هدف جلسه ۲۸ چیه؟

جلوگیری از ثبت مقادیر نامعتبر توسط کاربر (مثل فیلد خالی، یا ایمیل اشتباه) و ساخت یک کلاس کمکی به نام `Validator` **برای ساده‌سازی اعتبارسنجی داده‌ها.**

---

1 ساخت فایل Validator.php

داخل این فایل یک کلاس ایجاد می‌کنیم:

```
<?php

class Validator {
    public static function stringg($value , $min = 1 , $max = INF) {
        $value = trim($value);
        return strlen($value) >= $min && strlen($value) <= $max;
    }

    public static function email($value) {
        return filter_var($value, FILTER_VALIDATE_EMAIL);
    }
}
```

📌نکات مهم:

- `static`: **یعنی لازم نیست از کلاس Validator یک شی بسازی. مستقیماً با** `Validator::method()` **صدا زده می‌شه.**
- `trim($value)`: **برای حذف فاصله‌های اول و آخر متن.**
- `INF`: **به معنی بی‌نهایت (یعنی اگر حد بالا تعیین نشه، بی‌نهایت در نظر بگیره).**
- `filter_var(..., FILTER_VALIDATE_EMAIL)`: **برای بررسی معتبر بودن ایمیل.**

2️⃣استفاده در فایل note_create.php

در این فایل، ابتدا کلاس را اضافه می‌کنیم:

```
require "Validator.php";
```

سپس استفاده از آن به این شکل:

```
if (! Validator::stringg($_POST["body"], 1, 1000)) {
    $errors["body"] = "Body is required and must be
less than 1000 characters.";
}
```

یا برای ایمیل:

```
if (! Validator::email($_POST["email"])) {
    $errors["email"] = "Please enter a valid email
address.";
}
```

✅نتیجه نهایی چی می‌شه؟

- ما دیگه چک کردن خالی بودن یا معتبر بودن ایمیل رو داخل note_create.php نمی‌نویسیم.
- همه چک‌ها به صورت متد در کلاس Validatorتعریف شدن.
- کد مرتب‌تر، تمیزتر، قابل استفاده مجدد و قابل تست می‌شه.

🔹بررسی دقیق متد email در کلاس Validator

```
public static function email($value){
    return filter_var($value, FILTER_VALIDATE_EMAIL);
}
```

🔍خط به خط توضیح:

- public static:متد **عمومی و ایستا** است؛ یعنی مستقیماً با Validator::email()استفاده می‌کنی.

- `function email($value):` متدی به نام `email` تعریف شده که یک پارامتر `$value` می‌گیره (معمولاً یک ایمیل کاربر).
- `filter_var($value, FILTER_VALIDATE_EMAIL):` این تابع داخلی PHP بررسی می‌کنه که مقدار `$value` آیا **ساختار یک ایمیل معتبر** رو داره یا نه.
  - اگر ایمیل معتبر باشه، همون مقدار رو برمی‌گردونه.
  - اگر ایمیل اشتباه باشه (مثلاً بدون @ یا با فاصله یا ...)، مقدار `false` برمی‌گردونه.

✓*مثال:*

```
if (!Validator::email($_POST['email'])) {
    $errors['email'] = 'This email address is not
valid.';
}
```

اگر ایمیل معتبر نباشه، خطا به آرایه `$errors` اضافه میشه.

✓سؤال ۱:

کدام گزینه صحیح‌ترین روش برای صدا زدن متد `stringg` در کلاس `Validator` است که به صورت static تعریف شده؟

الف) `$validator->stringg('abc');`

ب) `Validator->stringg('abc');`

ج) `Validator::stringg('abc');` ✓

د) `stringg::Validator('abc');`

✓سؤال ۲:

متد `filter_var($value, FILTER_VALIDATE_EMAIL)` در صورت اعتبار ایمیل، چه مقداری برمی‌گرداند؟

الف) `true`

ب) `false`

ج) مقدار اصلی `$value` ✅

د) هیچ مقداری برنمی‌گرداند

✅سؤال ۳:

اگر ایمیل نامعتبر باشد، خروجی `filter_var($value, FILTER_VALIDATE_EMAIL)` چیست؟

الف) رشته خالی `""`

ب) `false` ✅

ج) `null`

د) `true`

✅سؤال ۴:

تفاوت اصلی متد static و غیر static-در چیست؟

الف) متد static فقط داخل توابع دیگر استفاده می‌شود

ب) متد static به `$this` دسترسی ندارد✅

ج) متد static فقط در کلاس‌هایی با نام خاص استفاده می‌شود

د) متد static حافظه کمتری مصرف می‌کند

✅سؤال ۵:

کدام گزینه زیر یک استفاده‌ی نادرست از متد static در PHP است؟

```
class Test {
```

```
    public static function hello() {
        return 'hi';
    }
}
```

الف) `Test::hello();`

ب) `$t = new Test(); $t->hello();` ✅

ج) `echo Test::hello();`

د) هیچ‌کدام

✅سؤال ۶:

اگر کد زیر را بنویسیم، چه اتفاقی می‌افتد؟

```
Validator::stringg("hello", 2, 4);
```

الف) (خروجی `true` **می‌دهد**

ب) (خروجی `false` **می‌دهد**✅

ج) (ارور syntax **می‌دهد**

د) (هیچ خروجی ندارد

چون طول کلمه "hello" **برابر با ۵ است و در محدوده ۲ تا ۴ نیست،** false **برمی‌گردد.**

✅سؤال ۷:

هدف اصلی استفاده از `trim()` **در تابع‌های** Validator **چیست؟**

الف) (تبدیل متن به عدد

ب) (حذف فاصله‌های اول و آخر رشته✅

ج) (چک کردن ایمیل

د) (جلوگیری از SQL Injection

✅سؤال ۸:

در کلاس `Validator`اگر متدی به صورت غیر static-تعریف شود، برای استفاده از آن چه کاری باید انجام دهیم؟

الف (نیازی به کاری نیست
ب (باید نام کلاس را با : : صدا زد
ج (باید اول یک object بسازیم✅
د (فقط در فایل‌های include شده قابل استفاده است

✅سؤال ۹:

فرض کن کدی داریم:

```
if (!Validator::email("test@@example.com")) {
    echo "Invalid!";
}
```

خروجی چیست؟

الف✅ !Invalid)
ب (هیچ خروجی ندارد
ج (ارور می‌دهد
دtrue)

✅سؤال ۱۰:

کاربرد `INF`در کد زیر چیست؟

```
public static function stringg($value , $min=1 , $max=INF)
```

الف (مقدار اولیه رشته را به بی‌نهایت تبدیل می‌کند

ب (اجازه می‌دهد حداکثر طول محدود نشود✓

ج (ارور syntax است

د INF (در PHP وجود ندارد

توضیحات خودم :

یک کلاس با نام validator.php ایجاد کرده و درونش کد زیر را مینویسیم:

```
<?php
   class Validator {
      public function string($value){
         return strlen($value) == 0;
      }
   }
```

حالا به فایل note_create.php رفته ابتدا این فایل را require میکنیم تا از کلاسش استفاده کنیم بعد کد زیر را مینویسیم:

```
$validator = new Validator();
if($validator->string($_POST['body'])){
    $error["body"] = "Body is required";
}
```

در این حالت وقتی کاربر space بزنه بازم مقدار خالی زخیره میشه که برای جلوگیری از این کار از ()trim در کلاس validator استفاده میکنیم.

```
public function string($value){
    return strlen(trim($value)) == 0;
}
```

وقتی در داخل یک کلاس نیازی به نمونه ساخت شده از کلاس نیست و مستقیما با خود کلاس کار داریم دراین صورت متد رو به صورت static تعریف میکنیم و بعدا برای استفاده نیازی ایجاد یک نمونه از کلاس نیست و مستقیما با :: میتونیم استفاده کنیم:

وقتی در PHP داخل یک کلاس، متدی رو به صورت **static** تعریف می‌کنی، یعنی این متد **مستقیماً به خود کلاس مربوطه** و نه به نمونه (object) ساخته‌شده از اون کلاس.

مثال:

```php
<?php
    class Validator {
        public static function stringg($value , $min=1 ,
$max=INF){
            return strlen(trim($value)) >= $min &&
strlen(trim($value)) <= $max;
        }
    }
```

حالا در فایل note_create.php به صورت زیر از کلاس استفاده میکنیم:

```php
if(validator::stringg($_POST["body"],1,1000)){
    $error["body"] = "body is too long ";
}
```

دسترسی به متد static با : :

وقتی متدی staticباشه، برای استفاده از اون **نیازی به ساختن شیء (object)** از کلاس نداری. فقط کافیه از **(scope resolution operator) : :** استفاده کنی:

```
echo Validator::string();  // خروجی: This is a string
validator!
```

بعد اومد برای بررسی ایمیل داخل کلاس یک متد به صورت زیر تعریف کرد:

```php
public static function email($value){
    return filter_var($value, FILTER_VALIDATE_EMAIL);
}
```