03-05-H -709.53 Informatik für Nichtinformatiker (10)

Prof. Dr. Udo Frese
Tobias Hammer

Beispiel: automatischer Webcamdownload

Ausnahmebehandlung

Was bisher geschah

- **Fehler im Programm finden**
  - Schritt für Schritt entwickeln, jeden Schritt testen, Kontrollausgaben
  - Debugger für schrittweise Ausführung verwenden
- **Grafik**
  - Bilder im Computer als 2D Array von Farbwerten (Pixel)
  - Rendern von geometrischen Grundelementen mit Grafikbibliothek (für uns: Rubygame)
- **Animation und Computerspiele**
- **Illusion der Bewegung durch schnelle Folge von Einzelbildern**
- **Endlos wiederholte Hauptschleife (main-loop-Architektur):**
  - Ereignisse holen
  - Zustand weiterrechnen
  - zeichnen
  - anzeigen
  - warten

Automatischer Webcamdownload

- **Webcam der Universität Bremen (Mineralogie)**
  - http://www.min.uni-bremen.de/cgi-bin/Mincam.cgi
  - 4 tägliche Bilder vom Fallturm
- **Aufgabe: Rubyprogramm das in gegebenem Zeitraum**
  - alle Bilder herunterlädt
  - mit geeigneten Dateinamen versieht
  - abspeichert
- **Rolle des Beispiels**
  - Wiederholen praktisches Programmieren
  - Fehlerbehandlung

Automatischer Webcamdownload

- **Bilder finden sich unter URL**
  - Jahreszahl (0000-9999)
  - Monat (01-12)
  - Tag (01-31)
  - Stunde (08,12,16, 20)
- **Unterverzeichnisse pro Jahr und Monat**
- **Reihenfolge und führende Nullen damit alphabetische Sortierung chronologisch ist**

http://www.min.uni-bremen.de/mincam/2010/01/0108.jpg

Basis URL

Jahr

Monat

Tag

Stunde

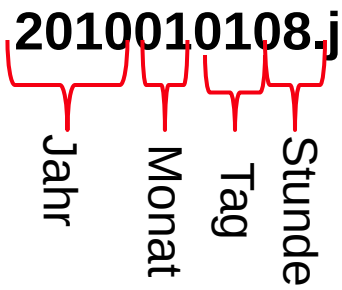
Automatischer Webcamdownload

<u>Frage an das Auditorium:</u> Wir wollen einen eindeutigen Dateinamen pro Bild (ohne Unterverzeichnisse). Wie können wir den konstruieren?

Automatischer Webcamdownload

<u>Frage an das Auditorium:</u> Wir wollen einen eindeutigen Dateinamen pro Bild (ohne Unterverzeichnisse). Wie können wir den konstruieren?

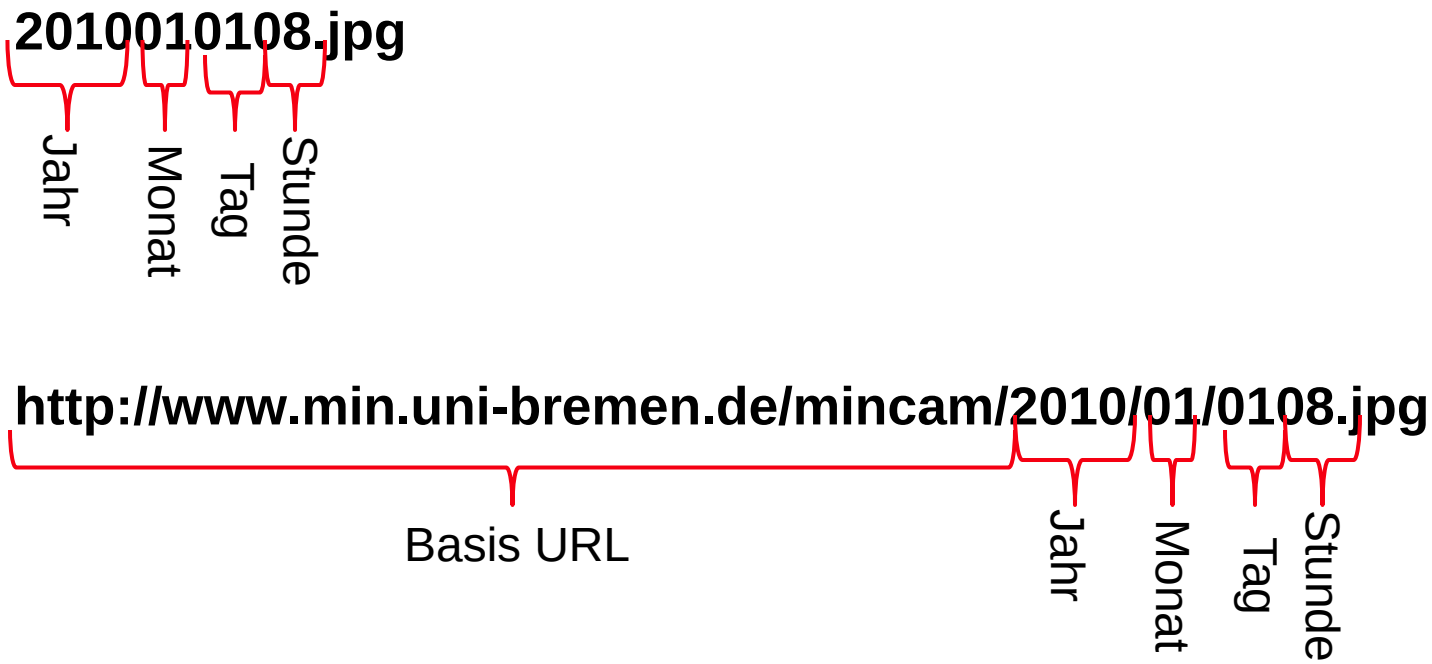
- **Jahr, Monat, Tag, Stunde direkt hintereinanderhängen**



Automatischer Webcamdownload

Frage an das Auditorium:

In welche Methoden zerlegen wir das Programm?



Automatischer Webcamdownload

Frage an das Auditorium:

In welche Methoden zerlegen wir das Programm?

- **`download (url, filename):`**
  - load `url` from the web and store the data in `filename`
- **`url_of_datetime (baseurl, datetime):`**
  - build the webcam .jpg url for the date/time `datetime`
- **`filename_of_datetime (datetime):`**
  - build the filename for the date/time `datetime`
- **`download_from_webcam (baseurl, datetime):`**
  - download the webcame .jpg corresponding to date/time `datetime`
- **`download_day (baseurl, date):`**
  - download all 4 .jpgs of a certain day (`date`)
- **`download_days (baseurl, from, to):`**
  - download all .jpgs for the day from `from` to `to` (inclusively)

Automatischer Webcamdownload

- **Vergleiche: inifrese0910_webcamdownload**

```
def download (url, filename)
  web  = open(url)
  file = open(filename, "wb")
  data = web.read
  file.write(data)
  file.close
  web.close
end
```

Ausnahmebehandlung

- **(D. Thomas, Programming Ruby 1.9, The Pragmatic Programmer, Kapitel 8)**

Ausnahmebehandlung

<u>Frage an das Auditorium</u>: Was passiert im Webcam-Beispiel, wenn…

- die URL der Webseite falsch ist
- ein .jpg fehlt
- die Internetverbindung abbricht
- die Webseite vorübergehend überlastet ist
- die Festplatte voll ist
- das Verzeichnis schreibgeschützt ist

```
def download (url, filename)
  web  = open(url)
  file = open(filename, "wb")
  data=web.read
  file.write(data)
  file.close
  web.close
end
```

Ausnahmebehandlung

<u>Frage an das Auditorium</u>: Was passiert im Webcam-Beispiel, wenn…

- die URL der Webseite falsch ist
- ein .jpg fehlt
- die Internetverbindung abbricht
- die Webseite vorübergehend überlastet ist
- die Festplatte voll ist
- das Verzeichnis schreibgeschützt ist

- **Das Programm bricht mit einer (technischen) Fehlermeldung ab**
- **Ein Teil der Bilder liegt auf der Platte**
- **Unvollständig heruntergeladene Bilder liegen auf der Platte**
- **⇒ Fehlerfälle müssen im Programm behandelt werden**
  - Dateien schließen
  - unvollständige downloads löschen
  - Fehler an Hauptprogramm weiterleiten
  - dort Anwendungsabhängige Fehlermeldung ausgeben

Ausnahmebehandlung

Klassische Fehlerbehandlung mit Fehlercodes (z.B. unter C)

- **Methoden liefern Fehlerzustand als Ergebnis zurück**
- **Aufrufende Methode muss Fehlerzustand überprüfen**
- **Verschiedene Möglichkeiten:**
- **Methoden liefern `nil` bei Fehler**
- **Methoden liefern zusätzlich zu einem Ergebnis einen Fehlercode zurück**
  - Fehlercode 0 $\Rightarrow$ korrekt abgelaufen
  - Sonst $\Rightarrow$ Fehler
- **Objekte haben Fehlerzustand (.error)**

Ausnahmebehandlung

Weiterleiten von Fehlerzuständen

- Hypothetisches Programm

```
def download (url, filename)
  web  = open(url)
  if web==nil then return 1 end
  file = open(filename, "wb")
  if file==nil then return 2 end
  data=web.read
  if web.error!=0 then return web.error end
  error = file.write(data)
  if error!=0 then return error end
  file.close
  web.close
  return 0
end
```

Ausnahmebehandlung

- **Klassische Fehlerbehandlung mit Fehlercodes (z.B. unter C)**
- **Uneinheitlich**
- **Sehr viele zusätzliche Anweisungen für Fehlerbehandlung**
- **Fehlerbehandlung in das normale Programm eingestreut**
- **Fehlerbehandlung kann leicht vergessen werden**
- **Oft lange Folge von Anweisungen zum Durchleiten von Fehlerzuständen durch die Aufrufhierarchie**
- **⇒ sehr unpraktisch**

Ausnahmebehandlung

Moderne Ausnahmebehandlung (Exception handling)

- **Bei auftretendem Fehler löst Methode eine Ausnahme (Exception) aus**
  - Daten zum Fehler werden in einem Exception Objekt (Klasse Exception oder Unterklasse) gespeichert
- **Normaler Programmablauf wird unterbrochen**
- **Aufrufende Methoden werden nacheinander abgebrochen,…**
- **… bis zu einer Methode, die diese Art Exception explizit behandelt**
- **Dort wird ein spezieller Programmabschnitt für diese Ausnahme ausgeführt**
  - Der den Fehler behandelt (Gegenmaßnahme, Fehlermeldung,…)
  - und/oder als neuen Fehler weiterleitet.

Ausnahmebehandlung

Ausnahmebehandlung in Ruby

- **Auslösen einer Ausnahme mit**
  - `raise` ExceptionClass, message
  - `raise` message (StandardError)
- **Behandeln einer Ausnahme mit `rescue ExceptionClass=>error`**
- **behandelt Ausnahmen der angegebenen Klasse**
- **bei `raise` übermittelten Daten (z.B. `.message`) in error**
  - Fehler inhaltlich behandeln
  - `retry` : Block nochmal
  - `raise` : Fehler weiterleiten
  - `raise` : neuen Fehler generieren

```
begin
    # normaler Ablauf
    rescue ExceptionClass=>error
    # Behandlung von error
    ensure
    # sowohl im Fehlerfall,
    # als auch im Normalfall
end
```

Ausnahmebehandlung

Ausnahmebehandlung in Ruby

- **Aufräumarbeiten mit ensure**
- **Ensure wird immer ausgeführt**
  - Im Normalfall
  - Bei Ausnahme
- **Z.B. für Dateien schließen**

```
begin
    # normaler Ablauf
    rescue ExceptionClass=>error
    # Behandlung von error
    ensure
    # sowohl im Fehlerfall,
    # als auch im Normalfall
end
```

Ausnahmebehandlung

```
def do_dangerous_things (object)
   while … do
     …
     if … then raise StandardError, “something bad happened”
   end
end
def do_with_list (list)
   list.each do |obj| do_dangerous_things (obj) end
end
def do_with_errors (list)
   begin
      do_with_list (list)
   rescue StandardError => error
      puts “An error occurred “, error.message
   ensure
      puts “Always done”
   end
end
```

Ausnahmebehandlung

Beispiel: Automatischer Webcamdownload

- **Ausnahmen werden ausgelöst, wenn**
  - die URL der Webseite falsch ist (open, OpenURI::HTTPError)
  - ein .jpg fehlt (open, OpenURI::HTTPError)
  - die Internetverbindung abbricht (read, OpenURI::HTTPError)
  - die Webseite vorübergehend überlastet ist (open, read, OpenURI::HTTPError, „503 Service unavailable“)
  - die Festplatte voll ist (write, IOError)
  - das Verzeichnis schreibgeschützt ist (open, IOError)
- **Massnahmen**
  - Dateien schließen
  - Unvollständige downloads löschen
  - Service unavailable: Noch mal probieren
  - Für den Nutzer verständliche Fehlermeldung

Ausnahmebehandlung

Behandeln von Fehlerzuständen

- Schließen von `web` und `file`

```
def download (url, filename)
  begin
    web  = open(url)
    file = open(filename, "wb")
    data=web.read
    file.write(data)
  ensure
    if file!=nil then file.close end
    if web!=nil then web.close end
  end
end
```

Ausnahmebehandlung

Behandeln von Fehlerzuständen

- **Löschen unvollständiger downloads**

```
def download (url, filename)
  begin
    web  = open(url)
    file = open(filename, "wb")
    data=web.read
    file.write(data)
  rescue OpenURI::HTTPError => error
    file.close; file = nil
    File.delete (filename)
  ensure
    if file!=nil then file.close end
    if web!=nil then web.close end
  end
end
```

Ausnahmebehandlung

Behandeln von Fehlerzuständen

- **Wiederholen bei temporär nicht verfügbarem Server**

```
…
  rescue OpenURI::HTTPError => error
    if error.message =~ /\b503\b/ then
      sleep 10
      retry
    elsif file!=nil then
      file.close; file = nil
      File.delete (filename)
      raise
    end
…
```

Ausnahmebehandlung

Behandeln von Fehlerzuständen

- Für den Nutzer verständliche Fehlermeldung

```
def download_days_with_errormessage (baseurl, from, to)
  begin
     download_days(baseurl, from, to)
     puts "Download finished."
  rescue StandardError => error
    puts "An error occurred:", error.message
  end
end
```

Ausnahmebehandlung

Klassenhierarchie der Ausnahmen

- **Rote Klassen sind relevant**
- **Exception**
  - NoMemoryError
  - ScriptError
    - *LoadError*
    - *NotImplementedError*
    - *SyntaxError*
  - SignalException
    - *Interrupt*
  - StandardError
  - SystemExit
  - fatal

- *ArgumentError*
- *IOError*
  - EOFError
- *IndexError*
- *LocalJumpError*
- *NameError*
  - NoMethodError
- *RangeError*
  - FloatDomainError
- *RegexpError*
- *RuntimeError*
- *SecurityError*
- *SystemCallError*
- *SystemStackError*
- *ThreadError*
- *TypeError*
- *ZeroDivisionError*
- *(OpenURI::HTTPError)*

Ausnahmebehandlung

Welche Ausnahme auslösen?

- **Schnelle Lösung**
  - Suche passende Unterklasse von `StandardError`
  - In der Standardhierarchie oder in passender Bibliothek (z.B. OpenURI)
  - Verwende aussagekräftige Fehlermeldung
  - Gebe relevante Werte in Fehlermeldung an
  - Für Fehler, die zum Programmabbruch führen sollen (bugs)
  - Für Fehler, die pauschale Fehlermeldungen geben sollen
- **Hochwertige Lösung**
  - Definiere passende Unterklasse (einer Unterklasse) von `StandardErrror`
  - Speichere relevante Werte als Attribute der Klasse
  - Für Fehler, die spezifisch behandelt werden sollen

Ausnahmebehandlung

Frage an das Auditorium: Ist der angegebene Zeitpunkt (from..to) ungültig, weil from>to soll eine Ausnahme ausgelöst werden.

- **Welche Klasse sollte man verwenden?**
- **Welche Fehlermeldung?**
- **Wie lautet die Anweisung?**

- *ArgumentError*
- *IOError*
  - EOFError
- *IndexError*
- *LocalJumpError*
- *NameError*
  - NoMethodError
- *RangeError*
  - FloatDomainError
- *RegexpError*
- *RuntimeError*
- *SecurityError*
- *SystemCallError*
- *SystemStackError*
- *ThreadError*
- *TypeError*
- *ZeroDivisionError*
- *(OpenURI::HTTPError)*

Ausnahmebehandlung

Frage an das Auditorium: Ist der angegebene Zeitpunkt (from..to) ungültig, weil from>to soll eine Ausnahme ausgelöst werden.

- **Welche Klasse sollte man verwenden?**
- **Welche Fehlermeldung?**
- **Wie lautet die Anweisung?**

```
if from>to then
  raise ArgumentError,
    "from=#{from} larger than
    to=#{to}"
end
```

- *ArgumentError*
- *IOError*
  - EOFError
- *IndexError*
- *LocalJumpError*
- *NameError*
  - NoMethodError
- *RangeError*
  - FloatDomainError
- *RegexpError*
- *RuntimeError*
- *SecurityError*
- *SystemCallError*
- *SystemStackError*
- *ThreadError*
- *TypeError*
- *ZeroDivisionError*
- *(OpenURI::HTTPError)*

Ausnahmebehandlung

Frage an das Auditorium: Welche Unterklasse von StandardError eignet sich für:

- **Wurzel(-1)**
- **Minimum einer leeren Liste**
- **Bruch+String**
- **Kein Weg im Graph gefunden**
- **Zu steuernde Maschine nicht eingeschaltet**
- **Übergebene Position eines Zeichens ist jenseits des Textendes**

- *ArgumentError*
- *IOError*
  - EOFError
- *IndexError*
- *LocalJumpError*
- *NameError*
  - NoMethodError
- *RangeError*
  - FloatDomainError
- *RegexpError*
- *RuntimeError*
- *SecurityError*
- *SystemCallError*
- *SystemStackError*
- *ThreadError*
- *TypeError*
- *ZeroDivisionError*
- *(OpenURI::HTTPError)*

Ausnahmebehandlung

Frage an das Auditorium: Welche Unterklasse von StandardError eignet sich für:

- **Wurzel(-1): `FloatDomainError`**
- **Minimum einer leeren Liste: `ArgumentError`**
- **Bruch+String: `TypeError`**
- **Kein Weg im Graph gefunden: `StandardError`**
- **Zu steuernde Maschine nicht eingeschaltet: `IOError`**
- **Übergebene Position eines Zeichens ist jenseits des Textendes: `IndexError`**

- *ArgumentError*
- *IOError*
  - EOFError
- *IndexError*
- *LocalJumpError*
- *NameError*
  - NoMethodError
- *RangeError*
  - FloatDomainError
- *RegexpError*
- *RuntimeError*
- *SecurityError*
- *SystemCallError*
- *SystemStackError*
- *ThreadError*
- *TypeError*
- *ZeroDivisionError*
- *(OpenURI::HTTPError)*

Zusammenfassung

- **Ausnahmebehandlung (Exception Handling) ist ein Sprachmechanismus die Behandlung von Fehlern vom normalen Programmablauf zu trennen und dadurch übersichtlicher zu machen**
  - Ausnahme wird ausgelöst (raise ExceptionClass, message)
  - Programm ablauf wird unterbrochen
  - Methoden werden nacheinander abgebrochen, bis…
  - … die Ausnahme behandelt wird (rescue ExceptionClass=>error)
  - Programmteile zum „aufräumen“ werden immer ausgeführt (ensure)
- **Vorteile gegenüber traditioneller Vorgehensweise mit Fehlercodes**
  - Keine Kaskaden von if-then Anweisungen
  - Keine Weiterleitung von Fehlern nötig
  - Man kann Fehlerbehandlung nicht vergessen
  - Daten zum Fehler sind verfügbar (im Exception Objekt)
  - Programmteile zum Aufräumen nur einmal
- **Eigene Fehlermeldungen: Unterklasse von `StandardError` suchen**