

Nachtrag: **StandardError** vs. **Exception**

(Quelle: [Honeybadger Blog](#))

- Alle "Anwendungsfehler" erben von StandardError
- Exception vor allem für sehr schlimme Dinge

Exception

- `SignalException::Interrupt`
- `ScriptError::SyntaxError`
- `NoMemoryError`

StandardError

- NoMethodError
- TypeError
- ...

StandardError

```
begin  
  stuff_that_breaks  
rescue => e  
  stuff_that_fixes_things  
end
```

Ableiten StandardError vs. Exception

- Ableiten von Exception bricht erwartetes Verhalten: `rescue => e`
- `rescue Exception => e` bricht alles



Kaffeemaschine mit Exceptions

- Entfernt alle interaktiven Elemente (puts, gets) aus der Klasse Kaffeemaschine.
- Ersetzt Ausgaben durch Rückgabewerte der Methoden.
- Erweitert die Klasse Kaffeemaschine um eine Fehlerausgabe mit Hilfe von Exceptions.
- Definiert hierfür eigene Exceptions, die den drei möglichen Fehlerfällen beim Kaffeekochen entsprechen.
- Sorgt dafür, dass beim Auftreten eines Fehlerfalls die passende Exception geworfen wird.

Module

- Code-Organisation (Namespace)
- Kapselung
- Wiederverwendung (Mixins)

Module

```
module MyModule  
  # [...]  
end
```

Namespaces

```
module MusicPlayer
  VERSION = "0.0.1"
  class Song
  end
end

puts MusicPlayer::VERSION
s = MusicPlayer::Song.new
```

Mixins

- Besser für "cross cutting concerns" als Vererbung (Stuhl, Tisch, Benutzer)
- Verhindert unendliche Vererbungsbäume

Mixins

```
module Logger
  def log(msg) ; puts msg ; end
end

class User
  include Logger
end

u = User.new
u.log("hello")
```

Klassenmethoden

```
class System  
  extend Logger  
end  
System.log("test")
```

Benutzung erkennen

```
module MyModule
  def self.included(klass)
    puts "MyModule wurde in #{klass} inkludiert"
  end
  def self.extended(klass)
    puts "#{klass} wurde durch MyModule erweitert"
  end
end
```

"Concern"

```
module Logger
  module ClassMethods
    def setup_logger; end
  end
  def log ; end
  def self.included(klass); klass.extend(ClassMethods); end
end

class MyClass
  include Logger
end

MyClass.setup_logger
MyClass.new.log("foo")
```

Testen, Teil 2

Assertions

- `assert` / `refute`
- `assert_equal`
- `assert_instance_of`
- `assert_kind_of`
- `assert_nil`
- `refute_nil`
- `assert_raises`
- `assert_match`

Setup

```
class SongTest < MiniTest::Test
  def setup
    @song = Song.new('test', 300)
  end
  def test_duration_in_minutes
    assert_equal 5, @song.duration_in_minutes
  end
end
```

Mocks/ Stubs

- Unit-Tests sollen immer nur eine Einheit testen
- Manche Komponenten sollten beim Testen nicht real ausgeführt werden (Bezahlungssystem, Emails)
- Andere verlangsamen Tests vielleicht unnötig
- Wieder andere liefern nicht die zum Testen notwendigen reproduzierbaren Ergebnisse (Time.now !)
- Bei “Test first”-Ansätzen existieren Komponenten möglicherweise noch gar nicht

Mocks/ Stubs in Minitest

```
mock = MiniTest::Mock.new
mock.expect(:to_s, 'test')
mock.to_s
mock.verify
Time.stub(:now, Time.at(1234567890)) do
  Time.now # 2009-02-14 00:31:30 +0100
end
```

Tipps & Lebenslektionen

- Test first / TDD ausprobieren
- Testen lohnt bei großen / langlebigen Projekten fast immer
- Wenn Testen kompliziert ist, ist das ein Hinweis, dass die Klasse zu komplex ist
- Wenn ein Bug entdeckt wird, immer erst Test schreiben, dann fixen (Regressionstest)
- Auch Tests refactorn (Aber nicht zu sehr!)
- Üben, üben, üben



Kaffeemaschine

Schreibt Tests für Eure Kaffeemaschine.

Code coverage

- Was ist ein guter Test?
- Metriken, z.B. wieviel Code wurde tatsächlich durchlaufen?
- Für Ruby z.B. SimpleCov (`gem install simplecov`)

Debugging

- Skript debuggen: `require "debug"`
- Breakpoint setzen mit `debugger`
- Ausdrücke evaluieren mit `p`
- Hilfe: `h(elp)`
- Kommandobsp.: `list`, `where`, `step`, `next`, `break`, `cont`
- Alternativ: Gems: `byebug`



Debugging

- Benutzt "debug" in einer Kernfunktion der Kaffeemaschine
- Benutzt den Debugger

Core vs. standard library vs. default gems

- Klassen aus Core stehen immer zur Verfügung (ohne require)
- Klassen der Stdlib werden mit Ruby zusammen ausgeliefert und können als gegeben betrachtet werden
- Trotzdem explizites require
- Stdlib besteht aus Klassen, die mit Ruby entwickelt werden, und vorinstallierten Gems (default gems)

Standard library

- Datentypen: `BigDecimal`, `Date`, `IPAddr`, `Set`
- Parser/ Generatoren: `csv`, `json`, `yaml`, `rexml`
- Kryptographie: `digest`, `securereandom`, `openssl`
- Netzwerk: `socket`, `net/http` etc.
- Web: `cgi`, `erb`, `webrick`
- Tools: `irb`, `rake`, `rdoc`, `debug`

CSV-Dateien verarbeiten

- Bibliothek 'csv' zum Lesen und Schreiben
- Gängige Eigenschaften können konfiguriert werden, wie z.B.
 - Trennzeichen
 - Anführungszeichen
 - Zeilenumbruch
 - Kopfzeile
 - Encoding

Lesen

```
require "csv"  
CSV.foreach("/tmp/test.csv") do |row|  
  pp row  
end
```

Schreiben

```
CSV.open("/tmp/test.csv", "w") do |csv|  
  csv << [23, "test", 11]  
  csv << [42, "hallo", 12]  
end
```

Logger

- Simple Bibliothek zum Schreiben von Logfiles
- Ausgabe in Datei oder Ausgabestrom
- Log-Level: debug, info, warn, error, fatal, unknown
- Sinnvolles Standardformat der Nachrichten
- Rotation von Logdateien

Logger

```
require 'logger'  
logger = Logger.new("/tmp/logfile.log")  
logger.level = Logger::WARN  
logger.debug("setup complete")  
logger.error("nothing much is happening...")
```

Fileutils

- Methoden, die benannt sind und funktionieren wie auf der Shell
- Bsp: cp, rm, mv usw.
- Besonders nützlich: `mkdir_p`

Rake

- Ruby-Variante des bekannten make-Tools
- sucht im aktuellen Verzeichnis automatisch nach Datei “Rakefile”
- einfache Syntax, aber volle Ruby-Power
- rake -T zum Auflisten aller Tasks
- Eingebaute Tasks z.B. für Tests

Beispiel Rakefile

```
task :hello do
  puts "hello"
end
task :world do
  puts "world"
end

task :hello_world => [:hello, :world]
```

```
rake hello_world
```

Rake testtask

```
require "rake/testtask"  
Rake::TestTask.new do |t|  
  t.pattern = 'test/**/*_test.rb'  
end
```

```
rake test
```



Addressdatenbank

- Benutzt eine (oder eine andere) Stdlib-Methode für die Persistenz
- YAML, JSON, CSV aus der Standard-Library

Rubygems

- Paketverwaltung für Ruby-Programme und -Bibliotheken
- CLI ähnlich apt-get
- `gem install <paketname>`
- unzählige Pakete verfügbar
- rubygems.org
- ruby-toolbox.com

Bundler

- Deklaration der Abhängigkeiten in `Gemfile`
- Lockfile: `Gemfile.lock` sorgt für reproduzierbare Installationen
- `bundle install`



Praxis

Kaffeemaschinen-Gem bauen

- Paketiert die Kaffeemaschine als gem
- Falls noch Zeit ist: Schreibt ein Kaffeemaschinen-Kommando und paketierte es ins gem

Kaffeemaschinen-Gem

- `bundle gem` erzeugt "scaffold"