Graphics

- Klasse, die alle grundlegenden Methoden zum Zeichnen und Schreiben bereitstellt
- (draw -Methoden, Methoden zu Fonts und Colors)
- Bereitstellung eines Graphics-Objektes
  - durch das AWT bei Aufruf der paint() Methode
  - durch ImageObject.getGraphics()
  - durch create() kann eine Kopie eines Graphics Object erzeugt werden
  - durch visibleComponent.getGraphics()

Images

- Kommen auf zweierlei Weise zur Anwendung
  - Darstellung von Pixelbildern(.jpg, .gif, .png)
  - Bilden die Grundlage für Offscreendrawing (Zeichnen von Diagrammen, Grafiken ect.)
- Images sind keine Komponenten wie Buttons oder Labels. Images werden über Methoden der Klasse Graphics auf die Oberfläche einer Component ausgegeben.
- Die einfachste Methode ist folgende:

```
public abstract boolean drawImage(Image img,
                                  int x,
                                  int y,
                                  ImageObserver observer)
```

- ImageObserver ist dabei die Component auf der das Bild erscheinen soll (sehr oft this).
- Das Laden eines Bildes erfolgt über die Methode
  - `Img=getToolkit().getImage(FileName);`
  - `Img=getToolkit().getImage(Url);`
- Das Laden der Pixel erfolgt asynchron nachdem die Methode getImage längst verlassen worden ist.
- Der ImageObserver lädt die Pixel, wenn das Bild erstmalig angezeigt werden soll, die Applikation wird aller 100 ms benachrichtigt um die inzwischen nachgeladenen Pixel anzuzeigen.

paint-Methode

- Jede Component oder von ihr abgeleitete Klasse enthält die Methode paint.
- `public void paint(Graphics g)`
- Um ein Bild auszugeben, wird eine Klasse von Component oder Panel abgeleitet und die Methode paint überschrieben.
- In der überschriebenen Methode paint können nun draw-Funktionen aufgerufen werden, um Bilder auszugeben oder zu zeichnen.
- Der Aufruf der Methode paint erfolgt von der virtuellen Maschine automatisch.

```
import java.awt.*;
import java.awt.event.*;
public class ImgPanel extends Panel
{
  private Image Img;

  ImgPanel(Image Img)
  {    this.Img=Img;  }

  public void paint(Graphics g)
  { g.drawImage(Img,0,0,this);  }

  public static void main(String args[])
  {
    Frame    F=new Frame("ImagepanelDemo");
    Image    I=F.getToolkit().getImage(args[0]);
    ImgPanel P=new ImgPanel(I);
    F.add(P);
    F.addWindowListener(new WindowAdapter()....);
    F.pack();
    F.setVisible(true);
  }
}
```

Erste einfache Anwendung, ein Image anzuzeigen. Die Anwendung erscheint oben links am Bildschirm winzig klein und muss erst mit der Maus aufgezogen werden.
Der Grund liegt darin, dass AWT nicht weiß, wie groß unsere Komponente sein möchte.

Größe der abgeleiteten Component

- Soll sich eine selbst programmierte Component bezüglich Ihrer Größe gegenüber dem Layoutmanager genauso verhalten, wie Standardkomponenten, muss die Methode

  `public Dimension getPreferredSize()`
  überschrieben werden.

- Die Methode `getPreferredSize` wird beim Aufbau der Oberfläche automatisch aufgerufen, um die Größe einer Component zu ermitteln.
- Die zurückzugebende Dimension kann aus der Größe des Bildes berechnet werden, **wenn es fertig geladen** ist.

```
public Dimension getPreferredSize()
{
   return new Dimension(Img.getWidth(this),Img.getHeight(this));
}
```

- Genau hier liegt der Hund begraben!
- Das Image wird erst geladen, wenn es denn wirklich dargestellt werden soll, also während des Aufrufes der paint-Methode.
- Einziger Ausweg besteht darin, das Bild vor der Ausgabe über drawImage in der paint-Methode zu laden.
- Das gezielte Laden eines Images kann mit einem Objekt der Klasse MediaTracker erfolgen

MediaTracker

- Erlaubt das Laden von Bildern bevor sie Verwendung finden.
- Verhindert so die „stückweise“ Ausgabe von Bildern.
- Ermöglicht die Bestimmung der Größe von Bildern vor ihrer Verwendung

```
private Image Img;
ImgPanelMT(Image Img) // Constructor
{
  this.Img=Img;
  MediaTracker M=new MediaTracker(this);
  M.addImage(Img,1);
  try {M.waitForID(1);}catch (Exception e){}
}
```

- Mit der Methode addImage können dem Mediatracker Aufträge zum Laden von Bildern übertragen werden.
- Das Laden der Bilder erfolgt nun asynchron
- Mit waitForID kann gewartet werden, bis ein bestimmtes Bild geladen worden ist.
- Mit waitForAll kann gewartet werden, bis alle Bilder geladen worden sind.
- Die Anzeige des Bildes erfolgt dann ich gleicher Weise in der Methode paint, wie im ersten Beispiel.

Skalieren von Bildern

- Bilder werden in ihrer Originalauflösung ausgegeben
- Ist das Bild zu groß, wird nur ein Bildausschnitt sichtbar.
- Zwei Möglichkeiten ein Bild zu skalieren:
  - Skaliertes Bild als Objekt erzeugen
  - Bild bei der Ausgabe via drawImage skalieren

Erzeugen skaliertes Image

```
public ImgComponent(Image I)
{
 Img=I.getScaledInstance(1440, 1050, Image.SCALE_SMOOTH);
 MediaTracker M=new MediaTracker(this);
 M.addImage(this.Img,1);
 try {M.waitForID(1);}catch (Exception e){}
}
```

Skalieren durch drawImage

```
public void paint(Graphics g)
{
  g.drawImage(Img,0,0,1440, 1050,this);
}
```

Filter

- Filter ermöglichen die Berechnung eines neuen Bildes aus den Pixeln eines vorhandenen Bildes.
- Es gibt
  - vorgefertigte Filter, wie den CropImageFilter zum Ausschneiden.
  - Filter mit zu überschreibenden Filterfunktionen, wie den RGBImageFilter

Vorgehen

- Erzeugen einer Instanz des Filters
- Erzeugen einer FilteredImageSource
- Erzeugen des neuen Bildes

```
CropImageFilter F=new CropImageFilter(x,y,width,height);
FilteredImageSource S=
               new FilteredImageSource(Img.getSource(),F);
Img=createImage(S);
```

Oder:

```
Img=createImage(new FilteredImageSource(
Img.getSource(),
                  new CropImageFilter(x,y,width,height)));
```

RGBImageFilter

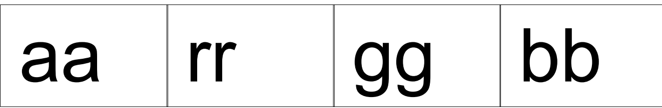
```
// Aus der api-Dokumentation
class RedBlueSwapFilter extends RGBImageFilter
{
    public RedBlueSwapFilter()
    {
        // The filter's operation does not depend on the
        // pixel's location, so IndexColorModels can be
        // filtered directly.
        canFilterIndexColorModel = true;
    }

    public int filterRGB(int x, int y, int rgb)
    {
        return ((rgb & 0xff00ff00)
                | ((rgb & 0xff0000) >> 16)
                | ((rgb & 0xff) << 16));
    }
}
```

True: Filterung positionsunabhängig
False: Filterung von der Position abhängig

Ein Pixel=4Byte
0x xx000000 Alpha
0x 00xx0000 Red
0x 0000xx00 Green
0x 000000xx Blue

Das Pixel



aa	rr	gg	bb

FF	FF	00	00	Rot	
FF	00	FF	00	Grün	
FF	00	00	FF	Blau	

Ein Pixel
32 Bit
4 Byte
aa=0x00 transparent
aa=0xFF undurchsichtig

Warum gibt es keine Klasse Pixel mit get- und set-Funktionen?
Der Zugriff auf die Farbwerte jedes Pixels wäre wohl zu langsam.
Die Filterfunktion wird für jedes Pixel des Bildes aufgerufen.
Laufzeiteffizienz ist hier in hohem Maße gefragt.

Aufbau der Filterfunktion

```
int filterRGB(int x, inty, int rgb)
{
int a, r, g, b;
  a=(rgb & 0xff000000) >>> 24;//oder
//a=(rgb & 0xff000000) /0x1000000;
  r=(rgb & 0x00ff0000) >>> 16;//oder
//r=(rgb & 0x00ff0000) /0x10000;
  g=(rgb & 0x0000ff00) >>>  8;//oder
//g=(rgb & 0x0000ff00) /0x100;
  b=(rgb & 0x000000ff);

// jetzt kann mit den Farbwerten
// oder dem Alphakanal gerechnet werden

// zuletzt neues Pixel zusammenbauen:
return a * 0x1000000 // oder a<<24
     + r * 0x10000   // oder r<<16
     + g * 0x100     // oder g<< 8
     +b;
}
```

Ein Rechenbeispiel, bei dem der Rotanteil verändert werden soll zeigt, warum die einzelnen Farbteile nach rechts geschoben werden müssen:

```
rgb=0xFF220000
&     0x00FF0000
--------------
      0x00220000
+     0x00FF0000
--------------
      0x01210000
/              2
--------------
      0x00908000
```

Übergriff auf Grün!
Falschfarben sind Folge

Ein einfacher „disable“-Filter

```
class DisableFilter extends RGBImageFilter
{
  DisableFilter()
  {System.out.println("new DisableFilter");}

  public int filterRGB(int x, int y, int rgb)
  {
    {
      int a,r,g,b;
      a =  rgb & 0xff000000;
      r = ((((rgb & 0xff0000)>>>16)+0xff)/2)<<16;
      g = ((((rgb & 0x00ff00)>>> 8)+0xff)/2)<< 8;
      b = (((rgb & 0x0000ff)       +0xff)/2)    ;
      return a | r | g | b;
    }
}
```

Berechnung eines gefilterten Randes

- Constructor mit Argument ist notwendig, um die Breite des Randes in eine Instanzvariable zu übernehmen
- `canFilterIndexColorModel = false;` um positionsabhängig zu filtern.
- Filterfunktion muss nun die Pixelposition x und y auswerten und davon abhängig filtern.
- Um einen gleitenden Rand zu berechnen, kann der Alphakanal berechnet, oder anteilig Pixel- und Hintergrundfarbe verrechnet werden.

```
class RedBorder extends RGBImageFilter
{
    int a,w,h,b;
    public RedBorder(int w, int h, int b)
    {
      canFilterIndexColorModel = false;
      this.w=w; this.h=h; this.b=b;
    }
    public int filterRGB(int x, int y, int rgb)
    {

       if (x<b    )  return 0xffff0000;//rgb|a|0xff0000;
       if (y<b    )  return 0xffff0000;//rgb|a|0xff0000;
       if ((w-x)<b)  return 0xffff0000;//rgb|a|0xff0000;
       if ((h-y)<b)  return 0xffff0000;//rgb|a|0xff0000;
       return rgb;
    }
}
```