Klassen und Objekte

Klassen

Eine **Klasse**
definiert die Eigenschaften (Instanzvariablen oder Felder) und das Verhalten (Methoden) von (gleichartigen) Objekten.

In Perl sind Klassen als Pakete oder Module umgesetzt. Diese Module definieren Subroutinen, um Objekte einer Klasse zu erzeugen (Konstruktoren) und um auf Objekten der Klasse zu arbeiten (Methoden).

Objekte

Ein **Objekt**
ist eine Instanz (Ausprägung) einer Klasse.

In Perl sind Objekte Datenstrukturen (häufig Hashes aber auch Arrays, ...) auf die eine Referenz verweist. Auǧerdem „wissen“ Objekte zu welcher Klasse sie gehören. Man arbeitet in Perl nur auf der Referenz auf das Objekt und greift mit dem Pfeil-Operator auf Eigenschaften und Methoden zu.

In **Instanzvariablen**
sind Eigenschaften von Objekten gespeichert.

In Perl gibt es keine Instanzvariablen im eigentlichen (üblichen) Sinn. Objekteigenschaften werden stattdessen als Werte innerhalb des Objekt-Hashes oder -Arrays dargestellt.

Methoden und Konstrukoren

Eine **Methode**
ist eine Funktion oder Prozedur, die auf Objekten arbeitet, um Eigenschaften abzufragen oder zu setzen, Objekte zu manipulieren oder Berechnungen auf diesen auszuführen.

In Perl sind Methoden Subroutinen innerhalb einer Klassendefinition. Methoden erwarten als erstes Argument ein Objekt der Klasse in der sie definiert wurden oder die Klasse selbst (oder eine Subklasse).

Ein **Konstruktor**
ist eine spezielle Methode, die ein Objekt (eine Instanz) einer Klasse erzeugt und zurückgibt.

In Perl sind Konstruktoren genau so implementiert. Es gibt keine spezielle Syntax oder spezielle Keywords für Konstruktoren.

Methoden

Erstes Argument jeder Methode ist Referenz auf aufrufendes Objekt oder aufrufende Klasse.
$\Rightarrow$ keine Unterscheidung (syntaktisch) zwischen Klassen- und Instanzmethoden.

```
sub getName{
    my $caller = shift;
    ref( $caller ) or die "Not a class-method\n";
    my $name = $caller->{"name"};
    return $name;
}

$obj->getName();

$numInstances=0;
sub getNumInstances{
    return $numInstances;
}

Klasse->getNumInstances();
```

Objekteigenschaften

Getter- und Setter-Methoden:

```
sub getName{
    $caller = shift;
    ref( $caller ) or die "No object\n";
    return $caller->{"name"};
}

sub setName{
    $caller = shift;
    ref( $caller ) or die "No object\n";
    $caller->{"name"} = shift;
}
```

Objekteigenschaften (2)

Gemeinsame Methode für Lesen und Setzen:

```
sub name{
    $caller = shift;
    ref( $caller ) or die "No object\n";
    if (@_){ $caller->{"name"} = shift; }
    return $caller->{"name"};
}
```

Konstruktoren

Ein **Konstruktor**
ist in Perl eine Methode, die eine Objektreferenz zurückgibt.

```
sub create{
    my $self = {} # Referenz auf anonymes Hash
    bless( $self ,"Color"); # Hash wird zu Objekt
                            # der Klasse Color
    return $self ;
}
```

Die Funktion **bless** erwartet als erstes Argument eine Referenz und als zweites optionales Argument die Klasse zu der das Objekt gehören soll. Wird das zweite Argument weggelassen, wird das Objekt in das aktuelle Paket "gesegnet".

Konstruktoren (2)

Standard-Konstruktor mit Argumenten:

```
sub define{
    my $caller = shift;
    my $class = ref( $caller ) || $caller ;
    my $self = { @_ };
    bless( $self , $class );
    return $self ;
}

$red = Color->define(name => "red", rgb => "255,0,0");
```

Vererbung

- Vererbung über spezielles Array @ISA
- Mehrfachvererbung möglich (darum ein Array)

package Mule;
our @ISA = ("Horse", "Donkey");

Suche nach einer Methode (dfs):

1. Suche in aktueller Klasse (Mule)
2. Suche in erster Oberklasse (Horse)
3. Suche der Reihe nach in den Oberklassen von Horse (wenn es sie gibt)
4. Suche in zweiter Oberklasse (Donkey)
5. ...

Methoden der Oberklasse

- durch @ISA werden Oberklassen definiert
- um Methoden der Oberklassen in der aktuellen Klasse zu nutzen: **require**

```
package Mule;
our @ISA = ("Horse", "Donkey");
require Horse;
require Donkey;
```

Gleiche Funktion, aber kürzer (und einfacher):

```
package Mule;
use base ("Horse", "Donkey");
```

Methoden der Oberklasse (2)

Zugriff innerhalb einer Methode (um Code wiederzuverwenden):

```
package Mule;
use base ("Horse", "Donkey");

sub speak{
    my $caller = shift;
     $caller->Horse::speak();
     ...
}
```

Häufig unbekannt, in welcher Oberklasse Methode implementiert ist:

```
package Mule;
use base ("Horse", "Donkey");

sub speak{
    my $caller = shift;
     $caller->SUPER::speak();
     ...
}
```

Interfaces

. . . gibt es in Perl nicht.

Interfaces

. . . gibt es in Perl nicht.
. . . aber können durch Mehrfachvererbung „emuliert“ werden:

- Gemeinsame Oberklasse mehrerer Klasse
- Oberklasse definiert Methoden
  - können („müssen“ hier sogar) überladen werden
  - sind nicht (inhaltlich) implementiert
  - Klassenvertrag durch Dokumentation

Bioperl

. . . endlich . . .

Bioperl

Sammlung von Modulen und Klassen zum

- Zugriff auf Sequenzen
  - aus lokalen Dateien
  - lokalen und entfernten Datenbanken
- Format-Umwandlungen
- Manipulation von Sequenzen
- Suche nach Sequenzähnlichkeiten (BLAST,...)
- Erstellen von Alignments
- ...

Sequenz-Klassen

- Bio::Seq: Standard-Sequenz
- Bio::PrimarySeq: „abgespeckte“ Variante von Bio::Seq
- Bio::Seq::RichSeq: speziell für „ausladende“ Annotation
- Bio::Seq::LargeSeq: speziell für lange ($>$ 100 MB) Sequenzen
- Bio::LiveSeq::: Paket für häufig geänderte Sequenzen

Abstraktion durch Interfaces Bio::PrimarySeqI und Bio::SeqI
$\Rightarrow$ genutzte Implementierung oft nicht relevant

Sequenz-I/O

Klasse SeqIO zum Lesen und Schreiben von

- FastA
- EMBL
- GenBank
- Swissprot
- raw
- ...

Eingelesene Sequenzen je nach Format als Seq, PrimarySeq, RichSeq repräsentiert.

Sequenzdatenbanken

Paket Bio::DB:: enthält Klassen, um Verbindungen zu (Sequenz-) Datenbanken aufzubauen und Sequenzen nach

- Accession number
- Name
- Version
- ID
- ...

abzurufen und weiter zu verwenden.

Bio::Perl

Bio :: Perl enthält Subroutinen, um einfache Operationen auf Sequenzen (auch ohne tieferes Wissen über Objekte) zu ermöglichen.
Subroutinen z.B.

- get_sequence
- new_sequence
- revcom
- translate
- write_sequence

Dokumentation

www.bioperl.org

- BioPerl Tutorial
- weitere (speziellere) Tutorials
- HowTos
- Deobfuscator (ähnlich Javadoc)
  - Liste aller verfügbaren Klassen
  - Liste aller Methoden (inkl. ererbten) pro Klasse
  - Dokumentation/Kommentare zu Klassen, Methoden