PM1/PT Ruby: Objektsammlungen und Iteratoren

Konzepte und Techniken

- Objektsammlungen (Array)
- Anonyme Objekte
- Schnittstelle vs. Implementierung
- Objektsammlungen mit beliebigen Schlüsselwerten (Dictionaries / Hashes)
- Iteratoren für Hashes
- Mengen

Auktion: ein weiteres Beispiel zu Objektsammlung

- Das Projekt Auktion modelliert einen kleinen Ausschnitt einer Onlineauktion.
- Eine Auktion besteht aus Gegenständen den Posten.
- Personen können Gebote für diese Gegenstände abgeben, indem sie Beträge für einen Posten bieten. Dabei werden die Posten über ihre Nummerierung angesprochen.
- Posten werden von einer Auktion erzeugt und nummeriert.
- Beim Bieten gewinnt immer das Höchstgebot. Liegt eine Gebot unter dem Höchstgebot, so erscheint ein entsprechender Hinweise und das Gebot wird verworfen.
- Nach außen in der Bedienung der Auktion treten nur die Objekte Auktion und Person in Erscheinung.
- Die Klassen Gebot und Posten sind interne Klassen, die für die Verwaltung einer Auktion verwendet werden.
- Die Klasse Gebot und Person sind sehr einfach gehalten, so dass diese dem Selbststudium überlassen bleiben.

Posten zur Verwaltung der Gegenstände einer Auktion

- Die Klasse ***Posten*** speichert eine Beschreibung, eine Postennummer, die beim Erzeugen übergeben wird und das höchste Gebot, das für den Posten abgegeben wurde.
- Die Methode ***hoeheres_gebot?(gebot)*** prüft, ob ***gebot*** das Höchstgebot übersteigt und speichert ggf. das neue Höchstgebot. War das neue Gebot das Höchstgebot, dann wird ***true*** ansonsten ***false*** zurückgegeben.
- ***to_s()*** liefert für den Posten eine lesbare Zeichenkette.

```
class Posten
  attr_reader :posten_nummer,
    :beschreibung, :hoechstes_gebot
  def
    initialize(posten_nummer,beschreibu
    ng)
     @posten_nummer = posten_nummer
     @beschreibung = beschreibung
     @hoechstes_gebot
  end
  def hoeheres_gebot?(gebot)
      if @hoechstes_gebot.nil?()
        @hoechstes_gebot=gebot
        return true
      end
      if gebot.hoehe() <=
    @hoechstes_gebot.hoehe()
          return false
      end
      @hoechstes_gebot = gebot
      return true
  end
  def to_s()
     return "#@posten_nummer
    #@beschreibung"
  end
end
```

Klasse *Auktion*

- Objektmethoden der Klasse ***Auktion***:
  - ***initialize()***: Erzeugt ein leeres Array für die Posten und setzt den Zähler für Posten.
  - ***posten_anmelden(beschreibung***): Erzeugt zu einer Beschreibung ein Posten Objekt. Dabei wird eine Nummer für den Posten übergeben.
  - ***posten_liste_ausgeben()***: Gibt die Posten der Auktion auf der Konsole aus.
  - ***bieten_fuer(posten_nummer,person,betrag***): eine Person kann für einen Posten, der über seine Nummer identifiziert wird einen Betrag bieten.
  - ***posten_mit_nummer(nummer***): Liefert zu einer Postennummer das zugehörige Objekt.

```
class Auktion
  def initialize()
    @posten_liste = []
    @naechste_posten_nummer = 1
  end
  def posten_anmelden(beschreibung)
    @posten_liste<<
   Posten.new(@naechste_posten_nummer,beschreibung)
    @naechste_posten_nummer +=1
  end
end
```

Klasse *Auktion*

- *posten_anmelden(beschreibung)*:
  - Erzeugt zu einer Beschreibung ein Posten Objekt. Dabei wird eine Nummer für den Posten übergeben.
  - Merkt sich das Postenobjekt in der Postenliste. (Fügt den Posten mit << hinten an die Liste an.)

```
class Auktion
  ...
  def posten_anmelden(beschreibung)
    @posten_liste << Posten.new(@naechste_posten_nummer,beschreibung)
    @naechste_posten_nummer +=1
  end
  ...
end
```

Klasse *Auktion*

- ***posten_liste_ausgeben()***:
  - Gibt die Posten der Auktion auf der Konsole aus.
  - Iteriert über die Postenliste.
  - Verlässt sich dabei auf ***to_s()*** Methode von Posten. Es wird hier nur puts auf den Posten aufgerufen.

```
class Auktion
  ...

  def  posten_liste_ausgeben()
    @posten_liste.each { |posten| puts("#{posten}") }
  end
 ...
end
```

Klasse *Auktion*

- ***bieten_fuer(posten_nummer,person,betrag)***:
  - eine Person kann für einen Postennummer einen Betrag bieten.
  - Die Auktion sucht zu der Postennummer das zugehörige Postenobjekt. Dazu verwendet sie die interne Methode ***posten_mit_nummer(posten_nummer)***.
  - Ist der Posten in der Sammlung enthalten, wird mit Hilfe des Postens geprüft, ob der angebotene Betrag das Höchstgebot ist (***gewaehlter_posten.hoeheres_gebot?(Gebot.new(person,betrag))***).
  - Je nach Ergebnis der Anfrage an den gewählten Posten werden entsprechende Ausgaben auf die Konsole gemacht.

Klasse *Auktion*

```
class Auktion
  ...
 def bieten_fuer(posten_nummer,person,betrag)
    gewaehlter_posten = posten_mit_nummer(posten_nummer)
    if !gewaehlter_posten.nil?()
      erfolgreich =
            gewaehlter_posten.hoeheres_gebot?(Gebot.new(person,betrag))
      if (erfolgreich)
        puts "Gebot fuer Posten Nummer #{posten_nummer} war erfolgreich"
        return
      end
      hoechstes_gebot = gewaehlter_posten.hoechstes_gebot()
      puts("Posten Nummer #{posten_nummer}: Hoechstes Gebot bereits:
   #{hoechstes_gebot.hoehe()}")
    end
  end
  ...
end
```

Klasse *Auktion*

- *posten_mit_nummer(nummer)*
  - liefert zu einer Postennummer das zugehörige Objekt
  - Iteriert über die Postenliste (*find*) und sucht nach dem Posten, dessen Nummer mit der angefragten *nummer* übereinstimmt.
  - Liefert den Posten mit der Nummer zurück, falls vorhanden, sonst *nil*.

```
class Auktion
 ...
 def posten_mit_nummer(nummer)
    posten =  @posten_liste.find{|pst| pst.posten_nummer == nummer }
    if (posten.nil?())
      puts "Posten mit #{nummer} existiert nicht "
    end
    return posten
  end
  ...
end
```

Besonderheiten im Projekt Auktion

- Die Klasse ***Auktion*** verwendet intern die Klassen ***Posten*** und ***Gebot***, die nach außen verborgen bleiben.
- Die Klasse ***Auktion*** erzeugt in der Methode ***bieten_fuer*** ein Objekt der Klasse ***Gebot***, ohne dass es sich dieses Objekt in einer Variable merkt. Es handelt sich um ein **anonymes Objekt**.
- Wenn es für ein Objekt in einer Klasse nur an einer Stelle Verwendung gibt, dann brauchen wir uns dieses Objekt nicht in einer Variable zu merken, sondern verwenden anstelle dessen ein anonymes Objekt.
- Die Klasse ***Auktion*** verwendet den Iterator ***find,*** der Objekte in einer Sammlung aufspürt, die eine Bedingung erfüllen.
- Iteratoren, die mit Bedingungen arbeiten, geben ***nil*** zurück, wenn kein Objekt in der Sammlung die Bedingung erfüllt.
- Ob ein solcher Iterator erfolgreich war, können wir mit dem Test des Ergebnisses auf ***nil?()*** prüfen.

Übungen

- **Ü6-a-1:** Fügen Sie bitte die Methode ***beenden*** zur Klasse ***Auktion*** hinzu! Die Methode iteriert über alle Posten und gibt alle Details aus. Posten mit mindestens einem Gebot sollen als verkauft angesehen werden. Bei verkauften Posten sollen der Bieter und das Gebot ausgegeben werden.
- **Ü6-a-2:** Schreiben Sie bitte für die ***Auktion*** die Methode ***unverkaufte_posten()***, die ein Array mit allen Posten zurückgibt, für die kein Gebot abgegeben wurde!
- **Ü6-a-3:** Schreiben Sie bitte für die ***Auktion*** die Methode ***verkaufte_posten()***, die ein Array mit allen Posten zurückgibt, für die ein Gebot abgegeben wurde!
- **Ü6-a-4:** Schreiben Sie bitte für die ***Auktion*** die Methode ***loesche_posten(nummer)***, die den Posten mit der angegeben Nummer aus der Auktion löscht.
- **Ü6-a-5:** Schreiben Sie bitte für die Klasse eine Methode ***verkaufte_posten_loeschen()***, die alle verkauften Posten aus der ***Auktion*** löscht. Nutzen Sie dazu die Methoden aus **Ü6-a-3** und **Ü6-a-4.**
- **Ü6-a-6:** Schreiben bitte Sie die Methode ***verkaufte_posten_loeschen()*** mit einer passenden Löschmethode der Klasse ***Array***.

Übungen

- **Ü6-a-7:** Schreiben Sie bitte für die Klasse ***Auktion*** die Methode ***umsatz()***, die die Höchstgebote aller verkauften Posten aufaddiert! Schreiben Sie dies mit dem Basisiterator und der Methode ***inject***!

- **Ü6-a-8**: Schreiben Sie bitte für die Klasse ***Auktion*** die Methode ***beste_posten()***, die aus allen verkauften Posten diejenigen ermittelt, die das höchste Gebot erhalten haben und diese als Array zurückgibt!

Projekte Technischer Kundendienst

- Ziel ist es einen technischen Kundendienst zu entwickeln, der inhaltlich auf Kundenfragen eingeht.
- Die Lösung wird in drei Schritten (Projekten) entwickelt, wobei in jedem Projekt das Verhalten des Systems verbessert wird.
- Dabei werden neue Bibliotheksklassen eingeführt und die Verwendung bereits bekannter Klassen vertieft und erweitert.
- Sie sollen mit diesen Projekt auch den selbständigen Umgang mit den Ruby-Bibliotheken üben.

- Der technische Kundendienst ist angelehnt an das berühmte **Eliza** Programm, das von Joseph Weizenbaum in den 60‘er Jahren am MIT entwickelt wurde.
- Wir beginnen mit dem Projekt ***v-6-b-TechnischerKundendienst1***

Klasse *Beantworter*

- enthält eine sehr unkommunikative Methode zur Beantwortung von Fragen, die immer die gleiche Zeichenkette liefert.

```
class Beantworter

  def generiere_antwort()
    return  "Das klingt " +
   "interessant. Erzählen " +
   "Sie mehr ..."
  end

end
```

Klasse *KundendienstSystem*

Methoden

- ***initialize()***: erzeugt einen ***Eingabeleser*** und einen ***Beantworter***.
- ***starten()***: gibt zu jeder Eingabe vom Eingabeleser eine Antwort des Beantworters auf der Konsole aus, solange der Benutzer nicht **'end'** eingegeben hat.
- ***willkommenstext_ausgeben()***: Gibt einen vorgefertigten Text aus.
- ***abschiedstext_ausgeben():*** Gibt einen vorgefertigten Text aus.

```
require "Eingabeleser"
require "Beantworter"

class KundendienstSystem

  def initialize()
    @leser = Eingabeleser.new()
    @antworter = Beantworter.new()
  end
end
```

Klasse *KundendienstSystem*

- ***starten()***: gibt zu jeder Eingabe vom Eingabeleser eine Antwort des ***Beantworters*** auf der Konsole aus, solange der Benutzer nicht **'end'** eingegeben hat.
- Der boolesche Ausdruck ***!fertig*** kontrolliert die Schleife. Wenn der Benutzer **'end'** eingibt, wird die Variable fertig auf ***true*** gesetzt und mit dem nächsten Schleifendurchlauf das Programm beendet.

```
def starten()
    fertig = false
    while (!fertig)
      eingabe = @leser.naechste_eingabe();
      if (eingabe.include?('end'))
        fertig = true
      else
        antwort = @antworter.generiere_antwort()
        puts(antwort)
      end
    end
end
```

Methoden auf *Zeichenketten*

- Frage: Was genau ist das Verhalten der Methode ***include?(a_str)*** in der Klasse ***String***?
- Googlen (Ruby ***class:String***).
- Das Verhalten ist nicht das Gewünschte. Wir wollen, dass der Benutzer nur **end** eingibt.
- Zulassen wollen wir beliebig viele führende und endende Leerzeichen in der Eingabe.
- Zulassen wollen wir auch alle Varianten von Groß- und Kleinschreibung.

- **Ü-6-a-9:** Welche Eingaben außer **end** in der Originalfassung würden das Kundendienst- System beenden?
- **Ü-6-a-10**: Mit welchen Methoden der Klasse ***String***, lässt sich dann das rechts geforderte Verhalten erreichen?

RubyDoc der Klasse *String*

Class: String - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Mash Hilfe

http://ruby-doc.org/core/classes/String.html#M000819

Ruby class:String

Erste Schritte Aktuelle Nachrichten

Mash Maker is disabled. Click on the blue 'M' icon to enable it.

Class: String

Class **String**

In: string.c
lib/jcode.rb
lib/mkmf.rb
lib/scanf.rb
ext/nkf/lib/kconv.rb

A `String` object holds and manipulates an arbitrary sequence of bytes, typically representing characters. String objects may be created using `String::new` or as literals.

Because of aliasing issues, users of strings should be aware of the methods that modify the contents of a `String` object. Typically, methods with names ending in ``!'' modify their receiver, while those without a ``!'' return a new `String`. However, there are exceptions, such as `String#[]=`.

Methods

% * + << <=> == =~ [] []= block_scanf capitalize capitalize! casecmp center chomp chomp! chop chop chop! chop! concat count crypt delete delete delete! delete! downcase downcase! dump each each_byte each_char each_line empty? end_regexp eql? gsub gsub! hash hex include? index initialize_copy insert inspect intern iseuc issjis isutf8 jcount jlength jsize kconv length ljust lstrip lstrip! match mbchar? new next next! oct quote replace reverse reverse! rindex rjust rstrip rstrip! scan scanf size slice slice! split squeeze squeeze squeeze! squeeze! strip strip! sub sub! succ succ succ succ! succ! sum swapcase swapcase! to_f to_i to_s to_str to_sym toeuc tojis tosjis toutf16 toutf8 tr tr tr! tr! tr_s tr_s tr_s! tr_s! unpack upcase upcase! upto

Included Modules

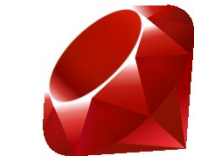
Comparable Enumerable

Constants

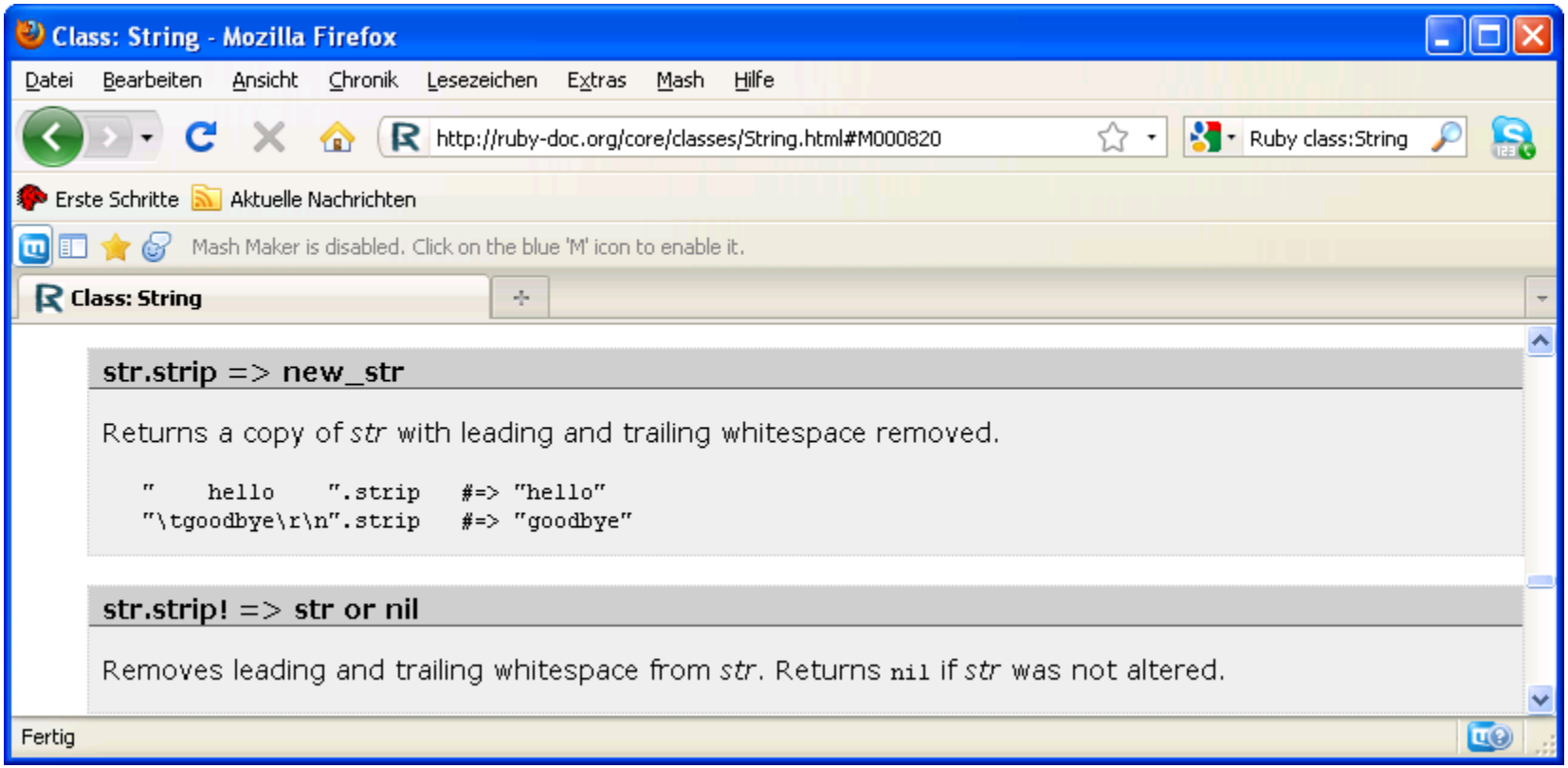
PATTERN_SJIS	= '[\x81-\x9f\xe0-\xef][\x40-\x7e\x80-\xfc]'
PATTERN_EUC	= '[\xa1-\xfe][\xa1-\xfe]'
PATTERN_UTF8	= '[\xc0-\xdf][\x80-\xbf]|[\xe0-\xef][\x80-\xbf][\x80-\xbf]'
RE_SJIS	= Regexp.new(PATTERN_SJIS, 0, 'n')
RE_EUC	= Regexp.new(PATTERN_EUC, 0, 'n')
RE_UTF8	= Regexp.new(PATTERN_UTF8, 0, 'n')
SUCC	= {}
HashCache	= {}
TrPatternCache	= {}
DeletePatternCache	= {}
SqueezePatternCache	= {}

Public Class methods

String.new(str="") => new_str



RubyDoc der Methode *strip*



Schnittstellen versus Implementierungen

- Wenn wir uns die Dokumentation der Bibliotheksklassen anschauen, dann sehen wir
  - Den Klassennamen mit allen Methodennamen und ggf. eine allgemeine Beschreibung der Funktion der Klasse
  - Für jede Methode, den Namen und deren Ergebnis (hinter dem =>)
  - eine Beschreibung des Verhaltens der Methoden
  - ggf. Beispiele
- Diese Informationen beschreiben die **Schnittstelle** einer Klasse und nicht deren **Implementierung**.
- Wenn wir Bibliotheksklassen nutzen interessiert uns die **Schnittstelle** der Klasse.
- Die Schnittstelle informiert über das **was** eine Klasse kann. Sie abstrahiert von den Details der Implementierung.
- Die **Implementierung** ist der Quelltext der Klasse. Diese müssen wir **nicht** kennen, um die Klasse zu nutzen. Die Implementierung gibt Auskunft über **wie** die Klasse ihren Job erledigt.
- In Teamprojekten sprechen sich die Teams über die Schnittstelle der Klassen ab, die sie für die jeweiligen anderen Teams entwickeln.

Übung

- **Ü-6-a-11:** Implementieren Sie bitte die verbes-serte Version für das Erkennen der Ein-gabe von '**end**,!

- **Ü-6-a-12:** Können wir die Zeichenketten mit dem == auf Gleichheit prüfen? Schau-en Sie in der Methodendefinition nach!

Beantworter optimieren: Zufälliges Verhalten einbauen

- In ***v-6-a-TechnischerKundendienst2*** sollen jetzt aus einer Liste von möglichen Texten, Antworten zufällig gewählt werden.

- Dazu müssen wir den ***Beantworter*** erweitern:
  1. Wir legen bei der Erzeugung des Beobachters eine Sammlung von Zeichenketten an.
  2. Wir wählen in der Methode ***generiere_antwort()*** eine Zufallszahl, die im Intervall [0,length-1] liegt und verwenden diese, um eine Antwort aus der Sammlung zu lesen. Dabei gibt **length** die Länge der Sammlung an.

- **Ü-6-a-13:** Wie werden in Ruby Zufallszahlen generiert? Wie stellen wir sicher, dass diese Zahl im geforderten Intervall liegt?

Übungen – Legen Sie für die Übungen 13-17 eine Klasse *ZufallszahlenTest* an

- **Ü-6-a-14**: Schreiben Sie bitte eine Methode würfeln, die Zahlen zwischen 1 und 6 würfelt!
- **Ü-6-a-15**: Schreiben Sie bitte eine Methode antwort, die zufällig die Antworten „ja“, „nein“, „bin nicht sicher“ zurückgibt!
- **Ü-6-a-16**: Schreiben Sie bitte eine Methode 6 aus 49, die eine Lottoziehung simuliert! Achten Sie darauf, dass Zahlen nur genau einmal vorkommen dürfen.
- **Ü-6-a-17**: Schreiben Sie bitte eine Methode mit den Parametern ***min*** und ***max***, die eine Zufallszahl im Intervall min, einschließlich ***max*** generiert!
- **Ü-6-a-18:** Stellen Sie sich für einen kurzen Augenblick vor Sie sitzen in einem Seminar in einem höheren Semester. Zu Beginn wird die Reihenfolge der Seminarvorträge festgelegt. Ihre Dozentin beschließt dieses aus Gründen der Gerechtigkeit zufällig zu tun, indem Sie jedem Teilnehmenden eine eindeutige Zufallszahl zuordnet. Die Teilnehmer sind alphabetisch nach Namen geordnet. Schreiben Sie bitte eine Methode, die diese Reihenfolge zufällig berechnet und als Array zurückgibt!
- **Ü-6-a-19**: Schreiben Sie bitte die Methode ***generiere_antwort()***, die aus einer beliebigen Sammlung eine Antwort zufällig auswählt!

Beantworter optimieren: Zufälliges Verhalten einbauen

- Die Methode für das Erzeugen von Zufallszahlen ist ***rand(grenze)***.
- ***rand(grenze)*** erzeugt ganzzahlige Zufallszahlen von 0 bis ausschließlich ***grenze***, wenn ***grenze*** eine ganze Zahl ist.
- Wenn wir die Länge ***length*** der Sammlung als Argument für ***rand*** verwenden, bekommen wir Zufallszahlen zwischen 0 und ***length-1,*** die wir als Index für die Sammlung möglicher Antworten verwenden können.

```
class Beantworter
  def initialize()
    @antworten =
          antworten_liste_erzeugen()
  end

  def generiere_antwort()
    zufalls_index =
          rand(@antworten.length)
    return  @antworten[zufalls_index]
  end
end
```

Der Vollständigkeit halber …

```
def antworten_liste_erzeugen()
   ["Das klingt seltsam. Können Sie das Problem" +
     " ausführlicher beschreiben?",
     "Bisher hat sich noch kein Kunde darüber\n" +
     "beschwert. Welche Systemkonfiguration haben Sie?",
     "Das klingt interessant. Erzählen Sie mehr...",
     "Da brauche ich etwas ausführlichere Angaben.",
     "Haben Sie geprüft, ob Sie einen Konflikt mit" +
     " einer DLL haben?",
     "Das steht im Handbuch. Haben Sie das Handbuch" +
     " gelesen?",
     "Das klingt alles etwas Wischi-Waschi. Haben Sie\n" +
     "einen Experten in der Nähe, der das etwas\n" +
     "präziser beschreiben kann?",
     "Das ist kein Fehler, das ist eine" +
     " Systemeigenschaft!",
     "Könnten Sie es anders erklären?"
   ]
 end
```

Dictionaries (Hashes)

- Kerneigenschaften:
  - Ein Hash ist eine Sammlung von Schlüssel Wert Paaren.
  - Der Zugriff auf die Werte erfolgt über den Schlüssel.
  - Die Ordnung der Schlüssel-Wert Paare ist beliebig und entspricht nicht der Reihenfolge des Einfügens von Schlüssel-Wert Paaren in den Hash.
- Ähnlichkeiten zu Array
  - Ein Hash ist eine Sammlung, kann also beliebig viele Elemente aufnehmen.
  - Es gibt Methoden für den Elementzugriff und –zuweisung.
  - Es gibt noch weitere Gemeinsamkeiten, die wir im Folgenden kennenlernen werden.
- Unterschiede zu Array
  - Der Elementzugriff geht über Schlüssel und nicht über Indizes.
  - Die Elemente sind **nicht** geordnet.

Hashes sind Abbildungen von Schlüsseln auf Werte

- Über einen Schlüssel liefert mir der Hash den zugehörigen Wert.
- Das verhält sich wie eine Abbildung, die Schlüsseln Werte zuordnet.
- Abbildungen sind eindeutig, d.h. jedem Schlüssel ist genau ein Objekt (Wert) zugeordnet.

- Ein gutes Alltagsbeispiel für einen Hash ist ein Telefonbuch.
- Ein Telefonbuch ordnet Namen Telefonnummern zu. Die Schlüssel im Telefonbuch sind die Namen, die Werte die Telefonnummern.
- Wir nutzen den Namen und keinen Index, um Telefonnummern nachzuschlagen.
- Der umgekehrte Weg, über eine Telefonnummer einen Namen zu erfragen, ist mit einer solchen Abbildung nicht so einfach.

Das Telefonbuch

- Wir erzeugen eine Klasse Telefonbuch. Das Telefonbuch hat einen Vorwahlkreis und ist für ein Gebiet gültig. Diese Informationen geben wir bei der Erzeugung mit.
- Wir schreiben eine Methode, mit der wir zu Namen Telefonnummern in das Telefonbuch eintragen können und füllen das Telefonbuch mit 3 Einträgen.
- Wir schreiben eine Methode für das Telefonbuch, die das Suchen von Telefonnummern über Namen ermöglicht.
- Wir schreiben eine Methode, die Einträge in einem Telefonbuch löscht.
- Wir schreiben eine Methode, die alle Namen im Telefonbuch liefert.
- Wir schreiben eine Methode die alle Telefonnummern im Telefonbuch liefert.
- Wir schreiben eine Methode, die zu einer Telefonnummer den Namen liefert.

Das Telefonbuch

- Wir schreiben eine Klasse Telefonbuch. Das Telefonbuch hat einen Vorwahlkreis und ist für ein Gebiet gültig. Diese Information übergeben wir bei der Erzeugung.

- Bei der Erzeugung des Telefonbuchs legen wir einen leeren Hash an (Literal ***{}***) Alternativ erzeugen wir einen leeren Hash über ***Hash.new().***

- Mit Hash-Literalen können bei der Erzeugung Schlüssel-Wert Paare direkt hingeschrieben werden.

- ```
  buch= {
     "hugo" => "0401236790",
     "erna" => "040875638"}
  ```

```
class Telefonbuch

  def
   initialize(gebiet,vorwahl)
    @gebiet = gebiet
    @vorwahl = vorwahl
    @eintraege= {}
  end


end
```
```

Das Telefonbuch

- Die Methode, um Telefonnummern einzutragen, hat zwei Parameter. Der erste ist der Schlüssel, der zweite der Wert.
- Unter dem Schlüssel ***name*** wird der Wert ***nummer*** abgelegt (Elementzuweisung).
  @eintraege[name] = nummer
- Die Methode, um Telefonnummern über einen Namen zu suchen hat nur den Schlüssel als Parameter. Der Wert wird im Hash über den Schlüssel ***name*** ermittelt. (Elementzugriff). ***@eintraege[name]***

```
class Telefonbuch

  ...

  def
    telefon_nummer_eintragen(name,numme
    r)
     # Elementzuweisung gleiche Syntax
    wie bei Array
     # Aber name ist kein Index
     @eintraege[name] = nummer
  end

  def telefon_nummer_suchen(name)
     # Elementzugriff gleiche Syntax wie
    bei Array
     # Aber name ist ein Schluessel
     return @eintraege[name]
  end

end
```

Syntaktischer Zucker

- Die Methoden für Elementzuweisung und der Elementzugriff lassen sich auch so schreiben, dass wir die Notation der eckigen Klammer für das Telefonbuch beim Aufruf der Methoden anstelle langer Methodennamen verwenden.

```
# Elementzuweisung
def []=(name,nummer)
  @eintraege[name] = nummer
end

# Elementzugriff
def [](name)
  return @eintraege[name]
end

tb = Telefonbuch.new("040",
   "Hamburg")
tb["hugo"]= "04078787887"
puts tb["hugo"]
```

Das Telefonbuch

- Wenn Personen aus dem Vorwahlkreis wegzieht, muss es möglich sein, Telefonbucheinträge zu löschen.
- Wir löschen Einträge im ***Hash*** durch Angabe des Schlüssels (***name***). Dann wird das Schlüssel-Wert Paar, das zu diesem Schlüssel gehört aus dem Hash gelöscht.
- Löschen gibt als Ergebnis den zum Schlüssel gehörenden Wert zurück, ***nil***, wenn der Schlüssel nicht enthalten war.

```
class Telefonbuch
  ...
  def loesche_eintrag(name)
    return @eintraege.delete(name)
  end
  ...
end
```

Iteratoren für Hashes

- Die Anforderung alle Namen bzw. alle Telefonnummern eines Telefonbuchs zurückzuliefern, lösen wir mit speziellen Iteratoren für Hashes.

- Hashes sind iterierbar (***each***). Darüber hinaus haben Hashes spezielle Iteratoren für Schlüssel, Werte und Schlüssel-Wert Paare: ***each_key, each_value, each_pair***

- Iteratoren für Hashes erzeugen zunächst Arrays mit einer definierten Reihenfolge auf Schlüsseln, Werten oder Schlüssel-Wert-Paaren, über die sie dann iterieren.

```
class Telefonbuch
 ...
 def alle_namen()
    key_ary = []
    @eintraege.each_key{ |key|
   key_ary << key}
    return key_ary
 end

 def alle_nummern()
    val_ary = []
    @eintraege.each_value{ |val|
   val_ary << val}
    return val_ary
 end ...
end
```

Einsammeln von Schlüsseln

- Wir iterieren mit ***each_key*** über die Schlüssel und sammeln diese in einem Array. Die Methode ***keys()*** in Hash liefert exakt das Ergebnis der Methode ***alle_namen***

- Analog ist die Methode ***alle_nummern***, nur dass wir in diesem Fall mit ***each_value*** iterieren. Die Methode ***values()*** in Hash liefert exakt das Ergebnis der Methode ***alle_nummern***

```
class Telefonbuch
 ...
 def alle_namen()
    key_ary = []
    @eintraege.each_key{ |key| key_ary << key}
    return key_ary
  end
end
```

Einsammeln von Paaren mit *each* oder *each_pair*

- Bei den Iteratoren für Schlüssel-Wert Paare, wird ein zwei-elementiges Array an die Blockvariable gebunden. Das Array enthält auf der ersten Position den Schlüssel und auf der 2‘ten Position den Wert.
- Statt einer Blockvariable, kann man auch zwei Variablen verwenden, dann wird der Schlüssel an die erste und der Wert an die zweite Variablen gebunden.
- Das Ergebnis ist ein Array mit zwei-elementigen Arrays
- Die Methode ***to_a()*** von Hash liefert exakt das Ergebnis wie die Methoden für ***alle_eintraege***.

```
class Telefonbuch
  ...

  def alle_eintraege()
    e_ary = []
    @eintraege.each_pair {|pair| e_ary
    << pair }
    return e_ary
  end

  def alle_eintraege()
    e_ary = []
    @eintraege.each {|key,val|
                e_ary << [key,val] }
   return e_ary
  end
  ...
end
```

Weitere Methoden von Hash

- Mit ***each*** verfügt Hash gleichzeitig über die Iteratoren ***collect, inject, select*** … wie wir diese auch bereits von Array kennen.
- Den technischen Grund lernen wir mit der Einführung von Vererbung kennen.

- Neben den bereits vorgestellten Methoden hat ***Hash*** noch Abfrage-methoden wie
  - ***empty?***
  - ***has_key?(key)***
- **Ü-6-a-20**: Schreiben Sie bitte ein Szenario, in dem Sie das Telefonbuch mit drei Einträgen füllen, dann alle Schlüssel, alle Werte und alle Paare ausgeben lassen, ein Paar löschen und dann nochmals die Inhalt ausgeben!

Ein interaktives Kundendienst System

- Unsere ***Beantworter*** Klasse wird auf einen Hash umgestellt, der als Schlüssel die typischen Worte und als Werte die dazu passenden Antworten enthält.

- Wenn die Frage des verzweifelten Benutzers eines der Schlüssel-Worte enthält, dann wird die unter dem Schlüssel abgelegte Antwort zurückgegeben.

- Es gibt zwei Lösungen, von denen die erste etwas umständlicher aber gut geeignet ist, um weitere Sammlungsobjekte einzuführen.

- Die zweite nutzt die String Methode ***include?()***, um Schlüssel-Wörter in einer Frage zu erkennen.

Die erste Lösung

- Die Idee:
  - Der ***Eingabeleser*** liefert eine Menge von Wörtern zurück, die in der Eingabe enthalten sind.
  - Der ***Beantworter*** iteriert über diese Menge und prüft für jedes Wort, ob es als Schlüssel im Hash vorhanden ist, und gibt die passende Antwort zurück.
  - Bei negativer Prüfung wird ein Standardtext zurückgegeben

- Rechts sehen wir die modifizierte ***starten*** Methoden für das Kunden-dienst System

```
def starten()
   willkommenstext_ausgeben()
   fertig = false
   while (!fertig)
     woerter =
        @leser.naechste_eingabe();
     if (woerter.length == 1 &&
           woerter.include?('end'))
       fertig = true
     else
       antwort =
        @antworter.
         generiere_antwort(woerter)
       puts(antwort)
     end
   end
   abschiedstext_ausgeben();
end
```

Die erste Lösung

- Der ***Eingabeleser*** liefert eine Menge von Wörtern.
- Dazu zerlegt er eine Zeile mit der Methode ***split(" ")*** in Einzelwörter. Split entfernt alle Leerzeichen und gibt die Wörter zwischen den Leerzeichen zurück
- Doppelte Worte können ignoriert werden, daher sammeln wir die Einzelwörter in einer Menge (***Set***), die sicherstellt, dass keine Dubletten entstehen.

```
class Eingabeleser
  def naechste_eingabe()
    print(">> ")
    zeile = gets().chomp!()
    return Set.new(zeile.split(" "))
  end

end
```

Die erste Lösung

- Der ***Eingabeleser*** liefert eine Menge von Wörtern, daher muss die Prüfung im Kundendienstsystem sicher stellen, dass das Wort '**end**' in der Menge der Wörter ist
- Die Menge muss dann dem ***Beantworter*** übergeben werden.

```
def starten()
   willkommenstext_ausgeben()
   fertig = false
   while (!fertig)
     woerter = @leser.naechste_eingabe();
     if (woerter.length == 1 && woerter.include?('end'))
       fertig = true
     else
       antwort = @antworter.generiere_antwort(woerter)
       puts(antwort)
     end
   end
   abschiedstext_ausgeben();
 end
```

Die erste Lösung

- Die aufwändigsten Änderungen müssen wir bei der Klasse ***Beantworter*** vornehmen.

  1. Wir erzeugen einen ***Hash*** für die typischen Worte und die Antworten im ***initialize()***
  2. In ***generiere_antwort(wort_menge)*** iterieren wir über die ***wort_menge*** und prüfen ob eines dieser Worte Schlüssel im Hash ist.

```
def initialize()
    @antworten_standard = standard_antworten_liste_erzeugen()
    @antwort_hash = antwort_hash_erzeugen()
 end
```

Die erste Lösung

- Der ***Hash*** für die typischen Schlüssel Worte und Antworten (Auszug).
- Beachte: Auch wenn Schlüssel eindeutig sind, Werte können mehrfach auftreten

```
def antwort_hash_erzeugen()
  { "absturz" =>              "Tja, auf unserem System kommt es nie zu einem Absturz. Das muss \n" +
              "an Ihrem System liegen. Welche Konfiguration haben Sie?",
   "abstürzt" =>              "Tja, auf unserem System kommt es nie zu einem Absturz. Das muss \n" +
              "an Ihrem System liegen. Welche Konfiguration haben Sie?",
    "stürzt" =>               "Tja, auf unserem System kommt es nie zu einem Absturz. Das muss \n" +
              "an Ihrem System liegen. Welche Konfiguration haben Sie?„,
    "performance" =>          "Bei all unseren Tests war die Performance angemessen. Haben Sie\n" +
              "andere Prozesse, die im Hintergrund laufen?",
    "fehler" =>               "Wissen Sie, jede Software hat Fehler. Aber unsere Entwickler arbeiten\n" +
              "sehr hart daran, diese Fehler zu beheben. Können Sie das Problem ein\n" +
   "speicher" =>        "Wenn sie die Systemanforderungen gründlich lesen, werden Sie feststellen,\n" +
              "dass die Speicheranforderung 1,5 Gigabyte beträgt. Sie sollten Ihren\n" +
              "Hauptspeicher unbedingt aufrüsten. Können wir sonst noch etwas für Sie tun?",
 } end
```

Die erste Lösung

- In ***generiere_antwort(wort_menge)*** iterieren wir über die ***wort_menge*** mit ***each***
- Wir prüfen mit der Methode ***has_key?(wort),*** ob eines dieser Worte Schlüssel im Hash ist.
- Sobald die Bedingung erfüllt ist, wird die Antwort über den Schlüssel gelesen und mit ***return*** zurückgegeben. Dies beendet gleichzeitig die Methode ***generiere_antwort***
- Wenn das Programm hinter dem ***each*** steht, war keines der Eingabeworte Schlüssel im Hash und es wir eine Standardantwort generiert.

```
def generiere_antwort(wort_menge)

   wort_menge.each {|wort|
     if @antwort_hash.has_key?(wort)
       return  @antwort_hash[wort]
     end
   }
   # kein Schlüssel gefunden
   return standard_antwort_erzeugen()
end
```

Die erste Lösung

- ***standard_antwort_erzeugen()*** arbeitet wie die Methode ***generiere_antwort()*** in der Version 2 des Kundendienst Systems.

```
def standard_antwort_erzeugen()
    zufalls_index = rand(@antworten_standard.length)
    return  @antworten_standard[zufalls_index]
end
```

Übung

- Für die zweite Lösung müssen wir folgende Erweiterungen vornehmen:
  1. Die Methode ***antwort_generieren*** bekommt einen Parameter für die Zeichenkette Frage, die im Kundendienst-System an den ***Beantworter*** übergeben wird.
  2. ***Beantworter*** Für alle im ***@antwort_hash*** enthaltenen Schlüssel müssen wir überprüfen, ob der Schlüssel in der Frage enthalten ist, wenn ja, dann geben wir die hinterlegte Antwort zurück, wenn nein, dann geben wir einen Standardtext zurück.
  3. Dazu nutzen wir die Methode ***include?(key_of_antwort_hash)*** der Klasse ***String.***
  4. Die Klasse ***Eingabeleser*** bleibt unverändert**.**

- **Ü6-a-21: Implementieren Sie bitte Lösung 2! Verwenden Sie den Hash von Lösung 1 für die Speicherung der Antworten.**

Übungen

- **Ü6-a-22:** Schreiben Sie bitte eine nicht-destruktive Methode, die die Schlüssel und Werte eines Hashes invertiert und den invertierten Hash zurückgibt! Hat der Ergebnishash immer die gleiche Größe wie das Original?

  Bsp.: {1=>35, 4=> 70, 2=> "String muss sein"} ergibt

  {35 => 1, 70 => 4, "String muss sein" => 2}

  Die Reihenfolge der Elemente des Ergebnis-hashes in Ihrer Lösung kann natürlich abweichen.

- **Ü6-a-23:** Iterieren Sie bitte mit ***each*** über einen Hash und sammeln Sie die Schlüssel-Wert Paare in einem Array von 2-elementigen Arrays! Das erste Element in dem 2-elementigen Array ist der ***key*** des Paares, das zweite Element ist der ***value*** des Paares.

  Lösen Sie bitte die gleiche Aufgabe mit ***inject*** und ***collect***!

- **Ü6-a-24:** Schreiben Sie bitte eine nicht-destruktive Methode, die die Schlüssel und Werte eines Hashes invertiert und den invertierten Hash zurückgibt! Wenn in dem Original mehrere Schlüssel auf den gleichen Wert abgebildet werden, dann sollen diese im Ergebnishash in einem Array gesammelt werden. Iterieren Sie über das Ergebnis und geben Sie dessen Schlüssel-Wert Paare aus.

  Bsp.: {1=>35, 2=> 70, 3 => 35} ergibt

  { 35 => [1,3], 70 => [2]}

- **Ü6-a-25:** Welche der Aufgaben zu den Zufallszahlen lässt sich besser mit einem Set als einem Hash lösen?

Mengen: ungeordnete Sammlungen ohne Dubletten

- Die Eigenschaft einer Menge sind uns aus der Mathematik bekannt:
  - Mengen können beliebig viele Objekte enthalten.
  - Sie enthalten ein Objekt immer nur einmal. Es sind keine Dubletten möglich.
  - Mengen sind ungeordnet.
  - Mengen sind gleich, wenn Sie die gleiche Anzahl von Objekten enthalten und es für jedes Objekt in der ersten Menge ein Objekt in der zweiten Menge gibt, das zu diesem Objekt gleich ist.
  - Eine Menge B ist eine **Teilmenge** von A, wenn jedes Objekt in B in A enthalten ist.

- Ebenfalls bekannt sind uns typische Operationen auf Mengen:
  - Durchschnitt
  - Vereinigung
  - Differenz

- In Ruby ist die Klasse für eine Menge ***Set***.

Übungen

- **Ü6-a-26:** Studieren Sie bitte die Schnittstelle der Klasse Set!
  - Welche aus der Mathematik bekannten Operationen implementiert ***Set***? Es gibt diese Operation als Kurzform und mit einem sprechende Methodennamen. Stellen Sie bitte eine Tabelle zusammen!
  - Wie werden Sets in Arrays und Arrays in Sets umgewandelt?
  - Warum ist für Sets ein indizierter Elementzugriff nicht möglich?
  - Wie können Sie dann auf Objekte in Sets zugreifen? Mit welcher Methode löschen Sie Elemente aus Sets?
  - Stellen Sie bitte die Iteratoren für Sets zusammen. Welche aus ***Array*** bekannten Iteratoren machen für Sets keinen Sinn?
  - Welche sonstigen für Arrays bekannten Methoden machen für Sets keinen Sinn?
  - Aus ***Array*** kennen wir die Methoden ***+*** und ***concat***. Welcher Methode entsprechen diese in etwa in ***Set***?

Übungen

- **Ü6-a-27:** Schreiben Sie bitte eine Klasse ***MySet!*** Diese Klasse soll sich wie eine Menge verhalten, verwendet aber für die Verwaltung der Elemente ein ***Array***.
  - Schreiben Sie bitte eine Methode ***add(an_obj)***, mit der Sie Objekte in ***MySet*** einfügen.!
  - Schreiben Sie bitte eine Methode ***delete(an_obj)***, mit der Sie Objekte aus ***MySet*** löschen! Wurde das Objekt in ***MySet*** gefunden, dann soll dieses Objekt zurück-gegeben werden, sonst ***nil***.
  - Schreiben Sie bitte die Methode ***size()***!
- **Ü6-a-28:** Implementieren Sie bitte die Methoden ***intersection, union, difference*** und ***subset?*** als Methode der Klasse ***MySet***!

Ü6-a-29: Schreiben Sie bitte die Methode ***merge(eine_sammlung),*** die ***MySet*** mit beliebigen Sammlungen mischt!

Ü6-a-30: Wenn Sie gleiche Kreise in eine anfangs leere Menge legen, wie viele Elemente enthält die Menge danach?

Ü6-a-31: Erzeugen Sie bitte ein Objekt der Klasse ***Set***! Erzeugen Sie zwei Kreisobjekte, die gleichen Mittelpunkt, Radius und gleiche Farbe haben. Legen Sie die Kreisobjekte in den Set. Wie viele Elemente enthält das Set? Inspizieren Sie das Ergebnis im Debugger.

- **Anmerkung:** Das Ergebnis scheint der Definition einer Menge zu widersprechen. Die Erklärung für das Ergebnis erhalten Sie im Abschnitt über Objektgleichheit.

Zusammenfassung

- **Anonyme Objekte** sind Objekte, die nicht über Variablen referenzierbar sind. Sie werden erzeugt und danach nicht wieder im Kontext der Erzeugung benutzt.

- Die **Schnittstelle** einer Klasse ist bestimmt durch den Klassennamen und die Methoden der Klasse mit ihren formalen Parametern. Sie beschreibt, **was** eine Klasse nach **außen** anbietet.

- Die **Implementierung** einer Klasse ist der Quelltext einer Klasse. Sie zeigt, **wie** eine Klasse und ihre Methoden **intern** arbeiten.

Vier Typen von Objektsammlungen

- **Arrays** sind Objektsammlungen beliebiger Größe, die über den Index eine Ordnung für die Objekte definieren.
- **Ranges** sind Objektsammlungen gleich-artiger Objekte mit einem endlichem Wertebereich.
- **Hashes** sind Objektsammlung beliebiger Größe mit Objektpaaren bestehend aus Schlüssel und Werten. Hashes definieren eine Abbildung von Schlüsseln auf Werte. Die Paare in einem Hash sind nicht geordnet; indizierter Zugriff ist nicht möglich.
- **Sets** sind Objektsammlungen beliebiger Größe, die keine zwei gleichen Objekte enthalten können. Die Elemente in einem Set sind nicht geordnet; indizierter Zugriff ist nicht möglich.

Zusammenfassung

Vier Typen von Objektsammlungen

- **Arrays** sind Objektsammlungen beliebiger Größe, die über den Index eine Ordnung für die Objekte definieren.
- **Ranges** sind Objektsammlungen gleichartiger Objekte mit einem endlichem Wertebereich.
- **Hashes** sind Objektsammlung beliebiger Größe mit Objektpaaren bestehend aus Schlüssel und Werte. Hashes definieren eine Abbildung von Schlüsseln auf Werte. Die Paare in einem Hash sind nicht geordnet; indizierter Zugriff ist nicht möglich.
- **Sets** sind Objektsammlungen beliebiger Größe, die keine zwei gleichen Objekte enthalten können. Die Elemente in einem Set sind nicht geordnet; indizierter Zugriff ist nicht möglich.
- **Mehrdimensionale Arrays**, sind Arrays, die wiederum Arrays enthalten. In jeder Dimension müssen die enthaltenen Arrays gleiche Länge haben. Der Elementzugriff erfolgt über einen kombinierten Index der Dimensionen.