

Angewandte Programmierung in der Computerlinguistik - Objekte, Objektorientierte Module

Andreas Neumann M.A.

Themen

- * Objektorientierung, Begriffsklärung
- * Objektorientierung in Perl
- * Objektorientierte Module verwenden

Objektorientierung - Grundlagen und Realisierung in Perl

Warum Objektorientierung

- * 5 +- 2 : Gedanken, Einheiten die man gleichzeitig im Kopf behalten kann
- * Gehirn entlasten
- * Ähnlichkeiten zur realen Welt, weniger abstrakt
- * Wiederverwendbarkeit von Code erhöhen

Objektorientierung - Begriffe

- * Vererbung
- * Polymorphie
- * Kapselung

Objekte - Klassen - Instanzen

- * Klasse: Bauplan für ein Objekt
- * Instanz: ein „gebautes“ Objekt

Klasseninstanz

- * Um eine Klasse zu benutzen, muss man aus ihr eine Instanz erzeugen
- * Bevor eine Klasse nicht erzeugt wurde kann man nicht auf deren Methoden und Variablen zugreifen (Ausnahme statische Methoden)
- * Das erfolgt normalerweise mit dem **new** Operator

Methoden und Attribute

- * Funktionen einer Klasse werden **Methoden** genannt
- * Variablen einer Klasse werden **Attribute** , **members** oder **Member-Variablen** genannt.

Perl und Objektorientierung

- * teilweise objektorientiert
- * Objektorientierung über Namespaces und Module und Perl-eigene Datenstrukturen
- * Kapselung kann nach belieben durchbrochen werden
- * Polymorphismus durch Vererbung spielt keine Rolle, DuckTyping-Ansatz
- * Vererbung und sogar Mehrfachvererbung möglich

Ein Objekt benutzen



Stein
gewicht
grafitto
da_liegen()
add_grafitto()

Ein Perl-Objekt benutzen

- * mit **use** einbinden
- * -> **new()** zum erzeugen
- * Pfeilnotation um Methoden und Member zu benutzen

Objekt Stein benutzen

```
#!/usr/bin/perl -w
use Stein; #ist ein Modul

# Konstruktor
my $brocken = Stein->new(50);

# Methodenaufruf
$brocken -> da_liegen();

#Membervariable
print $brocken -> {gewicht}."\n";

#Eine Methode mit Argument verwenden
$brocken -> add_graffito("O tempore, oh mores");

# Gibt Objektnamen und Hashreferenz aus
print "$brocken\n";
```

Running “use_stein.pl”...

```
...
Der Stein liegt da mit 50kg.
50
Stein=HASH(0x7fbe68803ed0)
```

Ein Objekt schreiben



Umsetzung von Objektorientierung in Perl

- * *Klassen* werden durch **packages** dargestellt
- * *Objekte/Instanzen sind im normalfall hash references denen mit der Funktion **bless()** ein Klassenname zugeordnet wurde*
- * *Attribute sind **Schlüssel- /Wertpaare** im oben genannten Hash*

Beispiel: Stein

```
#!/usr/bin/perl -w
package Stein; # Package für Klasse

sub new {
    my($class, $gewicht) = @_;
    my $self = { gewicht => $gewicht };
    bless($self, $class);
    return $self;
}
# Methode ohne Argumente, hat selbst stets automatisch ein Argument: die Referenz auf
$self
sub da_liegen {
    $self = shift;
    print "... \n". "Der Stein liegt da mit ".$self -> {gewicht}."kg. \n"
}
# Methode mit einem Argument
sub add_graffito {
    ($self,$text) = @_;
    $self ->{graffito} = $text;
}
1 # ist ein Modul
```

Beispiel: Stein - Klasse

- * Klasse = package

```
package Stein;
```

Beispiel Stein - Konstruktor

```
sub new {
    my($class, $gewicht) = @_;
    my $self = { gewicht => $gewicht };
    # hier befinden sich die Attribute
    bless($self, $class);
    # bless macht aus Hash-Referenz eine Klasse
    return $self;
}
```

- * \$class entspricht Klassennamen
- * \$self ist ein anonymes Hash
- * bless(name, hash) macht \$self zu einer Klasse
- * \$self muss zurückgegeben werden -> Klasse instanziert

Beispiel Stein - einfache Methode

```
sub da_liegen {  
    $self = shift;  
    print "...\\n"."Der Stein liegt da mit ".$self -> {gewicht}."kg. \\n"  
}
```

- * Alle Klassenmethoden brauchen als erstes Argument eine Referenz auf sich selbst
- * Ansonsten wüsste die Klasseninstanz nichts über ihren Zustand

Beispiel Stein - Methode mit Argumenten

```
sub add_graffito {  
    ($self,$text) = @_;  
    $self ->{graffito} = $text; #Membervariable graffito setzen  
}
```

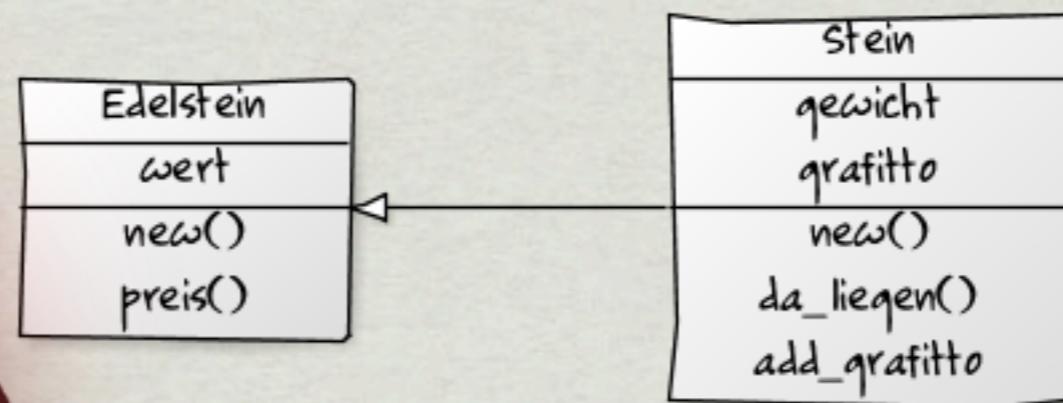
- * Selbstreferenz als erstes Argument
- * Zweites Argument ist erstes Argument der Funktion

Beispiel Stein - Zugriff auf eine Membervariable

```
$self ->{graffito} = $text; #Membervariable graffiti setzen
```

- ✳ Zugriff innerhalb der Objektmethode über selbst-Referenz

Vererbung



Erben

- * Erben mit **use base „Basisklasse“**
- * **SUPER** um auf Elternklasse zuzugreifen
- * Methoden überschreiben
- * Methoden und Attribute erben

Beispiel Erben: Edelstein

```
#!/usr/bin/perl -w
package Edelstein; # Package für Klasse
use base 'Stein'; # Hier wird geerbt

# Konstruktor von Stein überschrieben
sub new {
    my($class, $gewicht,$wert) = @_;
    my $self = { gewicht => $gewicht, wert => $wert };
    bless($self, $class);
    return $self;
}

# Neue Methode
sub preis {
    $self = shift;
    return $self -> {wert};
}

# Methode überschrieben, aber mit super Zugriff auf Elternklasse
sub da_liegen {
    $self = shift;
    $self->SUPER::da_liegen();
    print("und ist ".$self->{wert}." wert.\n");
}
1
```

Beispiel Erben: Edelstein benutzen

```
#!/usr/bin/perl -w
use Edelstein;

# Konstruktor
my $rubin = Edelstein->new(0.5,50000);
my $diamant = Edelstein->new(0.4,700000);

# Methodenaufruf, Implementierung in der Elternklasse
$rubin -> da_liegen();

# Methodenaufruf, Implementierung in Klasse
print "Preis:".($diamant -> preis)."\n";

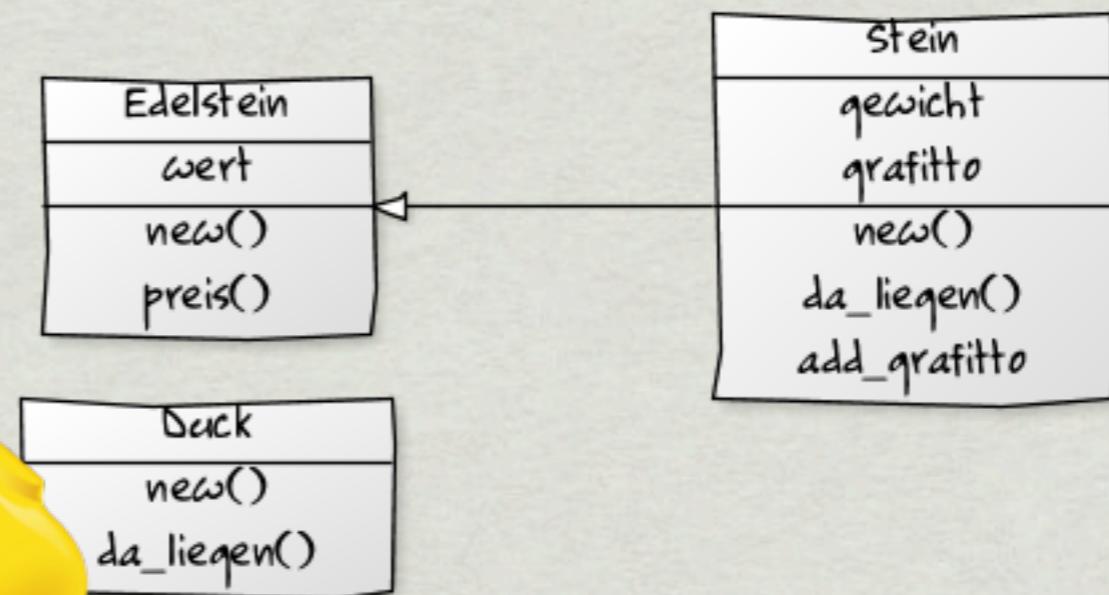
# Gibt Objektnamen und Hashreferenz aus
print "$rubin\n";
print "$diamant\n";

$diamant -> da_liegen();
```

Running “use_edelstein.pl”...

```
...
Der Stein liegt da mit 0.5kg.
und ist 50000 wert.
Preis:700000
Edelstein=HASH(0x7fa1a1003ed0)
Edelstein=HASH(0x7fa1a1028938)
...
Der Stein liegt da mit 0.4kg.
und ist 700000 wert.
```

Polymorphismus



Polymorphismus in Perl

- * Prüfung nur zur Laufzeit
- * „Duck-Typing“-Strategie

**„WHEN I SEE A BIRD THAT WALKS LIKE A DUCK AND SWIMS LIKE A DUCK
AND QUACKS LIKE A DUCK, I CALL THAT BIRD A DUCK.“**
– JAMES WHITCOMB RILEY

QUELLE: WIKIPEDIA

Polymorphismus: Beispiel Duck

```
#!/usr/bin/perl -w
package Duck;

sub new {
    my($class) = @_;
    # Klassenname als ersten Parameter
    my $self = { gewicht => 20 }; #Vorbelegt mit 20
    bless($self, $class);
    return $self;
}
# Methode ohne Argumente, hat selbst stets automatisch ein Arument: die
Referenz auf $self
sub da_liegen {
    $self = shift;
    print "...`n"."Quack, Quack.`nDie Ente wiegt ".$self -> {gewicht}."`n"
}
1 # ist ein Modul
```

Polymorphismus: Beispiel

```
#!/usr/bin/perl -w
use Stein;
use Edelstein;
use Duck;

# Konstruktor
my $brocken = Stein->new(50);
my $smaragd = Edelstein->new(0.3,40000);
my $donald = Duck->new();

@queue = ($brocken,$smaragd,$donald);

# Duck Typing, die Stein und Edelstein sind im gleichen Vererbungsbaum, Die Klasse Duck
# hat einen eignen Baum. Das Programm funktioniert aber problemlos da alle Klassen die
# Methode da_liegen() implementieren
foreach $entry (@queue) {
    $entry->da_liegen();
}

foreach $entry (@queue) {
    $entry->add_graffito("meins!");
    print "Gravur:". $entry ."->". $entry->{graffito} ."\n";
}
```

Polymorphismus: Beispiel - Output

- * `add_grafitto()` ist zwar für Stein und Edelstein definiert aber nicht für Duck
- * Laufzeitfehler - >Programmabbruch

The screenshot shows a terminal window with the following content:

```
Running "polymorphism.pl"...
...
Der Stein liegt da mit 50kg.
...
Der Stein liegt da mit 0.3kg.
und ist 40000 wert.
...
Quack, Quack.
Die Ente wiegt 20 kg.
Gravur:Stein=HASH(0x7fa6d0803ed0)->meins!
Gravur:Edelstein=HASH(0x7fa6d0828920)->meins!
```

Can't locate object method "add_graffito" via package "Duck" at /Users/andi/Dropbox/CIS/Angewandte Programmierung in der Computerlinguisitk/objects/polymorphism.pl line 20.

exception_handler::die in polymorphism.pl at line 20

Program exited with code #0 after 0.06 seconds. [copy output](#)

Schönere Objektorientierung mit Moose

- * Vereinfacht Objektorientierung
- * Bessere Möglichkeiten zur Kapselung (Accessors,Getters,Setters)
- * Typchecks
- * uvm.
- * <http://moose.iinteractive.com/>

Beispielklasse mit Moose

```
package Person;

use Moose;

has 'first_name' => (
    is  => 'rw',
    isa => 'Str',
);

has 'last_name' => (
    is  => 'rw',
    isa => 'Str',
);

no Moose;
__PACKAGE__->meta->make_immutable;
```

Quellen

- * Grafiken: Wikimedia Commons