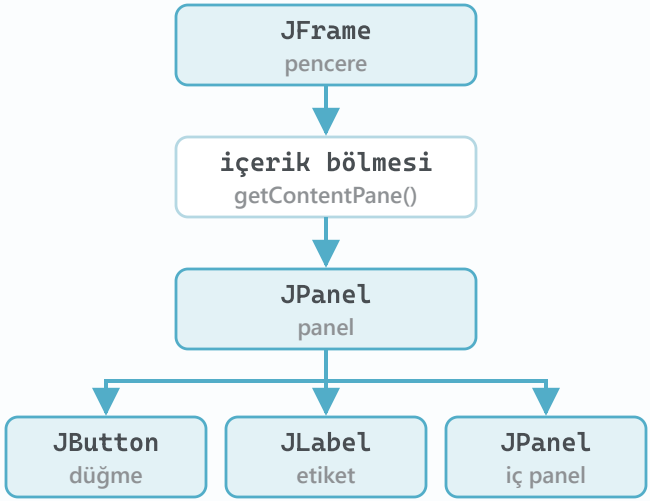


İçi boş, 400 × 300 boyutlu pencere (ölçekli).

- `setVisible(true)` unutulursa pencere oluşur ama **görünmez**.
- `EXIT_ON_CLOSE` verilmezse varsayılan davranış `HIDE_ON_CLOSE` 'dur: pencere gizlenir ama program **arka planda çalışmaya devam eder**.
- Boyut verilmezse pencere 0 × 0'a yakın, yalnız başlık çubuğu kadar açılır.

JPanel Sınıfı

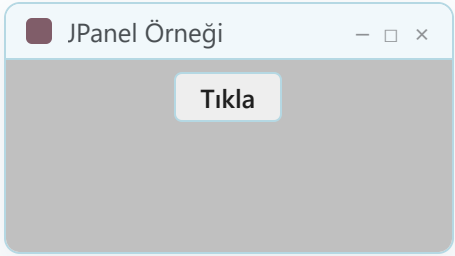
- `javax.swing` paketinde bulunan bir **kapsayıcıdır** (container): içinde başka GUI öğeleri barındırır.
- Bir pencerenin içindeki bileşenleri **gruplamak ve düzenlemek** için kullanılır; birden fazla panel iç içe konarak karmaşık düzenler kurulur.
- `setBackground()`, `setBorder()`, `setPreferredSize()` ile renk, kenarlık ve boyut özelleştirilir.
- Kendi çizimimizi, animasyon ve grafikleri yaptığımız **tuvaldir**: `JPanel` 'den türetip `paintComponent` 'i geçersiz kılarız.
- Kodu parçalara ayırmaya yarar: her panel ayrı bir sınıf olabilir.



JPanel Örneği

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

public class PanelOrnegi {
    public static void main(String[] args) {
        JPanel panel = new JPanel();
        panel.setBackground(Color.LIGHT_GRAY);
        panel.add(new JButton("Tıkla"));
        JFrame pencere = new JFrame("JPanel Örneği");
        pencere.add(panel);
        pencere.setSize(300, 150);
        pencere.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pencere.setVisible(true);
    }
}
```

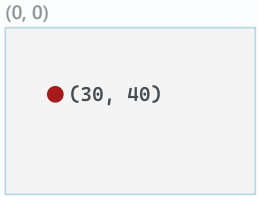


`Color.LIGHT_GRAY` = (192, 192, 192). Düğme panelin üst ortasında durur.

Neden üst ortada? `JPanel` 'in varsayılan yerleşim yöneticisi **`FlowLayout`** 'tur: bileşenleri tercih ettikleri boyutta, satır satır ve ortalı dizer. `JFrame` 'in içerik bölmesi ise **`BorderLayout`** kullanır; bu yüzden panel bütün pencereyi kaplar.

Point Sınıfı: (x, y) Noktası

```
▶ Point nokta = new Point(30, 40);
2 double xKoordinati = nokta.getX();
3 double yKoordinati = nokta.getY();
4 System.out.println(xKoordinati + " " + yKoordinati);
5 System.out.println(nokta.x + " " + nokta.y);
6 System.out.println(nokta.distance(0, 0));
7 nokta.setLocation(60, 80);
8 System.out.println(nokta);
9 nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`Point` nesnesi oluşur; `x` ve `y` alanları `public int` 'tir.

Adım 1 / 11

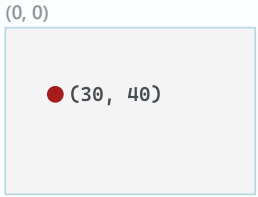
ÇIKTI

DEĞİŞKENLER

`nokta = (30, 40)`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
▶  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`getX()` `double` döndürür: 30.0.

Adım 2 / 11

ÇIKTI

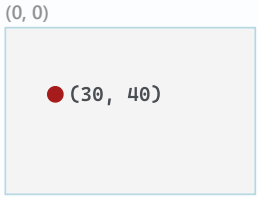
DEĞİŞKENLER

`nokta = (30, 40)`

`xKoordinati = 30.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
▶  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`getY()` : 40.0.

Adım 3 / 11

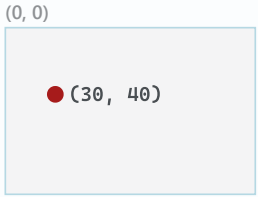
ÇIKTI

DEĞİŞKENLER

- `nokta = (30, 40)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
▶  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`double` değerler ondalıklı yazılır.

Adım 4 / 11

ÇIKTI

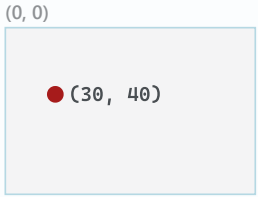
```
30.0 40.0
```

DEĞİŞKENLER

- `nokta = (30, 40)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
▶  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



Alanlara doğrudan erişim `int` verir.

ÇIKTI

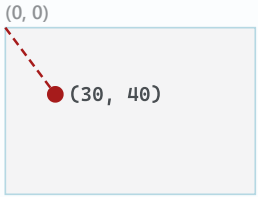
```
30.0 40.0
30 40
```

DEĞİŞKENLER

- `nokta = (30, 40)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
▶  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



Orijine uzaklık: $\sqrt{(30^2 + 40^2)} = 50.0$.

Adım 6 / 11

ÇIKTI

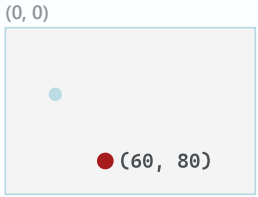
```
30.0 40.0
30 40
50.0
```

DEĞİŞKENLER

- nokta = **(30, 40)**
- xKoordinati = **30.0**
- yKoordinati = **40.0**

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
▶  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`setLocation` noktayı yeni konuma taşır. Değişkenler eski değerleri saklar.

Adım 7 / 11

ÇIKTI

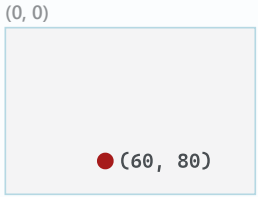
```
30.0 40.0
30 40
50.0
```

DEĞİŞKENLER

- `nokta = (60, 80)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
▶  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`toString()` sınıf adını ve alanları yazar.

Adım 8 / 11

ÇIKTI

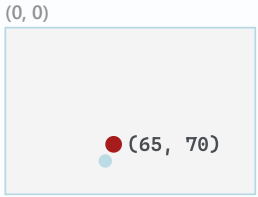
```
30.0 40.0
30 40
50.0
java.awt.Point[x=60,y=80]
```

DEĞİŞKENLER

- `nokta = (60, 80)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
▶  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



`translate(dx, dy)` göreli kaydırır. y azaldı: nokta **yukarı** çıktı.

Adım 9 / 11

ÇIKTI

```
30.0 40.0
30 40
50.0
java.awt.Point[x=60,y=80]
```

DEĞİŞKENLER

- `nokta = (65, 70)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
1  Point nokta = new Point(30, 40);
2  double xKoordinati = nokta.getX();
3  double yKoordinati = nokta.getY();
4  System.out.println(xKoordinati + " " + yKoordinati);
5  System.out.println(nokta.x + " " + nokta.y);
6  System.out.println(nokta.distance(0, 0));
7  nokta.setLocation(60, 80);
8  System.out.println(nokta);
9  nokta.translate(5, -10);   // 5 sağa, 10 yukarı
▶  System.out.println(nokta);
11 System.out.println(Point.distance(0, 0, 3, 4));
```



Yeni konum yazılır.

Adım 10 / 11

ÇIKTI

```
30.0 40.0
30 40
50.0
java.awt.Point[x=60,y=80]
java.awt.Point[x=65,y=70]
```

DEĞİŞKENLER

- `nokta = (65, 70)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Point Sınıfı: (x, y) Noktası

```
 1 Point nokta = new Point(30, 40);
 2 double xKoordinati = nokta.getX();
 3 double yKoordinati = nokta.getY();
 4 System.out.println(xKoordinati + " " + yKoordinati);
 5 System.out.println(nokta.x + " " + nokta.y);
 6 System.out.println(nokta.distance(0, 0));
 7 nokta.setLocation(60, 80);
 8 System.out.println(nokta);
 9 nokta.translate(5, -10);   // 5 sağa, 10 yukarı
10 System.out.println(nokta);
▶ System.out.println(Point.distance(0, 0, 3, 4));
```



`static distance(x1, y1, x2, y2)` : (0, 0) ile (3, 4) arası 5.0.

Adım 11 / 11

ÇIKTI

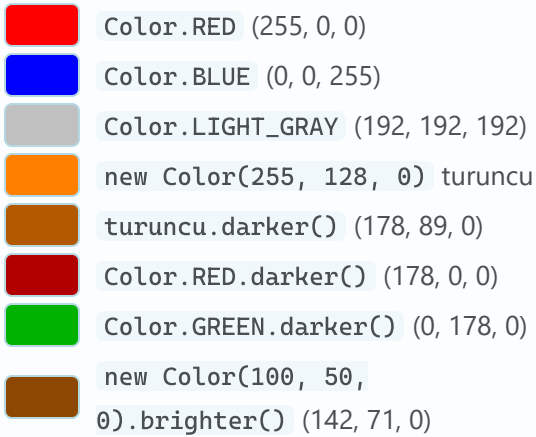
```
30.0 40.0
30 40
50.0
java.awt.Point[x=60,y=80]
java.awt.Point[x=65,y=70]
5.0
```

DEĞİŞKENLER

- `nokta = (65, 70)`
- `xKoordinati = 30.0`
- `yKoordinati = 40.0`

Color Sınıfı

- Arayüzün ve çizimlerin **renklendirilmesini** sağlar (`java.awt.Color`).
- **Hazır sabitler**: `Color.RED` , `Color.GREEN` , `Color.WHITE` , `Color.LIGHT_GRAY` ...
- **RGB**: kırmızı, yeşil, mavi bileşenleri 0–255 arası: `new Color(int red, int green, int blue)` .
- `darker()` bileşenleri 0,7 ile çarpar, `brighter()` 0,7'ye böler (en çok 255): **yeni** bir renk nesnesi döner.




```
Color turuncu = new Color(255, 128, 0);
System.out.println(turuncu);
System.out.println(turuncu.getGreen());
System.out.println(Color.RED.darker());
```

```
java.awt.Color[r=255,g=128,b=0]
128
java.awt.Color[r=178,g=0,b=0]
```

Font Sınıfı

Metnin yazı tipini, stilini ve boyutunu belirler: `new Font(ad, stil, punto)`.

Parametre	Değerler
ad	`"Serif"`, `"SansSerif"`, `"Monospaced"`, `"Dialog"` (her sistemde bulunan mantıksal adlar) veya `"Arial"` gibi gerçek bir yazı tipi
stil	`Font.PLAIN` (0), `Font.BOLD` (1), `Font.ITALIC` (2), ikisi birden: `Font.BOLD | Font.ITALIC`
punto	Tam sayı boyut, ör. 24

```
public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(new Color(0, 102, 204));
    g.setFont(new Font("Serif", Font.BOLD, 24));
    g.drawString("Java", 20, 40);
    g.setFont(new Font("Monospaced", Font.PLAIN, 14));
    g.drawString("Grafik", 20, 70);
}
```

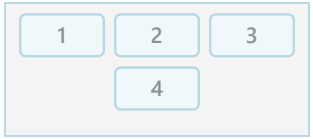


`JLabel` gibi bileşenlerde de aynı nesne kullanılır: `etiket.setFont(f)`. Mevcut bir yazı tipinden türetmek için: `f.deriveFont(Font.ITALIC, 18f)`.

Yerleşim Yöneticisi

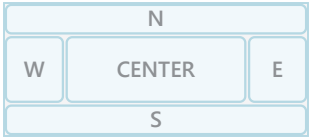
Yerleşim yöneticisi (layout manager), bir kapsayıcının içindeki bileşenlerin **boyutunu ve konumunu** kurallara göre hesaplayan nesnedir: `panel.setLayout(new GridLayout(2, 3));`. Koordinatları elle vermek yerine kullanılır, çünkü pencere büyüyünce, yazı tipi veya ekran çözünürlüğü değişince düzen kendiliğinden yeniden hesaplanır.

FlowLayout



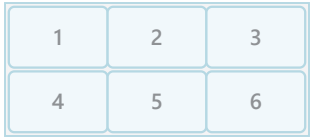
Soldan sağa, satır dolunca alta. **JPanel varsayılanı**.

BorderLayout



Beş bölge. **JFrame içerik bölmesi varsayılanı**.

GridLayout



Eşit boyutlu hücrelerden ızgara: `new GridLayout(2, 3)`.

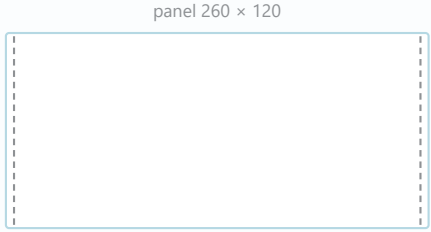
BoxLayout

Tek satır veya tek sütun: `BoxLayout.Y_AXIS`.

Daha esnek ama karmaşık olan `GridBagLayout`, farklı boyutlarda hücrelerden oluşan bir ızgara kurar. Panelleri iç içe koyup her birine basit bir yönetici vermek çoğu zaman yeterlidir.

FlowLayout Satırları Nasıl Doldurur?

```
▶ int[] gen = {90, 60, 110, 70, 80};
▶ JPanel panel = new JPanel();   // FlowLayout(CENTER, 5, 5)
▶ panel.setSize(260, 120);
4 for (int i = 0; i < gen.length; i++) {
5     JButton b = new JButton("B" + i);
6     b.setPreferredSize(new Dimension(gen[i], 30));
7     panel.add(b);
8 }
9 panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



eklendi:

`JPanel`'in varsayılan yöneticisi `FlowLayout`: ortalı, bileşenler arası 5 piksel boşluk. Kesikli çizgiler kenar boşluğu.

Adım 1 / 9

DEĞİŞKENLER

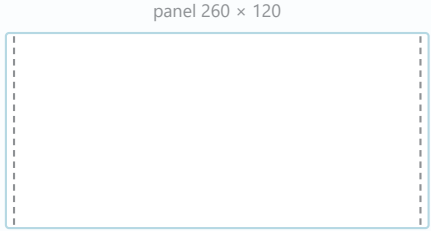
`gen = [90, 60, 110, 70, 80]`

`panel = 260×120`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5, 5)
3  panel.setSize(260, 120);
4  for (int i = 0; i < gen.length; i++) {
▶      JButton b = new JButton("B" + i);
▶      b.setPreferredSize(new Dimension(gen[i], 30));
▶      panel.add(b);
8  }
9  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



`add` düğmeyi yalnızca listeye ekler; konumu henüz (0, 0), boyutu 0 × 0.

Adım 2 / 9

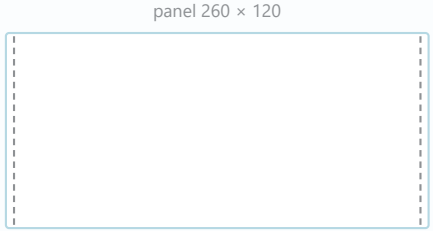
DEĞİŞKENLER

`i = 0` `b = B0 (90×30)`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5, 5)
3  panel.setSize(260, 120);
▶  for (int i = 0; i < gen.length; i++) {
▶      JButton b = new JButton("B" + i);
▶      b.setPreferredSize(new Dimension(gen[i], 30));
▶      panel.add(b);
8  }
9  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



Döngü bitti: beş düğme eklendi, hiçbiri yerleşmedi. Yerleşim ancak düzen hesaplanınca yapılır.

Adım 3 / 9

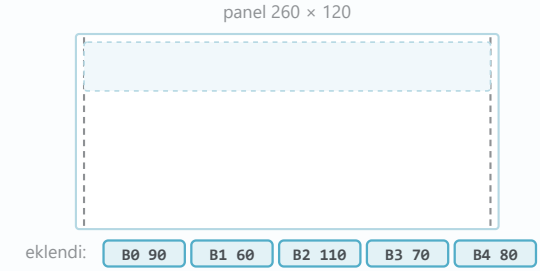
DEĞİŞKENLER

`i = 4` `b = B4 (80×30)`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5, 5)
3  panel.setSize(260, 120);
4  for (int i = 0; i < gen.length; i++) {
5      JButton b = new JButton("B" + i);
6      b.setPreferredSize(new Dimension(gen[i], 30));
7      panel.add(b);
8  }
▶  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



`doLayout()` : kullanılabilir genişlik 260 − 2 × 5 = 250. B0 ve B1 sığar: 155 piksel.

Adım 4 / 9

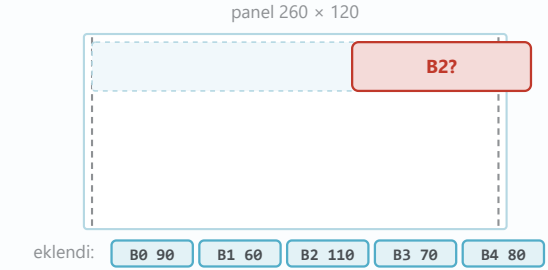
DEĞİŞKENLER

`satır = 1` `genişlik = 90+5+60=155`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5,
   5)
3  panel.setSize(260, 120);
4  for (int i = 0; i < gen.length; i++) {
5      JButton b = new JButton("B" + i);
6      b.setPreferredSize(new Dimension(gen[i], 30));
7      panel.add(b);
8  }
▶  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



B2 de eklenirse satır 270 piksel olur: sığmaz, **alt satıra** iner.

Adım 5 / 9

DEĞİŞKENLER

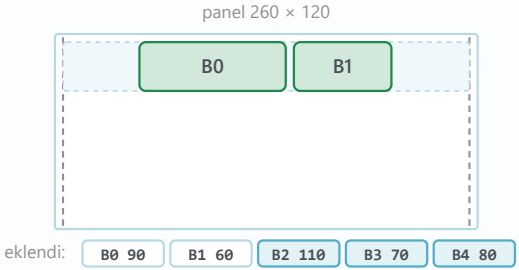
`satır = 1`

`deneme = 155+5+110 = 270 > 250`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5,
   5)
3  panel.setSize(260, 120);
4  for (int i = 0; i < gen.length; i++) {
5      JButton b = new JButton("B" + i);
6      b.setPreferredSize(new Dimension(gen[i], 30));
7      panel.add(b);
8  }
▶  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



Satır 1 ortalanır: B0 x = 52, B1 x = 52 + 90 + 5 = 147, y = 5.

Adım 6 / 9

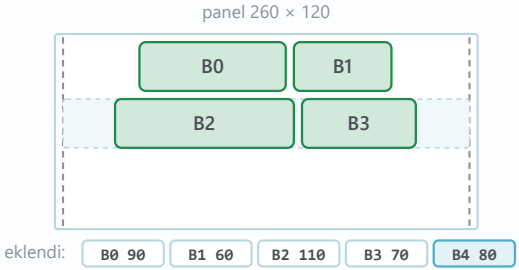
DEĞİŞKENLER

`satır = 1` `x0 = 5+(250−155)/2 = 52`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5,
   5)
3  panel.setSize(260, 120);
4  for (int i = 0; i < gen.length; i++) {
5      JButton b = new JButton("B" + i);
6      b.setPreferredSize(new Dimension(gen[i], 30));
7      panel.add(b);
8  }
▶  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



Satır 2: B2 ve B3 (185 piksel); B4 eklense 270 olur. Ortalama: x0 = 5 + (250 − 185)/2 = 37.

Adım 7 / 9

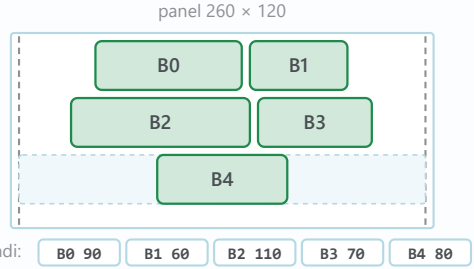
DEĞİŞKENLER

`satır = 2` `genişlik = 110+5+70 = 185`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1  int[] gen = {90, 60, 110, 70, 80};
2  JPanel panel = new JPanel();   // FlowLayout(CENTER, 5,
   5)
3  panel.setSize(260, 120);
4  for (int i = 0; i < gen.length; i++) {
5      JButton b = new JButton("B" + i);
6      b.setPreferredSize(new Dimension(gen[i], 30));
7      panel.add(b);
8  }
▶  panel.doLayout();              // konumları hesapla
10 for (Component c : panel.getComponents())
11     System.out.println(c.getX() + ", " + c.getY());
```



B4 tek başına üçüncü satırda; y = 5 + 30 + 5 + 30 + 5 = 75.

Adım 8 / 9

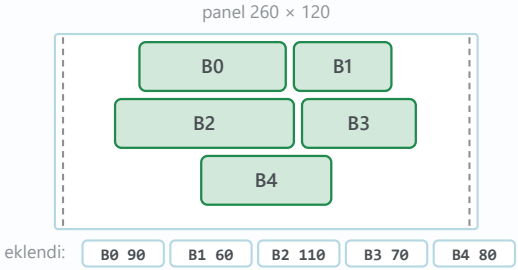
DEĞİŞKENLER

`satır = 3` `x0 = 5+(250−80)/2 = 90`

ÇIKTI

FlowLayout Satırları Nasıl Doldurur?

```
1 int[] gen = {90, 60, 110, 70, 80};
2 JPanel panel = new JPanel();   // FlowLayout(CENTER, 5,
  5)
3 panel.setSize(260, 120);
4 for (int i = 0; i < gen.length; i++) {
5     JButton b = new JButton("B" + i);
6     b.setPreferredSize(new Dimension(gen[i], 30));
7     panel.add(b);
8 }
9 panel.doLayout();              // konumları hesapla
▶ for (Component c : panel.getComponents())
▶     System.out.println(c.getX() + ", " + c.getY());
```



Hesaplanan sınırlar yazdırılır. Panel genişlerse satırlar yeniden hesaplanır ve düğmeler yer değiştirir.

Adım 9 / 9

DEĞİŞKENLER

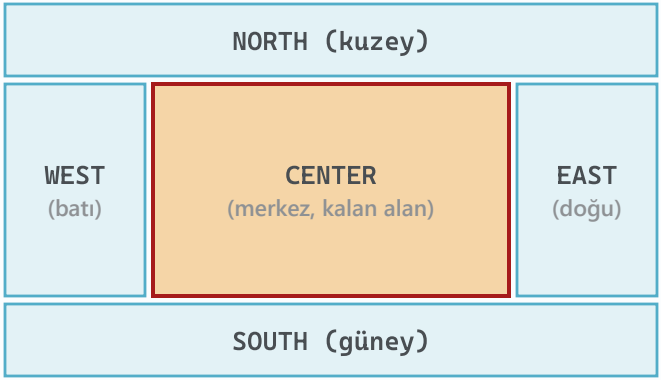
`c = B0 … B4`

ÇIKTI

```
52, 5
147, 5
37, 40
152, 40
90, 75
```

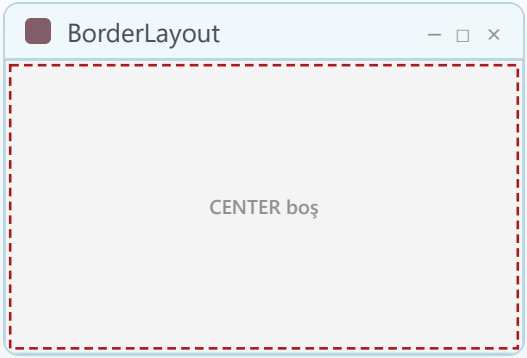
BorderLayout Sınıfı

- Bileşenleri **beş bölgeye** yerleştirir: `NORTH`, `SOUTH`, `WEST`, `EAST`, `CENTER`.
- Önceliği farklı bileşenleri düzenli göstermeye yarar: araç çubuğu üstte, durum satırı altta, asıl içerik ortada.
- Bölge belirtilmeden eklenen bileşen **merkeze** yerleşir: `add(b)` = `add(b, BorderLayout.CENTER)`.
- Her bölgede **tek** bileşen olur; aynı bölgeye ikinci ekleme öncekinin yerini alır.
- Kuzey ve güney tüm genişliği, batı ve doğu kalan yüksekliği alır; **merkez geriye kalan her şeyi** doldurur.
- `setHgap()` ve `setVgap()` bölgeler arasındaki yatay ve dikey boşluğu (piksel) ayarlar.



BorderLayout: Bölgeleri Doldurma

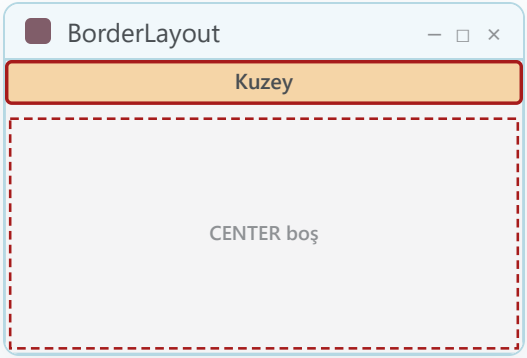
```
JPanel panel = new JPanel();
BorderLayout yerlesim = new BorderLayout();
yerlesim.setHgap(5);
yerlesim.setVgap(5);
panel.setLayout(yerlesim);
panel.add(new JButton("Kuzey"), BorderLayout.NORTH);
panel.add(new JButton("Güney"), BorderLayout.SOUTH);
panel.add(new JButton("Batı"), BorderLayout.WEST);
panel.add(new JButton("Doğu"), BorderLayout.EAST);
panel.add(new JButton("Merkez"), BorderLayout.CENTER);
```



Panel `BorderLayout` kullanacak; bölgeler arasında 5 piksel boşluk var. Henüz bileşen yok.

BorderLayout: Bölgeleri Doldurma

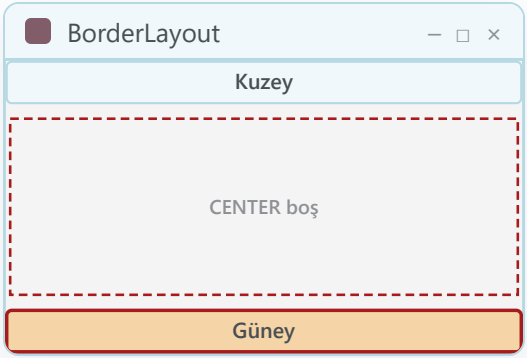
```
1  JPanel panel = new JPanel();
2  BorderLayout yerlesim = new BorderLayout();
3  yerlesim.setHgap(5);
4  yerlesim.setVgap(5);
5  panel.setLayout(yerlesim);
▶  panel.add(new JButton("Kuzey"), BorderLayout.NORTH);
7  panel.add(new JButton("Güney"), BorderLayout.SOUTH);
8  panel.add(new JButton("Batı"), BorderLayout.WEST);
9  panel.add(new JButton("Doğu"), BorderLayout.EAST);
10 panel.add(new JButton("Merkez"), BorderLayout.CENTER);
```



Kuzey düğmesi **tüm genişliği** alır, yüksekliği düğmenin tercih ettiği kadardır.

BorderLayout: Bölgeleri Doldurma

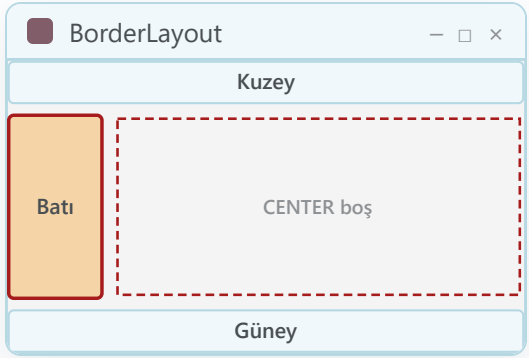
```
1  JPanel panel = new JPanel();
2  BorderLayout yerlesim = new BorderLayout();
3  yerlesim.setHgap(5);
4  yerlesim.setVgap(5);
5  panel.setLayout(yerlesim);
6  panel.add(new JButton("Kuzey"), BorderLayout.NORTH);
▶  panel.add(new JButton("Güney"), BorderLayout.SOUTH);
8  panel.add(new JButton("Batı"), BorderLayout.WEST);
9  panel.add(new JButton("Doğu"), BorderLayout.EAST);
10 panel.add(new JButton("Merkez"), BorderLayout.CENTER);
```



Güney de tüm genişlikte, en alta yerleşir.

BorderLayout: Bölgeleri Doldurma

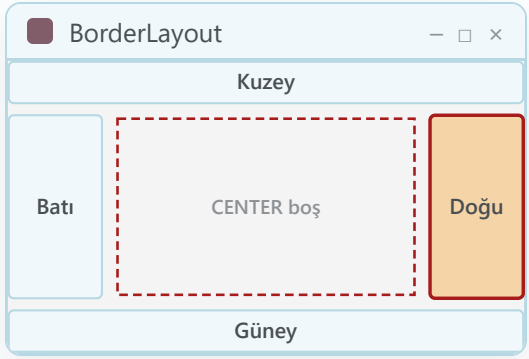
```
1  JPanel panel = new JPanel();
2  BorderLayout yerlesim = new BorderLayout();
3  yerlesim.setHgap(5);
4  yerlesim.setVgap(5);
5  panel.setLayout(yerlesim);
6  panel.add(new JButton("Kuzey"), BorderLayout.NORTH);
7  panel.add(new JButton("Güney"), BorderLayout.SOUTH);
▶  panel.add(new JButton("Batı"), BorderLayout.WEST);
9  panel.add(new JButton("Doğu"), BorderLayout.EAST);
10 panel.add(new JButton("Merkez"), BorderLayout.CENTER);
```



Batı, kuzey ile güney arasında kalan yüksekliği alır; genişliği metne göre.

BorderLayout: Bölgeleri Doldurma

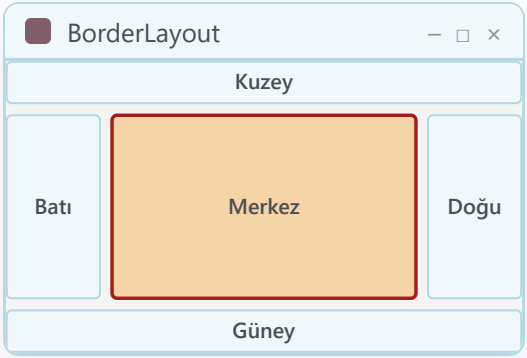
```
1  JPanel panel = new JPanel();
2  BorderLayout yerlesim = new BorderLayout();
3  yerlesim.setHgap(5);
4  yerlesim.setVgap(5);
5  panel.setLayout(yerlesim);
6  panel.add(new JButton("Kuzey"), BorderLayout.NORTH);
7  panel.add(new JButton("Güney"), BorderLayout.SOUTH);
8  panel.add(new JButton("Batı"), BorderLayout.WEST);
▶  panel.add(new JButton("Doğu"), BorderLayout.EAST);
10 panel.add(new JButton("Merkez"), BorderLayout.CENTER);
```



Doğu sağ kenara yerleşir.

BorderLayout: Bölgeleri Doldurma

```
1 JPanel panel = new JPanel();
2 BorderLayout yerlesim = new BorderLayout();
3 yerlesim.setHgap(5);
4 yerlesim.setVgap(5);
5 panel.setLayout(yerlesim);
6 panel.add(new JButton("Kuzey"), BorderLayout.NORTH);
7 panel.add(new JButton("Güney"), BorderLayout.SOUTH);
8 panel.add(new JButton("Batı"), BorderLayout.WEST);
9 panel.add(new JButton("Doğu"), BorderLayout.EAST);
▶ panel.add(new JButton("Merkez"), BorderLayout.CENTER);
```



Merkez **geriye kalan bütün alanı** doldurur. Pencere büyütülürse en çok merkez büyür.

Giriş Yasak İşareti

Hedef: trafikteki “girişi olmayan yol” işaretini çizen bir panel. Şekil üst üste boyanan dört katmandan oluşur:

1. Yeşil arka plan (`setBackground`)
2. Beyaz büyük daire (dış halka)
3. Kırmızı, biraz küçük daire
4. Ortada beyaz yatay bar

Bütün ölçüler `getWidth()` ve `getHeight()` 'tan **hesaplanır**: pencere büyütülünce işaret de büyür ve hep ortada kalır.



paintComponent Kodu

```
public class GirisYasaktirIsareti extends JPanel {
    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        var c = new Point(getWidth() / 2, getHeight() / 2); // merkez
        var r = Math.min(getWidth() / 2, getHeight() / 2) - 5; // yarıçap
        var r2 = (int) (r * 0.9);   // iç yarıçap
        var bw = (int) (r2 * 1.4);  // bar genişliği
        var bh = (int) (r2 * 0.35); // bar yüksekliği
        g.setColor(Color.WHITE);
        g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
        g.setColor(Color.RED);
        g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
        g.setColor(Color.WHITE);
        g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
    }
}
```

`var` (Java 10+): tipi derleyici sağdaki ifadeden çıkarır; burada `c` bir `Point`, diğerleri `int`. Klasik yazım: `Point c = new Point(…); int r = …;`. `(int)` dönüşümü ondalık kısmı **atar**. Gerekli içe aktarmalar: `java.awt.*` ve `javax.swing.*`.

İşareti Çizme: `paintComponent` İzleme

```
▶ super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



Panel 384 × 261 piksel. `super.paintComponent` zemini arka plan rengiyle (yeşil) boyar.

Adım 1 / 9

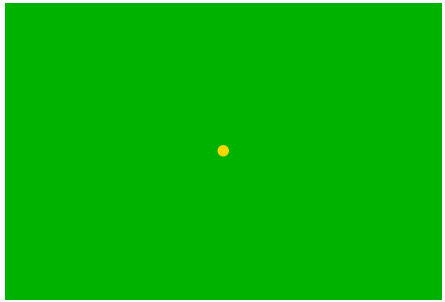
DEĞİŞKENLER

`getWidth() = 384`

`getHeight() = 261`

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



Merkez: (384 / 2, 261 / 2) = (192, 130). Tam sayı bölmesi: 130,5 → 130.

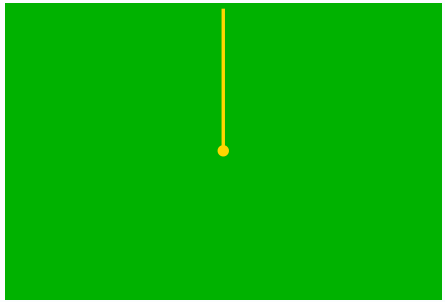
Adım 2 / 9

DEĞİŞKENLER

- `getWidth() = 384`
- `getHeight() = 261`
- `c = (192, 130)`

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
▶ 3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



Yarıçap: min(192, 130) − 5 = 125. Kısa kenara göre seçilir, 5 piksel pay kalır.

Adım 3 / 9

DEĞİŞKENLER

- `getWidth()` = **384**
- `getHeight()` = **261**
- `c` = **(192, 130)**
- `r` = **125**

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
▶  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



İç yarıçap: (int) 112,5 = 112.

Adım 4 / 9

DEĞİŞKENLER

- `getWidth()` = **384**
- `getHeight()` = **261**
- `c` = **(192, 130)**
- `r` = **125**
- `r2` = **112**

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
▶  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



Bar genişliği: (int) (112 × 1,4) = (int) 156,8 = 156.

Adım 5 / 9

DEĞİŞKENLER

`getWidth() = 384`

`getHeight() = 261`

`c = (192, 130)` `r = 125`

`r2 = 112` `bw = 156`

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
▶  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



Bar yüksekliği: (int) 39,2 = 39.

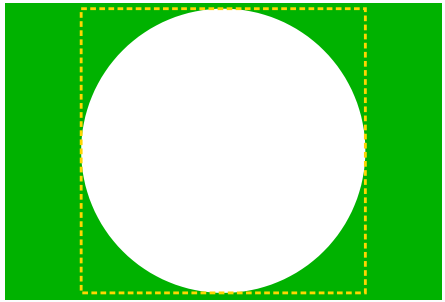
Adım 6 / 9

DEĞİŞKENLER

- `getWidth()` = **384**
- `getHeight()` = **261**
- `c` = **(192, 130)**
- `r` = **125**
- `r2` = **112**
- `bw` = **156**
- `bh` = **39**

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
▶  g.setColor(Color.WHITE);
▶  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



`fillOval(67, 5, 250, 250)` : beyaz daire, sınırlayıcı kare kesikli çizgide.

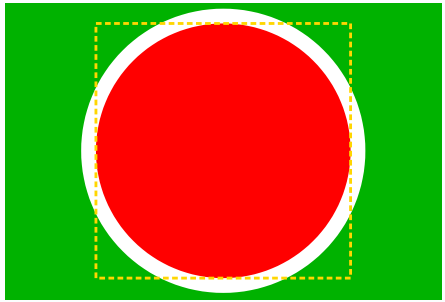
Adım 7 / 9

DEĞİŞKENLER

- `getWidth()` = **384**
- `getHeight()` = **261**
- `c` = **(192, 130)**
- `r` = **125**
- `r2` = **112**
- `bw` = **156**
- `bh` = **39**

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
▶  g.setColor(Color.RED);
▶  g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
11 g.setColor(Color.WHITE);
12 g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



`fillOval(80, 18, 224, 224)` : kırmızı daire beyazın üstüne boyanır; 13 piksellik beyaz halka kalır.

Adım 8 / 9

DEĞİŞKENLER

`getWidth()` = **384**

`getHeight()` = **261**

`c` = **(192, 130)** `r` = **125**

`r2` = **112** `bw` = **156** `bh` = **39**

İşareti Çizme: `paintComponent` İzleme

```
1  super.paintComponent(g);
2  var c = new Point(getWidth() / 2, getHeight() / 2);
3  var r = Math.min(getWidth() / 2, getHeight() / 2) - 5;
4  var r2 = (int) (r * 0.9);
5  var bw = (int) (r2 * 1.4);
6  var bh = (int) (r2 * 0.35);
7  g.setColor(Color.WHITE);
8  g.fillOval(c.x - r, c.y - r, 2 * r, 2 * r);
9  g.setColor(Color.RED);
10 g.fillOval(c.x - r2, c.y - r2, r2 * 2, r2 * 2);
▶  g.setColor(Color.WHITE);
▶  g.fillRect(c.x - bw / 2, c.y - bh / 2, bw, bh);
```



`fillRect(114, 111, 156, 39)` : beyaz bar en üste. İşaret tamam.

Adım 9 / 9

DEĞİŞKENLER

`getWidth() = 384`

`getHeight() = 261`

`c = (192, 130)` `r = 125`

`r2 = 112` `bw = 156` `bh = 39`

main ve Pencere

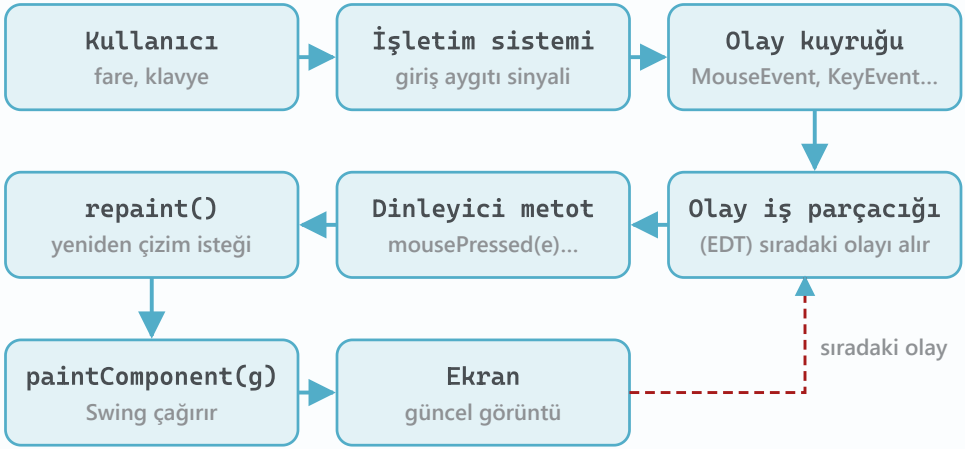
```java
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        var panel = new GirisYasaktirIsareti();
        panel.setBackground(Color.GREEN.darker());
        var frame = new JFrame("Basit bir grafik programı");
        frame.setSize(400, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add(panel, BorderLayout.CENTER);
        frame.setVisible(true);
    });
}
```

- `SwingUtilities.invokeLater(() -> { … })` : arayüzü **olay iş parçacığında** kurar.
- `() -> { … }` bir **lambda ifadesidir** (Java 8+): "sonra çalıştırılacak kod parçası".
- `Color.GREEN.darker()` = (0, 178, 0): saf yeşilden biraz koyu.
- Panel, pencerenin içerik bölmesinin **merkezine** eklenir; tek bileşen olduğu için tüm alanı kaplar.
- 400 × 300 pencerede kenarlık ve başlık çubuğu düşülünce panel yaklaşık 384 × 261 olur (işletim sistemine göre değişir).

FARE OLAYLARI

Etkileşimli Bilgisayar Grafikleri

- **Bilgisayar grafikleri**, görsel içerik oluşturma ve işleme sürecidir. **Etkileşimli** (interactive) grafiklerde kullanıcı fare, klavye gibi giriş aygıtlarıyla görüntüyü değiştirir: çizim programları, oyunlar, haritalar.
- Grafik kütüphaneleri şekil çizmeyi, renklendirmeyi ve grafik nesnelerini değiştirmeyi sağlar; etkileşim ise aşağıdaki **olay döngüsü** ile gerçekleşir.



Dinleyici metot ekranı doğrudan boyamaz: veriyi günceller ve `repaint()` ile **yeniden çizim ister**. Swing uygun anda `paintComponent`'i çağırır.

FARE OLAYLARI

Dinleyici Arayüzü ve Adaptör

ARAYÜZ: BEŞ METODUN HEPSİ YAZILMALI

```
class Dinleyici implements MouseListener {
    public void mouseClicked(MouseEvent e) {
        System.out.println("tık");
    }
    public void mousePressed(MouseEvent e) { }
    public void mouseReleased(MouseEvent e) { }
    public void mouseEntered(MouseEvent e) { }
    public void mouseExited(MouseEvent e) { }
}
```

`MouseAdapter` : YALNIZ GEREKENİ GEÇERSİZ KIL

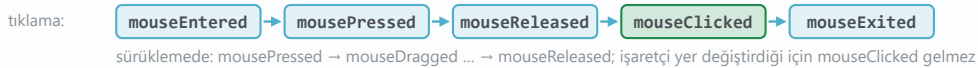
```
class Dinleyici extends MouseAdapter {
    @Override
    public void mouseClicked(MouseEvent e) {
        System.out.println("tık");
    }
}
// panel.addMouseListener(new Dinleyici());
```

- `MouseAdapter`, `MouseListener` (ve `MouseMotionListener`, `MouseWheelListener`) arayüzünü **boş gövdelerle** uygulayan hazır bir sınıftır. Ondan türeyen sınıf yalnızca ilgilendiği metodu yazar.
- Hareket olayları ayrı bir arayüzdedir: `MouseMotionListener` (`mouseDragged`, `mouseMoved`); adaptörü `MouseMotionAdapter`. Kayıt: `addMouseListener(…)`, `addMouseMotionListener(…)`.
- `new MouseAdapter() { … }` biçimi bir **anonim iç sınıftır** (anonymous inner class): adı olmayan, tek kullanımlık bir alt sınıf.

FARE OLAYLARI

Fare Olay Metotları

Metot	Arayüz	Ne zaman çağrılır?
`mousePressed(e)`	`MouseListener`	Fare düğmesine basıldığı an
`mouseReleased(e)`	`MouseListener`	Düğme bırakıldığında
`mouseClicked(e)`	`MouseListener`	Basıp **aynı yerde** bırakınca (pressed → released → clicked)
`mouseEntered(e)` / `mouseExited(e)`	`MouseListener`	İşaretçi bileşenin içine girince / dışına çıkınca
`mouseMoved(e)`	`MouseMotionListener`	Düğmeye basmadan hareket edince
`mouseDragged(e)`	`MouseMotionListener`	Düğme basılıyken hareket edince (sürükleme)
`mouseWheelMoved(e)`	`MouseWheelListener`	Tekerlek döndürülünce



Her metot bir `MouseEvent e` parametresi alır: olayın **nerede, hangi düğmeyle, hangi tuşlar basılıyken** olduğu bu nesnededir.

FARE OLAYLARI

MouseEvent Sınıfı

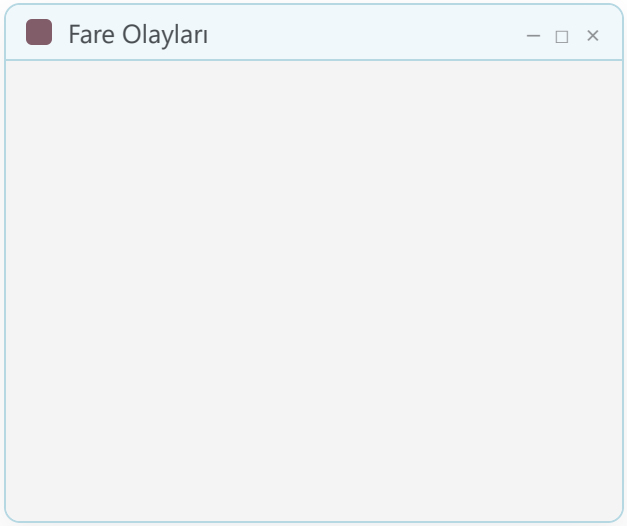
Metot	Döndürdüğü
`getX()`, `getY()`	İşaretçinin bileşene göre konumu
`getPoint()`	Aynı konum, `Point` olarak
`getButton()`	`BUTTON1` sol, `BUTTON2` orta, `BUTTON3` sağ
`getClickCount()`	Art arda tıklama sayısı (2: çift tık)
`isShiftDown()`, `isCtrlDown()`, `isAltDown()`	O anda tuş basılı mı?

```
panel.addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        String dugme = "sol";
        if (e.getButton() == MouseEvent.BUTTON3) {
            dugme = "sağ";
        }
        System.out.println(dugme + " (" + e.getX()
                + ", " + e.getY() + ")"
                + (e.isShiftDown() ? " + Shift" : ""));
    }
});
```

Fare olayı sonrasında oluşan durumu temsil eder. Koordinatlar **ekrana değil bileşene** göredir: panelin sol üst köşesi (0, 0).

FARE OLAYLARI

Fare Olayları Deneme Alanı



Panelde tıklama ve sürükleme olay üretir (Shift veya Ctrl basılıyken de).

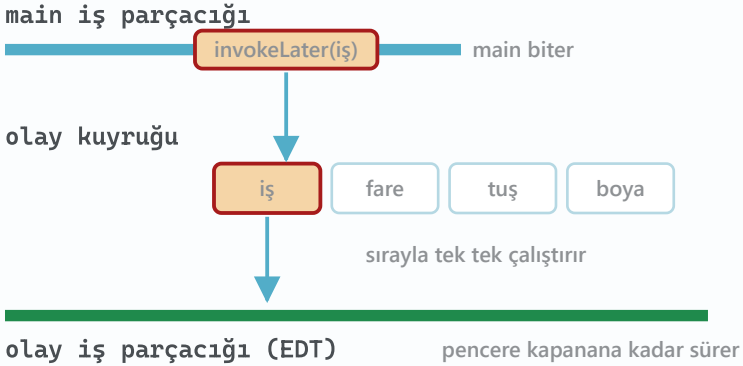
JAVA ÇAĞRISI VE `MouseEvent` DEĞERLERİ

```
(henüz olay yok)
```

Tarayıcı olayları Java'daki karşılıklarıyla gösterilir. Bir tıklama **pressed** → **released** → **clicked** sırasıyla üç olay üretir; sürüklerken ise çok sayıda `mouseDragged` gelir. Sağ tık `BUTTON3` 'tür.

SwingUtilities ve Olay İş Parçacığı

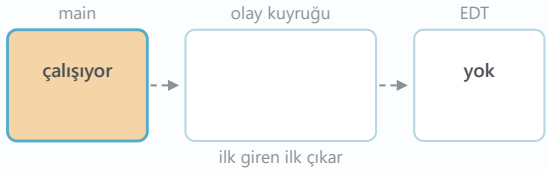
- `javax.swing.SwingUtilities` yardımcı metotlar içerir.
- Swing bileşenleri **iş parçacığı güvenli değildir** (not thread-safe): aynı anda birden çok iş parçacığı dokunursa tutarsızlık olabilir.
- Kural: bileşenler yalnızca **olay iş parçacığından** (Event Dispatch Thread, EDT) kurulur ve değiştirilir. Dinleyici metotlar zaten EDT'de çalışır.
- Benzetme: mutfakta tek bir aşçı (EDT) vardır; diğerleri yemeği kendileri yapmaz, **sipariş fişi** (`Runnable`) bırakır.



Metot	Davranış
`invokeLater(iş)`	İşi kuyruğa koyar ve **hemen döner** (asenkron). GUI başlatmanın standart yolu.
`invokeAndWait(iş)`	İş EDT'de **bitene kadar bekler** (senkron). EDT'nin içinden çağrılırsa `Error` fırlatır (EDT kendini bekleyemez).

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
▶    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
   System.out.println(
9        "5 iş C, EDT mi? " +
   SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



Program `main` iş parçacığında başlar. Olay iş parçacığı (EDT) henüz yok.

Adım 1 / 9

DEĞİŞKENLER

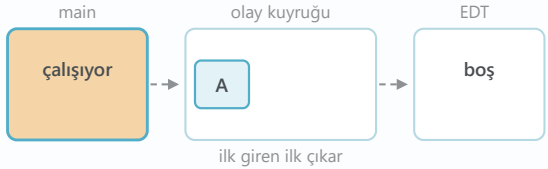
`iş parçacığı = main`

ÇIKTI

```
1 main
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
▶    SwingUtilities.invokeLater(
▶        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
   System.out.println(
9        "5 iş C, EDT mi? " +
   SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



`invokeLater` iş A'yı kuyruğa koyar ve **hemen döner**. EDT başlatılır.

Adım 2 / 9

DEĞİŞKENLER

`kuyruk = [A]`

ÇIKTI

```
1 main
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
▶    SwingUtilities.invokeLater(
▶        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
   System.out.println(
9        "5 iş C, EDT mi? " +
   SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



İş B de kuyruğa eklenir; `main` yine beklemez.

Adım 3 / 9

DEĞİŞKENLER

`kuyruk = [A, B]`

ÇIKTI

```
1 main
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
▶    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
   System.out.println(
9        "5 iş C, EDT mi? " +
   SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



`main` kendi işine devam eder. (Bu çalıştırmada EDT henüz bir iş almadı; iki iş parçacığı bağımsızdır.)

Adım 4 / 9

DEĞİŞKENLER

`kuyruk = [A, B]`

ÇIKTI

```
1 main
2 main devam ediyor
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
▶    SwingUtilities.invokeAndWait(() ->
     System.out.println(
▶        "5 iş C, EDT mi? " +
     SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



`invokeAndWait` iş C'yi kuyruğa koyar ve `main`'i **durdurur**.

Adım 5 / 9

DEĞİŞKENLER

`kuyruk = [A, B, C]`

ÇIKTI

```
1 main
2 main devam ediyor
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
▶        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
       System.out.println(
9        "5 iş C, EDT mi? " +
       SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



EDT kuyruğun başındaki iş A'yı çalıştırır (iş parçacığının adı `AWT-EventQueue-0`).

Adım 6 / 9

DEĞİŞKENLER

`kuyruk = [B, C]`

ÇIKTI

```
1 main
2 main devam ediyor
3 iş A
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
▶        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
   System.out.println(
9        "5 iş C, EDT mi? " +
   SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



Sıra korunur: iş B, iş A'dan sonra.

Adım 7 / 9

DEĞİŞKENLER

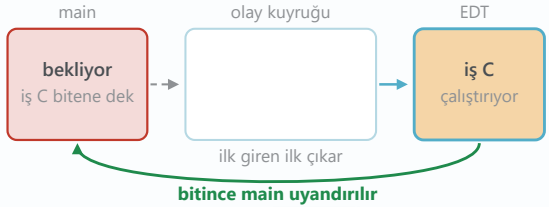
`kuyruk = [C]`

ÇIKTI

```
1 main
2 main devam ediyor
3 iş A
4 iş B
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
▶    SwingUtilities.invokeAndWait(() ->
     System.out.println(
▶        "5 iş C, EDT mi? " +
     SwingUtilities.isEventDispatchThread()));
10   System.out.println("6 main, EDT mi? "
11       + SwingUtilities.isEventDispatchThread());
12 }
```



İş C de EDT'de çalışır (`true`). Bitince bekleyen `main` uyandırılır.

Adım 8 / 9

DEĞİŞKENLER

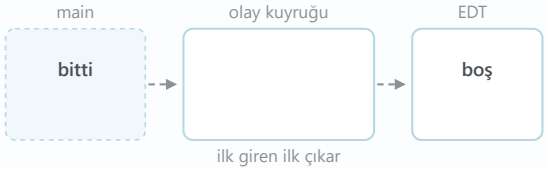
`kuyruk = []`

ÇIKTI

```
1 main
2 main devam ediyor
3 iş A
4 iş B
5 iş C, EDT mi? true
```

Olay Kuyruğu: `invokeLater` ve `invokeAndWait`

```
1  public static void main(String[] a) throws Exception {
2    System.out.println("1 main");
3    SwingUtilities.invokeLater(
4        () -> System.out.println("3 iş A"));
5    SwingUtilities.invokeLater(
6        () -> System.out.println("4 iş B"));
7    System.out.println("2 main devam ediyor");
8    SwingUtilities.invokeAndWait(() ->
   System.out.println(
9        "5 iş C, EDT mi? " +
   SwingUtilities.isEventDispatchThread()));
▶    System.out.println("6 main, EDT mi? "
▶        + SwingUtilities.isEventDispatchThread());
12 }
```



`main` EDT değildir (`false`). Bileşenlere bu yüzden yalnız EDT'den dokunulur.

Adım 9 / 9

DEĞİŞKENLER

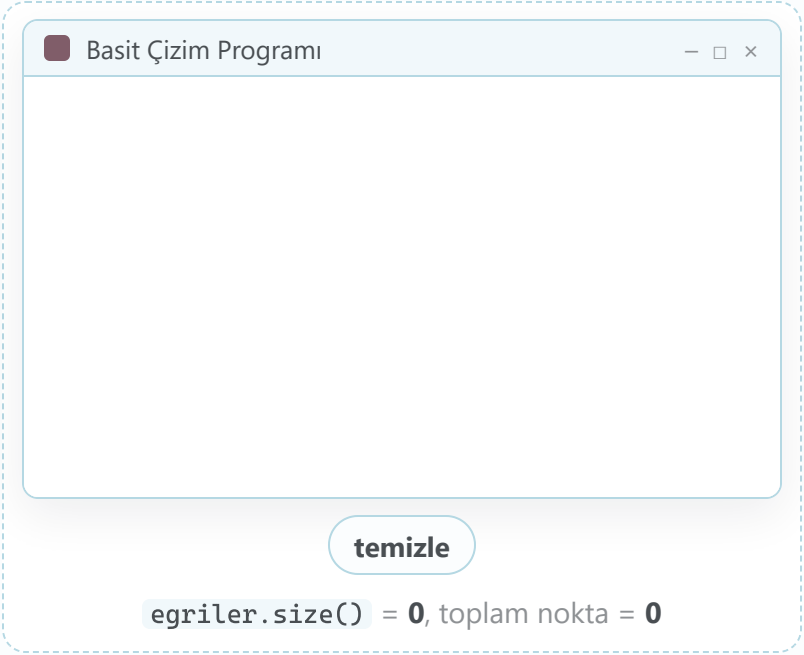
`iş parçacığı = main`

ÇIKTI

```
1 main
2 main devam ediyor
3 iş A
4 iş B
5 iş C, EDT mi? true
6 main, EDT mi? false
```

ÇİZİM PROGRAMI

Basit Çizim Programı



Fareyle basılı tutup sürükleyerek çizim yapılan program.

- **Basınca** (`mousePressed`) yeni bir eğri başlar.
- **Sürükledikçe** (`mouseDragged`) eğrinin sonuna nokta eklenir ve `repaint` istenir.
- **Çizerken** (`paintComponent`) her eğrinin ardışık noktaları çizgilerle birleştirilir.

Program çizimi bir resim olarak değil, **noktaların listesi** olarak saklar; ekran her seferinde bu veriden yeniden çizilir.

Eğrileri Saklamak

```
public class BasitCizimPaneli extends JPanel {
    private List<List<Point>> egriler = new ArrayList<>();

    public BasitCizimPaneli() {
        addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                var yeniEgri = new ArrayList<Point>();
                yeniEgri.add(new Point(e.getX(), e.getY()));
                egriler.add(yeniEgri);
            }
        });
        addMouseMotionListener(new MouseMotionAdapter() {
            @Override
            public void mouseDragged(MouseEvent e) {
                egriler.get(egriler.size() - 1)
                       .add(new Point(e.getX(), e.getY()));
                repaint(0, 0, getWidth(), getHeight());
            }
        });
    }
}
```

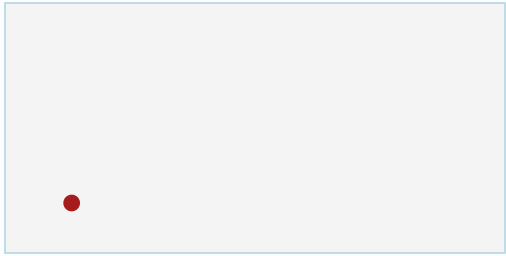


`eğri 0:` `(40,120)` `(60,60)` `(80,120)`
`eğri 1:` `(50,95)` `(70,95)`

- `List<List<Point>>`: **listelerin listesi**. Dış liste eğrileri, her iç liste bir eğrinin noktalarını tutar.
- Son eğri: `egriler.get(egriler.size() - 1)`. Son öğe indisi `size() - 1`'dir; `get(size())` `IndexOutOfBoundsException` fırlatır.

Fare Olaylarını İzleme

```
1  addMouseListener(new MouseAdapter() {
2      @Override
3      public void mousePressed(MouseEvent e) {
▶          var yeniEgri = new ArrayList<Point>();
▶          yeniEgri.add(new Point(e.getX(), e.getY()));
▶          egriler.add(yeniEgri);
7      }
8  });
9  addMouseMotionListener(new MouseMotionAdapter() {
10     @Override
11     public void mouseDragged(MouseEvent e) {
12         egriler.get(egriler.size() - 1)
13                .add(new Point(e.getX(), e.getY()));
14         repaint(0, 0, getWidth(), getHeight());
15     }
16 });
```



`eğri 0:` `(40,120)`

mousePressed: Basıldı (40, 120): yeni bir iç liste oluşur, ilk noktası eklenir, dış listeye konur.

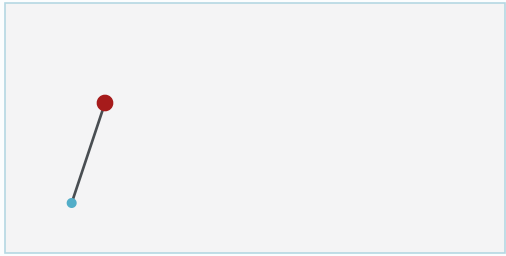
Adım 1 / 6

DEĞİŞKENLER

`e.getX()` = **40** `e.getY()` = **120** `egriler.size()` = **1**

Fare Olaylarını İzleme

```
1  addMouseListener(new MouseAdapter() {
2      @Override
3      public void mousePressed(MouseEvent e) {
4          var yeniEgri = new ArrayList<Point>();
5          yeniEgri.add(new Point(e.getX(), e.getY()));
6          egriler.add(yeniEgri);
7      }
8  });
9  addMouseMotionListener(new MouseMotionAdapter() {
10     @Override
11     public void mouseDragged(MouseEvent e) {
▶          egriler.get(egriler.size() - 1)
▶                 .add(new Point(e.getX(), e.getY()));
▶          repaint(0, 0, getWidth(), getHeight());
15     }
16 });
```



eğri 0: (40,120) (60,60)

mouseDragged: Sürüklendi (60, 60): son eğriye nokta eklenir; `repaint` yeniden çizimi ister, çizgi görünür.

Adım 2 / 6

DEĞİŞKENLER

`e.getX() = 60` `e.getY() = 60` `egriler.size() = 1`

Fare Olaylarını İzleme

```
1  addMouseListener(new MouseAdapter() {
2      @Override
3      public void mousePressed(MouseEvent e) {
4          var yeniEgri = new ArrayList<Point>();
5          yeniEgri.add(new Point(e.getX(), e.getY()));
6          egriler.add(yeniEgri);
7      }
8  });
9  addMouseMotionListener(new MouseMotionAdapter() {
10     @Override
11     public void mouseDragged(MouseEvent e) {
▶          egriler.get(egriler.size() - 1)
▶                 .add(new Point(e.getX(), e.getY()));
▶          repaint(0, 0, getWidth(), getHeight());
15     }
16 });
```



eğri 0: (40,120) (60,60) (80,120)

mouseDragged: Sürüklendi (80, 120): aynı eğri uzar. Gerçekte saniyede onlarca `mouseDragged` gelir.

Adım 3 / 6

DEĞİŞKENLER

`e.getX() = 80` `e.getY() = 120` `egriler.size() = 1`

Fare Olaylarını İzleme

```
addMouseListener(new MouseAdapter() {
    @Override
    public void mousePressed(MouseEvent e) {
        var yeniEgri = new ArrayList<Point>();
        yeniEgri.add(new Point(e.getX(), e.getY()));
        egriler.add(yeniEgri);
    }
});
addMouseMotionListener(new MouseMotionAdapter() {
    @Override
    public void mouseDragged(MouseEvent e) {
        egriler.get(egriler.size() - 1)
               .add(new Point(e.getX(), e.getY()));
        repaint(0, 0, getWidth(), getHeight());
    }
});
```



eğri 0: `(40,120)` `(60,60)` `(80,120)`

mouseReleased: Bırakıldı: `mouseReleased`'ı yazmadık; `MouseAdapter`'ın **boş gövdesi** çalışır, hiçbir şey olmaz.

Adım 4 / 6

DEĞİŞKENLER

`e.getX() = 80` `e.getY() = 120` `egriler.size() = 1`

Fare Olaylarını İzleme

```
1  addMouseListener(new MouseAdapter() {
2      @Override
3      public void mousePressed(MouseEvent e) {
           var yeniEgri = new ArrayList<Point>();
           yeniEgri.add(new Point(e.getX(), e.getY()));
           egriler.add(yeniEgri);
7      }
8  });
9  addMouseMotionListener(new MouseMotionAdapter() {
10     @Override
11     public void mouseDragged(MouseEvent e) {
12         egriler.get(egriler.size() - 1)
13                .add(new Point(e.getX(), e.getY()));
14         repaint(0, 0, getWidth(), getHeight());
15     }
16 });
```



eğri 0: (40,120) (60,60) (80,120)

eğri 1: (50,95)

mousePressed: Yeniden basıldı (50, 95): **ikinci** eğri başlar. Eski eğri olduğu gibi kalır.

Adım 5 / 6

DEĞİŞKENLER

`e.getX() = 50` `e.getY() = 95` `egriler.size() = 2`

Fare Olaylarını İzleme

```
1  addMouseListener(new MouseAdapter() {
2      @Override
3      public void mousePressed(MouseEvent e) {
4          var yeniEgri = new ArrayList<Point>();
5          yeniEgri.add(new Point(e.getX(), e.getY()));
6          egriler.add(yeniEgri);
7      }
8  });
9  addMouseMotionListener(new MouseMotionAdapter() {
10     @Override
11     public void mouseDragged(MouseEvent e) {
▶          egriler.get(egriler.size() - 1)
▶                 .add(new Point(e.getX(), e.getY()));
▶          repaint(0, 0, getWidth(), getHeight());
15     }
16 });
```



eğri 0: (40,120) (60,60) (80,120)

eğri 1: (50,95) (70,95)

mouseDragged: Sürüklendi (70, 95): ekleme yine **son** eğriye, yani eğri 1'e yapılır.

Adım 6 / 6

DEĞİŞKENLER

`e.getX() = 70` `e.getY() = 95` `egriler.size() = 2`

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
▶       super.paintComponent(g);
4      for (var egri : egriler) {
5          var onceki = egri.get(0);
6          for (var nokta : egri) {
7              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
8              onceki = nokta;
9          }
10     }
11 }
```


Zemin temizlenir; önceki çizimden iz kalmaz.

Adım 1 / 8

DEĞİŞKENLER

—

ÇIKTI

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
▶      for (var egri : egriler) {
▶          var onceki = egri.get(0);
6          for (var nokta : egri) {
7              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
8              onceki = nokta;
9          }
10     }
11 }
```

DEĞİŞKENLER

`egri` = **eğri 0** `onceki` = **(40,120)**

ÇIKTI



İlk eğri; `onceki` onun ilk noktası.

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
4      for (var egri : egriler) {
5          var onceki = egri.get(0);
▶          for (var nokta : egri) {
▶              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
▶              onceki = nokta;
9          }
10     }
11 }
```


İç döngü **ilk noktadan** başlar: kendisiyle birleşir, tek piksel çizilir. Ardından `onceki = nokta`.

Adım 3 / 8

DEĞİŞKENLER

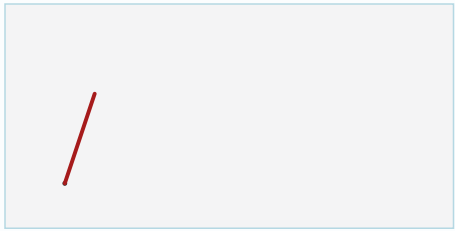
`egri` = **eğri 0** | `onceki` = **(40,120)** | `nokta` = **(40,120)**

ÇIKTI

```
drawLine(40, 120, 40, 120)
```

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
4      for (var egri : egriler) {
5          var onceki = egri.get(0);
▶          for (var nokta : egri) {
▶              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
▶              onceki = nokta;
9          }
10     }
11 }
```



(40,120) → (60,60) çizgisi.

Adım 4 / 8

DEĞİŞKENLER

`egri` = **eğri 0** | `onceki` = **(40,120)** | `nokta` = **(60,60)**

ÇIKTI

```
drawLine(40, 120, 40, 120)
drawLine(40, 120, 60, 60)
```

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
4      for (var egri : egriler) {
5          var onceki = egri.get(0);
▶          for (var nokta : egri) {
▶              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
▶              onceki = nokta;
9          }
10     }
11 }
```

DEĞİŞKENLER

`egri` = **eğri 0** | `onceki` = **(60,60)** | `nokta` = **(80,120)**

ÇIKTI

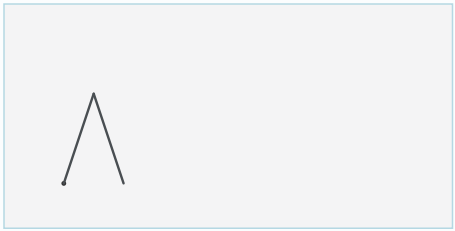
```
drawLine(40, 120, 40, 120)
drawLine(40, 120, 60, 60)
drawLine(60, 60, 80, 120)
```



(60,60) → (80,120). Eğri 0 bitti.

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
▶      for (var egri : egriler) {
▶          var onceki = egri.get(0);
6          for (var nokta : egri) {
7              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
8              onceki = nokta;
9          }
10     }
11 }
```



Dış döngü ikinci eğriye geçer; `onceki` yeniden onun ilk noktası olur. Böylece iki eğri **birleşmez**.

Adım 6 / 8

DEĞİŞKENLER

`egri = eğri 1` `onceki = (50,95)`

ÇIKTI

```
drawLine(40, 120, 40, 120)
drawLine(40, 120, 60, 60)
drawLine(60, 60, 80, 120)
```

Eğrileri Çizmek: `paintComponent`

```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
4      for (var egri : egriler) {
5          var onceki = egri.get(0);
▶          for (var nokta : egri) {
▶              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
▶              onceki = nokta;
9          }
10     }
11 }
```

DEĞİŞKENLER

`egri` = **eğri 1** | `onceki` = **(50,95)** | `nokta` = **(50,95)**

ÇIKTI

```
drawLine(40, 120, 40, 120)
drawLine(40, 120, 60, 60)
drawLine(60, 60, 80, 120)
drawLine(50, 95, 50, 95)
```



Yine tek piksel.

Eğrileri Çizmek: `paintComponent`

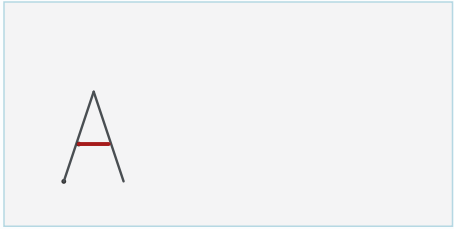
```
1  @Override
2  public void paintComponent(Graphics g) {
3      super.paintComponent(g);
4      for (var egri : egriler) {
5          var onceki = egri.get(0);
▶          for (var nokta : egri) {
▶              g.drawLine(onceki.x, onceki.y, nokta.x, nokta.y);
▶              onceki = nokta;
9          }
10     }
11 }
```

DEĞİŞKENLER

`egri` = **eğri 1** `onceki` = **(50,95)** `nokta` = **(70,95)**

ÇIKTI

```
drawLine(40, 120, 40, 120)
drawLine(40, 120, 60, 60)
drawLine(60, 60, 80, 120)
drawLine(50, 95, 50, 95)
drawLine(50, 95, 70, 95)
```



(50,95) → (70,95). Beş `drawLine` çağrısıyla çizim tamam.

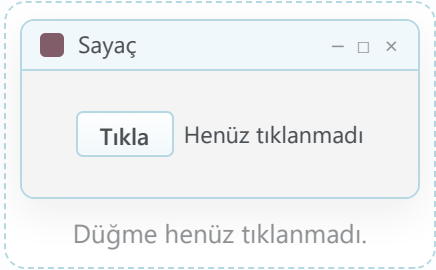
main ve repaint

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        var frame = new JFrame("Basit Çizim Programı");
        frame.getContentPane().add(new BasitCizimPaneli(),
                BorderLayout.CENTER);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);
        frame.setVisible(true);
    });
}
```

- `repaint()` çizimi **hemen yapmaz**: kuyruğa bir "yeniden boya" isteği koyar. Art arda gelen istekler birleştirilebilir.
- `repaint(x, y, en, boy)` yalnızca o dikdörtgenin yeniden çizilmesini ister. Swing bunun dışını **kırpar** (clipping): `g` o bölgenin dışına boyamaz, iş azalır. `repaint(0, 0, getWidth(), getHeight())` = `repaint()`.
- `super.paintComponent(g)` unutulursa zemin temizlenmez; eski görüntü artıkları kalabilir.
- `paintComponent`'i kendimiz **çağırmayız**; veriyi değiştirip `repaint()` isteriz.

JButton Sınıfı

- Tıklanabilir **düğmeler** oluşturur; metin, ikon ve olay işleyicilerle özelleştirilir.
- `setText()` metni, `setIcon()` ikonu belirler; `setEnabled(false)` pasif yapar.
- `addActionListener()` tıklanınca çalışacak dinleyiciyi ekler (`ActionListener`, tek metodu `actionPerformed`). Dinleyici eklemek **Gözlemci** (Observer) desenidir: düğme konu, dinleyiciler gözlemcidir.

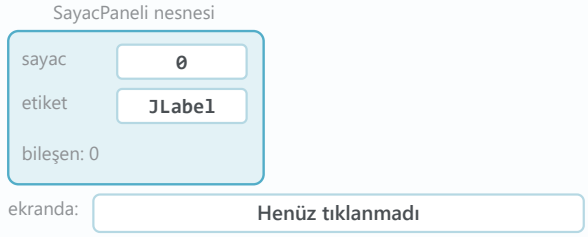



```
class SayacPaneli extends JPanel {
    private int sayac = 0;
    private JLabel etiket = new JLabel("Henüz tıklanmadı");

    SayacPaneli() {
        JButton dugme = new JButton("Tıkla");
        dugme.addActionListener(e -> {
            sayac++;
            etiket.setText(sayac + " kez tıklandı");
        });
        add(dugme);
        add(etiket);
    }
}
```

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
▶      private int sayac = 0;
▶      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
8              sayac++;
9              etiket.setText(sayac + " kez tıklandı");
10         });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



Nesne oluşurken önce alanlara ilk değerleri verilir: `sayac` 0, etiket hazır.

Adım 1 / 8

DEĞİŞKENLER

`sayac = 0` `etiket = JLabel`

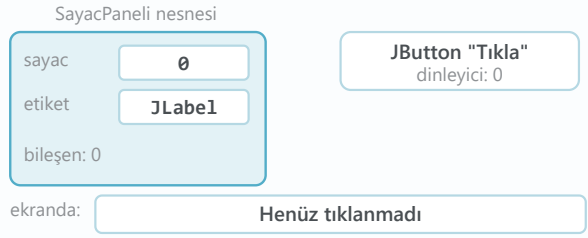
ÇAĞRI YIĞINI

`new SayacPaneli()`

`main()`

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
▶          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
8              sayac++;
9              etiket.setText(sayac + " kez tıklandı");
10         });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



Yapıcı bir düğme oluşturur; `dugme` yerel değişkendir.

Adım 2 / 8

DEĞİŞKENLER

`dugme = JButton("Tıkla")`

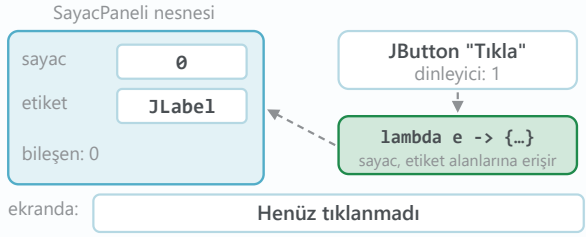
ÇAĞRI YIĞINI

`SayacPaneli()`

`main()`

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
▶          dugme.addActionListener(e -> {
▶              sayac++;
▶              etiket.setText(sayac + " kez tıklandı");
▶          });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



Lambda **çalıştırılmaz**, yalnızca düğmeye kaydedilir. Gövdesi tıklanınca çalışacak.

Adım 3 / 8

DEĞİŞKENLER

`dinleyici sayısı = 1`

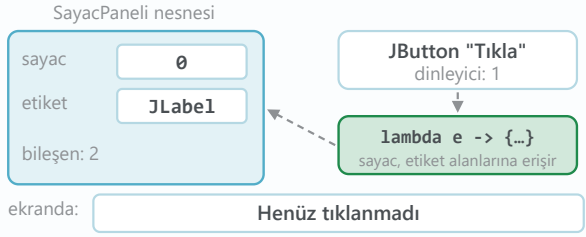
ÇAĞRI YIĞINI

`SayacPaneli()`

`main()`

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
8              sayac++;
9              etiket.setText(sayac + " kez tıklandı");
10         });
▶          add(dugme);
▶          add(etiket);
13     }
14 }
```



Düğme ve etiket panele eklenir (`FlowLayout` : yan yana). Yapıcı biter.

Adım 4 / 8

DEĞİŞKENLER

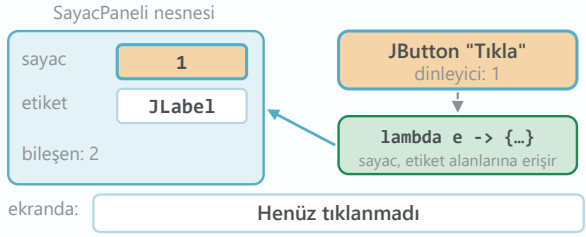
`bileşen sayısı = 2`

ÇAĞRI YIĞINI

`main()`

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
▶              sayac++;
9              etiket.setText(sayac + " kez tıklandı");
10         });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



Kullanıcı tıklar: EDT `actionPerformed`'ı, yani lambda gövdesini çağırır. `sayac` bir **alan** olduğu için artırılabilir.

Adım 5 / 8

DEĞİŞKENLER

`sayac = 1`

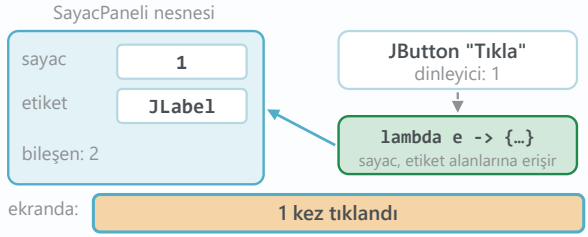
ÇAĞRI YIĞINI

`actionPerformed(e)`

EDT

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
8              sayac++;
▶              etiket.setText(sayac + " kez tıklandı");
10         });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



`setText` etiketin metnini değiştirir; Swing etiketi yeniden boyar.

Adım 6 / 8

DEĞİŞKENLER

`sayac = 1`

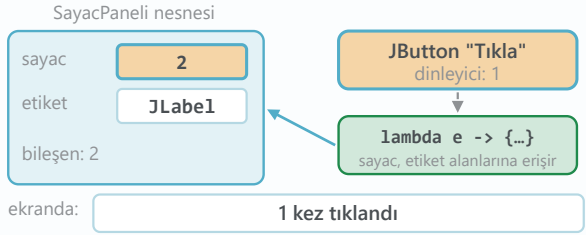
ÇAĞRI YIĞINI

`actionPerformed(e)`

EDT

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
▶              sayac++;
9              etiket.setText(sayac + " kez tıklandı");
10         });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



İkinci tıklama: aynı lambda yeniden çağrılır, alan 2 olur.

Adım 7 / 8

DEĞİŞKENLER

`sayac = 2`

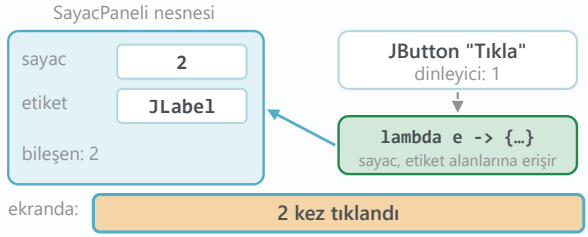
ÇAĞRI YIĞINI

`actionPerformed(e)`

EDT

Sayaç Düğmesi: Tıklama Adım Adım

```
1  class SayacPaneli extends JPanel {
2      private int sayac = 0;
3      private JLabel etiket = new JLabel("Henüz
   tıklanmadı");
4
5      SayacPaneli() {
6          JButton dugme = new JButton("Tıkla");
7          dugme.addActionListener(e -> {
8              sayac++;
▶              etiket.setText(sayac + " kez tıklandı");
10         });
11         add(dugme);
12         add(etiket);
13     }
14 }
```



Etiket güncellenir. Tıklamalar arasında program hiçbir kod çalıştırmaz; yalnızca **olay bekler**.

Adım 8 / 8

DEĞİŞKENLER

`sayac = 2`

ÇAĞRI YIĞINI

`actionPerformed(e)`

EDT

METİN BİLEŞENLERİ

ActionListener: Anonim Sınıf ve Lambda

ANONİM İÇ SINIF (KLASİK)

```
dugme.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        sayac++;
        etiket.setText("Sayı: " + sayac);
    }
});
```

LAMBDA İFADESİ (JAVA 8+)

```
dugme.addActionListener(e -> {
    sayac++;
    etiket.setText("Sayı: " + sayac);
});

// tek ifade ise süslü parantez gerekmez:
dugme.addActionListener(e -> System.exit(0));
```

- `ActionListener` yalnızca **tek** soyut metodu olan bir arayüzdür (fonksiyonel arayüz). Bu yüzden derleyici `e -> …` lambdasını `actionPerformed` 'ın gövdesi olarak kabul eder.
- `MouseListener` 'ın beş metodu olduğu için lambda ile yazılamaz; orada adaptör sınıfı kullanılır.
- Lambda içinden **yerel değişken değiştirilemez**: `int sayac = 0; … e -> sayac++` derlenmez. Sayaç gibi değişen değerler **alan** (field) olmalıdır.

METİN BİLEŞENLERİ

JLabel ve JTextField

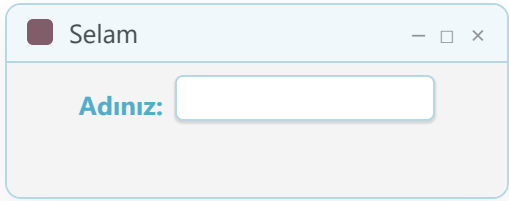
`JLabel` : etiket

Metin veya resim **gösterir**; kullanıcı değiştiremez. `setText()` metni, `setIcon()` resmi, `setFont()` yazı tipini, `setForeground()` metin rengini belirler.

`JTextField` : tek satırlık metin alanı

Kullanıcının **metin girmesi** için; formlardan veri toplanır. `getText()` / `setText()` okur ve yazar, `setColumns()` görünen sütun sayısını belirler, `addActionListener()` Enter'a basılınca çalışır.

```
JLabel etiket = new JLabel("Adınız:");
etiket.setFont(new Font("SansSerif", Font.BOLD, 14));
etiket.setForeground(Color.BLUE);
JTextField adAlani = new JTextField(12);
JLabel selam = new JLabel(" ");
adAlani.addActionListener(e -> selam.setText(
        "Merhaba, " + adAlani.getText() + "!"));
```



Ad yazılıp Enter'a basılınca selam görünür.

METİN BİLEŞENLERİ

JTextArea ve JScrollPane

`JTextArea`: çok satırlı metin

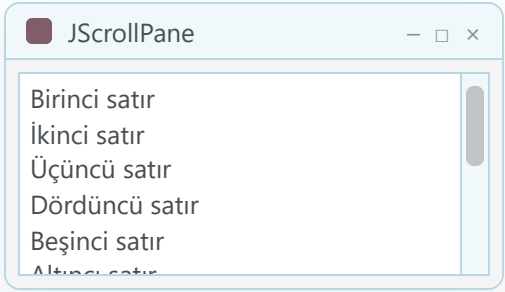
İçerik okunabilir, atanabilir, güncellenebilir. `new JTextArea(satır, sütun)` ile görünür boyut önceden verilir. `getText()`, `setText()`, `append()` (sona ekle), `setEditable(false)` (salt okunur), `setLineWrap(true)` (satır kaydır).

`JScrollPane`: kaydırma bölmesi

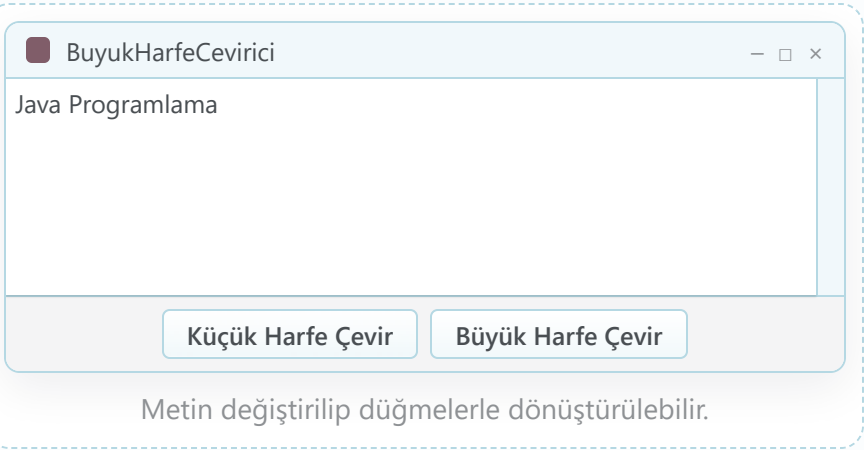
İçine konan bileşen (metin alanı, tablo, resim) ekrana sığmadığında **otomatik kaydırma çubukları** ekler; içerik boyutu değişince kendini uyarlar. `setPreferredSize()` ile boyutu, `setViewportView()` ile içeriği değiştirilir.

```
JTextArea alan = new JTextArea(5, 20);
alan.setText("Birinci satır\n");
alan.append("İkinci satır\n");
JScrollPane kaydirma = new JScrollPane(alan);
System.out.print(alan.getText());
System.out.println(alan.getLineCount());
```

```
Birinci satır
İkinci satır
3
```



Metin Dönüştürücü: Pencere



Metin değiştirilip düğmelerle dönüştürülebilir.

- Ortada (`CENTER`) bir `JScrollPane` içinde 8 satır × 50 sütunluk `JTextArea` .
- Altta (`SOUTH`) iki düğmeyi taşıyan bir `JPanel` (`FlowLayout` : ortalı).
- `pack()` pencere boyutunu bu bileşenlerin tercih ettiği boyuttan hesaplar.
- Düğmeler metni `toLowerCase()` / `toUpperCase()` ile dönüştürüp alana geri yazar.

Not: `toUpperCase()` bilgisayarın yerel ayarını (locale) kullanır. İngilizce ayarlı bir makinede `"istanbul"` → `"ISTANBUL"` , Türkçe ayarlıda `"İSTANBUL"` olur. Sonucu sabitlemek için: `toUpperCase(Locale.forLanguageTag("tr"))` . Bu demo Türkçe kuralları kullanır.

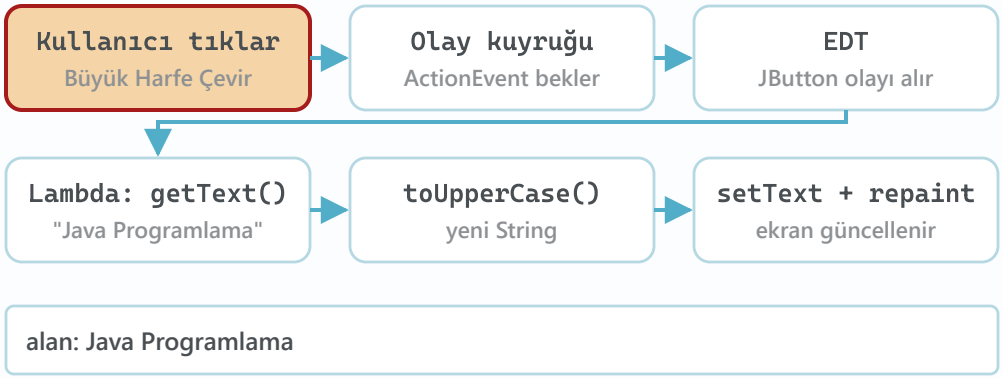
Metin Dönüştürücü: Kod

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

public class BuyukHarfeCevirici {
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> {
            var alan = new JTextArea("Java Programlama", 8, 50);
            var kucukHarfButton = new JButton("Küçük Harfe Çevir");
            var buyukHarfButton = new JButton("Büyük Harfe Çevir");
            kucukHarfButton.addActionListener(e ->
                    alan.setText(alan.getText().toLowerCase()));
            buyukHarfButton.addActionListener(e ->
                    alan.setText(alan.getText().toUpperCase()));
            var panel = new JPanel();
            panel.add(kucukHarfButton);
            panel.add(buyukHarfButton);
            var frame = new JFrame("BuyukHarfeCevirici");
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.getContentPane().add(new JScrollPane(alan),
                    BorderLayout.CENTER);
            frame.getContentPane().add(panel, BorderLayout.SOUTH);
            frame.pack();
            frame.setVisible(true);
        });
    }
}
```

- Tüm arayüz tek bir lambda içinde, EDT'de kurulur.
- `alan` yerel bir değişkendir ama lambda içinde yalnızca **okunduğu** (değiştirilmediği) için kullanılabilir (effectively final).
- `toUpperCase()` yeni bir `String` döndürür; `String` değiştirilemez olduğu için sonucu `setText` ile **geri yazmak** gerekir.
- `BorderLayout.CENTER` tam adıyla yazılır; yalnız `CENTER` derlenmez.

Bir Tıklamada Neler Olur?



Kullanıcı “Büyük Harfe Çevir” düğmesine tıklar; işletim sistemi bunu Java'ya iletir.

Bir Tıklamada Neler Olur?



Java bir olay nesnesi oluşturup **olay kuyruğuna** koyar.

Bir Tıklamada Neler Olur?



EDT sıradaki olayı alır; düğme, kayıtlı tüm `ActionListener`'ların `actionPerformed`'ını çağırır: bizim lambdamız çalışır.

Bir Tıklamada Neler Olur?



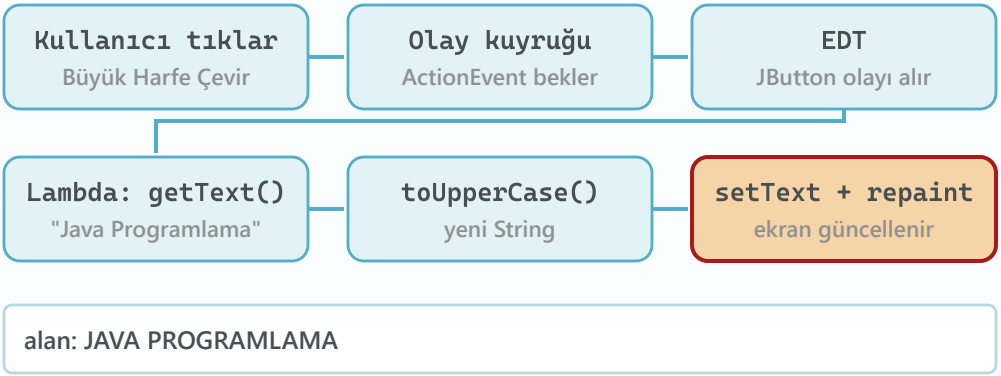
`alan.getText()` mevcut metni okur.

Bir Tıklamada Neler Olur?



`toUpperCase()` **yeni** bir dizgi üretir; eski dizgi değişmez.

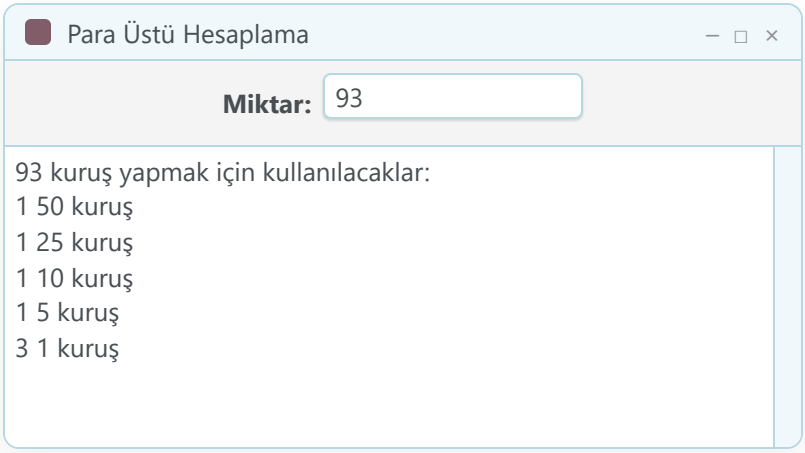
Bir Tıklamada Neler Olur?



`setText` alanın modelini günceller; alan kendini `repaint` ile yeniden çizdirir. EDT sıradaki olaya geçer.

PARA ÜSTÜ UYGULAMASI

Para Üstü Hesaplama



Girilen kuruş miktarının en az sayıda madeni parayla (50, 25, 10, 5, 1) nasıl ödeneceğini gösterir.

- Düğme yoktur: rapor her tuş vuruşunda **kendiliğinden** güncellenir.
- Bunu metin alanının **belgesine** (`Document`) eklenen bir `DocumentListener` sağlar.
- Örnek girdiler: 93, 287, boş alan, `abc` , `-7` .

PARA ÜSTÜ UYGULAMASI

Alanlar ve Yapıcı

```
public class ParaUstuHesaplama extends JFrame {
    private JTextField miktarAlani = new JTextField(12);
    private Document metin = miktarAlani.getDocument();
    private JTextArea rapor = new JTextArea(8, 40);

    public ParaUstuHesaplama() {
        var ustPanel = new JPanel();
        ustPanel.add(new JLabel("Miktar:"));
        ustPanel.add(miktarAlani);
        getContentPane().add(ustPanel, BorderLayout.NORTH);
        getContentPane().add(new JScrollPane(rapor),
                BorderLayout.CENTER);
        setBackground(Color.LIGHT_GRAY);
        rapor.setEditable(false);
        metin.addDocumentListener(new DocumentListener() {
            public void insertUpdate(DocumentEvent e) { raporuGuncelle(); }
            public void removeUpdate(DocumentEvent e) { raporuGuncelle(); }
            public void changedUpdate(DocumentEvent e) { raporuGuncelle(); }
        });
    }
```

- Sınıf `JFrame`'den **türer**. Swing'de veri ile görünüm ayrıdır (MVC): `JTextField` **görünüm**, `getDocument()` ile alınan `Document` metni tutan **model**.
- `DocumentListener` : `insertUpdate` (eklendi), `removeUpdate` (silindi), `changedUpdate` (stil değişti; düz metinde çağrılmaz). Arayüz olduğu için üçü de yazılır.

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
▶          var miktar = Integer.parseInt(
▶                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



Alandaki metin `"93"` okunur ve tam sayıya çevrilir.

DEĞİŞKENLER

`miktar = 93`

ÇIKTI

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
▶          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



kalan miktar: 93

Para değerleri büyükten küçüğe bir dizide.

Adım 2 / 13

DEĞİŞKENLER

`miktar = 93`

`dizi = [50, 25, 10, 5, 1]`

ÇIKTI

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
▶          rapor.setText(miktar
▶                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



`setText` raporu **sıfırdan** başlatır: eski içerik silinir.

Adım 3 / 13

DEĞİŞKENLER

`miktar = 93`

`dizi = [50, 25, 10, 5, 1]`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
▶              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



x = 50: 93 / 50 = **1** adet (tam sayı bölmesi), satır sona eklenir.

DEĞİŞKENLER

`miktar = 93` `x = 50`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
▶              miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



Kalan: 93 % 50 = 43.

Adım 5 / 13

DEĞİŞKENLER

`miktar = 43` `x = 50`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
▶              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



x = 25: 43 / 25 = **1** adet (tam sayı bölmesi), satır sona eklenir.

Adım 6 / 13

DEĞİŞKENLER

`miktar = 43` `x = 25`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
▶              miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



Kalan: 43 % 25 = 18.

Adım 7 / 13

DEĞİŞKENLER

`miktar = 18` `x = 25`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
▶              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



kalan miktar: 18

x = 10: 18 / 10 = **1** adet (tam sayı bölmesi), satır sona eklenir.

Adım 8 / 13

DEĞİŞKENLER

`miktar = 18` `x = 10`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
1 10 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
▶              miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



Kalan: 18 % 10 = 8.

Adım 9 / 13

DEĞİŞKENLER

`miktar = 8` `x = 10`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
1 10 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
▶              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



x = 5: 8 / 5 = **1** adet (tam sayı bölmesi), satır sona eklenir.

Adım 10 / 13

DEĞİŞKENLER

`miktar = 8` `x = 5`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
1 10 kuruş
1 5 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
▶              miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



Kalan: 8 % 5 = 3.

Adım 11 / 13

DEĞİŞKENLER

`miktar = 3` `x = 5`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
1 10 kuruş
1 5 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
▶              rapor.append(miktar / x + " " + x + "
   kuruş\n");
10             miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



x = 1: 3 / 1 = **3** adet (tam sayı bölmesi), satır sona eklenir.

Adım 12 / 13

DEĞİŞKENLER

`miktar = 3` `x = 1`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
1 10 kuruş
1 5 kuruş
3 1 kuruş
```

`raporuGuncelle`: 93 kuruş

```
1  void raporuGuncelle() {
2      try {
3          var miktar = Integer.parseInt(
4                  metin.getText(0, metin.getLength()));
5          int[] dizi = {50, 25, 10, 5, 1};
6          rapor.setText(miktar
7                  + " kuruş yapmak için kullanılacaklar:\n");
8          for (int x : dizi) {
9              rapor.append(miktar / x + " " + x + "
   kuruş\n");
▶              miktar %= x;
11         }
12     } catch (NumberFormatException e) {
13         rapor.setText("Tam sayı değil veya aralık
   dışında");
14     } catch (Exception e) {
15         rapor.setText(e.toString());
16     }
17 }
```



Kalan: 3 % 1 = 0.

Adım 13 / 13

DEĞİŞKENLER

`miktar = 0` `x = 1`

ÇIKTI

```
93 kuruş yapmak için
kullanılacaklar:
1 50 kuruş
1 25 kuruş
1 10 kuruş
1 5 kuruş
3 1 kuruş
```

main ve Olayların Sırası

```
public static void main(String[] args) {
    SwingUtilities.invokeLater(() -> {
        var frame = new ParaUstuHesaplama();
        frame.setTitle("Para Üstü Hesaplama");
        frame.pack();
        frame.setDefaultCloseOperation(
                JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    });
}
```

Kullanıcı / kod	Olay → rapor
`9` yazar	`insertUpdate` → 9 kuruş: 1 × 5, 4 × 1
`3` yazar	`insertUpdate` → 93 kuruş raporu
Hepsini siler	`removeUpdate` → `parseInt("")` : "Tam sayı değil..."
`abc` veya `99999999999`	`NumberFormatException` → aynı mesaj
`setText("287")`	**iki** olay: önce `removeUpdate`, sonra `insertUpdate` → 5, 1, 1, 0, 2
`-7`	Geçerli sayı! 0, 0, 0, −1, −2: kontrol eksik

İlk `catch` beklenen hatayı (sayı olmayan metin, `int` sınırını aşan değer) yakalar. İkinci `catch` diğer her şey için genel güvencedir; `getText(0, n)` işaretli (checked) `BadLocationException` fırlatabildiği için gereklidir.

ÖZET

İyi Uygulamalar

Yerleşim yöneticisi

Koordinatlar elle verilmez; yerleşim yöneticisiyle farklı ekran boyutu ve çözünürlükte düzen bozulmaz.

Anlamlı adlar

`b1`, `t2` yerine `buyukHarfButton`, `miktarAlani`: onlarca bileşenli kod okunur kalır.

Olay güdümlü düşünmek

Tasarım "Kullanıcı şunu yapınca ne olmalı?" sorusuyla yapılır; her eylem için bir dinleyici.

Arayüz ile iş mantığını ayırmak

Para üstü hesabı gibi kurallar ayrı bir metotta veya sınıfta tutulur; arayüzsüz test edilebilir, başka arayüzde de kullanılır.

Arayüzü EDT'de kurmak

`main`'in içi: `SwingUtilities.invokeLater(() -> { … })`.

EDT'yi meşgul etmemek

Dinleyicide uzun bir iş (dosya indirme, büyük hesap) yapılırsa pencere **donar**; bu işler ayrı iş parçacığına taşınır.

ÖZET

Özet

Kavram	Özü
Paketler	`java.awt` : Graphics, Color, Font, Point, yerleşim, olaylar · `javax.swing` : J… bileşenleri
`JFrame`	Pencere: `setSize` / `pack` , `setDefaultCloseOperation(EXIT_ON_CLOSE)` , en son `setVisible(true)`
`JPanel`	Kapsayıcı ve tuval; varsayılan `FlowLayout`
Çizim	`paintComponent(Graphics g)` geçersiz kılınır, önce `super.paintComponent(g)` ; değişince `repaint()`
Koordinat	(0, 0) sol üst; x sağa, y aşağı; ovaller sınırlayıcı dikdörtgenle verilir
Yerleşim	`BorderLayout` (5 bölge, içerik bölmesi varsayılanı), `FlowLayout` , `GridLayout` , `BoxLayout`
Olaylar	Dinleyici arayüzü + `add…Listener` ; çok metotlu arayüzler için adaptör; tek metotlu için lambda
Fare	`MouseListener` , `MouseMotionListener` ; `MouseEvent` : `getX` , `getY` , `getButton`
EDT	Swing tek iş parçacıklıdır; `invokeLater` asenkron, `invokeAndWait` senkron
Metin	`JLabel` gösterir, `JTextField` tek satır, `JTextArea` çok satır, `JScrollPane` kaydırır

ÖZET

Sık Yapılan Hatalar

İçe aktarmayı unutmak

`JFrame` için `import javax.swing.*;` yoksa: `cannot find symbol … class JFrame`.

setVisible(true) ya da EXIT_ON_CLOSE yok

Pencere hiç görünmez veya kapatılınca program arka planda çalışmaya devam eder.

Metot adını yanlış yazmak

`paintcomponent` yeni bir metottur, hiç çağrılmaz. `@Override` ile: `method does not override or implement a method from a supertype`.

super.paintComponent(g) unutmak

Zemin temizlenmez; eski çizim artıkları ekranda kalır.

Arayüzün tüm metotlarını yazmamak

`Dinleyici is not abstract and does not override abstract method mouseExited(MouseEvent) in MouseListener`. Çözüm: `MouseAdapter`.

Son öğe için get(size())

`IndexOutOfBoundsException: Index 1 out of bounds for length 1`. Son öğe `size() - 1`.

Lambda içinde yerel değişken değiştirmek

`e -> sayac++` : `local variables referenced from a lambda expression must be final or effectively final`. Çözüm: sayaç bir alan olur.

Sabiti sınıf adı olmadan yazmak

`add(bilesen, CENTER)` : `cannot find symbol … variable CENTER`. Doğrusu `BorderLayout.CENTER`.

ÖZET

Örnek Problemler

1. Giriş yasak işareti paneli 200 × 100 piksel olsaydı `r`, `r2`, `bw`, `bh` kaç olurdu?
2. Para üstü programına `68` yazılınca rapor ne olur?
3. `g.drawOval(50, 50, 100, 60)` ile çizilen elipsin merkezi neresidir?
4. `BorderLayout` kullanan bir panele önce `A`, sonra `B` düğmesi aynı `NORTH` bölgesine eklenirse kuzeyde hangisi görünür?
5. Kullanıcı panele bir kez tıklayıp bırakırsa `MouseListener`'ın hangi metotları hangi sırayla çağrılır?
6. `Color.RED.brighter()` hangi rengi verir?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. **Hedef tahtası**: panel boyutuna göre ölçeklenen, kırmızı-beyaz 5 iç içe halka çizin.
2. **Sayaç penceresi**: “−1”, “+1”, “Sıfırla” düğmeleri ve değeri gösteren bir `JLabel`.
3. Çizim programına bir **“Temizle”** düğmesi ekleyin.
4. Çizim programında **sağ tık** son eğriyi silsin.
5. Para üstü programı **negatif** miktarlar için “Miktar pozitif olmalı” yazsın.
6. **Sıcaklık dönüştürücü**: `JTextField`'a °C girilip düğmeye basılınca °F gösterilsin.

▶ **Cevaplar / ipuçları**

ÖZET

Kaynaklar

- Oracle, *The Java Tutorials*: Creating a GUI With Swing; Performing Custom Painting; Laying Out Components Within a Container; Writing Event Listeners; Concurrency in Swing.
- Oracle, *Java SE 21 API Documentation*: `java.awt.Graphics`, `java.awt.Color`, `javax.swing.JFrame`, `javax.swing.SwingUtilities`.
- P. Deitel ve H. Deitel, *Java: How to Program*, 11. baskı, Pearson (çevrim içi bölümler: Swing GUI Components; Graphics and Java 2D).
- K. Sierra, B. Bates ve T. Gee, *Head First Java*, 3. baskı, O'Reilly (Bölüm 14–15: GUI, Swing ve olaylar).
- C. S. Horstmann, *Core Java, Volume I: Fundamentals*, 12. baskı, Pearson (Bölüm 10–11: grafik programlama, olay işleme, Swing).
- A. B. Downey ve C. Mayfield, *Think Java*, 2. baskı, O'Reilly (Ek C: Graphics).

Son

Bölüm 9: Grafikler ve Kullanıcı Arayüzleri

Sonraki bölüm: Bölüm 10: İş Parçacıkları · Terim sözlüğü