



Praktikum zu LV 7328 - Ruby: **Übung 02**

Klassen und Methoden anlegen
Die Klasse Array



Organisatorisches



- Arbeitsverzeichnis:

`~/lv/ruby/02/`

- Dateinamen:

`02-container.rb` # neu erstellen & abgeben

- Werkzeuge:

`ruby` # Der Interpreter

`emacs` # mit Ruby-Mode. NICHT X-Emacs

`scite` # Ein portabler Editor, auch

 # mit Ruby-Mode

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

- Vorlagen:

`(keine)`



- Allgemeine Beschreibung
 - Erzeugen Sie zwei „Klassiker“ der Informatik als einfache Klassen auf der Basis der eingebauten Klasse `Array`
 - Entwickeln Sie dazu passenden Testcode
- Material, Hinweise:
 - Nutzen Sie Attribute zum Speichern der internen Werte ihrer Objekte.
 - Nutzen Sie das Delegationsprinzip – delegieren Sie die eigentliche Arbeit an Methoden der Klasse `Array`
 - Die Klasse `Array` ist sehr leistungsfähig. Kenntnis ihrer Methoden ist lohnend!



Die Aufgabe



A: Implementieren Sie eine Klasse **Queue** (FIFO) mit folgenden Methoden:

```
initialize (max_size=0)
    # Eine leere Warteschlange mit max_size Plätzen
    # anlegen. max_size==0 : Keine Platzbegrenzung

enqueue (obj)                --> anInteger
    # Objekt in Warteschlange einreihen

dequeue                       --> anObject
    # Objekt am Ende der Warteschlange entnehmen

peek                          --> anObject
    # Referenz auf Objekt am Ende der Warteschlange

length, size                 --> anInteger
    # Länge der Warteschlange. Nutzen Sie „alias“!

clear                         --> anInteger
    # Warteschlange löschen, alte Länge liefern.

empty?                       --> aBoolean
    # true, falls Warteschlange leer
```



Die Aufgabe



B: Implementieren Sie eine Klasse **Stack** (LIFO) mit folgenden Methoden:

```
initialize (max_size=0)
    # Einen leeren Stack mit max_size Fächern anlegen
    # max_size==0 : Keine Platzbegrenzung

push (obj)                --> aBoolean
    # Objekt auf den Kellerspeicher legen

pop                        --> anObject
    # Objekt vom Kellerspeicher nehmen

peek                       --> anObject
    # Referenz auf oberstes Objekt des stacks

length, size, depth       --> anInteger
    # Anzahl gespeicherter Objekte. Nutzen Sie „alias“!

clear                      --> anInteger
    # Kellerspeicher löschen, alte Tiefe liefern.

empty?                     --> aBoolean
    # true, falls Kellerspeicher leer
```



C: Schreiben Sie Test-Code zu beiden Klassen

Testen Sie möglichst viele Aspekte der Klassen

- a) Testen Sie jede Methode.
- b) Testen Sie Grenzfälle wie leere Warteschlange oder Stack-Überlauf.
- c) Ausgabe einer Test-Statistik?

HINWEISE

- Beide Klassen sowie der Test-Code sollen gemeinsam in der Datei „02-container.rb“ stehen.
- (*) Ausbaustufe 1: Redundanzen vermeiden mittels einer gemeinsamen Basisklasse „Container“
- (*) Ausbaustufe 2: enqueue() und push() akzeptieren auch eine Liste von Objekten.



Hilfestellung: Etwas Source-Code als Ausgangspunkt

```
class Queue
  def initialize(max_size=0)
    @buf = []
    @max_size = max_size
  end

  def size
    @buf.size # Arbeit an @buf delegieren...
  end
end

# Test-Code:
q = Queue.new(5)
puts q.size
0
```