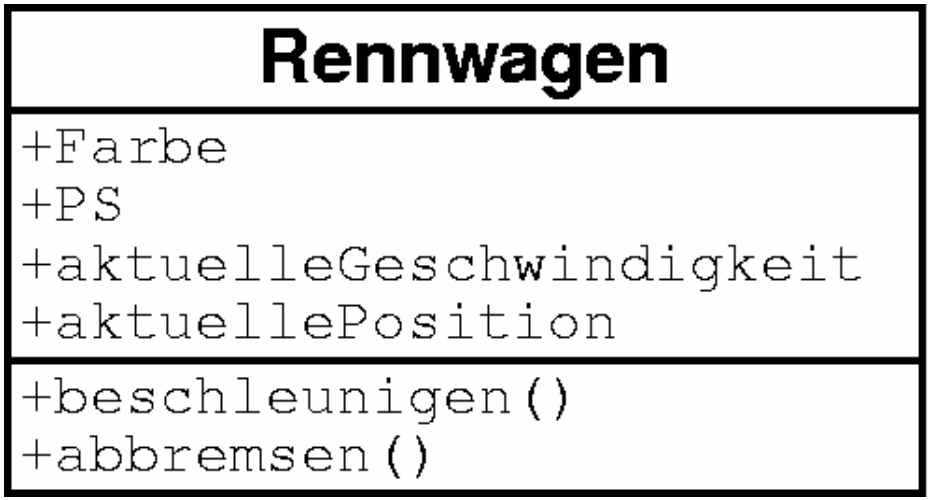
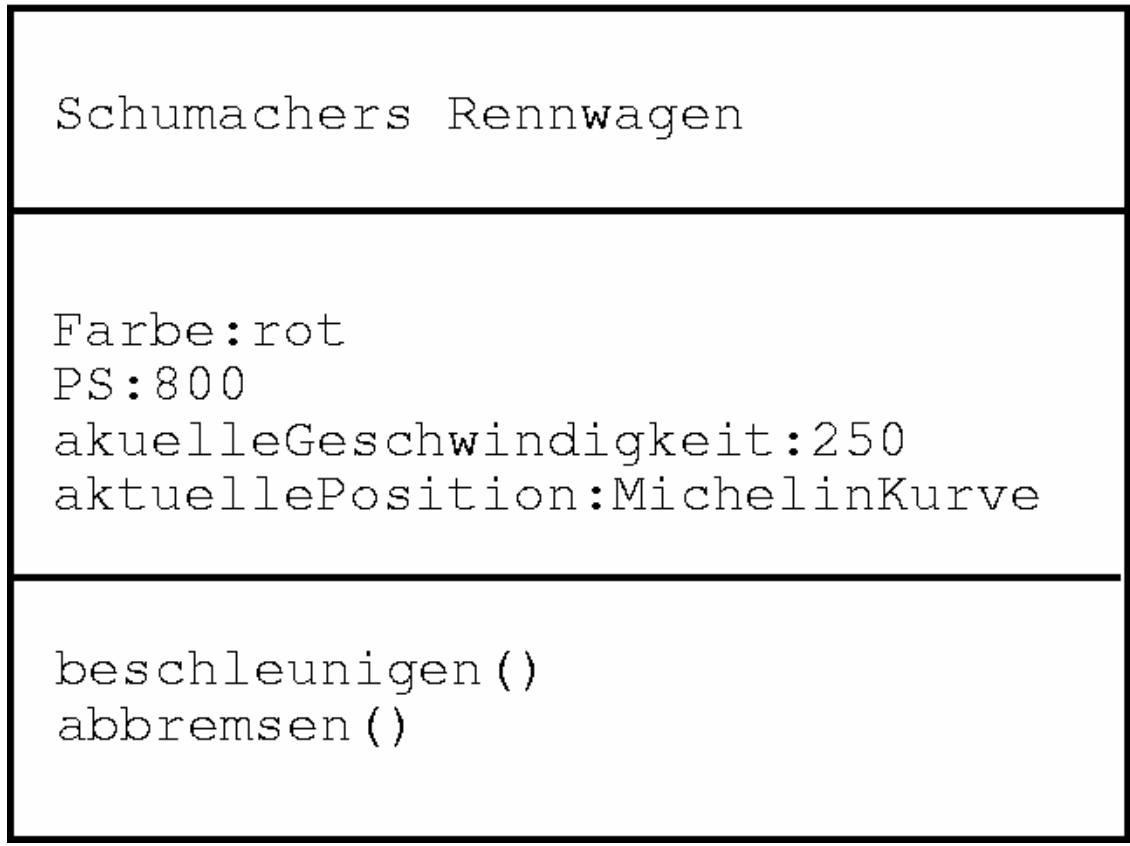
Software-Entwicklung

- Abbildung eines kleinen Ausschnitts der Realität in einem Programm
- Nachbildung von Dingen aus dem "wirklichen Leben"
- Abstraktion von Dingen aus dem "wirklichen Leben"
- Dinge haben Eigenschaften
- Mit Dingen sind Tätigkeiten verbunden.!

Objektorientierung





Klassenbeispiel

http://helga.mfh-iserlohn.de/internet/personen/metzger/phps/student.phps

Definitionen

Objekt
Ein Objekt ist ein konkretes Exemplar einer Klasse. Im Endeffekt wird nur mit den Objekten gearbeitet. Vom Prinzip her kann man Objekte mit Variablen und Klassen mit Datentypen vergleichen.

Klasse
In einer Klasse werden die Attribute und Methoden eines Objekts definiert

Methode
Methoden sind Funktionen, nur eben das sie in einer Klasse stehen

Attribut
Variablen innerhalb einer Klasse werden als Attribute bezeichnet

Konstruktor
Methode, die beim Instanzieren eines Objektes einmal automatisch aufgerufen wird. Diese wird in der Regel genutzt, um Initialisierungen vorzunehmen.

Destruktor
Das Gegenteil vom Konstruktor:
PHP kennt keine Destruktoren.

$this

Innerhalb einer Klasse wird mit $this auf die Eigenschaften und Methoden zugegriffen.
$this verweist auf die Klasse selbst
z.B:

```
class test
{
        var $groesse;
        function test()
        {
                $this->groesse=25;
        }

        function getGroesse()
        {
                $ergebnis=$this->groesse-20;
                return $ergebnis;
        }

}
$object=new test();
$foo=$object->getGroesse();
echo $foo;
?>
```

Kreditbeispiel

Eigenschaften:

Die Kredithöhe
Die monatliche Belastung, die mit dem Kredit verbunden ist.
Der Zinssatz zu dem der Kredit vergeben wird.
Die vom Benutzer gewünschte Tilgung.

Tätigkeiten

Die Kredithöhe kann berechnet werden (aus Eigenkapital und Immobilienpreis).
Die monatliche Belastung kann berechnet werden (aus Kredithöhe, Zins und Tilgung).

Raketenbeispiel

Eigenschaften sind Variablen oder Konstante:

- Kredithöhe
- Immobilienpreis
- Zinssätze
- Start- oder Landezeiten

Die Tätigkeiten sind der Programmablauf, die Sie in Funktionen auslagern können.

Tätigkeiten = Funktionen

Zusammenhänge mit unseren Programmierkonstrukten

Eigenschaften:

Die Startzeit.
Die Landezeit.
Die Flugzeit.

Tätigkeiten:

Wir müssen Zeiten in Sekunden umrechnen.
Der umgekehrte Vorgang mußte auch programmiert werden.
Die Flugzeit insgesamt mußte berechnet werden.

Prozedurale Entwicklung

Entwicklung des Algorithmus

Ermittlung der benötigten Variablen

Implementierung

Kein Zusammenhang zwischen Variablen und Funktionen

Trennung von Daten (dies sind ja die Variablen) und Funktionen.

Einfache Beispiele

Kreditrechnung: Alle Variablen und Funktionen beziehen sich auf Kredite

Euro-Dollar-Umrechnung:
Alle Variablen und Funktionen beziehen sich auf Währungen

Raketenbeispiel: Alle Variablen und Funktionen beziehen sich auf Zeiten.

Prozedurale Entwicklung okay

Abbildung Lieferanten- und Kundenbeziehung

Hier gibt es Lieferanten, Kunden, Aufträge, Angebote, Lieferungen, Rechnungen und was der Dinge mehr sind.
Demzufolge später in der Anwendung Variable für:

Kundendaten,
Lieferantendaten,
Rechnungen von Lieferanten,
Rechnungen an Kunden usw.

Funktionen, um diese Variablen zu manipulieren.

Objektorientierung

Aufhebung der Trennung von Daten und Funktionen.
Objekte bestehen aus Daten und Funktionen.
Beschreibung des Problembereich
Objekt ist Abstraktion eines Gegenstandes oder Wesens der realen Welt:

Kunde
Lieferanten
Auftrag

Abstraktion auf das Wesentliche des zu lösenden Problems
Nicht relevante Details werden nicht modelliert.
Objektorientiertes Modell übernimmt also die für die betrachtete Problemstellung erforderlichen Aspekte
Die jeweils sinnvolle und hilfreiche Abstraktion zu erkennen ist gerade das zentrale Problem der objektorientierten Softwareentwicklung.

Beispiel: Software für Online-Autorennen

Autos (hier Rennwagen) sind Objekte

Eigenschaften: Farbe,
PS,
gegenwärtige Geschwindigkeit,
gegenwärtige Position auf der Rennstrecke

Funktionen:
beschleunigen
abbremsen.

Beispiel: Software für Autoverleih

Eigenschaften und Funktionen des Online-Autorennens spielen keine Rolle.

Eigenschaften:

Preis des Wagens,
Klassenzugehörigkerit
Anzahl

Funktionen:

Wagen verleihen
Wagen zurücknehmen

Fazit

unterschiedliche Anwendung <--> unterschiedliche Abstraktion

Hauptidee

Objekt ist Abstraktion eines Dings der Wirklichkeit.
Objekt wird bestimmt durch:

- Daten (Eigenschaften) und
- Funktionen (Verhalten)

Neue Namen

Daten:

Attribute,
Instanzvariablen
Eigenschaften.
properties. geprochen

Funktionen:

Methoden

Klassen und Objekte

Formel 1-Spiel:

22 Rennwagen (ab 2002 24)

Alle haben

eine Farbe,
PS,
gegenwärtige Geschwindigkeit,
gegenwärtige Position auf der Rennstrecke

Alle können

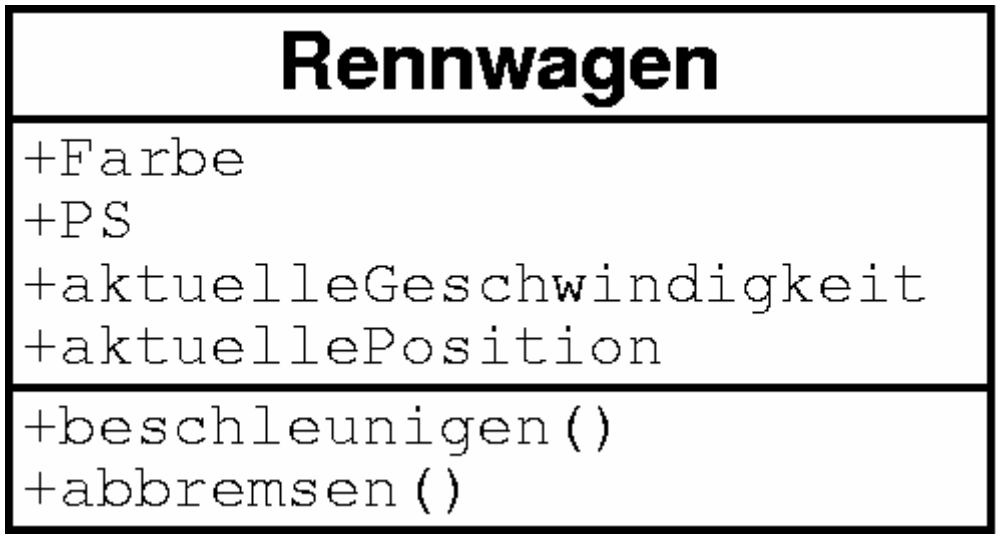
beschleunigen und
abbremsen.

ABER:

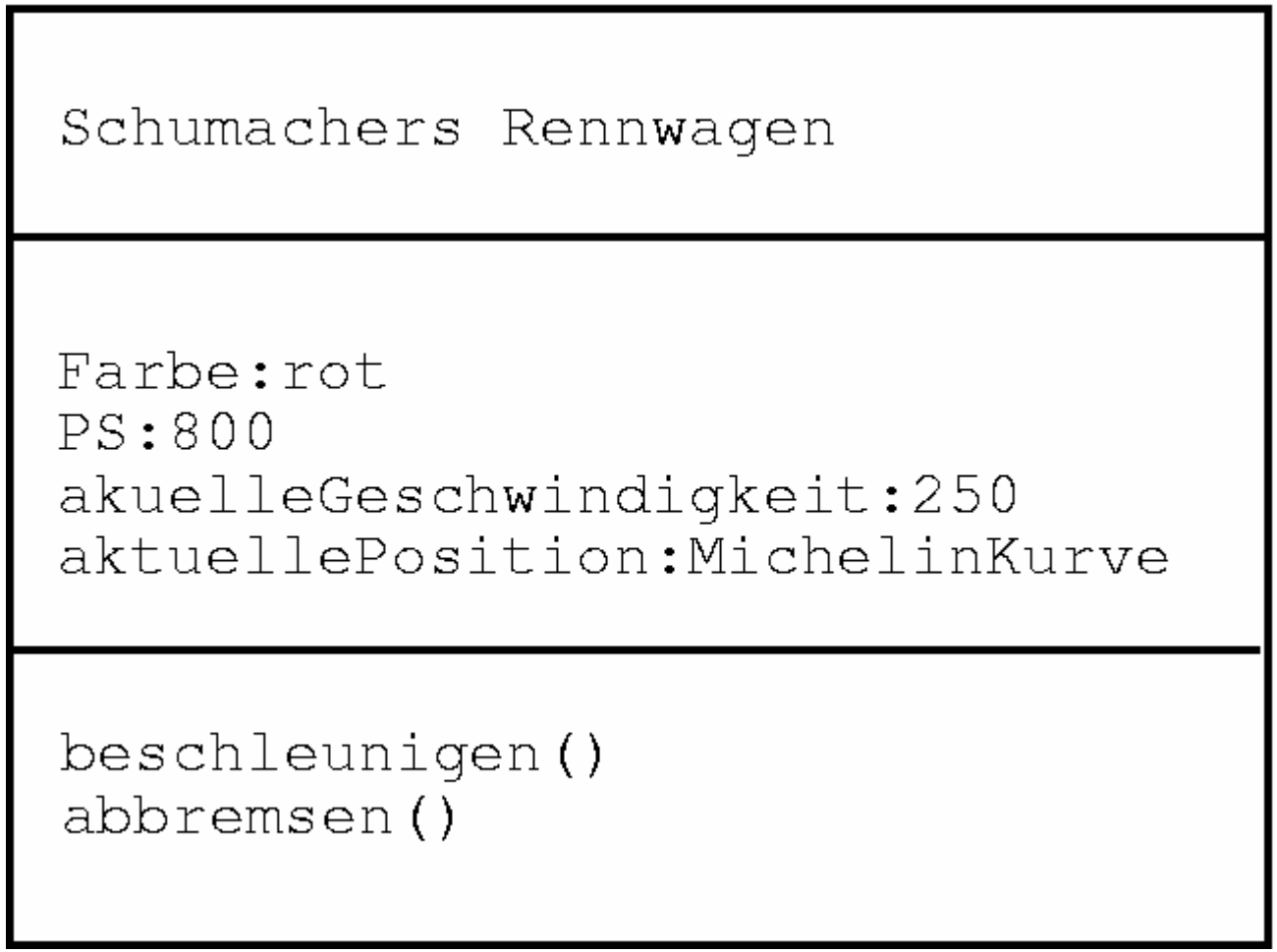
verschiedene Farben,
unterschiedliche PS
unterschiedliche gegenwärtige Geschwindigkeiten
unterschiedliche gegenwärtige Positionen

Ein Rennwagen in unserer Abstraktion (Klasse)

Aufhebung der Trennung von Daten und Funktionen.



Schumachers Rennwagen in unserer Abstraktion (Objekt)

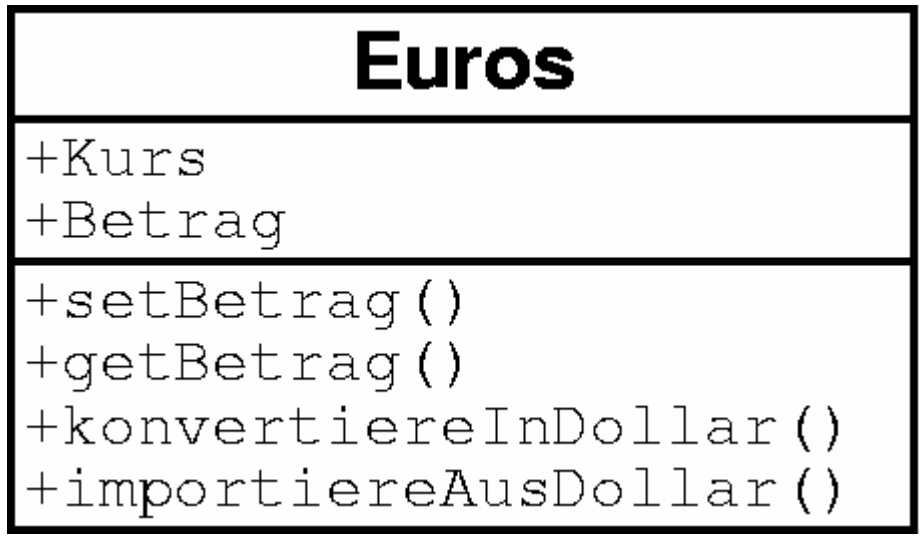


Klassen und Objekte

Klasse ist Konstruktionsvorschrift.

Objekt ist Konkretisierung einer Klasse.

Klasse Euro


Euros
+Kurs
+Betrag
+setBetrag()
+getBetrag()
+konvertiereInDollar()
+importiereAusDollar()

Klasse Euros: Die Implementierung

```
<?php
   class Euro
   {
      var $betrag;
      var $kurs=0.9;
      function setBetrag($betrag)
      {
         $this->betrag=$betrag;
      }
      function getBetrag()
      {
         return $this->betrag;
      }
      function konvertiereInDollar()
      {
         return($this->betrag*$this->kurs);
      }
      function importiereAusDollar($betrag)
      {
         $this->betrag=$betrag*(1/$this->kurs);
      }
   }
?>
```

Klasse Euros: Der Aufruf

```
else
  {
        if(($zielwaehrung=="Dollar")||($zielwaehrung=="dollar"))
        {
           $euro=new Euro();
           $euro->setBetrag($betrag);
           $dollars=$euro->konvertiereInDollar();
           echo "Ihre Eingabe entspricht $dollars! Dollar";
        }
        else
        {
           if(($zielwaehrung=="euro")||($zielwaehrung=="Euro"))
           {
              $euro=new Euro();
              $euro->importiereAusDollar($betrag);
              $euros=$euro->getBetrag();
              echo "Ihre Eingabe entspricht $euros! Euro";
           }
           else
           {
              echo("Falsche Zielw&auml;hrung: <br>" .
                     "Erlaubt sind: Euro oder Dollar!");
           }
        }
  }
```

Klasse Rakete

Klasse Rakete – Die Implementierung

```
<?php
class Rakete
{
// Raketenklasse
// Datei:Rakete.inc.php
// Verzeichnis: klassen
    var $startmonat;
    var $starttag;
    var $startstunde;
    var $startminute;
    var $startsekunde;
    var $landemonat;
    var $landetag;
    var $landestunde;
    var $landeminute;
    var $landesekunde;
```

Klasse Rakete – Die Implementierung (2)

```
 /*
*    Konstruktor
*/
function Rakete($startmonat, $starttag, $startstunde,
                $startminute, $startsekunde, $landemonat,
                $landetag, $landestunde, $landeminute,
                $landesekunde)
  {
     $this->startmonat=$startmonat;
     $this->starttag=$starttag;
     $this->startstunde=$startstunde;
     $this->startminute=$startminute;
     $this->startsekunde=$startsekunde;
     $this->landemonat=$landemonat;
     $this->landetag=$landetag;
     $this->landestunde=$landestunde;
     $this->landeminute=$landeminute;
     $this->landesekunde=$landesekunde;
  }
```

Klasse Rakete – Die Implementierung (3)

```
function berechneFlugzeit(&$flugzeitTage,&$flugzeitStunden, &$flugzeitMinuten, &$flugzeitSekunden)
 {
    $starttag=$this->tageInMonaten($this->startmonat,$this->starttag);
    // nun landetage ausrechnen
    $landetag=$this->tageInMonaten($this->landemonat,$this->landetag);
    $startzeitInSekunden=$this->berechneSekunden ($starttag,$this->startstunde,$this->startminute, $this->startsekunde);
    $landezeitInSekunden=$this->berechneSekunden($landetag,$this->landestunde,$this->landeminute,$this->landesekunde);
    // flugzeitInSekunden berechnen
    $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
    if($flugzeitSekunden<0)
    {
       return false;
    }
    $this->tageStundenMinutenSekundenAusSekunden($flugzeitTage, $flugzeitStunden, $flugzeitMinuten, $flugzeitSekunden);
    return true;
 }
```

Klasse Rakete – Die Implementierung (4)

```
function tageInMonaten($monat, $tag)
 {
    switch($monat)
    {
       case 1:
          $tag=$tag; //ueberflussig, nur der Klarheit weegen
          break;
       case 2:
          $tag=31+$tag;
          break;
       case 3:
          $tag=31+28+$tag;
          break;
       case 4:
          $tag=31+28+31+$tag;
          break;
       case 5:
          $tag=31+28+31+30+$tag;
          break;
       case 6:
          $tag=31+28+31+30+31+$tag;
          break;
```

Klasse Rakete – Die Implementierung (5)

```
case 7:
        $tag=31+28+31+30+31+30+$tag;
        break;
    case 8:
        $tag=31+28+31+30+31+30+31+$tag;
        break;
    case 9:
        $tag=31+28+31+30+31+30+31+31+$tag;
        break;
    case 10:
        $tag=31+28+31+30+31+30+31+31+30+$tag;
        break;
    case 11:
        $tag=31+28+31+30+31+30+31+31+30+31+$tag;
        break;
    case 12:
            $tag=31+28+31+30+31+30+31+31+30+31+30+$tag;
        break;
  }
  return $tag;
}
```

Klasse Rakete – Die Implementierung (6)

```
    function berechneSekunden($tag, $stunden, $minuten, $sekunden)
    {
        return($tag*24*3600+$stunden*3600+$minuten*60+$sekunden);
    }

    function tageStundenMinutenSekundenAusSekunden(&$tage,
                    &$stunden, &$minuten, &$sekunden)
    {
        // zuerst sekunden
        $minuten=floor($sekunden/60);
        $sekunden=$sekunden%60;
        //nun minuten und stunden
        $stunden=floor($minuten/60);
        $minuten=$minuten%60;
        //und tage
        $tage=floor($stunden/24);
        $stunden=$stunden%24;
    }
}
?>
```

Geheimnisprinzip

Die Benutzer unserer Klasse müssen nur zwei Dinge wissen:

- Die Argumente des Konstruktors.
- Die Signatur von berechneFlugzeit.

Klassenmethoden

Klassenmethoden vergleichbar mit Funktionen (zum Verständnis)

Methoden wurden von Objekten einer Klasse durchgeführt.

Manchmal umständlich:

Klassen ohne Instanzvariablen

Raketenbeispiel verbessert

Datumfunktionen in Raketenklasse

Schlechter Entwurf:

Andere Anwendungen benötigen vielleicht auch Datumfunktionen
Datumfunktionen gehören an sich nicht zu Raketen!

Also:

Eigene Klasse für Datumfunktionen

Klasse Datumsfunktionen

```
<?php
class Datumfunktionen
// Datei:Datumfunktionen.inc.php
// Verzeichnis: klassen
{
    function tageInMonaten($monat, $tag)
    {
        switch($monat)
        {
            case 1:
                $tag=$tag; //ueberflussig, nur der Klarheit weegen
                break;
            case 2:
                $tag=31+$tag;
                break;
            case 3:
                $tag=31+28+$tag;
                break;
            case 4:
                $tag=31+28+31+$tag;
                break;
            case 5:
                $tag=31+28+31+30+$tag;
                break;
```

Klasse Datumsfunktionen

```
        case 6:
        $tag=31+28+31+30+31+$tag;
        break;
    case 7:
                $tag=31+28+31+30+31+30+$tag;
        break;
    case 8:
        $tag=31+28+31+30+31+30+31+$tag;
        break;
    case 9:
        $tag=31+28+31+30+31+30+31+31+$tag;
        break;
    case 10:
        $tag=31+28+31+30+31+30+31+31+30+$tag;
        break;
    case 11:
        $tag=31+28+31+30+31+30+31+31+30+31+$tag;
        break;
    case 12:
        $tag=31+28+31+30+31+30+31+31+30+31+30+$tag;
        break;
  }
  return $tag;
}
```

Klasse Datumsfunktionen

```
 function berechneSekunden($tag, $stunden, $minuten,  $sekunden)
{
   return($tag*24*3600+
            $stunden*3600+$minuten*60+$sekunden);
}

function tageStundenMinutenSekundenAusSekunden(&$tage,
                &$stunden, &$minuten, &$sekunden)
{
   // zuerst sekunden
   $minuten=floor($sekunden/60);
   $sekunden=$sekunden%60;
   //nun minuten und stunden
   $stunden=floor($minuten/60);
   $minuten=$minuten%60;
   //und tage
   $tage=floor($stunden/24);
   $stunden=$stunden%24;
}
}
?>
```

Klasse Datumsfunktionen

```
function berechneFlugzeit(&$flugzeitTage,
                &$flugzeitStunden, &$flugzeitMinuten,
                &$flugzeitSekunden)
{
  $starttag=Datumfunktionen::tageInMonaten ($this->startmonat,$this->starttag);
  // nun landetage ausrechnen
  $landetag=Datumfunktionen::tageInMonaten ($this->landemonat,$this->landetag);
  $startzeitInSekunden=Datumfunktionen::berechneSekunden ($starttag,$this->startstunde,$this->startminute,
                        $this->startsekunde);
  $landezeitInSekunden=Datumfunktionen::berechneSekunden ($landetag,$this->landestunde,$this->landeminute,
                        $this->landesekunde);
  // flugzeitInSekunden berechnen
  $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
  if($flugzeitSekunden<0)
  {
     return false;
  }
  Datumfunktionen::tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                $flugzeitStunden, $flugzeitMinuten,
                $flugzeitSekunden);
  return true;
}
```

Gründe Funktionen in Klassen zu kapseln

Konvention: Dateiname entspricht Klassenname. Dateiname in der eine Funktion implementiert ist, ist sofort bekannt

Namensraum

Keine Kennzeichnung von Klassenmethoden

```
function berechneFlugzeit(&$flugzeitTage,&$flugzeitStunden, &$flugzeitMinuten,
            &$flugzeitSekunden)
  {
     $datum=new Datumfunktionen();
     $starttag=$datum->tageInMonaten($this->startmonat,$this->starttag);
     // nun landetage ausrechnen
     $landetag=$datum->tageInMonaten($this->landemonat,$this->landetag);
     $startzeitInSekunden=$datum->berechneSekunden($starttag,$this->startstunde,$this->startminute,
                                    $this->startsekunde);
     $landezeitInSekunden=$datum->berechneSekunden
            ($landetag,$this->landestunde,$this->landeminute,
                                    $this->landesekunde);
     // flugzeitInSekunden berechnen
     $flugzeitSekunden=$landezeitInSekunden-$startzeitInSekunden;
     if($flugzeitSekunden<0)
     {
        return false;
     }
     $datum->tageStundenMinutenSekundenAusSekunden($flugzeitTage,
                    $flugzeitStunden, $flugzeitMinuten,
                    $flugzeitSekunden);
     return true;
   }
```

Von JavaScript zur Verfügung gestellte Objekte

Objekte zur Vereinfachung der Programmierung (z.B. date, String)
Diese Objekte müssen erzeugt werden (new, wie in php)

Die Objekte des Browsers
Diese Objekte werden vom Browser erzeugt und stehen zur Verfügung.

Von JavaScript zur Verfügung gestellte Objekte

Aufruf von Methoden und Ansprechen von Eigenschaften

Methodenaufruf: nameDesObjekts.nameDerMethode
Beispiel: document.write("Cleverer Satz
");

Ansprechen von Eigenschaften:
Beispiel: document.write("Datum letzte Änderung" + document.lastModified);

Der Punkt-Operator ersetzt den Pfeil-Operator von php

Erstellung eigener Klassen und Objekte in JavaScript

Nicht relevant

Aufruf von statischen Methoden

NameDerKlasse.NameDerMethode

Punkt-Operator ersetzt den Doppelpunkt-Operator von php

asin

Returns the arcsine (in radians) of a number.

Method of	Math
Static	
Implemented in	JavaScript 1.0, NES 2.0
ECMA version	ECMA-262

Syntax

asin(x)

Parameters

x A number

Description

The asin method returns a numeric value between -pi/2 and pi/2 radians. If the value of number is outside this range, it returns NaN. Because asin is a static method of Math, you always use it as Math.asin(), rather than as a method of a Math object you created.

Das Schlüsselwort static sagt, dass diese Methode eine Klassenmethode ist.

Examples

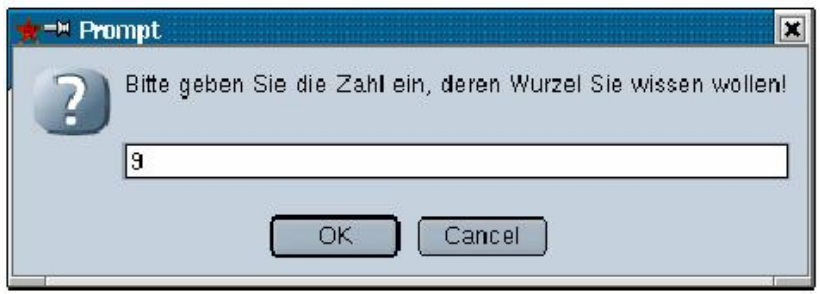
The following function returns the arcsine of the variable x:

```
function getAsin(x) {
   return Math.asin(x)
}
```

If you pass getAsin the value 1, it returns 1.570796326794897 (pi/2); if you pass it the value 2, it returns NaN because 2 is out of range.

Aufruf von Klassenmethoden (statische Methoden) Beispiel

```
<!-- Programm zur Demonstration statischer Methoden
  Dateiname: matheRichtig.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Statische Methoden</title>
</head>
<body>
    <script language = "JavaScript">
        var eingabe;
        var ergebnis;
        eingabe = parseFloat(prompt("Bitte geben Sie die Zahl ein, "+
                    "deren Wurzel Sie wissen wollen!", ""));
        ergebnis = Math.sqrt(eingabe);
        document.write("Die Wurzel ist: " + ergebnis + "<BR>");
    </script>
</body>
</html>
```



Aufruf von statische Methoden: Falsches Beispiel

```
<!-- Programm zur Demonstration statischer Methoden 2
  Dateiname: matheFalsch.html //-->
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<html>
<head>
  <title>Statische Methoden</title>
</head>
<body>
    <script language = "JavaScript">
        var matheObjekt;
        matheObjekt = new Math();
        var eingabe;
        var ergebnis;
        eingabe = parseFloat(prompt("Bitte geben Sie die Zahl ein, "+
                    "deren Wurzel Sie wissen wollen!"));
        ergebnis = matheObjekt.sqrt(eingabe);
        document.write("Die Wurzel ist: " + ergebnis + "<BR>");
    </script>
</body>
</html>
```