Graphikprogrammierung in Java

Grundlegendes

- Bildschirmausgabe definiert durch spezielle Methode:

```
public void paint(Graphics g) {
      ...
}
```

- wird in Unterklassen von `java.awt.Component` überschrieben
- Methode wird Graphikkontext (`java.awt.Graphics`) übergeben
- (Neu-)Aufbau der Komponente ≈ Ausführen der `paint`-Methode
  - *systemgesteuert:* Sichtbarwerden, Größenänderung, …
  - *anwendungsgesteuert:* Aufruf der Methode `repaint`

Wie verwendet ?

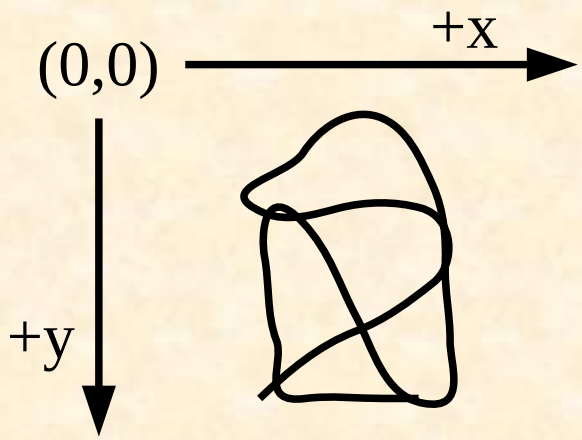
```
import java.awt.*;

public class MyPanel extends Panel {
        public void paint(Graphics g) {
                int dia = Math.min(getSize().width,
                                   getSize().height);
                g.fillOval((getSize.width–dia)/2,
                                (getSize.height–dia)/2, dia, dia);
        }
        public static void main(String args[]) {
                Frame myFrame = new Frame();
                myFrame.add(new MyPanel());
                myFrame.setSize(400,200);
                myFrame.show();
        }

}
```

java.awt.Graphics

- beschreibt den aktuellen *Graphikkontext*
  - Komponente auf der gezeichnet wird
  - aktuell gesetzte Farbe, Schrifttype
  - Koordinantensystem, ...
- enthält Methoden zum Zeichnen (Linien, Rechtecke, ...)

- Koordinatensystem:
  - Ursprung : linke obere Ecke

    x-Werte steigen nach rechts
    y-Werte steigen nach unten



Zeichnen

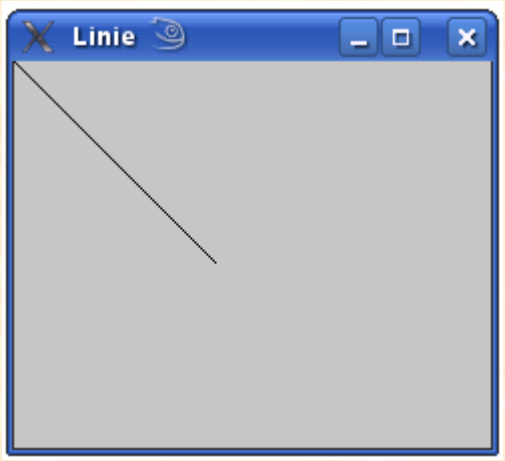
Linie

- Zeichnen einer Linie

```
drawLine(int x1, int y1, int x2, int y2)
```

  - zeichnet Linie zwischen Punkten (x1, y1) und (x2, y2)
  - verwendet dazu aktuell gesetzte Vordergrundfarbe

```
public void paint(Graphics g) {

    g.drawLine(0, 0, 100, 100);

}
```



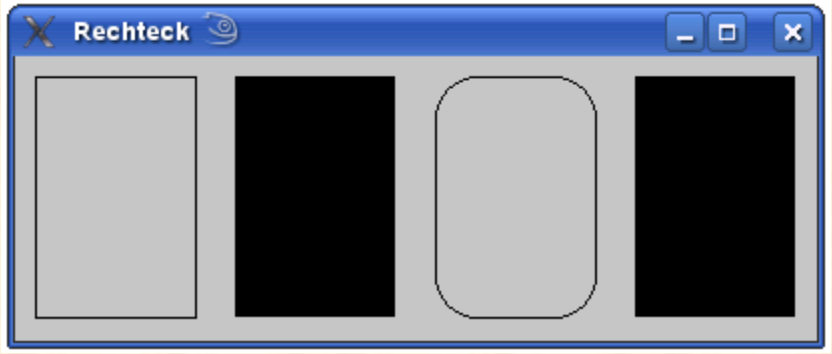
Rechteck

- Zeichnen von Rechtecken

```
drawRect(int x, int y, int width, int height)
fillRect(int x, int y, int width, int height)

drawRoundRect(int x, int y, int width,  int height,
                    int arcWidth, int arcHeight)
fillRoundRect(int x, int y, int width, int height,
                    int arcWidth, int arcHeight)

draw3DRect(int x, int y, int width, int height,
                  boolean raised)
fill3DRect(int x, int y, int width, int height,
                  boolean raised)
```

Wie verwendet ?

```
public void paint(Graphics g) {
        g.drawRect(10, 10, 80, 120);
        g.fillRect(110, 10, 80, 120);
        g.drawRoundRect(210, 10, 80, 120, 40, 40);
        g.fill3DRect(310, 10, 80, 120, false);
}
```



Ellipse und Kreisbogen

- Zeichnen von Ellipsen

```
drawOval(int x, int y, int width, int height)
fillOval(int x, int y, int width, int height)
```

- zeichnet Ellipse, die in das angegebene Rechteck passt

- Zeichnen von Kreisbögen

```
drawArc(int x, int y, int width, int height,
        int startAngle, int arcAngle)
fillArc(int x, int y, int width, int height,
        int startAngle, int arcAngle)
```

  - 0° = 3 Uhr, Mittelpunkt im angegebenen Rechteck
  - positive Winkel entgegen der Uhrzeigerrichtung

Polygon und Polygonzug

- Zeichnen von Polygonen

```
drawPolygon(int[] xPoints, int[] yPoints, int n)
fillPolygon(int[] xPoints, int[] yPoints, int n)
```

  - x- und y-Koordinaten durch Reihungen übergeben
  - n bestimmt Anzahl der Polygonsegmente

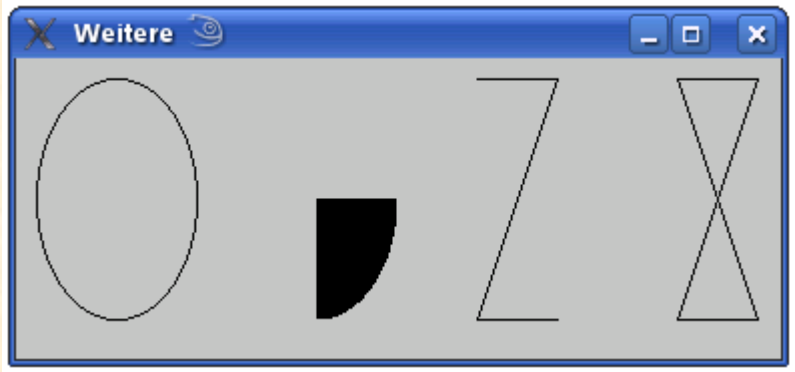
- Zeichnen eines Polygonzugs

```
drawPolyline(int[] xPoints, int[] yPoints, int n)
```

  - nur geschlossen, falls erster und letzter Punkt identisch

Wie verwendet ?

```
public void paint(Graphics g) {
        g.drawOval(10, 10, 80, 120);
        g.fillArc(110, 10, 80, 120, 0, -90);
        g.drawPolyline(new int[]{230, 270, 230, 270},
                       new int[]{10, 10, 130, 130}, 4);
        g.drawPolygon(new int[]{330, 370, 330, 370},
                       new int[]{10, 10, 130, 130}, 4);
}
```



Text

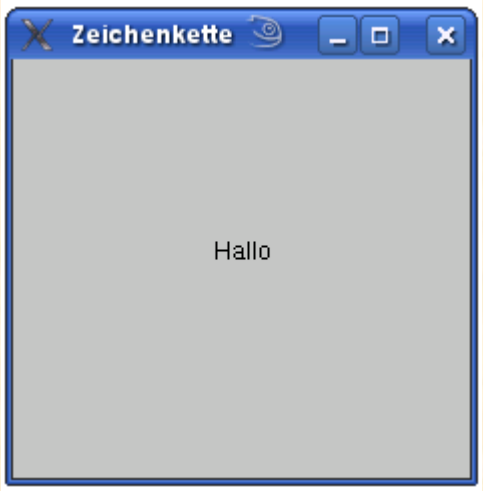
Ausgabe einer Zeichenkette

- Ausgabe einer Zeichenkette mittels

```
drawString(String str, int x, int y)
```

  - zeichnet `str` beginnend bei (x, y) (= *Grundlinie*)
  - verwendet dazu aktuell gesetzte Vordergrundfarbe

```
public void paint(Graphics g) {

g.drawString("Hallo", 100, 100);

}
```



java.awt.Font

- Repräsentation und Verwendung verschiedener Schrifttypen

```
new Font(String name, int style, int size)
```

Parameter name: logischer oder physikalischer Schrifttyp
(SansSerif, Monospaced, Dialog, DialogInput, Helvetica, ...)

Parameter style: Schriftstil (Font.PLAIN, Font.BOLD, Font.ITALIC)

Parameter size: Schriftgröße in Pixeln

Verwendung

- Festlegen der zu verwendenden Schrifttype

```
public void paint(Graphics g) {
        ...
        g.setFont(new Font("Serif",Font.PLAIN,16));
        ...
}
```

- Abfragen der gesetzten Schrifttype `Font f = g.getFont();`
- Ableiten einer Schrifttype `f = f.deriveFont(Font.BOLD);`

Beispiel

```
public void paint(Graphics g) {
      g.setFont(new Font("Serif", Font.ITALIC, 30));
      g.drawString("Hallo", 40, 60);
      Font myFont = new Font("SansSerif", Font.PLAIN, 16);
      g.setFont(myFont);
      g.drawString("Hallo", 200, 60);
      myFont.deriveFont(Font.BOLD, 30f);
      g.setFont(myFont);
      g.drawString("Hallo", 320, 60);
}
```

Farben, geometrische Figuren, Bilddateien

java.awt.Color

- (Standard-)Farbmodell in Java nutzt Farbdarstellung mit 32 Bit
  - 1 Byte pro Farbe: `Rot,Grün und Blau = RGB-Darstellung`
  - 1 Byte für *Alpha*-Wert: `deckend oder transparent`
- Explizite Konstruktion oder Zugriff auf konstante Werte

```
Color myColor = new Color(0, 0, 255);
myColor = Color.BLUE;
```

- Setzen und Abfragen der gesetzten (Vordergrund-)Farbe

```
g.setColor(Color.RED); Color c = g.getColor();
```

Geometrische Objekte

- Zwei grundsätzliche Möglichkeiten:

  Nutzen der Methoden aus `java.awt.Graphics` oder
  Implementierung der Schnittstelle `java.awt.Shape` und
  Nutzung der *Java 2D API* (`java.awt.Graphics2D`)

- Implementierung im Paket `java.awt.geom`

  `Linien, Kurven, Polygone, ...`

  Vorteile:

  explizite und verwendbare Datenstrukturen
  weitere Methoden (`intersects`, `contains`)

Bilddateien

- von Java unterstützte Formate: GIF, JPEG, PNG, (BMP), ...
- Repräsentation über Objekte der Klasse `java.awt.Image` bzw. `java.awt.image.BufferedImage`
- verschiedene Methoden für
  - Lesen und Anzeigen von Bilddateien
  - Manipulieren von Bilddateien
  - Schreiben von Bilddateien
  - Skalieren

Schreiben und Laden

Laden und Schreiben mittels `javax.imageio.ImageIO`

```
BufferedImage myImage = null;
try {
        myImage = ImageIO.read(new File("pict.gif"));
        ImageIO.write(myImage, "jpeg",
                        new File("copy_of_pict.jpg"));
} catch (IOException  e) { }
```

Anzeigen, Manipulieren, Skalieren

- Zeichnen
```
public void paint(Graphics g) {
    ...
    g.drawImage(myImage, 100, 100, this);
}
```
- Erzeugen
```
Image scr = frame.createImage(800, 600);
```
-
- Manipulieren
```
Graphics g = scr.getGraphics();
g.drawString("SCREEN", 100, 100);
```
-
- Skalieren
```
Image scaled = scr.getScaledInstance(
    (scr.getWidth() * 175) / 100,
    (scr.getHeight() * 175) / 100,
     Image.SCALE_DEFAULT);
```

Beispiel: Bild einlesen und in einem Fenster darstellen

```
import java.awt.*;
import java.io.*;
import javax.imageio.*;
import java.awt.image.*;

public class MyPanel extends Panel {
	public void paint(Graphics g) {
		BufferedImage myImage = null;
		try{ myImage = ImageIO.read(new File("picture.gif")); }
		catch (IOException  e) { }
		g.drawImage(myImage, 100, 100, this);
	}

	public static void main(String args[]) {
		Frame myFrame = new Frame();
		myFrame.add(new MyPanel());
		myFrame.setSize(400,200);
		myFrame.setVisible(true);
	}
}
```