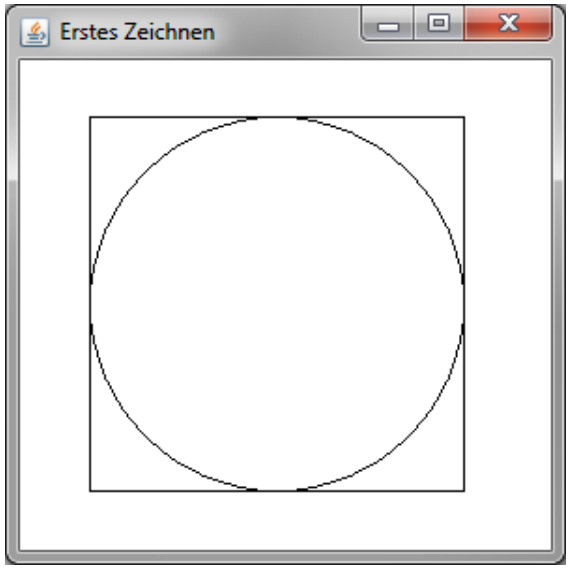
GRAFIK

Der Grafikkontext Graphics

- für ein Fenster (`JFrame/Frame`) wird automatisch eine Methode `paint()` aufgerufen, wenn das Fenster erstellt oder verändert (bewegt, Größe geändert, maximiert, …) wird
- diese Methode `paint(Graphics g)` übergibt ein Objekt vom Typ `Graphics` (eigentlich `Graphics2D`)
- der Grafikkontext stellt die Schnittstelle zwischen Hardware und dem Fenster dar; ein Grafikkontext existiert für ein Fenster genau ein Mal
- innerhalb des Grafikkontextes kann gezeichnet werden
- der Grafikkontext kann überall (für jede Komponente) mit `getGraphics()` zur Verfügung gestellt werden
- neben `paint()` erbt jede Swing-Komponente auch `paintBorder()`, `paintChildren()` und `paintComponent()`
- zum Zeichnen überschreiben wir stets **`paintComponent(Graphics g)`**

Zeichnen

- man kann direkt in das Fenster zeichnen:

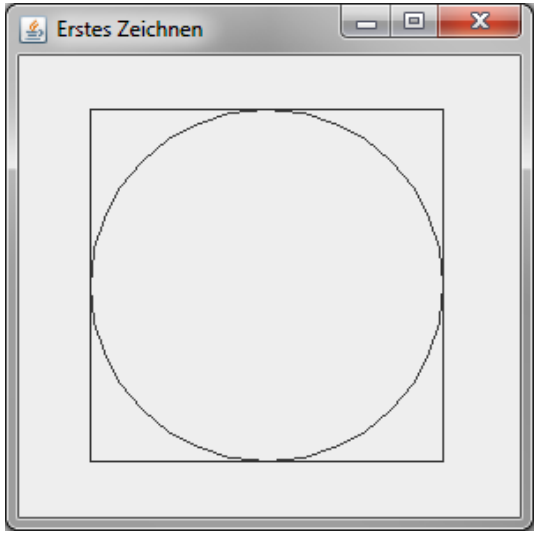



```
import java.awt.Graphics;
import javax.swing.JFrame;

public class InFrameZeichnen extends JFrame{

	InFrameZeichnen()
	{
		super("Erstes Zeichnen");
		setSize(300,300);
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		setVisible(true);
	}

	@Override
	public void paint(Graphics g)
	{
		g.drawRect(45, 60, 200, 200);
		g.drawOval(45, 60, 200, 200);
	}

	public static void main(String[] args) {
		new InFrameZeichnen();
	}
}
```

→ das ist aber unüblich (z.B. auch, weil (0,0) in Titelleiste liegt) …

… stattdessen

- fügen wir ein `JPanel` dem ContentPanel des Fensters hinzu und zeichnen in das `JPanel`



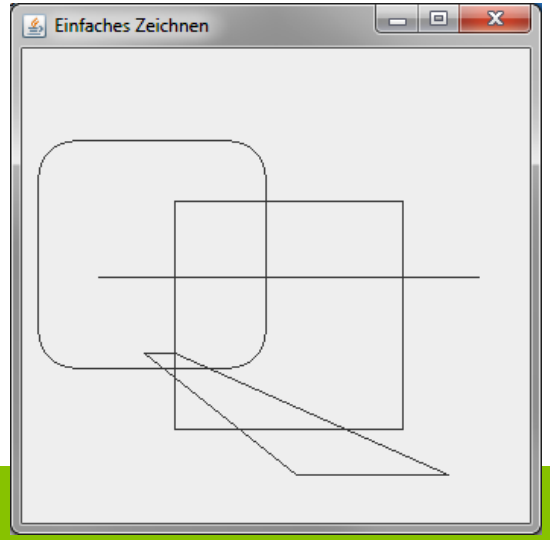

```
import java.awt.*;
import javax.swing.*;

public class InFrameZeichnen extends JFrame{

     private class Zeichenflaeche extends JPanel
     {

          @Override
          protected void paintComponent(Graphics g)
          {
               g.drawRect(40, 30, 200, 200);
               g.drawOval(40, 30, 200, 200);
          }
     }

     InFrameZeichnen()
     {
          super("Erstes Zeichnen");
          getContentPane().add(new Zeichenflaeche(),
          BorderLayout.CENTER);
          setSize(300,300);
          setDefaultCloseOperation(EXIT_ON_CLOSE);
          setVisible(true);
     }


     public static void main(String[] args) {
          new InFrameZeichnen();
     }
}
```

`paintComponent()`

- kann ruhig `protected` bleiben, da wir nicht vorhaben, sie aus anderen Klassen heraus aufzurufen
- mithilfe von `repaint()` kann das Neuzeichnen (der Aufruf von `paint()` für `JFrame`) manuell erzwungen werden
  - `repaint()`: sofortiges Neuzeichnen der Komponente
  - `repaint(long ms)`: Neuzeichnen in ms Millisekunden
  - `repaint(int x, int y, int width, int height)`: Neuzeichnen der Komponente im angegebenen Bereich
- in `paintComponent(Graphics g)` häufig zunächst der Aufruf `super.paintComponent(g);` (falls die Oberklasse zunächst noch ihre Inhalte zeichnen soll; bei vollständig eigenem Inhalt unnötig)
- `Graphics g` ist seit JDK 1.2 ein `Graphics2D`-Objekt; d.h. `Graphics2D g2 = (Graphics2D) g;` stets möglich

Linien, Rechtecke, Polygone
(EinfacheGeometrie.java)

```
protected void paintComponent(Graphics g)
{
    super.paintComponent(g); // alles andere zeichnen ... plus ...
    // Linien, Rechtecke, Polygone
    g.drawLine(50, 150, 300, 150); // x1,y1,x2,y2
    g.drawRect(100, 100, 150, 150); // x,y,breite,höhe
    g.drawRoundRect(10, 60, 150, 150, 50, 50); // Radius für Ecken
    int[] arx = {80, 180, 280, 100};
    int[] ary = {200, 280, 280, 200};
    g.drawPolygon(arx,ary,4);
}
```



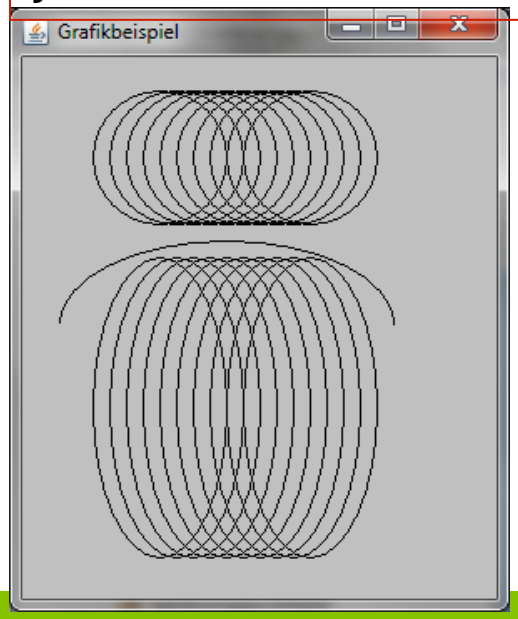
Anmerkungen

- für Polygone können auch zunächst Objekte der Klasse `java.awt.Polygon` erzeugt werden
- Polygon stellt die Methode `addPoint()` zur Verfügung
- mit `drawPolygon(Polygon p)` kann dann das Polygon ausgegeben werden

- Polygone sind geschlossen
- `drawPolyline(int[] arx, int[] ary, int cnt)` schließt nicht

Kreise, Ellipsen und Kreisbögen
(EinfacheGeometrie.java)

```
protected void paintComponent(Graphics g)
{
        super.paintComponent(g);
        for(int i=0; i<10; i++)
                g.drawOval(50+i*10, 50, 80, 80);
        for(int i=0; i<10; i++)
                g.drawOval(50+i*10, 150, 80, 180);
        g.drawArc(30, 140, 200, 100, 0, 180 );
}
```



Anmerkungen

- Kreise und Ellipsen werden mitderselben Methode
  `public void drawOval(int x, int y, int width, int heigh)`
  gezeichnet (ein Kreis ist eine spezielle Ellipse)

- Es werden nicht Mittelpunkt und Radius angegeben, sondern das Rechteck, das die Ellipse an engsten umschließt

- der Winkel bei `drawArc` beginnt mit 0 Grad bei 3 Uhr und läuft entgegen des Uhrzeigersinns

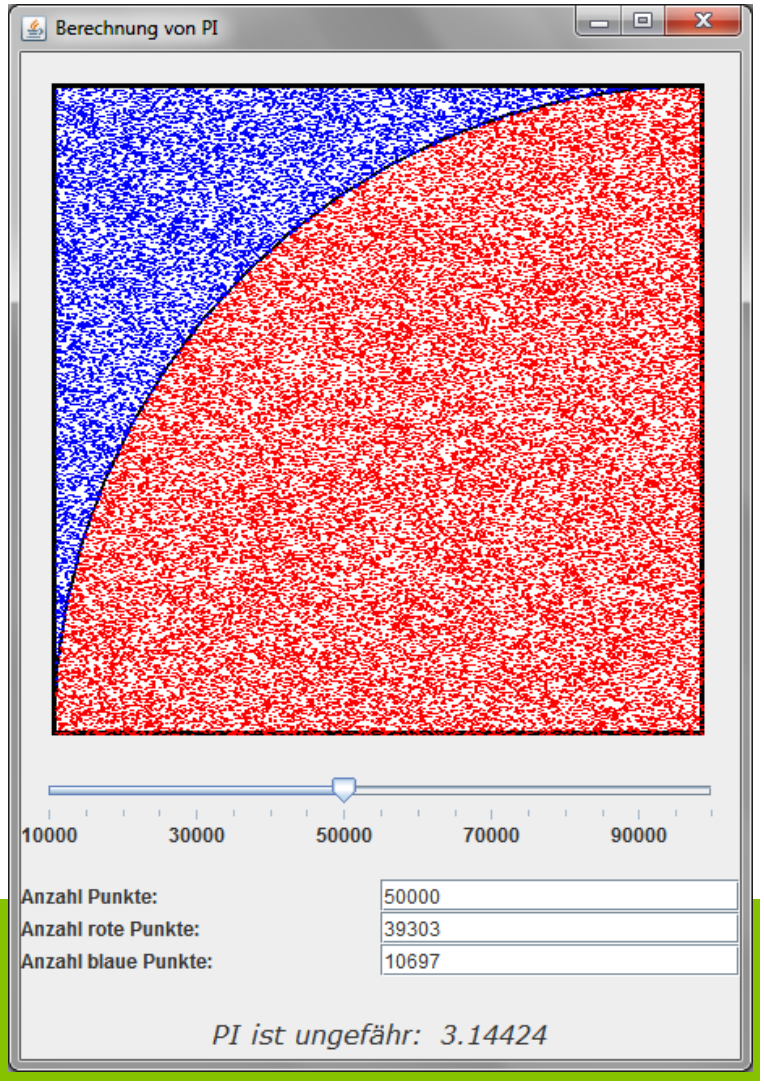
Füllmodus

- außer `drawLine` und `drawPolyline` existieren alle `drawXXX`-Methoden auch als `fillXXX`-Methoden:

  - **`public void fillRect(int x, int y, int w, int h)`**
  - **`public void fillRoundRect(int x, int y, int w, int h, int xr, int yr)`**
  - **`public void fillPolygon(int[] arx, int[] ary, int cnt)`**
  - **`public void fillPolygon(Polygon p)`**
  - **`public void fillOval(int x, int y, int w, int h)`**
  - **`public void fillArc(int x, int y, int w, int h, int start, int arc)`**

Aufgabe 7

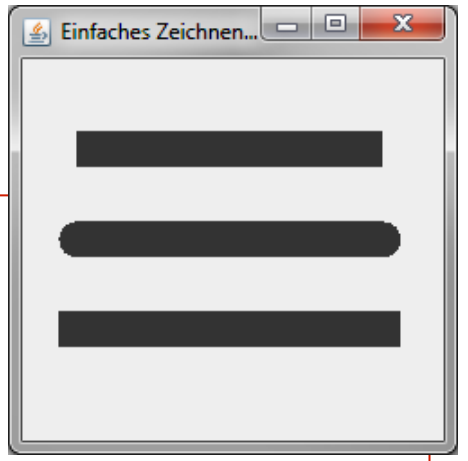
- Berechnen Sie PI mithilfe der Monte-Carlo-Methode.
- Erzeugen Sie dazu ein Quadrat mit der Länge laenge (z.B. = 400px)
- In dieses Quadrat zeichnen Sie einen Kreisbogen (Viertelkreis mit dem Radius laenge)
- Nun ermitteln Sie zufällig x- und y-Koordinaten für Kreise mit dem Durchmesser 2. Wenn diese Koordinaten innerhalb des Kreisbogens liegen, füllen Sie die Kreise rot, sonst blau
- Mithilfe eines Sliders bestimmen Sie die Gesamtanzahl der Punkte. Geben Sie die Gesamtanzahl, die Anzahl der roten und die Anzahl der blauen Punkte in Textfelder aus.
- Der Quotient aus der Anzahl der roten Punkte und der Gesamtanzahl der Punkte ist eine Näherung von PI/4. Geben Sie Ihre Annäherung von PI aus.
- Bei jeder Neueinstellung des Sliders erfolgt eine Neuberechnung von PI.



Graphics2D

Graphics2D g2 = (Graphics2D) g;

Linieneigenschaften – Stroke
(EinfacheGeometrie2D.java)




```
protected void paintComponent(Graphics g)
{
     Graphics2D g2 = (Graphics2D) g;
     g2.setStroke(new BasicStroke(20, BasicStroke.CAP_BUTT,
     BasicStroke.JOIN_MITER));
     g2.drawLine(30, 50, 200, 50);
     g2.setStroke(new BasicStroke(20, BasicStroke.CAP_ROUND,
     BasicStroke.JOIN_MITER));
     g2.drawLine(30, 100, 200, 100);
     g2.setStroke(new BasicStroke(20, BasicStroke.CAP_SQUARE,
     BasicStroke.JOIN_MITER));
     g2.drawLine(30, 150, 200, 150);
}
```

Anmerkungen

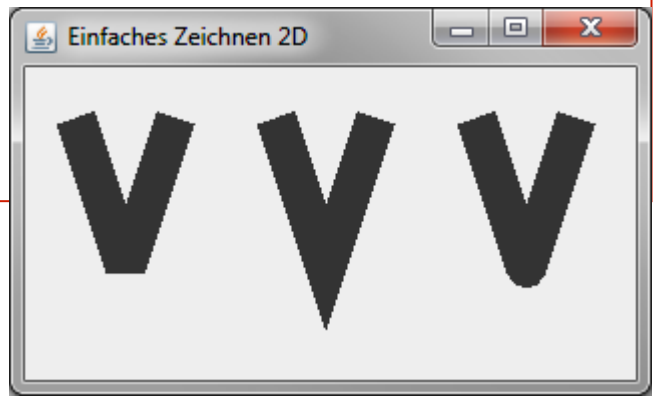
- CAP_BUTT : belässt das Ende, wie es ist

CAP_ROUND : abrunden mit Halbkreis

CAP_SQUARE : Rechteck anhängen

Linieneigenschaften – Stroke
(EinfacheGeometrie2D.java)

```
protected void paintComponent(Graphics g)
{
BasicStroke stroke = new BasicStroke(20,
BasicStroke.CAP_BUTT, BasicStroke.JOIN_BEVEL);
g2.setStroke(stroke);

Path2D shape = new GeneralPath();
shape.moveTo(25, 25); shape.lineTo(50, 100); shape.lineTo(75, 25);
g2.draw(shape);

// und die beiden anderen Shapes
}
```



Anmerkungen

- JOIN_BEVEL : zieht eine Linie zwischen den äußeren Eckpunkten

JOIN_MITER : erweitert die äußeren Linien bis sie sich treffen
JOIN_ROUND : rundet die Verbindungen ab