OOP Objektorientierte Programmierung in PHP - Part 6

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]
- [wiki]OOP Objektorientierte Programmierung in PHP - Part 4[/wiki]
- [wiki]OOP Objektorientierte Programmierung in PHP - Part 5[/wiki]

Dieser Part behandelt die Vererbung nochmal genauer.

Hier der Quellcode den ich in diesem Part besprechen will. Er setzt sich aus dem Quellcode von Part 4, dem Wissen aus Part 5 und dem neuen aus diesem Part zusammen:

Quellcode

```
1. abstract class Raumschiff
2. {
3. protected $leben; //protected ist aus Part 5
4. protected $panzer;
5. protected $ladung;
6. 
7. public function __construct()
8. {
9. $this->leben = 100;
10. $this->panzer = 100;
11. $this->ladung = array();
12. }
13. 
14. public function fliegen()
15. {
16. echo "Captain, wir fliegen mit ".self::GESCHWINDIKEIT." Lichtjahren/Sekunde durchs All";
17. }
18. 
19. public function attacke()
20. {
21. echo "Captain, wir greifen mit ".self::ANGRIFFSWERT." Schuss/Sekunde an";
22. }
23. 
24. public function einladen($ware)
25. {
26. $bereitsGeladen = 0; //speichert was schon eingeladen wurde
27. foreach($this->ladung as $geladen) //Schleife die alle ladungen durchläuft
28. {
29. $bereitsGeladen += $geladen->getPlatz(); //die "aktuelle" ware prüfen, wieviel platz sie braucht
30. }
31. if(($ware->getPlatz()+$bereitsGeladen)<=self::LADERAUM_KAPAZITAET) //passt die neue ladung noch rein?
32. {
33. $this->ladung[$ware->name] = $ware; //wenn ja, einladen
34. } else
35. {
36. echo "Nicht genug Platz!"; //sonst fehler erzeugen
37. }
```

```
38. }
39. public function ausladen($warenname)
40. {
41. $ware = $this->ladung[$warenname]; //ware zwischenspeichern
42. unset($this->ladung[$warenname]); //ware ausladen
43. return $ware; //zwischenspiecher zurückgeben
44. }
45. public function minorLeben($diff) //Leben verkleinern
46. {
47. if(is_numeric($diff)) //Das Leben muss um eine Zahl verkleinert werden
48. {
49. if($diff<0) //Wenn die Zahl kleiner als 0 ist...
50. {
51. $this->leben += $diff; //...wird eine negative Zahl zum leben hinzugefügt...
52. } else //..sonst...
53. {
54. $this->leben -= $diff; //...wird die positive zahl abgezogen!
55. }
56. }
57. }
58. public function reparieren($diff) //Leben erhöhen
59. {
60. if(is_numeric($diff)) //Das Leben muss um eine Zahl verkleinert werden
61. {
62. if($diff>0) //Wenn die Zahl größer als 0 ist...
63. {
64. $this->leben += $diff; //...wird eine positive Zahl zum leben hinzugefügt...
65. } else //..sonst...
66. {
67. $this->leben -= $diff; //...wird die negative zahl abgezogen! (minus und minus macht plus)
68. }
69. }
70. }
71. public function getLeben()
72. {
73. return $this->leben;
74. }
75. public function getPanzerung()
76. {
77. return $this->panzer;
78. }
79. }
80. class Jaeger extends Raumschiff //Die Klasse Jäger "erbt" von der Klasse Raumschiff
81. {
82. const GESCHWINDIKEIT = 50;
83. const ANGRIFFSWERT = 50;
84. const LADERAUM_KAPAZITAET = 10;
85. public function __construct()
86. {
87. parent::__construct(); //Standardwerte über den geerbten Konstruktor laden
88. $this->panzer = 25; //Standard Panzerung ist nicht für den Jäger gültig...
89. }
90. }
```

Alles anzeigen

Das neue findet sich gleich in der 1.Zeile. Das keyword abstract (abstrakt). Wird eine Klasse als abstact definiert, so kann man keine Instanz dieser Klasse (also kein Objekt der Klasse) erzeugen. Man kann auch Methoden als abstract

definieren, jedoch MUSS eine Klasse, sobald sie auch nur eine einzige abstakte Methode hat, auch abstrakt sein.
Jetzt auf unsere Raumschiffe bezogen: es sollte NIE das Raumschiff "Raumschiff" geben, das ist ja nur der Prototyp für alle anderen Raumschiffe, deswegen muss diese Klasse abstract sein! Jetzt zu den abstract methods:

Quellcode

```
1. abstract class Raumschiff
2. {
3. //Das ganze BlaBla von oben
4. abstract public function spezialLaserKanone();
5. }
6. class Jaeger extends Raumschiff
7. {
8. //auch das Bla von oben
9. public function spezialLaserKanone()
10. {
11. echo "keine Laserkanone!";
12. }
13. }
```

Alles anzeigen

Wer möchte kann den Code mal ohne die Funktionsdefinition in der Klasse Jaeger laufen lassen. Das wird Fehler erzeugen! Einer abstrakten Methode wird kein Quellcode zugeordnet. Sie hat nur eine Funktion: die Kindklasse dazu "zwingen", eine Methode mit dem selben Namen und mit den selben Parametern (die Parameter dürfen sich nicht in Anzahl oder Namen unterscheiden) zu definieren. Dabei ist zu beachten das es nur strengere Sichtbarkeiten gibt. Also wenn man protected angegeben hat muss die Funktion nachher protected oder private sein.

Das abstract keyword wird mit PHP 5 eingeführt und ist voher nicht verfügbar!

Im nächsten Part wird nochmals auf Vererbung eingegangen.
n0x-f0x