22.3 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 Bild ist nicht stabil !
- Offenbar ist der Bildaufbau der Animation ein längerer Vorgang ...

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 immer noch 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 immer noch 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()`.