**In diesem Kapitel:**
- *Die URLConnection Clas*
- *API Beschreibung*
- *einführende Beispiele*
- *MIME Types*
- *einige brauchbare Programme*

# 10

# *Die URLConnection Klasse*

## *10.1. Einführung*

Die URLConnection Klasse ist, wie wir sehen werden, eine abstrakte Klasse, mit der eine aktive Verbindung zu einer URL beschrieben werden kann. Die Klasse kann zwei unterschiedliche Aufgaben erfüllen:

- die Klasse erlaubt eine bessere Kontrolle über die Kommunikation mit dem Server als die URL Klasse.
  Mit Hilfe der URLConnection kann man den MIME Header, den der HTTP Server sendet, analysieren und daher eine passende Antwort zurück senden.
  Sie können mit Hilfe einer URLConnection auch eine binäre Datei herunter laden.
  Die URLConnection erlaubt es auch, mit Hilfe der POST Methode, Daten an einen HTTP Server zu senden.
- die URLConnection Klasse gehört zu den *Protocol Handler Klassen*, zu denen auch die URLStreamHandler Klasse gehört.
  Die Idee, die einer Protocol Handler Klasse zu Grunde liegt, ist sehr einfach:
  das Verarbeiten der Protokolle und die Verarbeitung der Datentypen wird getrennt; die Arbeiten, die ein Web Browser als Monolith erledigt, werden in die Bestandteile zerlegt und einzeln betrachtet.

Die URLConnection Klasse ist abstrakt. Um ein bestimmtes Protokoll zu implementieren, muss man eine Unterklasse definieren.

Das lässt der Phantasie viel Spielraum: es wäre im Prinzip möglich, beim Herunterladen einer Datei rasch fest zu stellen, von welchem Typus diese Datei ist, dann den passenden Protokoll Handler zu suchen und über das Netzt herunter zu laden und schliesslich die Datei gemäss Protokoll auszuwerten.

Im nächsten Kapitel werden wir uns mit den Protocol Handlern genauer befassen.

Sun hat zwei solcher spezieller Protokoll Handler mitgeliefert:
1. die HttpURLConnection,
2. die JarURLConnection (zur Behandlung von Java Archive Dateien)

### 10.2. Globale Lernziele

Sie kennen die Methoden, Datenfelder und den Konstruktor der Klasse URLConnection. Sie können die (abstrakte) Klasse in konkreten Klassen einsetzen.

### 10.3. Aufbau

Wie beschreiben zuerst die abstrakte Klasse mit ihren Datenfeldern und Methoden und zeigen, wie die Klasse in einfachen Beispielen eingesetzt werden könnte.
Anschliessend werden wir komplexere Beispiel kennen lernen:

- binäre Dateien herunter laden (Sound und Bilder)
- Formulare an eine HTTP Server senden

Schauen wir uns zuerst den Aufbau der Klasse, gemäss Klassenbeschreibung in JavaDoc an:

### 10.4. java.net Class URLConnection

```
java.lang.Object
   |
   +--java.net.URLConnection
```

**Direkt Bekannte Unterklassen:**
HttpURLConnection, JarURLConnection

public abstract class **URLConnection**
extends Object

Die abstrakte Klasse `URLConnection` ist die Oberklasse aller Klassen, die eine Kommunikationsverbindung zwischen der Applikation und einer URL repräsentieren. Mit Hilfe von Instanzen dieser Klasse kann man von einer Resource im Netz, die als URL bekannt ist, lesen und schreiben. Im Allgemeinen besteht die Kreation einer Verbindung zu einer URL aus einem mehrstufigen Prozess.:

| `openConnection()` | `connect()` |
|---|---|
| Festlegen der Paramter, welche die Verbindung zur entfernten Resource beeinflussen. | Wechselwirkung mit der Resource : Header Felder abfragen und Inhalt analysieren. |

---------->
Zeitachse

1. Das Verbindungsobjekt wird kreiert. Dazu wird die `openConnection` Methode einer URL ausgeführt.
2. Die Setup Parameter und allgemeine Anfrage- Eigenschaften werden gesetzt.
3. Die aktuelle Verbindung zum entfernten Objekt wird aufgebaut, mit Hilfe der `connect` Methode.
4. Das remote Objekt wird verfügbar. Auf die Header Felder und der Inhalt des remote Objektes kann zugegriffen werden.

Die Setup Parameter werden mit folgenden Methoden modifiziert:

- `setAllowUserInteraction`

- `setDoInput`
- `setDoOutput`
- `setIfModifiedSince`
- `setUseCaches`

und die allgemeinen Anfrage Parameter werden mit Hilfe der folgenden Methode gesetzt:

- `setRequestProperty`

Default Werte für `AllowUserInteraction` und `UseCaches` Parameter kann man mit Hilfe der Methoden

- `setDefaultAllowUserInteraction` und
- `setDefaultUseCaches setzen`

Default Werte für generelle Request Eigenschaften kann man mit Hilfe der Methode

- `setDefaultRequestProperty` setzen.

Zu jeder der obigen Methoden gibt es eine entsprechende `get` Methode, um die Werte der Parameter zu lesen. Die spezifischen Parameter und Request Eigenschaften sind Protokoll abhängig.

Die folgenden Methoden werden eingesetzt, um auf die Header Felder und den Inhalt zuzugreifen, nachdem die Verbindung zum entfernten Objekt aufgebaut ist:

- `getContent`
- `getHeaderField`
- `getInputStream`
- `getOutputStream`

Einige der Header Felder werden häufig benutzt. Die Methoden:

- `getContentEncoding`
- `getContentLength`
- `getContentType`
- `getDate`
- `getExpiration`
- `getLastModifed`

erlauben den bequemen Zugriff auf diese Felder. Die `getContentType` Methode wird von der `getContent` Methode eingesetzt, um den Typus eines remote Objektes festzustellen; Unterklassen überschreiben oft die `getContentType` Methode.

Im Allgemeinen kann man die Einstellungen übernehmen: alle Prä-Verbindungsparameter und allgemeine request Eigenschaften kann man ignorieren. Für die meisten Clients, die dieses Interface einsetzen, gibt es eigentlich nur zwei interessante Methoden:

- `getInputStream` und
- `getObject`,

Mehr Information zu Request Eigenschaften und Header Felder einer `http` Verbindung findet

man im Netz:

http://www.w3.org/hypertext/WWW/Protocols/HTTP1.0/draft-ietf-http-spec.html

**Seit:**

JDK1.0

**Siehe Auch:**

`URL.openConnection()`, `connect()`, `getContent()`, `getContentEncoding()`, `getContentLength()`, `getContentType()`, `getDate()`, `getExpiration()`, `getHeaderField(int)`, `getHeaderField(java.lang.String)`, `getInputStream()`, `getLastModified()`, `getOutputStream()`, `setAllowUserInteraction(boolean)`, `setDefaultRequestProperty(java.lang.String, java.lang.String)`, `setDefaultUseCaches(boolean)`, `setDoInput(boolean)`, `setDoOutput(boolean)`, `setIfModifiedSince(long)`, `setRequestProperty(java.lang.String, java.lang.String)`, `setUseCaches(boolean)`

### 10.4.1. Datenfelder Übersicht

| | |
|---|---|
| protected boolean | **`allowUserInteraction`**<br>Falls `true`, dann wird auf diese `URL` in einem Kontext zugegriffen, in dem es Sinn macht, Benutzerinteraktionen, wie zum Beispiel einen Authentisierungs-Dialog anzuzeigen. |
| protected boolean | **`connected`**<br>Falls `false`, dann hat dieses Verbindungsobjekt keine Kommunikations-Verbindung zum spezifizierten URL aufgebaut. |
| protected boolean | **`doInput`**<br>Diese Variable wird mit Hilfe der `setDoInput` Methode gesetzt. |
| protected boolean | **`doOutput`**<br>Diese Variable wird mit Hilfe der `setDoOutput` Methode gesetzt. |
| protected long | **`ifModifiedSince`**<br>Einige Protokolle erlauben es, Objekte nur dann zu lesen, falls diese Objekte vor weniger als einem bestimmten Zeitpunkt modifiziert werden. |
| protected URL | **`url`**<br>Die URL beschreibt das remote Objekt im World Wide Web zu dem diese Verbindung führt. |
| protected boolean | **`useCaches`**<br>Falls `true`, dann kann das Protokoll auch Caching verwenden. |

### 10.4.2. Konstruktor Übersicht

| | |
|---|---|
| protected | **`URLConnection`**`(URL url)`<br>Konstruiert eine URL Verbindung zu einer spezifizierten URL. |

### 10.4.3. Methoden Übersicht

| | |
|---|---|
| abstract void | **`connect`**`()`<br>Öffnet eine Verbindung zu einer URL, falls die Verbindung noch nicht besteht. |

| | |
|---|---|
| boolean | **getAllowUserInteraction**()<br>Liefert den Wert des allowUserInteraction Feldes. |
| Object | **getContent**()<br>Liefert den Inhalt dieser URL Verbindung. |
| String | **getContentEncoding**()<br>Liefert den Wert des content-encoding Header Feldes. |
| int | **getContentLength**()<br>Liefert den Wert des content-length Header Feldes. |
| String | **getContentType**()<br>Liefert den Wert des content-type Header Feldes. |
| long | **getDate**()<br>Liefert den Wert des date Header Feldes. |
| static boolean | **getDefaultAllowUserInteraction**()<br>Liefert den Defaultwert des allowUserInteraction Feldes. |
| static String | **getDefaultRequestProperty**(String key)<br>Liefert den Wert des Feldes Default Request Property. |
| boolean | **getDefaultUseCaches**()<br>Liefert der Defaultwert des useCaches Flags. |
| boolean | **getDoInput**()<br>Liefert den Wert des doInput Flags. |
| boolean | **getDoOutput**()<br>Liefert den Wert des doOutput Flags. |
| long | **getExpiration**()<br>Liefert den Wert des expires Header Feldes. |
| static FileNameMap | **getFileNameMap**()<br>Liefert das FileNameMap Feld. |
| String | **getHeaderField**(int n)<br>Liefert den Wert des $n^{th}$ Header Feldes. |
| String | **getHeaderField**(String name)<br>Liefert den Namen des spezifizierten Header Feldes. |
| long | **getHeaderFieldDate**(String name, long Default)<br>Liefert den Wert des Datum Feldes. |
| int | **getHeaderFieldInt**(String name, int Default)<br>Liefert des benannten Header Feldes als eine Nummer. |
| String | **getHeaderFieldKey**(int n)<br>Liefert den Schlüssel für das $n^{th}$ Header Feld. |
| long | **getIfModifiedSince**()<br>Liefert den Wert des ifModifiedSince Feldes. |
| InputStream | **getInputStream**()<br>Liefert einen Input Stream , der von dieser offenen Verbindung liest. |
| long | **getLastModified**()<br>Liefert den Wert des last-modified Header Feldes. |
| OutputStream | **getOutputStream**()<br>Liefert einen Output Stream, der in die offene Verbindung schreibt. |
| Permission | **getPermission**()<br>Liefert ein permission Objekt, welches beschreibt, welche Privilegien nötig sind, um mit dem Objekt eine Verbindung aufbauen zu dürfen. |
| String | **getRequestProperty**(String key)<br>Liefert den Wert der benannten Eigenschaft für diese Verbindung. |

| | |
|---|---|
| URL | **getURL**()<br>Liefert den Wert des URL Feldes dieser Verbindung. |
| boolean | **getUseCaches**()<br>Liefert den Wert des useCaches feldes dieser Verbindung. |
| protected static String | **guessContentTypeFromName**(String fname)<br>versucht den Content Type eines Objektes zu bestimmen, basierend auf den spezifizierten "file" Komponenten der URL. |
| static String | **guessContentTypeFromStream**(InputStream is)<br>Versucht den Typus des Eingabestromes auf Grund des ersten Zeichens des Eingabestromes zu bestimmen. |
| void | **setAllowUserInteraction**(boolean allowuserinteraction)<br>Setzt den Wert des allowUserInteraction Feldes dieser URLConnection. |
| static void | **setContentHandlerFactory**(ContentHandlerFactory fac)<br>Setzt die ContentHandlerFactory einer Applikation. |
| static void | **setDefaultAllowUserInteraction**(boolean defaultallowuserinteraction)<br>Setzt den Wert des allowUserInteraction Feldes für alle weiteren URLConnection Objekte gleich einem spezifizierten Wert. |
| static void | **setDefaultRequestProperty**(String key, String value)<br>Setzt den Defaultwert einer Default Request Eigenschaft. |
| void | **setDefaultUseCaches**(boolean defaultusecaches)<br>Setzt den Defaultwert des useCaches Feldes. |
| void | **setDoInput**(boolean doinput)<br>Setzt das doInput Feld für diese URLConnection. |
| void | **setDoOutput**(boolean dooutput)<br>Setzt den Wert des doOutput field für diese URLConnection. |
| static void | **setFileNameMap**(FileNameMap map)<br>Setzt den FileNameMap (Zuordnung eines MIME Types zu einem Dateinamen). |
| void | **setIfModifiedSince**(long ifmodifiedsince)<br>Setzt den Wert des ifModifiedSince Feldes dieser URLConnection. |
| void | **setRequestProperty**(String key, String value)<br>Setzt Request Eigenschaften. |
| void | **setUseCaches**(boolean usecaches)<br>Setzt den Wert des useCaches Feldes dieser URLConnection. |
| String | **toString**()<br>Liefert eine Zeichenketten-Darstellung dieser Verbindung. |

Nach dieser Übersicht schauen wir uns einige der Methoden und den Konstruktor genauer ein.

Starten wir diesmal mit einem Beispiel:

### *10.5. Einführendes URLConnection Beispiel*

```
//Titel:       öffnen einer URL Verbindung
```

```
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  einführendes Beispiel
package Beispiel1;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel1 {
  public static void main(String args[]) {
    System.out.println("Start Beispiel 10_1");
    URL u;
    URLConnection uc;

    String strURL="http://localhost";
//    int port=80;

    if (args.length > 0) {
      strURL = args[0];
    } else System.out.println("Default URL="+strURL+";");

    try {
      u = new URL(strURL);
      try {
        uc = u.openConnection();
      }
      catch (IOException e) {
        System.err.println(e);
      }
    }
    catch (MalformedURLException e) {
      System.err.println(e);
    }
    System.out.println("Ende Beispiel 10_1");
  }
}
```

#### 10.5.1. Ausgabe

Die Ausgabe hängt von Ihrer Eingabe (Host, Port) ab. Hier die Default- Ausgabe:

```
Start Beispiel 10_1
Default URL=http://localhost;
Ende Beispiel 10_1
```

### *10.6. Methoden, zum Lesen und Schreiben von Daten von/ zu einem Server*

#### 10.6.1. public abstract void connect() throws IOException

Diese Methode ist eine abstrakte Methode, muss also implementiert werden. Die Methode öffnet eine Verbindung zu einer URL.

Eine Möglichkeit wäre, eine FileURLConnection Klasse zu definieren, welche die URL in einen Dateinamen umwandelt. Diese könnte dann mit Hilfe eines Stromes gelesen werden.

### 10.6.2. public InputStream getInputStream()

Diese Methode ist allgemeiner als die getContent() Methode der URL und URLConnection Klasse. Der Vorteil ist der, dass damit auch MIME Typen gelesen werden können, die nicht plain Text sind.

Schauen wir uns ein Beispiel an, mit dem eine Webseite gelesen und herunter geladen werden kann.

#### 10.6.2.1. Programmbeispiel

```
//Titel:       public InputStream getInputStream()
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_2:
//lesen und herunter laden einer HTML Seite
package Beispiel10_2;
import java.net.*;
import java.io.*;

public class KlasseBeispiel10_2 {
  public static void main (String args[]) {

    System.out.println("Start Beispiel 10_2");
    String strZeile;
    URL u;
    URLConnection uc;

    if  (args.length > 0) {
      try {
        u = new URL(args[0]);
        try {
          uc = u.openConnection();
          DataInputStream disHTML = new DataInputStream(uc.getInputStream());
          try {
            while ((strZeile = disHTML.readLine()) != null) {
              System.out.println(strZeile);
            } // while
          }  // try
          catch (Exception e) {
            System.err.println(e);
          }
        } // try
        catch (Exception e) {
```

```
        System.err.println(e);
       }

     } // try
     catch (MalformedURLException e) {
       System.err.println(args[0] + " ist keine korrekte URL");
       System.err.println(e);
     }
   } //  if
   System.out.println("Ende Beispiel 10_2");
  } // main
}
```

#### 10.6.2.2. Ausgabe

Diese hängt davon ab, welchen Parameter Sie eingegeben haben.
Hier ein Beispiel (JavaScript, welches dem Benutzer, der versucht den Quellcode anzusehen, eine irritierende Meldung auf den Bildschirm schreibt):

```
Start Beispiel 10_2
<!-- <script LANGUAGE="JavaScript">  //-->
<!-- Begin
function right(e) {
        var msg = "Sorry, the right hand mouse click is not allowed on this page!";
        if (navigator.appName == 'Netscape' && e.which == 3) {
                alert(msg);  // Delete this line to disable but not alert user
                return false;
        }
        else
        if (navigator.appName == 'Microsoft Internet Explorer' && event.button==2) {
                alert(msg); // Delete this line to disable but not alert user
                return false;
        }
        return true;
}
document.onmousedown = right;
// End -->
<!-- </script>          //-->
Ende Beispiel 10_2
```

### 10.6.3. public OutputStream getOutputStream()

Manchmal benötigt man einen Ausgabestrom, über den man Daten an die URL senden kann. Dazu würde man bei einem HTTP basierten Server die POST Methode implementieren müssen, oder die spezielle HttpURLConnection Klasse einsetzen.

#### 10.6.3.1. Programm Fragment

```
try {
        URL u = new URL("http://www.meinHost.com/cgi-bin/poster.cgi");
        URLConnection uc = u.openConnection();
        uc.setOutput(true);

        DataOutputStream dos = new DataOutputStream( uc.getOutputStream() );
        dos.writeBytes("Jetzt schicke ich Daten");
        dos.close();
} catch (Exception e)  {
```

```
        System.err.println(e);
}
```

## *10.7. Parsen des Headers*

Jetzt wollen wir uns den Header genauer analysieren. Im Falle des HTTP Protokolls liefert der Header sehr viel Informationen. Im Header kann auch Information über die Verschlüsselung, Datum und Zeit, die Länge des Dateninhaltes, allfällige Gültigkeitsdaten und Modifikationsdaten.

Die Methode `getContent`() funktioniert nicht für alle Protokolle. HTTP ist einfach zu beschreiben, daher auch die detaillierte Auswertbarkeit.

Falls die Methode die Inhaltsstruktur nicht erkennen kann, dann wird eine `UnknownServiceException` geworfen.

### 10.7.1. public String getContentType()

Diese Methode liefert den MIME Typus der Daten. Es liegt am Web Server, den korrekten MIME Header zu senden, inklusive Inhaltstypus / Content Type. Falls der Inhaltstypus nicht bekannt ist, wird keine Ausnahme geworfen. text/html wird bei HTTP der übliche Inhaltstypus sein. Daneben werden image/gif und image/jpeg Bilder angezeigt.

### 10.7.2. public int getContentLength()

Diese Methode zeigt an, wie viele Bytes vorhanden sind. Einige Server senden die Länge nur, falls eine binäre Datei geschickt wird; bei Text Dateien fehlt die Angabe.

Im letzteren Fall liefert getContent() den Wert -1.

### 10.7.3. public String getContentEncoding()

Diese Methode liefert eine Zeichenkette, die erklärt, wie die Daten verschlüsselt sind. Falls die Daten nicht verschlüsselt werden, dann liefert die Methode den Wert null.

### 10.7.4. public long getDate()

Diese Methode liefert einen long Integer, in dem steht, wann das Dokument gesendet wurde. Die Zeit wird ab 12:00 A.M. GMT 1. Januar 1970 gemessen.

### 10.7.5. public long getExpiration()

Falls ein Dokument ein Gültigkeitsdatum besitzt, typischerweise Cookies und Cache Dateien, dann liefert diese Methode das Datum, wie oben, seit dem 1. Januar 1970.

### 10.7.6. public long getLastModified()

Die Methode liefert, wie der Name sagt, das Datum der letzten Modifikation. Das Datum wird in Sekunden seit dem 1. Januar 1970 angegeben.

Schauen wir uns ein Beispiel an, welches die obigen Methoden testet:

### 10.7.7. Programmbeispiel

```
//Titel:       Parsen des MIME Headers
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_3
//getDate
//getExpiration
//...
package Beispiel10_3;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_3 {
  public static void main(String args[]) {
    System.out.println("Start Beispiel 10_3");

    URL u;
    URLConnection uc;
    String header;

    for (int i=0; i < args.length; i++) {
      try {
        u = new URL(args[i]);
        uc = u.openConnection();
        System.out.println("URL : "+u);
        System.out.println("Connection : "+uc+"\n ----------");

        for (int j = 1; ; j++) {
          header = uc.getHeaderField(j);
          if (header == null) break;
          System.out.println(uc.getHeaderFieldKey(j) + " " + header);
        }  // for
      }  // try
```

```
    catch (MalformedURLException e) {
      System.err.println(args[i] + " ist nicht als URL interpretierbar.");
    }
    catch (IOException e) {
      System.err.println(e);
    }
   }  // for
   System.out.println("Ende Beispiel 10_3");
  }  // main
}
```

#### 10.7.8. Ausgabe

```
Start Beispiel 10_3
URL : http://localhost/www.swarovski.com/crystal/gifts/index.html
Connection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/www.swarovski.com/crystal/gifts/index.html
 ----------
Server HTTPS/0.991
Allow GET HEAD POST
MIME-version 1.0
Content-type text/html
Date Wednesday, 8-Sep-99 15:50:43 GMT
Last-modified Saturday, 9-Oct-99 8:5:35 GMT
Content-length 984
Ende Beispiel 10_3
```

### *10.8. Lesen von beliebigen MIME Header Feldern*

Die Anzahl Header Felder in einem MIME Header ist nicht vorgegeben. Im Prinzip kann man also beliebig viele Header Felder definieren und jeweils mitsenden; das dürfte aber kaum sinnvoll sein.

Wenn wir Felder abfragen, die es nicht gibt, dann liefert die Methode den Wert null.

#### 10.8.1. public String getHeaderField(String name)

Diese Methode liefert, falls vorhanden, Werte eines benannten MIME Header Feldes.
Die Namen der Header Felder sind nicht case sensitiv; es spielt also keine Rolle, ob die Namen gross oder klein geschrieben werden.

#### 10.8.2. public String getHeaderFieldKey(int i)

Diese Methode liefert den Namen des Header Feldes, einfach durch nummeriert mit einer ganzen Zahl (i).

#### 10.8.3. public String getHeaderField(int i)

Diese Methode liefert das i-te Feld, beginnend mit 0.

Schauen wir uns ein Programm Beispiel an:

### 10.8.4. Programmbeispiel

```
//Titel:       Ausgabe des MIME Headers
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_4
package Beispiel10_4;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_4 {
  public static void main(String args[]) {
    System.out.println("Start Beispiel 10_4");
    URL u;
    URLConnection uc;
    String header;

    for (int i=0; i < args.length; i++) {
      try {
        u = new URL(args[i]);
        uc = u.openConnection();
        System.out.println("URL : "+u);
        System.out.println("Connection : "+uc+"\n -----------");

        for (int j = 1; ; j++) {
          header = uc.getHeaderField(j);
          if (header == null) break;
          System.out.println(uc.getHeaderFieldKey(j) + " " + header);
        }  // for
      }  // try
      catch (MalformedURLException e) {
        System.err.println(args[i] + " kann nicht als URL interpretiert werden.");
      }
      catch (IOException e) {
        System.err.println(e);
      }
    }  // for
    System.out.println("Ende Beispiel 10_4");
  }  // main
}
```

### 10.8.5. Ausgabe

```
Start Beispiel 10_4
URL : http://localhost/www.swarovski.com/crystal/gifts/html/index.html
Connection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/www.swarovski.com/crystal/gifts/html/index.html
 -----------
Server HTTPS/0.991
Allow GET HEAD POST
MIME-version 1.0
Content-type text/html
```

Ende Beispiel 10_4

## *10.9. Setzen von Eigenschaften*

Die set Methoden (setRequestProperty(), ...) bewirken nichts. Sie müssen erst vom Programmierer implementiert werden. Per Default geschieht nichts.

## *10.10. Der Konstruktor*

### 10.10.1. protected URLConnection(URL u)

Der Konstruktor zeichnet sich durch zwei Sachen aus:

1. er ist protected
2. er hat nur ein Argument, die URL, mit der die Verbindung aufgebaut werden soll.
3. es gibt *keinen* Default- Konstruktor!

Alle anderen Werte werden über Defaultwerte gesetzt oder aber mit Hilfe der Methoden, auf die wir gleich zurück kommen.

Die Tatsache, dass der Konstruktor protected ist, hat zur Folge, dass nur Objekte innerhalb des java.net Packages ein URLConnection Objekt kreieren können.

Da die Klasse URLConnection eine abstrakte Klasse ist, kann man den Konstruktor nur aus einer Unterklasse aufrufen.

Da es keinen Default Konstruktor gibt, muss im Konstruktor der Unterklasse explizit der Konstruktor der URLConnection Klasse aufgerufen werden:

#### 10.10.1.1. Programm Fragment

In etwa so könnte das aussehen:

```
...
// Konstruktor meiner URLConnection Klasse
..
meineURLConnectionKlasse(URL u) {
        super(u);
        // ohne diesen Aufruf wird versucht den Konstruktor URLConnection() aufzurufen.
        // da es diesen Konstruktor nicht gibt, wird ein Compiler Fehler generiert
        ...
}
...
```

### *10.11. Datenfelder und dazugehörende Methoden*

Insgesamt gibt es dreizehn Datenfelder in java.net.URLConnection.
Sieben davon sind static, also Klassenvariablen und definieren Defaultwerte der URLConnection Instanz.
Die anderen sechs definieren den Zustand der URLConnection.
Einige davon besitzen get und set Methoden, können also verändert werden.

### 10.11.1. protected URL url;

Dieses Feld beschreibt den URL dieser URLConnection. Das Feld wird beim Konstruieren eines URLConnection Objektes gesetzt.

Die Methode getURL() liefert den Wert des Feldes.

#### 10.11.1.1. Programmbeispiel

```
//Titel:       protected URL url;  public void getURL()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_5
//Abfrage des URL Feldes von URLConnection
package Beispiel10_5;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_5 {
  public static void main(String args[]) {
    System.out.println("Start Beispiel 10_5");
    URL u;
    URLConnection uc;

    try {
      u = new URL("http://localhost/");
      try {
        uc = u.openConnection();
        System.out.println(uc.getURL());
      }
      catch (IOException e) {
        System.err.println(e);
      }
    }
    catch (MalformedURLException e) {
      System.err.println(e);
    }
    System.out.println("Ende Beispiel 10_5");
  }
}
```

#### 10.11.1.2. Ausgabe

```
Start Beispiel 10_5
http://localhost/
Ende Beispiel 10_5
```

### 10.11.2. protected boolean allowUserInteraction

Falls das Objekt URLConnection mit dem Benutzer kommunizieren muss, zum Beispiel zur Eingabe eines Passwortes, dann muss das Datenfeld auf true gesetzt sein.

Da dieses Feld protected ist, ist das Setzen und Lesen nicht ohne weiteres möglich. Schauen wir uns ein Beispiel an:

#### 10.11.2.1. Programmbeispiel

```
//Titel:       protected boolean allowUserInteraction()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_6 :
//protected boolean allowUserInteraction;
//public void setAllowUserInteraction(boolean allowuserinteraction)
package Beispiel10_6;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_6 {
  public static void main(String[] args) {
    System.out.println("Start Beispiel 10_6");
    URL u;
    URLConnection uc;
    meinHttpHandler handler;
    try {
      u = new URL("http://localhost");
      handler = new meinHttpHandler();
      uc = handler.openConnection(u);
      System.out.println("URLConnection : "+uc);
      System.out.println("UserInteraction Standard: "+uc.getAllowUserInteraction());
      if (!uc.getAllowUserInteraction()) {
        uc.setAllowUserInteraction(true);
      }
      System.out.println("UserInteraction neu gesetzt: "+uc.getAllowUserInteraction());
    }
    catch (MalformedURLException e) {
      System.err.println(e);
    }
    catch (IOException e) {
      System.err.println(e);
    }
    System.out.println("Ende Beispiel 10_6");
  }
}
```

```
class meinHttpHandler extends sun.net.www.protocol.http.Handler {
  public URLConnection openConnection(URL u) throws IOException {
    return super.openConnection(u);
  }
}
```

#### 10.11.2.2. Ausgabe

```
Start Beispiel 10_6
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/
UserInteraction Standard: false
UserInteraction neu gesetzt: true
Ende Beispiel 10_6
```

### 10.11.3. private static boolean defaultAllowUserInteraction;

Das Default Verhalten können wir analog wie im obigen Beispiel setzen.

#### 10.11.3.1. Programmbeispiel

```
//Titel:       private static boolean defaultAllowUserInteraction
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  setDefaultAllowUserInteraction(boolean b)
package Beispiel10_7;
import java.net.*;

public class KlasseBeispiel10_7 {
  public static void main(String args[]) {
    System.out.println("Start Beispiel10_7");
    System.out.println("DefaultAllowUserInteraction :
"+URLConnection.getDefaultAllowUserInteraction());
    if(!URLConnection.getDefaultAllowUserInteraction()) {
      URLConnection.setDefaultAllowUserInteraction(true);
    }
    System.out.println("DefaultAllowUserInteraction :
"+URLConnection.getDefaultAllowUserInteraction());
    System.out.println("Ende Beispiel 10_7");
  }
}
```

#### 10.11.3.2. Ausgabe

```
Start Beispiel10_7
DefaultAllowUserInteraction : false
DefaultAllowUserInteraction : ftrue
Ende Beispiel 10_7
```

### 10.11.4. protected boolean doInput;

Jeder Web Verbindung, die den GET Befehl verwendet, erlaubt eine Eingabe.
Der POST Befehl natürlich nicht.

#### 10.11.4.1. Programmbeispiel

```
//Titel:       protected boolean doInput
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_8
package Beispiel10_8;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_8 {
  public static void main(String[] args) {
    System.out.println("Start Beispiel 10_8");
    URL u;
    URLConnection uc;

    try {
      u = new URL("http://localhost");
      try {
        uc = u.openConnection();
        System.out.println("URLConnection : "+uc);
        System.out.println("getDoInput():"+uc.getDoInput());
        uc.setDoInput(false);
        System.out.println("getDoInput():"+uc.getDoInput());
        if (!uc.getDoInput()) {
          uc.setDoInput(true);
        }
        System.out.println("getDoInput():"+uc.getDoInput());
      }
      catch (IOException e) {
        System.err.println(e);
      }
    }
    catch (MalformedURLException e) {
      System.err.println(e);
    }
    System.out.println("Ende Beispiel 10_8");
  }
}
```

#### 10.11.4.2. Ausgabe

```
Start Beispiel 10_8
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/
getDoInput():true
getDoInput():false
getDoInput():true
Ende Beispiel 10_8
```

### 10.11.5. protected boolean doOutput

Die POST Methode kann nur eingesetzt werden, wenn das obige Flag auf true gesetzt ist.

#### 10.11.5.1. Programmbeispiel

```
//Titel:       doOutput()
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_9
//doOutput()
package Beispiel10_9;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_9 {
  public static void main(String[] args) {
    System.out.println("Start Beispiel 10_9");
    URL u;
    URLConnection uc;

    try {
      u = new URL("http://localhost");
      try {
        uc = u.openConnection();
        System.out.println("URLConnection : "+uc);
        System.out.println("getDoOutput() : "+uc.getDoOutput());
        if (!uc.getDoOutput()) {
          uc.setDoOutput(true);
        }
        System.out.println("getDoOutput() : "+uc.getDoOutput());
      }
      catch (IOException e) {
        System.err.println(e);
      }
    }
    catch (MalformedURLException e) {
      System.err.println(e);
    }
    System.out.println("Ende Beispiel 10_9");
  }
```

}

#### 10.11.5.2. Ausgabe

```
Start Beispiel 10_9
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/
getDoOutput() : false
getDoOutput() : true
Ende Beispiel 10_9
```

### 10.11.6. protected long ifModifiedSince

Der Zeitpunkt, zu dem zuletzt modifiziert wurde, spielt bei vielen Servern, speziell mit Cache, eine grosse Rolle.

#### 10.11.6.1. Programmbeispiel

```
//Titel:       protected long ifModifiedSince
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_10
package Beispiel10_10;
import java.net.*;
import java.io.*;
import java.util.Date;

public class KlasseBeispiel10_10 {
  public static void main (String[] args) {
    System.out.println("Start Beispiel 10_10");
    URL u;
    URLConnection uc;
    String thisLine;
    // ein Date Objekt mit aktuellem Datum und Zeit
    Date today = new Date();

    try {
      u = new URL("http://localhost/index.html");
      uc = u.openConnection();
      System.out.println("Die Datei soll gelesen werden, falls sie seit "
        + new Date(uc.getIfModifiedSince()).toLocaleString()+" modifiziert wurde.");
      uc.setIfModifiedSince(Date.UTC(today.getYear(), today.getMonth(),
        today.getDate() - 1, today.getHours(), today.getMinutes(),
        today.getSeconds()));
      System.out.println("Die Datei soll gelesen werden, falls sie seit "
        + new Date(uc.getIfModifiedSince()).toLocaleString()+" modifiziert wurde.");
    }
    catch (Exception e) {
      System.err.println(e);
    }
```

```
    System.out.println("Ende Beispiel 10_10");
  }
}
```

#### 10.11.6.2. Ausgabe

```
Start Beispiel 10_10
Die Datei soll gelesen werden, falls sie seit 01.01.1970 01:00:00 modifiziert wurde.
Die Datei soll gelesen werden, falls sie seit 18.11.1999 20:38:12 modifiziert wurde.
Ende Beispiel 10_10
```

### 10.11.7. protected boolean useCaches

Web Browser verwenden, je nach Einstellung, Daten wann immer möglich aus dem lokalen Cache. Mit den Methoden zum useCache Parameter kann man abfragen, ob ein URL aus einem Cache gelesen werden soll.

#### 10.11.7.1. Programmbeispiel

```
//Titel:       protected boolean useCache
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_11
package Beispiel10_11;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_11 {
  public static void main(String[] args) {
    System.out.println("Start Beispiel 10_11");
    URL u;
    URLConnection uc;

    try {
      u = new URL("http://localhost");
      try {
        uc = u.openConnection();
        System.out.println("URLConnection : "+uc);
        System.out.println("getUseCaches : "+uc.getUseCaches());
        if (uc.getUseCaches()) {
          uc.setUseCaches(false);
        }
        System.out.println("getUseCaches : "+uc.getUseCaches());
      }
      catch (IOException e) {
        System.err.println(e);
      }
    }
    catch (MalformedURLException e) {
      System.err.println(e);
```

```
    }
    System.out.println("Ende Beispiel 10_11");
  }
}
```

#### 10.11.7.2. Ausgabe

```
Start Beispiel 10_11
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/
getUseCaches : true
getUseCaches : false
Ende Beispiel 10_11
```

### 10.11.8. protected static boolean defaultUseCaches

Jetzt fragen wir noch den Defaultwert ab:

#### 10.11.8.1. Programmbeispiel

```
//Titel:       protected static boolean defaultUseCaches
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_12
package Beispiel10_12;
import java.net.*;
import java.io.IOException;

public class KlasseBeispiel10_12 {
  public static void main(String[] args) {
    System.out.println("Start Beispiel 10_12");
    try {
      URL u = new URL("http://localhost/");
      URLConnection uc = u.openConnection();
      System.out.println("URLConnection : "+uc);
      System.out.println("getDefaultUseCaches():"+uc.getDefaultUseCaches());
      if (uc.getDefaultUseCaches()) {
        uc.setDefaultUseCaches(false);
      }
      System.out.println("getDefaultUseCaches():"+uc.getDefaultUseCaches());
    }
    catch (IOException e) {
      System.err.println(e);
    }
    System.out.println("Ende Beispiel 10_12");
  }
}
```

#### 10.11.8.2. Ausgabe

```
Start Beispiel 10_12
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/
getDefaultUseCaches():true
getDefaultUseCaches():false
```

Ende Beispiel 10_12

## *10.12. Bestimmen des MIME Types*

In Java steht eine Methode zur Verfügung `protected static String guessContentTypeFromName(String name)`, mit der im Prinzip den MIME Type feststellen könnte, allerdings eben nur im Prinzip.

Java hat dafür eine interne Liste zur Verfügung.

### 10.12.1. `guessContentTypeFromName(..)` Beispiele

.pdf -> application/pdf
.gif -> image/gif
.au -> audio/basic

Noch heisser wäre das Bestimmen des MIME Typs auf Grund des Streams.
In Java steht dafür die Methode `static protected String guessContentTypeFromStream(InputStream is)` zur Verfügung. Dabei werden die ersten 6 Bytes analysiert.

### 10.12.2. `guessContentTypeFromStream(..)` Beispiele

$def -> image/x-bitmap
!XPM2 -> image/x-pixmap
<! -> text/html

Die Mappings sind Case sensitiv.

## *10.13. Einige komplexere Beispiele*

Wir haben bereits gesehen, dass man mit URLConnection binäre Dateien herunter laden kann. Zudem kann man mit URLConnection auf bidirektional kommunizieren, um zum Beispiel die POST oder PUT Methode zusammen mit CGI Skripts einzusetzen.

### 10.13.1. Herunterladen binärer Dateien von einer HTTP Verbindung

Web Server schicken oft eine Datei zum Client ohne sauber abzuschliessen, das heisst, das EOF fehlt. Zuverlässiger kann man Daten mit Hilfe eines `URLConnection` Objektes herunter laden, indem man mit `getContentLength`() die Länge der Datei in Bytes bestimmt.

#### 10.13.1.1. Programm Code

```
//Titel:        Binäre Dateien herunterladen
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_13
//URLConnection
package Beispiel10_13;
import java.net.*;
import java.io.*;

public class KlasseBeispiel10_13 {
  public static void main (String args[]) {
    System.out.println("Start Beispiel 10_13");
    for (int i = 0; i < args.length; i++) {
      try {
        URL root = new URL(args[i]);
        System.out.println("Datei : "+root);
        saveBinaryFile(root);
      }
      catch (MalformedURLException e) {
        System.err.println(args[i] + " ist kein korrekter URL.");
      }
    } // end for
    System.out.println("Ende Beispiel 10_13");
  } // end main


  public static void saveBinaryFile(URL u) {

    int bfr = 128;
```

```
    try {
      URLConnection uc = u.openConnection();
      String ct = uc.getContentType();
      int cl = uc.getContentLength();
      if (ct.startsWith("text/") || cl == -1 ) {
        System.err.println("Das ist keine binäre Datei.");
        return;
      }
      System.out.println("URLConnection : "+uc);
      System.out.println("getContenType : "+ct);
      System.out.println("getContentLength : "+cl);
      InputStream is = uc.getInputStream();
      byte[] b = new byte[cl+bfr];  // +bfr : Byte Array gross genug wählen
      int bytesread = 0;
      int offset = 0;
      while (bytesread >= 0) {
         bytesread = is.read(b, offset, bfr); //bfr = 128;
         // bytesread = -1 bei EOF
         if (bytesread == -1) break;
         offset = offset + bytesread;
      }
      System.out.println("Die Datei ist im Byte Array b gespeichert");
      if (offset != cl) {
        System.err.println("Fehler: es wurden nur " + offset + " Bytes gelesen");
        System.err.println("Erwartet wurden " + cl + " Bytes");
      }
      String strFile = u.getFile();
      strFile = strFile.substring(strFile.lastIndexOf('/') + 1);
      System.out.println("Die Datei heisst "+strFile);
      FileOutputStream fout = new FileOutputStream(strFile);
      fout.write(b);
      System.out.println("Die Datei "+strFile+" wurde lokal gespeichert!");
    }  // end try
    catch (Exception e) {
     System.err.println(e);
    }
    return;
  } // end
}
```

#### 10.13.1.2. Ausgabe

```
Start Beispiel 10_13
Datei : http://localhost/www.swarovski.com/products/7478-000-001.JPG
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://localhost/www.swarovski.com/products/7478-000-
001.JPG
getContenType : image/jpeg
getContentLength : 12474
Die Datei ist im Byte Array b gespeichert
Die Datei heisst 7478-000-001.JPG
Die Datei 7478-000-001.JPG wurde lokal gespeichert!
Ende Beispiel 10_13
```

### 10.13.2. Herunterladen und starten eines Applets

Das Problem, ein Applet herunter zu laden und zu starten, in einem selbst entwickelten Applet Viewer, ist komplexer als das einfache Lesen und kopieren einer binären Datei.

Um ein Class File zu laden, muss man einen ClassLoader definieren. Die ClassLoader Klasse gehört zum Package `java.lang.ClassLoader` und ist abstrakt, muss also implementiert werden.

#### 10.13.2.1. Java.lang Class ClassLoader

```
java.lang.Object
   |
   +--java.lang.ClassLoader
```

**Direkt Bekannte Unterklassen:**
SecureClassLoader

public abstract class **ClassLoader**
extends Object

Die Klasse `ClassLoader` ist eine abstrakte Klasse. Ein class loader ist ein Objekt, welches für das Laden von Klassen zuständig ist. Bei gegebenem Klassennamen versucht das Class Loader Objekt die Daten zu finden, welche die Klasse definieren (Byte Code).
Typischerweise wird der Klassenname in einen Dateinamen umgewandelt und dann die Klassendatei vom Dateisystem gelesen.
Jedes `Class` Objekt enthält eine `Referenz` zum `ClassLoader` der das Objekt definiert hat.
Klassen Objekte für Array Klassen werden nicht von einem Class Loader kreiert, sondern vom Java Laufzeitsystem, dann wenn sie benötigt werden. Der Class Loader für eine Array Klasse, `Class.getClassLoader()` , ist der selbe wie der Class Loader für seine Elemente; falls die Elemente primitive Datentypen sind, dann besitzt die Klasse / das Objekt keinen Class Loader.
Applikationen implementieren Unterklassen der `ClassLoader` Klasse, um optimierte Möglichkeiten zum Laden von Klassen zu schaffen, welche die Möglichkeiten der Java Virtual Machine dynamisch erweitern.
Class Loaders werden typischerweise von Security Managers eingesetzt, um Security Domains zu definieren.
Die `ClassLoader` Klasse verwendet ein Delegation Modell zum Suchen von Klassen und Resourcen. Jede Instanz des `ClassLoader` ist mit einem Parent Class Loader verknüpft. Falls der Class Loader eine Klasse oder eine Resource suchen muss, dann delegiert die `ClassLoader` Instanz die Suche für die Klasse oder Resource an seinen Parent Class Loader bevor die Klasse oder Resource selber gesucht wird. Die Virtuelle Maschine besitzt einen speziellen Class Loader, den bootstrap Class Loader, der als einziger Class Loader kein Elternteil besitzt und als Parent für eine Class Loader Instanz dienen kann.
Normalerweise lädt die Java Virtual Machine Klassen vom lokalen Dateisystem, zum Beispiel mit Hilfe der CLASSPATH Variable.

Einige Klassen stammen eventuell nicht aus einem lokalen, sondern aus einem Netzwerk System. Die Methode `defineClass` konvertiert ein Array von Bytes in eine Klasseninstanz.

### 10.13.2.2.Programm Fragment

Eine Applikation kreiert einen Netzwerk Class Loader ,um Klassendateien von einem Server herunter zu laden:

```
ClassLoader loader = new NetworkClassLoader(host, port);
Object main = loader.loadClass("Main", true).newInstance();
    . . .


  class NetworkClassLoader extends ClassLoader {
      String host;
      int port;

      public Class findClass(String name) {
          byte[] b = loadClassData(name);
          return defineClass(name, b, 0, b.length);
      }

      private byte[] loadClassData(String name) {
          // load the class data from the connection
           . . .
      }
  }
```

### 10.13.2.3.Konstruktor Übersicht

| | |
|---|---|
| protected | **ClassLoader**()<br>Kreiert einen neuen Class Loader, wobei der ClassLoader aus getSystemClassLoader() als Parent Class Loader verwendet wird. |
| protected | **ClassLoader**(ClassLoader parent)<br>Kreiert einen Class Loader mit dem parent Class Loader für Delegationen. |

### 10.13.2.4.Methoden Übersicht

| | |
|---|---|
| protected Class | **defineClass**(byte[] b, int off, int len)<br>**Veraltet.** *Neu: defineClass(java.lang.String, byte[], int, int)* |
| protected Class | **defineClass**(String name, byte[] b, int off, int len)<br>Konvertiert ein Byte Array in eine Instanz der Klasse Class. |
| protected Class | **defineClass**(String name, byte[] b, int off, int len, ProtectionDomain protectionDomain)<br>Konvertiert ein Byte Array in eine Instanz der Klasse Class, mit einer optionalen ProtectionDomain. |
| protected Package | **definePackage**(String name, String specTitle, String specVersion, String specVendor, String implTitle, String implVersion, String implVendor, URL sealBase)<br>Definiert ein Package in diesem ClassLoader. |
| protected Class | **findClass**(String name)<br>Versucht die spezifizierte Klasse zu finden. |
| protected String | **findLibrary**(String libname)<br>Liefert den Pfadnamen der Bibliothek. |
| protected Class | **findLoadedClass**(String name)<br>Sucht die bereits geladene Klasse. |

| | |
|---|---|
| protected URL | **findResource**(String name)<br>Sucht die benannte Resource. |
| protected Enumeration | **findResources**(String name)<br>Liefert eine Auflistung aller URL's , mit der Resource name. |
| protected Class | **findSystemClass**(String name)<br>Sucht die Klasse name und lädt sie, falls nötig. |
| protected Package | **getPackage**(String name)<br>Liefert das Package, welches mit dem Loader definiert wurde |
| protected Package[] | **getPackages**()<br>Liefert alle Packages, die vom Loader definiert wurden. |
| ClassLoader | **getParent**()<br>Liefert den Parent Class Loader |
| URL | **getResource**(String name)<br>Sucht die Resource name. |
| InputStream | **getResourceAsStream**(String name)<br>Liefert einen Input Stream zum Lesen der spezifizierten Resource. |
| Enumeration | **getResources**(String name)<br>Sucht alle Resources name. |
| static ClassLoader | **getSystemClassLoader**()<br>System Class Loader. |
| static URL | **getSystemResource**(String name)<br>Sucht Klasse name im Systempfad (Pfad aus dem Klassen geladen werden) . |
| static InputStream | **getSystemResourceAsStream**(String name)<br>Liest Resourcen aus demSystempfad. |
| static Enumeration | **getSystemResources**(String name)<br>Sucht alle Resourcen name im Systempfad. |
| Class | **loadClass**(String name)<br>Laden der Klasse name. |
| protected Class | **loadClass**(String name, boolean resolve)<br>Laden der Klasse name mit Namensauflösung. |
| protected void | **resolveClass**(Class c)<br>Klasse c linken. |
| protected void | **setSigners**(Class c, Object[] signers)<br>Signers einer Klasse setzen (digitale Signatur). |

Schauen wir uns zuerst den Programm Code des ClassLoaders an.

### 10.13.2.5.Programmbeispiel : eigener ClassLoader

```
/**
```

```
//Titel:       ClassLoader
//Version:
//Copyright:    Copyright (c) 1999
//@author       J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_14  ClassLoader Klasse
*/
package Beispiel10_14;
import java.util.Hashtable;
import java.net.*;
import java.io.*;

public class KlasseBeispiel10_14 extends ClassLoader {
   Hashtable cache;
   /** die gelesenen Applets werden in eine Hashtabelle eingetragen
   *   um die Anzahl Lesebefehle zu reduzieren
   */
   URL u;

   public void URLClassLoader(URL u) {
     /** die URLClassLoader Klasse kreiert lediglich eine Hashtabelle
     */
     System.out.println("URLClassLoader Konstruktor");
     this.u = u;
     cache = new Hashtable();
   }
   public synchronized Class loadClass(String  name, boolean resolve)
    throws ClassNotFoundException {
     /** eine Klasse wird zuerst im Cache gesucht
     */
     Class c = (Class) cache.get(name);
     System.out.println("Laden der Klasse "+name);
     /** falls sie dort nicht ist, dann werden die System Klassen durchsucht
     */
     if (c == null) {
       try {
         System.out.println("Suchen der Klasse "+name+" im Systempfad.");
         c = findSystemClass(name);
         /** Class Loader findSystemClass Methode zum Suchen und Laden
         *   der Klasse name
         */
       }
       catch (ClassNotFoundException e) {
       }
     }

     // if the class still hasn't been found,
     // load it from the network
     if (c  ==  null) {
       byte b[] = loadClassData(name);
```

```
      c = defineClass(b, 0, b.length);
      cache.put(name, c);
    }
    if (resolve) {
      resolveClass(c);
    }

    return  c;

  }  // end loadClass

  private byte[] loadClassData(String name)
   throws ClassNotFoundException {
    System.out.println("Laden der Klasse "+name);
    byte[] b;
    InputStream theClass = null;
    int bfr = 128;

    try {

      URL classURL = new URL(u, name + ".class");

      URLConnection uc = classURL.openConnection();
      uc.setAllowUserInteraction(false);

      try {
        theClass = uc.getInputStream();
      }
      catch (NullPointerException e) {
        System.err.println(e);
        throw new ClassNotFoundException(name + " Input Stream Problem");
      }
      int cl = uc.getContentLength();
      /** Die Länge der .class Dateien wird vom Server oft übermittelt
      */
      if (cl == -1 ) {
        b = new byte[bfr * 17; // 16+1 : zur Sicherheit +1
      }
      else {
        b = new byte[cl+bfr]; // +bfr : zur Sicherheit, da die Länge oft falsch übertragen wird
      }

      int bytesread = 0;
      int offset = 0;
      System.out.println("Lesen der Datei in ein Byte Array");
      while (bytesread >= 0) {
        bytesread = theClass.read(b, offset, bfr);
        if (bytesread == -1) break;
        offset += bytesread;
        if (cl == -1 && offset == b.length) { // grow the array
```

```
          byte temp[] = new byte[offset * 2];
          System.arraycopy(b, 0, temp, 0, offset);
          b = temp;
        }
        else if (offset > b.length) {
          throw new ClassNotFoundException(name
           + " Fehler beim Lesen der Daten");
        }
      }
      System.out.println("Datei wurde in das Byte Array gelesen");
      /** Optimierung des Byte Arrays (Länge)
      */
      if (offset < b.length) {
        byte temp[] = new byte[offset];
        System.arraycopy(b, 0, temp, 0, offset);
        b = temp;
      }

      /** wurden alle Bytes empfangen?
      */
      if (cl != -1 && offset != cl) {
        throw new ClassNotFoundException("Es wurden nur " + offset +
          " Bytes von " + name + " empfangen."+
          "\n Erwartet wurden " + cl + " Bytes");
      }
    } // end try
    catch (Exception e) {
     throw new ClassNotFoundException(name + " " + e);
    }
    finally {
      try {
        if (theClass != null) theClass.close();
      }
      catch (IOException e) {
      }
    }
    return b;
  }  // end loadClassData
}  // end ClassLoader
```

Jetzt brauchen wir nur noch ein Rahmenprogramm:

#### 10.13.2.6. Programmbeispiel : Laden eines Applets aus einem URL Objekt

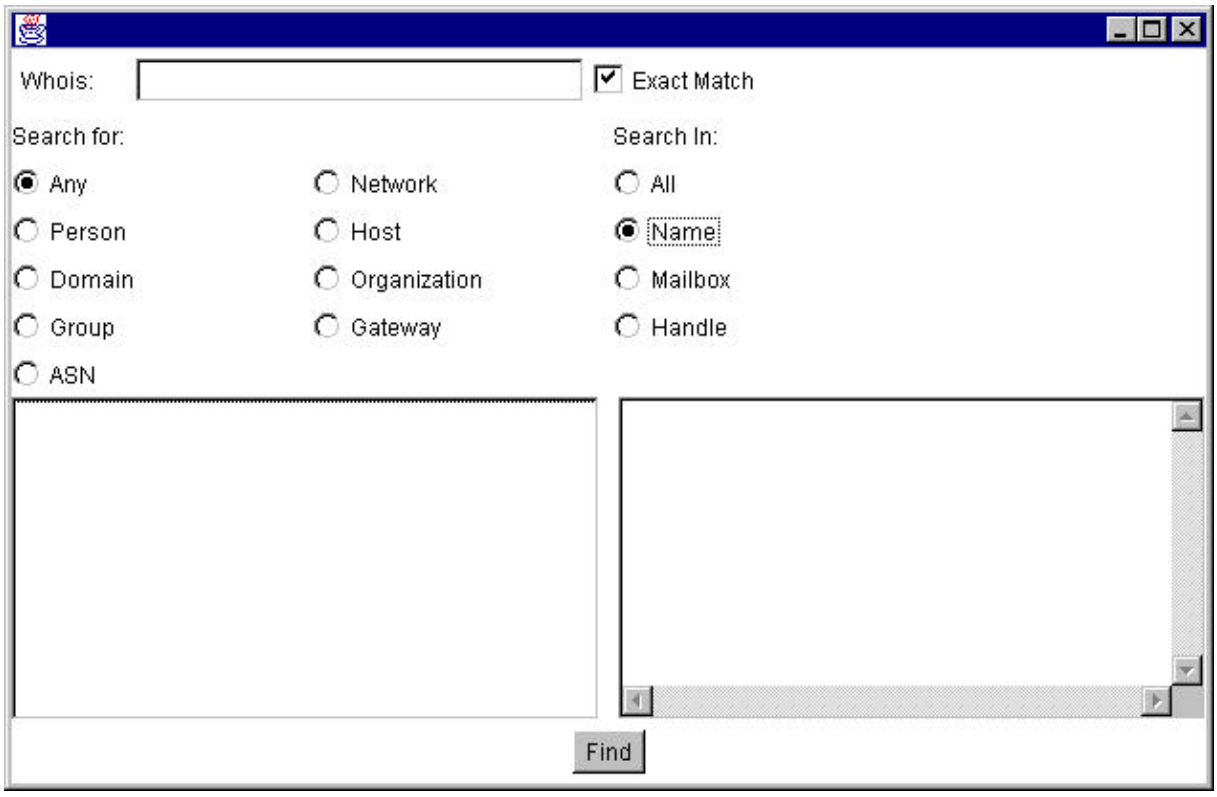
```
//Titel:       Laden eines Applets aus einem URL Objekt
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Einsatzbeispiel für einen ClassLoader
package Beispiel10_15;
import java.applet.*;
import java.awt.*;
import java.net.*;

public class KlasseBeispiel10_15 {
  public static void main(String args[]) {

    int x = 50;
    int y = 50;

    for (int i = 0; i < args.length; i++) {
      try {
        if (!args[i].endsWith(".class")) {
          System.err.println("Die Datei ist keine Byte Code Datei!");
          break;
        }

        URL u = new URL(args[i]);
           // URLClassLoader ist unser Beispiel 10_14
        URLClassLoader ucl = new URLClassLoader(u);

        // parsen des Class Namens aus dem URL
        String s = u.getFile();
        String classname = s.substring(s.lastIndexOf('/'),
         s.lastIndexOf(".class"));
        System.err.println(classname);
        Class AppletClass = ucl.loadClass(classname, true);
        Applet apl = (Applet) AppletClass.newInstance();
        /** vielleicht ist das Frame auch zu klein; dann sieht man nichts!
        */
        Frame f = new Frame();
        f.resize(200, 200);
        f.move(x, y);
        x += 50;
        y += 50;
        f.add("Center", apl);
        apl.init();
        apl.start();
```

```
      f.show();
    }  // end try
    catch (MalformedURLException e) {
      System.err.println(args[i] + " kann nicht als URL interpretiert werden.");
    }
    catch (Exception e) {
      System.err.println(e);
    }
  }  // end for
 }  // end main
}
```

#### 10.13.2.7. Ausgabe

So sieht die Ausgabe aus, falls das WhoIs Applet geladen wird. Hier erkennt man die Schwächen dieser Variante. Da die Dimensionen nicht abgefragt werden, wird das Applet nur in einem zu kleinen Frame angezeigt.
Es ist kaum zu erkennen, dass es sich um das WhoIs Applet handelt, würde es nicht explizit auf dem Fenster stehen.


Hier ein Screenshot des WhoIs Applets.



### 10.13.2.8. Ausführungsprotokoll

```
URLClassLoader Konstruktor
/whoisApplet
Laden der Klasse /whoisApplet
Suchen der Klasse /whoisApplet im Systempfad.
Laden der Klasse /whoisApplet
Lesen der Datei in ein Byte Array
Datei wurde in das Byte Array gelesen
Laden der Klasse java.applet.Applet
Suchen der Klasse java.applet.Applet im Systempfad.
Laden der Klasse java.lang.Throwable
Suchen der Klasse java.lang.Throwable im Systempfad.
Laden der Klasse java.io.IOException
Suchen der Klasse java.io.IOException im Systempfad.
Laden der Klasse java.awt.TextComponent
Suchen der Klasse java.awt.TextComponent im Systempfad.
Laden der Klasse java.awt.TextArea
Suchen der Klasse java.awt.TextArea im Systempfad.
Laden der Klasse java.awt.TextField
Suchen der Klasse java.awt.TextField im Systempfad.
Laden der Klasse java.awt.LayoutManager
Suchen der Klasse java.awt.LayoutManager im Systempfad.
Laden der Klasse java.awt.Label
Suchen der Klasse java.awt.Label im Systempfad.
Laden der Klasse java.awt.Checkbox
Suchen der Klasse java.awt.Checkbox im Systempfad.
Laden der Klasse java.awt.List
Suchen der Klasse java.awt.List im Systempfad.
Laden der Klasse java.awt.Button
Suchen der Klasse java.awt.Button im Systempfad.
Laden der Klasse java.awt.Frame
Suchen der Klasse java.awt.Frame im Systempfad.
Laden der Klasse java.awt.Window
Suchen der Klasse java.awt.Window im Systempfad.
Laden der Klasse java.awt.BorderLayout
Suchen der Klasse java.awt.BorderLayout im Systempfad.
Laden der Klasse java.awt.Container
Suchen der Klasse java.awt.Container im Systempfad.
Laden der Klasse java.awt.Panel
Suchen der Klasse java.awt.Panel im Systempfad.
Laden der Klasse java.awt.GridLayout
Suchen der Klasse java.awt.GridLayout im Systempfad.
Laden der Klasse java.awt.FlowLayout
Suchen der Klasse java.awt.FlowLayout im Systempfad.
Laden der Klasse java.awt.CheckboxGroup
Suchen der Klasse java.awt.CheckboxGroup im Systempfad.
Laden der Klasse java.awt.Event
Suchen der Klasse java.awt.Event im Systempfad.
```

### 10.13.3. Daten eines Formulars an ein CGI Skript übermitteln

Wir möchten mit Hilfe der POST Methode Daten an ein CGI Skript übermitteln, zum Beispiel an eine Suchmaschine, oder ein Mailingsystem, oder ein Datenbankabfrage-Skript, oder ... Dabei soll alles ohne Hacking, völlig legal geschehen.

Hier kurz eine Skizze, wie wir vorgehen:
1. kreieren eines URLConnect Objektes
2. senden eines Query Stringes mit Hilfe des OutputStreams

Zu erklären gibt's recht wenig. Schauen wir uns das Programm gleich an:

#### 10.13.3.1. Programmbeispiel

```
//Titel:        POST von Fomulardaten an CGI Skript
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Beispiel 10_16:
//URLConnect Objekt kreieren, wobei
//der URL ein CGI Skript beschreibt.
//Mit Hilfe des OutputStreams POST Methode
//simulieren / implementieren.
package Beispiel10_16;
import java.net.*;
import java.io.*;

//  KlasseBeispiel10_16 : besser wäre ein Name wie POSTForm
public class KlasseBeispiel10_16 {
  URL u;

  public static void main(String args[]) {
    System.out.println("Start Beispiel 10_16");
    String s;
    try {
      s = args[0];
    }
    catch (ArrayIndexOutOfBoundsException e) {
      s = "http://hoohoo.ncsa.uiuc.edu/cgi-bin/post-query";
    }
    System.out.println("CGI Skript Adresse : "+s);
    try {
      KlasseBeispiel10_16 pf = new KlasseBeispiel10_16();
      pf.u = new URL(s);
      pf.submitData();
    }
    catch (MalformedURLException e) {
```

```
      System.err.println(args[0] + " kann nicht als URL interpretiert werden");
    }
  }
  void submitData() {
    System.out.println("Start submitData()");
    String query = "name=" + URLEncoder.encode("Josef M. Joller");
    query += "&";
    query += "email=" + URLEncoder.encode("josef.m.joller@switzerland.org");
    System.out.println("Query : "+query);
    int cl = query.length();

    try {
      URLConnection uc = u.openConnection();
      System.out.println("URLConnection : "+uc);
      uc.setDoOutput(true);
      uc.setDoInput(true);
      uc.setAllowUserInteraction(false);
      DataOutputStream dos = new DataOutputStream(uc.getOutputStream());
      // Schicke die Daten
      dos.writeBytes(query);
      dos.close();
      // Lesen der Antwort
      DataInputStream dis = new DataInputStream(uc.getInputStream());
      String nextline;
      while((nextline = dis.readLine()) != null) {
        // Ausgabe der Antwort
       System.out.println(nextline);
      }
      dis.close();
    }
    catch (Exception e) {
      System.err.println(e);
    }
  }
}
```

#### 10.13.3.2. Ausgabe

```
Start Beispiel 10_16
CGI Skript Adresse : http://hoohoo.ncsa.uiuc.edu/cgi-bin/post-query
Start submitData()
Query : name=Josef+M.+Joller&email=josef.m.joller%40switzerland.org
URLConnection : sun.net.www.protocol.http.HttpURLConnection:http://hoohoo.ncsa.uiuc.edu/cgi-bin/post-query
java.net.UnknownHostException: hoohoo.ncsa.uiuc.edu
```

Da fehlt doch was?

1. das CGI Skript wurde nicht gefunden, weil der PC nicht am Netz angeschlossen war
2. das CGI Skript ist mir unbekannt; ich weiss also nicht, was es für eine Zeichenkette erwartet.

### *10.14. Aufgaben*

Die Lösungen für die folgenden Aufgaben müssen auf dem Server abgelegt werden!

1. Im Beispiel 10_13 wurde eine binäre Datei ab einer URL in das lokale Verzeichnis kopiert. Das Programm kann auch *.exe oder *.pdf oder *.zip Dateien kopieren. Das Programm kann aber eine einfache HTML *nicht* kopieren!
   Erweitern Sie das Programm so, dass eine beliebige Datei herunter geladen werden kann. Testen Sie Ihr Programm mit einer HTML Seite!
2. Installieren Sie und Testen Sie das URLRequester Programm.
   Aufgabe:
   bauen Sie zum Programm aus Aufgabe 1 eine grafische Benutzeroberfläche, die besser ist als jene des URLRequester Programmes.
   Das URLRequester Programm wird beim Starten von Windows (NT) jeweils automatisch gestartet. Wie könnte Ihr URLRequester Programm automatisch gestartet werden?
3. Entwickeln oder finden Sie ein CGI Skript, mit dem Sie das Programm Beispiel 10_16 testen können.
   Sie können auch (und das wäre sicher angebracht) einen anderen String an das Skript senden, zum Beispiel an eine Suchmaschine

## *10.15. Anhang : HttpURLConnection Klasse*

### 10.15.1. java.net Class HttpURLConnection

```
java.lang.Object
  |
  +--java.net.URLConnection
        |
        +--java.net.HttpURLConnection
```

public abstract class **HttpURLConnection**
extends URLConnection

Eine URLConnection Klassenwerweiterung, welche die HTTP-specifischen Details einer URL Verbindung unterstützt.
**Seit:**
JDK1.1

#### 10.15.1.1. Felder Übersicht

| | |
|---|---|
| static int | **HTTP_ACCEPTED** |
| static int | **HTTP_BAD_GATEWAY** |
| static int | **HTTP_BAD_METHOD** |
| static int | **HTTP_BAD_REQUEST**<br>4XX: client error |
| static int | **HTTP_CLIENT_TIMEOUT** |
| static int | **HTTP_CONFLICT** |
| static int | **HTTP_CREATED** |
| static int | **HTTP_ENTITY_TOO_LARGE** |
| static int | **HTTP_FORBIDDEN** |
| static int | **HTTP_GATEWAY_TIMEOUT** |
| static int | **HTTP_GONE** |
| static int | **HTTP_INTERNAL_ERROR** |
| static int | **HTTP_LENGTH_REQUIRED** |
| static int | **HTTP_MOVED_PERM** |

| | |
|---|---|
| static int | **HTTP_MOVED_TEMP** |
| static int | **HTTP_MULT_CHOICE**<br>3XX: relocation/redirect |
| static int | **HTTP_NO_CONTENT** |
| static int | **HTTP_NOT_ACCEPTABLE** |
| static int | **HTTP_NOT_AUTHORITATIVE** |
| static int | **HTTP_NOT_FOUND** |
| static int | **HTTP_NOT_MODIFIED** |
| static int | **HTTP_OK**<br>2XX: generally "OK" |
| static int | **HTTP_PARTIAL** |
| static int | **HTTP_PAYMENT_REQUIRED** |
| static int | **HTTP_PRECON_FAILED** |
| static int | **HTTP_PROXY_AUTH** |
| static int | **HTTP_REQ_TOO_LONG** |
| static int | **HTTP_RESET** |
| static int | **HTTP_SEE_OTHER** |
| static int | **HTTP_SERVER_ERROR**<br>5XX: server error |
| static int | **HTTP_UNAUTHORIZED** |
| static int | **HTTP_UNAVAILABLE** |
| static int | **HTTP_UNSUPPORTED_TYPE** |
| static int | **HTTP_USE_PROXY** |
| static int | **HTTP_VERSION** |
| protected String | **method** |
| protected int | **responseCode** |
| protected String | **responseMessage** |

### 10.15.1.2. Felder aus class java.net.URLConnection

| |
|---|
| allowUserInteraction, connected, doInput, doOutput, ifModifiedSince, url, useCaches |

### 10.15.1.3. Konstruktor Übersicht

| | |
|---|---|
| protected | **HttpURLConnection**(URL u)<br>Konstruktor für den URLStreamHandler. |

### 10.15.1.4. Methoden Übersicht

| | |
|---|---|
| abstract void | **disconnect**()<br>Verbindung zum Server abbauen. |
| InputStream | **getErrorStream**()<br>Der Server hat zwar korrekte Daten geliefert, aber eine Fehlermeldung anzeigt. |
| static boolean | **getFollowRedirects**() |
| Permission | **getPermission**()<br>Liefert die Berechtigung, die benötigt wird, um auf ein Objekt zuzugreifen. |
| String | **getRequestMethod**()<br>Bestimmen der Request Methode. |
| int | **getResponseCode**()<br>HTTP Response Status. |
| String | **getResponseMessage**()<br>HTTP Antwort, falls überhaupt, mit Fehlercode. |
| static void | **setFollowRedirects**(boolean set)<br>HTTP redirects (Requests mit Response Code 3xx). |
| void | **setRequestMethod**(String method)<br>Setzt die URL Request Methode, also: GET POST HEAD OPTIONS PUT DELETE TRACE. |
| abstract boolean | **usingProxy**()<br>Zeigt an, ob die Verbindung durch einen Proxy Server geleitet wird.. |

.