Inhaltsverzeichnis

Perl Klassen und Objekte

Klassendefinition

Eine Klasse in Perl ist ein Package. Der Dateiname des Moduls sollte gleich sein.

Der name des Konstrucktors ist egal, es bietet sich aber new an.

MyClass.pm

```
package MyClass;

sub new {
   # erstes Argument ist der Klassenname
   my $Klasse = shift;

   my $Referenz = {}; # anonymer Hash

   # setze object eigenschaft
   $Referenz->{"myEigenschaft"} = "Wert meiner Eigenschaft";

   # Objekt-Referenz wird erzeugt
   bless( $Referenz, $Klasse );

   return($Referenz);
}
  ...
1;
```

```
my $myObj = new MyClass( ... );

# oder:
my $myObj = MyClass->new( ... );
```

Eigenschaften

Objekteigenschaften werden in der Objektreferenz gespeichert. Siehe auch oben im Konstruktor (`sub new`).

Statische Eigenschaften werden wie folgt umgesetzt:

```
package MyClass;

  my $statischeEigenschaft; # private

  our $statischeEigenschaft2;

  ...

1;
```

Zugriff auf Eigenschaften

```
sub methode
{
   my $self = shift;

   my $var1 = $myClass::statischeEigenschaft; # Zugriff auf statische
Eigenschaft

   my $var2 = $self->{"myEigenschaft"}; # Zugriff auf Objekt-
Eigenschaft
}
```

Methoden

Methoden Deklaration:

```
   # private Methode
   my $privateMethode = sub
   {
      ...
   };

   sub methode
   {
      ...

      # private Funktion aufrufen
      $obj->$privateMethode ( $param );

      # oder:
      $privateMethode->($obj, $param);
```

```
    # oder:
    MyClass->$privateMethode ( $param );
}
```

Je nach Aufruf verhalten sich die Methoden - genauer gesagt die Parameterübergabe - anders:

```
$obj->methode( $param );
```

- Hier werden der Function zwei Parameter übergeben:
  1. $obj
  2. $param

```
MyClass->methode( $param );
```

- Auch hier werden der Function zwei Parameter übergeben:
  1. „MyClass“
  2. $param

```
MyClass::methode( $param );
```

- Hier wird der Function nur ein Parameter übergeben:
  1. $param

Ableiten

```
package MyClass;

    # von MySuperClass ableiten:
    use parent [-norequire,] 'MySuperClass' [, ... ];

    # entspricht den früher benutzten Ausdruck:
    use MySuperClass; # anstatt use kann auch 'require' verwendet
werden
    push @MyClass::ISA, qw( MySuperClass [, ...] );

    ...
```

```
   1;
```

In überladender Methode die Elternmethode aufrufen

```
   sub methode
   {
      my $self = shift;

      $self->SUPER::methode();

      # oder:
      $self->MySuperClass::methode();
   }
```

Klassen Methoden aufrufen

Methoden können über ein Object in der Form

```
$obj->methode();
```

aufgerufen werden.

Klassenmethoden, die **NICHT** überladen sind können **NICHT** mit

```
MyClass::methode();
```

aufgerufen werden. Dazu muss in der abgeleiteten Klasse die Methode nochmals definiert werden, die dann die ursprünliche Methode aufruft. z.B.:

```
sub methode {
   return ParentClass::methode();
}
```