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

Programmazione a oggetti
(II parte)

Corso Backend System Integrator

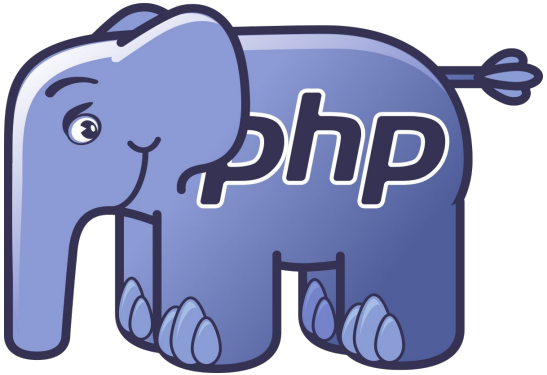
Modulo **Programmazione PHP**

Docente: Dott. Enrico Zimuel

in collaborazione con:
MINISTERO DELL'ISTRUZIONE, DELL'UNIVERSITÀ E DELLA RICERCA
UNIONE EUROPEA
REGIONE PIEMONTE
per una crescita intelligente, sostenibile ed inclusiva
www.regione.piemonte.it/europa2020
INIZIATIVA CO-FINANZIATA CON FSE

Programma

- Include
- Require
- Namespace
- Use & Alias
- Ereditarietà
- Classi astratte



Include

- **Include** è un'istruzione che consente di caricare (includere) un file PHP
- Tipicamente un'applicazione **è suddivisa in più file** (un file per classe)
- Per poter utilizzare una classe memorizzata in un altro file è necessario caricare quel file (includerlo)

Include (2)

- E’ possibile includere un file tramite il percorso del file
- Se si utilizza soltanto il nome del file, senza percorso, il PHP cercherà il file nella cartella attuale o in quella di default (specificata nella direttiva del php.ini include_path)
- Se il file esiste l’istruzione **include** restituirà *true*, altrimenti restituirà *false* generando anche un WARNING

Esempio

- Quando un file viene incluso, il codice eredita lo spazio delle variabili (*scope*) a partire dalla linea di inclusione

```php
// vars.php

$color = 'green';
$fruit = 'apple';
```

```php
//test.php

echo "A $color $fruit"; // A

include 'vars.php';

echo "A $color $fruit"; // A green apple
```

Esempio con classi

```php
// Foo.php

class Foo
{
    // ...
}
```

```php
// Bar.php
include 'Foo.php';

class Bar
{
    // ...
}
$foo = new Foo();
$bar = new Bar();
```

Esempio: file not found

```php
$file = 'dasdasad.php';
$result = include $file; // WARNING
if (false === $result) {
   printf("Il file %s non esiste", $file);
   exit(1);
}
```

Include e Return

- **Include** può essere utilizzato per assegnare il ritorno di una variabile
- Il file incluso deve terminare con un'istruzione **return**
- Questa tecnica è utilizzare per gestire file di configurazione in PHP

Esempio

```php
// db.php
return [
    'db' => [
        'username' => 'root',
        'password' => '1234567890',
        'host'     => 'localhost'
    ]
];
```

```php
$config = include 'db.php';
var_dump($config);

// array(1) {
//   'db' => array(3) {
//     'username' => string(4) "root"
//     'password' => string(10) "1234567890"
//     'host'     => string(9) "localhost"
//   }
// }
```

Require

- L’istruzione **require** può essere utilizzata al posto di **include**
- L’unica differenza è che **require** genera un **FATAL ERROR** se il file non esiste (mentre require soltanto un WARNING)

Include_once, require_once

- **Include_once** include un file solo se non è stato già caricato in precedenza
- **Require_once** fa la stessa cosa di **include_once** e se il file non esiste generare un FATAL ERROR

Esempio

```php
// a.php
$a = 2;
```

```php
$a = 1;
include 'a.php';
echo $a; // 2

$a = 1;
include 'a.php';
echo $a; // 2
```

```php
$a = 1;
include_once 'a.php';
echo $a; // 2

$a = 1;
include_once 'a.php';
echo $a; // 1
```

Esercizio

- Utilizzare la classe **Studente** (su github Corso_PHP_ITS) per creare un oggetto con i vostri dati, aggiungendo il corso **“PHP Programming”**
- Eseguire il var_dump() dell’oggetto. Qual’è il risultato?

Constructor property promotion

```php
class Customer
{
    public string $name;
    public string $email;
    public DateTimeInterface $birth_date;

    public function __construct(
        string $name,
        string $email,
        DateTimeInterface $birth_date
    ) {
        $this->name = $name;
        $this->email = $email;
        $this->birth_date = $birth_date;
    }
}
```

```php
class Customer
{
    public function __construct(
        public string $name,
        public string $email,
        public DateTimeInterface $birth_date,
    ) {}
}
```

Namespace

- I **namespace** sono dei nomi utilizzati per identificare univocamente classi o funzioni
- Si possono creare delle gerarchie di namespace separando ogni livello con un **backslash** (\), es. **Foo\Bar\Baz**

Sintassi

- Per definire un namespace si utilizza la seguente sintassi:

```
namespace <nome>;
```

dove <nome> è il nome del namespace da utilizzare

Esempio: classe

```php
namespace ITSPiemonte;

class Studenti
{
    // ...
}

$a = new ITSPiemonte\Studenti();
```

- La classe **Studenti** fa parte del namespace **ITSPiemonte**

Esempio: funzione

```php
namespace ITSPiemonte {
    function hello(string $name): void
    {
        printf("Hello %s!\n", $name);
    }
};

namespace {
    ITSPiemonte\hello('Alberto'); // Hello Alberto!
}
```

Esempio: un namespace per file

```php
// function_hello.php
namespace ITSPiemonte;

function hello(string $name): void
{
    printf("Hello %s!\n", $name);
}
```

```php
require 'function_hello.php';

ITSPiemonte\hello('Alberto');
// Hello Alberto!
```

NOTA: se un file contiene un solo namespace si possono omettere le parentesi graffe { … }

Use

- E’ possibile definire l’insieme delle classi (o funzioni) da utilizzare tramite l’istruzione **use**

Esempio

```php
require 'Studente.php';

use ITSPiemonte\Studente;

$alunno = new Studente();
```

```php
require 'function_hello.php';

use function ITSPiemonte\hello;

hello('Alberto'); // Hello Alberto!
```

Alias

- E’ possibile utilizzare un **alias** per il nome di una classe o una funzione
- L’alias è un numero alternativo per identificare una classe o funzione

Esempio

```php
require 'Studente.php';

use ITSPiemonte\Studente as Alunno;

$alunno = new Alunno();
```

Namespace: buone pratiche

- Un solo **namespace** per ogni file
- Una sola **classe** per ogni file
- Utilizzo di **use** in ogni file, sia per classi che per funzioni

Ereditarietà

- Nella programmazione OOP è possibile **ereditare** le proprietà di una classe
- In PHP è possibile **estendere** una classe con l'istruzione **extends**, ereditandone le proprietà
- L'ereditarietà consente di creare delle **gerarchie di classi**, con un meccanismo di padre/figlio

Extends

- Sintassi dell'istruzione **extends**:

```
class <classe-figlio> extends <classe-padre>
{
    // ...
}
```

Esempio: classe User

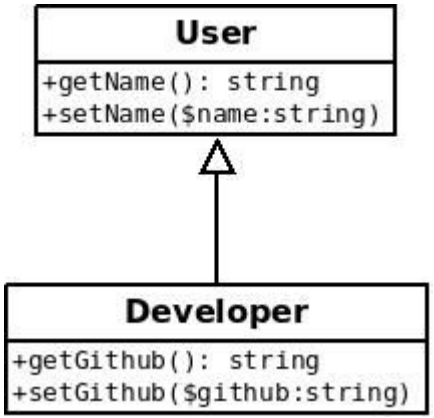
```php
class User
{
    protected $name = '';
    public function __construct(string $name)
    {
        $this->name = $name;
    }
    public function setName(string $name)
    {
        $this->name = $name;
    }
    public function getName(): string
    {
        return $this->name;
    }
}
```

Esempio: classe Developer

```php
class Developer extends User
{
    protected $github = '';
    public function setGithub(string $github)
    {
        $this->github = $github;
    }
    public function getGithub(): string
    {
        return $this->github;
    }
}
```

Diagramma delle classi

- Diagramma delle classi **User** e **Developer**



Sovrascrittura

- E’ possibile sovrascrivere il comportamento di una o più funzioni della classe padre
- In una classe che ne estende un’altra è possibile richiamare la funzione della classe padre con l’istruzione **parent::**

Esempio

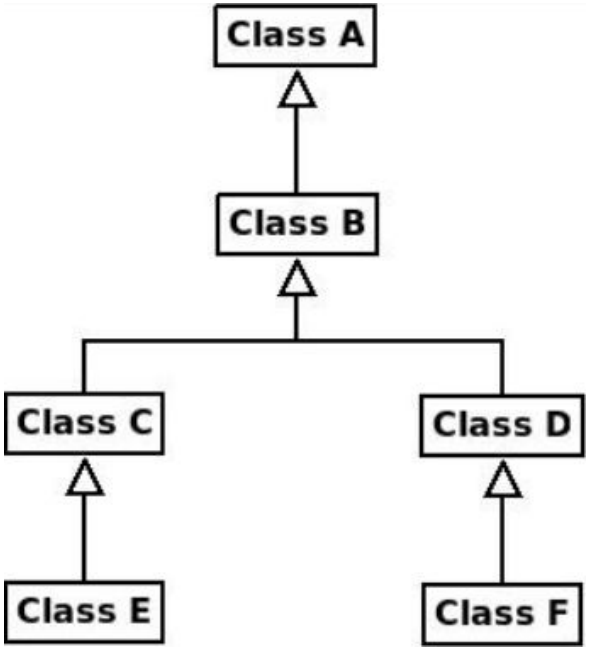
```php
class Developer extends User
{
    // ...
    public function __construct(string $name, string $github) {
        $this->github = $github;
        parent::__construct($name);
    }
    // ...
}
```

Ereditarietà multipla

- Il PHP non supporta l'ereditarietà multipla, ossia non è possibile estendere una classe partendo da più classi
- E' però possibile estendere una classe che ne estende un'altra è così via

Esempio

```
class A {};
class B extends A {};
class C extends B {};
class D extends B {};
class E extends C {};
class F extends D {};
```



Classi astratte

- Una classe astratta è una tipologia di classe che **non può essere istanziata direttamente** ma che funge da classe base per altre classi
- Con una classe astratta è possibile **condividere il codice** tra classi che sono in relazione tra di loro

Esempio

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

```php
class User extends AbstractUser
{
}
class Developer extends AbstractUser
{
    // ...
}
```

Esercizio (da consegnare)

- Creare una classe astratta **Studente** per la gestione di tutti gli studenti dell'ITS Piemonte
- Creare una classe **ICT** che estenda la classe Studente per gestire le specificità dei corsi di Informatica (es. Backend System Integrator)

Grazie dell’attenzione!

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

POR Piemonte
FSE 2014-2020