



LV 4342

Skriptsprachen-Praktikum

Übung 05

Umgang mit Exceptions
am Beispiel File I/O



Organisatorisches



- Arbeitsverzeichnis:

`~/lv/skriptspr/05/`

- Dateinamen:

`05-xcopy.rb` # neu erstellen & abgeben

- Werkzeuge:

`ruby` # Der Interpreter

`irb` # Interactive Ruby-Shell

`emacs, xemacs` # mit Ruby-Mode

`scite` # Ein portabler Editor, auch
mit Ruby-Mode

`ri` # Ruby-Dokumentation auf der Kommando-
zeile

`komodo` # IDE für Ruby u.a. Skriptsprachen

- Vorlagen:

`(keine)`



Die Aufgabe



- Kopieren Sie Dateien mit interaktiver Bedienung:

```
$ 05-xcopy.rb
Copy from: foo
Copy to: bar
# ... Ausführung & Reporting, s.u.
Copy to: baz
# ... Ausführung & Reporting
Copy to: <Ctrl-C>, <Ctrl-D> bzw. <Enter>
$
```

- Kern der Aufgabe: **Reaktion auf I/O-Fehler mit den Methoden der Ausnahmebehandlung!**

- Nebeneffekt:

- Methoden der Klasse IO kennen lernen.
- Optional: Effizienz dreier Alternativen vergleichen, per Profiling:

```
$ ruby -r profile 05-xcopy.rb # oder per "time":
$ time 05-xcopy.rb
```



Vorgaben



- Drei Kopier-Alternativen implementieren und nacheinander ausführen, zwecks Vergleich.
 - Controller-Klasse mit folgenden Methoden (Rückgabe beliebig):

```
class CopyCtrl
  def CopyCtrl.copy_by_line(src, dest)
    # Zeilenweise lesen & schreiben
    # mittels IO#each_line, IO#print
  end
  def CopyCtrl.copy_by_block(src, dest, bsize=4096)
    # Low-level & blockweise lesen & schreiben
    # mittels IO#sysread, IO#syswrite
  end
  def CopyCtrl.copy_by_oscall(src, dest)
    # Durch Aufruf eines geeignete OS-Programms.
    # mittels %x{cp ...} bzw. %x{copy ...} (Win)
  end
  def CopyCtrl.really_write?      ==> true or false
    # User fragen, zurück: true oder false
  end # Helfer-Methode
end
```



Ausnahmebehandlungen zu Beginn:



1. Quelldatei lässt sich nicht zum Lesen öffnen
 - Fehlermeldung an Anwender (Einzeiler)
 - Abfrage "Copy from: " wiederholen

 2. Quelldatei nicht vorhanden
 - Wie (1), aber mit entsprechend anderer Meldung

 3. (*) Zieldatei schon vorhanden
 - Sicherheitsabfrage: "Overwrite (y/n)?"
 - Mittels catch/throw implementieren (keine "Ausnahme")

 4. Zieldatei lässt sich nicht zum Schreiben öffnen
 - Fehlermeldung an Anwender (Einzeiler)
 - Abfrage "Copy to: " wiederholen

 5. Eingabe war leer bzw. Ctrl-C bzw. Ctrl-D
 - Für beide Fälle (Frage nach Quelle wie auch Frage nach Ziel)
 - Gewünschte Aktion: Programm regulär beenden.
-



Spätere Ausnahmebehandlungen



- Beim Lesen aus geöffneter Datei stößt man irgendwann auf das Dateiende.
 - Ggf. Fehler wie EOFError abfangen!
- Beim Schreiben in eine geöffnete Datei können ebenfalls Probleme auftauchen.
 - Beispiel: Ein überlaufendes Dateisystem, etwa beim Kopieren auf eine Diskette.
 - Entsprechenden Fehler abfangen. Testen Sie den Fall z.B. mit einer Diskette oder einem (fast vollen) USB-Stick!
- Blockweises Kopieren:
 - Besondere Behandlung des letzten, angebrochenen Blocks?



(*) Optionaler Teil:

- Hintergrund: Nach %x(...) steht in \$? ein Fehlercode. \$? != 0 ist eine Fehlerbedingung. Eine Ausnahme wird aber nicht ausgelöst.
- Werten Sie \$? aus, erzeugen Sie Ausnahmefehler mit raise, gehen Sie dann analog wie in den ersten zwei Fällen vor.
- Erzeugen Sie je nach Wert von \$? passende Exceptions. Führen Sie für den Fall, dass ein Kommando nicht existiert bzw. nicht ausführbar ist, einen eigenen Fehlercode ExecError ein:

```
class ExecError < SystemCallError
  def initialize(errno)
    ...
  end
  def to_s ... end
end
```



- Implementieren Sie zunächst die Kernfunktionen!
 - Implementieren Sie erst nur eine Kopiermethode
 - Tipp: Ca. 40-50 *Sourcecode*-Zeilen genügen, um alle drei Kopiervarianten incl. User Interface zu realisieren.
- Testen Sie dann mit verschiedenen Fehlersituationen

Nicht vorhandene Quelldatei, nicht lesbare Quelldatei, nicht beschreibbare Zieldatei, Zieldatei vorhanden, Ctrl-C / Ctrl-D bei der Eingabe, write: Dateisystem voll.
- Ergänzen Sie nun *exception handling*, bis alle Fälle berücksichtigt sind.
- Nun können Sie aus der ersten Kopiermethode die beiden anderen ableiten und dabei die gemeinsamen Fehlerbehandlungen übernehmen.
 - Tipp: Codemenge der Referenz-Impl. verdreifachte sich etwa.



- Erinnerung
 - Aufrufe zum Umgang mit Ausnahmefehlern:
`raise` bzw. `fail`
`begin ... rescue ... else ... ensure ... end`
 - Ergänzender Mechanismus:
`catch :sym do ... end`
`throw :sym`
- Tests
 - Testen Sie mit einer kleinen, mittleren und großen Datei:
(einige kB, ca. 1MB, \geq ca. 10 MB)
 - Profiling (Aufruf s.o.) bzw. Laufzeitmessung mit "time",
Nutzen Sie die Standardbibliothek „profile“
Welches Verfahren ist besonders effizient?
 - Fehlersituationen simulieren
- Demo beachten im Rahmen der Vorbesprechung!