HPI Hasso Plattner Institut

Ruby-Programme strukturieren

Ruby-Programme strukturieren

- Objektorientierung
  - Funktionalität auf Klassen verteilen
  - Komplexe Aufgaben durch Komposition von Objekten

- Dateien
  - Konvention: Eine Datei pro Klasse
  - `require`: Dateien laden (absolut oder aus dem Load-Path)
  - `require_relative`: Dateien mit relativem Pfad laden

- Namespaces
  - Module und Klassen verschachteln

require

require(name_or_path)

- Ruby-Skripte oder “Shared Libraries” (*.so, *.dll) laden
- Verzeichnispfad oder Suche in vorbestimmten Verzeichnissen
- `LoadError`-Exception, wenn Datei nicht auffindbar

```
require "fileutils" # => true

require "missing_file"
# LoadError (cannot load such file -- missing_file)
```

require

- Dateien aus Rubys stdlib laden:

```ruby
Time.respond_to? :parse # => false

require "time" # => true
Time.respond_to? :parse # => true
```

- Dateien mit Pfad laden:

```ruby
require "/home/openhpi/quizzes" # => true
require "./videos" # => true
```

- Datei wird nur einmal geladen (Erweiterung optional):

```ruby
require "time.rb" # => true
require "time" # => false
```

$LOAD_PATH

```ruby
puts $LOAD_PATH
# /usr/local/lib/ruby/gems/2.5.0/gems/did_you_mean-1.2.0/lib
# /usr/local/lib/ruby/site_ruby/2.5.0
# /usr/local/lib/ruby/site_ruby/2.5.0/x86_64-linux
# /usr/local/lib/ruby/site_ruby
# /usr/local/lib/ruby/vendor_ruby/2.5.0
# /usr/local/lib/ruby/vendor_ruby/2.5.0/x86_64-linux
# /usr/local/lib/ruby/vendor_ruby
# /usr/local/lib/ruby/2.5.0
# /usr/local/lib/ruby/2.5.0/x86_64-linux
```

- Liste von Verzeichnissen, aus denen Ruby-Dateien mit `require` geladen werden können

$LOAD_PATH

- Globale Variable
  - Zusätzliche Pfade können hinzugefügt werden

```
require "./lib/utils" # => true
```

- Bestimmtes Verzeichnis ans Ende hängen:

```
$LOAD_PATH << "./lib" # => [..., "./lib"]
require "utils" # => false
```

- Aktuelles Verzeichnis an den Anfang hängen:

```
$LOAD_PATH.unshift __dir__ # => [".", ...]
require "lib/utils" # => false
```

Relative Pfade laden

- Relative Pfade sind relativ zum Arbeitsverzeichnis:

lib/task.rb

```
class Task; end
```

lib/incorrect.rb

```
require "./task"
```

lib/correct.rb

```
require "./lib/task"
```

```
require "./lib/incorrect"
# LoadError (cannot load such file -- ./task)

require "./lib/correct"
# => true
```

Relative Pfade laden

- Aufwendige Lösung:

lib/expand.rb

```
require File.expand_path("task", __dir__)
```

- Mit `require_relative`:

lib/relative.rb

```
require_relative "task"
```

```
defined?(Task) # => nil

require "./lib/relative" # => true
defined?(Task) # => "constant"
```

Dateien laden

- “Bibliotheks”-Code:
  - Abhängigkeiten explizit in jeder Datei mit `require`
  - Keine Seiteneffekte in geladenen Dateien!

- Dateien aus externen Bibliotheken: `require`
- Für Applikationen mit Bibliotheks-Verzeichnissen:
  - `$LOAD_PATH` anpassen + `require`
  - oder `require_relative` für Pfade innerhalb der Bibliothek

Ordnung im Chaos: Namespaces

HPI Hasso Plattner Institut

Namespaces

- Verschachtelte Module als Namensraum

```
module A
  module B
    class MyClass
      # ...
    end
  end
end

A::B::MyClass.new
```

Ordnung im Chaos: Namespaces

Namespaces

- Konvention: Langer Name entspricht Datei-Pfad

lib/a/b/my_class.rb

```
module A
  module B
    class MyClass
      # ...
    end
  end
end
```

main.rb

```
require "a/b/my_class"

A::B::MyClass.new
```

HPI Hasso Plattner Institut

Ruby-Programme strukturieren