# Grundlagen der Informatik für Ingenieure I

## 11. Threads, Images, Animation and Sound








## 11.1 Aktivitätsträger

- Grundsätzlich laufen in Rechensystemen zur Benutzerapplikation immer parallele Aktivitäten ab.
- Inwieweit diese Möglichkeiten aber auch Benutzerapplikationen verfügbar gemacht werden, hängt von den Betriebssystemen ab, die den Rechner steuern und dem Benutzer eine abstrakte Sicht auf das Rechensystem bereitstellen.
- Java stellt aber auch selbst einen “Parallelisierungsmechanismus” zur Verfügung, den man immer dann einführen sollte, wenn CPU-intensive Aktivitäten zur Bearbeitung anstehen, z. B. bei einer Animation.
- Der Browser sollte immer in der Lage sein Aktivitäten wahrzunehmen:
  - z. B. während wir mit dem Applet einen Ausgabestrom auslösen, sollte trotzdem der Scrollbar des Browserfensters veränderbar sein.
  - Wenn Browser und Applet aber denselben Aktivitätsträger (Thread) benutzen, kann der Browser erst dann wieder aktiv werden, wenn das Applet "fertig" ist.

# 11.1 Aktivitätsträger

- Grundsätzlich gibt es zwei Möglichkeiten *Threads* zu erzeugen:
  - Ableiten von der Klasse *Thread*
  - Implementieren des *Runnable-Interfaces*

- Für Applets müssen wir das *Runnable-Interface* verwenden:

```
public class MeinProblem extends java.applet.Applet
implements Runnable {
...
}
```

- Da wir einen “eigenen” Aktivitätsträger kontrollieren müssen, benötigen wir einen *“handle”* dies zu tun. Dafür vereinbaren wir eine Variable vom Typ *Thread*:

```
Thread animation;
```






# 11.1 Aktivitätsträger

- Wir verwenden in diesem Kapitel nur zwei Methoden zur Threadkontrolle:

  - Die *run()-method,* definiert im *Runnable-Interface*.
    Diese Methode enthält den Code, den der *Thread* abarbeiten soll:

```
public void run {
... Code des Threads ...
}
```

  - Die *start()-method* startet einen *Thread*:

```
public void start() {          // Start des Applets!
    if (animation == null) {
        animation = new Thread( this );
        animation.start();     // Start des Threads!
    }
}
```

# 11.1 Aktivitätsträger

◆ Ein vollständiges Beispiel: “Die Java-Digital-Clock”

```
import java.awt.Graphics;
import java.awt.Font;
import java.util.Calendar;
import java.util.GregorianCalendar;

public class DigitalClock extends java.applet.Applet
    implements Runnable {

    Font theFont = new Font( "TimesRoman", Font.BOLD, 24 );
    GregorianCalendar theDate;
    Thread runner;
    boolean running;

    public void start() {
        if ( runner == null ) {
            runner = new Thread( this );
            runner.start();
            running = true;
        }
    }
```





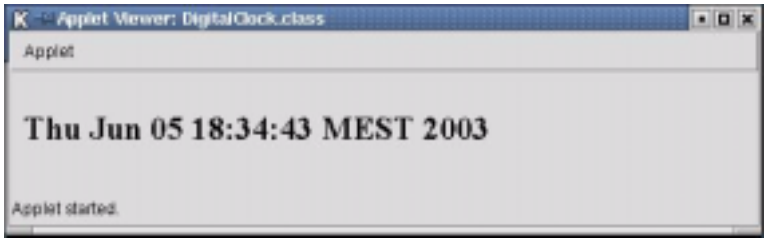

# 11.1 Aktivitätsträger

◆ Ein vollständiges Beispiel: “Die Java-Digital-Clock” (cont.)

```
    public void stop() {
        running = false;
    }

    public void run() {
        while ( running == true ) {
               repaint();
               try { Thread.sleep( 1000 ); }
               catch ( InterruptedException e ) { }
        }
    }

    public void paint( Graphics g ) {
            theDate = new GregorianCalendar();
            g.setFont( theFont );
            g.drawString( "" + theDate.getTime(), 10, 50 );
    }
}
```

# 11.1 Aktivitätsträger

- Ergebnis mit Appletviewer:









# 11.2 MIME - Formate und Datei-*Extensions*

- HTML ist **nicht** das einzige Format, das im Internet übertragen wird.
- Damit Browser und andere Applikationen die verschiedenen Dokumenttypen erkennen und verarbeiten können, wurden sie standardisiert und klassifiziert.
- x- heißt: noch nicht standardisiert.
- MIME steht für: *Multipurpose Internet Mail Extension*

# 11.2 MIME - Formate und Datei-*Extensions*

◆ Hier eine Auswahl:

| Dokumententyp | Format | Date-Extensions | MIME-Typ |
|---|---|---|---|
| Text | HTML | htm; html | text/html |
| Text | *"plain"* text | txt | text/plain |
| Anwendung | PostScript | ps; PS; ai; eps | application/postscript |
| Gemischtes Dokument | mehrere | offen | multipart/x-mixed-replace |
| Bild | GIF | gif | image/gif |
| Bild | JPEG | jpg; jpeg | image/jpeg |
| Ton | AU | au | audio/basic |
| Ton | WAV | wav | audio/x-wav |
| Ton | mp3 | mp3 | ??? |
| Video | MPEG | mpeg; mpg | video/mpeg |
| Virtuelle Welt | VRML | wrl | x-world/x-vrml (model/vrml) |






# 11.3 Images

◆ Von *Images* spricht man, wenn man Bilder, Graphiken oder Texte punkt- (bzw. pixel-) weise darstellt.

◆ Diese Darstellungsart ist sehr speicherintensiv.

◆ Ein Standard Videobild z. B. mit 625 x 800 Punkten und 24 Bit Farbtiefe benötigt 1.5 MByte Speicherplatz.

◆ Aus diesem Grund werden diese Bilder in der Regel komprimiert abgelegt.

◆ Das bedeutet, daß das *Image* - Laden mit einer Datendekompression einhergehen muß.

◆ Laden und Dekomprimieren von *Images* ist zeitintensiv.

◆ Images liegen in separaten Dateien und müssen aus diesen heraus geladen werden.

# 11.3 Images

- Die Klasse *Applet* stellt dafür die ***getImage()-method*** bereit, die das *Image* lädt und zur Weiterverarbeitung ein *Image*-Objekt zur Verfügung stellt.

- Wie wir schon im Umgang mit HTML- und Applet-Dateien gelernt haben, gibt es verschiedene Methoden separat angelegte Dateien im Java-Umfeld anzusprechen:

  - Die *Image*-Datei (z.B. bild.gif) ist unter einer festen URL zu finden:

```
Image img = getImage(
new URL( “http://www.cip/mybase/images/bild.gif” ) );
```

  - Die *Image*-Datei liegt im gleichen *Directory* wie die HTML-Datei, die die Referenz auf das Java-Applet enthält:

```
Image img = getImage( getDocumentBase(), “bild.gif” );
```






# 11.3 Images

  - Die *Image*-Datei liegt in der gleichen Directory wie das Java-Applet:

```
Image img = getImage( getCodeBase(), “bild.gif” );
```

  - Man kann natürlich auch relative Pfadnamen angeben, wie:

```
Image img = getImage( getCodeBase(),
                      “gif_images/bild.gif” );
```

- *Image*-Formate:

  Java unterstützt zur Zeit folgende *Image*-Formate:

  - *GIF - Graphical Interchange Format*
  - *JPEG - Joint Photographer Expert Group*

# 1 Mit *Images* arbeiten

- Wie bereits erwähnt, ist das Laden und Dekomprimieren ein aufwendiger Vorgang, so daß für diese Aktivität ein eigener *Thread* kreiert wird.
- Die Kontrolle geht “sofort” an den Aufrufer zurück, der weitere “Aktivitäten” ausführen kann.
- Die Parallelarbeit im Hintergrund wird vom *ImageObserver* überwacht, der z. B. entscheidet, wann ein *Image* wirklich auf den *Screen* ausgegeben wird.
- Für die Ausgabe eines *Images* gibt es zwei Methoden ***drawImage()*:**






# 1 Mit *Images* arbeiten

- Zwei *drawImage()* - Methoden ( Die Angabe des Parameter ‘this’ bedeutet, man verwendet den default ImageObserver):
  - In der einen Methode wird das *Image* in Originalform relativ zur Koordinate (0,0) plaziert:

```
public void paint() {
    g.drawImage( img, 10, 10, this );
}
```

  - In der zweiten Methode wird die Größe der *Image-Bounding-Box* zusätzlich explizit mit zwei Parametern festgelegt:

```
public void paint() {
    g.drawImage( img, 10, 10, breite, hoehe, this );
}
```

  - mit folgenden Methoden erhält man die Orginalgrößen:

```
breite = img.getWidth( this );
hoehe = img.getHeight( this );
```

# 2 Ein vollständiges *Image*-Beispiel

◆ Bilddatei ***ladybug.gif*** (Käfer) ausdrucken:

- In Normalgröße
- verkleinert
- verzerrt






# 2 Ein vollständiges *Image*-Beispiel

```
import java.awt.Graphics;
import java.awt.Image;

public class LadyBug extends java.applet.Applet {

    Image bugimg;

    public void init() {
        bugimg = getImage( getCodeBase(),
            "images/ladybug.gif" );
    }

    public void paint( Graphics g ) {
        int iwidth = bugimg.getWidth( this );
        int iheight = bugimg.getHeight( this );
        int xpos = 10;
```

## 2 Ein vollständiges *Image*-Beispiel

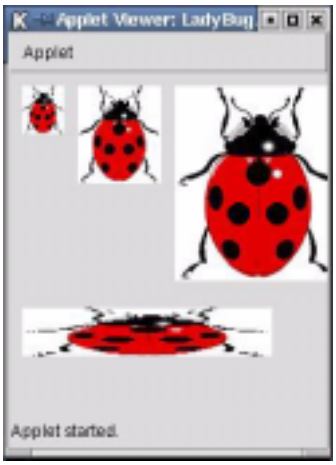
```
            // 25 %
            g.drawImage( bugimg, xpos, 10,
                iwidth / 4, iheight / 4, this );

            // 50 %
            xpos += ( iwidth / 4 ) + 10;
            g.drawImage( bugimg, xpos , 10,
                iwidth / 2, iheight / 2, this );

            // 100%
            xpos += ( iwidth / 2 ) + 10;
            g.drawImage( bugimg, xpos, 10, this );

            // 150% x, 25% y
            g.drawImage( bugimg, 10, iheight + 30,
                (int)( iwidth * 1.5), iheight / 4, this );
        }
    }
```






## 2 Ein vollständiges *Image*-Beispiel

- Ergebnis mit Appletviewer:



# 11.4 Animation

- Animation - bewegte Bilder - sind letztendlich optische Täuschungen.
- Wie bei einem herkömmlichen Film, der uns in einer schnellen Folge leicht unterschiedliche Einzelbilder projeziert, wird auch in einem Rechner verfahren.
- Im Java-Umfeld bedeutet das, einzelne - leicht unterschiedliche - *Graphics-Objekte* mit der *draw()-method* auf den Screen auszugeben.






## 1 Animation mit *Images*

- Beispiel Neko-Applet:
  - Neko ist eine kleine Animation auf dem Macintosh von Kenji Gotoh (1989):
    - Eine Katze (japanisch “Neko”) versucht den *Mouse*-Zeiger zu fangen.
    - Neun dieser *Images* stehen uns zur Verfügung:

# 1 Animation mit *Images*

◆ Wir organisieren die *Images* als *array_of Images.*

◆ In der *init()-method* laden wir diese *Images.*

◆ Unsere Animation:

- Neko läuft von links in den Screen,
- hält in der Mitte an und gähnt,
- kratzt sich 4 mal,
- schläft,
- wacht auf und läuft nach rechts aus dem Screen.






# 1 Animation mit *Images*

◆ Erläutert direkt am Beispiel:

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

public class Neko extends java.applet.Applet
    implements Runnable {

    Image nekopics[] = new Image[9];
    Image currentimg;
    Thread runner;
    int xpos;
    int ypos = 50;

    public void init() {

        String nekosrc[] = { "right1.gif", "right2.gif",
            "stop.gif", "yawn.gif", "scratch1.gif",
            "scratch2.gif","sleep1.gif", "sleep2.gif",
            "awake.gif" };

        for ( int i=0; i < nekopics.length; i++ ) {
            nekopics[i] = getImage( getCodeBase(),
            "images/" + nekosrc[i] );
        }
    }
```

# 1 Animation mit *Images*

```
public void start() {
    if ( runner == null ) {
        runner = new Thread( this );
        runner.start()
    }
}

public void run() {
    setBackground( Color.white );
// run from one side of the screen to the middle
    nekorun( 0, size().width / 2 );
// stop and pause
    currentimg = nekopics[2];
    repaint();
    pause(1000);
// yawn
    currentimg = nekopics[3];
    repaint();
    pause( 1000 );
// scratch four times
    nekoscratch(4);
```






# 1 Animation mit *Images*

```
    // sleep for 5 "turns"
       nekosleep(5);
    // wake up and run off
       currentimg = nekopics[8];
       repaint();
       pause( 500 );
       nekorun( xpos, size().width + 10 );
    }
    void nekorun( int start, int end ) {
        for ( int i = start; i < end; i += 10 ) {
            xpos = i;
            // swap images
            if ( currentimg == nekopics[0] )
                currentimg = nekopics[1];
            else currentimg = nekopics[0];
            repaint();
            pause( 150 );
        }
    }
```

# 1 Animation mit *Images*

```
    void nekoscratch( int numtimes ) {
        for ( int i = numtimes; i > 0; i-- ) {
            currentimg = nekopics[4];
            repaint();
            pause( 150 );
            currentimg = nekopics[5];
            repaint();
            pause( 150 );
        }
    }
    void nekosleep( int numtimes ) {
        for ( int i = numtimes; i > 0; i-- ) {
            currentimg = nekopics[6];
            repaint();
            pause( 250 );
            currentimg = nekopics[7];
            repaint();
            pause( 250 );
        }
    }
```



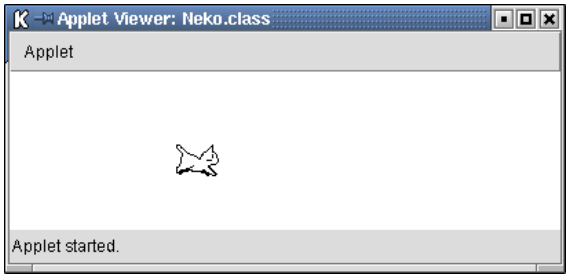


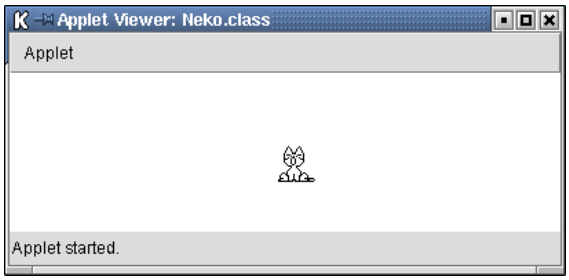

# 1 Animation mit *Images*

```
    void pause( int time ) {
        try { Thread.sleep( time ); }
        catch ( InterruptedException e ) { }
    }

    public void paint( Graphics g ) {
        if ( currentimg != null )
            g.drawImage( currentimg, xpos, ypos, this );
    }
}
```

# 1 Animation mit *Images*

- Ergebnis mit Appletviewer:



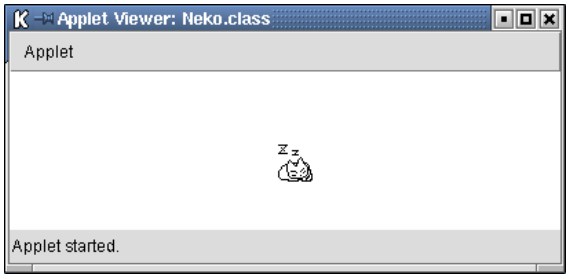



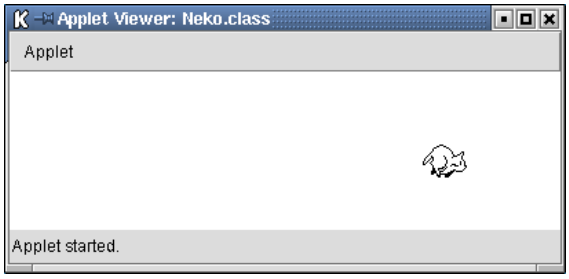






# 1 Animation mit *Images*



# 11.5 Sound

- Analog zu *Images* kann man auch Audio-Dateien laden und ausgeben, natürlich nicht auf den *Screen*, aber auf das “Audio-System”, sofern am Rechner eines vorhanden ist.
- Java unterstützt zur Zeit nur das *AU*-Format.
- Analog zu den entsprechenden *Images*-Methoden existieren auch Audio-Methoden:
  - Laden eines Audioclips

```
AudioClip myClip =
getAudioClip( getCodebase(),”miau.au” );
```

  - Abspielen; Immer wieder abspielen; Stoppen eines Clips

```
myClip.play(); myClip.loop(); myClip.stop()
```

  - Laden und Abspielen:

```
play( getCodeBase(), “miau.au” );
```






# 11.5 Sound

- Ein Beispiel - Audio-Thread läuft im Hintergrund!

```
import java.awt.Graphics;
import java.applet.AudioClip;

public class AudioLoop extends java.applet.Applet
    implements Runnable {

    AudioClip bgsound;
    AudioClip beep;
    Thread runner;

    public void start() {
         if ( runner == null ) {
             runner = new Thread( this );
             runner.start();
         }
    }
    public void stop() {
        if ( runner != null ) {
            if ( bgsound != null ) bgsound.stop();
            runner = null;
        }
    }
```

# 11.5 Sound

```
    public void init() {
        bgsound = getAudioClip( getCodeBase(), "audio/loop.au" );
        beep = getAudioClip( getCodeBase(), "audio/beep.au" );
    }

    public void run() {
        if ( bgsound != null ) bgsound.loop();
        while ( runner != null ) {
            try { Thread.sleep( 5000 ); }
            catch ( InterruptedException e ) { }
            if ( beep != null ) beep.play();
        }
    }

    public void paint( Graphics g ) {
        g.drawString( "Playing Sounds....", 10, 10 );
    }
}
```






# 11.5 Sound

- Ergebnis mit Appletviewer: