In diesem Kapitel:

- *Daten herunter laden mit Hilfe der Applet Klasse*
- *Das Image Observer Interface*
- *Die MediaTracker Klasse*
- *Die Netzwerk Methoden des AppletContext Interface*
- *Beispielprogramme*

6

Die Netzwerk Methoden von java.applet.Applet

6.1. Herunterladen von Daten mit Hilfe der Applet Klasse

Wir haben uns bisher nicht wesentlich mit Applets befasst. Das ist auch nicht ein echtes Defizit, da Applets im Vergleich zu anderen Web Techniken eher mühsam sind, sprich langsam.

6.2. Übersicht über die java.applet.Applet Klasse

Die Beschreibung der Klasse Applet aus dem Paket java.applet sieht recht umfangreich aus. Wir werden aber nur einige speziell interessanten Methoden näher betrachten.

6.2.1. java.applet
Class Applet

```
java.lang.Object
  |
  +--java.awt.Component
        |
        +--java.awt.Container
              |
              +--java.awt.Panel
                    |
                    +--java.applet.Applet
```

Direkte Subklassen:

JApplet

public class **Applet**
extends Panel
Ein Applet ist ein kleines Programm, welches nicht selbständig lauffähig ist, sondern nur eingebettet in einer andern Applikation lauffähig ist. Die Applet Klasse muss Superklasse aller Applets sein, die in Web Seiten eingebettet sind oder mit dem AppletViewer betrachtet werden sollen. Die Applet Klasse definiert eine Standard Schnittstelle zwischen den Applets und deren Umgebung.

Seit:

JDK1.0

Siehe auch: Serialized Form

6.2.1.1. Von class java.awt.Component geerbte Datenfelder

BOTTOM_ALIGNMENT, CENTER_ALIGNMENT, LEFT_ALIGNMENT, RIGHT_ALIGNMENT, TOP_ALIGNMENT

6.2.1.2. Konstruktor Kurzfassung

Applet()	

6.2.1.3. Methoden Kurzfassung

void	**destroy**() Diese Methode wird vom Browser oder AppletViewer aufgerufen um dem Applet mit zu teilen, dass es beendet wird und dass es alle Resourcen, die es gebunden hat, wieder frei geben muss.
AppletContext	**getAppletContext**() Bestimmt die Applet Umgebung und erlaubt es dem Applet, die Umgebung abzufragen und zu beeinflussen.
String	**getAppletInfo**() Liefert Informationen über das Applet.
AudioClip	**getAudioClip**(URL url) Liefert das Audio Clip Objekt, welches durch die URL (als Parameter) spezifiziert wird.
AudioClip	**getAudioClip**(URL url, String name) Liefert das Audio Clip Objekt, welches durch die URL (Parameter) und den Namen (zweites Argument) spezifiziert wird.
URL	**getCodeBase**() Liefert die Basis URL.
URL	**getDocumentBase**() Liefert die Dokumenten URL.
Image	**getImage**(URL url) Liefert ein Image Objekt, welches am dem Bildschirm angezeigt werden kann.
Image	**getImage**(URL url, String name) Liefert ein Image Objekt, welches am dem Bildschirm angezeigt werden kann.
Locale	**getLocale**() Liefert Locale für das Applet, falls dieses gesetzt wurde..
String	**getParameter**(String name) Liefert den Wert eines benannten Parameters in einem HTML Tag.
String[][]	**getParameterInfo**() Liefert Informationen über die Parameter, die vom Applet verstanden werden.
void	**init**() Wird vom Browser oder dem AppletViewer aufgerufen, um dem Applet mit zu teilen, dass es ins System geladen wurde.
boolean	**isActive**() Bestimmt, ob dieses Applet aktiv ist oder nicht.
static AudioClip	**newAudioClip**(URL url) Lies ein Audio Clip aus einer gegebenen URL.
void	**play**(URL url)

	Spielt das Audio Clip, welches sich an der spezifizierten absoluten URL befindet
void	**play**(URL url, String name) Spielt das Audio Clip, welches sich an der angegebenen URL und einer relativen Adresse (String) von dieser URL befindet.
void	**resize**(Dimension d) Verlangt, dass dieses Applet seine Grösse verändert.
void	**resize**(int width, int height) Verlangt, dass as Applet seine Grösse gemäss den Parametern verändert.
void	**setStub**(AppletStub stub) Setzt den AppletStub dieses Applets.
void	**showStatus**(String msg) Verlangt, dass die Zeichenkette, welche als Parameter übergeben wird, im Status Fenster angezeigt wird..
void	**start**() Wird vom Browser oder AppletViewer aufgerufen, um dem Applet mit zu teilen, dass es mit seiner Ausführung starten soll.
void	**stop**() Wird vom Browser oder AppletViewer aufgerufen, um dem Applet mit zu teilen, dass es seine Ausführung beenden soll.

6.2.1.4. Von der Klasse java.awt.Panel geerbte Methoden

addNotify

6.2.1.5. Von der Klasse java.awt.Container geerbte Methoden

add, add, add, add, add, addContainerListener, addImpl, countComponents, deliverEvent, doLayout, findComponentAt, findComponentAt, getAlignmentX, getAlignmentY, getComponent, getComponentAt, getComponentAt, getComponentCount, getComponents, getInsets, getLayout, getMaximumSize, getMinimumSize, getPreferredSize, insets, invalidate, isAncestorOf, layout, list, list, locate, minimumSize, paint, paintComponents, paramString, preferredSize, print, printComponents, processContainerEvent, processEvent, remove, remove, removeAll, removeContainerListener, removeNotify, setFont, setLayout, update, validate, validateTree

6.2.1.6. Von der Klasse class java.awt.Component geerbte Methoden

action, add, addComponentListener, addFocusListener, addInputMethodListener, addKeyListener, addMouseListener, addMouseMotionListener, addPropertyChangeListener, addPropertyChangeListener, bounds, checkImage, checkImage, coalesceEvents, contains, contains, createImage, createImage, disable, disableEvents, dispatchEvent, enable, enable, enableEvents, enableInputMethods, firePropertyChange, getBackground, getBounds, getBounds, getColorModel, getComponentOrientation, getCursor, getDropTarget, getFont, getFontMetrics, getForeground, getGraphics, getHeight, getInputContext, getInputMethodRequests, getLocation, getLocation, getLocationOnScreen, getName, getParent, getPeer, getSize, getSize, getToolkit, getTreeLock, getWidth, getX, getY, gotFocus, handleEvent, hasFocus, hide, imageUpdate, inside, isDisplayable, isDoubleBuffered, isEnabled, isFocusTraversable, isLightweight, isOpaque, isShowing, isValid, isVisible, keyDown, keyUp, list, list, list, location, lostFocus, mouseDown, mouseDrag, mouseEnter, mouseExit, mouseMove, mouseUp, move, nextFocus, paintAll, postEvent, prepareImage, prepareImage, printAll, processComponentEvent, processFocusEvent, processInputMethodEvent, processKeyEvent, processMouseEvent, processMouseMotionEvent, remove, removeComponentListener, removeFocusListener, removeInputMethodListener, removeKeyListener, removeMouseListener, removeMouseMotionListener, removePropertyChangeListener, removePropertyChangeListener, repaint, repaint, repaint, repaint, requestFocus, reshape, setBackground, setBounds, setBounds, setComponentOrientation, setCursor, setDropTarget, setEnabled, setForeground, setLocale, setLocation, setLocation, setName, setSize, setSize, setVisible, show, show, size, toString, transferFocus

6.2.1.7. Von der Klasse java.lang.Object geerbte Methoden

clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait

6.2.2. Konstruktor Detail

Applet

```
public Applet()
```

6.2.3. Methoden Detail

setStub

```
public final void setStub(AppletStub stub)
```

Setzt den Stub des Applets. Dies wird automatisch durch das System erledigt.

Parameter:

`stub` - der neue Stub.

---

isActive

`public boolean` **`isActive`**`()`
Bestimmt, ob dieses Applet aktiv ist. Ein Applet wird als aktiv gekennzeichnet, direkt bevor die start Methode ausgeführt wird. Es wird inaktiv, kurz die stop Methode aufgerufen wird.
Returns:
`true` , falls das Applet aktiv ist; `false` sonst.
Siehe auch:
`start()`, `stop()`

---

getDocumentBase

`public URL` **`getDocumentBase`**`()`
Liest das Argument von der URL. Dies ist die URL des Dokumentes, in dem sich das Applet befindet.
Returns:
die `URL#_top_` des Dokuments, welche das Applet enthält.
Siehe auch:
`getCodeBase()`

---

getCodeBase

`public URL` **`getCodeBase`**`()`
Liest die URL. Dies ist die URL des Applets.
Returns:
die `URL#_top_` des Applets.
Siehe auch:
`getDocumentBase()`

---

getParameter

`public String` **`getParameter`**`(String name)`
Liefert den Wert des benannten Parameters im HTML Tag. Zum Beispiel :

```
<applet code="Clock" width=50 height=50>
<param name=Color value="blue">
</applet>
```

`getParameter("Color")` liefert den Wert `"blue"`.
Das Argument ist nicht Gross / Kleinschreibung sensitiv.
Parameters:
`name` -ein Parameter Name.
Returns:
den Wert des benannten Parameters.

---

getAppletContext

```
public AppletContext getAppletContext()
```
Bestimmt den Context dieses Applets, der vom Applet abgefragt und beinflusst werden kann..
Diese Umgebung des Applets beschreibt das Dokument, welches das Applet enthält.
Returns:
Applet Context.

---

resize

```
public void resize(int width,
          int height)
```
Verändert die Grösse des Applets.
Parameters:
`width` - die neue vom Applet verlangte Breite.
`height` - die neue vom Applet verlangte Höhe.
Überschreibt:
resize in class Component

---

resize

```
public void resize(Dimension d)
```
Grösse des Applets verändern.
Parameters:
`d` - ein Objekt mit der neuen Breite und Höhe.
Überschreibt:
resize in class Component

---

showStatus

```
public void showStatus(String msg)
```
Verlangt, dass die Zeichenkette im "Status Window" angezeigt wird. Viele Browser und Applet Viewer stellen ein solches Fenster zur Verfügung In diesem Fenster kann die Applikation den Benutzer informieren.
Parameters:
`msg` - eine Zeichenkette, die im Statusfenster dargestellt werden soll..

---

getImage

```
public Image getImage(URL url)
```
Liefert ein `Image` Object, welches auf dem Bildschirm dargestellt werden kann. Der URL, der als Argument übergeben wird, muss eine absolute Adresse sein.
Diese Methode endet, unabhängig davon, ob das Bild existiert oder nicht. Falls das Applet versucht ein Bild auf dem Bildschirm darzustellen, dann werden die Daten geladen Die

Grafikbausteine aus denen das Bild besteht, werden inkrementell auf dem Bildschirm dargestellt.
Parameters
`url` - eine absolute URL , die Lokation des Bildes.
Returns:
das Bild von der URL.
Siehe auch:
`Image`

---

getImag

```
public Image getImage(URL url,
 String name)
```
Liefert ein Bildobjekt , welches dann auf dem Bildschirm dargestellt werden kann. Die URL Adresse muss eine absolute URL sein. `Das Namensargument ist eine Spezifikation relativ zur URL`. Die Methode schliesst immer ab, unabhängig davon, ob das Bild existiert oder nicht. Falls das Applet das Bild zeichnen will, dann werden die Daten gelesen. Das Bild wird schrittweise aufgebaut.
Parameters:
`url` - eine absolute URL , die Basis des Bildes.
`name` - die Adresse des Bildes, relativ zum URL Argument.
Returns:
das Bild von der URL.
Siehe auch:
`Image`

---

newAudioClip

```
public static final AudioClip newAudioClip(URL url)
```
Liest ein Audio Clip vom URL
Parameters:
`url` - zeigt auf das Audio Clip
Seit:
JDK1.2

---

getAudioClip

```
public AudioClip getAudioClip(URL url)
```
Liefert das `AudioClip` Object ,welches sich bei der Adresse URL befindet.
Diese Methode schliesst immer ab, auch falls das Audio Clip nicht gefunden wird. Die Daten des Audio Clips werden geladen, falls das Applet versucht das Clip abzuspielen.
Parameters:
`url` - die absolute URL des Audio Clips.
Returns:
das Audio Clip von der absoluten URL.
Siehe auch:

AudioClip

---

getAudioClip

```
public AudioClip getAudioClip(URL url,
String name)
```
Liefert das Audio Clip, welches sich an der Adresse URL plus name befindet.
Die Methode schliesst immer ab, auch wenn das AudioClip nicht gefunden wurde. Falls das Applet die Daten darstellen möchte, werden diese geladen.
Parameters:
`url` - die URL von der Basisadresse des Audio Clips.
`name` - die Lokation des Audio Clips, relativ zur absoluten URL.
Returns:
das Audio Clip an der spezifizierten URL.
Siehe auch:
AudioClip

---

getAppletInfo

```
public String getAppletInfo()
```
Liefert Informationen über das Applet. Ein Applet sollte diese Methode überschreiben, um eine Zeichenkette zu liefern, die Informationen über den Autor, die Version, das Copyright des Applets enthält.
Die Implementation dieser Methode durch die Applet Klasse liefert null als Rückgabewert..
Returns:
eine Zeichenkette, die Informationen über den Autor, die Version und das Copyright des Applets enthält.

---

getLocale

```
public Locale getLocale()
```
Liefert die Locale des Applets, falls dies gesetzt wurde.
Returns:
Locale des Applets
Overrides:
getLocale in class Component
Since:
JDK1.1

---

getParameterInfo

```
public String[][] getParameterInfo()
```
Liefert Informationen über die Parameter, die interpretiert werden können. Ein Applet sollte jeweils diese Methode überschreiben, damit ein array von Zeichenketten zurück gegeben wird, die diese Parameter beschreiben.

Jedes Element des array sollte aus drei `Strings` bestehen, die Name, Type, und eine Beschreibung enthält.

Zum Beispiel:

```
String pinfo[][] = {
{"fps",    "1-10",    "frames per second"},
{"repeat", "boolean", "repeat image loop"},
{"imgs",   "url",     "images directory"}
};
```

Die Standardimplementation dieser Methode in der Applet Klasse liefert null als Wert.
Returns:
ein array , welches die Parameters dieses Applets benötigt.

---

play

```
public void play(URL url)
```
Spielt das Audio Clip, welches sich bei der absoluten Adresses URL befindet. Falls das Audio Clip nicht gelesen werden kann, geschiet nichts.
Parameters:
`url` - absolute URL , die Adresse des Audio Clips.

---

play

```
public void play(URL url,
                 String name)
```
spielt das Audio Clip, welches sich an der absoluten URL und der Spezifikation befindet,. Falls das Audio Clip nicht gefunden wird, geschieht nichts.
Parameters:
`url` - eine absolute URL , die Lokation des Audio Clips.
`name` - die Lokation des Audio Clips, relative zum URL Argument.

---

init

`public void init()`
wird durch den Browser oder den Applet Viewer gestartet und informiert das Applet darüber, dass es in das System geladen wurde. Diese Methode wird immer bevor die start() Methode das erste Mal gestartet wird..
Eine Subklasse von `Applet` sollte diese Methode immer dann überschreiben, wenn das Applet spezielle Initialisierungen benötigt.
Zum Beispiel : falls eine Methode Threads einsetzt, dann sollte das Applet die init() Methode verwenden, um die Threads zu kreieren und zu zerstören.
Die Implementation der init() Methode, so wie sie in der Applet Klasse definiert ist, bewirkt nichts, sie ist leer, ein Platzhalter.
Siehe auch:
`destroy()`, `start()`, `stop()`

---

start

`public void start()`
Wird durch den Browser oder den Applet Viewer aufgerufen und informiert das Applet, dass es mit seiner Ausführung starten soll. Die Methode wird aufgerufen, nachdem die init() Methode aufgerufen wurde und jedesmal beim Besuch der Webseite, die das Applet enthält.
Eine Subklasse der Applet Klasse sollte diese Methode überschreiben, falls das Applet bestimmte Aktivitäten ausführen soll, sofern die Webseite besucht wird.
Beispiel :
Falls das Applet eine Animation durchführt, dann macht es Sinn, dass mit Hilfe der start() Methode die Animation jeweils weitergeführt wird und mit Hilfe der stop() Methode unterbrochen wird..
Die Standard Implementation dieser Methode führt keine Anweisung aus.
Siehe auch:
`destroy()`, `init()`, `stop()`

---

stop

`public void stop()`
Diese Methode wird vom Browser oder dem Appletviewer aufgerufen. Sie informiert das Applet, dass dieses seine Ausführung anhalten soll. Diese Methode wird aufgerufen, falls die Webseite, welche das Applet enthält, durch eine andere Seite ersetzt wird und bevor das Applet zerstört, also aus dem Speicher entfernt wird..
Eine Unterklasse der `Applet` Klasse sollte diese Methode überschreiben, falls das Applet jeweils beim Verlassen der Webseite bestimmte Operationen durch zu führen hat..
Beispiel : falls ein Applet eine Animation durch führen möchte, dann kann es mit start() die Animation aktivieren, und mit stop() die Animation unterbrechen..
Die Standard Implementation dieser Methode führt keine Anweisung aus.
Siehe auch:
`destroy()`, `init()`

destroy

`public void` **`destroy`**`()`
Diese Methode wird vom Browser oder dem Appletviewer aufgerufen. Sie informiert das Applet darüber, dass es vernichtet wird und dass es alle Resourcen, die es im Speicher besetzt hat, freigeben muss. Die `stop` Methode wird immer aufgerufen, bvor die Methode `destroy` `aufgerufen wird`.
Eine Unterklasse der `Applet` Klasse sollte diese Methode überschreiben, falls das Applet bestimmte Operationen durchführen, bevor es aus dem System entfernt wird.
Beispiel: ein Applet, welches mit Threads arbeitet, verwendet die `init` Methode um Threads zu starten und die `destroy` Methode, um sie zu zerstören.
Die Standard Implementation dieser Methode führt keine Anweisung aus.
Siehe auch:
`init()`, `start()`, `stop()`

6.3. Einsatz der Applet Klassen zum Herunterladen von Daten

Applets sind Java Programme, die im Gegensatz zu einem klassischen Java Programm mit der init Methode gestartet werden, nicht mit einer public static void main(String args[]).

Als erstes bestimmen wir generelle Informationen des Applets.

6.3.1. Bestimmen von Informationen über das Applet

Da wir Daten über das Web zum Client herunter laden möchten, dies aber nur ab dem Host möglich ist, von dem das Applet stammt, müssen wir dieses Verzeichnis bestimmen, die *Codebase*.
Die Appletklasse verfügt über mehrere Methoden, die bei der Bestimmung des Ursprung Verzeichnisses eingesetzt werden können.

6.3.1.1. public URL getDocumentBase()

Diese Methode liefert, wie Sie aus der JavaDoc Dokumentation oben erkennen können, die URL der Seite, die das Applet enthält.

Ein Applet Programm, welches diese Methode einsetzen möchte, enthält etwa folgenden Programmcode:

```
public void paint(Graphics g) {
        g.drawString(getDocumentBase().toString(), 25, 25);
}
```

Wir sehen gleich ein vollständigeres Beispiel!

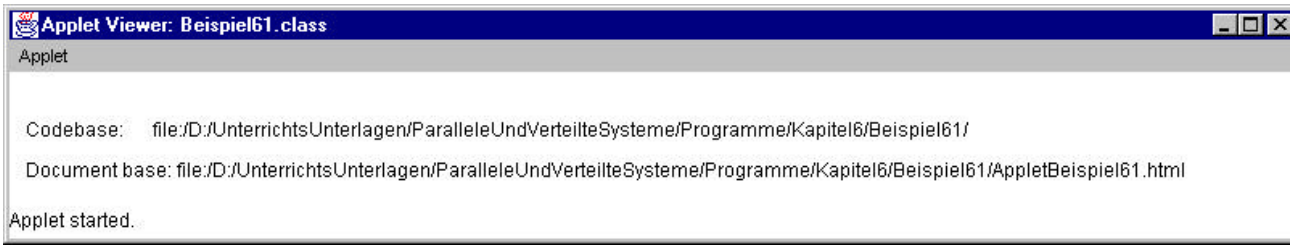
6.3.1.2. public URL getCodeBase()

Die getCodeBase() Methode liefert die URL, von der das Applet stammt. Das folgende Beispiel verdeutlicht dies:

```
//Titel:       getDocumentBase() getCodeBase()
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: einfaches Applet , welches die Verzeichnisse
//Codebase und Documentbase bestimmt und
//auf dem Bildschirm anzeigt

// kein Package !

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

public class Beispiel61 extends Applet {


  public void paint(Graphics g) {

    g.drawString("Codebase:      " + getCodeBase().toString(), 10, 40);
    g.drawString("Document base: " + getDocumentBase().toString(), 10, 65);

  }

}
```

Und hier der Vollständigkeit halber noch der HTML Tag

```
<applet code=Beispiel61.class width=800 height=80>
</applet><P>

<A HREF="Beispiel61.java">Applet Java Quellcode</A><P>
```



6.3.2. Herunterladen von Bildern

Im letzten Kapitel haben wir kennen gelernt, wie man Daten von einem URL lesen kann. Die Methoden, die dafür eingesetzt werden sind

- getContent() und
- getInputStream()

Diese Methoden sind jedoch nur geeignet für Textdaten, die zeilenweise gelesen werden können. Für binäre Daten, Bilder zum Beispiel, benötigen wir andere Methoden.

6.3.2.1. public Image getImage(URL u)

Mit Hilfe dieser Methode können Bilder aus einem spezifizierten URL gelesen werden und einem Bild Objekt zugewiesen werden.

```
Image meinPhoto = getImage(new
URL(http://www.switzerland.org/Joller/assets/images/its_me.jpg) );
```

Die getImage() Methode hängt vom AppletContext ab. Dieser wird vom Web Browser oder dem Appletviewer bestimmt. Die Methode kann also nur Bilder in Formaten herunter laden, die vom AppletContext verstanden werden.
Das Standard Format für Bilder war bisher GIF oder JPEG; aber die Zahl der interpretierten Formate wächst ständig.

```
//Titel:       public Image getImage(URL u)
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: Herunterladen eines Bildes von einem URL

import java.awt.*;
import java.net.*;
import java.applet.*;

public class Beispiel62 extends Applet {
  Image imgBild;

  //Das Applet initialisieren
  public void init() {
    try  {
      URL u = new URL(getCodeBase(), getParameter("AppletBild") );
      imgBild = getImage(u);
    }
    catch(MalformedURLException e)  {
      e.printStackTrace();
      System.err.println(e);
```

```
   } }

  // Anzeige
  public void paint (Graphics g)  {
    g.drawImage(imgBild, 0, 0, this);
  }
}
```

mit dem HTML Code:

```
<applet code=Beispiel62 width=166 height=188>
<param name=AppletBild value=Joller.gif>
</applet>
```

Wichtige Punkte, die beachtet werden müssen:
- am Anfang des Java Codes steht *kein* Package
- der Name des Parameters (oben "param name=...") muss mit dem Parameter Namen im Applet übereinstimmen.
- die Dimensionen müssen so gewählt werden, dass das Bild als Ganzes dargestellt werden kann.

Die Methode getImage() gibt es in mehreren Varianten. Neben der eben eingesetzten Methode kann man auch die URL als Basisadresse, plus einem Dateinamen für das Bild als Parameter angeben.

6.3.2.2. public Image getImage(URL path, String filename)

Die Methode wird in der Regel eingesetzt, falls bereits CodeBase() und DocumentBase() bestimmt wurden.
Die Grundstruktur des Programmes sieht gleich aus, wie das vorherige Beispiel. Lediglich der Methodenaufruf ist unterschiedlich.

```
Image imgBild = getImage(new URL(http://www.switzerland.org/Joller/), "PokalAnimated.gif");
```

Falls wir das Applet universell einsetzbar, also Webserver unabhängig auslegen wollen, dann müssen wir die absolute URL (www.switzerland.org) durch ein anderes Konstrukt ersetzen. Eine gängige Variante ist, die absolute Adresse durch die CodeBase Adresse zu ersetzen:

```
Image imgBild = getImage(getCodeBase(), "PokalAnimated.gif");
```

Falls das Applet zwar nur auf einem Webserver, aber in unterschiedlichen Seiten eingesetzt wird, kann man an Stelle der CodeBasis die Dokumentbasis einsetzen:

```
Image imgBild = getImage(getDocumentBase(), "../assets/images/PokalAnimated.gif") );
```

Das (animierte) GIF Format Bild befindet sich an einer relativ Adresse ("../.....").

Hier eine vollständige Version inklusive dem HTML Tag. Der Applet Parameter fehlt, da das Bild im Applet spezifiziert wurde:

```
//Titel:       public Image getImage(URL path, String filename)
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: lesen eines Bildes aus einer URL plus Dateinamen
//package Beispiel62a;

import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.net.*;

public class getImageBeispiel extends Applet {

  Image imgBild;



  //Das Applet initialisieren
  public void init() {
    try  {
      imgBild = getImage(getDocumentBase(),"\\h225W195.gif"); //animated GIF
    }
    catch(Exception e)  {
      e.printStackTrace();
    }
  }


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

}


<applet code=getImageBeispiel width=200 height=230>
</applet><P>
```

6.3.3. Herunterladen von Sound

Das Paket java.applet.Applet besitzt mehrere Methoden, mit denen Audio Clips vom Web herunter geladen werden können. Zusätzlich können die Clips auch noch gleich gespielt werden. Schauen wie uns die Methoden an.

AudioClip	**getAudioClip**(URL url) liefert das Audio Clip, welches sich beim URL befindet..
AudioClip	**getAudioClip**(URL url, String name) liefert das AudioClip Objekt, welches mit Hilfe der URL und dem Namen spezifiziert wird.

Das Herunterladen des Clips wird durch den AppletContext besorgt. Ältere Browser und der AppletViewer unterstützen lediglich das *au* Format.

Es gibt verschiedene Werkzeuge, mit denen Audio Formate in das au Format umgewandelt werden können. Zudem werden in absehbarer Zeit sicher weitere Formate unterstützt, da im JavaMediaFramework bereits wesentlich mehr Formate bekannt sind (unter anderem MPEG).

Das eigentliche Abspielen des Clips erfolgt mit Hilfe der play(URL u) Methoden:

void	**play**(URL url) spielt das Clip, welches sich an der Adresse URL befindet.
void	**play**(URL url, String name) spielt das Clip, welches sich an der Adresse URL und der Relativadresse specifier befindet.

Schauen wir uns die zwei Methoden kurz an.

6.3.3.1. public void play(URL u)

Die play(URL u) Methode liest die Audio Datei von der URL Adresse. Falls das Clip gefunden wird, wird es herunter geladen und gespielt. Sonst geschieht nichts!

Programm Fragment:

```
try {
        play(new URL("http://www.pilatus.ch/mp3/hackersDream.au") );
} catch (MalformedURLException  mue) {
        System.err.println(mue);
}
```

6.3.3.1.1. Beispiel : Abspielen eines Audio Clips

Bei der Erstellung und beim Abspielen müssen einige Punkte beachtet werden:
1. im Applet Programm Code wird das Package auskommentiert
2. zum Applet gehört eine HTML Seite, hier mit folgendem HTML Code
   <applet code=AppletBeispiel63 width=50 height=20>
   <param name=AudioClip value=spacemusic.au></applet><P>
3. im Applet müssen die entsprechenden Parameter gleich genannt werden
   name=AudioClip, value=spacemusic.au
   stellt die Beziehung her zwischen dem Applet und dem Dateipfad

Und so sieht der Programmcode aus:

```
//Titel:       public void play(URL u)
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: Lesen eines Audio Clips von einer Adresse,
//die in Form eines URL Objektes gegeben ist.
//Herunterladen des Clips;
//Abspielen des Clips.
//package Beispiel63;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.net.*;

public class AppletBeispiel63 extends Applet {

  //Das Applet initialisieren
  public void init() {
    try  {
      URL urlAudioClip = new URL(getCodeBase(), getParameter("AudioClip"));
      play(urlAudioClip);
    }
    catch(MalformedURLException mue)  {
      System.err.println(mue);
      mue.printStackTrace();
    }
  }
}
```

Das Applet sieht nicht gerade sehr phantasievoll aus:

6.3.3.2. public void play(URL u, String filename)

Diese Methode funktioniert analog zur eben besprochenen, mit dem evidenten Unterschied, dass jetzt die URL aufgeteilt wird. Der Dateiname ist relativ zum URL definiert.

Programm Fragment:

```
play(new URL("http://www.3fach.ch/mp3/hackersParadies.au") , "jamiroquai,au") );
```

Die beiden Methoden getCodeBase() und getDocumentBase() weren in der Regel verwendet umd die Basis des URL's fest zu legen.

Falls die au Dateien immer im selben Verzeichnis sind wie die Applets, dann wird getCodeBase() eingesetzt;
falls die au Dateien im selben Verzeichnis stehen wie die HTML Seiten, dann ist getDocumentBase() die angepasste Methode.

Bei einer andern Verzeichnis kann man relativ zu diesen einfach zu bestimmenden Verzeichnissen (URL's) ein Verzeichnis festlegen.

Beispiel:

```
play(getDocumentBase(), "../assets/sounds/dragan.au");
```

Das AudioClip Interface stellt den eigentlichen Sound dar. Schauen wir uns dieses Interface genauer an.

6.3.3.3. java.applet
Interface AudioClip

---

public abstract interface **AudioClip**
Das `AudioClip` Interface ist eine einfache Abstraktion zum Spielen von Audio Clips. Mehrere `AudioClip` können gleichzeitig gespielt werden; der resultierende Sound ist ein Mix aller Clips.
Seit:
JDK1.0

6.3.3.3.1. Methoden Übersicht

`void`	**`loop`**`()` startet das Audio Clip mit einer endlosen Schleife.
`void`	**`play`**`()` startet das Abspielen des Audio Clips.
`void`	**`stop`**`()` stop das Abspielen des Audio Clips.

6.3.3.3.2. Methoden Details

play : public void play()

Startet das Abspielen des Audio Clips. Jedesmal, wenn diese Methode neu gestartet wird, startet das Abspielen von Neuem..

---

loop : public void loop()

Startet das Abspielen des Audio Clip in einer endlosen Schleife.

---

stop : public void stop()

Stoppt das Abspielen des Audio Clip.

Jetzt müssen wir nur noch genauer beschreiben, wie das Audio Clip herunter geladen wird.

Dafür benötigen wir die folgenden Methoden:

6.3.3.4. public AudioClip getAudioClip(URL u)

Zuerst müssen wir ein AudioClip Objekt haben. Dann haben wir die Möglichkeit diese Clips mit Hilfe der Methoden, die wir eben kennen gelernt haben, abspielen.

Und so sieht ein Programm Fragment aus:

```
...
import java.applet.AudioClip;
...
        AudioClip audioBeastieBoys_SureShot01;
        URL urlAudioClip;

        try {
                urlAudioClip = new URL("http://www.hiphopfreesite.ru/au/b/SureShot01.au");
                audioBeastieBoys_SureShot01 = getAudioClip(urlAudioClip);
        } catch (MalformedURLException mue) {
                System.err.println(mue);
        }
...
```

6.3.3.5. public AudioClip getAudioClip(URL u, String filename)

Im Gegensatz zur vorherigen Methode wird bei dieser die Adresse, an der sich das Audio Clip befindet, in zwei Teile (Basis URL und Dateinamen) aufgeteilt ist.

Ein Programm Fragment sieht entsprechend anders, aber ähnlich aus:

```
AudioClip ac = getAudioClip(new URL("http://www.pilatus.ch/", "Eden(2).au") );
```

Wie in der früheren Methoden eignet sich diese Form besser für die Adressierung, falls man mit getCodeBase() und / oder getDocumentBase() arbeitet.

Unter Verwendung dieser Methoden sieht das Programm Fragment wie folgt aus:

```
AudioClip ac = getAudioClip(getCodeBase(), "Eden(2).au") );
```

Die Dokumenten Basis ist das Verzeichnis, in dem sich die HTML Dateien befinden. die Code Basis beschreibt das Verzeichnis, in dem sich das Applet befindet.

Fassen wir die verschiedenen Methoden zusammen:

```
//Titel:       Applet zum Herunterladen eines Audio Files
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: einfaches Applet als Illustrationsbeispiel
```

```
//für die Audio Clip Methoden.
//package Beispiel64;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.net.*;

public class AppletBeispiel64 extends Applet implements Runnable{
  AudioClip audioSample;
  Thread threadPlayer;
  //Das Applet initialisieren
  public void init() {

      audioSample = getAudioClip(getDocumentBase(), "/sounds/pain.au");
      if (audioSample != null) {
        threadPlayer = new Thread(this);
        threadPlayer.start();
      }
  }
  // starten des Applets
  public void start() {
      threadPlayer.resume();
  }

  public void stop() {
      threadPlayer.suspend();
  }

  public void run() {

    Thread.currentThread().setPriority(Thread.MIN_PRIORITY);
    while(true) {
      audioSample.play();
      try {
        Thread.sleep(5000);
      } catch(InterruptedException e) {
      }
    }
  }
}


<applet code=AppletBeispiel64 width=180 height=150>
</applet><P>
```

6.4. Das ImageObserver Interface

Bei umfangreichen Web Seiten kann es vorkommen, dass das Herunterladen der Bilder sehr lange dauert. Es ist daher naheliegend, Bilder und Text getrennt, mit Hilfe zweier Threads, herunter zu laden.

Stellt sich nur die Frage, wie man überwachen oder kontrollieren kann, ob das Bild oder eine Multimedia Datei bereits am herunter laden ist, und wieviel der Datei bereits herunter geladen sind. Das java.awt.image.ImageObserver Interface erlaubt die Überwachung dieses Prozesses.

6.4.1. java.awt.image
Interface ImageObserver

wird implementiert von: Component

---

public abstract interface **ImageObserver**
ein asynchron arbeitendes Interface, welches Informationen über Bilder und den Aufbau von Bildern erhält.

6.4.1.1. Datenfelder

static int	**ABORT** Dieses Flag wird dann gesetzt, wenn der Aufbau eines Bildes abgebrochen wird, bevor das Bild zu Ende aufgebaut ist.
static int	**ALLBITS** Dieses Flag wird dann gesetzt, wenn das Bild vollständig aufgebaut ist.
static int	**ERROR** Dieses Flag wird dann gesetzt, wenn beim Aufbau eines Bildes ein Fehler auftrat.
static int	**FRAMEBITS** Dieses Flag zeigt an, dass ein Frame eines Multiframe Bildes bereit ist.
static int	**HEIGHT** Dieses Flag zeigt an, dass die Höhe des Bildes nun bekannt ist.
static int	**PROPERTIES** Dieses Flag zeigt an, dass die Informationen über das Bild nun zur Verfügung stehen.
static int	**SOMEBITS** Dieses Flag zeigt an, dass weitere Bits für den Aufbau des Bildes zur Verfügung stehen.
static int	**WIDTH** Dieses Flag zeigt an, dass die Breite des Bildes nun zur Verfügung steht.

6.4.1.2. Methoden Übersicht

boolean	**imageUpdate**(Image img, int infoflags, int x, int y, int width, int height) Diese Methode wird aufgerufen, falls die Informationen über ein Bild welches geladen wird, zur Verfügung stehen.

Die Methoden getWidth(), getHeight() mit denen die Breite und Höhe eines Bildes bestimmt werden, liefern -1 bis das Bild vollständig geladen ist.

Dafür gibt es eben das ImageObserver Interface.

Diverse Methoden, die sich mit Bildern befassen, verwenden dieses Interface:

Image.getWidth(ImageObserver)
Image.getHeight(ImageObserver)
Image.getProperty(String,ImageObserver)
Component.checkImage(Image,int,int,ImageObserver)

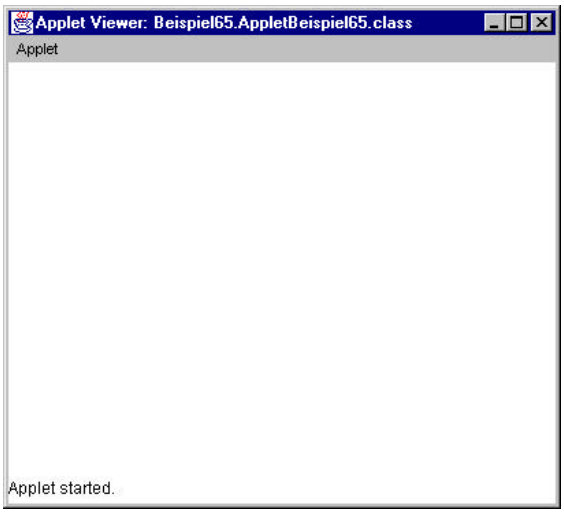
Schauen wir uns an einem Beispiel an, wie dies in der Praxis aussehen könnte:

6.4.2. Beispiel : Laden eines Bildes

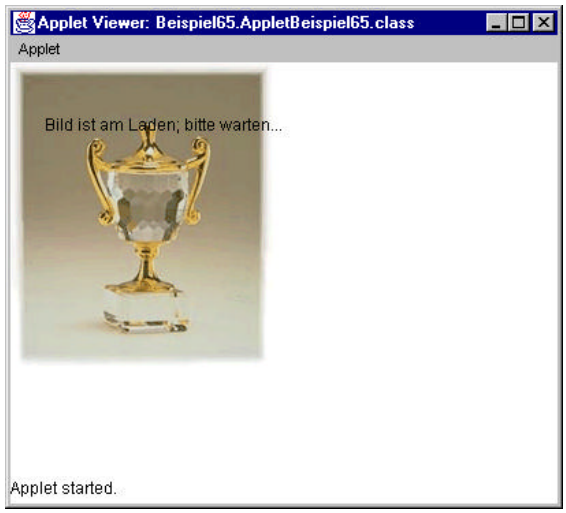
```
//Titel:       Asynchronous Image Loader
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: Multithreaded Image Loader
package Beispiel65;

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.awt.image.*;
import java.net.*;

public class AppletBeispiel65 extends Applet {

  Image imgBild;

  //Das Applet initialisieren
  public void init() {
     imgBild = getImage(getDocumentBase(), "Beispiel65/cq_winner.GIF");
```

```
  }
  public void paint(Graphics g) {
    if (g.drawImage(imgBild, 0, 0, this) ) {
      g.drawString("Bild ist am Laden; bitte warten...", 25,50);
    }
  }
  public boolean ImageUpdate(Image img, int infoflags, int x, int y, int width, int height) {
    if ((infoflags & ImageObserver.ALLBITS) == ImageObserver.ALLBITS) {
      repaint();
      return false;
    }
    else {
      return true;
    }
  }
}
```

Hier ein Screenshot vom Start



und nach dem Laden



Ein Problem hat das Applet offensichtlich: nachdem das Bild geladen wurde, steht der Text immer noch im Bild.

Aber das Grundschema des asynchronen Ladens eines Bildes dürfte trotzdem erkennbar sein.

6.5. Die Media Tracker Klasse

Das ImageObserver Interface ist hilfreich für das Verfolgen des Ladens von einem und nur einem Bild. Wenn man dagegen den Ladevorgang für mehrere Bilder oder umfangreicherer Multimedia Anwendungen verfolgen möchte, braucht man dazu ein ausgefeilteres Werkzeug.

Dieses wird von der Klasse java.awt.MediaTracker angeboten :
java.awt
Class MediaTracker

java.lang.Object
|
+--**java.awt.MediaTracker**

public class **MediaTracker**
extends Object
implements Serializable

Die Dokumentation dieser Klasse ist recht umfangreich und beschreibt auch gleich ein Anwendungsbeispiel. Die `MediaTracker` class ist eine Hilfsklasse, mit deren Hilfe der Status einer ganzen Menge von Medienobjekten verfolgt werden können. Media Objekte sind dabei Audio Clips oder Bilder. Zur Zeit werden jedoch nur Bilder unterstützt.

Um den Media Tracker benutzen zu können, muss eine Instanz der `MediaTracker` Klasse für jedes Bild, welches verfolgt werden soll, gebildet werden. Zusätzlich wird jedem Bild eine eindeutige Identifikation zugeordnet. Diese Identifikation kontrolliert die Priorität, mit welcher die Bilder geladen werden. Es können auch Gruppen von Bildern definiert werden. Bilder mit einem tieferen ID werden bevorzugt geladen.

Schauen wir uns ein einfaches Applet Beispiel an:

6.5.1. Beispiel : MediaTracker Applet

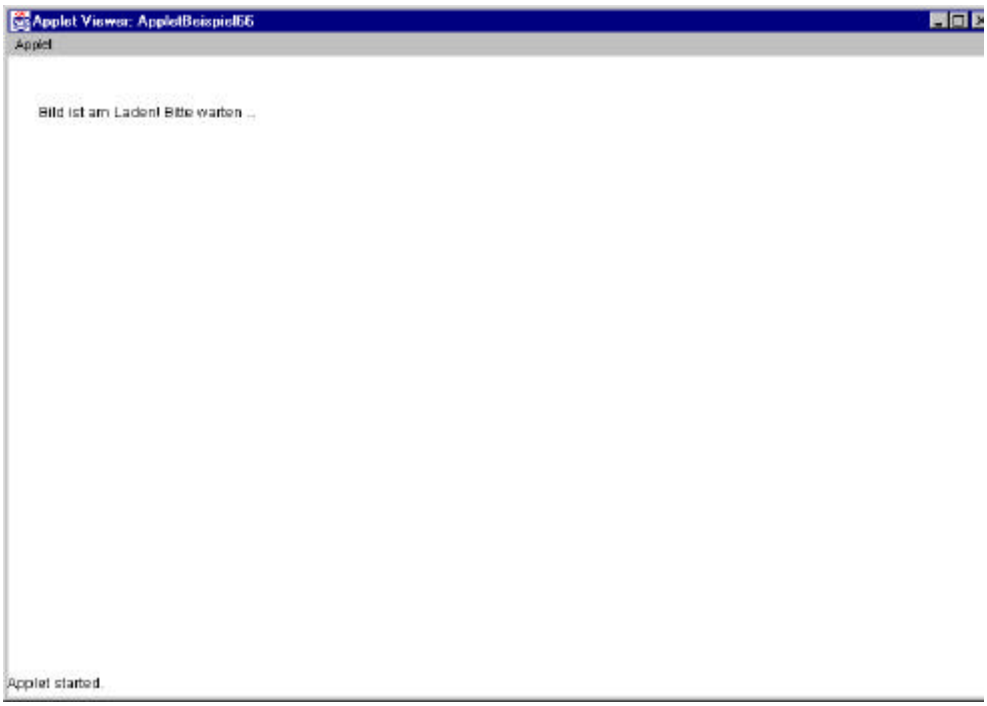
```
//Titel:       MediaTracker Class
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: Anwendung der MediaTracker Klasse
//in Form eines Applets, welches
//ein Bild über das Web herunter lädt
//package Beispiel66;

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

```
public class AppletBeispiel66 extends Applet implements Runnable{

  Thread play;
  Image imgBild;
  MediaTracker mtTracker;

  public void init() {

    imgBild = getImage(getDocumentBase(), getParameter("Bild"));
    mtTracker = new MediaTracker(this);
    mtTracker.addImage(imgBild, 1);

    play = new Thread(this);
    play.start();

  }

  public void run() {

    try {
      mtTracker.waitForID(1);
      repaint();
    }
    catch (InterruptedException ie) {
    }

  }

  public void paint(Graphics g) {

    if (mtTracker.checkID(1, true)) {
      g.drawImage(imgBild, 0, 0, this);
      play.stop();
    }
    else {
      g.drawString("Bild ist am Laden! Bitte warten ...", 25, 50);
    }

  }
}
```

Mit dem HTML Tag:

```
<applet code=AppletBeispiel66 width=100 height=100>
<param name=Bild value=Lokomotive.JPG>
</applet><P>
```

und dem Screenshot



Wie funktioniert das Applet?
Der Thread wird nur benötigt, um das Bild neu zu malen, sobald es vollständig geladen ist. Dies geschieht so, dass mtTracker.waitForID(1) abgefragt wird.

Schauen wir uns die Klasse etwas genauer an:

6.5.2. Der Konstruktor

Die Klasse hat einen Konstruktor

6.5.2.1. public MediaTracker(Component comp)

Dieser liefert ein MediaTracker Objekt, welches Bilder einer bestimmten Komponente beobachtet.

In der Regel wird der Konstruktor im folgenden Kontext aufgerufen:

```
MediaTracker mtTracker = new MediaTracker(this);
```

Der Konstruktor hat ein Argument, die Konponente Component, welche das Bild darstellen wird. Da der Konstruktor meistens innerhalb der Komponente, auf die sie angewandt wird, aufgerufen wird, ist das Argument in der Regel this.

6.5.3. Bilder dem MediaTracker zuordnen

Diese Methode :

6.5.3.1. public void addImage(Image img, int id)

fügt das Bild img mit der Identifikation id der Liste der zu beobachtenden Bilder des MediaTrackers zu.

Hier ein Programm Fragment für die Definition der Verbindung zum Bild:

```
Image imBild = getImage(getDocumentBase(), "Fiat500Turbo.gif");
MediaTracker mtTracker = new MediaTracker(this);
mtTracker.addImage(imgBild, 1);
```

Die Bilder werden in der Reihenfolge ihrer ID's geladen. Falls mehrere Bilder die selbe ID haben, dann wird ein Bild zufällig ausgewählt.

Das eigentliche Laden des Bildes geschieht nicht mit Hilfe dieses Hinzufügens.
Man muss im Programm die Methode checkID(ID, true) oder checkAll(true) oder eine der wait() Methoden aufrufen; falls die Grösse des Bildes skaliert werden soll, dann addImage() aufgerufen werden:

6.5.3.2. public synchronized void addImage(Image im, int id, int w, int h)

Diese Version von addImage() fügt ein Bild img mit derIdentität id zur Liste der Bildobjekte hinzu, die vom Medien Tracker beobachtet werden.

Betrachten wir zum bessern Verständnis ein Programm Fragment:

```
Image imBild = getImage(getDocumentBase(), "Fiat500Turbo.gif");
MediaTracker mtTracker = new  MediaTracker(this);
mtTracker.addImage(imgBild, 1, 30, 30);
```

6.5.4. Überprüfen des Medienstatus

Jetzt müssen wir uns dem Ladestatus der einzelnen Bilder etwas genauer annehmen:

6.5.4.1. public boolean checkID(int id)

Diese Methode prüft, ob alle Bilder mit der ID, welche vom Medientracker beobachtet werden, vollständig geladen sind.

Falls dies der Fall ist, liefert der Methodenaufruf den Wert true.

Im folgenden Programm Fragment zeichnet die paint() Methode ein Bild nur dann, falls es bereits vollständig geladen wurde.

```
public void paint(Graphics g) {
        if (mtTracker.checkID(1) ) {
                g.drawImage(imgBild, 0, 0, this);
        } else {
                g.drawString("Bilder werden geladen! Bitte warten ...", 25, 50);
        }
}
```

6.5.4.2. public boolean checkAll()

Diese Methode prüft, ob alle Image Objekte, deren Laden wir verfolgen wollen, vollständig geladen wurden. Die ID Nummern werden dabei nicht berücksichtigt.
Falls alle Bilder geladen wurden, dann liefert checkAll() den Wert true, sonst false. Methode lädt kein Bild.

6.5.4.3. public synchronized boolean checkAll(boolean load)

Diese Version der Methode funktioniert analog zur eben beschriebenen Methode mit einer Ausnahme:
falls die Methode den Wert false zurück liefert, also noch nicht alle Bilder geladen wurden, und falls der Parameter load true ist,
dann werden alle noch nicht geladenen Bilder geladen.

6.5.5. Warten auf das Laden vom Medium

Die nächsten drei Methoden laden Bilder, die vom Media Tracker verfolgt werden, und warten, bis die Bilder geladen sind.
Die Programmausführung stoppt so lange, bis die Media Objekt geladen sind. Falls man also mit der Programmausführung weiter fahren möchte, dann sollte man diese Methoden innerhalb spezieller Threads aufrufen.

6.5.5.1. public void waitForID(int id) throws InterruptedException

Diese Methode lädt das Bild mit der ID id und wartet bis das Bild vollständig geladen ist oder der Ladevorgang abgebrochen wurde, oder eine Exception geworfen wird.
Die InterruptException wird geworfen, falls ein anderer Thread diesen Thread stoppt.

Programm Fragment zum Laden eines Image Objektes mit der ID id:

```
try {
        mtTracker.waitForID(id)
} catch(InterruptedException e) {
        …}
```

6.5.5.2. public synchronized boolean waitForID(int id, long ms) throws InterruptedException

Diese Wartemethode lädt das Bild mit der ID id und wartet, bis das Bild geladen ist, ein Fehler auftrat oder eine Exception geworfen wird oder eine bestimmte Zeitperiode verstrichen ist. Die Zeitperiode wird in Milisekunden ms angegeben.

Programm Fragment (Wartezeit : maximal 120 000 Millisekunden, also 2 Minuten) :

```
try {
        mtTracker.waitForID(1, 120000);
} catch(InterruptedException) {
…}
```

6.5.5.3. public synchronized boolean waitForAll() throws InterruptedException

Diese Methode lädt alle Bilder. Sie wird unterbrochen entweder im Fehlerfall, durch einen anderen Thread oder abgeschlossen, wenn alle Bilder geladen sind.
Diese Methode wird typischerweise in Animationsprogrammen eingesetzt, da dort alle Frames einer Animation geladen werden müssen.

Programm Fragment:

```
try {
        mtTracker.waitForAll();
} catch(InterruptedException) {
…
}
```

Da der Ladeprozess lange dauern kann, sollte der Ladestatus angezeigt werden.

6.5.5.4. public synchronized boolean waitForAll(long ms)

Diese Methode ist der eben beschrieben ähnlich: sie lädt alle Bilder und wartet aber maximal ms Millisekunden.

Ein Beispiel erübrigt sich, da es analog zum obigen Beispiel (ms=120000) ist.

6.5.6. Fehlerbehandlung

Fehler beim Laden des Bildes führen zum unmittelbaren Abbruch des Ladevorganges. Das Bild gilt zudem als geladen. Die Fehlerbehandlung muss also heraus finden, welche Bilder den Abbruch überlebt beziehungsweise welches Bild den Absturz verursacht hat.

Die folgenden Methoden dienen diesen Abklärungen.

6.5.6.1. public synchronized boolean isErrorAny()

Die Methode zeigt an, ob beim Laden des Objektes, ein Fehler auftrat.

Programm Fragment:

```
if (mtTracker.isErrorAny() ) {
        System.err.println("Es trat ein Fehler beim Laden eines Bildes auf");
}
```

Diese Methode ist indifferent und lässt keine Rückschlüsse zu. Es bleibt unklar, welches Bild nicht geladen werden konnte. Es kann auch sein, dass mehrere Bilder den Abbruch verursacht haben.
Es verbietet sich daher die Gruppen von zusammen gehörigen Bildern zu gross zu wählen : es wäre schwierig heraus zu finden, welches Bild den "Absturz" verursacht hat.

6.5.6.2. publc synchronized Object[ ] getErrorsAny()

Mit dieser Methode erhält man ein Datenfeld mit allen Bildern, die vom MediaTracker beobachtet wurden und die, einzeln oder als Gruppe, am Fehler beteiligt waren.
Falls kein Fehler auftrat, dann liefert die Methode das null Objekt.

Hier ein Programm Fragment:

```
if (mtMediaTracker.isErrorAny() ) {
        System.err.println("Das folgende Medien Objekt hat Probleme : ");
        Object[ ] medFailed = mtTracker.getErrorAny();
        for (int i=0; i<medFailed.length; i++) {
                System.err.println(medFailed[i]);
        }
}
```

6.5.6.3. public synchronized boolean isErrorID(int id)

Mit dieser Methode können wir prüfen, ob das Medien Objekt mit der ID id beim Laden eine Ausnahme ausgelöst hat. Wenn nicht, dann liefert die Methode den Wert false, sonst true.

Zu beachten ist, dass eine ID oft mehr als ein Medien Objekt beschreibt. Es ist also nicht ganz so einfach, den Verursacher des Fehlers zu eruieren.

Die Abfrage in einem Programm sieht beispielsweise wie folgt aus:

```
if (mtTracker.isErrorID(2)) {
        System.err.println("Beim Laden des Medien Objektes mit der ID 2 trat ein
Ausnahmezustand auf");
}
```

Soweit die Fehlerverfolgung. Schauen wir uns nun an, wie man den Ladezustand des Objektes beobachten und verfolgen kann.

6.5.7. Statusprüfung der Medien Objekte

Die Verfolgung des Ladezustandes von Medien Objekten geschieht gruppenweise, also ID bezogen.
Dieser Status wird mit Hilfe von Statusvariablen festgehalten:

Datenfelder Übersicht	
`static int`	**`ABORTED`** (Wert : 2) Das Herunterladen eines Objektes ist schief gelaufen und wurde abgebrochen.
`static int`	**`COMPLETE`** (Wert : 8) Das Laden des Medien Objektes war erfolgreich..
`static int`	**`ERRORED`** (Wert : 4) Beim Herunterladen trat ein Fehler auf.
`static int`	**`LOADING`** (Wert : 1) Ein Medien Objekt wird gerade herunter geladen..

Betrachten wir die zugeordneten Methoden:

6.5.7.1. public int statusAll(boolean load)

Die Methode liefert den Status des Ladevorganges. Falls die load Variable true gesetzt wird, dann startet der MediaTracker den Ladevorgang, falls das Objekt noch nicht geladen wurde.

Programm Fragment:

```
if (med.statusAll(false) & ImageObserver.COMPLETED) == ImageObserver.COMPLETED)
        ...
```

6.5.7.2. public int statusID(int i, boolean load)

Die Methode funktioniert analog zur eben besprochenen mit dem Zusatz, dass das Medien Objekt mit der ID i beobachtet wird.

Da die ID id mehrere Medien Objekte umfasst, ist die Aussage eher unpräzise.

6.6. Die Netzwerk Methoden der AppletContext Schnittstelle

Mit Hilfe der AppletContext Schnittstelle kommuniziert das Applet mit seinem Umfeld. Jedes Applet besitzt ein AppletContext Objekt. Dieses wird vom Browser oder dem AppletViewer geliefert.
Das resultierende Context Objekt stellt verschiedene Methoden zur Verfügung, mit dessen Hilfe der Status des Kontext abgefragt werden kann.

6.6.1. Methoden Übersicht

Tabellarische Übersicht	
Applet	**getApplet**(String name) Finde und liefere das Applet, zu dem dieser Kontext, spezifiziert mit Hilfe des Namens, gehört.
Enumeration	**getApplets**() Finde alle Applets im Dokument, die zum Kontext gehören..
AudioClip	**getAudioClip**(URL url) Kreiere ein Audio Clip.
Image	**getImage**(URL url) Liefert ein Bild Objekt, welches angezeigt werden kann.
void	**showDocument**(URL url) Ersetze die aktuelle Web Seite durch die im URL spezifizierte Seite.
void	**showDocument**(URL url, String target) Zeige die URL an; Target spezifiziert das HTML Frame.
void	**showStatus**(String status) Zeige die Zeichenkette im Statusfenster an.

Bemerkung:
showDocument() wird nicht von allen Browsern unterstützt.

Target : _top, _self, ... oder wie im HTML Code spezifiziert.

Schauen wir uns als Anwendungsbeispiel einen Redirector an.

6.6.1.1. Redirector Beispiel Applet

```
//Titel:       "Redirector"
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: Applet, welches vom Parameter den neuen Host liest , die neue URL berechnet
und dann "anzeigt"
package Beispiel67;

import java.applet.*;
import java.net.*;

public class AppletBeispiel67 extends Applet {
  //Das Applet initialisieren
  public void init() {

    AppletContext ac = getAppletContext();
    URL urlAlt = getDocumentBase();

    try  {
      URL urlNew = new URL(getParameter("neuerHost")+urlAlt.getFile() );
      ac.showDocument(urlNew);
    }
    catch(MalformedURLException mue)  {
      mue.printStackTrace();
    }
  }
}
```

Dieses Programm verdeutlicht nur das Grundschema. Diese Funktionalität würde man ganz anders implementieren.

6.7. Umfangreiche Beispiele

6.7.1. Image Sizer

In HTML spezifiziert man sinnvollerweise im <IMG Tag jeweils die Parameter "hight" und width". Dies ermöglicht es dem Browser, den Text schneller als die Bilder zu laden, da er die "Leerräume" bereits kennt.

Das folgende Programm liest eine HTML Seite und fügt allen <IMG Tags die Dimensionen hinzu.

Das Programm verwendet die Methode getImage(), um das Bild zu lesen. Das Bildobjekt wird dann mit Hilfe der zwei Methoden getWidth() und getHeight() analysiert und die so bestimmte Höhe und Weite in die Tags eingefügt, falls noch keine Angaben vorhanden sind.

```
//Titel:        Image Sizer
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  liest eine HTML Seite und ergänzt die Image Tags.
package Beispiel68;

import java.net.*;
import java.io.*;
import java.awt.*;
import java.awt.image.*;

public class ImageSizer extends Frame {

  TextField theURL;
  TextArea theOutput;
  Button getURL;

  public static void main(String[] args) {

    ImageSizer is = new ImageSizer();
    is.resize(300, 200);
    is.show();

  }

  public ImageSizer() {

    setLayout(new BorderLayout());
    theURL = new TextField(40);
    add("North", theURL);
    theOutput = new TextArea(80, 40);
```

```
  add("Center", theOutput);
  add("South", new Button("Get URL"));

 }

 public boolean action(Event e, Object o) {
  if (e.target instanceof Button) {
    fixThePage(theURL.getText());
    return true;
  }
  return false;

 }

 public void fixThePage(String thePage) {

  String thisLine;
  URL root;

  if  (thePage != null) {

    //Open the URL for reading
    try {
      if (thePage.indexOf(":") != -1) {
        root = new URL(thePage);
      }
      else {
        root = new URL("http://" + thePage);
      }
      sizeAPage(root);
    }
    catch (MalformedURLException e) {
      System.err.println(thePage + " is not a parseable URL");
      System.err.println(e);
    }
  } //  end if

 } // end main


 public void sizeAPage(URL u) {

  char thisChar;
  String theTag;

  try {
    DataInputStream theHTML = new DataInputStream(u.openStream());
    try {
      while (true) {
        thisChar = (char) theHTML.readByte();
```

```
        if (thisChar == '<') {
          theTag = readTag(theHTML);
          if (theTag.startsWith("<IMG")) {
            theTag = sizeImage(u, theTag);
          }
          theOutput.appendText(theTag);
        }
        else {
          theOutput.appendText(String.valueOf(thisChar));
        }
      } // end while
    }  // end try
    catch (EOFException e) {
      //
    }
  }  // end try
  catch (IOException e) {
    System.err.println(e);
  }

}

  public String readTag(DataInputStream is) {

    StringBuffer theTag = new StringBuffer("<");
    char theChar = '<';

    try {
       while (theChar != '>') {
         theChar = (char) is.readByte();
         theTag.append(theChar);
       } // end while
    }  // end try
    catch (EOFException e) {
      // Done with the Stream
    }
    catch (Exception e) {
       System.err.println(e);
    }

    return theTag.toString();

  }


  public String sizeImage(URL u, String tag) {

    String s1 = tag.toUpperCase();

    boolean hasHeightTag = s1.indexOf("HEIGHT") != -1;
```

```
boolean hasWidthTag = s1.indexOf("WIDTH") != -1;

if (hasHeightTag && hasWidthTag) {
  return tag;
}
else {
  String newTag;
  Image thePicture;
  int p1, p2, p3, p4;
  p1 = s1.indexOf("SRC");
  p2 = s1.indexOf ("=", p1);
  p3 = s1.indexOf("\"", p2);
  p4 = s1.indexOf("\"", p3+1);
  String theURL = tag.substring(p3+1, p4);
  URL thePictureURL;
  try {
    if (theURL.indexOf(":")  == -1) {
      // it's not an absolute URL
      thePictureURL = new URL(u, theURL);
    }  // end if
    else {
      thePictureURL = new URL(theURL);
    }
    Toolkit t = Toolkit.getDefaultToolkit();
    thePicture = t.getImage(thePictureURL);
    thePicture.getHeight(this);
    int last = tag.indexOf(">");
    newTag = tag.substring(0,last);
    if (!hasWidthTag) {
      while (thePicture.getWidth(this) == -1) {
        try {
          Thread.currentThread().sleep(100);
        }
        catch (InterruptedException e) {
        }
      }
      newTag = newTag + " width=" + thePicture.getWidth(this);
    }
    if (!hasHeightTag) {
      while (thePicture.getHeight(this) == -1) {
        try {
          Thread.currentThread().sleep(100);
        }
        catch (InterruptedException e) {
        }
      }
      newTag = newTag + " height=" + thePicture.getHeight(this);
    }
    newTag = newTag +  ">";
  }
```

```
      catch (MalformedURLException e) {
        newTag = tag;
      }
      return newTag;
    }

  }

  // Das Applet liest soviel vom Bild, bis es die Grösse bestimmen kann
public boolean imageUpdate(Image img, int infoflags, int x, int y, int width, int height) {
    if ((infoflags & ImageObserver.HEIGHT) == ImageObserver.HEIGHT  &&
        (infoflags & ImageObserver.WIDTH) == ImageObserver.WIDTH)  {
        return false;
    }
    else {
      return true;
    }
  }
}
```

Die Ausgabe der modifizierten Seite geschieht in das (zentrale) Fenster. Das Programm verwendet Teile des Page Downloaders. Es hat auch die selben Probleme: oft lässt sich die Applikation nicht problemlos beenden. Es gäbe also viel zu tun, wollte man daraus eine brauchbare Applikation machen.

6.7.2. Animation Player

Die Animation geschieht dadurch, dass die einzelnen Bilder geladen werden und der Reihe nach angezeigt werden. Wir haben im Programmieren II bereits ein solches Programm erstellt, allerdings nicht als Applet und somit ohne MediaTracker.

```
//Titel:       Animation Player
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: einfacher Animator : lädt eine Bildsequenz und zeigt diese an
package Beispiel69;
import java.net.*;
import java.awt.*;
import java.awt.image.*;
import java.applet.*;
import java.util.*;

public class Animator extends Applet implements Runnable {

  boolean running = false;
  int thisCell = 0;
  Vector cells;
  Thread play;
  MediaTracker theTracker;

  public void init() {

    String nextCell;
    cells = new Vector();
    theTracker = new MediaTracker(this);
    for (int i = 0; (nextCell = getParameter("Cell" + i)) != null ; i++) {
      Image img = getImage(getDocumentBase(), nextCell);
      cells.addElement(img);
      theTracker.addImage(img, i);
      System.out.println("i="+i); // zum Testen
    }

    // start loading the images
    theTracker.checkAll(true);
    play = new Thread(this);
    play.start();
    running = true;

  }
```

```
  public void run() {

    for (thisCell=0; thisCell < cells.size(); thisCell++) {

      try {
        // make sure this cell is loaded
        theTracker.waitForID(thisCell);
        // paint the cell
        repaint();
        // sleep for a tenth of a second
        // i.e. play ten frames a second
        play.sleep(1000);
      }
      catch (InterruptedException ie) {
      }
    }
  }
  public void stop() {
    play.suspend();
  }
  public void start() {
    play.resume();
  }
  public void paint(Graphics g) {
    g.drawImage((Image) cells.elementAt(thisCell), 0, 0, this);
  }

  // The convention is that a mouseClick starts
  // a stopped applet and stops a running applet.
  public boolean mouseUp(Event e, int x, int y) {

    if (running) {
      play.suspend();
      running = false;
    }
    else {
      play.resume();
      running = true;
    }

    return true;

  }

}
```

6.7.3. Image Map in Java

Image Maps lassen sich sehr einfach Client-seitig mit Hilfe eines ImageMapper Programms erstellen. Das Beispiel dient lediglich der Illustration einiger Möglichkeiten, sollte aber so nicht im Web eingesetzt werden.

Und hier ist die Programme (die Klasse MapArea wird weiter unten als abstrakte Klasse definiert):

6.7.3.1. Image Map Komponente

```
import java.awt.*;
import java.net.*;
import java.util.*;
import java.applet.*;

public class ImageMap extends Canvas {

  Image theMap;
  Vector areas;
  AppletContext ac;

  public ImageMap(Image img, AppletContext browser) {

    theMap = img;
    // make sure the Image is loading
    theMap.getWidth(this);
    ac = browser;
    areas = new Vector();
  }

  public void addMapArea(String s) {

    if (s.startsWith("rect")) {
      try {
        RectArea r = new RectArea(s);
        areas.addElement(r);
      }
      catch (MalformedURLException e) {
        System.err.println(e);
      }
    }
    else if (s.startsWith("circle")) {
      ;
    }
    else if (s.startsWith("poly")) {
      ;
    }
    else if (s.startsWith("oval")) {
      ;
```

```
    }
    else {  // unrecognized tag
      ;
    }

  }


  public boolean mouseDown(Event evt, int x, int y) {
    for (Enumeration e = areas.elements() ; e.hasMoreElements() ;) {
      MapArea a = (MapArea) e.nextElement();
      if (a.contains(x, y)) {
        ac.showDocument(a.getURL());
        break;
      }
    }
    return true;

  }


  public void paint(Graphics g) {

     if(!g.drawImage(theMap, 0, 0, this)) {
       g.drawString("Loading Picture. Please hang on", 25, 50);
     }

  }

  public Dimension minimumSize() {

    return new Dimension(theMap.getWidth(this), theMap.getHeight(this));

  }

  public Dimension preferredSize() {

    return minimumSize();

  }
}
```

6.7.3.2. Applet Programm-Code

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

public class ImageMapApplet extends Applet {

  public void init() {

    String nextarea;
    int i = 1;

    setLayout(new BorderLayout());
    String filename = getParameter("filename");
    Image theMap = getImage(getDocumentBase(), filename);
    ImageMap im = new ImageMap(theMap, getAppletContext());
    add("Center", im);
    while ((nextarea = getParameter("area" + i++)) != null) {
      im.addMapArea(nextarea);
    }

  }


}
```

6.7.3.3. Abstrakte Map Area Klasse

```
import java.net.URL;

public abstract class MapArea {

  URL u;

  public abstract boolean contains(int x, int y);

  public URL getURL() {
    return u;
  }

  public abstract String toString();

}
```

6.7.3.4. Rechteck Klasse

```
import java.awt.Rectangle;
import java.util.StringTokenizer;
import java.net.*;

public class RectArea extends MapArea {

  Rectangle r;

  // An NCSA format String looks like
  // rect http://www.macfaq.com 227,0 340,65
  public RectArea (String s) throws MalformedURLException {

    int x1, y1, x2, y2;

    StringTokenizer st = new StringTokenizer(s, " \n\t\r,");
    // throw away the first token
    // This should be rect
    st.nextToken();
    u = new URL(st.nextToken());
    x1 = Integer.parseInt(st.nextToken());
    y1 = Integer.parseInt(st.nextToken());
    x2 = Integer.parseInt(st.nextToken());
    y2 = Integer.parseInt(st.nextToken());
    r = new Rectangle(x1, y1, x2 - x1, y2 - y1);
  }

  public boolean contains(int x, int y) {
    return r.inside(x, y);
  }

  public String toString() {

    return "rect " + u + " " + r.x + "," + r.y + " " + (r.x + r.width) + "," + (r.y + r.height);
  }


}
```

6.7.3.5. HTML Code

```
<applet code="ImageMapApplet.class" width=345 height=70>
<param name="area1" value="rect http://www.macfaq.com 227,0 340,65">
<param name="filename" value="map.gif">
<param name="area2" value="rect http://metalab.unc.edu 114,0 227,65">
<param name="area3" value="rect http://www.ora.com/ 1,0 114,65">
</applet><P>
```

6.7.3.6. Image (Map)

6.7.3.7. Funktionsweise

Mit Hilfe der Parameter werden die URL's an das Applet weitergegeben. Beim Klicken auf eines der obigen Rechtecke wird die entsprechende URL geladen.

6.7.4. Multihoming

Unter Multihoming versteht man die Technik, mehrere Web Sites auf einem Rechner zu speichern und einzeln an zu sprechen.

Das ist in der Regel kein Problem. Beim MacIntosh war das ein Problem, weil jedem Rechner nur eine IP Adresse zugeteilt werden kann.

6.7.4.1. Der Applet Programmcode

```
import java.applet.*;
import java.net.*;


public class Multihome extends Applet {

  public void init() {

    AppletContext ac = getAppletContext();
    URL oldURL = getDocumentBase();
    try {
      URL newURL = new URL(oldURL.getProtocol() + "://" + oldURL.getHost() + "/" +
oldURL.getHost() + oldURL.getFile());
      ac.showDocument(newURL);
    }
    catch (MalformedURLException e) {
    }

  }
}
```

6.8. Zusammenfassung

Die Applets erlauben das Lesen ab dem Host, von dem sie ursprünglich stammen.
Die Netzwerk Funktionalität beschränkt sich in der Regel auf das Lesen von Dateien ab einem URL.
Zusätzlich bietet die Applet Klasse viele Möglichkeiten, die Ladevorgänge zu verfolgen.
Diese Methoden sind allerdings mit Vorsicht einzusetzen. Das Tracken des Ladevorganges eines Media Objektes ist zwar verfolgbar; häufig jedoch nicht präzise genug.