

3. Das Java-AWT (I)

Elementares Zeichnen

Das AWT von Java

Abstract:

- allgemeiner, plattformunabhängiger Ansatz

Window:

- Umgang mit Fenstersystemen

Toolkit:

- sehr grosse Klassenbibliothek

Ziel dieser Session

- Überblick über das AWT
- Kritik
- Tips und Hinweise
- Notwendiges Material für die Übungen

BildschirmAusgabe in Windowsystemen

paint on demand:

Nicht der *Programmierer* bestimmt, wann etwas auf den Bildschirm gebracht wird, sondern der *User*

also:

Nicht das *Programm* bestimmt, wann etwas auf den Bildschirm gebracht wird, sondern das *Windowsystem*

Realisierung über

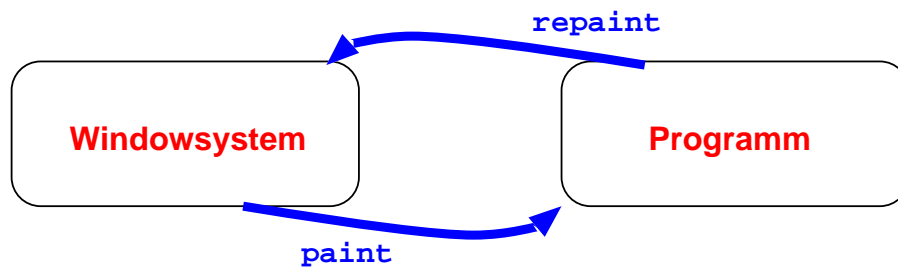
- callback-Funktionen (C) oder
- spezielle Methoden (OO)

Konsequenzen:

- Das Programm muss zu jedem Zeitpunkt in der Lage sein, die komplette BildschirmAusgabe zu rekonstruieren
- Das Programm muss das Windowsystem über neu anzuzeigende BildschirmInhalte informieren

Der AWT-Anzeigemechanismus

Das Windowsystem ruft die `paint`-Methode eines Applets auf, sobald das Applet gezeichnet werden soll.

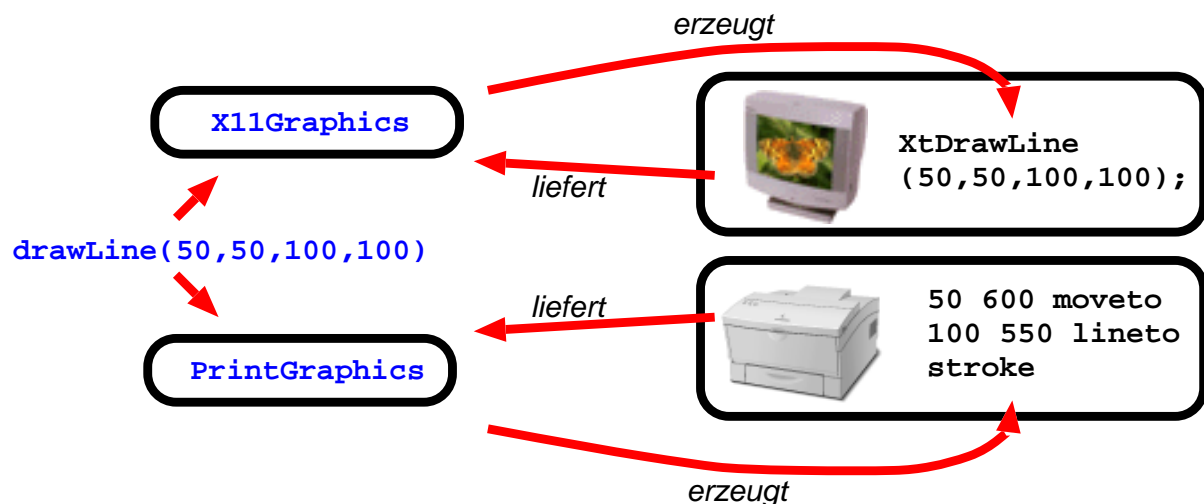


Das Applet ruft seine eigene `repaint`-Methode auf, sobald es wünscht, neu gezeichnet zu werden.

Dieses System gilt für alle sichtbaren *Komponenten*.

Graphics (I)

Eine Zeichenaufforderung wird stets von einem `Graphics`-Objekt begleitet, welches die notwendigen Zeichenroutinen zur Verfügung stellt. Dadurch wird *plattform-* und *device-*unabhängiges Zeichnen ermöglicht.



Das `Graphics`-Objekt “weiss”, wie man einfache Zeichenoperationen auf der anfragenden Oberfläche ausführt. Zusätzlich speichert es Informationen über den derzeitigen Zustand der Zeichenfläche (z.B. die aktuelle Farbe für nachfolgende Zeichenoperationen)

Wichtige Klassen des AWT (I)

Graphics: stellt die eigentlichen Zeichenoperationen zur Verfügung

Color: repräsentiert Farben

Font: repräsentiert Schrifttypen.

Point: abstrakter Punkt mit Koordinaten `x`, `y`

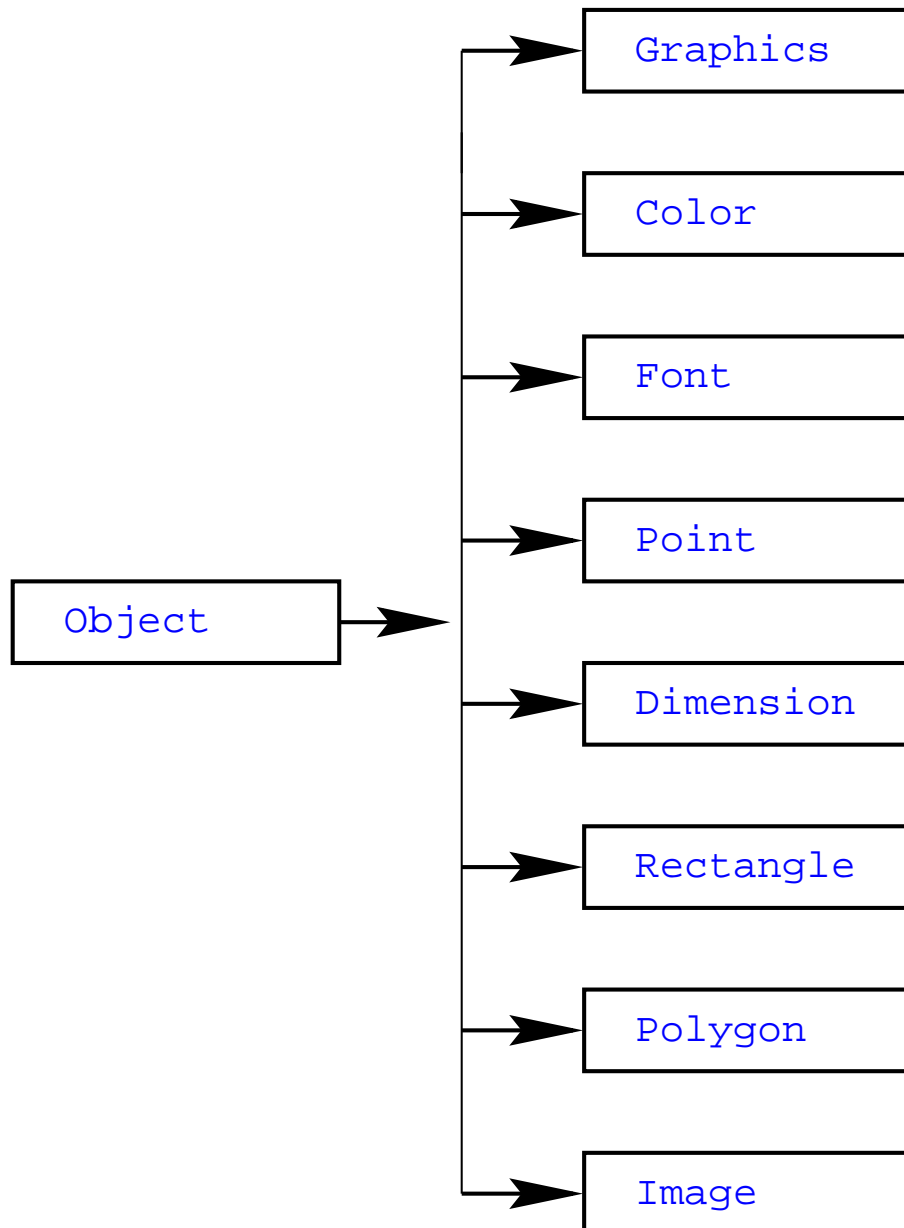
Dimension: abstrakte Grössenangabe mit `width`, `height`

Rectangle: abstraktes Rechteck.

Polygon: abstraktes n-Eck.

Image: abstrakte Klasse für Bitmap-Graphiken

AWT Hierarchie (I)



Graphics (II)

`Graphics`-Objekte dienen als eine Art Zeichenstift. Sie werden nicht selbst instantiiert, sondern vom Ausgabegerät zur Verfügung gestellt.

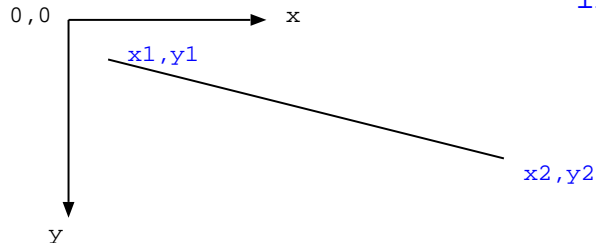
Die Klasse selbst ist `abstract`, die `Graphics`-Objekte sind stets Erweiterungen dieser Klasse, die durch das Ausgabegerät instantiiert werden. Mögliche Ausgabegeräte sind der Bildschirm, der Drucker, aber auch Bitmap-Grafiken im Speicher.

Zusätzlich speichert die Klasse den aktuellen Zustand des Gerätes:

- Zeichenfarbe
- Clipping-Region
- Font
- XOR-Mode

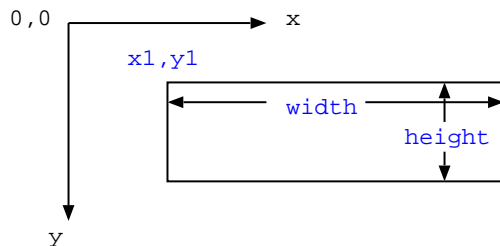
Zeichenoperationen (I)

```
public abstract void drawLine(int x1, int y1,  
                              int x2, int y2)
```



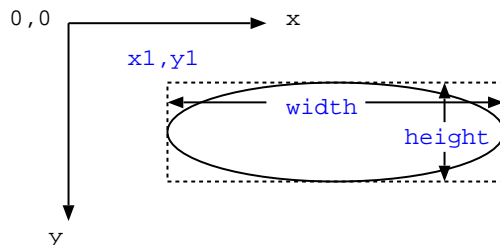
zeichnet eine Linie, ein Pixel dick
Wenn Anfang gleich Ende: Punkt
es gibt kein `drawPoint` !

```
public abstract void drawRect(int x1, int y1,  
                              int width, int height)
```



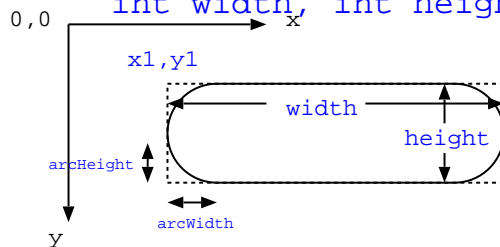
zeichnet ein Rechteck
Größenangabe darf nicht
negativ sein!

```
public abstract void drawOval(int x1, int y1,  
                              int width, int height)
```



zeichnet ein Oval
Größenangabe darf nicht
negativ sein!
width=height ergibt Kreis

```
public abstract void drawRoundRect(int x1, int y1,  
                                   int width, int height, int arcWidth, int arcHeight)
```



zeichnet ein abgerundetes
Rechteck
Größenangaben dürfen nicht
negativ sein!

Zeichenoperationen (II)

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

füllt das Rechteck mit der Zeichenfarbe

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

füllt das Oval mit der Zeichenfarbe

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

zeichnet ein Rechteck im 3D-Look, `raised` entscheidet, ob es aussieht wie "gedrückt" oder wie "nicht gedrückt"

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

füllt ein Rechteck im 3D-Look

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

zeichnet einen Oval-Bogen

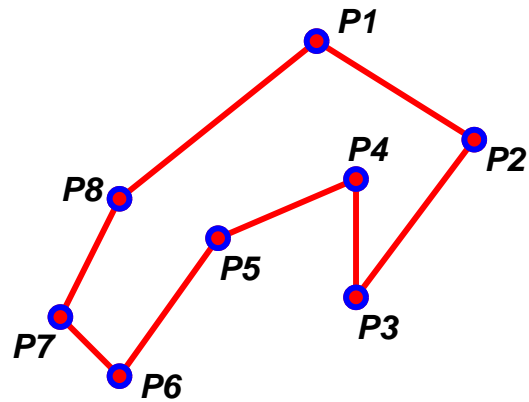
Zeichnen von abstrakten Objekten

```
void drawPolygon(Polygon p)
```

Zeichnet ein durch das abstrakte Polygon `p` vorgegebenes Polygon.

Exkurs: Abstrakte Datentypen im AWT

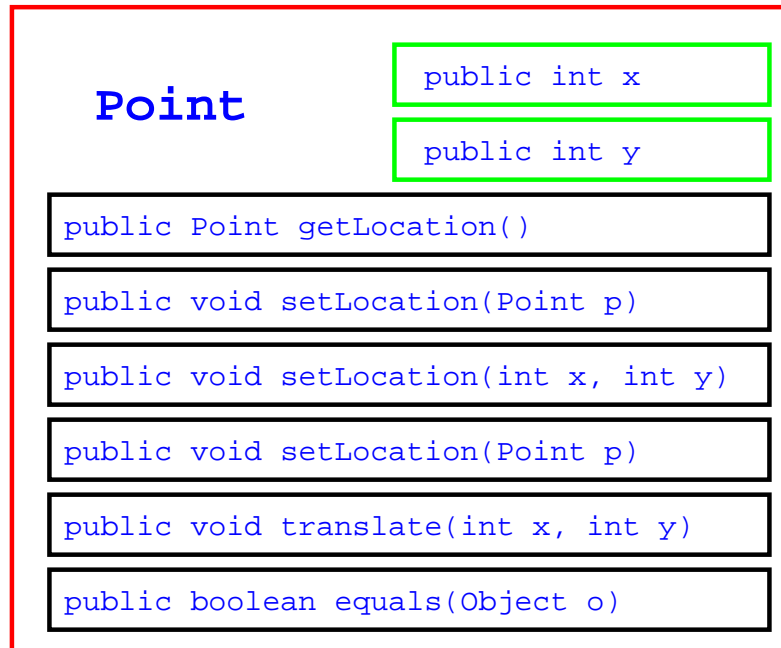
`Polygon` ist eine durch das AWT vorgegebene Datenstruktur, die eine Reihe von Punkten speichert, die zum Beispiel die Ecken eines Polygons darstellen können.



Es gibt diverse solcher abstrakter Datentypen — `Point`, `Dimension`, `Shape`, `Rectangle`, `Polygon` — aber leider ist das Konzept von abstrakten Datentypen im AWT nicht konsequent verfolgt worden. Meist werden sie nur als Rückgabewerte, nicht aber als Parameter eingesetzt.

`drawPolygon` und `fillPolygon` sind die einzigen Methoden in `Graphics`, die eine sinnvolle Übergabe abstrakter Datentypen erlauben.

Point



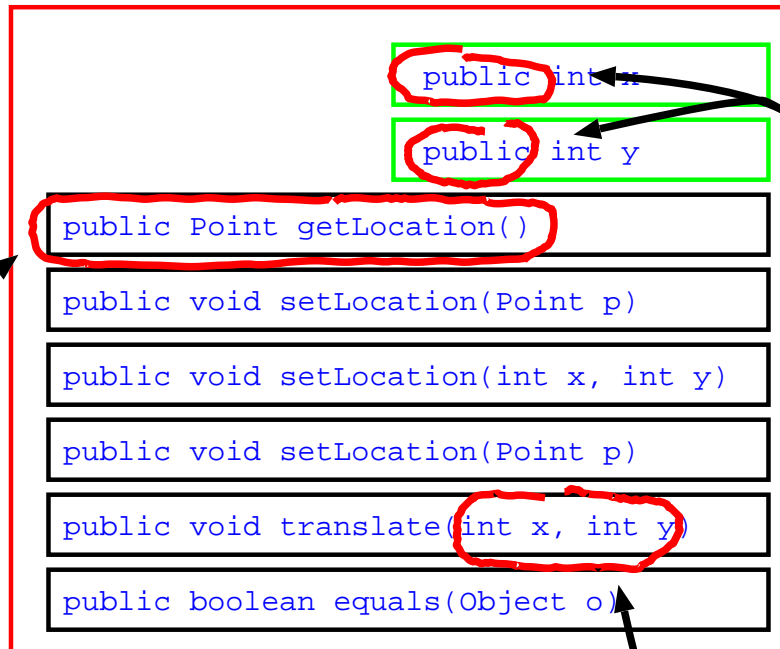
Point ist ein Behälter für zwei `int`-Werte.

`get-` und `setLocation` greifen auf `x` und `y` zu.

`translate` verschiebt den Punkt durch Addieren der Argumente zu den Koordinaten des Punktes.

`equals` überschreibt die Standard-`equals`-Methode von `Object`. Sie liefert `true`, falls das Argument ebenfalls ein Punkt ist und die Koordinaten des Arguments mit den Koordinaten des Punktes übereinstimmen.

Point — Kritik



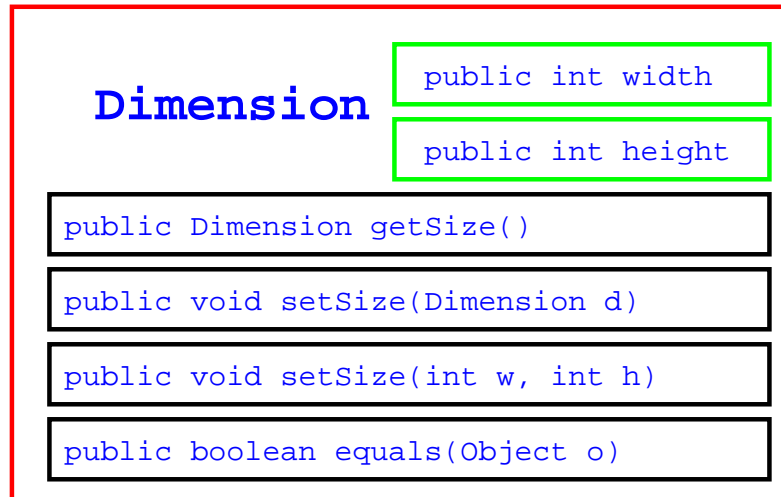
Un-OO!
Die öffentliche
Deklaration
erlaubt unkon-
trollierten Zugriff
und macht die
Zugriffsmethoden
unnötig!

Macht keinen Sinn: Ein Aufruf dieser Methode erzeugt nur eine neue Instanz des Punktes (Copy-Konstruktor), liefert aber keinesfalls die gewünschte Information

Warum gibt es keine Methode die einen Punkt übergeben bekommt?

An der Klasse `Point` erkennt man, wie Sun beim Design des AWT den schmalen Grat zwischen Effizienz und strenger Objektorientiertheit verfehlt hat.

Dimension



`Dimension` ähnelt sehr stark der Klasse `Point`, es fehlt eine Methode zur Verschiebung und die Instanz-Variablen sind anders benannt.

`Dimension`-Objekte treten hauptsächlich auf, wenn (Bildschirm-)Komponenten nach ihrer Grösse gefragt werden.

Beispiel: `myApplet.getSize()` liefert ein `Dimension`-Objekt zurück, welches die Grösse von `myApplet` repräsentiert.

Rectangle (I)

Ein `Rectangle` ist die Kombination eines Punktes (`Point`) und einer Ausdehnung (`Dimension`).

Erstaunlicherweise wird aber von den bereits bekannten Klassen kein Gebrauch gemacht. Sie tauchen nur in den Konstruktoren auf. Ebenfalls ist es nicht möglich, ein `Rectangle`-Objekt der `drawRect`-Methode von `Graphics` zu übergeben.

Zusätzlich zu den schon aus `Point` und `Dimension` bekannten Routinen werden Methoden zu Vereinigung und Schnitt zur Verfügung gestellt.

Konstruktoren:

```
public Rectangle();
public Rectangle(Rectangle r);
public Rectangle(int x, int y,
                 int width, int height);
public Rectangle(int width, int height);
public Rectangle(Point p, Dimension d);
public Rectangle(Point p);
public Rectangle(Dimension d);
```

Rectangle (II)

Zugriffsfunktionen auf die Instanz-Variablen

```
public Rectangle getBounds();
public Point getLocation();
public Dimension getSize();

public void setBounds(Rectangle r);
public void setBounds(int x, int y,
                      int width, int height);
public void setLocation(Point p);
public void setLocation(int x, int y);
public void setSize(Dimension d);
public void setSize(int width, int height);
```

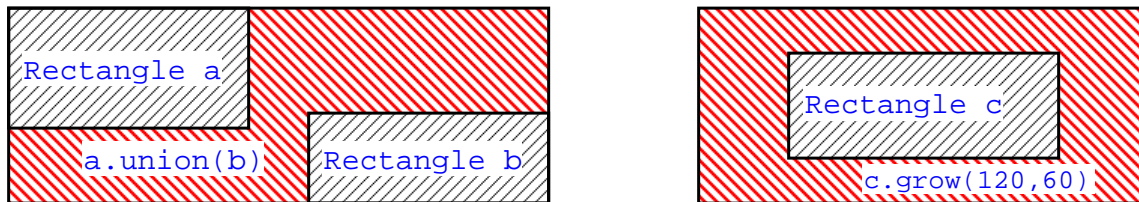
Tests

```
public boolean contains(Point p);
public boolean contains(int x, int y);
public boolean equals(Object obj);
public boolean intersects(Rectangle r);
public boolean isEmpty();
```

Rectangle-Manipulationen

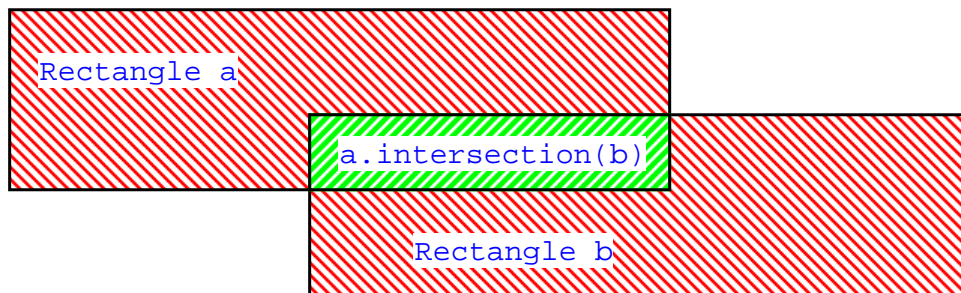
Vergrößerung des Rechtecks

```
public void add(int newx, int newy);  
public void add(Point pt);  
public void add(Rectangle r);  
public Rectangle union(Rectangle r);  
  
public void grow(int h, int v);
```

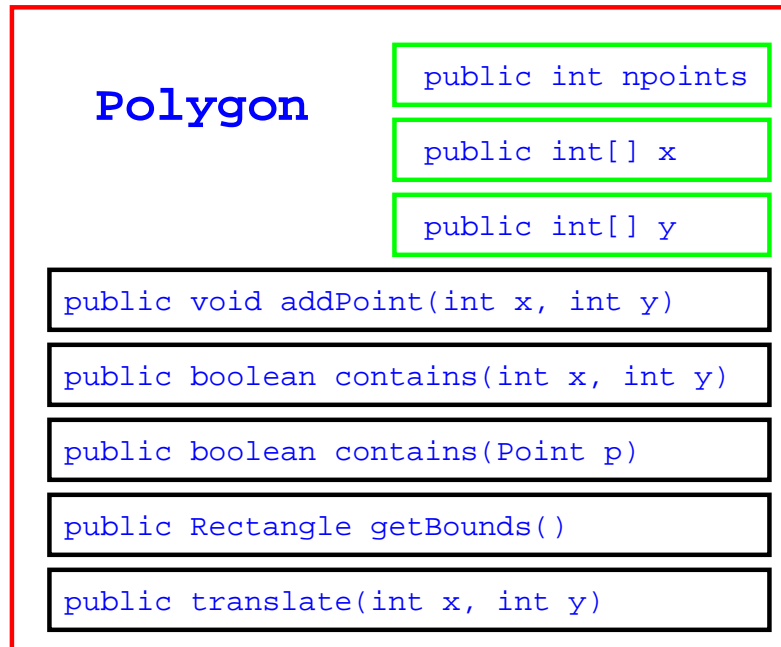


Verkleinerung und Verschiebung des Rechtecks

```
public Rectangle intersection(Rectangle r);  
public void translate(int x, int y);
```



Polygon



`Polygon`-Objekte können mit der `drawPolygon` eines `Graphics`-Objektes gezeichnet werden.

ACHTUNG: Man sollte die Instanz-Variablen niemals direkt verändern, da sonst die Funktionalität von `addPoint` nicht mehr gewährleistet ist!

Graphics und Color

Bislang hatte keine Methode eine Möglichkeit, die Farbe des gezeichneten Objektes festzulegen. Dies liegt daran, dass alle Objekte stets mit der *aktuellen Farbe* des `Graphics`-Objektes gezeichnet werden. Auf diese kann mit den Methoden

```
void setColor(Color c) und Color getColor()
```

zugegriffen werden.

`Color`-Objekte erzeugt man über Konstruktoren im RGB-Farbmodell.

```
public Color(int red, int green, int blue)
public Color(float red, float green, float blue)
public Color(int rgb)
```

Für `int`-Werte liegt der zulässige Bereich zwischen 0 und 255, für `float`-Werte zwischen 0.0 und 1.0. Der dritte Konstruktor interpretiert `rgb` bitweise (`0x00RRGGBB`).

Zudem stehen die Methoden `darker()` und `brighter()` in `Color` zur Verfügung.

Color-Konstanten

Color.black	schwarz
Color.blue	blau
Color.cyan	türkis
Color.darkGray	dunkelgrau
Color.gray	grau
Color.green	grün
Color.lightGray	hellgrau
Color.magenta	deutsche telekom
Color.orange	orange
Color.pink	rosa
Color.red	rot
Color.white	weiss
Color.yellow	gelb

Schriftausgabe mit **Graphics**

`void drawString(String text, int x, int y)`
schreibt `text` in der aktuellen Farbe und mit dem aktuellen Zeichensatz an der Stelle `(x,y)` auf den Bildschirm/Drucker/...

`String` ist die Java-Klasse zur Speicherung von Zeichenketten.

Einfache Beispiele für Stringoperationen:


```
String a = "Hello";
char data[] = {'H','e','l','l','o'};
String a1 = new String(data);
String b = "World";
String c = a + " " + b; // c enthält "Hello World"
System.out.println(a.equals(b)); // false
System.out.println(a.equals(a1)); // true
System.out.println("METHOD".substring(1,4)); // ETH
System.out.println(" abc ".trim()); // abc
System.out.println("abcdefg".startsWith("a")); // true
System.out.println(a.length()); // 5
```

Font

Der aktuelle Font wird mit Hilfe eines `Font`-Objektes gesetzt. Dieses beinhaltet Informationen über den Fontnamen, Stil und die Grösse.

Konstruktor:

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



- Grössenangabe in 1/72 Zoll (Punkt)

- `Font.BOLD`, `Font.ITALIC`, `Font.PLAIN`

- `Serif`, `SansSerif`, `Monospaced`, `Dialog`, `DialogInput...`

Eine Liste der verfügbaren Fonts liefert

```
public String[]  
Toolkit.getDefaultToolkit().getFontList()
```

FontMetrics

Die metrischen Daten eines Fonts werden nicht von einem `Font`-Objekt zur Verfügung gestellt, sondern — weil die Repräsentation deviceabhängig ist — über ein separates `FontMetrics`-Objekt.

Dieses erhält man für den aktuellen Font über die `getFontMetrics()`-Methode von `Graphics` oder zu einem speziellen `Font font` aus dem *Toolkit* mittels `Toolkit.getDefaultToolkit().getFontMetrics(font)`

`FontMetrics`-Objekte stellen Informationen über die Höhe und Breite von Text (in Pixeln) zur Verfügung.

Wichtigste Methoden:

```
public int getMaxAscent()
```

liefert die Höhe über der *baseline*

```
public int getMaxDescent()
```

liefert die Tiefe unter der *baseline*

```
public int stringWidth(String string)
```

liefert die Breite des Strings.

Beispiel: Zentrieren von Text

```
import java.awt.*;
import java.applet.*;

public class Center extends Applet {

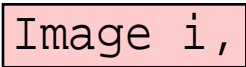
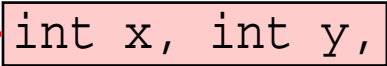
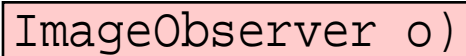
    String text = "Beispieltext";

    public void paint(Graphics g) {
        Dimension d = getSize();
        g.setFont(new Font("SansSerif",Font.BOLD,18));
        FontMetrics fm = g.getFontMetrics();
        int textwidth = fm.stringWidth(text);
        int maxA = fm.getMaxAscent();
        int maxD = fm.getMaxDescent();
        g.drawString( text,
                     (d.width - textwidth) / 2,
                     d.height / 2);
        g.drawRect( (d.width - textwidth) / 2 - 3,
                    d.height / 2 - maxA - 3,
                    textwidth + 6,
                    maxA + maxD + 6);
    }
}
```

Zeichnen von Bitmaps

`Graphics` unterstützt das Zeichnen von Bitmaps, die in `Image`-Objekten gespeichert werden.

Methoden:

```
public boolean drawImage( Image i,  
 int x, int y,  
 ImageObserver o)
```

- ein abstraktes Bild-Objekt
- ein Beobachter — z.B. `this`
- linke obere Ecke

Es stehen diverse Varianten zur Verfügung, die automatisches *scaling* und *cropping* unterstützen.

Image

Ein `Image` ist ein darstellbares Objekt, welches im Hauptspeicher gehalten wird. Das AWT unterstützt Bilddateien in verschiedenen Formaten (u.a. GIF und JPEG).

`Image` selbst hat keine Konstruktoren, man erhält `Image`-Objekte über

- `getImage()` in Applets
- `getImage()` in Toolkits
- `createImage()` in Komponenten
- `createImage()` in Toolkits.

→ lädt ein Bitmap aus einer Datei oder übers Netz

→ erzeugt Speicherplatz für ein (leeres) Bitmap

ImageObserver

Da Java für das WWW designed wurde, werden `Image`-Objekte nicht sofort geladen, sondern erst, wenn sie auf den Bildschirm gebracht werden.

Daher muss ein Protokoll definiert werden, welches das Laden des Bildes initiiert und überwacht.

Alle vom AWT zur Verfügung gestellten Bildschirm-Komponenten implementieren dieses Protokoll, `ImageObserver`, und für einfache Zwecke genügt es, in den `drawImage`-Aufrufen die jeweils malende Komponente (also `this`) anzugeben.

Der eigentliche Ladeprozess beginnt erst, wenn der `drawImage`-Aufruf stattfindet, nicht wenn `getImage` aufgerufen wird. Der Fortschritt und das Ende des Ladeprozesses wird dem `ImageObserver` mitgeteilt. Die AWT-Komponenten reagieren darauf mit `repaint`-Ereignissen.

Beispiel: Images in Applets

```
import java.awt.*;
import java.applet.*;

public class ImageDemo extends Applet {

    Image im;

    public void init() {

        im = getImage(getDocumentBase(), "monitor.gif");

    }

    public void paint(Graphics g) {
        g.drawImage(im,0,0,this);
        // "this" ist der Observer
    }

}
```

liefert ein [URL](#)-Objekt,  welches dem *Homedirectory* des Applets entspricht

Beispiel: **Images** in **Applets**

```
import java.awt.*;
import java.applet.*;

public class ImageDemo2 extends Applet {

    Image im;

    public void init() {
        im = getImage(getDocumentBase(), "monitor.gif");
    }

    public void paint(Graphics g) {
        Dimension d = getSize();
        g.drawImage(im,0,0,d.width,d.height,this);
        // automatisches Skalieren --- funktioniert!
    }

}
```

Obwohl im Augenblick des `drawImage`-Aufrufs die Grösse des Bitmaps noch nicht bekannt ist, funktioniert das Applet.

Clipping

Es ist möglich, eine Clipping-Region zu setzen, das heisst den Effekt der Zeichenoperationen im `Graphics`-Objekt einzuschränken.

In Java 1.0 funktioniert dies nicht sinnvoll.

In Java 1.1 funktioniert es besser, aber noch nicht völlig zufriedenstellend.

Wenn das Window-System nur partielle *redraws* wünscht, sendet es `update`-Aufrufe (statt `paint`) mit `Graphics`-Objekten, die eine Clippingregion gesetzt haben.

Achtung: Das normale `update` löscht die Clipping-Region mit der Hintergrundfarbe (`getBackground()`) und ruft danach `paint` auf.

Redraws, die durch `repaint()` initiiert werden, rufen direkt `paint` auf.

Konstrukturen für `Graphics`

Normalerweise erhält man `Graphics`-Objekte direkt vom Windowsystem. Es gibt aber Fälle, in denen man selbst welche erzeugen kann und muss.

Methoden in `Graphics`:

```
public Graphics create()
```

erzeugt eine Kopie des `Graphics`-Objekt.

Sinnvoll bei der Verwendung von `Clipping`

```
public Graphics create(int x, int y,  
                       int width, int height)
```

erzeugt ein neues `Graphics`-Objekt für das durch `x`, `y`, `width` und `height` spezifizierte Rechteck. Der Punkt `(x,y)` wird zu `(0,0)` im neuen `Graphics`-Objekt.

`Graphics`-Objekte für `Image`-Objekte, die “be-malt” werden dürfen (das heisst, sie wurden mit `createImage` erzeugt und nicht aus einer Datei gela-den) können mit der Methode

```
public Graphics getGraphics()
```

erzeugt werden.

Ausblick

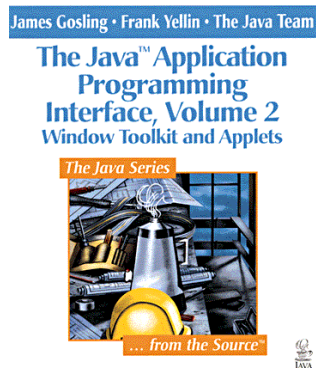
Java2D-API

- Standard ab Java 1.2
- aufwärtskompatibel zum Standard-AWT
- Postscript-ähnliches rendering
- sinnvollerer OO-Ansatz
- verschiedene Farbmodelle
- <http://java.sun.com/products/java-media/2D>

Literaturhinweise



Java AWT Reference
John Zukowski
O'Reilly
(ISBN 1-56592-240-9)



The Java[tm] Application Programming Interface
Volume 2: Window Toolkit and Applets
James Gosling, Frank Yellin, Java Team
Addison Wesley
(ISBN 0-201-63459-7)



...und natürlich die Original-
Dokumentation von Javasoft auf
<http://www.javasoft.com/products/jdk/1.1/>