In diesem Kapitel:
- *Was ist ein Protokoll Handler?*
- *Schreiben eines URLStreamHandler*
- *Schreiben eines Protokoll Handler*
- *URLStreamHandler Factories*
- *Protokoll Handler Beispiele und Techniken*

11. *Protokoll Handler*

11.1. Was ist ein Protokoll Handler?

Bei der Frage geht es darum, ob es in einer sinnvollen Art und Weise möglich ist, einen Browser zu bauen, der sich selbstständig erweitert. Immer wenn ein Inhalt daher kommt, der vom Browser nicht interpretiert werden kann, holt sich der Browser die zusätzlich benötigten Handler selbstständig aus dem Web.

Sun Microsystems hat eine generelle Aufteilung dieser Aufgabe in die zwei Teile vorgenommen:
- Protokoll Handler
- Inhalt / Content Handler

Protokoll Handling besteht in der korrekten Wechselwirkung zwischen dem Client und dem Server:
- generieren der korrekten Requests / Anfragen im vordefinierten Format
- Interpretation der Header, die mit den Daten zurück gesendet werden
- Bestätigen, dass die Daten empfangen wurden
- ...

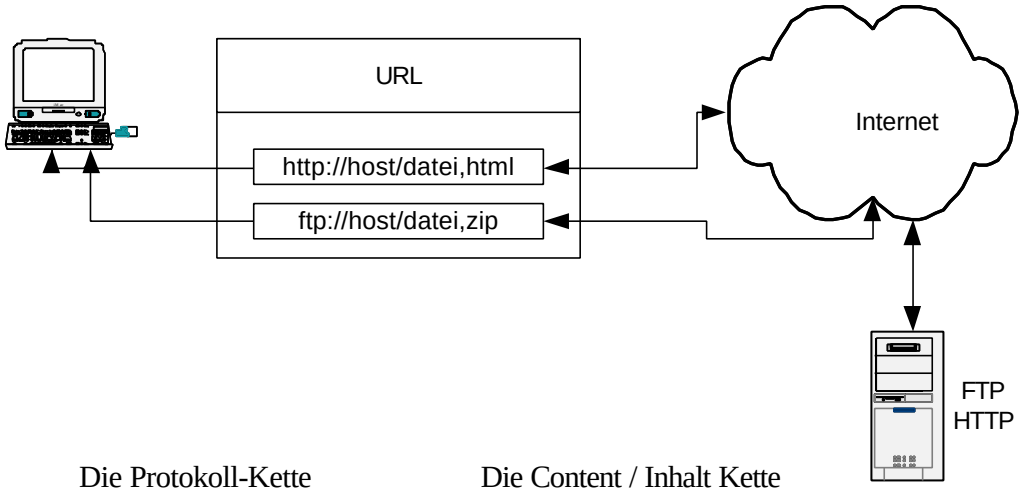
Inhalt Handling besteht im sinnvollen Bearbeiten und Interpretieren der empfangenen Daten
- ein GIF Bild anzeigen
- ...

Es scheint also durchaus sinnvoll, die zwei Aufgaben zu separieren:
- einem GIF Bild ist es egal wie es zum Client gelangt, ob mit HTTP, FTP, ...

Dem Protokoll Handler ist es egal, was er transportiert:
- ein MP3, ein GIF Bild, ..

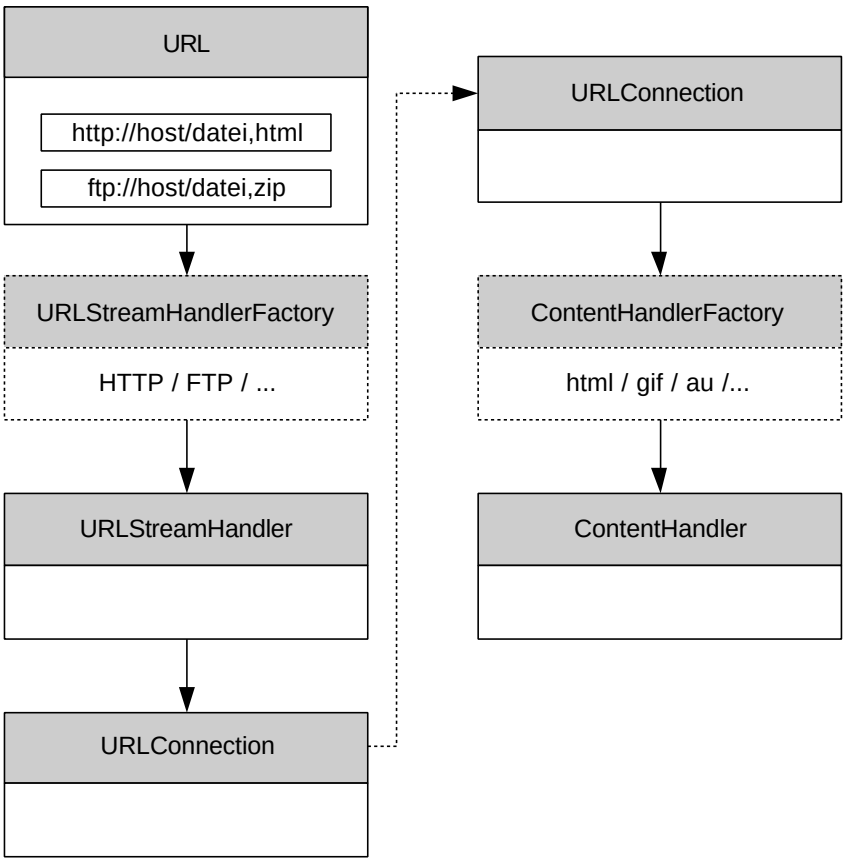
11.2. Illustration der Klassen URL, URLConnection, StreamHandler und ContentHandler

Anschaulich lassen sich die Klassen im URL Umfeld etwas besser verstehen und einordnen. Deswegen hier eine Zusammenstellung dieser Klassen, deren Zusammenwirkung und auch deren Bedeutung im Kommunikationsumfeld von Java:

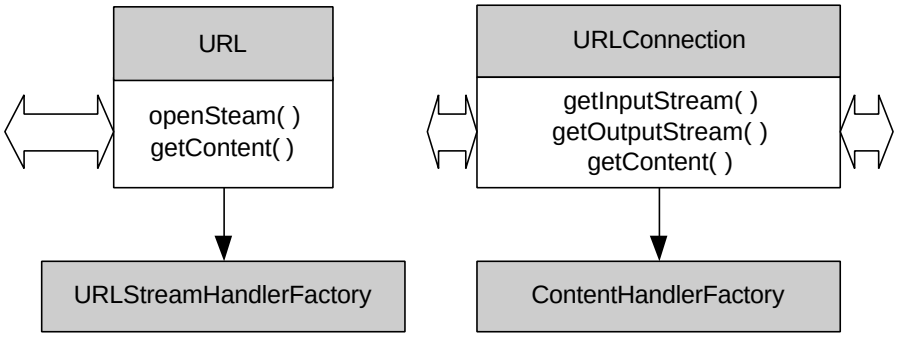


Die Protokoll-Kette

Die Content / Inhalt Kette



Aus dem obigen Schema wird auch der wesentliche Unterschied zwischen den Klassen URL und URLConnection klar:



11.3. Ziel : schreiben eines **eigenen** *Protokolls und Implementation dieses Protokolls mit Java. Schrittweises Vorgehen*

Wie sollen wir dabei vorgehen?

1. durchsuchen der Java Dokumentation : bietet Java bereits eine solche Klasse?
2. falls ja : wie wird diese eingesetzt, um mein Protokoll definieren und implementieren zu können?
3. wenn nein : bietet Java irgend welche Hilfsklassen an.

11.4. Klassen in java.net, mit deren Hilfe Protokolle implementiert werden

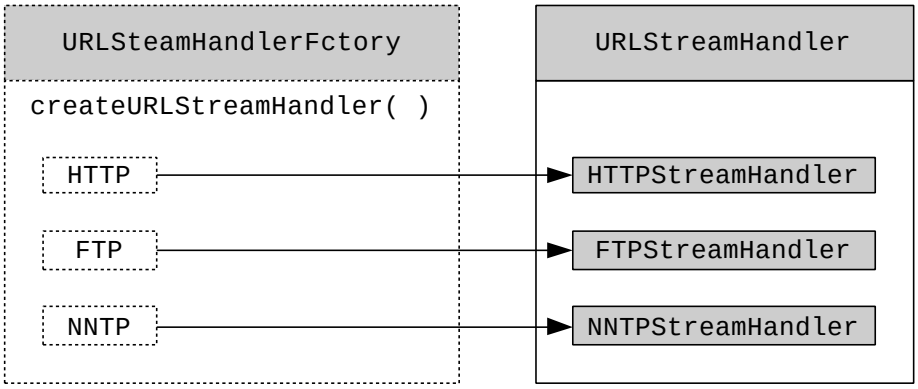
Java kennt ein Interface, die URLStreamHandlerFactory, eine Objekt Fabrik, mit der Stream Handler generiert werden können.
Dies reicht aber nicht aus!
Grund:

1. `URLStreamHandlerFactory` ist ein Interface, muss also implementiert werden
   Die Details des Protokolls müssen also von Hand implementiert werden.
2. zur Implementation braucht man weitere Java Klassen, zusätzlich zum implementierten Interface `URLStreamHandler`:

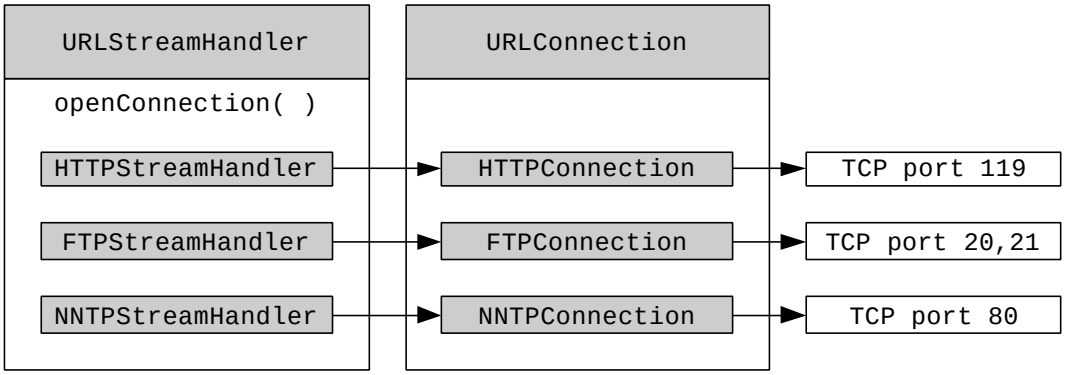
- `URL`
- `URLConnection`
- `URLStreamHandler`

Schauen wir uns dies Klassen beziehungsweise. das Interface URLStreamHandlerFactory an (die Klassen URL, URLConnection kennen wir schon). Ein guter Hinweis ist sicher auch das Internet: verschiedene Protokolle wurden durch unabhängige Firmen implementiert. Die Protokolle, welche von Sun mitgeliefert werden, sind immer unter sun.... zu finden, das heisst aber, dass diese eigentlich kein fixer Bestandteil des JDK sind!

In diesem Kapitel geht es um die Frage des Protokoll Handlers. Dieser ist mit dem Stream Handler verknüpft, wie das folgende Diagramm zeigt. Im folgenden Kapitel werden wir uns dann mit dem Dateninhalt, dem Content, beschäftigen.



Vom StreamHandler führt eine direkte Verbindung zur URLConnection:



11.4.1. java.net Interface URLStreamHandlerFactory

public interface **URLStreamHandlerFactory**

Dieses Interface definiert eine Factory Entwurfsmuster, also eine Methode, Objekte zu kreieren, ohne die Details dieses Prozesses darzulegen. In Java wird eine Factory durch ein Interface repräsentiert. In unserem Fall nutzen wir die Schnittstelle, um URL Stream Protokoll Handler zu definieren.

Dieses wird von der URL Klasse benutzt, um einen URLStreamHandler für ein spezifisches Protokoll zu definieren.

Seit:

JDK1.0

Siehe Auch:

URL, URLStreamHandler

11.4.1.1. Methoden Übersicht

URLStreamHandler	**createURLStreamHandler**(String protocol) Kreiert eine neue URLStreamHandler Instanz mit dem spezifizierten Protokoll.

11.4.1.2. Methoden Details

11.4.1.2.1. createURLStreamHandler

`public URLStreamHandler createURLStreamHandler(String protocol)`

Kreiert eine neue URLStreamHandler Instanz mit dem spezifizierten Protokoll.

Parameter:

protocol - das Protokoll ("ftp", "http", "nntp", etc.).
Achtung: nicht dass wir uns falsch verstehen: einige der Protokolle sind implementiert; aber die URLStreamHandlerFactory ist ein Interface, *muss* also implementiert werden! Dabei müssen alle Methoden eines Interfaces implementiert werden.

Liefert:

einen URLStreamHandler für das spezifizierte Protokoll.

Siehe Auch:

java.io.URLStreamHandler

11.4.2. Welche Protokolle werden bereits unterstützt?

Sie haben oder sollten in Kapitel 5 in einer Selbsttestaufgabe untersuchen, welche Protokolle im URL Konstruktor unterstützt werden.

Eine Java Lösung der Aufgabe finden Sie bei den Programmen zu Kapitel 5 (Selbsttestaufgabe) :

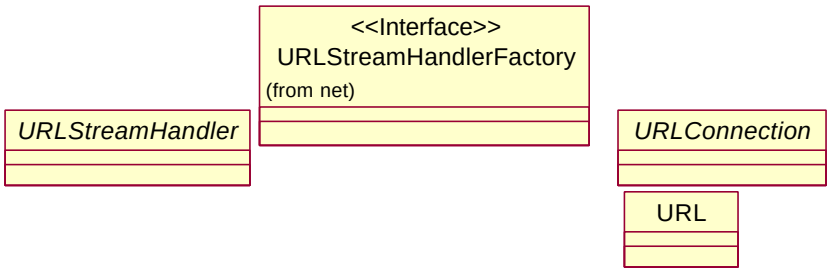
```
//Titel:       URL Klassen Konstruktor
//Beschreibung:  einfaches Programm, welches untersucht, welche Protokolle unterstützt werden.
package UnterstuetzteProtokolle;
import java.net.*;
class Selbsttestaufgabe5211 {
  public static void main(String args[]) {
    //String protocols[] = new String[10]; oder
    String protocol[]={"http","mailto","news","gopher","ftp", "telnet"};
    System.out.println("Länge des Arrays 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()); } } }}
mit folgender Ausgabe:

```
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
```

Aber selbst wenn das Protokoll ftp als bekannt erkannt wird, heisst dies noch nicht, dass dieses Protokoll auch implementiert ist in Java. Vielleicht wird das noch geschehen.

Sonst muss man es einfach selber machen, wie das so geht, werden wir jetzt sehen.



Zur Wiederholung: (im obigen UML Diagramm) alle Klassen mit kursiver Schrift sind abstrakte Klassen, müssen also implementiert werden.

Einzig die URL Klasse ist schon direkt nutzbar. Diese enthält auch eine Spezifikation, den Namen, des Protokolls:

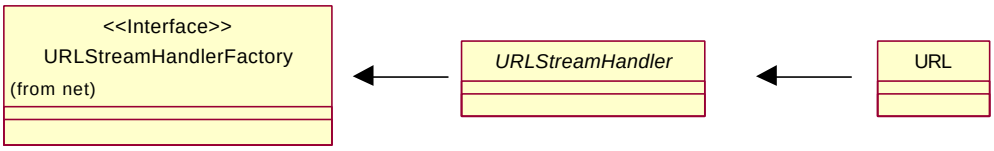
protocol://hostname[:port]/path/filename#section

Also könnte man ein eigenes Protokoll im gleichen Stil einsetzen:

meinProtokoll://Host[:Port]/Pfad

Gemäss Definition des URL Konstruktors versucht dieser ein *stream protocol handler* Objekt, eine Instanz der Klasse URLStreamHandler, für dieses Protokoll zu kreieren.

Ein Stream Protocol Handler muss erkennen, wie ein konkretes Protokoll wie http, ftp, oder gopher implementiert ist. In der Regel werden Instanzen der URLStreamHandler Subklasse nicht direkt durch eine Applikation kreiert. Vielmehr wird der Stream Protokoll Handler automatisch geladen, wenn zum ersten Mal ein bestimmtes Protokoll spezifiziert wird.



Die Stream (Protokoll) Handler Klasse verwendet diese Schnittstelle, um einen URLStreamHandler für ein spezifisches Protokoll zu generieren.

Nachdem der URL klar ist, wie sie mit dem Protokoll umgehen muss, besteht das nächste Problem im Schaffen einer Verbindung mit Hilfe einer Unterklasse von URLConnection, die weiss, wie man mit diesem Protokoll umgehen muss.

Zwei Standard- Unterklassen sind : HttpURLConnection, JarURLConnection, Klassen also, die sich speziell mit HTTP oder Jar (ZIP ähnliches Java Archiv; welches mit WinZIP geöffnet werden kann.).

Die zwei Klassen, abcURLConnection und die zum Protokoll abc gehörende Stream Handler Klasse, gehören zusammen, wie Zwillinge!

Die Funktion der URL Klasse wurde schon im Kapitel über URL's beschrieben. Erstaunlich an den URL Beispielen war, dass man URL Objekte kreieren konnte, mit beliebig verrückten Angaben, ohne dass eine Fehlermeldung generiert wurde.
Der Grund ist der, dass die URL Klasse eigentlich URLConnection und URLStreamHandler Methoden einsetzt, falls diese geladen wurden.

Falls wir also ein neues Protokoll implementieren möchten, müssen wir Unterklassen dieser Klassen bilden.

11.5. Ablauf zum Implementieren eines neuen Protokolls

Damit sieht der Ablauf, zum Implementieren eines neuen Protokolls wie folgt aus:

1. schreiben einer URLConnection und einer URLStreamHandler Klasse
   Eventuell muss auch das URLStreamHandlerFactory Interface implementiert werden.

   Die URLConnection Klasse ist für die Kommunikation mit dem Server zuständig, konvertiert alles, was der Server sendet, in einen InputStream; und zusätzlich konvertiert die Klasse alles was an den Server gesendet werden muss, in einen OutputStream. Unter Umständen müssen auch noch die Methoden getInputStream(), getOutputStream() und getContentType() in der Unterklasse überschrieben werden.
   Die Unterklasse muss zudem die abstrakte connect() Methode implementieren.

   Die Unterklasse der URLStreamHandler Klasse zerlegt die URL Beschreibung und kreiert ein neues URLConnecion Objekt, welches das angegebene Protokoll versteht.
   Diese Unterklasse muss zudem die abstrakte openConnectio() Methode implementieren.
   Falls die Spezifikation des neuen Protokolls nicht HTTP ähnlich spezifiziert werden kann, dann müssen auch noch die Methoden parseURL() und toExternalForm() überschrieben werden.

2. eventuell schreiben einer Klasse, welche die URLStreamHandlerFactory Schnittstelle implementiert. Diese Klasse muss der Applikation helfen, den korrekten Protokoll Handler zu finden und zu laden. Die Schnittstelle besitzt lediglich eine Methode : createURLStreamHandler(). Diese liefert ein URLStreamHandler Objekt zurück.

   Die Default Java Stream Handler von Sun Microsystems werden nach folgendem Schema benannt:
   sun.net.www.protocol.protocol.*Handler*
   zum Beispiel:

sun.net.www.protocol.http.Handler

Die neu gebaute URLStreamHandler Factory wird mit folgendem Programm Fragment an die URL angebunden:

```
URL.setURLStreamHandlerFactory(new meineURLStreamHF() );
```

11.5.1. Zusammenfassung : Ablauf

Hier der Ablauf im Einzelnen:

1. Die Applikation versucht ein URL Objekt zu konstruieren, mit Hilfe der Klasse java.net.URL().
2. Der Konstruktor bestimmt das spezifizierte Protokoll, zum Beispiel http
3. Der URL Konstruktor versucht einen URLStreamHandler für das gegebene Protokoll zu finden.

   Falls das Protokoll schon im Einsatz war, dann wird das Stream Handler Objekt aus dem Cache gelesen.

   Sonst wird geprüft, ob eine Factory gesetzt wurde (wie oben mit URL.setURLStreamHandlerFactory(....)).
   Falls ja, dann wird die Angabe zum Protokoll an die Methode createURLStreamHandler() weiter gereicht.

   Falls das Protokoll noch nicht im Einsatz war, dann versucht der URL Konstruktor ein Objekt der Klasse
   sun.net.www.*protocol*.Handler
   zu kreieren.

4. Die Applikation ruft die Methode openConnection() des URL Objektes auf.
5. Das URL Objekt verlangt vom URLStreamHandler ein URLConnection Objekt zu öffnen.
6. Mit Hilfe der URLConnection Klasse wird die Kommunikation mit dem remote Objekt (URL) mit Hilfe des Protokolls *protocol* abgewickelt.

Dieser Ablauf ist allerdings nicht stur nur genau so anwendbar.
Zum Beispiel:
bei Schritt 4 hätte man als Alternative die Möglichkeit
URL.getContent() oder URL.getInputStream() zu verwenden.

11.6. Schreiben eines URLStreamHandlers

Die abstrakte Klasse URLStreamHandler ist die Oberklasse für alle Klassen, die spezifische Protokolle, wie HTTP, ... abhandeln. Die Methoden dieser Klasse werden in der Regel nicht direkt, sondern versteckt hinter Methoden der Klasse URL oder der Klasse URLConnection. Eigene Protokolle kann man also einführen, indem man die Methoden der Klasse URLStreamHandler überschreibt.

11.6.1. java.net Class URLStreamHandler

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

public abstract class **URLStreamHandler**
extends Object

Die abstrakte Klasse URLStreamHandler ist die allgemeine Oberklasse für alle Stream Protokoll Handler. Ein Stream Protokoll Handler kennt, wie eine Verbindung für ein bestimmtes Protokoll, wie http, ftp, oder gopher geschehen muss.
In den meisten Fällen wird eine Instanz einer URLStreamHandler Unterklasse nicht direkt von einer Applikation kreiert.
Das erste Mal, wenn ein Protokoll Name beim Konstruieren eines URL angetroffen wird, wird der entsprechende Stream Protokoll Handler automatisch geladen.

Seit:

JDK1.0

Siehe Auch:

URL.URL(java.lang.String, java.lang.String, int, java.lang.String)

11.6.1.1. Konstruktor Übersicht

URLStreamHandler()	

11.6.1.2. Methoden Übersicht

protected abstract URLConnection	**openConnection**(URL u) Öffnet eine Verbindung zum Objekt, welches als URL Argument übergeben wird.
protected void	**parseURL**(URL u, String spec, int start, int limit) Zerlegt die Zeichenkette, die das URL Objekt u beschreibt.
protected void	**setURL**(URL u, String protocol, String host, int port, String file, String ref) Setzt die Felder des URL Argumentes wie angegeben.
protected String	**toExternalForm**(URL u) Konvertiert eine URL für ein spezifisches Protokoll in eine Zeichenkette.

11.6.1.3. Methoden, geerbt von class java.lang.Object

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

11.6.1.4. Konstruktor Detail

URLStreamHandler

```
public URLStreamHandler()
```

Erläuterungen:
Ein URLStreamHandler Objekt wird nicht direkt erzeugt. Vielmehr beauftragt Java beim Kreieren einer URL, für die noch kein Stream Handler existiert (weil das Protokoll zum ersten Mal auftaucht), die URLStreamHandlerFactory einen dem Protokoll angepassten StreamHandler zu kreieren.

11.6.1.5. Methoden Detail

openConnection

```
protected abstract URLConnection openConnection(URL u)
                                         throws IOException
```

Öffnet eine Verbindung zu dem Objekt, welches als URL Argument übergeben wird. Diese Methode sollte in einer Unterklasse überschrieben werden..
Falls für das Handler Protokoll (wie HTTP oder JAR) eine public, spezialisierte URLConnection Unterklasse existiert, wie im Falle der folgenden Packages oder einer der Subpackages: java.lang, java.io, java.util, java.net. In diesem Falle ist die Verbindung vom Typus dieser Unterklasse. Zum Beispiel: für HTTP wird eine HttpURLConnection und für JAR eine JarURLConnection zurück geliefert.

Parameter:
u - die URL mit der verbunden wird.

Liefert:
ein URLConnection Objekt für diese URL.

Throws:
IOException - falls ein I/O Fehler auftritt beim Öffnen dieser Verbindung.

Erläuterungen:
Die Methode ist protected, wird also in der Regel nicht direkt aufgerufen. Sie wird von der Methode openConnectio() der URL Klasse eingesetzt.
Die Implementierung ist so, dass normalerweise diese Methode den Konstruktor der Klasse URLConnection() aufruft.

11.6.1.5.1. Programm Fragment openConnection()

```
protected URLConnection openConnection(URL u) throws IOException {
        return new mailtoURLConnection(u);
}
```

Ein vollständigeres Beispiel würde etwa wie folgt aussehen:

11.6.1.5.2. Ein mailto URLStreamHandler

```
//Titel:       ein mailto Stream Handler
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Überschreiben der Methoden :
//URLConnection()
//parseURL()
//toExternalForm()
//package Beispiel11_1;

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

public class Handler extends java.net.URLStreamHandler {

  protected URLConnection openConnection(URL u) throws IOException {
    return new mailtoURLConnection(u);  // muss noch implementiert werden
  }

  protected void parseURL(URL u, String spec, int start, int limit) {

    StringTokenizer st = new StringTokenizer(spec.substring(start), ":@", false);
    String protocol = st.nextToken();   // mailto
    String file = st.nextToken();       // username
    String host = st.nextToken();
    String ref = null;
    int port = 25;

    setURL(u, protocol, host, port, file, ref);

   }

  protected String toExternalForm(URL u) {

    return "mailto:" + u.getFile() + "@" + u.getHost();;

  }

}
```

parseURL

protected void **parseURL**(URL u,
String spec,
int start,
int limit)

Zerlegt die Zeichenkettendarstellung eines URL in ein URL Objekt.
Die parseURL Methode eines URLStreamHandler zerlegt die Zeichenkette als ob es sich dabei um eine http Spezifikation handeln würde. Die meisten URL Protokoll Familien sind ähnlich aufgebaut. Falls ein Stream Protokoll Handler für ein Protokoll eine andere Syntax besitzt, dann muss diese Methode überschrieben werden.

Parameter:

u - die URL, welche durch die Elemente der zerlegten Zeichenkette beschrieben wird.
spec - die Zeichenkettendarstellung der URL , welche zerlegt werden muss.
start - der Zeichen Index ab dem die Zerlegung beginnen muss.
limit - das Zeichen, bis zu dem zerlegt werden muss.

Erläuterungen:

Die Parser Methode zerlegt die URL Beschreibung in die Bestandteile, in der Regel um anschliessend bestimmte URL Felder gezielt setzen zu können (mit setURL).
Die Methode setzt voraus, dass der Aufbau des URL im Wesentlichen wie folgt aussieht:

protocol://host:port/file#ref

Das trifft auf die Protokolle mailto, gopher, ftp, ... zu; unter Umständen aber nicht auf Protokolle, die Sie selber definieren. In diesen Fällen müssen Sie eine eigene Parse Methode implementieren, welche eventuell für bestimmte Teile die Methode parseURL einsetzt. In diesem Falle würde man die Methode parseURL als super.parse.URL(...) aufrrufen.

Betrachten wir als Beispiel mailto:
URL Beispiel : mailto:meinName@meinProvider.com.
Wie könnte man diese URL abbilden auf eine URL mit Protokoll, Port, Datei, Host und Ref Feld?
Das Protokoll ist : mailto.
Das Datei Feld fehlt. Wir verwenden dieses Feld, um den Benutzernamen abzuspeichern.
Port und Ref werden auf null beziehungsweise Default gesetzt.
Alles was nach dem @ steht ist der Host.

11.6.1.5.3. Programm Fragment : parseURL()

```
protected void parseURL(URL u, String spec, int start, int limit) {
        StringTokenizer st = new StringTokenizer(spec. substring(start), ":@", false);
        String protocol = st.nextToken();       // mailto
        String file = st.nextToken();           // Username
        String host = st.nextToken();
        String ref = null;
        int port = 25;                          // Default

        setURL(u, protocol, host, port, file, ref);
}
```

Als Alternative zu diesem Vorgehen könnte man die Felder in eigenen Datenfeldern in einer Unterklasse von URLStreamHandler definieren. Der Nachteil ist der, dass man dann die Standard Methoden wie URL.getFile(), nicht einsetzen kann.

11.6.1.5.4. Programm Fragment : parseURL() für mailto

```
String username;
protected void parseURL(URL u, String spec, int start, int limit) {
        StringTokenizer st = new StringTokenizer(spec, substring(start), ":@", false);
        String protocol = st.nextToken();      // mailto
        username = st.nextToken();
        String host = st.nextToken();
        String file = null;
        String ref = null;
        int port = 25;                         // Default
        setURL(u, protocol, host, port, file, ref);
}
```

toExternalForm

`protected String` **`toExternalForm`**`(URL u)`

konvertiert einen URL in eine Zeichenkette.

Parameter:

u - die URL.

Liefert:

eine Zeichenkettendarstellung des URL's

Erläuterungen:

Die Methode setzt die Zeichenkette zusammen, welche den URL beschreibt.
Falls die Methode parseURL() überschrieben wird, dann sollte auch diese Routine neu definiert werden.

11.6.1.5.5. Programm Fragment : toExternalForm()

```
protected String toExternalForm(URL u) {
        return "mailto:"+u.getFile()+"@"+u.getHost():
}
```

.

setURL

```
protected void setURL(URL u,
              String protocol,
              String host,
              int port,
              String file,
              String ref)
```

Setzt die Felder eines URL Arguments gemäss den angegebenen Werten.
Parameter:
u - die zu modifizierende URL.
protocol - der Protokoll Name.
host - der remote Host Wert dieses URL.
port - der Port an der remote Maschine.
file - die Datei.
ref - die Referenz.
Siehe Auch:
URL.set(java.lang.String, java.lang.String, int, java.lang.String, java.lang.String)

Die Hauptverantwortung des Protokoll Handlers ist die Zerlegung einer Zeichenkettendarstellung eines URL in seine Bestandteile und die "Verarbeitung" der Bestandteile.

Die Methode parseURL() zerlegt die URL Zeichenkette in die URL Bestandteile;
die Methode setURL() setzt die Werte des URL, häufig direkt in der Methode parseURL().

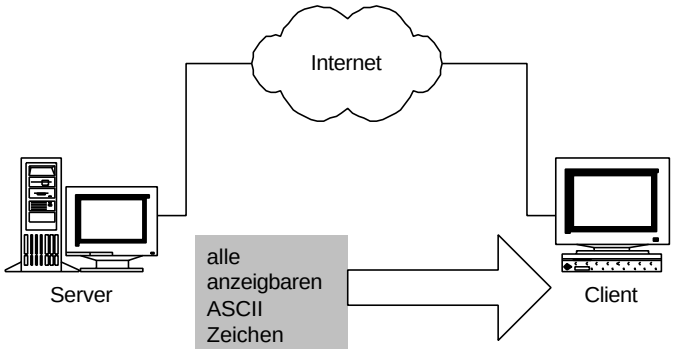
Nachdem wir die Bausteine kennen, widmen wir uns der Implementation eines einfachen Protokoll Handlers.

11.7. Schreiben eines Protokoll Handlers

Wir betrachten der Einfachheit ein sehr einfaches Protokoll, das chargen Protokoll, welches im RFC864 beschrieben wird. Das Protokoll wurde ursprünglich geschrieben, um Clients gezielt testen zu können.

Der Server wartet auf Port 19 auf eine Anfrage.

Falls ein Client eine Verbindung aufbaut, dann sendet der Server eine endlose Zeichenkette bis der Client die Verbindung abbricht.



Der Sender muss ja nur irgend etwas senden, was ist egal. Also bietet es sich an, die druckbaren ASCII Zeichen zu senden. Damit das gesendete Muster besser lesbar ist, sendet man jeweils 72 Zeichen pro Zeile. Insgesamt stehen 95 Zeichen zur Verfügung. Man rotiert einfach die Zeichen, wie in einem Schieberegister (man schiebt einfach jeweils um ein Zeichen nach links).

Alle Eingaben vom Client werden ignoriert.

Für das Chargen Protokoll existiert keine URL Präsentation. Wir definieren einfach ein neue Darstellung:

chargen://hostname:port

Der Content Type des Chargen Protokolls ist reiner ASCII Text. Die Methode getContentType() Methode muss also text/plain zurück liefern.

Java versteht diesen MIME Typ bereits; wir brauchen also nichts spezielles implementieren.

11.7.1. Programmbeispiel : chargenURLConnection Klasse

```
//Titel:       chargenURLConnection()
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Erweiterung der Klasse URLConnection zur Klasse
chargenURLConnection()
//Definition des Konstruktors
//Methoden:
//getInputStream()
//getContentType()
//connect()
//package Beispiel11_2;
```

```
package sun.net.www.protocol.chargen;
import java.net.*;
import java.io.*;

public class chargenURLConnection extends URLConnection {

  Socket theConnection = null;
  public final static int defaultPort = 19;

  public chargenURLConnection(URL u) {
    super(u);
  }

  public synchronized InputStream getInputStream() throws IOException {

    if (!connected) {
      connect();
    }
    return theConnection.getInputStream();

  }

  public String getContentType() {
    return "text/plain";
  }

  public synchronized void connect() throws IOException {

    int port;

    if (!connected) {
      port = url.getPort();
      if ( port < 0) {
        port = defaultPort;
      }
      theConnection = new Socket(url.getHost(), port);
      connected = true;
    }
   }
}
```

11.7.2. Erläuterungen

Die Verbindung wird, wie üblich, mit Hilfe von Sockets hergestellt:

theConnection = new Socket(url.getHost(), port);

Das zweite Datenfeld der Klasse ist der Default Port des Chargen Protokolls (Port 19).
Dies ist eine Konstante:

```
public final static int defaultPort = 19;
```

Die Struktur des Konstruktors ist denkbar einfach:

```
public chargenURLConnection(URL u) {
        super(u);
}                                                    .
```

Die connect() Methode öffnet die Verbindung zum spezifizierten Server:

```
public synchronized void connect() throws IOException {

  int port;

  if (!connected) {
    port = url.getPort();
    if ( port < 0) {
      port = defaultPort;
    }
    theConnection = new Socket(url.getHost(), port);
    connected = true;
  }

}
```

Das Socket Objekt wird im Datenfeld *theConnection* abgespeichert und von der Methode *getInputStream()* benutzt*:*

```
public synchronized InputStream getInputStream() throws IOException {

  if (!connected) {
    connect();
  }
  return theConnection.getInputStream();

}
```

Die Methode ist synchronisiert, damit nicht gleichzeitig mehrere "Benutzer" die Methode aufrufen können.

Die Methode *getContentType()* liefert eine Zeichenkette, in welche der MIME Typ für die Daten geschrieben werden.

```
public String getContentType() {
  return "text/plain";
}
```

Die Methode *getInputStream()* liefert wie oben bereits erwähnt, den InputStream des Socket Objektes, welches im Datenfeld *theConnection* steht.

```
public synchronized InputStream getInputStream() throws IOException {

  if (!connected) {
    connect();
  }
  return theConnection.getInputStream();

}
```

Damit die Handler Klasse von den Systemroutinen von java.net gefunden werden kann, muss der Paketnamen gemäss der Sun Konvention gewählt werden:

sun.net.www.protocol.chargen

Schauen wir uns noch ein Programmbeispiel für einen chargen Handler an:

11.7.3. Programmbeispiel : chargen Handler

```
//Titel:       chargen Handler
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Beispiel für eine Handler Klasse für das Chargen Protokoll

package Beispiel11_3;

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

public class chargenURLStreamHandler extends URLStreamHandler {

  protected URLConnection openConnection(URL u) throws IOException {
    return new chargenURLConnection(u);
  }

}
```

Das Problem ist der Pfad, der von der Package Angabe erzeugt wird. Dieser muss in den CLASSPATH aufgenommen werden.

11.8. Factories für URLStreamHandler

Im letzen Abschnitt haben wir, auf Grund der Package Angabe, ein neues Protokoll dem sun Package hinzugefügt.

Eleganter ist die Lösung mit Hilfe einer Factory, mit Hilfe des *URLStreamHandlerFactory* Interfaces. Diese Lösung ist allerdings nur für Applikationen erlaubt. Applets müssen die Standard URLStreamHandlerFactory benutzen, sonst wird eine *SecurityException* geworfen.

Die Standard URLStreamHandlerFactory schaut einfach im sun.net.www.*protocol* Package nach, ob eine Klasse *sun.net.www*.`protocol`.*Handler.class* existiert. protocol steht dabei für http, ftp, jar...

11.8.1. Installation eines URLStreamHandlers

Ein URLStreamHandler ist als abstrakte Klasse definiert:

```
public abstract class URLStreamHandler
extends Object
```

Diese Klasse besitzt lediglich einen default Konstruktor.

Weiter vorne haben wir die Schnittstelle URLStreamHandlerFactory bereits beschrieben:
`public interface URLStreamHandlerFactory`
Diese verfügt über eine Methode

```
public URLStreamHandler
createURLStreamHandler(String protocol)
```

die einen URLStreamHandler zu einem bestimmten Protokoll kreieren kann.

11.8.1.1. Vorgehen

Das Vorgehen sieht demnach wie folgt aus:

1. wir müssen eine Klasse definieren, die das Interface URLStreamHandlerFactory implementiert.
2. wir müssen die Methode `createURLHandler()` Methode so implementieren, dass sie einen Protokollhandler für das bestimmte Protokoll findet. Das ist aber recht einfach: im wesentlichen besteht dies in einer Verknüpfung mit den Definitionsklassen des Protokolls, welches wir implementiert haben. Falls das Protokoll einfach zu definieren ist, besteht dessen Definition eventuell nur aus einer Methode in einer anderen Klasse.

Falls keine URLStreamHandlerFactory gesetzt wurde, sucht Java nach einer Klasse mit dem Namen sun.net.www.*protocol*.Handler, wobei *protocol* unser neuer Protokollname ist.

Falls innerhalb der Applikation kein Handler für das Protokoll definiert wurde, dann sollte die Methode `createURLHandler()` den "Wert" `null` liefern. Damit wird dem Java Laufzeitsystem mitgeteilt, dass es sich um einen Default Protokoll Handler handelt, oder einen Handler, der sich in der Default Lokation befindet.

Die URLStreamHandlerFactory wird mit Hilfe der URL Klasse gesetzt:
diese verfügt über eine Methode setURLStreamHandlerFactory() und sollte zu Beginn des Programmes aufgerufen werden.

Schauen wir uns ein einfaches Beispiel an:

11.8.1.2. Der ChargenURLStreamHandler : Codefragment

```
//Titel:       chargen URLStreamHandler
//Version:
//Copyright:    Copyright (c) 1999
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  Fragment eines Stream Handlers
//für das Chargen Protokoll
package Beispiel11_4;

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

//public class KlasseBeispiel11_4 extends URLStreamHandler{
public class chargenURLStreamHandler extends URLStreamHandler {
  protected URLConnection openConnection(URL u) throws IOException {
    return new chargenURLConnection(u);
  }
}
```

11.8.1.3. Deklarieren der Applikation und Implementation der URLStreamHandlerFactory

Nachdem wir den Stream Handler definiert haben, müssen wir die eigentliche Applikation definieren. Diese muss das Interface URLStreamHandlerFactory implementieren.

Wir sehen gleich ein vollständiges Beispiel! Hier beschränken wir uns zuerst einmal auf das Vorgehen.

11.8.1.4. Implementieren der createURLStreamHandler Methode : Codefragment

```
public URLStreamHandler createURLStreamHandler(String protocol) {

        if (protocol.equalsIgnoreCase("chargen")) {
                return new chargenURLStreamHandler();
        } else {
                return null;
```

```
        }

}
```

Die Methode ist so gut wie selbsterklärend.
Wichtig ist, dass die Spezifikation des Character Generators wie folgt aussehen kann:

1. chargen://meinHost.com
2. CHARGEN://meinHost.com

also unabhängig von der Grossschreibung sein soll. Dafür wird ignoreCase eingesetzt.

11.8.1.5. Ein chargen Applet

```
//Titel:       char generator Protokoll
//Version:
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Organisation:
//Beschreibung: als Applet :
//läuft eigentlich nicht, wegen der Autorisierung / dem Sicherheitsmodell
package Beispiel11_5;

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

public class chargenApplet extends Applet implements URLStreamHandlerFactory, Runnable
{

  TextArea theText;
  URL theServer;
  DataInputStream dis = null;

public static void main(String[] args) {

    Frame f = new Frame("char Gen applet");
    f.resize(300, 300);
    f.move(50, 50);
    chargenApplet cg = new chargenApplet();

    f.add("Center", cg);
    cg.init();
    f.show();
    cg.start();

  }
```

```
  public void init() {

    URL.setURLStreamHandlerFactory(this);
    setLayout(new BorderLayout());
    theText = new TextArea();
    add("Center", theText);
    String s = "chargen://sunsite.unc.edu/";
    try {
      theServer = new URL(s);
      dis = new DataInputStream( theServer.openStream());
    }
    catch (MalformedURLException e) {
      theText.setText("Fehler: URL Problem " + s);
    }
    catch (IOException e) {
      theText.setText("Fehler: Verbindung zu " + s + " konnte nicht hergestellt werden");
      theText.appendText("Vielleicht läuft der chargen Server nicht, " +
        "auf diesem  Host, oder der Zugriff ist nicht gestattet.");
    }
    Thread t = new Thread(this);
    t.start();
  }

public URLStreamHandler createURLStreamHandler(String protocol) {
    if (protocol.equalsIgnoreCase("chargen")) {
      return new chargenURLStreamHandler();
    }
    else {
      return null;
    }
  }
public void run() {
    try {
      String theLine;
      if (dis != null) {
        while ((theLine = dis.readLine()) != null) {
          theText.appendText(theLine + "\r");
        }
      }
    }
    catch (IOException e) {
    }
  }
}
```

11.9. Weitere Protokoll Handler Beispiele und Techniken

Weitere Protokoll Handler lassen sich nach dem gleichen Schema implementieren:

1. definieren Sie eine URL für das Protokoll, sofern die Standardform der URL nicht ausreicht.
   Es bietet sich an, die URL so ähnlich wie möglich zu einer HTTP Form zu definieren.
2. legen Sie fest, welcher MIME Type von der Methode `getContentType`() des Protokoll Handlers zurück gegeben werden soll. `text/plain` ist in vielen Fällen ein brauchbarer MIME Type. Andere Möglichkeiten sind reines HTML, welches dann in der `getInputStream`() in `text/plain` umgewandelt wird.
   Die `guessContentTypeFromStream()` oder `guessContentTypeFromName()` Methoden können in Einzelfällen behilflich sein.
   Es empfiehlt sich den MIME Type für das Protokoll explizit fest zu halten.
3. kreieren Sie eine Subklasse von `URLConnection()` so, dass sie das Protokoll versteht.
   Die Klasse sollte die `connect()` implementieren und folgende Methoden überschreiben:
   `getContentType(), getOutputStream(), getInputStream()` (als Methoden der Klasse `URLConnection()`).
4. kreieren Sie eine Subklasse der `URLStreamHandler()` Klasse, mit einer `openConnection()` Methode, welche weiss wie eine Instanz der neuen Subklasse von `URLConnection()` kreiert werden kann.
   Falls der URL nicht HTTP ähnlich aussieht, dann müssen auch noch die Methoden `parseURL(), toExternalForm()` implementiert werden.
5. implementieren Sie das `URLStreamHandlerFactory` Interface und die `createStreamHandler()` Methode.

Schauen wir uns ein weiteres konkretes Beispiel an : das `daytime` Protokoll

11.9.1. daytime

Als erstes definieren wir die Syntax des `daytime` Protokolls:

`daytime://host:port/<Rest wird ignoriert>`
`Beispiel` : daytime://sunsite.switch-cnlab.ch

Das Protokoll ist also analog zu HTTP definiert. Wir werden die Standard- Methoden `toExternalForm(), parseURL()` einsetzen können.

Da das Protokoll `text/plain` liefert, könnten wir den Standard MIME Type einsetzen. Wir wollen aber den Text in HTML umwandeln und eine HTML Seite zurück liefern. Das Protokoll liefert also `text/html` zurück.
Wir werden alle Daten aus dem `Connection` Objekt lesen, den Text analysieren, den Text umformatieren in eine Zeichenkette und einen `StringBufferInputStream` zurück liefern. In diesem stehen dann die umformatierten Daten.

Schauen wir uns die `URLConnection` Klasse an:

11.9.1.1. Die daytime URLConnection Klasse

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

public class daytimeURLConnection extends URLConnection {
  Socket theConnection = null;
  public final static int defaultPort = 13;
  public daytimeURLConnection (URL u) {
    super(u);
  }
  public synchronized InputStream getInputStream() throws IOException {
    if (!connected) {
      connect();
    }
    DataInputStream dis = new DataInputStream(theConnection.getInputStream());
    String time = dis.readLine();
    String html = "<html><head><title>Die Zeit am Server " +
      url.getHost() + "</title></head><body><h1>" +
      time + "</h1></body></html>";
    return new StringBufferInputStream(html);
  }
  public String getContentType() {
    return "text/html";
  }
  public Object getContent() throws IOException {
    return getInputStream();
  }
```

```
  public synchronized void connect() throws IOException {
    int port;
    if (!connected) {
      port = url.getPort();
      if ( port < 0) {
        port = defaultPort;
      }
      theConnection = new Socket(url.getHost(), port);
      connected = true;
    }
  }
}
```

Wie ist die Klasse aufgebaut?

1. Die Klasse hat zwei Datenfelder:
   `theConnection`, ein Socket für die Verbindung zwischen dem Client und dem Server;
   `defaultPort`, der Standard Port für das daytime Protokoll (Port 13).
2. Der Konstruktor ist nicht speziell. Er ruft den Konstruktor der Oberklasse auf (`super`).
   Das Argument des Konstruktors ist die `URL u`.
3. Die `connect`() Methode öffnet eine Verbindung zum spezifizierten Server und dem spezifizierten Port, eventuell dem Standard Port, falls keiner spezifiziert wurde.
   Falls die Verbindung erfolgreich ist, dann wird die Boole'sche Variable `connected = true.`
   Der Socker, der geöffnet wurde, wird im Datenfeld `theConnection` abgespeichert und kann später von der Methode `getInputStream()` benutzt werden.
4. Die `getContentType()` Methode liefert eine Zeichenkette, welche den MIME Type beschreibt. Da die `getInputStream()` Methode den Text in HTML umformatiert, ist der MIME Type `text/html`.
5. Die `getInputStream()` Methode liefert den `InputStream` , wie der Name sagt. Falls kein Socket existiert, dann ruft die Methode die `connect`() Methode auf, um eine Verbindung herzustellen. Die Zeit wird gelesen und ein neuer `StringBufferInputStream` generiert.

Jetzt benötigen wir die StreamHandler Klasse:

11.9.1.2. Die daytimeURLStreamHandler Klasse

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

public class daytimeURLStreamHandler extends URLStreamHandler {

  protected URLConnection openConnection(URL u) throws IOException {
    return new daytimeURLConnection(u);
  }

}
```

Als Letztes müssen wir nun das Interface `URLStreamHandlerFactory` implementieren. Dies können wir gleich im eigentlichen Hauptprogramm machen.

11.9.1.3. Ein erweiterbarer Protokoll Tester

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

public class ProtocolTester implements URLStreamHandlerFactory {

  String theURL;

  public static void main (String[] args) {

    if (args.length == 1) {
      ProtocolTester pt = new ProtocolTester(args[0]);
      URL.setURLStreamHandlerFactory(pt);
      pt.test();
    }
    else {
      System.err.println("Aufruf: java ProtocolTester url");
    }
  }

  public ProtocolTester(String s) {

    theURL = s;
  }

public void test() {

    String theLine;

    try {
      URL u = new URL( theURL);
      DataInputStream dis = new DataInputStream( u.openStream());
      while ((theLine = dis.readLine()) != null) {
        System.out.println(theLine);
      }
    }
    catch (IOException e) {
      System.err.println(e);
    }
  }
```

```
public URLStreamHandler createURLStreamHandler(String protocol) {

   protocol = protocol.toLowerCase();
   try {
     Class ph = Class.forName(protocol + "URLStreamHandler");
     Object o = ph.newInstance();
     return (URLStreamHandler) o;
   }
   catch (Exception e) {
     return null;
   }
  }
}
```

Das Witzige an dieser Lösung ist das flexible Handhaben der unterschiedlichen Protokolle. Der Rest ist eher Standard.