PIEMONTE
FONDI STRUTTURALI E DI INVESTIMENTO EUROPEI 2014/2020
fondo sociale europeo
ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Programmazione a oggetti
(III parte)

Corso Backend System Integrator

Modulo **Programmazione PHP**

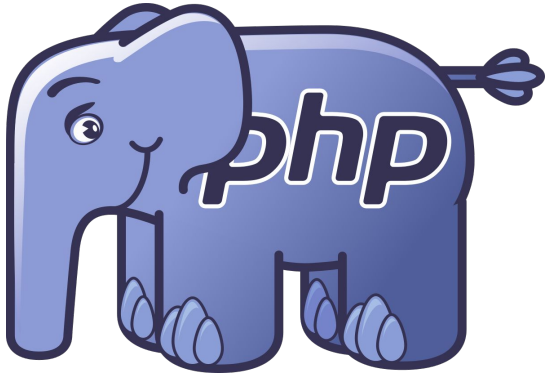
Docente: Dott. Enrico Zimuel

in collaborazione con:
MIUR
UNIONE EUROPEA
REGIONE PIEMONTE
per una crescita intelligente, sostenibile ed inclusiva
www.regione.piemonte.it/europa2020
INIZIATIVA CO-FINANZIATA CON FSE

ITS TECNOLOGIE DELL'INFORMAZIONE E DELLA COMUNICAZIONE

Programma

- Final
- Interfacce
- Entità statiche
- Traits
- Classi anonime



Final

- In PHP è possibile limitare la possibilità di modificare (estendere) una funzione o una classe con il costrutto **final**
- E' possibile di fatto impedire che una classe venga estesa o una funzione ridefinita

Esempio

```php
abstract class User
{
    protected $name = '';
    final public function getName(): string {
        return $this->name;
    }
    final public function setName(string $name) {
        $this->name = $name;
    }
    abstract public function save();
}
```

Esempio

```php
final class User
{
    protected $name = '';
    public function getName(): string {
        return $this->name;
    }
    public function setName(string $name) {
        $this->name = $name;
    }
}
```

Interfacce

- L’ereditarietà di classe consente di definire una gerarchia tra classi ma non consente di definire l’aspetto di una determinata classe
- Nella programmazione ad oggetti un’**interfaccia** è un **contratto** per la creazione di un classe
- Questo contratto è la specifica dei metodi, inclusi i parametri in ingresso e uscita, che una classe dovrà implementare

Sintassi

```
interface <name>
{
   public function <name-1>(<params-1>) : <return-1>;
   // ...
   public function <name-n>(<params-n>) : < return-n>;
}
```

Interfacce

- Non è possibile inserire l'implementazione di un metodo, solo la specifica dei parametri in ingresso e in uscita
- Si possono specificare solo metodi pubblici
- E' possibile definire anche il costruttore (__*construct*)

Esempio

```php
interface UserInterface
{
    public function getName(): string;
    public function setName(string $name): void;
}
```

Come utilizzare un’interfaccia

- Una classe può implementare un’interfaccia utilizzando la sintassi **implements**
- Se una classe non implementa tutti i metodi dell'interfaccia o se i metodi non rispettano il contratto, il PHP genera un errore

Esempio

```php
class User implements UserInterface
{
    protected $name = '';
    public function getName(): string {
        return $this->name;
    }
    public function setName(string $name): void {
        $this->name = $name;
    }
}
```

```php
interface UserInterface
{
    public function getName(): string;
    public function setName(string $name): void;
}
```

Interfacce multiple

- Una classe può implementare anche più interfacce contemporaneamente
- Le interfacce non possono avere metodi in comune
- Non deve esserci ambiguità sui metodi da implementare

Esempio

```
class User implements UserInterface, AdminInterface
{
    // ...
}
```

instanceof

- E’ possibile verificare che una classe implementi un’interfaccia con l’operatore **instanceof**

```php
if ($user instanceof UserInterface) {
    // ...
}
```

Esercizio

- Implementare una classe Studente rispettando le interfacce **StudenteInterface** e **CorsiInterface** presenti nel repository github: https://github.com/ezimuel/Corso_PHP_ITS

Entità statiche

- In PHP è possibile de nire metodi, proprietà e classi **statiche**
- Il termine **statico** viene utilizzato per identificare il legame alla classe e non alla sua istanza
- Per poter accedere a un’entità statica è necessario utilizzare l’operatore **::**

Esempio

```php
class Foo {
    public static function hi() {
        echo "hello!";
    }
}

Foo::hi(); // hello!
```

I metodi statici possono essere richiamati senza dover istanziare la classe

Metodi statici

- Non è possibile utilizzare l'istanza **$this** all'interno di un metodo statico

```php
class Foo {
   protected $msg = 'hello!';
   public static function hi() {
        echo $this->msg;
   }
}

Foo::hi(); // PHP Fatal error
```

Valori statici

- Una variabile statica può avere una sola istanza, quella della classe

```
class Foo {
   static public $i = 0;
}
$a = new Foo();
printf("%d %d\n", $a::$i, Foo::$i); // 0 0
Foo::$i++;
printf("%d %d\n", $a::$i, Foo::$i); // 1 1
$b = new Foo();
printf("%d %d\n", $b::$i, Foo::$i); // 1 1
```

Trait

- I **Trait** sono un costrutto del PHP che consente il **riutilizzo di codice** all'interno di classi di erenti
- Consente di **condividere** del codice comune tra più classi, senza nessun legame di ereditarietà
- Può essere utilizzato per mitigare il problema dell'ereditarietà multipla, funzionalità non supportata dal linguaggio

Esempio

```php
trait UserName {
    protected $name = '';
    public function getName() : string {
        return $this->name;
    }
    public function setName(string $name) {
        $this->name = $name;
    }
}
class User {
    use UserName;
    // ...
}
```

Trait multipli

```php
trait UserGithub {
    protected $github = '';
    public function setGithub(string $github)
    {
        $this->github = $github;
    }
    public function getGithub(): string
    {
        return $this->github;
    }
}
class Developer {
    use UserName, UserGithub;
}
```

insteadof

- Non si possono utilizzare contemporaneamente più Trait con nomi di funzioni in comune
- In caso di conflitti è possibile utilizzare l'istruzione **insteadof** per specificare quale entità utilizzare

Esempio

```
trait A {
    public function hello(){
        return 'Hello';
    }
    public function me(){
        return 'A';
    }
}
trait B {
    public function me(){
        return 'B';
    }
}
class C {
    use A, B { B::me insteadof A };
}
```

Alias

- Oltre alla risoluzione della collisione dei nomi, è possibile anche variare il nome, con un **alias**, e la tipologia di una funzione definita in un Trait

Esempio

```
trait A {
    public function hello() {
        return 'Hello';
    }
}
class B {
    use A { hello as protected; }
}
class C {
    use A { hello as private myPrivateHello; }
}
```

Classi anonime

- Il PHP offre la possibilità di creare delle classi **anonime**
- La fase della definizione e dell'istanza di una classe anonima avvengono in contemporanea; non è quindi possibile istanziare più di una copia della stessa classe anonima

Esempio

```
interface LoggerInterface {
    public function log(string $msg);
}
class Application {
    protected $logger;
    public function getLogger(): LoggerInterface {
        return $this->logger;
    }
    public function setLogger(LoggerInterface $logger) {
        $this->logger = $logger;
    }
}
```

Esempio (2)

```php
$app = new Application;
$app->setLogger(new class implements
LoggerInterface {
    public function log(string $msg){
        echo $msg;
    }
});
var_dump($app->getLogger());
// object(class@anonymous)#2 (0) {}
```

Riferimenti

- Marco Pivetta, When to declare classes final
- Joseph Benharosh, The essentials of Object Oriented PHP
- Marcel dos Santos, Learning OOP in PHP
- Junade Ali, Object oriented PHP
- Brandon Savage, Mastering Object Oriented PHP

Grazie dell’attenzione!

Per informazioni:
enrico.zimuel@its-ictpiemonte.it

POR Piemonte
FSE 2014-2020