In diesem Kapitel:
- *Was ist ein Content Handler?*
- *Die ContentHandler Klasse*
- *Content Handler Factories*
- *Content Handler Beispiele und Techniken*

12

Content Handler

12.1. Was ist ein Content 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

Content Handling besteht in der korrekten Interpretation des Inhaltes der Daten, die vom Server zum Client übermittelt wurden.
Die meisten Browser arbeiten mit Plugins:
- der Browser kennt einige Grunddatentypen und Subtypen
- falls die Daten von diesem Type sind, dann kann der Browser die Daten verarbeiten bezw. einer Applikation zuweisen (Sie können den Browser so konfigurieren, dass jede WinHelp Datei mit dem WinHelp Programm angezeigt wird).
- falls Daten nicht interpretiert werden können, dann wird eine Dateispeicherung vorgeschlagen

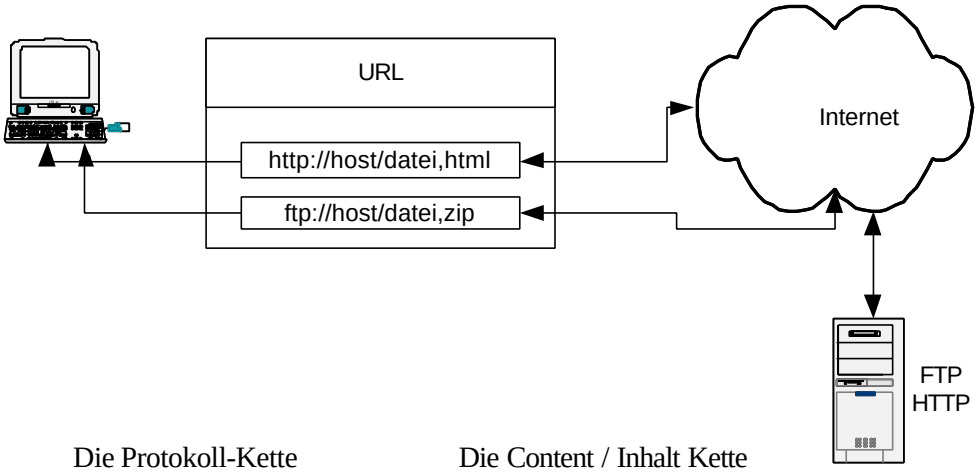
HotJava, der Browser von Sun Microsystems, ging einen anderen Weg:
- der Browser kennt einige Grunddatentypen und Subtypen
- falls die Daten von einem bekannten Typ sind, werden sie angezeigt oder an die entsprechende Applikation weiter gereicht
- falls die Daten nicht interpretiert werden können, versucht der Browser das nötige Plugin selbständig zu finden, herunter zu laden und die Daten zu interpretieren

Sun hat im Browser Krieg offensichtlich verloren. HotJava kann man aber immer noch vom Java Site (java.sun.com) herunter laden. Der Browser erinnert einem aber eher an die Pionierzeit des Internets, als an Konkurrenzprodukte.

Jeder Content Handler ist eine Unterklasse von `java.net.ContentHandler`.
Diese Klasse weiss, wie Daten aus einem Eingabestrom zusammen mit der Definition eines MIME Types in ein Java Objekt umgewandelt werden kann.
Da man mit Java Class Dateien aus dem Web herunter laden kann (Applets, ...), ist es ein kleiner Schritt zum Herunterladen von Content Handler Klassen. Aber die Browser Industrie schlug einen andern Weg ein.

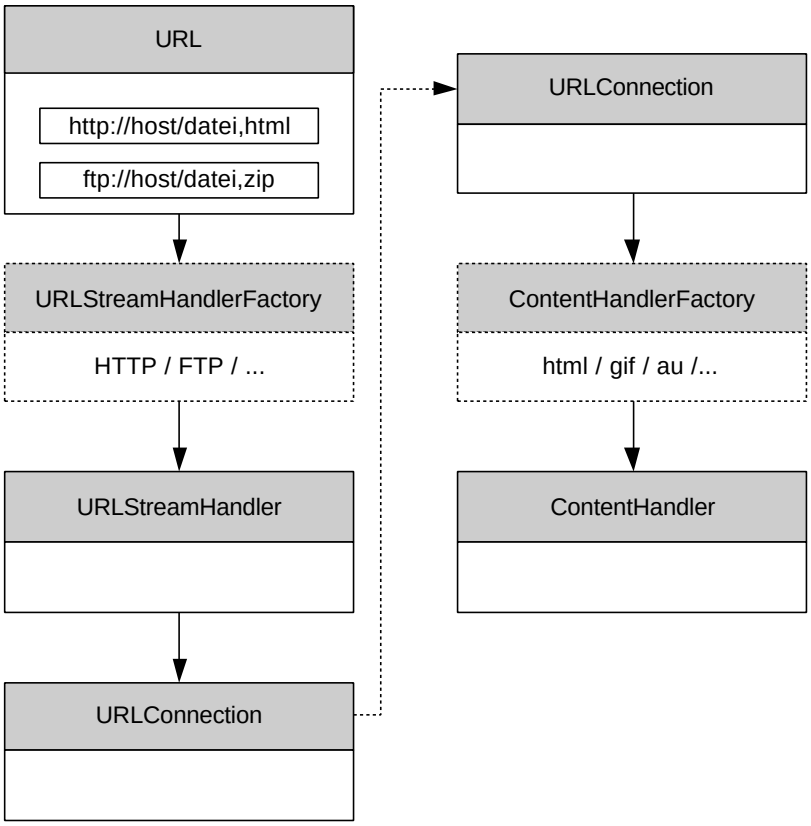
Was macht ein Content Handler konkret?

- der Content Handler liest Daten von einem `URLConnection` Objekt
- der Content Handler konstruiert ein (Java) Objekt für den bestimmten Datentyp



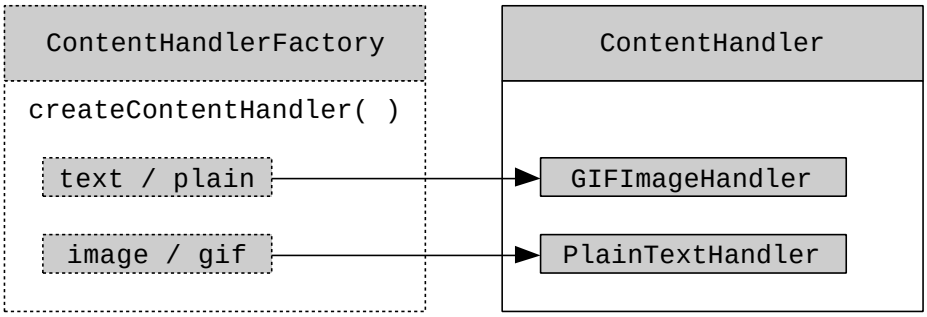
Die Protokoll-Kette

Die Content / Inhalt Kette



Beispiel:
- der Content Handler liest ein Bild
- er erkennt, dass der Datentyp / Unterdatentyp `image/gif` ist
- somit liefert der Content Handler ein `URLImageSource` Objekt, ein Objekt, welches das `ImageProducer` Interface implementiert.

Beispiel:
- der Content Handler liest einen Text
- er erkennt, dass der Datentyp / Unterdatentyp `text/plain` ist
- somit liefert der Content Handler ein `String` Objekt, ein Objekt



Content Handler sind mit Protokoll Handlern verknüpft. Sofern der Datentyp Text ist, gibt es wenig Probleme. Die Methode `URLConnection.getContent()` liefert in diesen Fällen einen Eingabestrom, aus dem die Daten gelesen werden können.
Falls der Datentyp komplexer ist, dann versagt dieses Vorgehen. Der Ablauf sieht dann wie folgt aus:
- `getContent()` prüft den MIME Type
- mit Hilfe von `createContentHandler()` wird dann ein Content Handler "produziert", mit Hilfe einer Klasse, welche das `ContentHandlerFactory` Interface implementiert
- nachdem der passende Content Handler existiert, ruft die `getContent()` Methode der Klasse `URLConnection()` die Methode `getContent()` des Content Handlers auf. Man verwendet also kaum Methoden des Content Handlers direkt.

Die Klasse, welche die `ContentHandlerFactory` Schnittstelle implementiert, ist für die korrekte Wahl es `ContentHandler` zu diesem MIME Type verantwortlich.Die Schnittstelle kann von irgend einer Klasse implementiert werden.

Jede Applikation kann lediglich einen `ContentHandler` auswählen. Beim Starten der Programme wird die `ContentHandlerFactory` auf `null` gesetzt.

Falls keine Factory gesetzt wird, dann wählt Java eine Content Handler Klasse aus. Diese hat einen Namen mit folgendem Aufbau:

`sun.net.www.content.`*`type.subtype`*

wobei `(Type,Subtype)` (MIME Type, MIME Subtype) sind.

12.1.1. Zusammenfassung des Ablaufs

Hier nochmals der Gesamtablauf:

1. es wird ein URL kreiert, welcher auf eine Internet Resource zeigt
2. die `getContent`() Methode der URL wird aufgerufen, um ein Objekt zu liefern, welches den Inhalt der Resource enthält.
3. die Verbindung zur URL wird, falls nicht bereits erfolgt, aufgebaut.
4. die `getContent`() Methode des `URLConnection` Objektes wird aufgerufen
5. `URLConnection.getContent`() ruft die Methode `URLConnection.getContentHandler`() auf, um den Content Handler für den MIME Typ und Subtyp zu finden.
6. `getContentHandler`() prüft, ob bereits ein anderer Handler für diesen Typ im Cache ist.
   Falls ja:
   dann wird dieser Handler an die Methode `URLConnection.getContent`() zurück gereicht.
   Im Falle eines Applets muss also kein neuer ContentHandler geladen oder gesucht werden.
   Falls nein:
   falls `ContentHandlerFactory` nicht `null` ist
   dann ruft die Methode `getContentHandler`() die `createContentHandler` Methode der `ContentHandlerFactory` auf, um einen neuen ContentHandler zu instanzieren.

   Falls dies erfolgreich ist
   dann wird das ContentHandler Objekt an die `URLConnection.getContent`() Methode zurück gegeben.

   falls die `ContentHandlerFactory` null ist
   oder `createContentHandler`() fehlgeschlagen ist,
   dann sucht Java eine Content Handler Klasse mit einem Namen, der wie folgt aufgebaut ist:
   sun.net.www.content.*type*.*subtype* mit MIME (type,subtype).

   falls der Content Handler gefunden wurde, wird er zurück gegeben, sonst liefert die Methode `createContentHandler`() `null`.
7. Nachdem nun ein Content Handler (der auch `null` sein kann) zur Verfügung steht, also ein `ContentHandler`() Objekt, wird die Methode `ContentHandler.getContent`() aufgerufen.

   Falls der `ContentHandler` null ist, dann wird eine `IOException` geworfen.

8. nun wird entweder ein Objekt oder eine Exception dem Aufrufbaum entlang hoch / zurück gereicht.

Diesen Ablauf kann man auf drei Arten beeinflussen:
a) durch Konstruktion eines URL Objektes und Aufruf der getContent() Methode
b) durch Definition einer neuen Unterklasse der `ContentHandler` Klasse, welche dann von der Methode `getContent`() eingesetzt werden kann.
c) durch Definition einer eigenen `ContentHandlerFactory` und setzen dieser Factory mit Hilfe der Methode `URLConnection.setContentHandlerFactory`()

12.2. Die ContentHandler Klasse

Schauen wir uns den Aufbau dieser Klasse etwas genauer an: die Klasse ist ABSTRACT

java.net

Class ContentHandler

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

---

public abstract class **ContentHandler**
extends Object
Die abstrakte Klasse `ContentHandler` ist die Oberklasse aller Klassen, die `Objekte` von einer `URLConnection` lesen.
Applikationen verwenden im Allgemeinen nicht direkt die `getContent` Methode dieser Klasse. In der Regel wird die Methode `getContent` aus der Klasse `URL` oder `URLConnection` aufgerufen.
Die Content Handler Factory der Applikation wird aufgerufen. Als Parameter wird eine Zeichenkette, der MIME Type übergeben. Die Content Handler Factory implementiert die Schnittstelle `ContentHandlerFactory` und wird mit `setContentHandler` gesetzt
Die Faktory liefert eine Instanz einer Subklasse von `ContentHandler`. Deren `getContent` Method wird aufgerufen, um ein Objekt zu kreieren.

Seit:
JDK1.0
Siehe Auch:
`getContent(java.net.URLConnection)`, `ContentHandlerFactory`, `URL.getContent()`, `URLConnection`, `URLConnection.getContent()`, `URLConnection.setContentHandlerFactory(java.net.ContentHandlerFactory)`

Konstruktor Übersicht

`ContentHandler``()`	

Methoden Übersicht

`abstract`	**`getContent`**`(URLConnection urlc)`

Object	Diese Methode liest den Stream der URL

Konstruktor Detail

ContentHandler

```
public ContentHandler()
```

Methoden Detail

getContent

```
public abstract Object getContent(URLConnection urlc)
                           throws IOException
```

Diese Methode liest den Stream, der zur URL gehört.

Parameters:

`urlc` - eine URL Verbindung.

Returns:

das Objekt, welches vom ContentHandler gelesen wurde

Throws:

IOException - falls ein I/O Fehler auftrat

Schauen wir uns das Ganze mal im Detail an:
eine Unterklasse der `ContentHandler` Klasse sollte die Methode `getContent()` überschreiben und ein Objekt liefern, welches dem Inhalt entspricht, einfach in einer gültigen Java Form. Die Methode kann recht einfach sein, oder aber sehr komplex, je nach dem Inhalt. Zum Beispiel wird die Methode im Falle von `text/plain` sehr einfach sein; bei `text/rtf` sieht die Methode komplexer aus.

Das Problem ist also eigentlich etwas verschoben:
es ist nicht eindeutig bestimmbar, welches Java Objekt einem MIME Typ zuzuordnen ist. Wir haben also von Fall zu Fall zu entscheiden.

12.2.1. Konstruktor

Die Klasse ContentHandler ist abstrakt. Sie muss also implementiert werden und benötigt daher eigentlich keinen Konstruktor.
Aus der obigen Klassenbeschreibung sehen wir, dass der einzige Konstruktor der Standard / Default Konstruktor angegeben wurde.

12.2.2. getContent Methode

Die Methode besitzt folgende Signatur:

```
public abstract Object getContent(URLConnection uc) throws
IOException
```

Die Methode wird lediglich innerhalb der `getContent()` Methode eines `URLConnection` Objektes aufgerufen. Die Methode wird in den Subklassen überschrieben beziehungsweise implementiert. Sie verwendet den `InputStream` des `URLConnection` Objektes.
Das Zusammenspiel zwischen der getContent() und dem InputStream der URLConnection Klasse sieht wie folgt aus:

1) Die URLConnection Klasse entfernt alle Protokoll spezifischen Headers und Zusatzinformationen des InputStreams der URLConnection
2) Der ContentHandler beschränkt sich auf den eigentlichen Inhalt des InputStreams

Schauen wir uns ein einfaches Beispiel an:
die folgenden Daten sind Tab getrennt

J.M.Joller	Sonnenbergstrasse 73	8610	USTER
Harry Braun	Heinrichweg 12	8004	Zürich

Wir haben also zwei Datensätze. Jeder Datensatz besteht aus mehreren Datenfeldern (Name, ...). Jedes Datenfeld hat eine bestimmte (sematische) Bedeutung, in der Regel in allen Datensätzen die selbe.

1. Frage: welche Java Objekte entsprechen diesen Daten?
2. Antwort : eine Zeichenkette (String)

Zusammenhängende Datensätze könnten wir auf einen Vector abbilden.
Allgemein gilt:
je genauer wir die Daten kennen, desto besser lassen diese sich mit einem ContentHandler bearbeiten.

In unserem Falle könnten wir eine Klasse "Person" definieren, die in etwa folgendermassen aussehen könnte:

```
public class person {
     String strName;
     String strStrasse;
     String strPLTZ;
     String strOrt;
}
```

Dazu definieren wir noch passende Methoden und gelangen damit zu einer brauchbaren Beschreibung des Inhaltes aus dem obigen Beispiel.

Im allgemeineren Fall könnten wir eine beliebige Anzahl Datenfelder zulassen.
Betrachten wir ein konkretes Beispiel:

12.2.3. ContentHandler für MIME Type text/tab-separared-values

```
//Titel:       Content Handler : Tabulator getrennte Daten
//Copyright:   Copyright (c) 1999
//Autor:       J.M.Joller
//Beschreibung:  Definition eines eigenen ContentHandlers für die Bearbeitung von Tab
//separierten Daten
package Beispiel12_1;

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

public class tabSeparatedValues_ContentHandler extends ContentHandler {

  public Object getContent(URLConnection uc) {

    String strDatensatz;
    Vector vec = new Vector();

    try {
      DataInputStream dis = new DataInputStream( uc.getInputStream());
      while ((strDatensatz = dis.readLine()) != null) {
        String[] strDatensatzArray = datensatzInArray(strDatensatz);
        vec.addElement(strDatensatzArray);
      }
    }
    catch (IOException e) {
    }

    return vec;

  }

  private String[] datensatzInArray(String strZeile)  {

    StringTokenizer st = new StringTokenizer(strZeile, "\t");
    int iAnzahlFelder = st.countTokens();
    String[] strDatenfelder = new String[iAnzahlFelder];
    for (int i = 0; i < iAnzahlFelder; i++) {
      strDatenfelder[i] = st.nextToken();
    }

    return strDatenfelder;

  }

}
```

12.2.4. Erläuterungen zum Programm

Das Programm besitzt zwei Methoden:

1. `private String[] datensatzInArray(String strZeile)`
   und
2. `public Object getContent(URLConnection uc)`

Die erste Methode erlaubt es einen Datensatz in seine Datenelemente zu zerlegen und in ein Datenfeld abzuspeichern. Die Zerlegung geschieht mit Hilfe eines Tokenizers, wobei Tab "\t" als Separationszeichen eingesetzt wird. Die Methode ...countTokens() bestimmt die Anzahl Trennzeichen in der zu analysierenden Zeichenkette. Der Rest der ersten Methode zerlegt die Eingabezeichenkette in deren Datenfelder und speichert diese im Datenfeld ab. Die Methode liefert dieses Datenfeld zurück. Da die Methode nur innerhalb der definierten Klasse eingesetzt werden soll, kann sie als *private* deklariert werden.

Die `getContent(URLConnection uc)` Methode definiert einen Vektor, in den die Ergebnisse abgespeichert werden sollen.
Wie bereits besprochen, wird dann der InputStream der URLConnection abgefragt. Damit dies leichter möglich ist, wird ein DataInputStream um den InputStream gewrapped. Dadurch stehen mächtigere Leseoperationen zur Verfügung.

12.3. Factories für ContentHandler

Nachdem wir einen einfachen Content-Handler konstruiert haben, müssen wir noch lernen, wie dieser eingesetzt werden kann.

12.3.1. MIME Type Notation nach Sun / Java

Viele MIME Type Definitionen enthalten Zeichen, die in Java Identifiern illegal sind, zum Beispiel der Bindestrich :
MIME Type application/octet-stream
Sun hat dafür folgende Ersatznotation eingeführt:
sun.net.content.application.octet_stream

Im Wesentlichen wird also der Bindestrich durch den Underscore ersetzt.

Wie sieht nun ein typischer Programmablauf aus, bei dem ein eigener Content Handler eingesetzt werden soll?

1. die Applikation bildet eine Instanz der URLConnection Klasse
2. die URLConnection Klasse verfügt über eine Methode setContentHandlerFactory(), mit der die ContentHandlerFactory genau einmal pro Applikation gesetzt werden kann.
3. die abstrakte ContentHandlerFactory definiert die Regeln, nach denen ein ContentHandler ausgewählt und geladen wird. Wir müssen also eine Klasse definieren, welche die (abstrakte) ContentHandlerFactory implementiert. Diese Klasse muss die Methode createContentHandler() implementieren.
4. die Methode createContentHandler() muss null zurück liefern, falls kein Content-Handler bekannt ist. In diesem Falle sucht Java einen Default Content-Handler zu finden.

12.3.2. Die createContentHandler Methode

Diese Methode hat folgende Signatur:

```
public abstract ContentHandler createContentHandler(String
mimetype);
```

Sie ist also abstrakt und liefert einen ContentHandler für den MIME Type mimetype.
Die Methode sollte, wenn nötig, nur von der Methode getContent() eines URLConnection Objektes aufgerufen werden.

12.3.2.1. Programmskizze

```
public ContentHandler createContentHandler(String mimeType) {

        if (mimeType.equals("text/tab-separated-values") {
                return new tabSeparatedValues_ContentHandler ();
        }
        else {
                return null;
         }
}
```

Damit ein Content-Handler gesetzt werden kann, muss das Abfrageprogramm oder der Browser den Datentyp überhaupt liefern können.

Ein einfacher Test, ob der MIME Type text/tab-separated-values unterstützt wird, besteht darin, mit Telnet auf einen Web Server zuzugreifen. Zuvor muss darauf eine Datei mit der Extension tsv, zum Beispiel "Personen.tsv", abgespeichert werden. Diese wird am einfachsten mit Hilfe von Excel generiert werden.

12.3.2.2. Beispielprotokoll

```
telnet localhost 80                    (HTTP Protokoll)
...
Escape character is '^]'
GET Personen.tsv  HTTP/1.0
```

im unglücklisten Fall erhalten Sie folgende Meldung:

```
HTTP/1.0 400 This document is of a type which is not supported by your browser
Server: HTTPS/0.991
Allow: GET HEAD POST
MIME-version: 1.0
Content-type: text/html

<TITLE>This document is of a type which is not supported by your browser</TITLE>
<BODY>
<H1>This document is of a type which is not supported by your browser</H1>
<P>HTTP status code: 400
```

</BODY>
Falls Sie mehr Glück haben, wird der Datentyp vom Web Server unterstützt und Sie erhalten folgende Meldung:

```
HTTP/1.0 200 OK
Date: Mon, 10 Jan 2000 14:30:34 GMT
Server: Apache/1.1.2
Content-type: text/tab-separated-values
Content-length: 127
Last-modified: Mon, 10 Jan 2000 14:10:12 GMT

Josef   Joller  Sonnenbergstrasse 73    8610    Uster   Zürich
Peter   Ruchti  Käferweg 23     8022    Zürich  Zürich
Hans Peter      Klopfstein      Mühlegasse 7    3170    Laupen  Bern
Klaus   Amrein  Rue de Payerne  3067    Murten  Fribourg
```

In diesem Fall sollte auch das folgende Programm, ein Content-Handler Tester für den TSV Content-Handler, funktionieren.

12.3.3. Programmbeispiel : Content Handler Tester

```
//Titel:        Content Handler Testprogramm
//Version:
//Copyright:    Copyright (c) 2000
//Autor:        J.M.Joller
//Firma:
//Beschreibung:  Das Programm testet den Content Handler aus Beispiel 12_1
//für Tab separierte Werte
package Beispiel12_2;

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

public class tsvContentTester implements ContentHandlerFactory {

  String theURL;

  public static void main (String[] args) {

    if (args.length == 1) {
      tsvContentTester ct = new tsvContentTester(args[0]);
      URLConnection.setContentHandlerFactory(ct);
      ct.test();
    }
    else {
      System.err.println("Aufruf: java tsvContentTester url");
    }
```

```
  }

  public tsvContentTester(String s) {

    theURL = s;

  }

  public void test() {

    try {
      URL u = new URL( theURL);
      Vector v = (Vector) u.getContent();
      for (Enumeration e = v.elements() ; e.hasMoreElements() ;) {
        String[] sa = (String[]) e.nextElement();
        for (int i = 0; i < sa.length; i++) {
          System.out.print(sa[i] + "\t");
        }
        System.out.println();
      }
    }
    catch (IOException e) {
      System.err.println(e);
    }

  }

public ContentHandler createContentHandler(String type) {

    if (type.equalsIgnoreCase("text/tab-separated-values")) {
      return new tabSeparatedValues_ContentHandler();
    }
    else {
      return null;
    }

  }

}
```

12.4. Das Time Protokoll

Als nächstes wollen wir einen einfachen Content Handler definieren, der die Zeichenkette des Time Protokolls (32 Bit binäre Darstellung der Anzahl Sekunden seit Mitternacht, 1 Januar 1900). Diese Darstellung wollen wir in ein java.util.Date Objekt umwandeln.

12.4.1. Der Time Content Handler

```
//Titel:       Time Content Handler
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  ein einfacher Content Handler:
//dieser wandelt die Antwort des Time Protokolls um in ein java.util.Date Objekt
package Beispiel12_3;

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

public class timeContentHandler extends ContentHandler {

  public Object getContent(URLConnection uc) {

    Date now = null;

    try {
      DataInputStream dis = new DataInputStream( uc.getInputStream());
      int theTime = dis.readInt();
      // 86400 seconds a day
      // midnight January 1970 = 2,208,988,800 Sekunden seit midnight January 1, 1900
      long secondsSince1970 = theTime - 2208988800L;
      long millisecondsSince1970 = secondsSince1970 * 1000L;
      now = new Date(millisecondsSince1970);
    }
    catch (IOException e) {
    }

    return now;

  }
```

}

12.5. Content Handler für ein Bildformat image/x-fits

Das Grafikformat FITS = Flexible Image Transport System wird vor allem in der Astronomie eingesetzt.
FITS Dateien beschreiben Graustufen Bilder, die einen Header enthalten, der die Bit Tiefe des Bildes, dessen Breite und Höhe beschreibt und die Anzahl Bilder in der Datei. In der Regel werden mehrere Bilder in einer Datei abgespeichert. Diese Bilder sind irgendwie zusammen hängend, zum Beispiel unterschiedliche Positionen eines Sterns, eines Mondes ...

Der Content Handler soll nur das erste Bild in der Datei analysieren. Das vereinfacht das Programm.

FITS Dateien haben einen strikten Aufbau:
1. die Dateien bestehen aus Blöcken zu je 2880 Bytes
2. falls ein Block nicht voll ist, dann wird er einfach mit Leerzeichen aufgefüllt
3. jede FITS Datei besteht aus zwei Teilen:
   a) Kopfdaten (Header)
   b) primäre Dateneinheit
4. der Header und Daten bestehen aus einer ganzzahligen Anzahl Blöcke (jeder Block enthält also nur Header Infos oder Daten)
5. zusätzlich zur primären Dateneinheit können Erweiterungen existieren, in denen auch Daten gespeichert sind. Diese zusätzliche Komplexität ignorieren wir hier.
6. der Header beginnt im ersten Block der FITS Datei. Er kann einen oder mehrere Blöcke umfassen. Der letzte Header Block kann mit Leerzeichen aufgefüllt sein.
7. der Header besteht aus ASCII Text; jede Headerzeile ist genau 80 Zeichen lang.
8. jede Headerzeile besteht aus 8 Zeichen, welche ein Kennwort enthalten, gefolgt von einem Gleichheitszeichen, gefolgt von einem Leerzeichen (das zehnte Zeichen).
9. Zeichen 11 bis 30 enthalten einen Wert : eine ganze Zahl, eine Gleitkommazahl oder T oder F für true oder false, oder eine Zeichenkette in Anführungszeichen.
10. Zeichen 31 bis 80 können einen Kommentar enthalten mit einem Slash (/) als Trennzeichen zu den Werten.

12.5.1. Beispiel FITS Header

```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
          1111111111222222222233333333334444444444555555555566
SIMPLE  =                    T /
BITPIX  =                   16 /
NAXIS   =                    2 /
NAXIS1  =                  242 /
NAXIS2  =                  252 /
DATE    = '10 Jan 2000         /
```

```
TELESC  = 'NSO/SP - VT'              /
IMAGE   = 'Continuum'                /
COORDS  = 'N29.1W34.2'               /
OBSTIDE = '13:59:00 UT'              /
END
```

Jede FITS Datei muss mit dem Schlüsselwort SIMPLE beginnen. Der Wert dieses Schlüsselwortes ist immer T. Falls dies nicht zutrifft, dann ist die Datei unkorrekt und muss ignoriert werden.

Die zweite Zeile der Datei ist immer BITPIX. Dieser Parameter beschreibt die Art und Weise, wie die Daten gespeichert werden. BITPIX kann fünf Werte annehmen:

1. 16 : 16 Bits per Pixel oder Java Short
2. 32 : ein Java int
3. -32 : ein Java Float
4. -64 : ein Java Double
5. 8 : also 8 Bits pro Pixel (entspricht in etwa einem Java Byte)

Die restlichen Schlüsselworte können in beliebiger Reihenfolge sein. Zur Analyse wird man die Schlüsselworte in eine Hashtabelle lesen und dann die gewünschte Reihenfolge herstellen.

NAXIS beschreibt, wie viele Dimensionen das Bild hat.
NAXISn gibt die Anzahl Pixel entlang der Achse n

TELESC gibt eine Kurzbezeichnung des Teleskopes
DATE beschreibt, wann die Aufnahme gemacht wurde.

Das END Schlüsselwort schliesst den Header ab.

12.5.2. Der FITS Content Handler

```
//Titel:       FIPS Content Handler
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  ein Content Handler für das FIPS Dateiformat zur Übermittlung von
Astrodaten
package Beispiel12_4;

import java.net.*;
import java.io.*;
import java.awt.image.*;
import java.util.Hashtable;
import java.util.Enumeration;
```

```
public class fitsContentHandler extends ContentHandler {

  public Object getContent(URLConnection uc) {

    int width = -1;
    int height = -1;
    int bitpix = 16;
    int[] theData = null;
    int naxis = 2;
    Hashtable header = null;

    try {
      DataInputStream dis = new DataInputStream( uc.getInputStream());
      header = readHeader(dis);

      bitpix = getIntFromHeader("BITPIX  ", -1, header);
      if (bitpix  <= 0) return null;
      naxis = getIntFromHeader("NAXIS   ", -1, header);
      if (naxis  < 1) return null;
      width = getIntFromHeader("NAXIS1  ", -1, header);
      if (width  <= 0) return null;
      if (naxis == 1) height = 1;
      else height = getIntFromHeader("NAXIS2  ", -1, header);
      if (height  <= 0) return null;

      if (bitpix == 16) {
          short[] theInput =  new short[height * width];
          for (int i = 0; i < theInput.length; i++) {
            theInput[i] = dis.readShort();
          }
          theData = scaleArray(theInput);
      }
      else if (bitpix == 32) {
        int[] theInput =  new int[height * width];
        for (int i = 0; i < theInput.length; i++) {
          theInput[i] = dis.readInt();
        }
        theData = scaleArray(theInput);
      }
      else if (bitpix == 64) {
        long[] theInput =  new long[height * width];
        for (int i = 0; i < theInput.length; i++) {
          theInput[i] = dis.readLong();
        }
       theData = scaleArray(theInput);
      }
      else if (bitpix == -32) {
        float[] theInput =  new float[height * width];
        for (int i = 0; i < theInput.length; i++) {
```

```
        theInput[i] = dis.readFloat();
       }
       theData = scaleArray(theInput);
     }
     else if (bitpix == -64) {
       double[] theInput =  new double[height * width];
       for (int i = 0; i < theInput.length; i++) {
         theInput[i] = dis.readDouble();
        }
        theData = scaleArray(theInput);
     }
     else {
       System.err.println("Invalid BITPIX");
       return null;
     } // end if-else-if

   }  // end try
   catch (IOException e) {

   }

   return new  MemoryImageSource(width,  height,  theData,  0,  width);

}  // end getContent

  Hashtable readHeader(DataInputStream dis) throws IOException {

   int blocksize = 2880;
   int fieldsize = 80;
   String key, value;
   int linesRead = 0;

   byte[] buffer = new byte[fieldsize];

   Hashtable h = new Hashtable();
   while (true) {
     dis.readFully(buffer);
     key =  new String(buffer, 0, 0, 8);
     linesRead++;
     if (key.substring(0, 3).equals("END")) break;
     if (buffer[8] != '=' || buffer[9] != ' ') continue;
     value = new String(buffer, 0, 10, 20);
     h.put(key, value);
   }
   int linesLeftToRead = (blocksize - ((linesRead * fieldsize) % blocksize))/fieldsize;
   for (int i = 0; i < linesLeftToRead; i++) dis.readFully(buffer);

   return h;

  }
```

```
int getIntFromHeader(String name, int Default, Hashtable header) {

    String s = "";
    int result = Default;

    try {
      s = (String) header.get(name);
    }
    catch (NullPointerException e) {
      return Default;
    }
    try {
      result = Integer.parseInt(s.trim());
    }
    catch (NumberFormatException e) {
      System.err.println(e);
      System.err.println(s);
      return Default;
    }

    return result;

}

int[] scaleArray(short[] theInput) {

  int theData[] = new int[theInput.length];
  int max = 0;
  int min = 0;
  for (int i = 0; i < theInput.length; i++) {
    if (theInput[i] > max) max = theInput[i];
    if (theInput[i] < min) min = theInput[i];
  }
  long r = max - min;
  double a = 255.0/r;
  double b = -a * min;
  int opaque = 255;
  for (int i = 0; i < theData.length; i++) {
    int temp = (int) (theInput[i] * a + b);
    theData[i] =  (opaque << 24)  | (temp << 16)  | (temp << 8) | temp;
  }
  return theData;

}

int[] scaleArray(int[] theInput) {

  int theData[] = new int[theInput.length];
  int max = 0;
```

```
  int min = 0;
  for (int i = 0; i < theInput.length; i++) {
    if (theInput[i] > max) max = theInput[i];
    if (theInput[i] < min) min = theInput[i];
  }
  long r = max - min;
  double a = 255.0/r;
  double b = -a * min;
  int opaque = 255;
  for (int i = 0; i < theData.length; i++) {
    int temp = (int) (theInput[i] * a + b);
    theData[i] =  (opaque << 24)  | (temp << 16)  | (temp << 8) | temp;
  }
  return theData;

}

int[] scaleArray(long[] theInput) {

  int theData[] = new int[theInput.length];
  long max = 0;
  long min = 0;
  for (int i = 0; i < theInput.length; i++) {
    if (theInput[i] > max) max = theInput[i];
    if (theInput[i] < min) min = theInput[i];
  }
  long r = max - min;
  double a = 255.0/r;
  double b = -a * min;
  int opaque = 255;
  for (int i = 0; i < theData.length; i++) {
    int temp = (int) (theInput[i] * a + b);
    theData[i] =  (opaque << 24)  | (temp << 16)  | (temp << 8) | temp;
  }
  return theData;

}

int[] scaleArray(double[] theInput) {

  int theData[] = new int[theInput.length];
  double max = 0;
  double min = 0;
  for (int i = 0; i < theInput.length; i++) {
    if (theInput[i] > max) max = theInput[i];
    if (theInput[i] < min) min = theInput[i];
  }
  double r = max - min;
  double a = 255.0/r;
  double b = -a * min;
```

```
    int opaque = 255;
    for (int i = 0; i < theData.length; i++) {
      int temp = (int) (theInput[i] * a + b);
      theData[i] =  (opaque << 24)  | (temp << 16)  | (temp << 8) | temp;
    }
    return theData;

  }

  int[] scaleArray(float[] theInput) {

    int theData[] = new int[theInput.length];
    float max = 0;
    float min = 0;
    for (int i = 0; i < theInput.length; i++) {
      if (theInput[i] > max) max = theInput[i];
      if (theInput[i] < min) min = theInput[i];
    }
    double r = max - min;
    double a = 255.0/r;
    double b = -a * min;
    int opaque = 255;
    for (int i = 0; i < theData.length; i++) {
      int temp = (int) (theInput[i] * a + b);
      theData[i] =  (opaque << 24)  | (temp << 16)  | (temp << 8) | temp;
    }
    return theData;

  }

}
```

Dieser Content Handler ist offensichtlich etwas komplexer, realistischer als die vorherigen Beispiele.
Die wichtigste Methode ist die getContent() Methode. Alle anderen Methoden sind lediglich Hilfsfunktionen.
Die MemoryImageSource Klasse erlaubt die Konstruktion von Bildern, im Memory, wie der Name bereits erraten lässt.
Etwas Aufwand muss auch für die Umwandlung in RGB getrieben werden.

Schauen wir uns ein konkretes Anwendungsbeispiel für diesen Content Handler an

12.5.3. Ein FITS Content Handler Testprogramm

Sie müssen eine Verbindung zum Web haben, da eine FITS Datei auf dem Server sunsite.unc.edu gelesen werden soll. Eine verbesserte Version wird auch:

1. schliessbar sein
2. mit einer lokalen Datei arbeiten.

```
//Titel:       Testprogramm für den FIPS Content Handler
```

```
//Version:
//Copyright:    Copyright (c) 2000
//Autor:       J.M.Joller
//Firma:
//Beschreibung:  ein Testprogramm
package Beispiel12_5;

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

public class fits extends Frame implements ContentHandlerFactory {

  URL u;
  Image theImage;
  String name;
  ContentHandler fc;

  public static void main(String[] args) {

    String name;
    if (args.length == 0) name = "test.fit";
    else name = args[0] + ".fits";

    fits f = new fits(name);
    URLConnection.setContentHandlerFactory(f);
    f.resize(252, 252);
    f.fc = new fitsContentHandler();
    f.init();
    f.show();

  }

  public ContentHandler createContentHandler(String mimetype) {

    if (mimetype.equalsIgnoreCase("image/x-fits")) return fc;
    return null;

  }

  public fits(String s) {

    super(s);
    name = s;

  }

  public void init() {
```

```
    try {
      // da müsste ein FITS File stehen
      // Sie können auch die Datei test.fit (lokal) verwenden)
      u = new URL("http://sunsite.unc.edu/javafaq/" + name);

      ImageProducer ip = (ImageProducer) u.getContent();
      if (ip == null)  {
        System.err.println("Content handler returned null");
      }
      else {
       theImage = createImage(ip);
      }

    }
    catch (MalformedURLException e) {
      System.err.println(e);
    }
    catch (IOException e) {
      System.err.println(e);
    }
    catch (NullPointerException e) {
      System.err.println(e);
      e.printStackTrace();
    }

  }

public void paint(Graphics g) {

    g.drawImage(theImage, 0, 0, this);

  }

}
```