Übungsaufgabe 23 - OOP - Klassendefinition

23.a) Klasse `User` für unser Projekt Benutzerverwaltung

Wir wollen unser Projekt Benuzterverwaltung natürlich auch objektorientiert umsetzen und dazu benötigen wir eine Klasse, deren Instanzen später jeweils genau einen User repräsentieren.

In den Übungsaufgaben

- Aufgabe 11 - Datenerzeugung
- Aufgabe 12 - Registrierungsformular und Formular für Benutzerdatenänderung
- Aufgabe 18 - Datenpersitierung
- und im Workshop zu PDO tw. in Aufgabe 20 PDO - standard Datenbankoperationen

hatten wir bereits die Datenstruktur als Array, JSON und auch bereits in MySQL (Datenbank `user_mangement` - Tabelle `users`) erstellt, so dass die Attribute für die Klasse bereits mehr oder weniger bestimmt sind.

```
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(2)      | NO   | PRI | NULL    | auto_increment |
| firstname  | varchar(50) | YES  |     | NULL    |                |
| lastname   | varchar(30) | YES  |     | NULL    |                |
| email      | varchar(40) | NO   | UNI | NULL    |                |
| password   | varchar(15) | YES  |     | NULL    |                |
| role       | varchar(8)  | YES  |     | NULL    |                |
| created_at | varchar(19) | YES  |     | NULL    |                |
| updated_at | varchar(19) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
```

Aufgabe: Erstelle eine Klasse `User`, die über die den Spalten unserer Tabelle entsprechende Attribute verfügt.

Alle Attribute sollen vor einem Direktzugriff geschützt sein (entweder `protected` oder `private`) und dementsprechend werden natürlich Getter- und Setter-Methoden für die Attribute benötigt.

> **Wichtig:** wir wollen uns an die Konventionen für PHP-Bezeichner halten, d. h. die Attributsbezeichner sollten in `camelCaseNotation` und nicht in 'snake_case_notation' definiert werden.

23.b) Refactoring der Klasse `User` für unser Projekt Benutzerverwaltung

Aufgabe: die Klasse `User` soll für die Verwendung als Datenklasse für die spätere Umsetzung des Acitve-Record-ORM-Pattern überarbeitet werden.

Folgende Schritte hatten wir alle bereits in den Workshopbeispielen gemacht. Du kannst natürlich hier nachsehen und/oder auch den bereits erstellten Code wiederverwenden (muss aber ggf. angepast werden):

1. Password-Hashing

In einer Datenbank sollte **niemals** Passwörter in Klartext gespeichert werden. Das sollten wir gleich in unserem Datenobjekten berücksichtigen.

Aufgabe 1: Ändere den Setter für das Attribut `password` so, dass ein übergebenes Passwort automatisch gehasht wird, wenn der Wert für das Attribut gesetzt wird.
Wie man ein Password hasht, kann Du in unserer Workshopdatei `/phpOOP/lek06/inc/User.php` nachsehen (bzw. `/phpOOP/lek04/benutzer_version_6.php`) oder auch auf php.net nachlesen.

2. magische Methoden `__get` und `__set`

Beim Arbeiten mit Objekten in PHP, gerade im View (bei der Ausgabe) ist es wünschenswert tw. auch üblich, dass man direkt auf die Attribute zugreifen kann (zmindest lesend), also z. B. `echo $user->firstname` alternativ zu `echo $user->getFirstname()`. Häufig verwendet man den direkten Zugriff dann, wenn der Attributswert unverändert ausgegeben werden soll und die Getteraufrufe, wenn noch Änderungen wie z. B. Formatierungen o. ä. durchgeführt werden sollen/müssen.

Aufgabe 2.1: implementiere die magische Methode `__get` in die Klasse `User`, so dass ein direkter lesender Zugriff (z. B. `$user->firstname`) auf unsere `protected` Attribute ermöglicht (ändere die Sichtbarkeit bitte nicht!) und der Wert des Attributes (unverändert, also nicht über den Getteraufruf) zurückgeben wird. Wird auf ein Attribut zugegriffen, dass nicht existiert, soll der Wert `NULL` zurückgebenen werden.

Aufgabe 2.2: implementiere nun die magische Methode `__set` in die Klasse `User`, so dass ein direkter schreibender Zugriff (z. B. `$user->firstname = 'Hans';`) auf unsere `protected` Attribute möglich wird. Dabei soll zunächst überprüft werden, ob der entsprechende Setter vorhanden ist und wenn ja, dieser aufgerufen werden. Ist der Setter nicht implementiert, soll überprüft werden, ob dass Attribut existiert und wenn ja, dann diesem der Wert direkt zu gewiesen werden. Wird auf ein Attribut zugegriffen, dass nicht existiert, soll nichts passieren.

3. magische Methoden `__call`

> ***"Programmierer sind faul"*** Das trifft zwar nicht auf uns zu, dennoch ist es nicht zwingend notwendig für jedes protected geschützte Attribut jeweils einen Getter- und Setter zu implementieren, insbesondere, wenn hier der eigentliche Attributswert unverändert bleibt, d. h. beim Lesen direkt der Attributswerte zurückgegeben wird und beim Schreiben der übergebene Wert unverändert dem Attribut zugewiesen wird.

Aufgabe 3.1: Lösche alle Getter- und Setter für die das zutreffend ist (also **nicht** den Setter für das Attribut `password` und auch **nicht** die für unser virtuelles Attribut `attributes` bzw. `daten` oder `data`!).

Aufgabe 3.2: Implementiere nun die magische Methode `__call`, so alle Getter- und Setteraufrufe nach wie vor möglich sind, auch wenn diese ja teilweise garnicht mehr vorhanden sind.

> Tipp: hier musst Du jetzt umgekehrt vorgehen wie bei unserer Konstruktormethode bzw. der `setAttributes`-Methode

a. prüfe erst, ob der Getter- oder Setter existiert, wenn ja rufe ihn auf
b. wenn nein, extrahiere den Namen des Attributs aus dem Parameter mit dem Methodennamen
c. prüfe, ob das Attribut existiert und wenn ja, führe den lesenden oder schreibenden Zugriff aus

> Achte dabei darauf, dass ein Getter einen Wert zurückgibt, d. h. hier brauchen wir auch ein `return`

Aufgabe 4: durch die Änderungen der Teilaufgabe 3 funktioniert nun unser Konstruktor bzw. der Setter für unser virtuelles Attribut `attributes` (bzw. `daten` oder `data`) nicht mehr.
Das liegt daran, dass wir ja die meisten Setter gelöscht haben und somit nur die Attribute befüllt werden, für die noch ein Setter vorhanden ist (das dürfte also nur noch das Attribut `password` sein)
Korrigiere den Fehler, in dem Du das bestehende `if`, in dem wir prüfen ob der Setter existiert, um eine `else`-Klausel erweiters, in der Du prüftst, ob das Attribute existiert und wenn ja, dann den Wert direkt diesen Attribut zuweist.