Object Oriented Programming

Prof. Ing. Loris Penserini

elpense@gmail.com

EREDITARIETA'

Ereditarietà

In PHP (come in Java) **non** è prevista l'ereditarietà multipla come invece è possibile nel C++, per cui in PHP una classe può al più ereditare da una sola altra classe.

In PHP, dalla versione 5.4, sono state aggiunte delle funzioni per ovviare a questa limitazione (i trait) che vedremo più avanti**.**

Tuttavia, per bravi programmatori OOP, l'ereditarietà singola non è considerata una limitazione ma un vantaggio per progettare SW efficiente ed modulare.

La classe Persona

```php
class Persona {
    //Proprietà
    public $nome = "";
    public $cognome = "";
    public $eta = "";
    public $interessi = "";
    public $saluto = "";

    //costruttore
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;
        $this->saluto = "Buongiorno sono ".$nome." ".$cognome;
    }

    //metodo che restituisce la pagina di saluto
    public function getPagBenvenuto() {
        $this->saluto = "Buongiorno sono ".$this->nome." ". $this->cognome;
        return $this->saluto;
    }
}
```

La classe Studente

L'operatore «extends»

```php
class Studente extends Persona {
    public $ind_studio = "";

    //metodo per inserire info specifiche per lo studente
    public function setIndStudio($ind_studio) {
        $this->ind_studio = $ind_studio;
    }

    public function getPagBenvenutoStud() {
        $saluto_persona = $this->getPagBenvenuto();
        return $saluto_persona.", e frequento ".$this->ind_studio;
    }
}
```

Le Istanze di classi sono Oggetti

```php
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <?php
        include 'Persona.php';
        //require_once 'Persona.php';
        include 'Studente.php';

        $nome = "Loris";
        $cognome = "Penserini";
        $eta = "50";
        $interessi = "DRONI";

        //CREO L'OGGETTO "Personal"
        $Personal = new Persona($nome, $cognome, $eta, $interessi);
        $Studente1 = new Studente($nome, $cognome, $eta, $interessi);
        $Studente1->setIndStudio("Sistemi Informativi Aziendali");

        echo "SALUTO DI PERSONA_1: <br>".$Personal->getPagBenvenuto();
        echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();

        ?>
    </body>
</html>
```

Costruttori di classe

Ogni classe dovrebbe avere il metodo costruttore, poiché definisce lo stato iniziale dell'oggetto associato alla classe. Cioè il metodo costruttore crea un punto di partenza nell'esecuzione del codice dell'oggetto.

Allora nella classe Studente da quale blocco di codice si inizia?

Project work 1

Riutilizzate il codice del progetto appena presentato. E con modifiche minimali, aggiungere la classe «Dirigente» (utilizzando come guida la classe Studente), poi dalla pagina index.php lanciare un'istanza e accodare a video il relativo saluto:

output

```
SALUTO DI PERSONA_1:
Buongiorno sono Loris Penserini

SALUTO DI STUDENTE_1:
Buongiorno sono Mario Rossi, e frequento Sistemi Informativi Aziendali

SALUTO DI DIRIGENTE_1:
Buongiorno sono Giovanna Rossini, e dirigo la scuola IIS POLO3 FANO
```

Project work (Dirigente.php)

La classe Dirigente (come Studente) eredita dalla classe Persona, ovviamente al contrario di Studente i metodi di Dirigente si specializzano per questo ruolo.

```php
class Dirigente extends Persona {
    public $scuola = "";

    //metodo per inserire info specifiche per lo studente
    public function setScuola($scuola) {
        $this->scuola = $scuola;
    }

    public function getPagBenvenutoDir() {
        $saluto_persona = $this->getPagBenvenuto();
        return $saluto_persona.", e dirigo la scuola ".$this->scuola;
    }
}
```

Project work (index.php)

```php
include 'Persona.php';
//require_once 'Persona.php';
include 'Studente.php';
include 'Dirigente.php';

//Simula una lettura dati che può avvenire tramite FORM HTML,
//tramite DB, tramite lettura file, ecc.
$nome1 = "Loris";
$cognome1 = "Penserini";
$eta1 = "50";
$interessi1 = "DRONI";

$nome2 = "Mario";
$cognome2 = "Rossi";
$eta2 = "40";
$interessi2 = "Pesca";

$nome3 = "Giovanna";
$cognome3 = "Rossini";
$eta3 = "40";
$interessi3 = "Opera";

//CREO GLI OGGETTI "Persona1", "Studente1" e "Dirigente1"
$Persona1 = new Persona($nome1, $cognome1, $eta1, $interessi1);
$Studente1 = new Studente($nome2, $cognome2, $eta2, $interessi2);
$Studente1->setIndStudio("Sistemi Informativi Aziendali");
$Dirigente1 = new Dirigente($nome3, $cognome3, $eta3, $interessi3);
$Dirigente1->setScuola("IIS POLO3 FANO");

echo "SALUTO DI PERSONA_1: <br>".$Persona1->getPagBenvenuto();
echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
echo "<br><br>SALUTO DI DIRIGENTE_1: <br>".$Dirigente1->getPagBenvenutoDir();
```

Sono state cerchiate le modifiche necessarie al file «index.php»

Project work 2

Riutilizzate il codice del progetto appena presentato. Inserire nella classe «Studente» i metodi necessari per leggere le proprietà: nome, cognome, età e indirizzo di studio.

Interfacce

Una interfaccia rappresenta una «definizione di progetto» per una classe, cioè ne definisce i metodi fondamentali che la classe corrispondente dovrà possedere obbligatoriamente.

Per cui a **design-time** si forniscono allo sviluppatore delle linee guida, utili a sviluppare classi specifiche senza dover avere la visione completa del progetto d'insieme.

Regole principali:

- I metodi dichiarati nelle interfacce devono sempre essere **public**, e perciò anche quelli implementati all'interno della classe.
- Una classe può implementare più interfacce contemporaneamente, utilizzando come separatore la virgola.
- Una interfaccia può sostenere **l'ereditarietà multipla** (extends).
- Le interfacce non possono contenere proprietà, ma **solo metodi**.
- Nell'interfaccia **non c'è costruttore**.

Interfacce: esempio

Prendiamo in considerazione l'esempio precedente e creiamo una interfaccia per il saluto…

```
interface Saluto {
    public function getPagBenvenuto();
}
```

«Implementare» una Interfaccia

```
class Persona implements Saluto{
    //Proprietà
    public $nome = "";
    public $cognome = "";
    public $eta = "";
    public $interessi = "";
    public $saluto = "";

    //costruttore
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;
        $this->saluto = "Buongiorno sono ".$nome." ".$cognome;
    }

    //metodo che restituisce la pagina di saluto
    public function getPagBenvenuto() {
        $this->saluto = "Buongiorno sono ".$this->nome." ". $this->cognome;
        return $this->saluto;
    }
}
```

Project work 3

Nella classe «Persona» provate a commentare il metodo «getPagBenvenuto()»… cosa accade?

Classi Astratte

La classe astratta, è una classe in cui uno o più metodi non sono implementati. Per cui, da una classe astratta **non** si può creare un oggetto.

Similmente alle interfacce, servono a definire delle **linee guida per creare le sottoclassi** che ne ereditano le proprietà. Cioè una sottoclasse che eredita da una classe astratta, ne deve implementare i metodi astratti, se vuole istanziare oggetti.

Nel caso una sottoclasse, che eredita da una classe astratta, non implementa tutti i metodi di quest'ultima, deve essere anch'essa dichiarata astratta.

- Possono contenere istruzioni, proprietà e metodi come le normali classi
- Possono utilizzare i modificatori di visibilità come le normali classi
- Possono avere il costruttore
- Vale l'ereditarietà singola come per le classi

Classi Astratte: esempio

Utilizziamo come guida per la creazione della classe «Studente» una classe astratta…

```php
abstract class StudentePrototipo{
    public $ind_studio = "";

    /*
     * Metodi per inserire/leggere info specifiche per lo studente.
     * Questi metodi sono definiti per intero.
     */
    public function setIndStudio($ind_studio) {
        $this->ind_studio = $ind_studio;
    }

    public function getIndStudio(){
        return $this->ind_studio;
    }

    /*
     * Metoto non implementato, per cui dichiarato come "abstract"
     */
    public abstract function getPagBenvenutoStud():string;
}
```

Classi Astratte: esempio con «extends»

Implementando la classe «Studente» sfruttiamo le linee guida della classe astratta «StudentePrototipo»…

```php
class Studente extends StudentePrototipo {

    public $nome = "";
    public $cognome = "";
    public $eta = "";
    public $interessi = "";
    public $saluto = "";

    //costruttore
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;
        $this->saluto = "Buongiorno sono ".$nome." ".$cognome;
    }

    public function getPagBenvenutoStud():string{
        return $this->saluto.", uno studente, e frequento ".parent::getIndStudio();
    }
}
```

Classi Astratte: utilizzo

```php
<?php
    include 'StudentePrototipo.php';
    include 'Studente.php';

    $nome = "Loris";
    $cognome = "Penserini";
    $eta = "50";
    $interessi = "DRONI";

    /*
     * CREO L'OGGETTO "Studente1"
     */
    $Studente1 = new Studente($nome, $cognome, $eta, $interessi);
    $Studente1->setIndStudio("Sistemi Informativi Aziendali");

    echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
?>
```

Project work 4

Nella classe astratta «StudentePrototipo» provate a cambiare il modificatore di visibilità del metodo «getIndStudio()» dal valore attuale «public» al valore «protected»…

cosa accade?

INCAPSULAMENTO

INCAPSULAMENTO

E' una proprietà della OOP che facilita a seguire la filosofia dell'ingegneria del software dell'usabilità e della modularità delle applicazioni.

In pratica, nella OOP con l'utilizzo dell'incapsulamento, si 'aggregano' all'interno di un oggetto i dati e le azioni che lo interessano, che diventano quindi elementi **visibili e gestiti** solo dall'oggetto stesso (o da una sua classe), mentre si rendono **pubblici** solo metodi (o attributi) che servono all'oggetto per poter essere riutilizzato in altri contesti applicativi.

Quindi, si parla spesso di limitare l'accesso diretto agli elementi di un oggetto, ovvero di occultamento delle informazioni (*information hiding*).

Modificatori di accesso/visibilità

Per cui in tutti i linguaggi OOP troverete questi operatori:

```
public $nome; // visibile ovunque nello script
protected $email; // visibile nella superclasse e nella sottoclasse
private $password; // Visibile solo nella classe
```

I modificatori sono applicabili pure ai metodi/funzioni.

Esempio di «incapsulamento»

Riprendiamo l'esempio precedente ma operiamo a livello di «design-time» un cambio di strategia di utilizzo della stessa applicazione: gli attributi della classe padre non devono essere accessibili dall'esterno, e solo la classe figlio può invocarne il metodo del saluto…

```php
<?php
include 'Persona.php';
//require_once 'Persona.php';
include 'Studente.php';

$nome = "Loris";
$cognome = "Penserini";
$eta = "50";
$interessi = "DRONI";

//CREO L'OGGETTO "Personal"
//$Personal = new Persona($nome, $cognome, $eta, $interessi);
$Studente1 = new Studente($nome, $cognome, $eta, $interessi);
$Studente1->setIndStudio("Sistemi Informativi Aziendali");

//Commentato poiché utilizzando "protected" su getPagBenvenuto() si
//considera tale metodo di Persona accessibile solo dalle sue sottoclassi.
//echo "SALUTO DI PERSONA_1: <br>".$Personal->getPagBenvenuto();
echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
```

Esempio di «incapsulamento»

Gli attributi sono visibili solo da dentro la classe: «**private**»

Il metodo getPagBenvenuto() diventa «**protected**»

```php
class Persona {
    //Proprietà di INCAPSULAMENTO: gli attributi della classe sono
    //visibili/accessibili solo da dentro la classe stessa.
    private $nome = "";
    private $cognome = "";
    private $eta = "";
    private $interessi = "";
    private $saluto = "";

    //costruttore
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;
        $this->saluto = "Buongiorno sono ".$nome." ".$cognome;
    }

    /* metodo che restituisce la pagina di saluto
     * Notare la forma di INCAPSULAMENTO adottata: il metodo sarà accessibile solo
     * dalle classi figlie (e ovviamente dalla classe padre)
    */
    protected function getPagBenvenuto() {
        $this->saluto = "Buongiorno sono ".$this->nome." ". $this->cognome;
        return $this->saluto;
    }
}
?>
```

Esempio di «incapsulamento»

Gli attributi sono visibili solo da dentro la classe: «**private**»

Metodi accessibili dall'esterno: «**public**»

```php
class Studente extends Persona {
    private $ind_studio = "";

    /*
     * Questa classe NON ha il costruttore, per cui usa quello della superclasse
     * Se aggiungete questo costruttore che segue, si farà overriding...
     *
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = "xxx";
        $this->cognome = "yyy";
        $this->eta = "aaaa";
        $this->interessi = "zzz";
    }*/

    //metodo per inserire info specifiche per lo studente
    public function setIndStudio($ind_studio) {
        $this->ind_studio = $ind_studio;
    }

    public function getPagBenvenutoStud() {
        $saluto_persona = $this->getPagBenvenuto();
        return $saluto_persona.", e frequento ".$this->ind_studio;
    }
}
?>
```

Project work

Riutilizzate il codice del progetto appena presentato. Applicate le modifiche alla pagina «index.php» affinché si istanzi la classe «Persona» e si richiami il metodo del saluto…

Deve comparirvi il seguente errore…

Fatal error: Uncaught Error: Call to protected method Persona::getPagBenvenuto() from global scope in C:\xampp\htdocs\OOP_Incaps\index.php:29 Stack trace: #0 {main} thrown in **C:\xampp\htdocs\OOP_Incaps\index.php** on line **29**

GRAZIE!