Grundlagen der Informatik für Ingenieure I

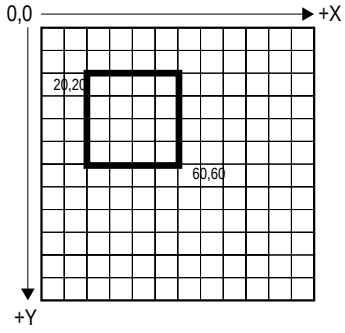
7. Die Graphics Class

7 Die *Graphics Class*

- Die Methode ***paint()*** der ***Graphics***-Klasse erhält als Aufrufparameter ein Objekt der ***Graphics-***Klasse mit diversen Methoden, graphischen Output zu erzeugen:

```
public void paint( Graphics g )
```

- Üblicherweise werden die einzelnen Methoden mit “Punkt”-Parametern versorgt, die die Lage des Objekts in einem Koordinatensystem festlegen. Dieses Koordinatensystem ist so definiert, daß der Punkt (0,0) in der oberen linken Ecke des Appletfensters liegt.



7.1 Linien, Rechtecke

◆ Zeichnen einer Linie

```
g.drawLine( xanf,yanf,xend,yend );
```

◆ Rechtecke
sind in drei Varianten möglich und zwar als

- Rechtecke mit eckigen Ecken
- Rechtecke mit runden Ecken
- “3-dim-Rechtecke” mit einer Schattenbegrenzung

Außerdem kann man sie ausgefüllt mit einer Farbe oder nur als Liniengraphik zeichnen.

Einige Beispiele:

```
g.drawRect( 20,20,40,40 );
g.fillRoundRect( 120,20,60,60,20,20 );
g.drawRoundRect( 20,20,60,60,60,60 ); //ein Kreis!
```


7.1 Linien, Rechtecke

◆ Rechtecke - ein vollständiges Beispiel:

```
import java.awt.Graphics;

public class MyRect extends java.applet.Applet {

  public void paint( Graphics g ) {

    g.drawRect( 20,20,60,60 );
    g.fillRect( 120,20,60,60 );
    g.fillRoundRect( 20,120,60,60,20,20 );
    g.drawRoundRect( 120,120,60,60,60,60 ); // Kreis
  }
}
```

7.1 Linien, Rechtecke

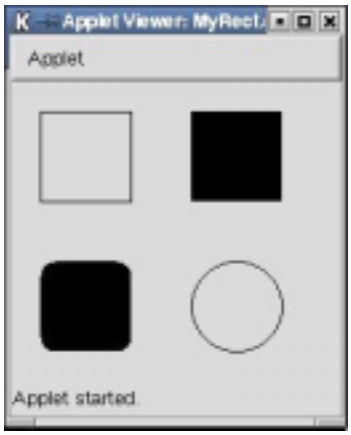
◆ Rechtecke - ein vollständiges Beispiel (HTML-File):

```
<html>
<head>
<title>Rectangles</title>
</head>
<body bgcolor="white">
<h2>Kapitel Sieben: Rectangles</h2>
<p>
A simple graphics example that draws rectangles:
<p>
<applet code="MyRect.class" width=200 height=120>
</applet>
<p>
<a href="MyRect.java">The Source</a>
</body>
</html>
```


7.1 Linien, Rechtecke

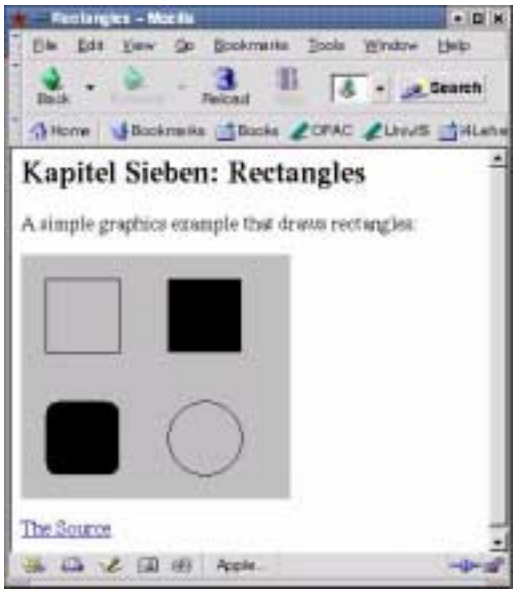
◆ Rechtecke - ein vollständiges Beispiel:

- Ergebnis mit Appletviewer:



7.1 Linien, Rechtecke

- Rechtecke - ein vollständiges Beispiel:
  - Ergebnis mit Browser (Mozilla):

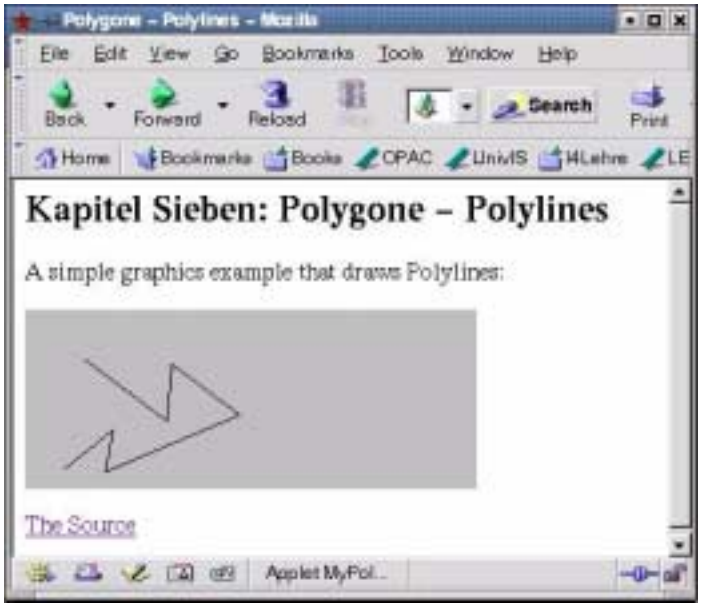


7.2 Polygone

- **Polylines** sind Linien mit beliebiger Anzahl von Geraden.
- Sind Anfangs- und Endpunkt identisch entsteht ein **Polygon** (Mehreck)
- Polylines werden beschrieben durch drei Parameter
  - *array of int:* x-Kordinaten
  - *array of int:* y-Koordinaten
  - *int:* Anzahl der Punkte

```
import java.awt.Graphics;

public class MyPoly extends java.applet.Applet {

    public void paint( Graphics g ) {
        int exes[] = { 39,94,97,142,53,58,26 };
        int whys[] = { 33,74,36,70,108,80,106 };
        int pts = exes.length;
        g.drawPolyline( exes,whys,pts );
    }
}
```

7.2 Polygone



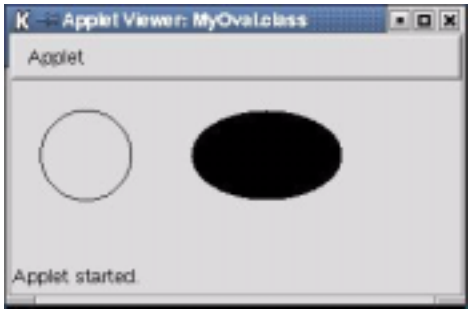
7.3 Ovale, Kreise

- Ovale/Kreise
  werden in Rechtecke/Quadrate “eingeschrieben”, also analog zu den Rechteck-*methods* verwendet:

```
import java.awt.Graphics;

public class MyOval extends java.applet.Applet {
  public void paint( Graphics g ) {
    g.drawOval( 20,20,60,60 );
    g.fillOval( 120,20,100,60 );
  }
}
-----------------------------------------------------------------
<html>
<head>
<title>Ovals</title>
</head>
<body bgcolor="white">
<h2>Ovals </h2>
<p>A simple graphics example that draws ovals:
<br><applet code="MyOval.class" width=300 heigth=120>
</applet>
<p>
</body>
</html>
```

7.3 Ovale, Kreise

- Ergebnis:



7.4 Bogen (Args)

◆ Bogen (Args)

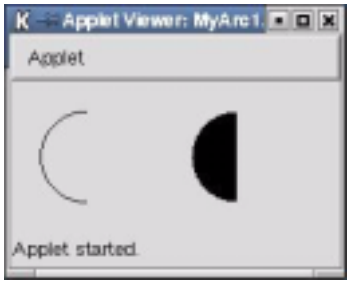
- werden wie Ovale konstruiert
- zusätzlich definiert man den Anfangspunkt und den Endpunkt mit einer Gradangabe,
  wobei:
  - 0^o 03:00
  - 90^o 12:00
  - 180^o 09:00
  - 270^o 06:00

  entspricht.

7.4 Bogen (Args)

Ein Beispiel:

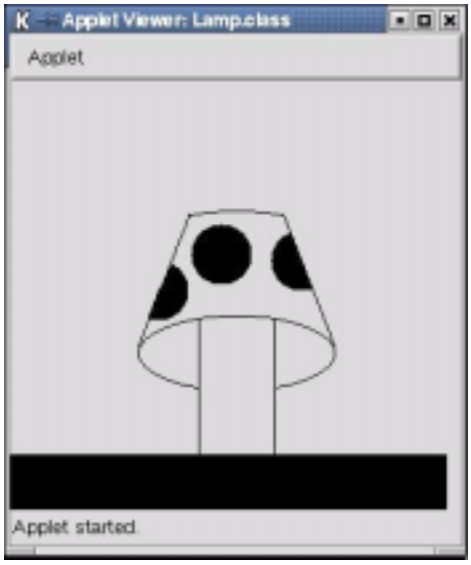
```
import java.awt.Graphics;

public class MyArc1 extends java.applet.Applet {

    public void paint( Graphics g ) {
        g.drawArc( 20,20,60,60,90,180 );
        g.fillArc( 120,20,60,60,90,180 );
    }
}
```



7.5 Ein vollständiges Beispiel:

```
import java.awt.*;
public class Lamp extends java.applet.Applet {
   public void paint( Graphics g ) {
        // the lamp platform
        g.fillRect( 0,250,290,290 );
        // the base of the lamp
        g.drawLine( 125,250,125,160 );
        g.drawLine( 175,250,175,160 );
        // the lamp shade, top and bottom edges
         g.drawArc( 85,157,130,50,-65,312 );
         g.drawArc( 85,87,130,50,62,58 );
         // lamp shade, sides
         g.drawLine( 85,177,119,89 );
         g.drawLine( 215,177,181,89 );
         // dots on the shade
         g.fillArc( 78,120,40,40,63,-174 );
         g.fillOval( 120,96,40,40 );
         g.fillArc( 173,100,40,40,110,180 );
   }
}
```

7.5 Ein vollständiges Beispiel:



7.6 Kopieren, Löschen

◆ Kopieren und Löschen:

```
g.copyArea( 0, 0, 100, 100, 100, 0 );
```

Kopiert ein Quadrat 100 x100, lokalisiert in der linken oberen Ecke, horizontal um 100 Pixels nach rechts.

```
g.clearRect(.....);
```

entspricht den anderen *regtangle-methods*. Löschen heißt, das Rechteck mit der Hintergrundfarbe füllen.

Will man den Inhalt des gesamten Applets löschen, kann man wie folgt vorgehen:

```
g.clearRect( 0, 0, getSize().width, getSize().height );
```

7.7 Text, Fonts

◆ Text und Fonts:
- Mit der *Graphics-Class* kann man - wie wir bereits wissen - auch Text ausgeben.
- Dieses geschieht in Zusammenhang mit der
  - *Font-Class* und der
  - *FontMetrics-Class*

◆ Zum Kreieren eines *Font-Objects* benötigt man 3 Parameter:
- den Namen des Fonts; z. B. TimesRoman, `Courier`, Helvetica
- den *style* des Fonts; z. B. plain, **bold** oder *italic* und
- die Größe des Fonts

◆ Beispiel:

```
Font f = new Font( “Helvetica”, Font.BOLD + Font.ITALIC, 20 );
```


7.7 Text, Fonts

7.7 Text, Fonts

◆ Es gibt zwei Methoden Zeichenketten auszugeben:

- mit der *drawString()-method:*

```
g.drawString( “Dies ist der Text”, 10, 100 );
```

- und mit der *g.drawChars()-method:*

```
g.drawChars( array_of_chars, indexFrom, indexTo, 10, 100 );
```

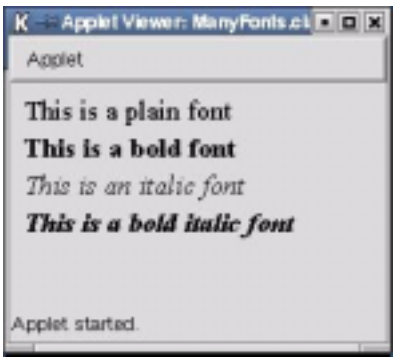
7.7 Text, Fonts

◆ Ein Beispiel:

```
import java.awt.Font;
import java.awt.Graphics;

public class ManyFonts extends java.applet.Applet {

   public void paint( Graphics g ) {
    Font f = new Font( "TimesRoman", Font.PLAIN, 18 );
    Font fb = new Font( "TimesRoman", Font.BOLD, 18 );
    Font fi = new Font( "TimesRoman", Font.ITALIC, 18 );
    Font fbi = new Font( "TimesRoman", Font.BOLD + Font.ITALIC, 18 );

       g.setFont(f);
       g.drawString( "This is a plain font", 10, 25 );
       g.setFont(fb);
       g.drawString( "This is a bold font", 10, 50 );
       g.setFont(fi);
       g.drawString( "This is an italic font", 10, 75 );
       g.setFont(fbi);
       g.drawString( "This is a bold italic font", 10, 100 );
   }
}
```


7.7 Text, Fonts

- Ergebnis mit Appletviewer:



7.7 Text, Fonts

◆ Eine Auswahl von *Font*-Methoden :

Method Name	Class	Action
getFont()	Graphics	Returns the current font object as previously set by setFont()
getName()	Font	Returns the name of the font as a string
getSize()	Font	Returns the current font size (an integer)
getStyle()	Font	Returns the current style of the font (styles are integer constants: 0 is plain, 1 is bold, 2 is italic, 3 is bold italic)
isPlain()	Font	Returns true or false if the font's style is plain
isBold()	Font	Returns true or false if the font's style is bold
isItalic()	Font	Returns true or false if the font's style is italic

7.8 Color

◆ Das Java-24-Bit-Farb-Modell erlaubt es, ca. 16,5 Millionen Farben darzustellen, zusammengesetzt aus drei Komponenten:
- rot, grün, blau: ***RGB***-Modell

◆ Jede Farbe kann in 256 "Nuancen", repräsentiert durch eine Integerzahl zwischen 0 und 255, dargestellt werden. Dies ist genau der Wertebereich eines Bytes.

◆ Wie viele Farben sind genau darstellbar?: 256 * 256 * 256 = 16.777.216
- weiß: 255, 255, 255
- schwarz: 0, 0, 0

◆ Es gibt zwei Möglichkeiten ein Objekt farbig auszugeben:
- man erzeugt ein Objekt der Color-Class, mit Hilfe des ***RGB***-Modells

```
Color c = new Color( 110, 120, 130 )
```

- man verwendet Standardfarben (siehe Tabelle auf der nächsten Seite)

7.8 Color

◆ Standardfarben:Color

Color Name	RGB Value
Color.white	255, 255, 255
Color.black	0, 0, 0
Color.lightGray	192, 192, 192
Color.gray	128, 128, 128
Color.darkGray	64, 64, 64
Color.red	255, 0, 0
Color.green	0, 255, 0
Color.blue	0, 0, 255
Color.yellow	255, 255, 0
Color.magenta	255, 0, 255
Color.cyan	0, 255, 255
Color.pink	255, 175, 175
Color.orange	255, 200, 0

7.8 Color

◆ Die Farben muß man im Graphik-Objekt setzen (analog zu den Fonts):

- Dazu dienen die Methoden:

```
g.setColor( Color.green );
oder
Color c = new Color( 120, 130, 140 );
g.setColor(c);
```

◆ Die Methoden zum Setzen der Hintergrundfarbe stammen aus der java.awt.Component-Class, einer Superklasse des Applets:

```
setBackground( Color.white );
setForeground( Color.black );
```

◆ Korrespondierend gibt es “Auskunftsmethoden” mit denen man den derzeitigen Farbzustand ermitteln kann:

```
getColor();       // definiert in Graphics  Class
getBackground();  // definiert in Component Class
getForeground();  // definiert in Component Class
```

7.8 Color

◆ A Random Color Box:

```
import java.awt.Graphics;
import java.awt.Color;

 public class ColorBoxes extends java.applet.Applet {

     public void paint( Graphics g ) {
         int rval, gval, bval;

         for ( int j = 30; j < ( getSize().height -25 ); j += 30 )
           for ( int i = 5; i < ( getSize().width -25 ); i += 30 ) {
                rval = (int)Math.floor( Math.random() * 256 );
                gval = (int)Math.floor( Math.random() * 256 );
                bval = (int)Math.floor( Math.random() * 256 );

                g.setColor( new Color( rval, gval, bval ) );
                g.fillRect( i, j, 25, 25 );
                g.setColor( Color.black );
                g.drawRect( i-1, j-1, 25, 25 );
            }
     }
}
```


7.8 Color

- Ergebnis mit Appletviewer: