20.2 Schreiben mit `Graphics`

Um (z.B. auf dem Bildschirm) schreiben zu können, benötigt man eine Schrift (Font).

Eine Schrift ...

- gehört zu einer Schrift-Familie;
- besitzt eine Ausprägung
- ... und eine Größe.

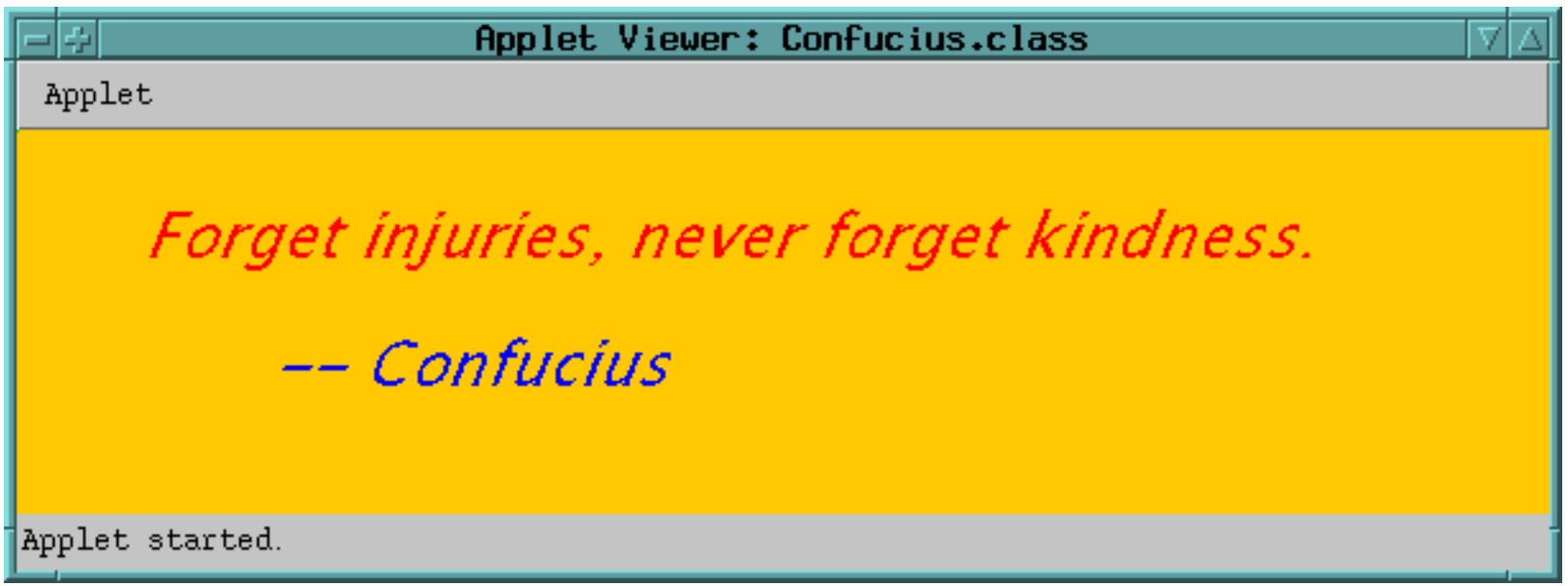
Betrachten wir erstmal ein Applet, das Zeichen mit dem voreingestellten Font darstellt ...

Beispiel: Das Confucius-Applet

```
import java.applet.Applet;
import java.awt.*;
public class Confucius extends Applet {
   public void paint (Graphics page) {
      setBackground(Color.orange);
      page.setColor(Color.red);
      page.drawString ("Forget injuries, never forget kindness.",
                       50, 50);
      page.setColor(Color.blue);
      page.drawString ("-- Confucius", 70, 70);
   } //      method paint()
}    //      end of class Confucius
```

- `public void drawString(String str, int x, int y);` ist eine Objekt-Methode der Klasse `Graphics`, die den String `str` ab der Position `(x, y)` in der aktuellen Farbe auf den Bildschirm malt.
- Der Effekt:

- Die Qualität der Wiedergabe ist so schlecht, weil
  - die Zeichen klein sind im Verhältnis zur Größe der Pixel;
  - der Screenshot für die Folie skaliert wurde :-)
- Wollen wir ein anderes Erscheinungsbild für die Zeichen des Texts, müssen wir einen anderen Font wählen ...



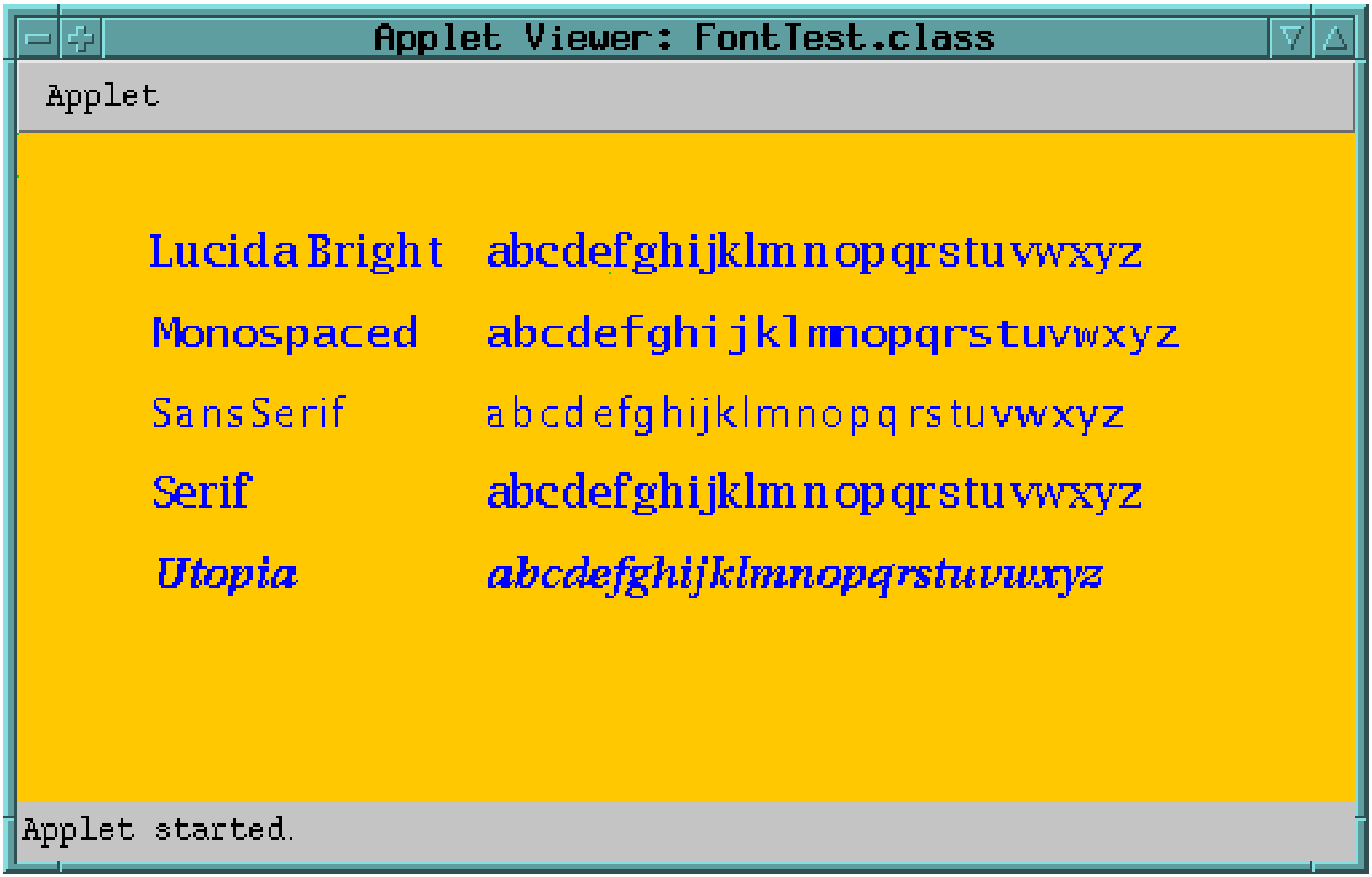
```
import java.applet.Applet;
import java.awt.*;
public class Confucius extends Applet {
   private Font font = new Font("SansSerif",Font.ITALIC,24);
   public void init() {
      setBackground(Color.orange);
      }
   public void paint (Graphics page) {
      page.setColor(Color.red);
      page.setFont(font);
      page.drawString ("Forget injuries, never forget kindness.",
                       50, 50);
      page.setColor(Color.blue);
      page.drawString ("-- Confucius", 100, 100);
   } //      method paint()
}    //      end of class Confucius
```

- Ein Java-Font wird repräsentiert durch ein Objekt der Klasse `Font` (wer hätte das gedacht? ;-)
- Eine Schrift-Familie fasst eine Menge von Schriften zusammen, die gewisse graphische und ästhetische Eigenschaften gemeinsam haben.
- SansSerif zum Beispiel verzichtet auf sämtliche Füßchen und Schwänzchen ...
- Einige Schrift-Familien, die mein Java-System kennt:

  Lucida Bright, Utopia,

  Monospaced, SansSerif, Serif

- Die untere Reihe enthält logische Familien, die verschiedenen konkreten (vom System zur Verfügung gestellten) Familien entsprechen können.


Applet Viewer: FontTest.class
Applet
LucidaBright abcdefghijklmnopqrstuvwxyz
Monospaced abcdefghijklmnopqrstuvwxyz
SansSerif abcdefghijklmnopqrstuvwxyz
Serif abcdefghijklmnopqrstuvwxyz
Utopia abcdefghijklmnopqrstuvwxyz
Applet started.

- Als Ausprägungen unterstützt Java Normalschrift, **Fettdruck**, *Schrägstellung* und ***fette Schrägstellung***.
- Diesen entsprechen die (`int`-) Konstanten `Font.PLAIN`, `Font.BOLD`, `Font.ITALIC` und `(Font.BOLD + Font.ITALIC)`.
- Als drittes benötigen wir die Größe der Schrift (in Punkten).
- Der Konstruktor `public Font(String family, int style, int size);` erzeugt ein neues `Font`-Objekt.
- Die Objekt-Methoden:
  - `public Font getFont();`
  - `public void setFont(Font f);`
  - `public void drawString(String str, int x, int y);`

  ... der Klasse `Graphics` erlauben es, die aktuelle Schrift abzufragen bzw. zu setzen und in der aktuellen Schrift zu schreiben (`(x, y)` gibt das linke Ende der Grundlinie an)

Achtung:

- Für die Positionierung und den Zeilen-Umbruch ist der Programmierer selbst verantwortlich :-)
- Dazu sollte er die Dimensionierung seiner Schrift-Zeichen bzw. der damit gesetzten Worte bestimmen können ...
- Dafür ist die abstrakte Klasse `FontMetrics` zuständig ...

Ein Beispiel:

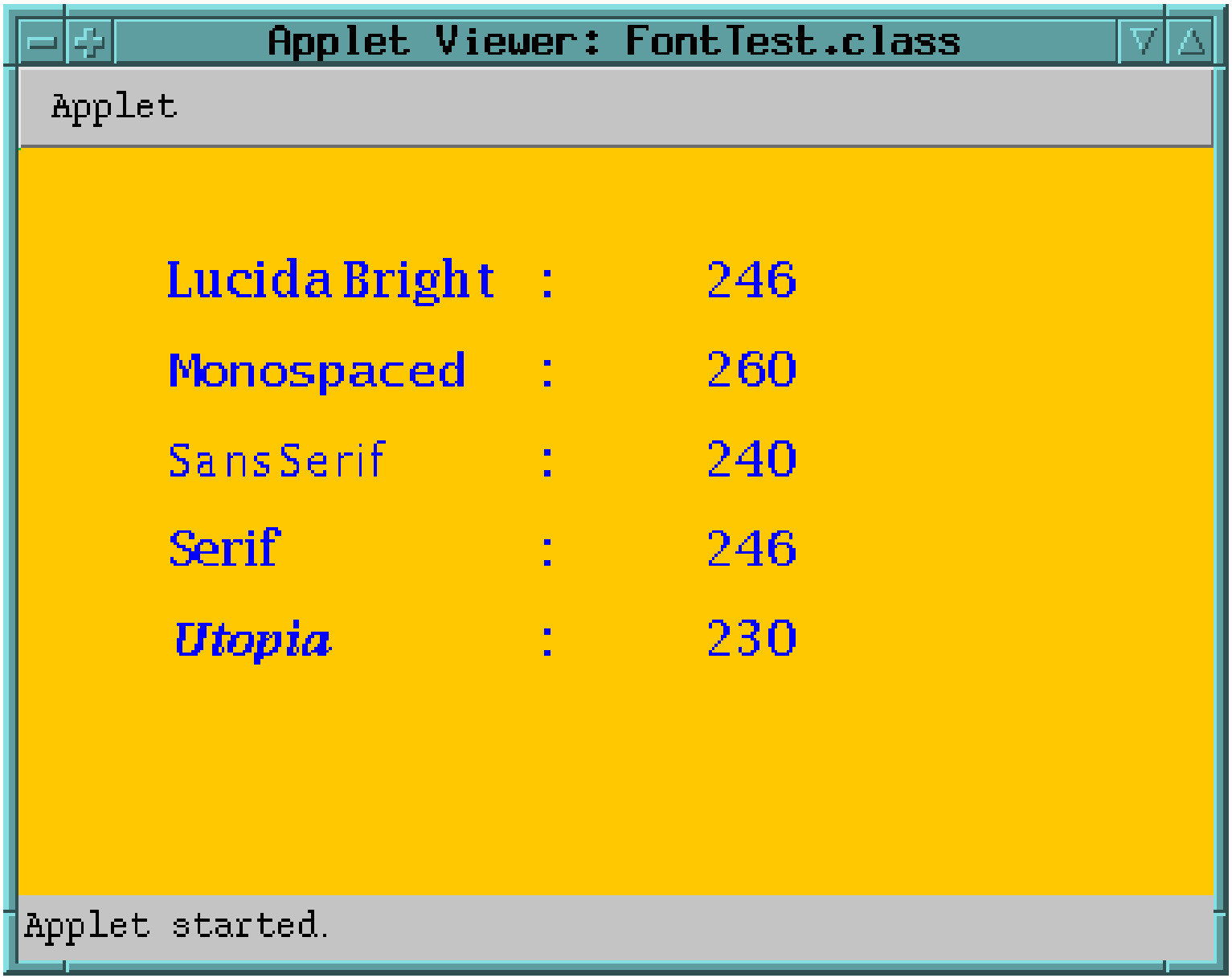
```
import java.applet.Applet;
import java.awt.*;
public class FontTest extends Applet {
   private String[] fontNames = {
      "Lucida Bright", "Monospaced",
      "SansSerif", "Serif", "Utopia"
      };
   private Font[] fonts = new Font[5];
   public void init() {
      for(int i=0; i<5; i++)
         fonts[i] = new Font(fontNames[i],Font.PLAIN,16);
   }
   public void start() {
      setBackground(Color.orange);
      }
   ...
```

```
   ...
   public void paint (Graphics page) {
      page.setColor(Color.blue);
      String length; FontMetrics metrics;
      for(int i=0; i<5; i++) {
         page.setFont(fonts[i]);
         page.drawString(fontNames[i],50,50+i*30);
         page.setFont(fonts[3]);
         page.drawString(":",175,50+i*30);
         metrics = page.getFontMetrics(fonts[i]);
         length = Integer.toString(metrics.stringWidth
                  ("abcdefghijklmnopqrstuvwxyz"));
         page.drawString(length,230,50+i*30);
      }
   }    //      method paint
}       //      class FontTest
```

- Die Objekt-Methoden

  ```
  public FontMetrics getFontMetrics();
  public FontMetrics getFontMetrics(Font f);
  ```

  der Klasse `Graphics` liefern das `FontMetrics`-Objekt für die aktuelle Schrift bzw. die Schrift `f`.
- Die Objekt-Methode `public int stringWidth(String string);` liefert die Länge von `string` in der aktuellen Font-Metrik ...
- Die Klassen `Font` und `FontMetrics` bieten noch viele weitere Einzelheiten bzgl. der genauen Größen-Verhältnisse :-)
- Die Objekt-Methode `public int getHeight();` liefert z.B. die maximale Höhe eines Worts in der aktuellen Schrift.
- Tja, und so sieht dann das Applet im `appletviewer` aus ...


Applet Viewer: FontTest.class
Applet
LucidaBright : 246
Monospaced : 260
SansSerif : 240
Serif : 246
Utopia : 230
Applet started.

20.3 Animation

- Animation ist eine Bewegung vortäuschende Abfolge von Bildern (evt. mit Ton unterlegt :-)
- Für das menschliche Auge genügen 24 Bilder pro Sekunde.
- In der Zeit dazwischen legen wir das Applet schlafen ...

```
import java.applet.Applet;
import java.awt.*;
public class Grow extends Applet {
   public void start() { setBackground(Color.orange); }
   public void grow(int x, int y, Color color, Graphics g) {
      g.setColor(color);
      for(int i=0; i<100; i++) {
         g.fillRect(x,y,2*i,i);
         try {Thread.sleep(40);}
         catch (InterruptedException e) {
            System.err.println("Growing interrupted!");
         }
      }
   }
   ...
```

- Die Objekt-Methode `grow()` erhält als Argument eine Position, eine Farbe und ein `Graphics`-Objekt `g`.
- An die gegebene Position malt es in der gegebenen Farbe sukzessive ein größer werdendes gefülltes Rechteck.
- Zwischen zwei Bildern schläft es 40 Millisekunden lang ...

```
    ...
    public void paint(Graphics g) {
       grow(50,50,Color.red,g);
       grow(100,100,Color.blue,g);
       grow(150,150,Color.green,g);
    }
} // end of Applet Grow
```

- Das Ergebnis sieht miserabel aus :-(
- Das Applelt ruckelt ("Flicker").
- Offenbar ist das Malen eines Rechtecks ein längerer Vorgang, der vom Auge durchaus wahrgenommen wird ...

Lösung: Buffering

- Statt direkt auf den Bildschirm zu malen, stellen wir die Pixel-Matrix erst in einem (unsichtbaren) Puffer her.
- Den Puffer zeigen wir dann auf einen Schlag an!

```
import java.applet.Applet;
import java.awt.*;
public class BufferedGrow extends Applet {
   private Image buffer;
   private Graphics g;
   public void init() {
      buffer = createImage(500,300);
      g = buffer.getGraphics();
   }
   public void start() {
      g.setColor(Color.orange);
      g.fillRect(0,0,500,300);
   }
   public void destroy() { g.dispose(); }
   ...
```

- Objekte der Klasse `Image` enthalten eine (i.a. implementierungsabhängige) Darstellung der Pixel-Representation eines Bilds.
- `public Image createImage(int width, int height);` (Objekt-Methode einer Oberklasse von `Applet`) liefert ein neues `Image`-Objekt der gegebenen Breite und Höhe.
- `public Graphics getGraphics();` (Objekt-Methode der Klasse `Image`) liefert ein `Graphics`-Objekt für das `Image`-Objekt.

  Malen auf diesem `Graphics`-Objekt modifiziert die Pixel des `Image`-Objekts.
- `public void dispose();` (Objekt-Methode der Klasse `Graphics`) gibt das `Graphics`-Objekt wieder frei (sollte man immer tun :-)

```
...
public void grow(int x, int y, Color color, Graphics page) {
   g.setColor(color);
   for(int i=0; i<100; i++) {
      page.drawImage(buffer,0,0,this);
      g.fillRect(x,y,2*i,i);
      try {Thread.sleep(40);}
      catch (InterruptedException e) {
         System.err.println("Growing interrupted!");
      }
   }
}
...
```

```
   ...
   public void paint(Graphics page) {
      page.setClip (0,0,500,300);
      grow(50,50,Color.red,page);
      grow(100,100,Color.blue,page);
      grow(150,150,Color.green,page);
   }
} // end of Applet BufferedGrow
```

- `public void setClip(int x, int y, int width, int height)` setzt den Bereich, der neu gemalt werden soll :-)
- Ein `Image`-Objekt enthält die komplette Pixel-Information.
- `public boolean drawImage(Image buf, int x, int y, ImageObserver obs);`
  `public boolean drawImage(Image buf, int x, int y, int width, int height, ImageObserver obs);`
  (Objekt-Methoden der Klasse `Graphics`) malen das Bild `buf` an die Stelle `(x, y)` (evt. skaliert auf die gegebene Größe).
- `ImageObserver` ist dabei ein Interface, das von `Applet` implementiert wird.

Hintergrund:

- Manchmal werden fertige Bilder aus dem Internet gezogen :-)
- Dabei helfen folgende Objekt-Methoden der Klasse `Applet`:

```
public Image getImage(URL base, String file);

public URL getCodeBase();

public URL getDocumentBase();
```

  ...

- Bis ein Bild ganz geladen ist, kann es evt. bereits partiell angezeigt werden.
- Damit das klappt, muss eine Interaktion zwischen dem Empfänger-Thread und der Hardware-Komponente erfolgen, die die Pixel einsaugt ...

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

public class DrFun extends Applet {
   public void paint(Graphics g) {
      Image image = getImage(getDocumentBase(),"df20050201.jpg");
      g.drawImage(image,0,0,this);
   }
} // end of Applet DrFun
```

... zeigt die jpg-Datei df20050201.jpg auf dem Bildschirm an:

Applet-Ansicht: DrFun.class
Applet
DOCTOR FUN
1 Feb 2005
WE CALL HIM THE TALL COOL ONE.
Copyright © 2005 David Farley, d-farley@ibiblio.org
http://ibiblio.org/Dave/drfun.html
This cartoon is made available on the Internet for personal viewing only. Opinions expressed herein are solely those of the author.
Penguin researchers often use nicknames to help keep track of their subjects.
Applet gestartet

Das Applet ist jetzt gepuffert, hat aber schreckliche Nachteile:

- Bei jedem `Window`-Ereignis wird die Animation neu gestartet.
- eine laufende Animation lässt sich nicht mehr unterbrechen.

Das Applet ist jetzt gepuffert, hat aber schreckliche Nachteile:

- Bei jedem `Window`-Ereignis wird die Animation neu gestartet.
- eine laufende Animation lässt sich nicht mehr unterbrechen.

Plan 1:

1. Die Animation wird von der `start()`-Methode gestartet ...
2. ... und läuft in einem separaten Thread.
3. `paint()` wiederholt nur den aktuellen Puffer-Inhalt.
4. Um die Animation zu unterbrechen, verwalten wir eine separate Variable `boolean stopped`, die von `stop()` gesetzt wird.
5. Ist `stopped == true`, beendet sich die Animation.

```
import java.applet.Applet;
import java.awt.*;
class StopAux extends Thread {
   private Image buffer;
   private Graphics gBuff;
   private StopThread app;
   public StopAux(Image b, StopThread a) {
      buffer = b; app = a;
      gBuff = buffer.getGraphics();
      gBuff.setColor(Color.orange);
      gBuff.fillRect(0,0,500,300);
      app.repaint();
   }
   ...
```

```
...
public void run() {
   try {
      grow(50,50,Color.red);
      grow(100,100,Color.blue);
      grow(150,150,Color.green);
   } catch (InterruptedException e) { }
   gBuff.dispose();
}
...
```

- Die Animation wird von der Klasse `StopAux` realisiert.
- Einem neuen `StopAux`-Threads wird der Puffer und das Applet selbst übergeben.
- Die `run()`-Methode führt die Animation aus.

- Damit die Animation an jeder beliebigen Stelle unterbrochen werden kann, verlassen wir sie mithilfe des Werfens einer `InterruptedException`, die von der `run()`-Methode aufgefangen wird.

```
public void grow(int x, int y, Color color) throws
                                        InterruptedException {
   gBuff.setColor(color);
   for(int i=0; i<100; i++) {
      synchronized (app) {
         if (app.stopped)
            throw (new InterruptedException());
         gBuff.fillRect(x,y,2*i,i);
      }
      ...
```

```
            ... // continuing the for loop
            try {Thread.sleep(40);}
            catch (InterruptedException e) {
                 System.err.println("Growing interrupted!"); }
            app.repaint();
        }       // end of for loop
    }           // end of grow ()
}               // end of class StopAux()
...
```

- Das Malen des Bilds erfolgt durch Aufruf der Methode `public void repaint();` für das Applet.
- Diese Methode sorgt dafür, dass die Applet-Darstellung neu gemalt wird. Dazu wird (für das gegebene `Applet`-Objekt) `update(getGraphics());` aufgerufen.
- Die Methode `public void update(Graphics page);` füllt die Fläche des Applets mit der Hintergrund-Farbe. Dann wird mithilfe von `paint(page);` das Applet neu gemalt.

```
   ...
public class StopThread extends Applet {
   public boolean stopped;
   private Image buffer;
   public void init() { buffer = createImage(500,300); }
   ...
```

```
    ...
    public void start() {
       synchronized (this) { stopped = false;}
       (new StopAux(buffer, this)).start();
    }
    public void stop() {
       synchronized (this) { stopped = true;}
    }
    public void update(Graphics page) {
       paint(page);
    }
    public synchronized void paint(Graphics page) {
       page.setClip (0,0,500,300);
       page.drawImage(buffer,0,0,this);
    }
} // end of Applet StopThread
```

- Auch der Zugriff auf die Variable `stopped` ist (sicherheitshalber :-) synchronisiert.
- `stop()` setzt die Variable `stopped` auf `true`, `start()` setzt sie wieder zurück.
- Außerdem legt `start()` ein neues `StopAux`-Objekt für die Animation an und startet die Animation.
- Damit wir nicht vor einem grauen Bildschirm sitzen müssen, setzen sowohl die `start()`- wie `stop()`-Methode die Sichtbarkeit auf `true`.
- die `paint()`-Methode wiederholt offensichtlich (wie beabsichtigt) das letzte Bild im Puffer.
- Die Methode `update()` wurde überschrieben, da es offenbar überflüssig ist, zuerst den Hintergrund zu malen, um dann `paint()` aufzurufen ...

Frage:

- Was, wenn beim `stop()` die Animation nicht unterbrochen, sondern nur angehalten werden soll?
- Auch soll das `start()` nicht immer eine neue Animation starten, sondern eine eventuell bereits angelaufene, aber angehaltene fortsetzen!!!

Frage:

- Was, wenn beim `stop()` die Animation nicht unterbrochen, sondern nur angehalten werden soll?
- Auch soll das `start()` nicht immer eine neue Animation starten, sondern eine eventuell bereits angelaufene, aber angehaltene fortsetzen!!!

Plan 2:

1. Ist `stopped == true`, wird die Animation nicht beendet, sondern führt ein `wait()` aus !!!
2. Damit `start()` feststellen kann, ob bereits eine Animation läuft, führen wir eine zusätzliche Variable `boolean running` ein.
3. Ist `running == true`, schickt `start()` der Animation `notify()`.

```
import java.applet.Applet;
import java.awt.*;
class WaitAux extends Thread {
   private Graphics gBuff;
   private Image buffer;
   private WaitingThread app;
   public WaitAux(Image b, WaitingThread a) {...}
   public void run() {
      grow(50,50,Color.red);
      grow(100,100,Color.blue);
      grow(150,150,Color.green);
      synchronized (app) { app.running = false; }
      gBuff.dispose();
   }
   ...
```

- Der Konstruktor für die Animation hat sich nicht geändert.
- Die `run()`-Methode braucht dagegen keine Exception mehr zu fangen.
- Dafür setzt sie am Ende die Variable `running` des Applets auf `false`.

```
   ...
   public void grow(int x, int y, Color color) {
      gBuff.setColor(color);
      for(int i=0; i<100; i++) { try {
            synchronized (app) {
               if (app.stopped) app.wait();
               gBuff.fillRect(x,y,2*i,i);
            }
            Thread.sleep(40);
         } catch (InterruptedException e) {
            System.err.println("Growing interrupted ...");
         }
         app.repaint();
      }
   }
} // end of class WaitAux()
```

- Die Methode `grow()` testet wieder jeweils vor jedem neuen Bild, ob die Animation unterbrochen wurde.
- Wurde sie unterbrochen, führt sie ein `wait()` auf das Applet selbst aus.
- Aufgeweckt werden soll sie dabei von einem erneuten Aufruf der Methode `start()`.
- Diese schickt das `notify()` natürlich nur, sofern `running == true`. Andernfalls muss eine neue Animation gestartet werden:

```
public class WaitingThread extends Applet {
   public boolean running = false;
   public boolean stopped;
   private Image buffer;
   public void init() { buffer = createImage(500,300); }
   public synchronized void start() {
      stopped = false;
      if (!running) {
         running = true;
         (new WaitAux(buffer, this)).start();
      } else notify();
   }
... // wie gehabt
}   // end of class WaitingThread
```

Fazit:

- Es ist nicht völlig trivial, eine überzeugende und robuste Animation zu programmieren :-)
- Eine Animation sollte in einem separaten Thread laufen.
- Mit den `Applet`-Methoden `start()`, `stop()`, `paint()` und `destroy()` sollte der Animations-Thread kontrolliert und auf `Window`-Ereignisse reagiert werden.
- Die Verfügbarkeit von Programmier-Hilfsmitteln wie z.B. der Klasse `Stroke` hängt hängt stark von der verwendeten Java-Version ab ...