Objektorientierung mit Exceptions

Definition

Exceptions sind Ausnahmebehandlungen im Programmcode.
Die Verwendung ist einfach. Durch werfen von Exceptions mittels throw new Exception('foo') sorgt man dafür, dass der Programmfluss abbricht.
Durch Verwendung von try {} catch(Exception $e) kann man den Fehler dann behandeln ohne, dass der komplette Programmcode abbricht. Es gleicht der Abbruchbedingung in einer Schleife.

Inhaltsverzeichnis

- 1 Definition
- 2 Das Spiel der Verantwortlichkeiten
- 3 Beispiel
  - 3.1 Unbehandelete Exception
  - 3.2 Behandelte Exception

Das Spiel der Verantwortlichkeiten

Das spannende an Exceptions ist die Frage wo man sie am besten platziert und wie man sie verwendet.
Objektorientierung und vor allem die Frage wo platziert man welchen Code ist ein Spiel mit Verantwortlichkeiten.
Ist es Aufgabe des Formulars (also dem "Controller") die Fehleingaben automatisch zu korrigieren, oder ist es Aufgabe des Benutzer Models optionale Angaben durch Defaults zu ersetzen.

Eine einfache Frage die ich häufig nutze um das Ganze zu erläutern ist: An welchen Stellen benötigt man die Funktion denn noch.
Wenn dann nicht sofort und ohne Zweifel eine Antwort kommt, dann heißt das doch, dass man die Funktion theoretisch überall einbauen kann.
Und damit gehört der Code an den abstraktesten Punkt an dem es möglich ist und was von überall verwendet wird.

Beispiel

Exceptions sind für Ausnahmen die man nicht automatisch auflösen kann oder will. Gehen wir von einem User Modell aus das auf dem abstraken Modell des Tutorials [wiki]Objektorientierung mit Models[/wiki] aufbaut.

Quellcode

```
1.  class User extends Model {
2.  const MIN_PASSWORD_LENGTH = 8;
3.  public function validate() {
4.  $error = array();
5.  if(empty($this->firstname)) {
6.  $error['firstname'] = 'firstname';
7.  }
8.  if(strlen($this->password) < self::MIN_PASSWORD_LENGTH) {
9.  $error['password'] = 'to short';
10. }
11. if(count($error)) {
12. throw new ValidateException($error);
13. }
14. }
15. }
```

Alles anzeigen

Man sieht, dass ich keine normale Exception Klasse gewählt habe, sondern stattdessen eine ValidateException verwende, die ein Array als Parameter empfängt. Das mache ich wegen der häufigen Anforderung, dass man mehrere Validierungen bei gleichzeitiger Benutzereingabe durchführt.

Die Klasse sieht wie folgt aus

Quellcode

```
1.  class ValidateException extends Exception {
```

```
2. protected $errors;
3. public function __construct($list) {
4. $this->errors = is_array($list) ? $list : array($list);
5. parent::__construct('validation');
6. }
7. public function getErrors() {
8. return $this->errors;
9. }
10. }
```

Unbehandelete Exception

Speicher ich ein Formular mit fehlenden Daten ab, dann erhalte ich eine unbehandelete Exception die zum Abbruch des Programms führt:

Quellcode

```
1. $user = new User();
2. $user->save(array( /* empty */ ));
```

Die Ausgabe lautet wie folgt: **Fatal error: Uncaught exception 'ValidateException' with message 'validation'**

Behandelte Exception

Im konkreten Fall wollen wir den Benutzer bei einer Formulareingabe aber darauf aufmerksam machen, dass er seine Eingaben doch möglichst ergänzen soll.
Dazu "fangen" wir die Exception und stellen die Fehlermeldungen dar. Der Programmfluss des Speicherns wurde beim throw new Exception abgebrochen - wir haben also keinen Speichervorgang durchgeführt.

Quellcode

```
1. <?php
2. function i18n($var) {
3. return 'translation for '.$var;
4. }
5. $errors = array();
7. $message = '';
8. $user = null;
9. if(count($_POST)) {
10. $user = new User($_POST);
11. try {
12. $user->save();
13. } catch(ValidateException $e) {
14. $errors = $e->getErrors();
15. foreach($errors as $error) {
16. $message .= i18n('error.'.$error).'<br/>';
17. }
18. }
19. }
20. ?>
21. <?= $message ?>
22. <form method="post">
23. <div<?if (array_key_exists('email', $errors)) echo ' style="background-color:#ffaaaa"' ?>>email:
24. <input type="text" name="email" value="<?if($user) echo $user->email?>" />
25. </div>
26. <div<?if (array_key_exists('password', $errors)) echo ' style="background-color:#ffaaaa"' ?>>password:
27. <input type="password" name="password" value="<?if($user) echo $user->password?>"/>
28. </div>
```

```
29. <div<?if (array_key_exists('firstname', $errors)) echo ' style="background-color:#ffaaaa"' ?>>firstname:
30. <input type="text" name="firstname" value="<?if($user) echo $user->firstname?>"/>
31. </div>
32. <div<?if (array_key_exists('lastname', $errors)) echo ' style="background-color:#ffaaaa"' ?>>lastname:
33. <input type="text" name="lastname" value="<?if($user) echo $user->lastname?>"/>
34. </div>
35. <input type="submit">
36. </form>
```

Alles anzeigen