Monkey-Patching

Sprache der offenen Klassen

Ruby-Klassen sind “offen”

- Existierende Klassen und Objekte können erweitert werden
- Neue Methoden ergänzen oder Module inkludieren

Beispiel: `Numeric#answer?`

HPI Hasso Plattner Institut

Numeric

- Gemeinsame Elternklasse von `Integer` und `Float`

```ruby
class Numeric
  def answer?
    self == 42
  end
end

21.answer?  # => false
5.5.answer? # => false
42.answer?  # => true
```

Beispiel: `Integer#days`

Dauer mehrerer Tage in Sekunden zurückgeben:

```
class Numeric
  def days
    self * 24 * 60 * 60
  end
end

2.days   # => 172800 (seconds)
2.5.days # => 216000.0 (seconds)
```

Warum?

- Erhöhte Ausdrucksstärke und flüssigeres Lesen

```
Time.now + 7.days
```

Beispiel: `String#blank?`

Testen ob ein String nur aus Leerzeichen besteht:

```
class String
  def blank?
    chars.all? {|c| c == " " }
  end
end

"".blank?      # => true
" ".blank?     # => true
"  ".blank?    # => true
" Ja ".blank? # => false
```

Intermezzo: Reguläre Ausdrücke

HPI Hasso Plattner Institut

Reguläre Ausdrücke

- Native Syntax in Ruby
- Akzeptiert von vielen Methoden zum Filtern, Suchen oder Ersetzen von Zeichenketten
- Mächtige Regex-Engine enthalten (Oniguruma)

```ruby
str = "aabbcc aabbcc"

if str =~ /a*b*c*/
  puts "It has an ABC!"
end

str.gsub /b+/, "B"
# => “aaBcc aaBcc”
```

Intermezzo: Reguläre Ausdrücke

`String#blank?`

- Nicht nur auf Leerzeichen testen
- Tabulator, Neue Zeile, ...

```ruby
class String
  def blank?
    not self =~ /[^\s]+/
  end
end

" ".blank?  # => true
"\n".blank? # => true
"\t".blank? # => true
```

Eigenes Modul inkludieren

Vorteile

- Zuordnung als Erweiterung bleibt erkennbar
- Fehler sind besser lokalisierbar

```ruby
module Answer
  def answer?
    self == 42
  end
end

Numeric.include Answer

42.answer? # => true
```

Module erweitern

- Auch Module sind erweiterbar, z.B. `Enumerable`
- Achten auf tatsächlich verfügbare Methoden:

```ruby
module Enumerable
  def second
    self[1]
  end
end

[1, 2, 3, 4].second # => 2

# Aber:

(1..100).second
# => NoMethodError (undefined method `[]' for 1..100:Range)
```

Module erweitern

- Enumerable arbeitet nur mit `#each`
  Wegen z.B. (unendlichen) Reihen oder IO

```ruby
module Enumerable
  def second
    take(2)[1]
  end
end

[1, 2, 3, 4].second # => 2
(1..100).second  # => 2
```

Warnung: Nur dosiert verwenden

Gefahren

- Schwere Lokalisierung der tatsächlich aufgerufen Methode
- Unerwartete Veränderung von anderen Programmteilen

Vorteile

- Elegante Ergänzung hilfreicher Methoden
- Keine Unterklassen nötig

Empfehlung

- Auf einfache Funktionen beschränken
- Mit Vorsicht zu genießen

Monkey-Patching