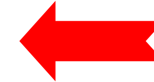
B2. 2D-Computergrafik mit Java

B2.1 Grundbegriffe der 2D-Computergrafik 
B2.2 Einführung in das Grafik-API "Java 2D"
B2.3 Eigenschaften von Grafik-Objekten
B2.4 Integration von 2D-Grafik in Programmoberflächen
B2.5 Wichtige Grafik-Operationen

Literatur:

J. Knudsen, Java 2D Graphics, O'Reilly 1999
http://java.sun.com/products/java-media/2D/

Rendering

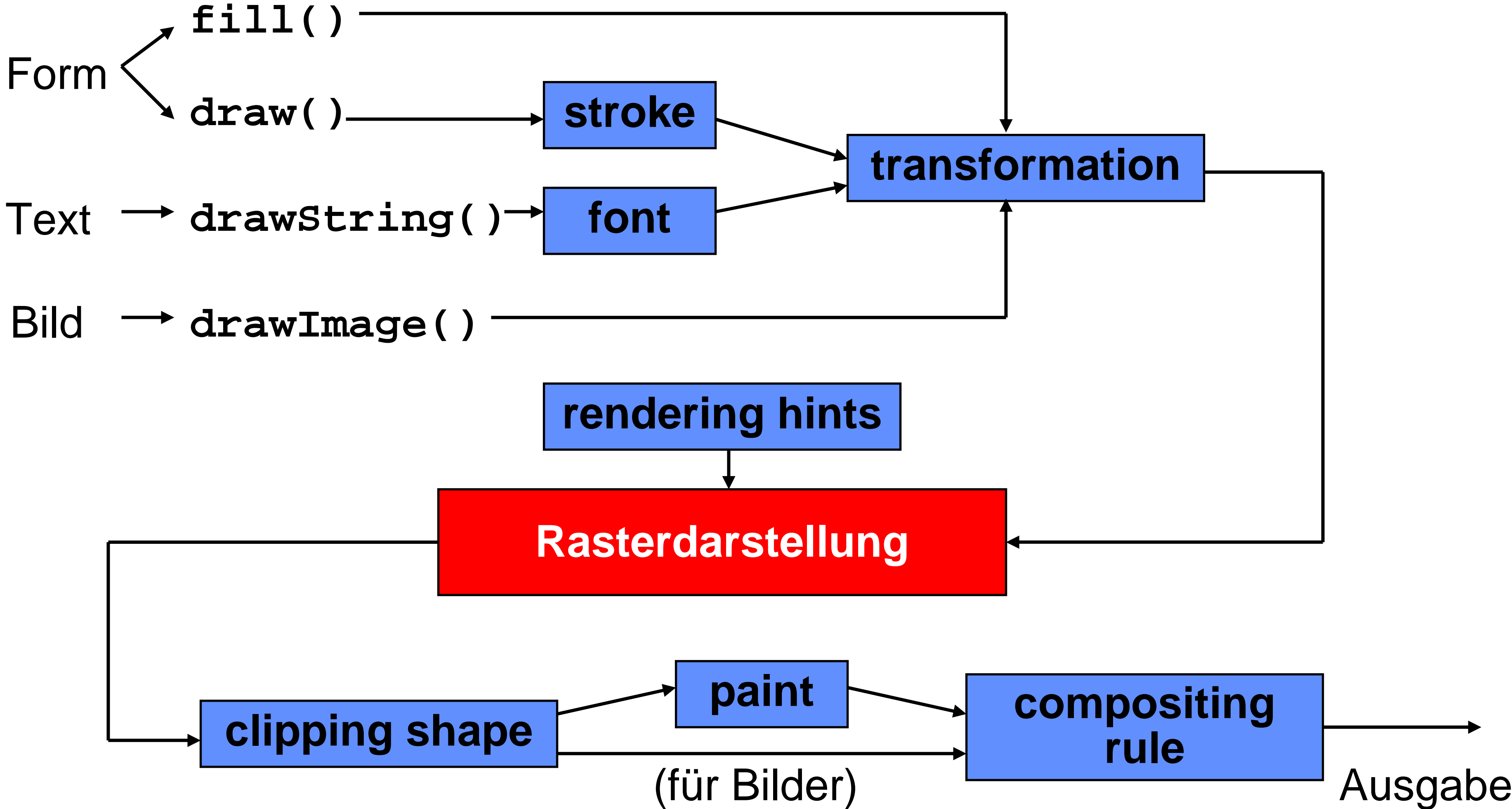
- *Rendering* ist die Umrechnung einer darzustellenden Information in ein Format, das auf einem Ausgabegerät in dem Menschen angemessener Form dargestellt werden kann.
- Rendering bei zweidimensionaler (2D-)Grafik:
  – Gegeben eine Ansammlung von Formen, Text und Bildern mit Zusatzinformation (z.B. über Position, Farbe etc.)
  – Ergebnis: Belegung der einzelnen Pixel auf einem Bildschirm oder Drucker
- *Grafikprimitive (graphics primitives):* Formen, Text, Bilder
- *Zeichenfläche (drawing surface):* Ansammlung von Pixeln
- *Rendering Engine:* Programm zur Rendering-Umrechnung
  – Bei Java-2D:

    Objekt der Klasse **`Graphics2D`**
    » ist Rendering Engine und
    » stellt Zeichenfläche bereit.

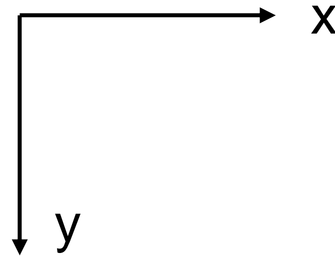
Rendering-Parameter

- Jedes primitive Grafikobjekt hat eigene Parameter, die die Darstellung beeinflussen:
  - Form *(shape)*: Ecken, Platzierung etc.
  - Text: Textinhalt
  - Bild *(image)*: Bildinhalt
- Weitere Parameter werden erst in der Rendering Engine festgelegt und beeinflussen ebenfalls die Darstellung:
  - Füllung *(paint):* Wie werden die Pixel für Formen, Linien und Text gefärbt?
  - Strich *(stroke):* Wie werden Linien gezeichnet (Stärke, Strichelung etc.)?
  - Schrift *(font):* Wie wird Text dargestellt (Schriftart, Schriftschnitt etc.)?
  - Transformation: Z.B. Verschieben, drehen, dehnen
  - Überlagerung *(compositing):* Kombination mit anderen Bildern (z.B. Hintergrund)
  - Zuschnitt *(clipping):* Bestimmung eines darzustellenden Ausschnitts
  - *Rendering hints:* Spezialtechniken zur Darstellungsoptimierung

Rendering-Pipeline



Koordinatensysteme



- Koordinatensysteme in der Computergraphik: y-Achse nach unten!
- Benutzerkoordinaten *(user space):*
  - Einheit *pixel* (picture elements)
  - Keine feste physikalische Größe
- Gerätekoordinaten *(device space):*
  - Einheit pixel
  - Abbildung der Benutzerkoordinaten abhängig vom Ausgabegerät
  - Standardabbildung (gut für Monitore geeignet): 72 pixel / Zoll
    - » d.h. US-amerikanische typografische Pica-Punkte
    - » Pixelbreite = 0,353 mm

B2. 2D-Computergrafik mit Java

B2.1 Grundbegriffe der 2D-Computergrafik

B2.2 Einführung in das Grafik-API "Java 2D"

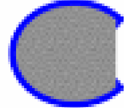
B2.3 Eigenschaften von Grafik-Objekten

B2.4 Integration von 2D-Grafik in Programmoberflächen

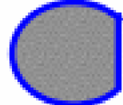
B2.5 Wichtige Grafik-Operationen

Java2D
Arcs_Curves
Clipping
Colors
Composite
Fonts
Images
Lines
Mix
Paint
Paths

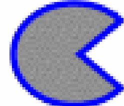
Arc2D.CHORD

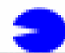


Arc2D.OPEN



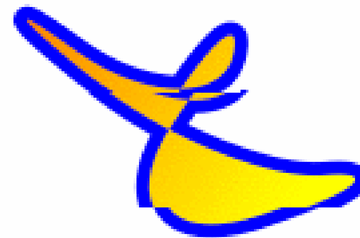
Arc2D.PIE



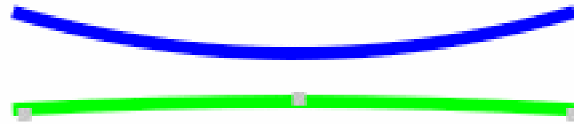


Draw Choice

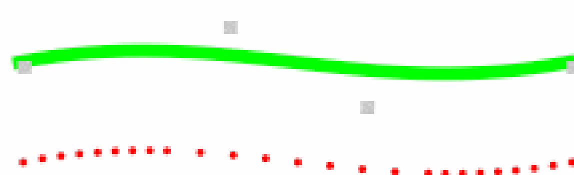
Fill Choice

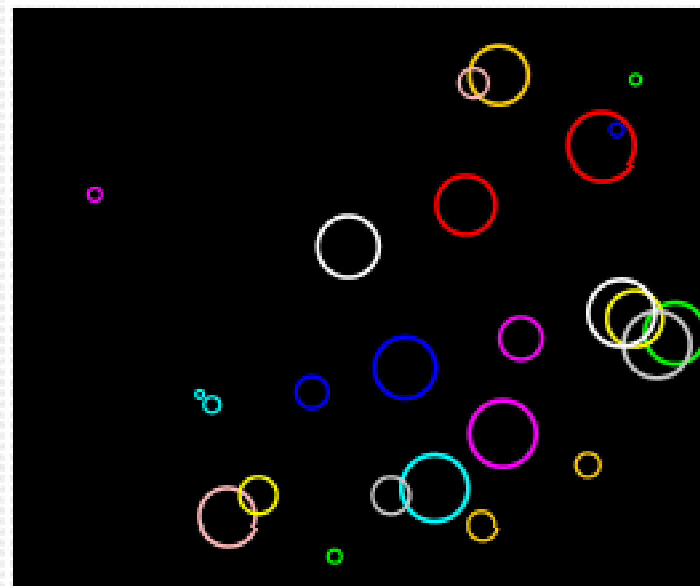


QuadCurve2D



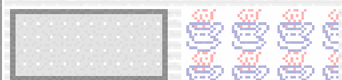
CubicCurve2D



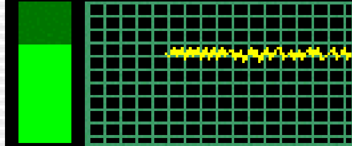


Global Controls
Anti-Aliasing
Rendering Quality
Texture
AlphaComposite
Auto Screen
Tools
Anim delay = 30 ms

Texture Chooser



Memory Monitor
17808K allocated
11931K used



Performance
Arcs 29.2 fps
BezierAnim 30.0 fps
Curves 32 ms
Ellipses 29.2 fps

Was ist Java 2D?

- Java 2D ist eine leistungsfähige Bibliothek zur Darstellung zweidimensionaler Grafik
  - Hier gewählt als konkretes *Beispiel*; es existieren viele ähnliche Plattformen
  - Java2D ist Bestandteil jeder Java 2-Installation und kostenfrei verfügbar
- Java 2D ist eines von vielen *Java Media APIs* (API = Application Programming Interface)
  - Andere Java Media APIs:
    - » Java Sound
    - » Java 3D
    - » Java Advanced Imaging
- Java 2D ist integriert in das Abstract Window Toolkit (AWT)
  - **`java.awt`**
  - **`java.awt.image`**
  - **`java.awt.color`**
  - **`java.awt.font`**
  - **`java.awt.geom`**

paint-Methode und Graphics-Objekte

- Jedes **`Component-`**Objekt, das auf dem Bildschirm erscheint, wird durch eine vordefinierte Methode **`paint()`** dargestellt.
- Grundprinzip der Grafik-Ausgabe: Überdefinieren von **`paint()`**
- Zeichenfläche wird vom System bereitgestellt und der **`paint()`**-Methode übergeben:
  - **`Graphics`**-Objekt
  - "Adressat" für alle Zeichenbefehle
  - **`public void paint(Graphics g) { ... }`**

- Historisch gewachsene Details in Java:
  - **`Graphics`**-Objekt war bereits in Java 1.1 vorhanden
  - Ab Java-Version 1.2 wesentlich erweiterte Grafikfunktionen (Java 2D)
    - » **`Graphics2D`**-Objekt wird übergeben und kann genutzt werden
    - » Typanpassung notwendig: **`Graphics2D g2 = (Graphics2D)g;`**

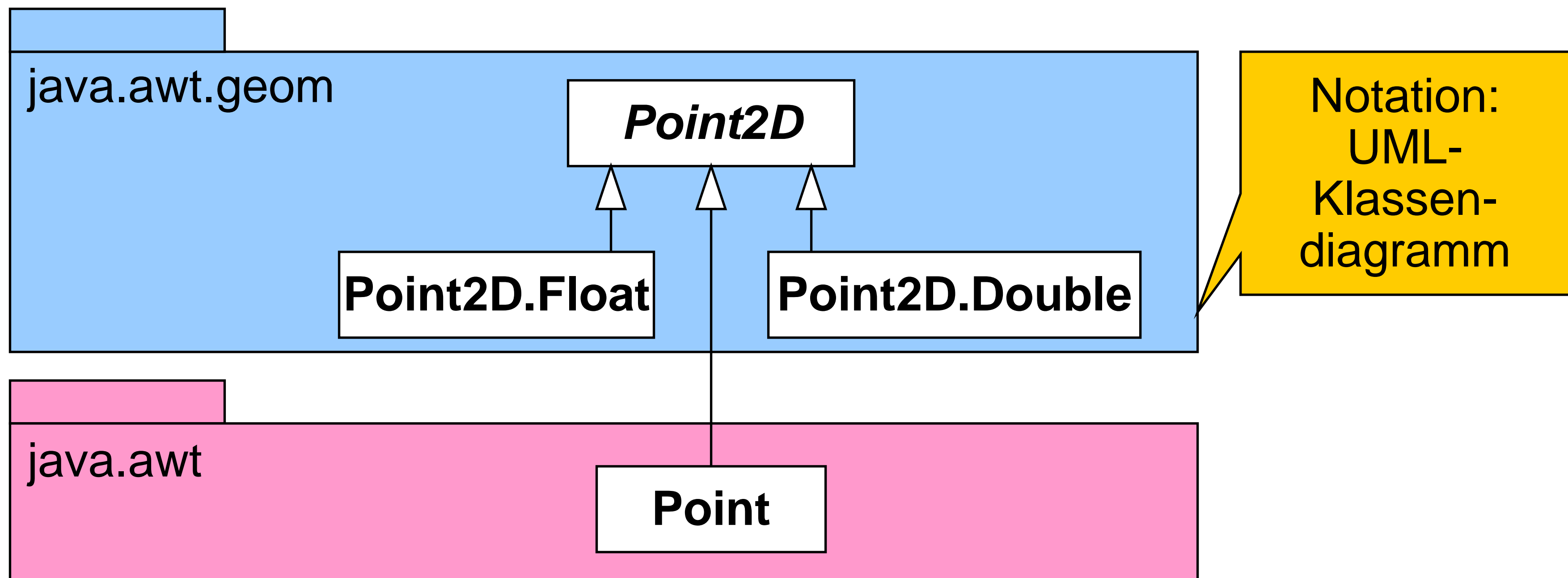
"Hello World" in Grafik-Version

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

public class HelloWorld extends Frame  {
  public static void main(String[] args) {
    new HelloWorld();
  }
  public HelloWorld() {
    setSize(500, 400);
    setLocation(200, 200);
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        System.exit(0);
      }
    });
    setVisible(true);
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;

    g2.draw(new Rectangle2D.Double(100, 100, 300, 200));
    g2.drawString("Hello World!", 150, 150);
  }
}
```

HelloWorld.java

Punkte

- Ein *Punkt* bezeichnet eine bestimmte (durch x- und y-Koordinaten beschriebene) Stelle der Zeichenfläche.
  - Ein Punkt hat keine Ausdehnung und keine weiteren Attribute
  - Ein Punkt ist verschieden von einem Pixel!
- Java 2D: **`java.awt.geom.Point2D`**
- Muster in Java 2D: Abstrakte Oberklasse mit *inneren Unterklassen*



Pfade

- Ein *Pfad* ist eine Folge von *n* Punkten p_i zusammen mit einer Spezifikation für die Verbindungsart für je zwei Punkte (p_{i-1}, p_i für $i = 2, \ldots, n$)
- Mögliche Verbindungsarten:
  - Keine Verbindung *("move to") – verpflichtend für ersten Punkt*
  - Gerade Verbindung *("line to")*
  - Quadratische Bezier-Kurve (mit 1 Steuerpunkt) *("quad to")*
  - Kubische Bezier-Kurve (mit 2 Steuerpunkten) *("curve to")*
- Java 2D: `java.awt.geom.GeneralPath`

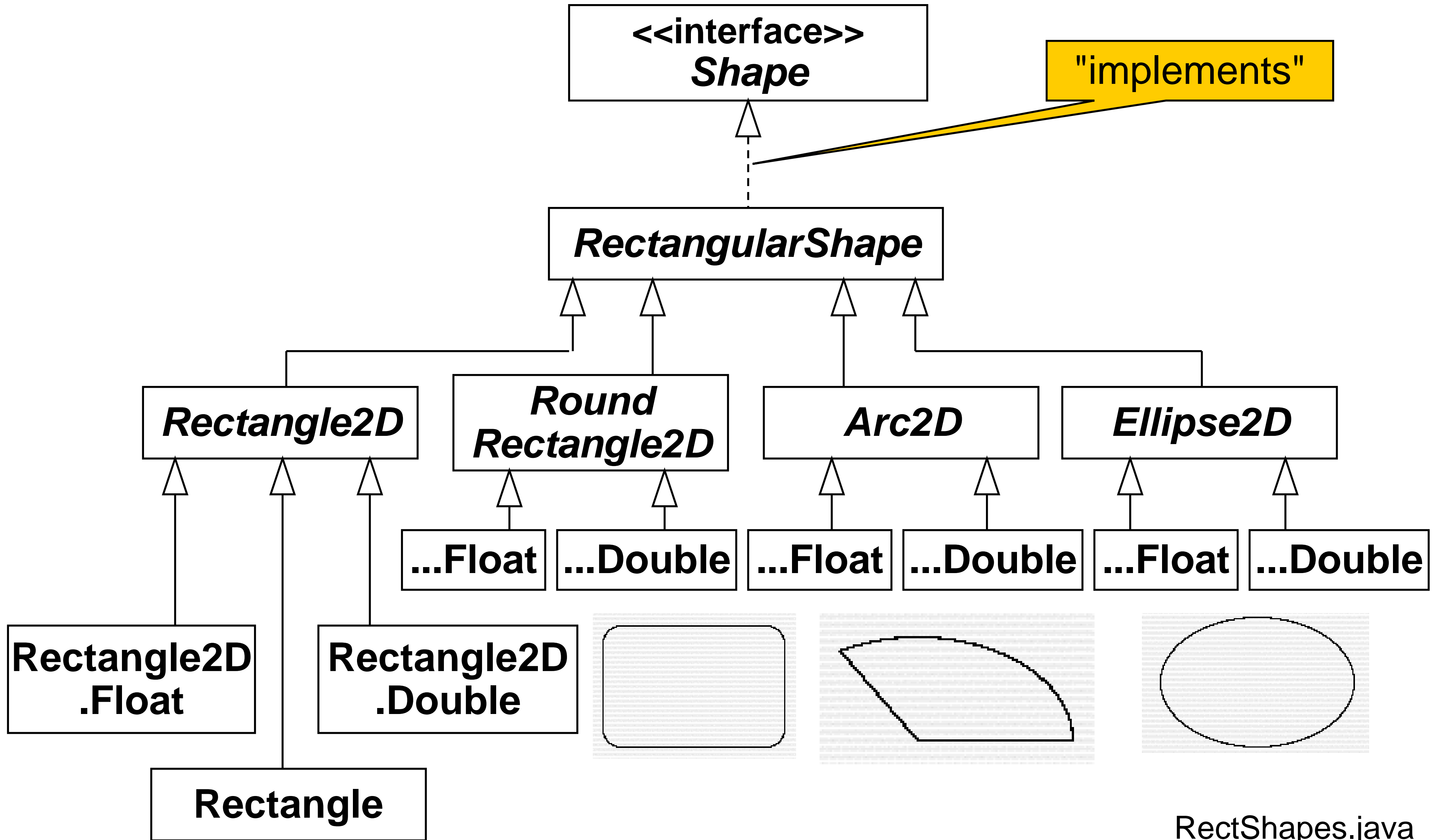
```
public void moveTo (float x, float y)
public void lineTo (float x, float y)
public void quadTo (float xc1, float yc1, float x, float y)
public void curveTo (float xc1, float yc1,
        float xc2, float yc2, float x, float y)
public void closePath() // Linie zum Ende des letzten moveTo
```

Beispiel: Pfad

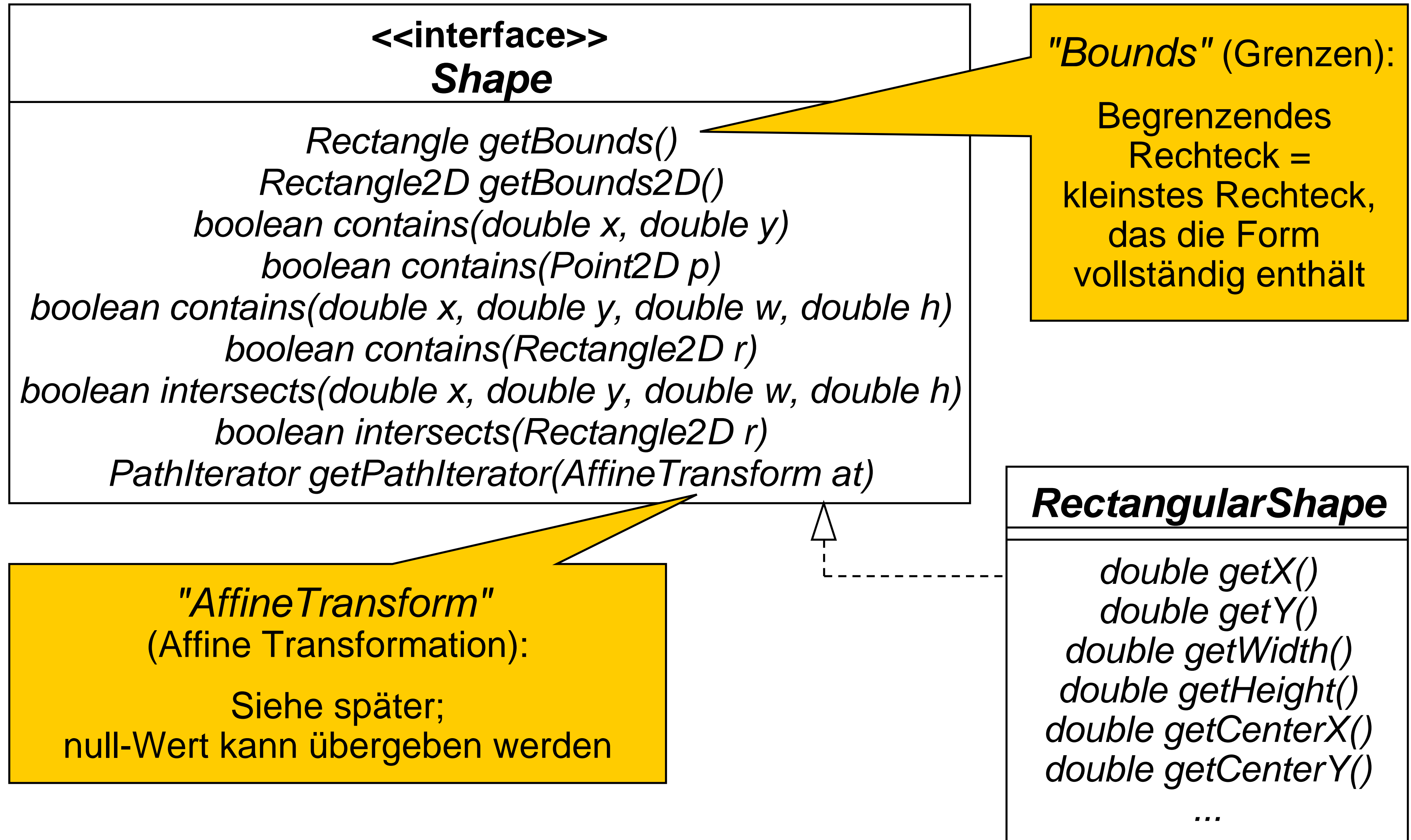
```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class Path extends Frame  {
  public static void main(String[] args) {
    new Path();
  }
  public Path() {
    // wie in HelloWorld
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;

    GeneralPath p = new GeneralPath();
    p.moveTo(50, 50);
    p.lineTo(70, 44);
    p.curveTo(100, 10, 140, 80, 160, 80);
    p.lineTo(190, 40);
    p.lineTo(200, 56);
    p.quadTo(100, 150, 70, 60);
    p.closePath();
    g2.draw(p);
  }
}
```

Path.java

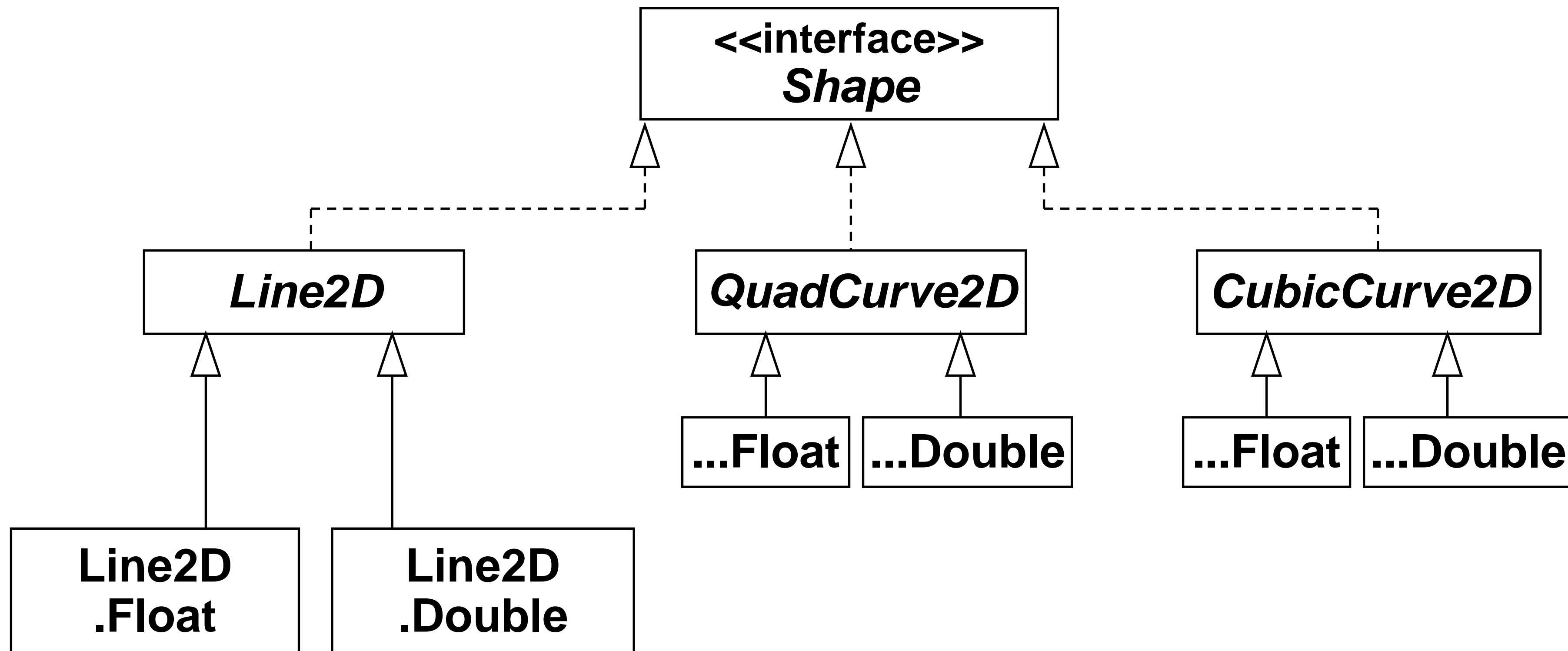
Basisformen (1): Flächige Formen



Formen



Basisformen (2): Linien und Kurven



Muß ich mir denn das alles merken?

- **NEIN !!! – Wichtig ist, die Dokumentation bedienen zu können**
- Aktuell: http://java.sun.com/j2se/1.5.0/docs/api/index.html

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages
java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event
java.awt.font

Rectangle
Rectangle2D
Rectangle2D.Double
Rectangle2D.Float
RectangularShape
ReentrantLock
ReentrantReadWriteLock
ReentrantReadWriteLock.
ReentrantReadWriteLock.
Ref
RefAddr
Reference
Reference
Referenceable
ReferenceQueue
ReferenceUriSchemesSup
ReferralException
ReflectionException

Overview Package **Class** Use Tree Deprecated Index Help

Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.awt.geom

Class RectangularShape

```
java.lang.Object
  └java.awt.geom.RectangularShape
```

All Implemented Interfaces:
Shape, Cloneable

Direct Known Subclasses:
Arc2D, Ellipse2D, Rectangle2D, RoundRectangle2D

```
public abstract class RectangularShape
extends Object
implements Shape, Cloneable
```

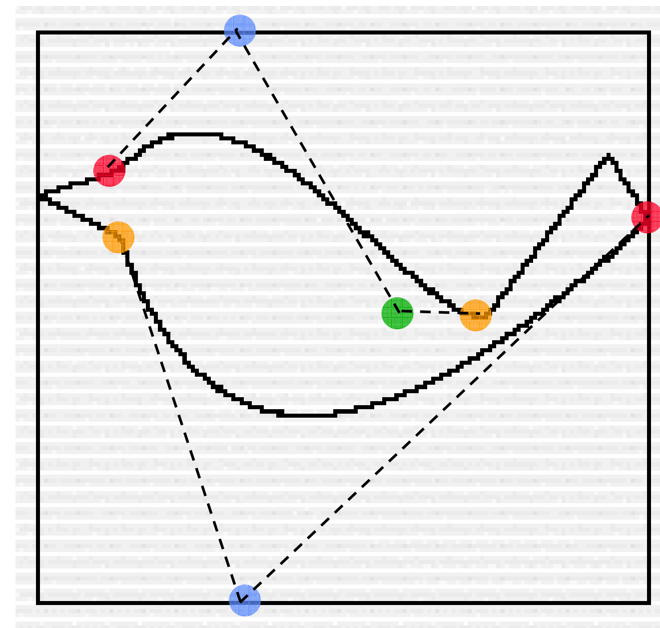
`RectangularShape` is the base class for a number of `Shape` objects whose geometry is defined by a rectangular frame. This class does not directly specify any specific geometry by itself, but merely provides manipulation methods inherited by a whole category of `Shape` objects. The manipulation methods provided by this class can be used to query and modify the rectangular frame, which provides a reference for the subclasses to define their geometry.

Java-Dokumentation (Auszug)

Method Summary	
Object	clone() Creates a new object of the same class and with the same contents as this object.
boolean	contains(Point2D p) Tests if a specified Point2D is inside the boundary of the Shape.
boolean	contains(Rectangle2D r) Tests if the interior of the Shape entirely contains the specified Rectangle2D.
Rectangle	getBounds() Returns the bounding box of the Shape.
double	getCenterX() Returns the X coordinate of the center of the framing rectangle of the Shape in double precision.
double	getCenterY() Returns the Y coordinate of the center of the framing rectangle of the Shape in double precision.
Rectangle2D	getFrame() Returns the framing Rectangle2D that defines the overall shape of this object.
abstract double	getHeight() Returns the height of the framing rectangle in double precision.
double	getMaxX() Returns the largest X coordinate of the framing rectangle of the Shape in double precision.
double	getMaxY() Returns the largest Y coordinate of the framing rectangle of the Shape in double precision.

Beispiel: Ausdehnung (Bounds)

```
GeneralPath p = new GeneralPath();
p.moveTo(50, 100);
p.lineTo(70, 94);
p.curveTo(100, 60, 140, 130, 160, 130);
p.lineTo(190, 90);
p.lineTo(200, 106);
p.quadTo(100, 200, 70, 110);
p.closePath();
g2.draw(p);

Rectangle2D r = p.getBounds2D();
g2.draw(r);
```



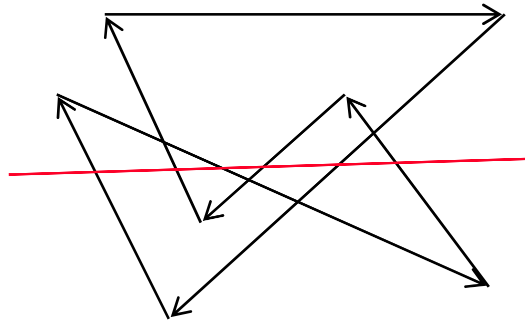
Konturen und Füllung

- Wie in der Rendering-Pipeline angedeutet, gibt es zwei Möglichkeiten, eine Form anzuzeigen:
  - `draw()` zeichnet die Konturen der Form
  - `fill()` füllt die Form aus
- Rendering-Parameter, z.B. `paint` bestimmen Details, wie Farbe, Strichstärke, Füllmuster
  - Werden beim `Graphics2D`-Objekt gesetzt! (nicht beim gezeichneten Objekt)
  - Setzen der Füllattribute: `setPaint(…)`
  - Standard-Farben: `Color.red, Color.blue, ...`
- Beispiel:

```
GeneralPath p = new GeneralPath();
p.moveTo(50, 50);
p.lineTo(70, 44);
...
p.closePath();
g2.setPaint(Color.red),
g2.fill(p);
```

PathFill.java

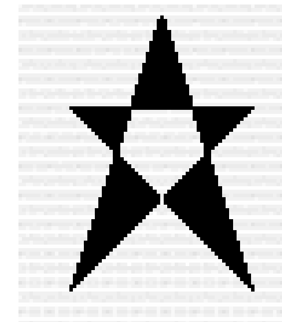
Innen und Außen (Winding Rules)



- Lege eine Schnittlinie durch die Figur und bestimme damit innen/außen für alle Punkte dieser Linie: Gedanklich der Linie durch die Figur folgen!
- EVEN_ODD-Regel:
  - Beginnend bei 0 außerhalb der Figur, erhöhe um 1, wenn eine Kante der Figur überquert wird. Innen: Zähler ungerade, außen: Zähler gerade
- NON_ZERO-Regel:
  - Beginnend bei 0 außerhalb der Figur, erhöhe (+1), wenn eine (von der Schnittlinie aus gesehen) von links nach rechts laufende Kante überquert wird, erniedrige (–1), wenn eine von rechts nach links laufende Kante überquert wird. Innen: Zähler ungleich null, außen: Zähler gleich null
- Java 2D: Winding Rule ist Parameter bei Konstruktor von **`GeneralPath`** (Default: NON_ZERO)

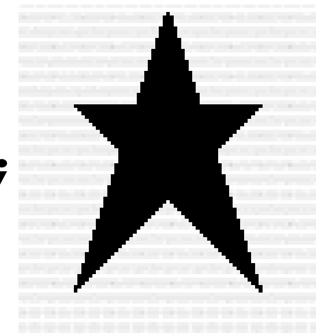
Beispiel: Winding Rules

```
 public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;

    GeneralPath p =
      new GeneralPath(GeneralPath.WIND_EVEN_ODD);
    p.moveTo(50, 50);
    p.lineTo(100, 50);
    p.lineTo(50, 100);
    p.lineTo(75, 25);
    p.lineTo(100, 100);
    p.closePath();
    g2.fill(p);
}
```



Alternativ:

```
    GeneralPath p =
      new GeneralPath(GeneralPath.WIND_NON_ZERO);
    ...
    g2.fill(p);
```



Winding.java

Interaktivität: Mausereignisse im 2D-Raum

- Jede Mausaktion des Benutzers wird
  - von der Hardware festgestellt
  - vom Betriebssystem (über entsprechende Treiber) registriert
  - dem Java-Laufzeitsystem als Ereignis mitgeteilt
  - von entsprechend registrierten "Listener"-Objekten ausgewertet
- Typische Mausaktionen sind:
  - Klicken
  - Drücken
  - Loslassen
  - "Betreten" eines bestimmten Bildschirmbereichs
  - "Verlassen" eines bestimmten Bildschirmbereichs
  - Bewegen
  - Ziehen (Bewegen mit gedrückter Maustaste)

java.awt.event.MouseListener

```
interface MouseListener {
  public void mouseClicked (MouseEvent e);
  public void mousePressed (MouseEvent e);
  public void mouseReleased (MouseEvent e);
  public void mouseEntered (MouseEvent e);
  public void mouseExited (MouseEvent e);
}
```

- Ein `MouseListener` kann zu jeder `Component` mittels `addMouseListener()` hinzugefügt werden.
- Ein trivialer `MouseListener` ist `MouseAdapter`.

Beispiel Winding.java

```
public class Winding extends Frame {
  public static void main(String[] args) {...}
  private GeneralPath p;
  private boolean in;
  public Winding() {
    ...
    addMouseListener(new MouseAdapter() {
      public void mouseClicked(MouseEvent me) {
        in = p.contains(me.getPoint());
        repaint();
      }
    });
    setVisible(true);
  }
  public void paint(Graphics g) {
    Graphics2D g2 = (Graphics2D)g;
    p = new GeneralPath(GeneralPath.WIND_EVEN_ODD);
    ...
    g2.fill(p);
    if (in)
      g2.drawString("in", 50, 150);
    else
      g2.drawString("out", 50, 150);
  }
}
```

java.awt.event.MouseMotionListener

```
interface MouseMotionListener {
  public void mouseDragged (MouseEvent e);
  public void mouseMoved (MouseEvent e);
}
```

- Ein `MouseMotionListener` kann zu jeder `Component` mittels `addMouseMotionListener()` hinzugefügt werden.
- Ein trivialer `MouseListener` ist `MouseMotionAdapter`.

- Hinweis: Das Swing-Interface `MouseInputListener` umfasst sowohl die Methoden von `MouseListener` als auch von `MouseMotionListener` . (Analog: `MouseInputAdapter`)

Beispiel: DragKing (1)

```
public class DragKing extends Frame
    implements MouseListener, MouseMotionListener {
  ...
  protected Point2D[] mPoints;
  protected Point2D mSelectedPoint;

  public DragKing() { ...
    mPoints = new Point2D[9];
    // Cubic curve.
    mPoints[0] = new Point2D.Double(50, 75);
    mPoints[1] = new Point2D.Double(100, 100);
    mPoints[2] = new Point2D.Double(200, 50);
    mPoints[3] = new Point2D.Double(250, 75);
    // Quad curve.
    mPoints[4] = new Point2D.Double(50, 175);
    mPoints[5] = new Point2D.Double(150, 150);
    mPoints[6] = new Point2D.Double(250, 175);
    // Line.
    mPoints[7] = new Point2D.Double(50, 275);
    mPoints[8] = new Point2D.Double(250, 275);

    mSelectedPoint = null;

    // Listen for mouse events.
    addMouseListener(this);
    addMouseMotionListener(this);

    setVisible(true);
  }
```

nach: J. Knudsen 99, S. 43

Beispiel: DragKing (2)

```
public void paint(Graphics g) {
   Graphics2D g2 = (Graphics2D)g;
   // Draw the tangents.
   Line2D tangent1 = new Line2D.Double(mPoints[0], mPoints[1]);
   Line2D tangent2 = new Line2D.Double(mPoints[2], mPoints[3]);
   g2.setPaint(Color.gray);
   g2.draw(tangent1);
   g2.draw(tangent2);
   // Draw the cubic curve.
   CubicCurve2D c = new CubicCurve2D.Float();
   c.setCurve(mPoints, 0);
   g2.setPaint(Color.black);
   g2.draw(c);
   // Draw the tangents.
   tangent1 = new Line2D.Double(mPoints[4], mPoints[5]);
   tangent2 = new Line2D.Double(mPoints[5], mPoints[6]);
   g2.setPaint(Color.gray);
   g2.draw(tangent1);
   g2.draw(tangent2);
   // Draw the quadratic curve.
   QuadCurve2D q = new QuadCurve2D.Float();
   q.setCurve(mPoints, 4);
   g2.setPaint(Color.black);
   g2.draw(q);
```

Beispiel: DragKing (3)

```
// public void paint(Graphics g) - continued

    // Draw the line.
    Line2D l = new Line2D.Float();
    l.setLine(mPoints[7], mPoints[8]);
    g2.setPaint(Color.black);
    g2.draw(l);

    for (int i = 0; i < mPoints.length; i++) {
      // If the point is selected, use the selected color.
      if (mPoints[i] == mSelectedPoint)
        g2.setPaint(Color.red);
      else
        g2.setPaint(Color.blue);
      // Draw the point.
      g2.fill(getControlPoint(mPoints[i]));
    }
  }
  protected Shape getControlPoint(Point2D p) {
    // Create a small square around the given point.
    int side = 4;
    return new Rectangle2D.Double(
        p.getX() - side / 2, p.getY() - side / 2,
        side, side);
  }
```

Beispiel: DragKing (4)

```
  public void mouseClicked(MouseEvent me) {}
  public void mousePressed(MouseEvent me) {
    mSelectedPoint = null;
    for (int i = 0; i < mPoints.length; i++) {
      Shape s = getControlPoint(mPoints[i]);
      if (s.contains(me.getPoint())) {
        mSelectedPoint = mPoints[i];
        break;
      }
    }
    repaint();
  }
  public void mouseReleased(MouseEvent me) {}
  public void mouseMoved(MouseEvent me) {}
  public void mouseDragged(MouseEvent me) {
    if (mSelectedPoint != null) {
      mSelectedPoint.setLocation(me.getPoint());
      repaint();
    }
  }

  public void mouseEntered(MouseEvent me) {}
  public void mouseExited(MouseEvent me) {}
}
```