**In diesem Kapitel:**
- *Die URL Klasse*
- *Die URL Encoder Klasse*
- *Beispielprogramme*

# 5

# *Lesen von Daten aus einem URL*

## *5.1. Worum geht es?*

Die einfachste Art und Weise Daten in verteilten Systemen zu lesen, besteht darin, dass man den URL angibt. Java, speziell die Netzklassen, sorgen für das korrekte Protokoll. Allerdings sind nicht alle denkbaren Protokolle und Inhalte bereits implementiert. Der Java Programmierer kann aber selber Erweiterungen programmieren. Wir werden später noch Erweiterungsmöglichkeiten kennen lernen.

Eine vollständige Übersicht über die URL Klasse ist, in Englisch, im Anhang enthalten; die meisten Teile davon sind im Text in Deutsch vorhanden. Es gibt weitere URL bezogene Klassen, auf die wir,wie oben erwähnt, später zurück kommen.

In diesem Kapitel besprechen wir die Klassen : java.net.url und java.net.URLEncoder/Decoder.

## *5.2. Die URL Klasse*

Die java.net.URL Klasse ist eine Abstraktion des Uniform Resource Locator Konzeptes. Das sehen Sie am Besten daran, dass im Klassenbaum von java.net die URL Klasse diverse Zusatzklassen hat.

```
class java.net.URL (implements java.io.Serializable)
        class java.net.URLConnection
            class java.net.HttpURLConnection
            class java.net.JarURLConnection
        class java.net.URLDecoder
        class java.net.URLEncoder
        class java.net.URLStreamHandler
```

Ein URL kann entweder als Zeichenkette oder als Objekt aufgefasst werden. Es erübrigt sich, im Java Umfeld, zu fragen, welche Version hier gewählt wurde. Die Darstellung eines URL als Objekt erlaubt es uns, verschiedene Methoden zu definieren, die den Zugriff auf bestimmte Teile des URL erlauben : Protokoll, Host, Port, Pfad, Dateiname, Dokument Sektion. Das ist in etwa das, was die URL Klasse bietet. Lediglich Pfad und Dateiname werden in der URL Klasse als *ein* Feld zusammen gefasst.


©J.M.Joller

Die Felder von java.net.URL sind nur den andern Klassen des Paketes java.net sichtbar. Alle anden Klassen können nicht direkt auf die Datenfelder zugreifen. Allerdings ist es allen Klassen möglich, mit Hilfe der Konstruktoren, die Felder zu setzen, und mit Hilfe von speziellen Methoden die Datenfelder abzufragen: getHost(), getPort(), ...

| | |
|---|---|
| Object | **getContent**()<br>liefert den Inhalt der URL. |
| String | **getFile**()<br>liefert den Dateinamen der URL. |
| String | **getHost**()<br>liefert, falls vorhanden,den Host Namen der URL. |
| int | **getPort**()<br>liefert die Port Nummer der URL. |
| String | **getProtocol**()<br>liefert den Namen des Protokolls dieser URL. |
| String | **getRef**()<br>liefert den Anker ("HREF", "REF" ) dieser URL. |

Nachdem die URL, das URL Objekt kreiert ist, steht eine Methode zur Modifikation der URL Datenfelder zur Verfügung:

| | |
|---|---|
| protected void | **set**(String protocol, String host, int port, String file, String ref)<br>setzt die Felder der URL. |

In der Regel sollten die Datenfelder eines einmal kreierten URL Objektes nicht mehr verändert werden. Die Methode ist auch geschützt (protected), kann also nicht von ausserhalb des Paketes genutzt werden. Diese Methode wird eigentlich nur benötigt, wenn man eigene, neue Protokolle implementieren möchte.

Eine weitere wichtige Methoden ist

| | |
|---|---|
| Object | **getContent**()<br>Liefert den Inhalt dieser URL. |

mit deren Hilfe der Inhalt der URL erhalten werden kann. Die Methode ist eine Kurzform für:

openConnection().getContent()

Die Methode kann eine IOException werfen, falls eine I/O Ausnahme auftritt.

Weitere Methoden ergeben sich aus der Oberklasse Object.

| | |
|---|---|
| boolean | **equals**(Object obj)<br>vergleicht zwei URLs. |
| String | **toString**()<br>konstruiert eine Zeichenketten-Darstellung des URL. |


©J.M.Joller

### 5.2.1. Methodenübersicht für die Klasse URL

Zusammenfassend erhalten wir folgende Methoden von der URL Klasse:

| | |
|---|---|
| boolean | **equals**(Object obj)<br>vergleiche zwei URLs. |
| Object | **getContent**()<br>liefret den Inhalt des URL. |
| String | **getFile**()<br>liefert den Dateinamen des URL. |
| String | **getHost**()<br>liefert den Host Namen des URL, falls vorhanden. |
| int | **getPort**()<br>liefert die Port Nummer des URl |
| String | **getProtocol**()<br>liefert den Protokollnamen dieses URL. |
| String | **getRef**()<br>liefert den Anker (die "Referenz") dieses URL. |
| int | **hashCode**()<br>kreiert eine ganze Zahl als Hashtabellen Index. |
| URLConnection | **openConnection**()<br>liefert ein URLConnection Objekt auf das URL Objekt.. |
| InputStream | **openStream**()<br>öffent eine Verbindung zum URL und liefert einen InputStream mit dessen Hilfe vom URL gelesen werden kann. |
| boolean | **sameFile**(URL other)<br>vergleicht zwei URLs, ohne die "Ref" Felder. |
| protected void | **set**(String protocol, String host, int port, String file, String ref)<br>setzt die Felder des URL. |
| static void | **setURLStreamHandlerFactory**(URLStreamHandlerFactory fac)<br>setzt die StreamHandlerFactory für eine Applikation(siehe später). |
| String | **toExternalForm**()<br>liefert eine Zeichenketten Darstellung des URL. |
| String | **toString**()<br>liefert auch eine Zeichenketten Darstellung des URL, wie aus der Oberklasse Object bekannt ist. |

| Von class java.lang.Object geerbte Methoden |
|---|
| clone, finalize, getClass, notify, notifyAll, wait, wait, wait |

Wenden wir uns jetzt den Konstruktoren und den Methoden im Einzelnen zu und betrachten wir verschiedene Beispiele.

## *5.3. Kreieren eines neuen URL's*

Im Gegensatz zur InetAddress, die wir im letzten Kapitel behandelt haben, kann man URL Objekte, Instanzen der URL Klasse, konstruieren (InetAddress hat bekanntliche keinen Konstruktor).


©J.M.Joller

Es gibt mehrere Konstruktoren für den"Bau" eines URL's, eines Uniform Resource Locators.

### 5.3.1. Konstruktoren der Klasse URL : Übersicht

| | |
|---|---|
| **URL**(String spec)<br>kreiert ein URL Objekt aus einer Zeichenketten Darstellung. | |
| **URL**(String protocol, String host, int port, String file)<br>kreiert ein URL Objekt, bestehend aus einem spezifizierten Protokoll, einem Host und einer Port Nummer und einem Dateinamen. | |
| **URL**(String protocol, String host, int port, String file, URLStreamHandler handler)<br>kreiert ein URL Objekt, bestehend aus einem spezifizierten Protokoll, Host, Port Nummer, Datei und Handler. | |
| **URL**(String protocol, String host, String file)<br>kreiert ein absolutes URL Objekt, bestehend aus dem specifizierten Protokoll Namen, Host Namen, und Datei Name. | |
| **URL**(URL context, String spec)<br>kreiert ein URL Objekt, indem eine Spezifikation (String spec) zerlegt wird, im Kontext eines bestehenden URL Objektes. | |
| **URL**(URL context, String spec, URLStreamHandler handler)<br>kreiert einen URL durch Analyse einer Zeichenkette im Kontext. | |

Alle diese Konstruktoren werfen die MalformedURLException , falls Sie versuchen ein URL Objekt zu einem nicht exisitierenden Protokoll zu kreieren. Welchen Konstruktor man konkret einsetzt, hängt von den Angaben ab, die man zur Verfügung hat.

Testen wir einmal, ob die gängigen Protokolle unterstützt werden.

#### 5.3.1.1. Selbsttestaufgabe : unterstützte Protokolle

Schreiben Sie ein Programm, welches ein URL Objekt kreiert, jeweils eines für die unterschiedlichen Protokolle :
HTTP, FTP, mailto, news, gopher,.. (was noch?)

Neben dieser Prüfung findet keinerlei Validation der URL Objekt-Parameter statt. Es liegt also an Ihnen, gültige Hosts, Dateinamen oder Sektionen zu finden und als Parameter zur Kreation eines URL Objektes zu verwenden

### 5.3.2. public URL(String url) throws MalformedURLException

Der einfachste Konstruktor besitzt einfach eine Zeichenkette als Parameter. Diese Zeichenkette umfasst alle Teile eines gültigen URL's .

Hier eine typische Programmzeile:

```
try {
        URL u = new URL("http://www.switzerland.org/Joller/meineStartseite.html");
}
catch (MalformedURLException mue)
{
        System.err.println(mue);                // oder
        System.err.println(mue.getMessage());
        // oder eine selber konstruierte Meldung
}
```


©J.M.Joller

Schauen wir uns ein vollständiges Beispiel an:

#### 5.3.2.1. Beispiel

```
//Titel:        public URL(String url) throws MalformedURLException
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Beispiel zum obigen Konstruktor.
//Als Parameter erhält der Konstruktor eine Zeichenkette, die alle Angaben zur Konstruktion eines
URL Objektes enthält.
package Beispiel51;
import java.net.*;
class Beispiel51Programm {
  public static void main(String args[]) {
    String[] urlString = {"http://www.switzerland.org/Joller/unsereFirma.html",
                          "telnet://telnet.switzerland.org"};
    int i=0;
    URL urlWeb, urlTelnet;
    try {
      urlWeb = new URL(urlString[i]);
      System.out.println("Web URL Objekt : "+urlWeb);
      i++;
      urlTelnet = new URL(urlString[i]);
      System.out.println("Telnet URL Objekt : "+urlTelnet);
    } catch (MalformedURLException mue)
    {
      System.err.println("Das URL Objekt "+urlString[i]+" konnte nicht kreiert werden");
      System.err.println("Message : "+mue.getMessage());
    }
  }
}
```

liefert die folgende Ausgabe:

```
Web URL Objekt : http://www.switzerland.org/Joller/unsereFirma.html
Das URL Objekt telnet://telnet.switzerland.org konnte nicht kreiert werden
Message : unknown protocol: telnet
```

### 5.3.3. public URL(String protocol, String host, String file) throws MalformedURLException

Dieser Konstruktor erlaubt eine differenzierte Spezifikation des URL Objektes:
die Parameter Protokoll, Host und Datei können einzeln angegeben werden.

Schauen wir uns einen typischen Programmauszug an:

```
try {
        URL urlAdresse = new URL("http", "www.switzerland.org","/meineStartseite.html");
} catch (MalformedURLException mue) {
        System.err.println(mue);}
```


©J.M.Joller

#### 5.3.3.1. Achtung : typische Fehlerquelle

Oft wird bei der Spezifikation der Datei der erste Slash vergessen. Sie müssen beachten, dass der Host nicht mit einem Slash endet. In einer Adresse wie zum Beispiel "http://www.switzerland.org/eMailListe.html" ist der Dateianteil der Adresse "/eMailListe.html", nicht "eMailListe.html"!

#### 5.3.3.2. Beispiel

Das Programmbeispiel unterscheidet sich nicht wesentlich vom vorherigen. Der einzige Unterschied ist de facto der Konstruktor.

```
//Titel:        public URL(String protocol, String host, String file)
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Bespiel für das Instanzieren der URL Klasse mit Hilfe des obigen URL Konstruktors
package Beispiel52;

import java.net.*;
public class Beispiel52Programm {

  public static void main(String[] args) {
    String[] strURLProtokoll = {"http", "telnet"};
    String[] strURLHost = {"www.switzerland.org", "www.schweiz.ch"};
    String[] strURLDatei = {"/diesenHostGibts.html", "/telnetDateienGibtsNicht"};
    int i=0;
    URL urlWeb, urlTelnet;
    try {
      urlWeb = new URL(strURLProtokoll[i], strURLHost[i], strURLDatei[i]);
      System.out.println("Das Objekt "+urlWeb+" wurde kreiert!");
      i++;
      urlTelnet = new URL(strURLProtokoll[i], strURLHost[i], strURLDatei[i]);
    } catch(MalformedURLException mue) {
      System.err.println("Das URL Objekt mit dem Protokoll "+strURLProtokoll[i]+", dem Host "+strURLHost[i]+" und der Datei "+strURLDatei[i]+" konnte nicht kreiert werden!");
      System.err.println(mue.getMessage());
    }
  }
}
```

Und hier die Ausgabe:

```
Das Objekt http://www.switzerland.org/diesenHostGibts.html wurde kreiert!
Das URL Objekt mit dem Protokoll telnet, dem Host www.schweiz.ch und der Datei /telnetDateienGibtsNicht konnte nicht kreiert werden!
unknown protocol: telnet
```


©J.M.Joller

### 5.3.4. public URL(String protocol, String host, int port, String file)

Dieser Konstruktor erlaubt eine differenzierte Spezifikation des URL Objektes:
die Parameter Protokoll, Host, Port und Datei können einzeln angegeben werden.

Schauen wir uns einen typischen Programmauszug an:

```
try {
        URL urlAdresse = new URL("http", "www.switzerland.org", 80, "/meineStartseite.html");
} catch (MalformedURLException mue) {
        System.err.println(mue);
}
```

#### 5.3.4.1. Beispiel

Das Programmbeispiel unterscheidet sich nicht wesentlich vom vorherigen. Der einzige Unterschied ist wieder der Konstruktor.

```
//Titel:        public URL(String protocol, String host, int port, String file)
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Bespiel für das Instanzieren der URL Klasse mit Hilfe des obigen URL
Konstruktors
package Beispiel53;

import java.net.*;
public class Beispiel53Programm {

  public static void main(String[] args) {
    String[] strURLProtokoll = {"http", "telnet"};
    String[] strURLHost = {"www.switzerland.org", "www.schweiz.ch"};
    String[] strURLDatei = {"/diesenHostGibts.html", "/telnetDateienGibtsNicht"};
    int[] iURLPort = {80, 120};

    int i=0;
    URL urlWeb, urlTelnet;
    try {
      urlWeb = new URL(strURLProtokoll[i], strURLHost[i], iURLPort[i], strURLDatei[i]);
      System.out.println("Das Objekt "+urlWeb+" wurde kreiert!");
      i++;
      urlTelnet = new URL(strURLProtokoll[i], strURLHost[i], iURLPort[i], strURLDatei[i]);
    } catch(MalformedURLException mue) {
      System.err.println("Das URL Objekt mit dem Protokoll "+strURLProtokoll[i]+", dem Host
"+strURLHost[i]+", dem Port "+iURLPort[i]+" und der Datei "+strURLDatei[i]+" konnte nicht
kreiert werden!");
      System.err.println(mue.getMessage());
    }  }}
```


©J.M.Joller

Und hier die Ausgabe:

```
Das Objekt http://www.switzerland.org:80/diesenHostGibts.html wurde kreiert!
Das URL Objekt mit dem Protokoll telnet, dem Host www.schweiz.ch, dem Port 120 und der Datei /telnetDateienGibtsNicht konnte nicht kreiert werden!
unknown protocol: telnet
```

### 5.3.5. public URL(URL u, String relativeAddress)

Dieser Konstruktor erlaubt eine Spezifikation des URL Objektes, ausgehend von einer Basisadresse. :
der Parameter URL legt die Basis fest;
die Zeichenkette definiet den Rest.

Schauen wir uns einen typischen Programmauszug an:

```
try {
        URL urlAdresse = new URL(urlBasis, strURLDatei);
} catch (MalformedURLException mue) {
        System.err.println(mue);
}
```

Dieser Konstruktor hat eine nette Eigenschaft:

- falls Sie bereits ein URL Objekt haben, dann können Sie ein weiteres URL Objekt relativ zum ersten Objekt kreieren, indem Sie einfach den neuen Dateinamen angeben. Der Rest des URL Objektes wird dann übernommen

Ein Programm Fragment würde wie folgt aussehen:

```
try {
        URL url1 = new URL("http://www.switzerland.org/index.html");
        ...
        URL url2 = new URL(url1, "meineStartseite.html");
} catch (MaformedURLException mue) {
        System.err.println(mue);
}
```

In url2 wird der Dateiname entfernt und der neue Dateinamen hinzu gefügt.

Hier ein vollständiges Beispiel:


©J.M.Joller

#### 5.3.5.1. Beispiel

```
//Titel:        public URL(URL u, String s) throws MalformedURLException
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Beispiel für das Ersetzen des Dateinamens aus einer URL mit Hilfe des obigen Konstruktors
package Beispiel54a;

import java.net.*;
public class Beispiel54aProgramm {

  public static void main(String[] args) {
    try {
      URL u1 = new URL("http://www.switzerland.org/index.html");
      System.out.println("URL Objekt : "+u1);
      URL u2 = new URL(u1, "meineStartseite.html");
      System.out.println("modifiziertes URL Objekt : "+u2);
    } catch(MalformedURLException mue) {
      System.err.println("Fehler beim Kreieren des URL Objektes");
      System.err.println(mue);
    }
  }
}
```

Und hier die Ausgabe:

```
URL Objekt : http://www.switzerland.org/index.html
modifiziertes URL Objekt : http://www.switzerland.org/meineStartseite.html
```

Nicht schlecht, oder was denken Sie darüber?

#### 5.3.5.2. Einsatzmöglichkeiten

Sie können damit Seiten herunter laden und die Adressen relativ zur Basis Adresse modifizieren. Wir lernen ein solche "Page Saver" Programm noch kennen.
Eine andere Einsatzmöglichkeit ist in Applets:
Applets liefern Ihnen die Basisadresse (DocumentBase()). Schauen wir uns ein weiteres Beispiel an:


©J.M.Joller

#### 5.3.5.3. Beispiel

Das Programmbeispiel unterscheidet sich insofern vom vorherigen, dass wir jetzt ein Applet konstruieren.

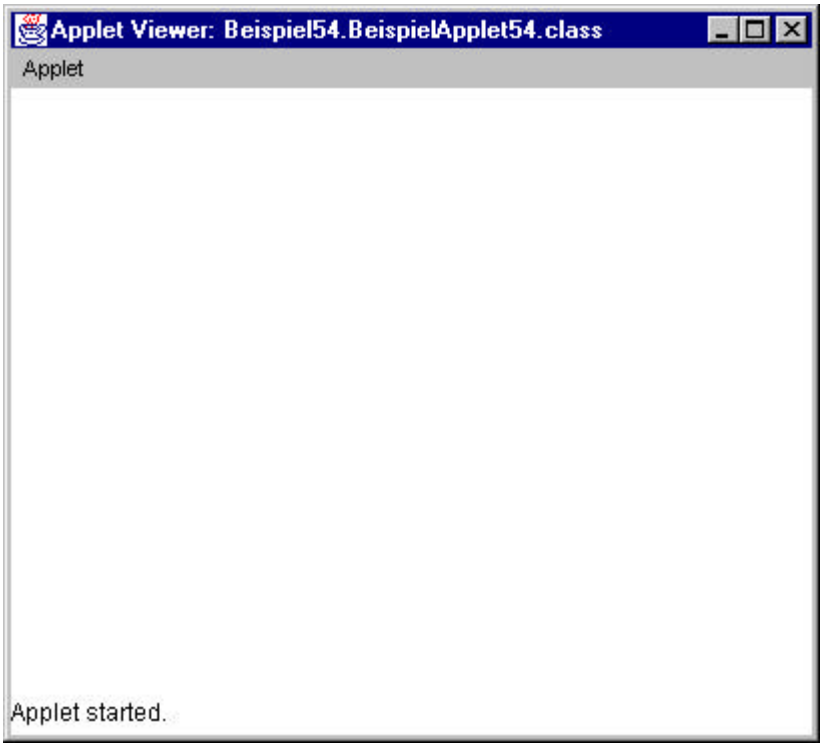
Hier das Programm :

```
//Titel:       public URL(URL u, String s) throws MalformedURLException
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: Beispiel Programm in Form eines Applets, welches den obigen Konstruktor zum Kreieren eines URL und eines relativen URL Objektes verwendet
package Beispiel54;

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

public class BeispielApplet54 extends Applet {

  //Das Applet initialisieren
  public void init() {
      URL u1, u2;
      u1 = getDocumentBase();
      System.out.println(u1);
      try {
        u2 = new URL(u1, "meineStartseite.html");
        System.out.println(u2);
      } catch(MalformedURLException mue) {
        System.err.println(mue);
        System.out.println("Stack Trace : ");
        mue.printStackTrace();
      }
  }
}
```

Die Ausgabe der Stabdard Streams (System.out.printl() ) werden in das Ausführungsprotokoll umgeleitet. Das Applet manifestiert sich als leerer Applet Rahmen.


©J.M.Joller

Hier der Bildschirm Snapshot: schön leer!




und hier das Ausführungsprotokoll:
file:/D:/UnterrichtsUnterlagen/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel54.BeispielApplet54.html
file:/D:/UnterrichtsUnterlagen/ParalleleUndVerteilteSysteme/Programme/Kapitel5/meineStartseite.html

auch nicht schlecht, oder?

Das obige Programm kann weiter vereinfacht werden:
statt zuerst u1= getDocumentBase(); zu setzen und dann mit u2=new (URL(u1, ...) das neue URL Objekt zu kreieren, kann man direkt und abgekürzt schreiben :

u1 = new URL(getDocumentBase(), "meineSeite.html");

Nachdem wir nun URL Objekte kreiert haben, wollen wir die einzelnen Bausteine der URL Objekte bestimmen.


©J.M.Joller

## *5.4. Zerlegung eines URL Objektes*

Ein URL Objekt hat folgende Komponenten, Felder :
1. Protokoll
2. Host
3. Port
4. Datei
5. URLStreamHandler (interpretiert den Datenstrom, static, also für alle Klassen aus java.net zugänglich)
6. Anker oder Sektion (Ref Feld)

Diese Felder sind mit Hilfe spezieller Methoden abfragbar:
1. getProtocol()
2. getHost()
3. getPort()
4. getFile()
5. getRef()

Schauen wir uns diese Methoden kurz einzeln, anschliessend in einem Programm an.

### 5.4.1. Kurzbeschreibung der Methoden

#### 5.4.1.1. public String getProtocol()

Als Programm Fragment:

```
...
URL urlObjekt = getCodeBase();
System.out.println("CodeBase : "+urlObjekt);
System.out.println("Dieses Applet wurde mit Hilfe des Protokolls
                                    "+urlObject.getProtocol() +" herunter geladen");
```

#### 5.4.1.2. public String getHost()

Programm Fragment :

```
...
URL urlObjekt = getCodeBase();
System.out.println("Dieses Applet wurde vom Host "+getHost()+" herunter geladen");
```

#### 5.4.1.3. public int getPort()

Programm Fragment :
beim Port kann das Problem auftreten, dass kein Port spezifiziert wurde. Nun gibt es zwei Möglichkeiten:
1. die Methode könnte den Standard Port dieses Protokolls angeben. Das ist aber eher unsinnig,speziell bei selbst definierten Protokollen.
2. die Methode könnte eine Ausnahme werfen

Die zweite Möglichkeit wurde in Java implementiert:


©J.M.Joller

*falls der Port nicht definiert ist, liefert die Methode getPort() den Wert -1*

```
...
try {
        URL u = new URL("http://switzerland.org/index.html#Anmeldeformular" );
        System.out.println("Das URL Objekt "+u+" besitzt folgenden Port : "+getPort() );
} catch(MalformedURLException mue) {
        System.err.println(mue);
}
...
```

#### 5.4.1.4. public String getFile()

Die getFile() Methode liefert den Datei Anteil des URL Objektes, also Pfad *plus* Dateiname. Alles nach dem ersten Slash wird als Datei interpretiert.

Hier ein Programm Fragment:

```
...
URL urlSeite = getDocumentBase();
System.out.println("Der Datei des URL Objektes "+urlSeite+" lautet : "+urlSeite.getFile());
...
```

Falls das URL Objekt keine Datei besitzt, wird einfach ein Slash "/" zurück gegeben.
Im Beispiel URL Objekt für "www.suisse.org" liefert getFile() die Zeichenkette "/".

#### 5.4.1.5. public String getRef()

Die getRef() Methode liefert den Anker des URL Objektes. Falls das URL Objekt keinen Anker besitzt liefert die Methode das null Objekt zurück.

Programm Fragment:

```
...
try {
        URL urlTest = new URL("http://www.suiss.org/index.html#Adressen");
        System.out.println("Das URL Objekt "+urlTest+" besitzt folgende Referenz :"
                        +urlTest.getRef());
...
```

Jetzt fügen wir alle Methoden in einem Test Programm zusammen.


©J.M.Joller

### 5.4.2. Beispiel : URL in seine Bestandteile zerlegen

```
//Titel:        URL in seine Bestandteile zerlegen
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Aufruf des Konstruktors und der meisten der verfügbaren Methoden der Klasse
URL
package Beispiel55;

import java.net.*;
public class BeispielProgramm55 {

  public static void main(String[] args) {

    String[] strTest = {"http://www.swizzera.org/index.html#TestSection",
"ftp://archive:89/test#alpha", "telnet://telnet.suisse.org:123/123.34.112.121#9"};

    for (int i=0;i<strTest.length; i++) {
      try {
        URL urlTest = new URL(strTest[i]);
        System.out.println("----------"+strTest[i]+"----------");
        System.out.println("URL Objekt : "+urlTest);
        System.out.println("  Protokoll : "+urlTest.getProtocol());
        System.out.println("  Host      : "+urlTest.getHost());
        System.out.println("  Port      : "+urlTest.getPort() );
        System.out.println("  Datei     : "+urlTest.getFile() );
        System.out.println("  Referenz  : "+urlTest.getRef() );
      } catch(MalformedURLException mue) {
        System.err.println("Im URL Objekt "+strTest[i]+" trat ein Fehler beim Bestimmen der
Bestandteile auf");
      }
    }
  }
}
```

Und als Ausgabe erhalten wir:

```
----------http://www.swizzera.org/index.html#TestSection----------
URL Objekt : http://www.swizzera.org/index.html#TestSection
  Protokoll : http
  Host      : www.swizzera.org
  Port      : -1
  Datei     : /index.html
  Referenz  : TestSection
----------ftp://archive:89/test#alpha----------
URL Objekt : ftp://archive:89/test
  Protokoll : ftp
  Host      : archive
  Port      : 89
  Datei     : /test
  Referenz  : null
Im URL Objekt telnet://telnet.suisse.org:123/123.34.112.121#9 trat ein Fehler beim Bestimmen der Bestandteile auf
```


©J.M.Joller

Interessant ist die Tatsache, dass beim FTP Protokoll die Referenz ingoriert wird (macht Sinn)!

## *5.5. Lesen von Daten aus einem Uniform Resource Locator URL*

Im Package java.net gibt es mehrere Klassen und Methoden, die einen Zugriff auf die Daten erlauben, die sich am Ort befinden, der durch den URL beschrieben wird.

1. public final InputStream openStream throws java.io.IOException
2. public URLConnection openConnection() throws java.io.IOException
3. public final getContent() throws java.io.IOException
   (Kurzform für : openConnection().getContent())

### 5.5.1. Kurzbeschreibung der Methoden für den Datenzugriff

Die obigen Methoden werden wir im Einzelnen kennen lernen, auch in Beispielen.
ls erstes wollen wir uns eine Übersicht verschaffen.

#### 5.5.1.1. public final InputStream openStream() throws java.io.IOException

OpenStream stellt eine Verbindung zum URL her und führt alle nötigen Handshakes durch. Als Benutzer dieser Methode erhalten wir einen InputStream. Die Daten im InputStream werden uninterpretiert geliefert. Java kümmert sich nicht darum, ob es sich um ein Bild, Text oder ein Audio Clip handelt.

Insbesondere erhält man keinerlei Protokoll Anweisungungen, wie GET ... vom HTTP Protokoll.

Ein Programm Fragment sieht wie folgt aus:

```
...
try {
        urlTest = new URL(" http:/www.switzerland.org ");
        is = urlTest.openStream();
} catch(Exception e) {
        System.err.println(e);
}
```

Die generische Exception hat einen grossen Vorteil:
in diesem Programm Fragment können zwei Exception auftreten:
1. IOException
2. MalformedURLException

Die obige Variante ist offensichtlich einfacher; die Fehlerbehandlung , die Fehlerfindung aber komplexer.

Schauen wir uns einmal die Daten an, die wir von einer (künstlich aufgebauten) URL Adresse lesen können.


©J.M.Joller

#### 5.5.1.1.1. Beispiel : Lesen und Herunterladen einer Web Seite

```
//Titel:        Herunterladen einer Web Seite
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Testen verschiedener URL Methoden:
//URL Konstruktor
//DataInputStream
//aus einer Textseite
package Beispiel56;

import java.net.*;
import java.io.*;
public class ProgrammBeispiel56 {

  public static void main(String[] args) {
    String strZeile;
    try {
      URL urlTest = new
URL("http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel56/Projekt56.ht
ml");
      // jetzt verbinden wir das URL Objekt mit einem Stream
      try {
        DataInputStream disHTML = new DataInputStream(urlTest.openStream());

        try {
          while( (strZeile = disHTML.readLine()) != null) System.out.println(strZeile);
        } catch(Exception e) {
          System.err.println(e);
          e.printStackTrace();
        }
      } catch(Exception e) {
        System.err.println(e);
        e.printStackTrace();
      }
    } catch(MalformedURLException mue) {
      System.err.println(mue);
      mue.printStackTrace();
    }
  }
}
```


©J.M.Joller

Und hier ist die Standard Ausgabe:

```
<HTML>
<HEAD>
<META HTTP-EQUIV="Content-Type" CONTENT="text/html; charset=windows-1252">
<TITLE>Jbuilder-Projekt Projekt56.jpr</TITLE>
</HEAD>
<BODY>
<H1>Projektdaten</H1>
<HR>
<FONT SIZE=+1>
<STRONG>Projekt:  </STRONG>Herunterladen einer Web Seite<BR>
<STRONG>Autor: </STRONG>J.M.Joller<BR>
<STRONG>Organisation: </STRONG><BR>
<STRONG>Beschreibung: </STRONG><BR>
Testen verschiedener URL Methoden:
URL Konstruktor
DataInputStream
aus einer Textseite
<HR>
<STRONG>Vormerken...</STRONG><BR>
</FONT>
<UL>
<!-- Bearbeiten Sie diesen Abschnitt, um die weiteren Aktivitäten verfolgen zu können -->
<LI>Eintrag 1
<LI>Eintrag 2
</UL>
</BODY>
</HTML>
```

Falls wir als URL ein Class File oder ein Bild eingeben, dann erhalten wir im Ausführungsprotokoll einen entsprechenden Eintrag:

```
Êþº¾
SourceFile_
openStream_
```

In diesem Falle müssten wir die Lesemethode anpassen.
Das Problem unterschiedliche Datentypen lesen zu müssen, kann man eleganter mit weiteren Methoden und Netzwerkklassen lösen.

#### 5.5.1.2. public URLConnection openConnection() throws java.io.IOException

Die Methode openConnection öffnet eine Socketverbindung zur URL Adresse und liefert ein URLConnection Objekt.

Wir werden in Kapitel 10 die Klasse URLConnection genauer untersuchen.

URLConnection liefert neben den Rohdaten (Bilder, HTML) auch die Protokoll Header.

#### 5.5.1.3. public final Object getContent() throws java.io.IOException

Die Methode getConnection ist eine andere Art und Weise, Daten von einem URL Objekt zu lesen. Das gute an dieser Klasse ist, dass das Objekt, welches zurück gegeben wird, je nach Datentyp angepasst wird:


©J.M.Joller

1. falls Text gelesen wird, beschreibt das Objekt ein Text Objekt (ASCII, HTML) und somit einen InputStream
2. falls ein Bild gelesen wird, wird ein Objekt geliefert, welches die Bilddaten interpretieren kann, zum Beispiel einen ImageProducer.

#### 5.5.1.4. public final Object getContent() throws java.io.IOException

Mit Hilfe der getContent() Methode können Daten aus einer URL gelesen werden. Die Methode versucht Daten von er URL Adresse zu lesen und inForm eines sinnvollen Objektes zurück zu geben.

Hier ein Beispiel Programm

##### 5.5.1.4.1. Programmbeispiel : Herunterladen eines Objektes

```
//Titel:       Herunterladen eines Objektes von einer URL
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  getContent()
//Einführendes Beispiel
package Beispiel57;

import java.net.*;
import java.io.*;

public class ProgrammBeispiel57 {

  public static void main(String[] args) {
    String strZeile;
    URL urlTest1, urlTest2;
    try {
      urlTest1 = new
URL("http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel57/Joller300x260
.gif");
      urlTest2 = new
URL("http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel57/BeispielProjek
t57.html");
      try {
        Object objTest = urlTest1.getContent();
        System.out.println("Vom URL "+urlTest1+" wurde folgendes Objekt
"+objTest.getClass().getName()+" herunter geladen");
      } catch(Exception e) {
        System.err.println(e);
        e.printStackTrace();
      }
      try {
        Object objTest = urlTest2.getContent();
```


©J.M.Joller

```
        System.out.println("Vom URL "+urlTest2+" wurde folgendes Objekt
"+objTest.getClass().getName()+" herunter geladen");
        System.out.println(" "+objTest.getClass().getName());
        System.out.println(" "+objTest.getClass());
        System.out.println(" "+objTest);
       } catch(Exception e) {
        System.err.println(e);
        e.printStackTrace();
       }
     } catch(MalformedURLException mue) {
      System.err.println(mue);
      mue.printStackTrace();
     }
   }
}
```

Und hier die Ausgabe:

```
Vom URL http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel57/Joller300x260.gif wurde folgendes Objekt
sun.awt.image.URLImageSource herunter geladen
Vom URL http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel57/BeispielProjekt57.html wurde folgendes
Objekt sun.net.www.MeteredStream herunter geladen
 sun.net.www.MeteredStream
 class sun.net.www.MeteredStream
 sun.net.www.MeteredStream@e32f0dc4
```

### 5.5.2. Verschiedene Hilfsmethoden der URL Klasse

Da wir nicht alle Methoden im Detail besprechen können, fassen wir einzelne wichtige Klassen zusammen.

#### 5.5.2.1. public boolean sameFile(URL urlAngabe)

Diese Methode macht genau das was Sie vermuten: sie liefert true, falls im Vergleich die zwei Dateien identisch sind. Wir müssen allerdings vorsichtig sein:

1. Die Dateien im URL http://www.suisse.ch/index.html und http://www.suisse.ch/index.html#a sind identisch
2. falls der Host http://www.suisse.ch und http://www.suisse.org die selben Hosts sind, wird dies nicht automatisch erkannt.

Schauen wir uns die Methode an einem konkreten Beispiel an.

##### 5.5.2.1.1. Beispiel : sameFile Methode

```
//Titel:        public boolean sameFile(URL u)
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Beispiel für den Vergleich zweier Dateien / URLs
package Beispiel58a;
import java.net.*;
public class ProgrammBeispiel58a {
```


©J.M.Joller

```
  public static void main(String[] args) {
    try {
      URL urlTest1 = new
URL("http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel57/BeispielProjek
t57.html");
      URL urlTest2 = new
URL("http://localhost/ParalleleUndVerteilteSysteme/Programme/Kapitel5/Beispiel57/BeispielProjek
t57.html#a");
      if (urlTest1.sameFile(urlTest2)) System.out.println("Die beiden Dateien sind identisch" );
      else System.out.println("Die beiden Dateien sind unterschiedlich");
    } catch (MalformedURLException mue) {
      System.err.println(mue);
      mue.fillInStackTrace();
    }
  }
}
```

Und hier die Ausgabe:

```
Die beiden Dateien sind identisch
```

#### 5.5.2.2. public boolean equals(Object o)

Ein Objekt ist gleich einem URL Objekt, falls
1. beide Objekte URL's sind
2. beide URL's zeigen auf die selbe Datei

Falls also zwei URL's gegeben sind, wie zum Beispiel
URL1 : "http://www.suisse.ch/startSeite.html" und
URL2 : "http://www.schweiz.org/startSeite.html"
auf dem selben Server, die selbe physische Seite sind, dann sind es trotzdem zwei unterschiedliche URL's.

### *5.6. Die URLEncoder, URLDecoder Klasse*

Die Abbildung der URL's auf die Betriebssysteme ist nicht immer einfach. Sie geschieht uneinheitlich.

Als Beispiel sei folgendes Problem erwähnt:
das *#* Zeichen wird im URL als ein Trennzeichen betrachtet. Es gibt aber viele Betriebssysteme, die das Zeichen # als Teil des Dateinamens erkennen könnten.

Daher muss geregelt werden, wie die Abbildund sinnvollerweise geschieht.

#### 5.6.1. URLEncoder

Diese Klasse enthält Hilfsmethoden, um eine Zeichenkette in ein MIME Format um zu wandeln, welches "x-www-form-urlencoded" genannt wird.

Die Konversion geschieht zeichenweise:


©J.M.Joller

Die ASCII Zeichen 'a' bis 'z', 'A' bis 'Z', und '0' bis '9' bleiben unverändert.
Das Leerzeichen ' ' wird in ein Plus Zeichen '+' umgewandelt.

Alle andern Zeichen werden in eine 3-stellige Zeichenkette "%xy" umgewandelt, wobei xy eine zweistellige hexadezimale Darstellung der unteren 8-Bits des Zeichens sind.

Diese Klasse gibt es seit JDK1.0

### 5.6.2. URLDecoder : public class URLDecoder extends Object

Diese Klasse enthält eine Hilfsmethode zum Konvertieren vom MIME Format "x-www-form-urlencoded" in eine Zeichenkette.

Die Konversion geschieht zeichenweise:

Die ASCII Zeichen 'a' bis 'z', 'A' bis 'Z', und '0' bis '9' bleiben unverändert.
Das Plus Zeichen '+'wird in ein Leerzeichen ' ' umgewandelt.

Die restlichen Zeichen werden in Form einer 3-Zeichen langen Zeichenkette dargestellt, welche mit einem Prozentzeichen anfängt, "%xy", wobei xy die zweistellige hexadezimale Darstellung der unteren 8-Bits des Zeichens sind.

Diese Klasse gibt es seit JDK1.2.

Betrachten wir ein Umwandlungsbeispiel

#### 5.6.2.1. URLEncoder Beispiel

```
//Titel:        URLEncoder / URLDecoder
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Umwandlung von Zeichenketten in das
//x-www-form-urlencoded  Format
package Beispiel59;
import java.net.*;
public class ProgrammBeispiel59 {

  public static void main(String[] args) {

    System.out.println("Test der URLEncoder.encode(String) Methode");
    System.out.println(URLEncoder.encode("Diese Zeichenkette hat Leerzeichen"));
    System.out.println(URLEncoder.encode("Diese*Zeichenkette*hat*Sterne"));
    System.out.println(URLEncoder.encode("Diese%Zeichenkette%hat%Prozent%Zeichen"));
    System.out.println(URLEncoder.encode("Diese+Zeichenkette+hat+Plus+Zeichen"));
    System.out.println(URLEncoder.encode("Diese/Zeichenkette/hat/Slashes"));

System.out.println(URLEncoder.encode("Diese\"Zeichenkette\"hat\"Anführungszeichen\"überall"));
    System.out.println(URLEncoder.encode("Diese:Zeichenkette:hat:Doppelpunkte"));
    System.out.println(URLEncoder.encode("Diese.Zeihenkette.hat.Punkte"));
    System.out.println(URLEncoder.encode("Diese=Zeichenkette=hat=Gleicheitszeichen=überall"));
```


©J.M.Joller

```
    System.out.println(URLEncoder.encode("Diese&zeichenkette&hat&Ampersand&Zeichen"));


  }
}
```

Und hier die Beispielausgabe:

```
Test der URLEncoder.encode(String) Methode
Diese+Zeichenkette+hat+Leerzeichen
Diese*Zeichenkette*hat*Sterne
Diese%25Zeichenkette%25hat%25Prozent%25Zeichen
Diese%2BZeichenkette%2Bhat%2BPlus%2BZeichen
Diese%2FZeichenkette%2Fhat%2FSlashes
Diese%22Zeichenkette%22hat%22Anf%FChrungszeichen%22%FCberall
Diese%3AZeichenkette%3Ahat%3ADoppelpunkte
Diese.Zeihenkette.hat.Punkte
Diese%3DZeichenkette%3Dhat%3DGleicheitszeichen%3D%FCberall
Diese%26zeichenkette%26hat%26Ampersand%26Zeichen
```

#### 5.6.2.2. URLDecoder Beispiel

Da die Decoder Methode seit Java 1.2 auch zur Verfügung steht, brauchen wir unser Beispiel nicht gross neu zu konstruieren. Als Eingabe Zeichenketten müssen wir das Ergebnis des obigen Encoder Beispiels einsetzen.

```
//Titel:        URLDecoder / URLDecoder
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Umwandlung von Zeichenketten in das
//x-www-form-urldecoded  Format
package Beispiel510;
import java.net.*;
public class ProgrammBeispiel510 {

  public static void main(String[] args) {
    try {
      System.out.println("Test der URLDecoder.decode(String) Methode");
      System.out.println(URLDecoder.decode("Diese Zeichenkette hat Leerzeichen"));
      System.out.println(URLDecoder.decode("Diese*Zeichenkette*hat*Sterne"));

System.out.println(URLDecoder.decode("Diese%25Zeichenkette%25hat%25Prozent%25Zeichen"));
      System.out.println(URLDecoder.decode("Diese%2BZeichenkette%2Bhat%2BPlus%2BZeichen"));
      System.out.println(URLDecoder.decode("Diese%2FZeichenkette%2Fhat%2FSlashes"));

System.out.println(URLDecoder.decode("Diese%22Zeichenkette%22hat%22Anf%FChrungszeichen
%22%FCberall"));
      System.out.println(URLDecoder.decode("Diese%3AZeichenkette%3Ahat%3ADoppelpunkte"));
      System.out.println(URLDecoder.decode("Diese.Zeihenkette.hat.Punkte"));

System.out.println(URLDecoder.decode("Diese%3DZeichenkette%3Dhat%3DGleicheitszeichen%3D
%FCberall"));
```


©J.M.Joller

```
System.out.println(URLDecoder.decode("Diese%26zeichenkette%26hat%26Ampersand%26Zeichen"
));
    } catch(Exception e) {
      System.err.println("Fehler im Decoding Test");
      System.err.println(e);
      e.printStackTrace();
    }

  }
}
```


©J.M.Joller

Das Ergebnis müssen, wie erwartet, die Ausgangs- Zeichenketten sein:

```
Test der URLDecoder.decode(String) Methode
Diese Zeichenkette hat Leerzeichen
Diese*Zeichenkette*hat*Sterne
Diese%Zeichenkette%hat%Prozent%Zeichen
Diese+Zeichenkette+hat+Plus+Zeichen
Diese/Zeichenkette/hat/Slashes
Diese"Zeichenkette"hat"Anführungszeichen"überall
Diese:Zeichenkette:hat:Doppelpunkte
Diese.Zeihenkette.hat.Punkte
Diese=Zeichenkette=hat=Gleicheitszeichen=überall
Diese&zeichenkette&hat&Ampersand&Zeichen
```

## *5.7. Einige hilfreiche Programme*

### 5.7.1. Web Seiten lokal abspeichern : Offline Reader Mark I

Wir wollen in den folgenden Kapiteln einen Offline Reader bauen und fangen schon mal mit dem einfachsten Fall an.
Die Aufgabe besteht darin, die relativen Adressen im HTML Text durch absolute Adressen zu ersetzen.
Das hat folgende Gründe:

1. wir werden die ursprüngliche Seiten vermutlich immer wieder finden, da wir einige absolute Adressen in unserer Webseite haben
2. wir müssen Media Objekte nicht herunterladen, da wir sie jederzeit mitHilfe der absoluten Adressen im Web holen können.

Natürlich macht dieses Verfahren ein Offline Lesen der Seite fast unmöglich, da wir ja die Anzahl Weblinks sogar noch erhöhen!

Hier ist das Programm. Der Programmtext ist so gut wie selbst erklärend.

```
import java.net.*;
import java.io.*;

public class ProgrammBeispiel512 {

  URL theURL;

  public static void main (String args[]) {

    // Loop through the command line arguments
    for  (int i = 0; i < args.length; i++) {

      //Open the URL for reading
      try {
        URL root = new URL(args[i]);
        ProgrammBeispiel512 ps = new ProgrammBeispiel512(root);
        ps.saveThePage();
      }
      catch (MalformedURLException e) {
        System.err.println(args[i] + " is not a parseable URL");
        System.err.println(e);
```


©J.M.Joller

```
    }
   } //  end for

 } // end main


 public ProgrammBeispiel512(URL u) {

   theURL = u;

 }

 // saveThePage opens a DataInputStream from the URL,
 // opens a PrintStream onto a file for the output,
 // and then copies one to the other while rewriting tags
 public void saveThePage() {

  char thisChar;
  String theTag;
  PrintStream  p = null;

  try {
    DataInputStream theHTML = new DataInputStream(theURL.openStream());
    p = makeOutputFile();

    while (true) {
      thisChar = (char) theHTML.readByte();
      if (thisChar == '<') {
        theTag = readTag(theHTML);
        theTag = convertTag(theTag);
        p.print(theTag);
      }
      else {
        p.print(thisChar);
      }
    } // end while
  }  // end try
  catch (EOFException e) {  // This page is done
  }
  catch (Exception e) {
     System.err.println(e);
  }
  finally {
    p.close();
  }

 }  // end SaveThePage


 // We need open a file on the local file system
 // with the same name as the remote file;
```


©J.M.Joller

```
  // then chain a PrintStream to the file
  public PrintStream makeOutputFile() throws IOException {

    FileOutputStream fout;

    String theFile = theURL.getFile();

    // the getFile method returns the filename prefixed with a slash,
    // e.g. /index.html instead of index.html. That slash needs to be removed.
    theFile = theFile.substring(1);
    System.err.println("\n\n\n" + theFile + "\n\n\n");
    if (theFile.equals("")) theFile = "index.html";

    // At this point you should check to see whether
    // the file already exists and, if it does,
    // ask the user if they wish to overwrite it

    fout = new FileOutputStream(theFile);

    return new PrintStream(fout);

  }

  // The readTag method is called when a < is encountered
  // in the input stream.  This method is responsible
  // for reading the remainder of the tag.
  // Note that when this method has been called the <
  // has been read from the input stream but has not yet been sent
  // to the output stream.
  // This method has trouble (as do most web browsers)
  // if it encounters a raw < sign in the Stream. Technically
  // raw < signs should be encoded as &lt; in the original HTML.
  public static 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();
```


©J.M.Joller

```
}

// The convertTag method takes a complete tag as
// a String and, if it's a relative link, converts it
// to an absolute link.  The converted tag is returned.
public String convertTag(String tag) {

  // temporary position variables
  int p1, p2, p3, p4;

  try {
    // HTML tags are cases insensitive so converting
    // it to upper case makes the problem slightly easier
    String s1 = tag.toUpperCase();
    // Find the beginning and the end of the URL
    //
    if (s1.startsWith("<A HREF")) {
      p1 = s1.indexOf("HREF");
    }
    else if (s1.startsWith("<IMG ")) {
      p1 = s1.indexOf("SRC");
    }
    else if (s1.startsWith("<APPLET ")) {
      p1 = s1.indexOf("CODEBASE");
    }
    else { // this is not a link based tag
      return tag;
    }
    // find the =
    p2 = s1.indexOf ("=", p1);
    if (p2 == -1) return tag;
    // Ideally the = sign is immediately followed by
    // a " mark followed by the URL which is closed by a ".
    // However since a lot of HTML is non-conforming we
    // need to be a little sneakier. In this case we read
    // characters in the URL until an character which is not
    // whitespace is encountered.
    p3 = p2+1;
    while (Character.isSpace(s1.charAt(p3))) {
      p3++;
    }
    if (s1.charAt(p3) == "") p3++;

    // p3 now points to the beginning of the URL
    // The URL is read until a closing " or whitespace is seen
    p4 = p3+1;
    while (!Character.isSpace(s1.charAt(p4)) &&
     s1.charAt(p4) != "") {
      p4++;
    }
```


©J.M.Joller

```
      // The URL is the text between p3 and p4
      // URL's are in general NOT case insensitive so the URL
      // must be read from the original tag and not from s1
      // which was uppercased
      String link = tag.substring(p3, p4);

      // Is it a relative URL?  Relative URLs
      // don't contain colons.
      if (link.indexOf(":")  == -1) {
           // build an absolute URL from the relative URL
           URL newURL = new URL(theURL, link);
           // replace the old URL with the new URL
           tag = s1.substring(0,p3) + newURL + s1.substring(p4,s1.length());
        }  // end if

    }  // end try
    catch (StringIndexOutOfBoundsException e) {
      // Most of the time a StringIndexOutOfBoundsException here means
      // the tag was not standard conforming so
      // the algorithm for finding the URL crapped out.
      // If that's the case, the original tag is returned.
    }
    catch (Exception e) {
      System.err.println(e);
    }


    return tag;

  }

}
```

Die Start URL können Sie im JBuilder unter Start eingeben. Dort haben Sie die Möglichkeit Parameter, also zum Beispiel eine URL, anzugeben.
Da ich lokal arbeite und einen HTTP Server installiert habe, ist der gesamte Zugriff lokal.

Die Ausgabe ist in meinem Fall (Ausführungsprotokoll)
ProjektBeispiel512.html

Dies ist der Start Parameter, die URL, die herunter geladen werden soll.

Interessant am Programm ist die Rekursion, sobald Tags gefunden wurden.


©J.M.Joller

### 5.7.2. URLRequestor Applet

Als weiteres Beispiel hier ein Applet, welches den HTML Text liest und in einem Fenster anzeigt.

Das Applet ist wie folgt definiert:

```
package Beispiel513;

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

public class ProgrammBeispiel513 extends Frame {

  TextField url;
  TextArea textDisplay;

  public static void main(String[] args) {

    ProgrammBeispiel513 a = new ProgrammBeispiel513();
    a.show();

  }

  public ProgrammBeispiel513() {

    super("URL Requestor");
    resize(500,300);
    move(50,50);
    init();

  }

  public void init() {

    textDisplay = new TextArea();
    add("Center", textDisplay);
    // We don't want the buttons and fields in the north and south
    // to fill their respertive sections so we'll add Panels there
    // and use FlowLayout's in the Panels
    Panel SouthPanel = new Panel();
    Panel NorthPanel = new Panel();
    NorthPanel.setLayout(new FlowLayout(FlowLayout.LEFT));
    SouthPanel.add(new Button("Get URL"));
    NorthPanel.add("North", new Label("URL: "));
```

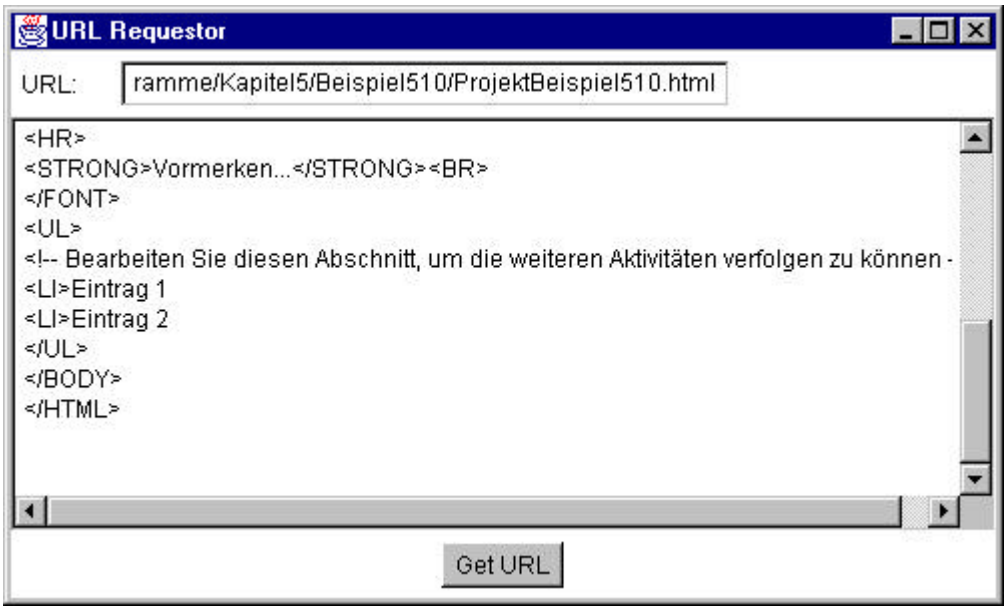

©J.M.Joller

```
    url = new TextField(40);
    NorthPanel.add("North", url);
    add("South", SouthPanel);
    add("North", NorthPanel);

  }

  public boolean action(Event e, Object o) {

    if (e.target instanceof Button) {
      fetchURL(url.getText());
      return true;
    }
    else if (e.target == url) {
      fetchURL(url.getText());
      return true;
    }
    else {
      return false;
    }

  }

 public void fetchURL(String s) {

   try {
     URL u = new URL(s);
     try {
       Object o = u.getContent();
       if (o instanceof InputStream)  {
         showText((InputStream) o);
       }
       else {
         showText(o.toString());
       }
     }
     catch (IOException e) {
       showText("Could not connect to " + u.getHost());
     }
     catch (NullPointerException e) {
       showText("There was a problem with the content");
     }
   }
   catch (MalformedURLException e) {
     showText(url.getText() + " is not a valid URL");
   }

 }

  void showText(String s) {
    textDisplay.setText(s);
```


©J.M.Joller

```
 }

 void showText(InputStream is) {

   String nextline;

   textDisplay.setText("");
   try {
     DataInputStream dis = new DataInputStream(is);
     while((nextline = dis.readLine()) != null) {
       textDisplay.appendText(nextline + "\n");
     }
   }
   catch (IOException e) {
     textDisplay.appendText(e.toString());
   }

 }

}
```

Als Beispiel erhalten wir auf dem Textfenster:





©J.M.Joller

### *5.8. Aufgaben*

Versuchen Sie im Team die zwei grossen Programme (Web Page Downloader und das obige Programm) zu vervollständigen:

1. das Web Downloader Programm soll mit dem obigen Programm so kombiniert werden, dass die Start URL, die herunter geladen werden soll, in einer Maske, wie oben, eingegeben werden kann.
2. verbessern Sie das obige Programm und das graphische Download Programm so, dass das Fenster problemlos geschlossen werden kann.


©J.M.Joller

## 5.9. Anhang : Die URL Klasse

*Java Platform 1.2*

**Class**

PREV CLASS NEXT CLASS FRAMES NO FRAMES
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

### 5.9.1. java.net
### Class URL

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

public final class **URL**
extends Object
implements Serializable

Class `URL` represents a Uniform Resource Locator, a pointer to a "resource" on the World Wide Web. A resource can be something as simple as a file or a directory, or it can be a reference to a more complicated object, such as a query to a database or to a search engine. More information on the types of URLs and their formats can be found at:

```
http://www.ncsa.uiuc.edu/demoweb/url-primer.html
```

In general, a URL can be broken into several parts. The previous example of a URL indicates that the protocol to use is `http` (HyperText Transport Protocol) and that the information resides on a host machine named `www.ncsa.uiuc.edu`. The information on that host machine is named `demoweb/url-primer.html`. The exact meaning of this name on the host machine is both protocol dependent and host dependent. The information normally resides in a file, but it could be generated on the fly. This component of the URL is called the *file* component, even though the information is not necessarily in a file.
A URL can optionally specify a "port", which is the port number to which the TCP connection is made on the remote host machine. If the port is not specified, the default port for the protocol is used instead. For example, the default port for `http` is `80`. An alternative port could be specified as:

```
http://www.ncsa.uiuc.edu:8080/demoweb/url-primer.html
```

A URL may have appended to it an "anchor", also known as a "ref" or a "reference". The anchor is indicated by the sharp sign character "#" followed by more characters. For example,

```
http://java.sun.com/index.html#chapter1
```

This anchor is not technically part of the URL. Rather, it indicates that after the specified resource is retrieved, the application is specifically interested in that part of the document that has the tag `chapter1` attached to it. The meaning of a tag is resource specific.
An application can also specify a "relative URL", which contains only enough information to


©J.M.Joller

reach the resource relative to another URL. Relative URLs are frequently used within HTML pages. For example, if the contents of the URL:

```
http://java.sun.com/index.html
```

contained within it the relative URL:

```
FAQ.html
```

it would be a shorthand for:

```
http://java.sun.com/FAQ.html
```

The relative URL need not specify all the components of a URL. If the protocol, host name, or port number is missing, the value is inherited from the fully specified URL. The file component must be specified. The optional anchor is not inherited.

**Since:**

JDK1.0

**See Also:**

Serialized Form

---

### 5.9.2. Constructor Summary

| | |
|---|---|
| **URL**(String spec)<br>Creates a URL object from the String representation. | |
| **URL**(String protocol, String host, int port, String file)<br>Creates a URL object from the specified protocol, host, port number, and file. | |
| **URL**(String protocol, String host, int port, String file, URLStreamHandler handler)<br>Creates a URL object from the specified protocol, host, port number, file, and handler. | |
| **URL**(String protocol, String host, String file)<br>Creates an absolute URL from the specified protocol name, host name, and file name. | |
| **URL**(URL context, String spec)<br>Creates a URL by parsing the specification spec within a specified context. | |
| **URL**(URL context, String spec, URLStreamHandler handler)<br>Creates a URL by parsing the specification spec within a specified context. | |


©J.M.Joller

### 5.9.3. Method Summary

| | |
|---|---|
| boolean | **equals**(Object obj)<br>Compares two URLs. |
| Object | **getContent**()<br>Returns the contents of this URL. |
| String | **getFile**()<br>Returns the file name of this URL. |
| String | **getHost**()<br>Returns the host name of this URL, if applicable. |
| int | **getPort**()<br>Returns the port number of this URL. |
| String | **getProtocol**()<br>Returns the protocol name this URL. |
| String | **getRef**()<br>Returns the anchor (also known as the "reference") of this URL. |
| int | **hashCode**()<br>Creates an integer suitable for hash table indexing. |
| URLConnection | **openConnection**()<br>Returns a URLConnection object that represents a connection to the remote object referred to by the URL. |
| InputStream | **openStream**()<br>Opens a connection to this URL and returns an InputStream for reading from that connection. |
| boolean | **sameFile**(URL other)<br>Compares two URLs, excluding the "ref" fields. |
| protected void | **set**(String protocol, String host, int port, String file, String ref)<br>Sets the fields of the URL. |
| static void | **setURLStreamHandlerFactory**(URLStreamHandlerFactory fac)<br>Sets an application's URLStreamHandlerFactory. |
| String | **toExternalForm**()<br>Constructs a string representation of this URL. |
| String | **toString**()<br>Constructs a string representation of this URL. |

### 5.9.4. Methods inherited from class java.lang.Object

clone, finalize, getClass, notify, notifyAll, wait, wait, wait


©J.M.Joller

### 5.9.5. Constructor Detail

**URL public** URL**(String protocol, String host, int port, String file) throws MalformedURLException**

Creates a `URL` object from the specified `protocol`, `host`, `port` number, and `file`. Specifying a `port` number of `-1` indicates that the URL should use the default port for the protocol.
If this is the first URL object being created with the specified protocol, a *stream protocol handler* object, an instance of class `URLStreamHandler`, is created for that protocol:

1. If the application has previously set up an instance of `URLStreamHandlerFactory` as the stream handler factory, then the `createURLStreamHandler` method of that instance is called with the protocol string as an argument to create the stream protocol handler.
2. If no `URLStreamHandlerFactory` has yet been set up, or if the factory's `createURLStreamHandler` method returns `null`, then the constructor finds the value of the system property:

   ```
   java.protocol.handler.pkgs
   ```

   If the value of that system property is not `null`, it is interpreted as a list of packages separated by a vertical slash character '`|`'. The constructor tries to load the class named:

   ```
   <package>.<protocol>.Handler
   ```

   where <*package*> is replaced by the name of the package and <*protocol*> is replaced by the name of the protocol. If this class does not exist, or if the class exists but it is not a subclass of `URLStreamHandler`, then the next package in the list is tried.
3. If the previous step fails to find a protocol handler, then the constructor tries to load the class named:

   ```
   sun.net.www.protocol.<protocol>.Handler
   ```

   If this class does not exist, or if the class exists but it is not a subclass of `URLStreamHandler`, then a `MalformedURLException` is thrown.

**Parameters:**

`protocol` - the name of the protocol.
`host` - the name of the host.
`port` - the port number.
`file` - the host file.

**Throws:**

MalformedURLException - if an unknown protocol is specified.

**See Also:**

`System.getProperty(java.lang.String)`, `setURLStreamHandlerFactory(java.net.URLStreamHandlerFactory)`, `URLStreamHandler`, `URLStreamHandlerFactory.createURLStreamHandler(java.lang.String)`

---


©J.M.Joller

### URL : public URL(String protocol, String host, String file) throws MalformedURLException

Creates an absolute URL from the specified `protocol` name, `host` name, and `file` name. The default port for the specified protocol is used.
This method is equivalent to calling the four-argument constructor with the arguments being `protocol`, `host`, `-1`, and `file`.
**Parameters:**
`protocol` - the protocol to use.
`host` - the host to connect to.
`file` - the file on that host.
**Throws:**
MalformedURLException - if an unknown protocol is specified.
**See Also:**
`URL(java.lang.String, java.lang.String, int, java.lang.String)`

---

### URL : public URL(String protocol, String host, int port, String file, URLStreamHandler handler) throws MalformedURLException

Creates a `URL` object from the specified `protocol`, `host`, `port` number, `file`, and `handler`. Specifying a `port` number of `-1` indicates that the URL should use the default port for the protocol. Specifying a `handler` of `null` indicates that the URL should use a default stream handler for the protocol, as outlined for:

```
java.net.URL#URL(java.lang.String, java.lang.String, int,
                 java.lang.String)
```

If the handler is not null and there is a security manager, the security manager's `checkPermission` method is called with a `NetPermission("specifyStreamHandler")` permission. This may result in a SecurityException.
**Parameters:**
`protocol` - the name of the protocol.
`host` - the name of the host.
`port` - the port number.
`file` - the host file.
`handler` - the stream handler.
**Throws:**
MalformedURLException - if an unknown protocol is specified.
SecurityException - if a security manager exists and its `checkPermission` method doesn't allow specifying a stream handler explicitly.
**See Also:**
`System.getProperty(java.lang.String)`, `setURLStreamHandlerFactory(java.net.URLStreamHandlerFactory)`, `URLStreamHandler`, `URLStreamHandlerFactory.createURLStreamHandler(java.lang.String)`, `SecurityManager.checkPermission(java.security.Permission)`, `NetPermission`

---

### URL : public URL(String spec)throws MalformedURLException

Creates a `URL` object from the `String` representation.


©J.M.Joller

This constructor is equivalent to a call to the two-argument constructor with a `null` first argument.
**Parameters:**
`spec` - the `String` to parse as a URL.
**Throws:**
MalformedURLException - If the string specifies an unknown protocol.
**See Also:**
`URL(java.net.URL, java.lang.String)`

---

### URL : public URL(URL context, String spec)throws MalformedURLException

Creates a URL by parsing the specification `spec` within a specified context. If the `context` argument is not `null` and the `spec` argument is a partial URL specification, then any of the strings missing components are inherited from the `context` argument.
The specification given by the `String` argument is parsed to determine if it specifies a protocol. If the `String` contains an ASCII colon ':' character before the first occurrence of an ASCII slash character '/', then the characters before the colon comprise the protocol.

- If the `spec` argument does not specify a protocol:
  - If the context argument is not `null`, then the protocol is copied from the context argument.
  - If the context argument is `null`, then a `MalformedURLException` is thrown.
- If the `spec` argument does specify a protocol:
  - If the context argument is `null`, or specifies a different protocol than the specification argument, the context argument is ignored.
  - If the context argument is not `null` and specifies the same protocol as the specification, the `host`, `port` number, and `file` are copied from the context argument into the newly created `URL`.

The constructor then searches for an appropriate stream protocol handler of type `URLStreamHandler` as outlined for:

```
java.net.URL#URL(java.lang.String, java.lang.String, int,
                 java.lang.String)
```

The stream protocol handler's `parseURL` method is called to parse the remaining fields of the specification that override any defaults set by the context argument.
**Parameters:**
`context` - the context in which to parse the specification.
`spec` - a `String` representation of a URL.
**Throws:**
MalformedURLException - if no protocol is specified, or an unknown protocol is found.
**See Also:**
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler`, `URLStreamHandler.parseURL(java.net.URL, java.lang.String, int, int)`

---

### URL : public URL(URL context, String spec, URLStreamHandler handler) throws MalformedURLException

Creates a URL by parsing the specification `spec` within a specified context. If the `context`


©J.M.Joller

argument is not `null` and the `spec` argument is a partial URL specification, then any of the strings missing components are inherited from the `context` argument.
The specification given by the `String` argument is parsed to determine if it specifies a protocol. If the `String` contains an ASCII colon ':' character before the first occurrence of an ASCII slash character '/', then the characters before the colon comprise the protocol.

- If the `spec` argument does not specify a protocol:
  - If the context argument is not `null`, then the protocol is copied from the context argument.
  - If the context argument is `null`, then a `MalformedURLException` is thrown.
- If the `spec` argument does specify a protocol:
  - If the context argument is `null`, or specifies a different protocol than the specification argument, the context argument is ignored.
  - If the context argument is not `null` and specifies the same protocol as the specification, the `host`, `port` number, and `file` are copied from the context argument into the newly created `URL`.

If the argument `handler` is specified then it will be used as the stream handler for the URL and will override that of the context. Specifying a stream handler requires the NetPermission `"specifyStreamHandler"` or a `SecurityException` will be thrown.
Otherwise, if `handler` is null and the context is valid then the protocol handler of the context will be inherited. The stream protocol handler's `parseURL` method is called to parse the remaining fields of the specification that override any defaults set by the context argument.

**Parameters:**

`context` - the context in which to parse the specification.
`spec` - a `String` representation of a URL.
`handler` - the stream handler for the URL.

**Throws:**

MalformedURLException - if no protocol is specified, or an unknown protocol is found.
SecurityException - if a security manager exists and its `checkPermission` method doesn't allow specifying a stream handler.

**See Also:**

`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler`, `URLStreamHandler.parseURL(java.net.URL, java.lang.String, int, int)`


©J.M.Joller

### 5.9.6. Method Detail

#### set

```
protected void set(String protocol,
                   String host,
                   int port,
                   String file,
                   String ref)
```

Sets the fields of the URL. This is not a public method so that only URLStreamHandlers can modify URL fields. URLs are otherwise constant.

**Parameters:**

`protocol` - the protocol to use
`host` - the host name to connecto to
`port` - the protocol port to connect to
`file` - the specified file name on that host
`ref` - the reference

---

#### getPort

```
public int getPort()
```

Returns the port number of this `URL`. Returns -1 if the port is not set.

**Returns:**

the port number

---

#### getProtocol

```
public String getProtocol()
```

Returns the protocol name this `URL`.

**Returns:**

the protocol of this `URL`.

---

#### Host

```
public String getHost()
```

Returns the host name of this `URL`, if applicable. For "`file`" protocol, this is an empty string.

**Returns:**

the host name of this `URL`.

---


©J.M.Joller

### getFile

`public String getFile()`
Returns the file name of this `URL`.
**Returns:**
the file name of this `URL`.

---

### getRef

`public String getRef()`
Returns the anchor (also known as the "reference") of this `URL`.
**Returns:**
the anchor (also known as the "reference") of this `URL`.

---

### equals

`public boolean equals(Object obj)`
Compares two URLs. The result is `true` if and only if the argument is not `null` and is a `URL` object that represents the same `URL` as this object. Two URL objects are equal if they have the same protocol and reference the same host, the same port number on the host, and the same file and anchor on the host.
**Parameters:**
`obj` - the URL to compare against.
**Returns:**
`true` if the objects are the same; `false` otherwise.
**Overrides:**
equals in class Object

---

### hashCode

`public int hashCode()`
Creates an integer suitable for hash table indexing.
**Returns:**
a hash code for this `URL`.
**Overrides:**
hashCode in class Object

---


©J.M.Joller

### sameFile

`public boolean` **`sameFile`**`(URL other)`
Compares two URLs, excluding the "ref" fields. Returns `true` if this `URL` and the `other` argument both refer to the same resource. The two `URL`s might not both contain the same anchor.
**Parameters:**
`other` - the `URL` to compare against.
**Returns:**
`true` if they reference the same remote object; `false` otherwise.

---

### toString

`public String` **`toString`**`()`
Constructs a string representation of this `URL`. The string is created by calling the `toExternalForm` method of the stream protocol handler for this object.
**Returns:**
a string representation of this object.
**Overrides:**
toString in class Object
**See Also:**
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler.toExternalForm(java.net.URL)`

---

### toExternalForm

`public String` **`toExternalForm`**`()`
Constructs a string representation of this `URL`. The string is created by calling the `toExternalForm` method of the stream protocol handler for this object.
**Returns:**
a string representation of this object.
**See Also:**
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandler.toExternalForm(java.net.URL)`

---


©J.M.Joller

## openConnection

```
public URLConnection openConnection()
                           throws IOException
```

Returns a `URLConnection` object that represents a connection to the remote object referred to by the `URL`.
A new connection is opened every time by calling the `openConnection` method of the protocol handler for this URL.
If for the URL's protocol (such as HTTP or JAR), there exists a public, specialized URLConnection subclass belonging to one of the following packages or one of their subpackages: java.lang, java.io, java.util, java.net, the connection returned will be of that subclass. For example, for HTTP an HttpURLConnection will be returned, and for JAR a JarURLConnection will be returned.
**Returns:**
a `URLConnection` to the URL.
**Throws:**
IOException - if an I/O exception occurs.
**See Also:**
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLConnection`, `URLStreamHandler.openConnection(java.net.URL)`

---

## openStream

```
public final InputStream openStream()
                           throws IOException
```

Opens a connection to this `URL` and returns an `InputStream` for reading from that connection. This method is a shorthand for:

```
openConnection().getInputStream()
```

**Returns:**an input stream for reading from the URL connection.
**Throws:**
IOException - if an I/O exception occurs.
**See Also:** `openConnection()`, `URLConnection.getInputStream()`

---

## getContent

```
public final Object getContent()
                       throws IOException
```

Returns the contents of this URL. This method is a shorthand for:
openConnection().getContent()
**Returns:**
the contents of this URL.
**Throws:**
IOException - if an I/O exception occurs.
**See Also:**
`URLConnection.getContent()`

---


©J.M.Joller

## setURLStreamHandlerFactory

`public static void` **`setURLStreamHandlerFactory`**`(URLStreamHandlerFactory fac)`
Sets an application's `URLStreamHandlerFactory`. This method can be called at most once in a given Java Virtual Machine.
The `URLStreamHandlerFactory` instance is used to construct a stream protocol handler from a protocol name.
If there is a security manager, this method first calls the security manager's `checkSetFactory` method to ensure the operation is allowed. This could result in a SecurityException.

**Parameters:**
`fac` - the desired factory.

**Throws:**
Error - if the application has already set a factory.
SecurityException - if a security manager exists and its `checkSetFactory` method doesn't allow the operation.

**See Also:**
`URL(java.lang.String, java.lang.String, int, java.lang.String)`, `URLStreamHandlerFactory`, `SecurityManager.checkSetFactory()`

---

**Class**

*Java Platform 1.2*

**PREV CLASS** **NEXT CLASS** **FRAMES** **NO FRAMES**
SUMMARY: INNER | FIELD | CONSTR | METHOD
DETAIL: FIELD | CONSTR | METHOD

---

Submit a bug or feature Version 1.2 of Java Platform API Specification
Java is a trademark or registered trademark of Sun Microsystems, Inc. in the US and other countries.
Copyright 1993-1998 Sun Microsystems, Inc. 901 San Antonio Road,
Palo Alto, California, 94303, U.S.A. All Rights Reserved.

## *5.10. Anhang : Lösungen zu den Selbsttestaufgaben*

### 5.10.1. Protokolle

Die Aufgabe bestand darin, ein Testprogramm zu schreiben, mit dessen Hilfe geprüft werden kann, welche Protokolle vom URL Konstruktor, also von der URL Klasse unterstützt werden.

Hier eine mögliche Lösung:

```
//Titel:        URL Klassen Konstruktor
//Version:
//Copyright:    Copyright (c) 1999
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  einfaches Programm, welches untersucht, welche Protokolle unterstützt werden.
package UnterstuetzteProtokolle;
import java.net.*;
class Selbsttestaufgabe5211 {
  public static void main(String args[]) {
    //String protocols[] = new String[10]; oder
    String protocol[]={"http","mailto","news","gopher","ftp", "telnet"};
    System.out.println("Länge der Zeichenkette protocol : "+protocol.length);
    for (int i=0; i<protocol.length; i++) {
      try {
        System.out.println("Protokoll "+protocol[i]);
        URL u = new URL(protocol[i], "www.switzerland.org", "/Joller/testPage.html#URL");
      } catch (MalformedURLException mue) {
          System.out.println("Protokoll "+protocol[i]+" führt zu einem Ausnahmezustand!");
          System.err.println(mue.getMessage());
      }
    }
  }
}
```

Dieses Programm liefert folgende Ausgabe:

```
Länge der Zeichenkette protocol : 6
Protokoll http
Protokoll mailto
Protokoll news
Protokoll news führt zu einem Ausnahmezustand!
unknown protocol: news
Protokoll gopher
Protokoll ftp
Protokoll telnet
Protokoll telnet führt zu einem Ausnahmezustand!
unknown protocol: telnet
```


©J.M.Joller

©J.M.Joller

©J.M.Joller