Institut für Informatik
Prof. Dr. Oliver Vornberger
Nils Haldenwang, B.Sc.

Universität Osnabrück
http://www-lehre.inf.uos.de/~ror
03.05.2012

Agile Webentwicklung mit Ruby on Rails

Sommersemester 2012

Blatt 2

Aufgabe 1: Vector

Schreiben Sie eine Klasse `Vector`, welche die folgenden Anforderungen erfüllt:

Der Initializer erwartet als Parameter ein Array mit den Elementen des Vektors. Die Dimension des Vektors soll daraus automatisch bestimmt werden und später auch abfragbar sein.

Um eine Array-Repräsentation des Vektors zu erhalten, wird die Methode `to_a` bereitgestellt, welche ein Array mit allen Elementen des Vektors liefert.

Die Methode `to_s` liefert eine geeignete String-Repräsentation des Vektors zurück.

Um zwei Vektoren zu vergleichen, soll die Methode `==(other)` geeignet überschrieben werden. Zwei Vektoren gelten dabei als gleich, wenn ihre Elemente gleich sind.

Die Methode `+(other)`, addiert den übergebenen Parameter `other` zum aktuellen Vektor. Wenn `other` ein `Vector` ist, dann soll die Vektoraddition[1] durchgeführt werden. Wird ein skalarer Wert übergeben, so soll der Wert auf alle Elemente des Vektors addiert werden. Der ursprüngliche Vektor darf nicht verändert werden. Die Methode liefert stets einen neuen Vektor mit entsprechenden Elementen zurück.

[1] http://de.wikipedia.org/wiki/Vektor#Addition

Zur Multiplikation steht die Methode `*(other)` zur Verfügung, welche sich ähnlich verhält wie `+(other)`. Wird ein `Vector` übergeben, so soll die Skalarmultiplikation[2] durchgeführt werden. Bei Übergabe eines skalaren Wertes wird jedes Element des Vektors mit diesem multipliziert. Auch hier soll der ursprüngliche Vektor nicht verändert werden.

Die Methode `-(other)` funktioniert analog zu `+(other)`.

Zur Iteration über alle Elemente soll die Methode `each` verwendet werden können. Sie verhält sich analog zur Methode `Array#each`. Sie erwartet einen Block mit einem Parameter und ruft diesen einmal mit jedem Element des Vektors auf.

Die Methode `map` verhält sich analog zu `Array#map`. Sie erwartet einen Block und liefert einen neuen `Vector`, dessen Elemente jeweils das Resultat des Blockaufrufes mit dem Ursprungselement sind.

Achten Sie an den nötigen Stellen auf eine geeignete Fehlerbehandlung. Definieren Sie dazu gegebenenfalls eine eigene Fehlerklasse.

[2] http://de.wikipedia.org/wiki/Skalarprodukt

Aufgabe 2: Ancestor-Chain

Betrachten Sie die Datei `modules.rb`.

Welche der dort in die Klasse `String` eingefügten Methoden `word_list` wird in Zeile 28 aufgerufen? Begründen Sie ihre Antwort mit einer Zeichnung der Ancestor-Chain.

Versuchen Sie mit den Ihnen bisher bekannten Mitteln den Konflikt aufzulösen, so dass der Klasse String alle vorgesehenen Operationen zur Verfügung stehen.

Aufgabe 3: Pretty-Print der Ancestors

Erweitern sie alle Klassenobjekte um die Methode `pretty_print_ancestors`. Die Methode soll die Kette der Vorfahren ausgeben. In der Ausgabe soll ersichtlich sein, ob es sich bei dem Vorfahren um eine Klasse oder um ein Modul handelt.