Perl Mongers im Ruhrgebiet

Ruhr.pm

Einführung in die Objektorientierte Programmierung in Perl

Autor: Ingo Wichmann

EMail: ingo AT ruhr.pm.org

Datum: 18. Juni 2007

http://ruhr.pm.org/

Ruhr.pm

Hello World 2.0

Neue Klasse: **HelloWriter**

```
#!/usr/bin/perl

package HelloWriter;

  my $hello = "Hello World";

  sub new {
    my ($class) = @_;
    return bless {}, $class;
  }

  sub run {
    print $hello;
  }

package main;

  $hello = HelloWriter->new;
  $hello->run;
```

Feldvariable

Konstruktor definieren

Deklaration einer neuen Methode

Neue Instanz der Klasse **HelloWriter** erzeugen

Instanzvariable vom Typ **HelloWriter**

Aufruf der **run**-Methode der **HelloWriter** -Instanz

Ruhr.pm

Objekte

- Der Befehl `bless` erzeugt aus einer Referenz ein neues Objekt einer Klasse.
- Ein Objekt ist ein Behälter, der Daten vom Typ dieser Klasse enthält.

Ruhr.pm

Beispielklasse Person (1)

```
package Person;

use strict;
use warnings;

sub new {
  my ($class) = @_;
  bless { name => "" }, $class;
}

sub getName {
  my ($self) = @_;
  return $self->{name};
}

sub setName {
  my ($self, $newName) = @_;
  $self->{name} = $newName;
}

1;
```

Neue Klasse **Person** anlegen

Konstruktor definieren

Methoden definieren, um auf die Objektdaten zugreifen zu können

Ruhr.pm

Beispielklasse Person (2)

```
package main;

use Person;

use strict;
use warnings;

my $tom = Person->new;
my $jerry = Person->new;

$tom->setName("Tom");
$jerry->setName("Jerry");

my $a = $tom->getName;
print "Name=$a", $/;
```

Erzeugen eines neuen Objektes der Klasse **Person**

Schreibzugriff auf die Objektdaten durch Aufruf der Objektmethode **setName**

Lesezugriff auf die Objektdaten durch Aufruf der Objektmethode **getName**

Methoden und Rückgabewerte (1)

- Erinnerung: Die Klasse Person enthielt eine Methode:

```
sub getName {
  my ($self) = @_;
  return $self->{name};
}
```

Liefere den nachfolgenden Wert als Rückgabewert der Methode zurück und beendet die Methode.

- Benutzung der Methode: z.B.

```
my $a=$person->getName;
printf("Name=%s$/",$person->getName);
```

Die Methode kann genauso in Ausdrücken eingesetzt werden wie eine Variable.

Methoden und Rückgabewerte (2)

- Eine Methode, die einen Rückgabewert liefert, kann wie eine Variable als Platzhalter eingesetzt werden.
- Das return-Statement legt fest, welchen Wert die Methode zurückliefert und beendet ihre Ausführung. Eine Methode ohne return-Statement liefert den Rückgabewert des letzten Ausdrucks zurück.

Methoden und Parameter (1)

- Erinnerung: Die Klasse Person enthielt eine Methode:

```
sub setName {
  my ($self, $newName) = @_;
  $self->{name} = $newName;
}
```

Parameter der Methode **setName**

Methoden und Parameter (2)

- Parameter werden der Methode beim Aufruf übergeben wird. Z.B.:

```
$person->setName ("Martina Muster");
```

Ruhr.pm

Übung 1

- Schreiben Sie eine Klasse Person, die als Datenelement einen Vor- und einen Nachnamen hat. Die Datenelemente sollen get/set-Methoden angesprochen werden können.
- Schreiben Sie ein Programm testPerson.pl, das nacheinander drei Personen erzeugt und deren volle Namen auf den Bildschirm schreibt.

Konstruktoren

- Erinnerung: Das Programm PersonTest enthielt das Kommando:

```
my $person=Person->new;
```

Erzeugung eines neuen Objektes der Klasse **Person**.

Aufruf des Constructors.

Ruhr.pm

Klassenmethoden und -attribute (1)

```
package Person;

use strict;
use warnings;

{
  my $count=0;
  sub getcount { $count }
  sub incr_count { ++$count }
}
[...]

package main;

my $person = Person->new;
printf("Anz:%d$/", Person->getcount);
```

Die Deklaration im Klassen block macht Variable und Methode zu Eigenschaften der Klasse selbst; insbesondere nicht zu Eigenschaften eines Objekts!

Ruhr.pm

Klassenmethoden und -attribute (2)

- Wird eine Variable im Klassenblock deklariert, dann existiert sie für die gesamte Klasse nur einmal, hat also für alle Objekte dieser Klasse denselben Wert.
- Eine im Klassenblock deklarierte Variable bzw. Methode einer Klasse kann abgefragt bzw. aufgerufen werden, ohne daß Objekte der Klasse existieren.
- Im Klassenblock definierte Methoden einer Klasse können nur auf andere Variablen dieser Klasse zugreifen, wenn diese ebenfalls im Klassenblock definiert wurden ("use strict" vorrausgesetzt).

Ruhr.pm

Übung 2

- Ergänzen Sie die Klasse um eine Variable, die das Geschlecht bezeichnet, sowie get/set-Methoden für den Zugriff darauf. Führen Sie Konstanten für die beiden möglichen Geschlechter ein.
- Ergänzen Sie die Klasse um einen Konstruktor, der einen Parameter für das Geschlecht erwartet.
- Schreiben Sie ein Programm personTest3, das nacheinander drei Personen erzeugt, mit unterschiedlichen Namen und Geschlechtern versieht und einschließlich Anrede auf den Bildschirm schreibt. (Z.B. "Frau Martina Muster") Abschließend soll die Gesamtzahl der vorhandenen Personen auf dem Bildschirm ausgegeben werden.

Ruhr.pm

Vererbung (1)

```
package Automobile;
use strict;
use warnings;

sub new {
  my ($class, $seats) = @_;
  my $self = { seats => $seats };
  bless $self ,$class;
}

sub getSeats {
  my ($self) = @_;
  return $self->{seats};
}
sub getStandingRoom {
  return 0;
}
1;
```

Anzahl der Sitzplätze

Anzahl der Stehplätze

Ein Auto hat normalerweise keine Stehplätze.

Vererbung (2)

Die Klasse **Bus** wird von der Klasse **Automobile** abgeleitet.

```
package Bus;
use base "Automobile";

use strict;
use warnings;

sub new {
  my ($class, $seats, $standingRoom) = @_;
  my $self = $class->SUPER::new($seats);
  $self->{standingRoom} = $standingRoom;
  return $self;
}

sub getStandingRoom {
  my ($self) = @_;
  return $self->{standingRoom};
}

1;
```

Constructor der abgeleiteten Klasse.

Aufruf des Constructors von **Automobile**.

Zusätzliche Erweiterung der Daten von **Automobil**.

Überschriebene Methode.

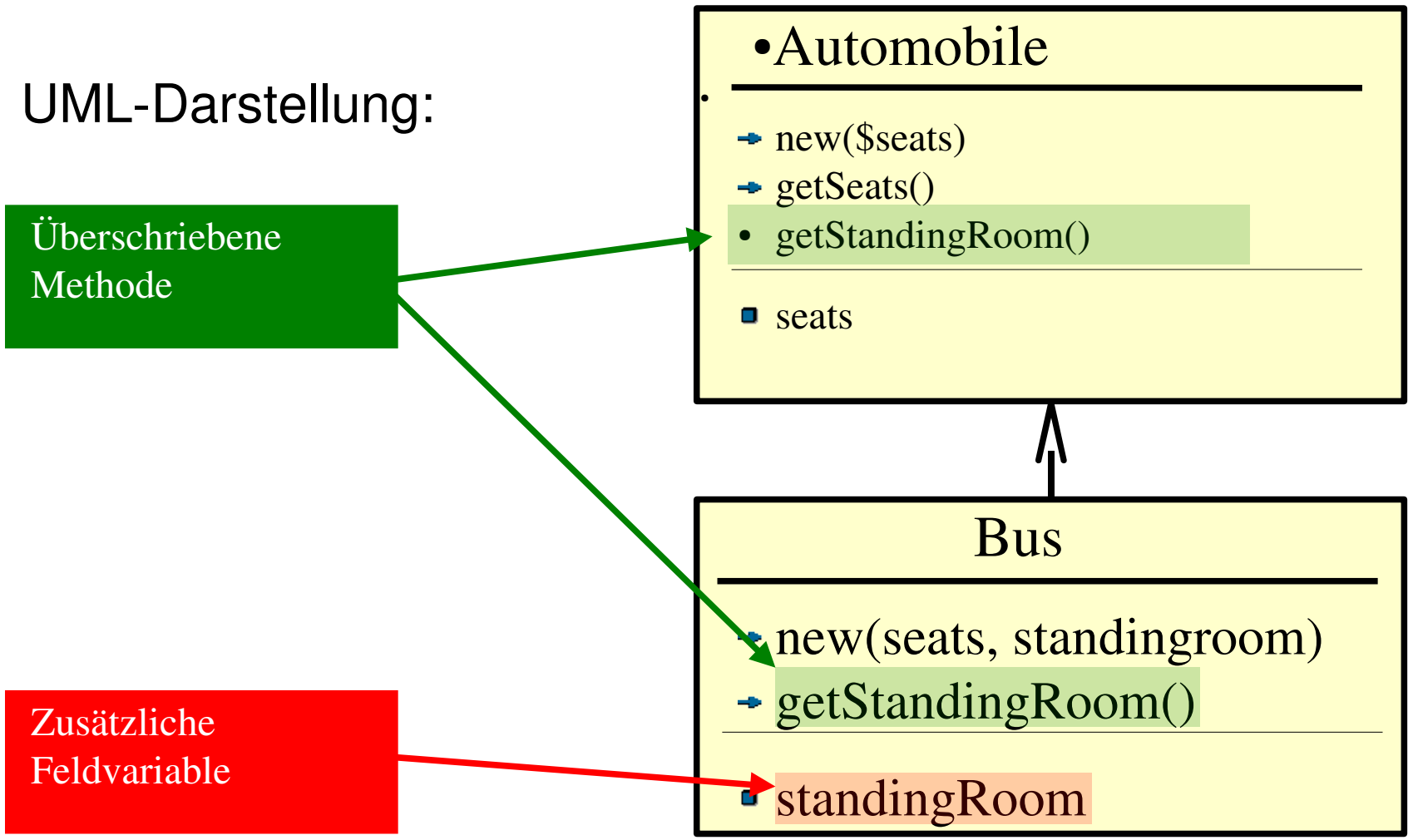
Ruhr.pm

Vererbung (3)

- Eine abgeleitete Klasse erbt alle Eigenschaften (Variablen, Methoden) ihrer Vorfahren.
- In der abgeleiteten Klasse definierte Variablen und Methoden erweitern oder überschreiben die Variablen und Methoden des Vorfahren.
- Im Constructor einer abgeleiteten Klasse wird der Constructor des Vorfahren aufgerufen. Dies geschieht über den Bezeichner SUPER.

Vererbung (4)

UML-Darstellung:



Übung 3

- Schreiben Sie die vorgestellte Klasse Automobile. Ergänzen Sie sie um eine Methode getTotalRoom, mit der sich die Summe aus Sitz- und Stehplätzen ermitteln läßt.
- Schreiben Sie die abgeleitete Klasse Bus.
- Schreiben Sie im Package test ein Programm testBus.pl, das ein Automobil mit 5 Sitzen sowie einen Bus mit 20 Sitzen und 30 Stehplätzen erzeugt. Lassen Sie von beiden Objekten die Gesamtzahl aller verfügbaren Plätze auf den Bildschirm schreiben.
- Ergänzen Sie die Klasse Automobile um eine Klassenmethode, die jeweils die Anzahl aller vorhandenen Automobile liefert (siehe Übung 9). Lassen Sie das Programm testBus.pl am Ende diese Zahl ausgeben.

Ruhr.pm

Übung 4

- Ergänzen Sie Ihre Klasse Automobile um eine Methode setSeats, um die Anzahl der Sitzplätze nachträglich zu ändern.
- Schreiben Sie ein Programm, das ein Objekt vom Typ Bus erzeugt und einer zweiten Variable zuweist, etwa so:
- `my $bus = Bus->new(20,30);`
- `my $auto = $bus;`
- Lassen Sie die Gesamtzahl der Plätze für beide Variablen auf den Bildschirm ausgeben. Ändern Sie anschließend mit Hilfe der Methode setSeats die Anzahl der Sitzplätze im bus und lassen Sie erneut die Zahl der Plätze für beide Variablen ausgeben.
- Überrascht Sie das Ergebnis?

Ruhr.pm

Achtung!

- Wenn zwei Variablen das gleiche Objekt als Wert haben, handelt es sich im Computerspeicher um identisch dieselben Daten, d.h. jede Änderung an der einen Variablen wirkt sich zwangsläufig auch auf die andere aus. Die Zuweisung

```
$auto2 = $auto1;
```

erzeugt also insbesondere keine Kopie des Objektes in **$auto1**. Zu diesem Zweck braucht man eine Methode.