OOP Objektorientierte Programmierung in PHP - Part 4

Hallo liebe Community!

Dies ist mein erstes Tutorial also seit nicht zu streng mit der Kritik, über Verbesserungsvorschläge würde ich mich dennoch freuen!

Ich setze voraus, dass man weiß wie Funktionen geschrieben werden und dass man mit Variablen umgehen kann. Außerdem sollte man folgende Parts meines Tutorials gelesen haben:

- [wiki]OOP Objektorientierte Programmierung in PHP - Part 1[/wiki]
- [wiki]OOP Objektorientierte Programmierung in PHP - Part 2[/wiki]
- [wiki]OOP Objektorientierte Programmierung in PHP - Part 3[/wiki]

Ich werde weiterhin mit unseren Tollen Raumschiffen spielen xD

Jetzt kommen wir zu einem wichtigen Thema der OOP, der Vererbung. In Part 1 habe ich schon etwas angedeutet, aber hier erstmal unser Raumschiffbeispiel:
Alle Raumschiffe können fliegen, angreifen und haben Leben sowie eine Panzerung. Jäger können schnell fliegen, Zerstörer langsam, Transporter mit Mittelmäßiger Geschwindigkeit. Zerstörer haben eine starke Attacke, Jäger eine Mittelmäßige, Transporter eine Schwache. Transporter können viel Transportieren, Zerstörer ein wenig und Jäger gerade mal den nötigen Treibstoff...
Alle Raumschiffe haben also einiges gemeinsam, nur mit anderen Werten.
Damit nicht zuviel neues auf einmal kommt, werden alle Methoden und Eigenschaften ersteinmal public sein. Dazu im nächsten Part mehr.
Jetzt erstmal den Code und dann erkläre ich die Vererbung dazu.

Quellcode

```
<?php
class Raumschiff
{
public $leben;
public $panzer;
public $ladung;
public function __construct()
{
$this->leben = 100;
$this->panzer = 100;
$this->ladung = array();
}
public function fliegen()
{
echo "Captain, wir fliegen mit ".self::GESCHWINDIKEIT." Lichtjahren/Sekunde durchs All";
}
public function attacke()
{
echo "Captain, wir greifen mit ".self::ANGRIFFSWERT." Schuss/Sekunde an";
}
public function einladen($ware)
{
$bereitsGeladen = 0; //speichert was schon eingeladen wurde
foreach($this->ladung as $geladen) //Schleife die alle ladungen durchläuft
{
$bereitsGeladen += $geladen->getPlatz(); //die "aktuelle" ware prüfen, wieviel platz sie braucht
}
```

```
32.  if(($ware->getPlatz()+$bereitsGeladen)<=self::LADERAUM_KAPAZITAET) //passt die neue ladung noch rein?
33.  {
34.  $this->ladung[$ware->name] = $ware; //wenn ja, einladen
35.  } else
36.  {
37.  echo "Nicht genug Platz!"; //sonst fehler erzeugen
38.  }
39.  }
40.  public function ausladen($warenname)
41.  {
42.  $ware = $this->ladung[$warenname]; //ware zwischenspeichern
43.  unset($this->ladung[$warenname]); //ware ausladen
44.  return $ware; //zwischenspiecher zurückgeben
45.  }
46.  public function minorLeben($diff) //Leben verkleinern
47.  {
48.  if(is_numeric($diff)) //Das Leben muss um eine Zahl verkleinert werden
49.  {
50.  if($diff<0) //Wenn die Zahl kleiner als 0 ist...
51.  {
52.  $this->leben += $diff; //...wird eine negative Zahl zum leben hinzugefügt...
53.  } else //..sonst...
54.  {
55.  $this->leben -= $diff; //...wird die positive zahl abgezogen!
56.  }
57.  }
58.  }
59.  public function reparieren($diff) //Leben erhöhen
60.  {
61.  if(is_numeric($diff)) //Das Leben muss um eine Zahl verkleinert werden
62.  {
63.  if($diff>0) //Wenn die Zahl größer als 0 ist...
64.  {
65.  $this->leben += $diff; //...wird eine positive Zahl zum leben hinzugefügt...
66.  } else //..sonst...
67.  {
68.  $this->leben -= $diff; //...wird die negative zahl abgezogen! (minus und minus macht plus)
69.  }
70.  }
71.  }
72.  public function getLeben()
73.  {
74.  return $this->leben;
75.  }
76.  public function getPanzerung()
77.  {
78.  return $this->panzer;
79.  }
80.  }
81.  ?>
```

Alles anzeigen

Das ist die Klasse für das, was ALLE Raumschiffe können müssen. Alle properties und methods sollen jetzt die einzelnen Raumschiffklassen "erben".
Wie das beim erben ist, gibt es Eltern (parent) Klassen von denen geerbt wird und Kinder (child) Klassen die Erben). Ein child kann natürlich auch noch zum parent werden.
Ein child "erbt" alle Methoden und Eigenschaften vom parent. Was heißt das "erben" nun genau? Nunja, alle properties und methods werden auf das Kind übertragen, auch wenn in der Kindklasse keine als Beispiel function reparieren existiert

wird sie vom parent übernommen. Möchte man nun aber eine andere function reparieren (weil es ein Raumschiff ist, das andere Schiffe reparieren kann mit der Funktion) so schreibt man für die Kindklasse eine function reparieren. Diese überschreibt die geerbte Funktion. Um auf die geerbte Funktion dennoch zurückzugreifen gibt es das Keyword "parent". So, genug der Worte, schauen wir uns mal eine Kindklasse an:

Quellcode

```
1. <?php
2. class Jaeger extends Raumschiff //Die Klasse Jäger "erbt" von der Klasse Raumschiff
3. {
4. const GESCHWINDIKEIT = 50;
5. const ANGRIFFSWERT = 50;
6. const LADERAUM_KAPAZITAET = 10;
7. public function __construct()
8. {
9. parent::__construct(); //Standardwerte über den geerbten Konstruktor laden
10. $this->panzer = 25; //Standard Panzerung ist nicht für den Jäger gültig...
11. }
12. }
13. ?>
```

Alles anzeigen

Dieser Jäger kann nun fliegen, angreifen transportieren, alles was Raumschiffe halt können. Wichtig zu erwähnen ist noch, dass wenn es eine Reihe von vererbungen gibt, immer die "nächst höhere" Klasse durch parent "referenziert" wird.

In diesem Tutorial-Part gab es viele neue Informationen eng gepackt... Ihr solltet auch jeden Fall die Funktionen die unsere Klasse Jaeger hat ausprobieren! Außerdem empfehle ich das ihr noch die Klassse zu Zerstörer und Transporter schreibt! Wenn ihr Probleme (beim Verständniss des Tutorials oder beim coden) habt, schreibt mir eine PN dann kann ich das im Tutorial vllt. noch vertiefen!

Im nächsten Part geht es um die Visibility: [wiki]OOP Objektorientierte Programmierung in PHP - Part 5[/wiki]
n0x-f0x