Object Oriented Programming

Prof. Ing. Loris Penserini, PhD

https://orcid.org/0009-0008-6157-0396

Informatica: Modellare il Mondo Reale

L'Informatica (o Computer Science) è la scienza che studia la gestione dell'informazione tramite procedure automatizzate. Questo è realizzabile attraverso: una consolidata teoria dell'informazione, linguaggi formali di rappresentazione, e sistemi elettronici per l'esecuzione automatica.

Attraverso specifici linguaggi di programmazione, si possono scrivere algoritmi (sequenze di istruzioni) che un calcolatore elettronico è in grado di elaborare in modo automatico al fine di fornire delle informazioni utili all'utente partendo da dati iniziali.

L'evoluzione delle tecnologie digitali nei settori della Robotica e dell'AI, spingono i ricercatori a sviluppare nuovi paradigmi di programmazione…

POLIMORFISMO e COSTRUTTORI MULTIPLI

POLIMORFISMO

Assieme alle proprietà di Ereditarietà e all'Incapsulamento facilita il programmatore a seguire la filosofia dell'ingegneria del software dell'usabilità e della modularità delle applicazioni.

In sintesi, nella OOP, il Polimorfismo è la proprietà di una classe di poter generare diverse specializzazioni, semplicemente fornendo alle classi figlie la possibilità di svolgere:

Overriding: **riscrivere uno stesso metodo della classe padre.**

Overloading: **invocare uno stesso metodo con tipologia e parametri diversi.**

Esempio di Polimorfismo (index.php)

Vedremo tre aspetti: un metodo per realizzare **costruttori multipli** in PHP, **OVERRIDING** e **OVERLOADING**.

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

/* STUDENTE (maschio)
 */
$nome1 = "Loris";
$cognome1 = "Penserini";
$eta1 = "50";
$interessi1 = "DRONI";
//OVERLOADING di __construct() di Studente
$Studente1 = new Studente($nome1, $cognome1, $eta1, $interessi1);
$Studente1->setIndStudio("Sistemi Informativi Aziendali");

/* STUDENTESSA (femmina)
 */
$nome2 = "Maria";
$cognome2 = "Bianchi";
$eta2 = "25";
$interessi2 = "Pallavolo";
//OVERLOADING di __construct() di Studente
$Studentessa1 = new Studente($nome2, $cognome2, $eta2, $interessi2, "F");
$Studentessa1->setIndStudio("Sistemi Informativi Aziendali");

echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
echo "<br><br>SALUTO DI STUDENTESSA_1: <br>".$Studentessa1->getPagBenvenutoStud();
?>
```

output

SALUTO DI STUDENTE_1:
Buongiorno sono Loris Penserini, uno studente che frequenta Sistemi Informativi Aziendali

SALUTO DI STUDENTESSA_1:
Buongiorno sono Maria Bianchi, una studentessa che frequenta Sistemi Informativi Aziendali

Esempio di Polimorfismo (index.php)

Vedremo tre aspetti: un metodo per realizzare **costruttori multipli** in PHP, **OVERRIDING** e **OVERLOADING**.

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

/* STUDENTE (maschio)
 */
$nome1 = "Loris";
$cognome1 = "Penserini";
$eta1 = "50";
$interessi1 = "DRONI";
//OVERLOADING di __construct() di Studente
$Studente1 = new Studente($nome1, $cognome1, $eta1, $interessi1);
$Studente1->setIndStudio("Sistemi Informativi Aziendali");

/* STUDENTESSA (femmina)
 */
$nome2 = "Maria";
$cognome2 = "Bianchi";
$eta2 = "25";
$interessi2 = "Pallavolo";
//OVERLOADING di __construct() di Studente
$Studentessa1 = new Studente($nome2, $cognome2, $eta2, $interessi2, "F");
$Studentessa1->setIndStudio("Sistemi Informativi Aziendali");

echo "<br><br>SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
echo "<br><br>SALUTO DI STUDENTESSA_1: <br>".$Studentessa1->getPagBenvenutoStud();
?>
```

4 param.

5 param.

Overloading di __construct() di Studente

Esempio di Polimorfismo (Persona.php)

```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 Polimorfismo (Studente.php)

```php
class Studente extends Persona {
    private $ind_studio = "";
    private $sesso = "M";
    private $nome = "";
    private $cognome = "";
    private $eta = "";
    private $interessi = "";
    //private $saluto = "";

    /* Questo costruttore riscrive quello di Persona (OVERRIDING)
     * e inoltre mostra come gestire costruttori multipli.
     * Si considera la forma seguente:
     * __construct($nome,$cognome,$eta,$interessi,$sesso)
     */
    public function __construct() {
        //si usano due funzioni speciali: "func_num_args()" e "func_get_arg()"
        if(4===func_num_args()){
            parent::setProfile(func_get_arg(0),func_get_arg(1),func_get_arg(2),func_get_arg(3));

        }elseif(5===func_num_args()){
            parent::setProfile(func_get_arg(0),func_get_arg(1),func_get_arg(2),func_get_arg(3));
            $this->sesso = func_get_arg(4);
        }
    }

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

    public function getPagBenvenutoStud() {
        $saluto_persona = parent::getPagBenvenuto();
        if($this->sesso === "M" || $this->sesso === "m" || $this->sesso === ""){
            return $saluto_persona.", uno studente che frequenta ".$this->ind_studio;

        }elseif($this->sesso === "F" || $this->sesso === "f"){
            return $saluto_persona.", una studentessa che frequenta ".$this->ind_studio;
        }
    }
}
```

Overriding di *__construct(...)* di Persona quando si richiama con 4 o 5 parametri

Costruttori multipli con l'ausilio delle funzioni speciali:

- **func_num_args()**
- **func_get_arg()**

Accedere alle proprietà della classe padre con: «**parent:: … **»

Project work 7

Dalla classe «Studente» fare OVERRIDING del metodo *getPagBenvenuto()* della classe «Persona». Sistemare il resto di conseguenza in modo da avere il minimo impatto di modifica e lo stesso output.

Possibile Soluzione del PW-7

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

    /* Questo costruttore riscrive quello di Persona (OVERRIDING)
     * e inoltre mostra come gestire costruttori multipli.
     * Si considera la forma seguente:
     * __construct($nome,$cognome,$eta,$interessi,$sesso)
     */
    public function __construct() {
        //si usano due funzioni speciali: "func_num_args()" e "func_get_arg()"
        if(4===func_num_args()){
            parent::setProfile(func_get_arg(0),func_get_arg(1),func_get_arg(2),func_get_arg(3));

        }elseif(5===func_num_args()){
            parent::setProfile(func_get_arg(0),func_get_arg(1),func_get_arg(2),func_get_arg(3));
            $this->sesso = func_get_arg(4);
        }
    }

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

    /*
     * Il metodo "public function getPagBenvenutoStud()" è stato sostituito con
     * "getPagBenvenuto()" che fa OVERRIDING dell'omonimo nella classe Persona
     */
    public function getPagBenvenuto() {
        $saluto_persona = parent::getPagBenvenuto();
        if($this->sesso === "M" || $this->sesso === "m" || $this->sesso === ""){
            return $saluto_persona.", uno studente che frequenta ".$this->ind_studio;

        }elseif($this->sesso === "F" || $this->sesso === "f"){
            return $saluto_persona.", una studentessa che frequenta ".$this->ind_studio;
        }
    }
}
```

Si riscrive la logica del metodo della classe padre, «Persona» (**Overriding**)

Proprietà e metodi STATICI

In OOP le proprietà e i metodi di una classe, quando si crea l'istanza dell'oggetto, diventano dei riferimenti in memoria. In particolare, praticare più istanze di una stessa classe significa creare altrettanti oggetti (processi in memoria RAM) ognuno allocato in spazi diversi della memoria.

Per avere proprietà e/o metodi di una classe che rimangono gli stessi (stesso indirizzo di memoria) per ogni istanza della classe stessa, dobbiamo definirli STATICI.

In particolare, creata una istanza in una classe, possiamo accedere alle proprietà e metodi dell'oggetto creato con l'operatore «**->**», poiché tutte le proprietà/metodi della classe fanno parte dello specifico oggetto in memoria.

Mentre, **se nella classe abbiamo una proprietà STATICA, questa in fase di implementazione dell'oggetto non farà parte delle sue proprietà, per cui è concettualmente sbagliato usare l'operatore «->». Cioè, la proprietà statica, definita a livello di classe, avrà lo stesso riferimento per tutte le istanze (oggetti) della classe.**

Accedere agli elementi di un Oggetto

- Per accedere allo spazio riservato delle variabili e/o metodi statici di una classe si usa:

  self::$variabile;** **parent::variabile();** **MyClasse::$variabile;

  self::metodo(); **parent::metodo();** **MyClasse::metodo();**

- Per accedere, dall'esterno, agli elementi (non statici) di un oggetto:

  $nome_oggetto -> metodo();

- Per accedere, dall'interno, agli elementi (non statici) di un oggetto:

  $this -> metodo();

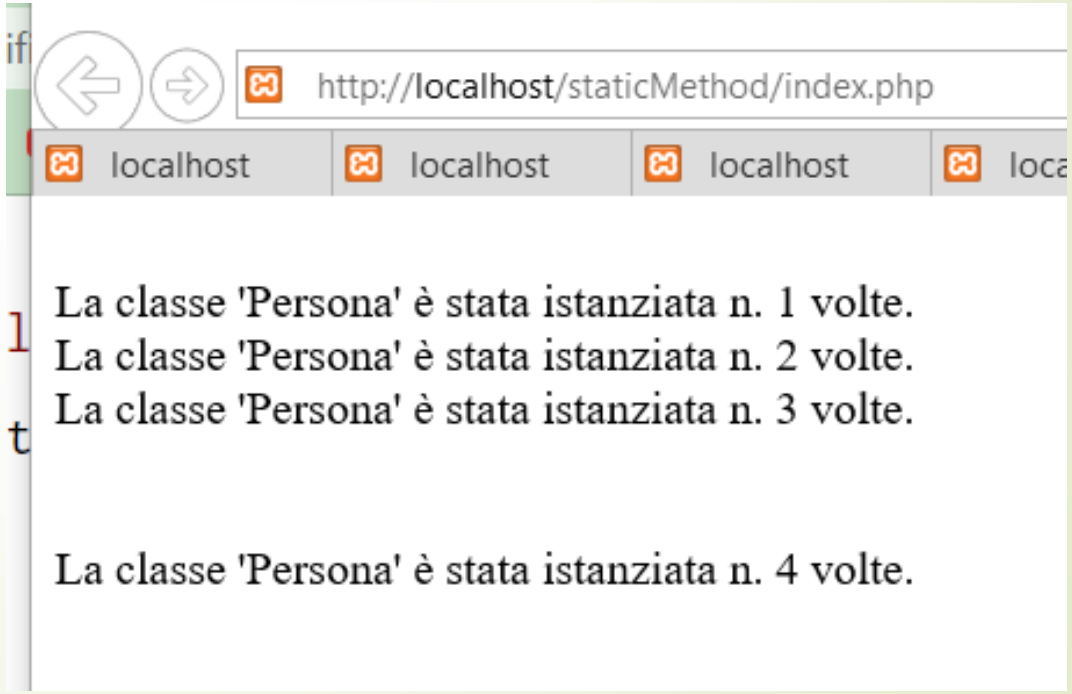
  $this -> variabile;

Project Work 8

Presa in esame la classe «Persona», si desidera realizzare un contatore che conti quante volte viene istanziata la classe, e per ogni volta stampi in output il valore.

Un tipo di problema come questo non può essere risolto se non si fa uso dell'operatore STATIC.

OUTPUT



Soluz. al PW-8: «Persona»

```php
class Persona {
    //Proprietà
    public $nome = "";
    public $cognome = "";
    public $eta = "";
    public $interessi = "";
    public $saluto = "";
    private static $numIstanze = 0;

    //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;

        self::$numIstanze++;
    }

    //metodo che restituisce la pagina di saluto
    public function getPagBenvenuto() {
        return $this->saluto;
    }

    //restituisce il numero di volte che è stato chiamato il costruttore
    public static function getInstancesOfPersona() {
        //return self::$numIstanze;
        return Persona::$numIstanze;
    }
}
```

$numIstanze come anche il metodo getInstanceOfPersona() sono gli stessi elementi per ogni istanza di oggetto.

Soluz. al PW-8: «index.php»

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

    $nome1 = "Loris";
    $cognome1 = "Penserini";
    $eta1 = "50";
    $interessi1 = "DRONI";

    $nome2 = "Mario";
    $cognome2 = "Rossi";
    $eta2 = "55";
    $interessi2 = "JAVA";

    $nome3 = "Anna";
    $cognome3 = "Fiore";
    $eta3 = "25";
    $interessi3 = "MUSICA";

    $nome4 = "John";
    $cognome4 = "Gate";
    $eta4 = "22";
    $interessi4 = "PHP";

    //CREO GLI OGGETTI "Personal", "Persona2" e "Persona3"
    $Personal = new Persona($nome1, $cognome1, $eta1, $interessi1);
    echo "<br> La classe 'Persona' è stata istanziata n. ".$Personal->getInstancesOfTotale()." volte.";

    $Persona2 = new Persona($nome2, $cognome2, $eta2, $interessi2);
    echo "<br> La classe 'Persona' è stata istanziata n. ".$Persona2->getInstancesOfPersona()." volte.";

    $Persona3 = new Persona($nome3, $cognome3, $eta3, $interessi3);
    echo "<br> La classe 'Persona' è stata istanziata n. ".$Persona3->getInstancesOfPersona()." volte.<br><br>";

    $Studente1 = new Studente($nome4, $cognome4, $eta4, $interessi4);
    echo "<br> La classe 'Persona' è stata istanziata n. ".Persona::getInstancesOfPersona()." volte.";
?>
```

Concettualmente non corretti… PERCHE'?

Soluz. al PW-8: «Studente»

Cosa accade rimuovendo i commenti del costruttore di «Studente» ?

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

    /*
     * Questa classe NON ha il costruttore, per cui usa quello della superclasse
     *
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;
    }
     */
    public function setIndStudio($ind_studio) {
        $this->ind_studio = $ind_studio;
    }

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

Project Work 9

Preso in esame il PW-9, si desidera realizzare un contatore che conti quante volte è istanziata direttamente ed indirettamente la classe «Persona». Inoltre, contare anche le istanze della classe «Studente».

PW-9: «Persona»

```php
class Persona {
    //Proprietà
    public $nome = "";
    public $cognome = "";
    public $eta = "";
    public $interessi = "";
    public $saluto = "";
    private static $numIstanze = 0;
    private static $numIstanzeIndiretta = 0;

    //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;

        self::$numIstanze++;
    }

    protected function setInstance(){
        self::$numIstanzeIndiretta++;
    }

    //metodo che restituisce la pagina di saluto
    public function getPagBenvenuto() {
         return $this->saluto;
    }

    //restituisce il numero di volte che è stato chiamato il costruttore
    public static function getInstancesOfPersona() {
        //return self::$numIstanze;
        return Persona::$numIstanze;
    }

    public static function getInstancesOfTotale() {
        //return self::$numIstanze;
        return Persona::$numIstanze + self::$numIstanzeIndiretta;
    }
}
```

PW-9: «Studente»

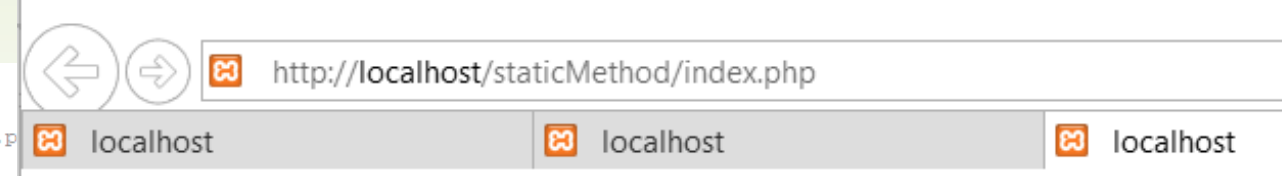
```php
class Studente extends Persona {
    public $ind_studio = "";
    public $nome = "PIPPO";
    private static $numIstanzeStud = 0;

    /*
     * Questa classe NON ha il costruttore, per cui usa quello della superclasse
     */
    public function __construct($nome,$cognome,$eta,$interessi) {
        //inizializzazione
        $this->nome = $nome;
        $this->cognome = $cognome;
        $this->eta = $eta;
        $this->interessi = $interessi;

        self::$numIstanzeStud++;
        Persona::setInstance();
    }

    public static function getInstancesOfStudente() {
        //return self::$numIstanze;
        return Studente::$numIstanzeStud;
    }

    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;
    }
}
```

PW-9: «index.php»

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

    $nome1 = "Loris";
    $cognome1 = "Penserini";
    $eta1 = "50";
    $interessi1 = "DRONI";

    $nome2 = "Mario";
    $cognome2 = "Rossi";
    $eta2 = "55";
    $interessi2 = "JAVA";

    $nome3 = "Anna";
    $cognome3 = "Fiore";
    $eta3 = "25";
    $interessi3 = "MUSICA";

    $nome4 = "John";
    $cognome4 = "Gate";
    $eta4 = "22";
    $interessi4 = "PHP";

    //CREO GLI OGGETTI "Persona1", "Persona2" e "Persona3"
    $Persona1 = new Persona($nome1, $cognome1, $eta1, $interessi1);
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";

    $Persona2 = new Persona($nome2, $cognome2, $eta2, $interessi2);
    //echo "<br> La classe 'Persona' è stata istanziata n. ".$Persona2->getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";

    $Persona3 = new Persona($nome3, $cognome3, $eta3, $interessi3);
    //echo "<br> La classe 'Persona' è stata istanziata n. ".$Persona3->getInstancesOfPersona()." volte.<br><br>";
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";

    $Studente1 = new Studente($nome4, $cognome4, $eta4, $interessi4);
    echo "<br> La classe 'Studente' è stata istanziata n. ".Studente::getInstancesOfStudente()." volte.<br><br>";
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";
?>
```

PW-9: output

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

    $nome1 = "Loris";
    $cognome1 = "Penserini";
    $eta1 = "50";
    $interessi1 = "DRONI";

    $nome2 = "Mario";
    $cognome2 = "Rossi";
    $eta2 = "55";
    $interessi2 = "JAVA";

    $nome3 = "Anna";
    $cognome3 = "Fiore";
    $eta3 = "25";
    $interessi3 = "MUSICA";

    $nome4 = "John";
    $cognome4 = "Gate";
    $eta4 = "22";
    $interessi4 = "PHP";

    //CREO GLI OGGETTI "Perso
    $Persona1 = new Persona($nome1, $cognome1, $eta1, $interessi1);
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";

    $Persona2 = new Persona($nome2, $cognome2, $eta2, $interessi2);
    //echo "<br> La classe 'Persona' è stata istanziata n. ".$Persona2->getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";

    $Persona3 = new Persona($nome3, $cognome3, $eta3, $interessi3);
    //echo "<br> La classe 'Persona' è stata istanziata n. ".$Persona3->getInstancesOfPersona()." volte.<br><br>";
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";

    $Studente1 = new Studente($nome4, $cognome4, $eta4, $interessi4);
    echo "<br> La classe 'Studente' è stata istanziata n. ".Studente::getInstancesOfStudente()." volte.<br><br>";
    echo "<br> La classe 'Persona' ha un numero di oggetti istanziati pari a n. ".Persona::getInstancesOfPersona()." volte.";
    echo "<br> La classe 'Persona' è stata istanziata in totale n. ".Persona::getInstancesOfTotale()." volte.";
?>
```



La classe 'Persona' ha un numero di oggetti istanziati pari a n. 1 volte.
La classe 'Persona' è stata istanziata in totale n. 1 volte.
La classe 'Persona' ha un numero di oggetti istanziati pari a n. 2 volte.
La classe 'Persona' è stata istanziata in totale n. 2 volte.
La classe 'Persona' ha un numero di oggetti istanziati pari a n. 3 volte.
La classe 'Persona' è stata istanziata in totale n. 3 volte.
La classe 'Studente' è stata istanziata n. 1 volte.

La classe 'Persona' ha un numero di oggetti istanziati pari a n. 3 volte.
La classe 'Persona' è stata istanziata in totale n. 4 volte.

TRAIT

TRAIT per superare limitazioni

I TRAIT sono script (o pezzi di codice) esterni alla classe che ne vuole fare uso.

Il loro utilizzo snellisce la duplicazione del codice e consente al programmatore di ovviare all'operatore «extend» per poter utilizzare metodi e/o variabili di altre classi. La differenza tra un TRAIT e l'ereditarietà multipla è che il TRAIT non viene ereditato ma incluso.

Tuttavia, un loro eccessivo utilizzo porta ad un codice poco chiaro e comprensibile.

Come è stato già detto, il PHP (come in Java) non prevede l'ereditarietà multipla. Quindi, il meccanismo del TRAIT consente di superare questa naturale limitazione.

Precedenza tra metodi con lo stesso nome

Consideriamo casi di trait e classi in cui si abbiano metodi con le stesso nome, per cui si risolve come segue:

- I metodi della classe sovrascrivono i metodi del trait
- I metodi ereditati da una classe vengono sovrascritti (sono sottoposti ad override) dai metodi inseriti da un trait

Quando una classe fa uso di un TRAIT con il comando «use» ne eredita tutti i metodi e variabili.

Esempio con TRAIT

TRAIT: **Gender.php**

```php
trait gender {

    public function getPagGender($sesso) {
        if($sesso === "M" || $sesso === "m" || $sesso === ""){
            return "Sono uno studente";

        }elseif($sesso === "F" || $sesso === "f"){
            return "Sono una studentessa";
        }
    }
}
```

Esempio con TRAIT (Studente.php)

```php
include 'gender.php';
include 'miPresento.php';

class Studente extends Persona {
    use gender,miPresento; //La classe Studente acquisisce le proprietà di due TRAIT
    private $ind_studio = "",$sesso = "M",$nome = "",$cognome = "",$eta = "",$interessi = "";

    /* Questo costruttore riscrive quello di Persona (OVERRIDING)
     * e inoltre mostra come gestire costruttori multipli.
     * Si considera la forma seguente:
     * __construct($nome,$cognome,$eta,$interessi,$sesso)
     */
    public function __construct() {
        //si usano due funzioni speciali: "func_num_args()" e "func_get_arg()"
        if(4===func_num_args()){
            parent::setProfile(func_get_arg(0),func_get_arg(1),func_get_arg(2),func_get_arg(3));

        }elseif(5===func_num_args()){
            parent::setProfile(func_get_arg(0),func_get_arg(1),func_get_arg(2),func_get_arg(3));
            $this->sesso = func_get_arg(4);
        }
    }
    //metodo per inserire info specifiche per lo studente
    public function setIndStudio($ind_studio) {
        $this->ind_studio = $ind_studio;
    }
    //metodo per estrarre l'indirizzo di studio dello studente
    public function getIndStudio() {
        return $this->ind_studio;
    }

    public function getPagBenvenutoStud() {
        $saluto_persona = parent::getPagBenvenuto();
        if($this->sesso === "M" || $this->sesso === "m" || $this->sesso === ""){
            return $saluto_persona.", uno studente che frequenta ".$this->ind_studio;

        }elseif($this->sesso === "F" || $this->sesso === "f"){
            return $saluto_persona.", una studentessa che frequenta ".$this->ind_studio;
        }
    }
}
```

Studente.php usa due TRAIT, è come se ne ereditasse le proprietà.

Esempio con TRAIT (Persona.php)

```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;
    }

    public function setProfile($nome,$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;
    }
}
```

Persona.php rimane invariata

Esempio con TRAIT (index.php)

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

/* STUDENTE (maschio)
 */
$nome1 = "Loris";
$cognome1 = "Penserini";
$eta1 = "50";
$interessi1 = "DRONI";
//OVERLOADING di __construct() di Studente
$Studente1 = new Studente($nome1, $cognome1, $eta1, $interessi1);
$Studente1->setIndStudio("Informatica Avanzata");

/* STUDENTESSA (femmina)
 */
$nome2 = "Maria";
$cognome2 = "Bianchi";
$eta2 = "25";
$interessi2 = "Pallavolo";
//OVERLOADING di __construct() di Studente
$Studentessa1 = new Studente($nome2, $cognome2, $eta2, $interessi2, "F");
$Studentessa1->setIndStudio("Sistemi Informativi Aziendali");

echo "Sfruttiamo il TRAIT 'gender.php': <br>";
echo $Studente1->getPagGender("F")."<br>";
echo $Studente1->getPagGender("m")."<br>";

//echo $Studente1->miPresento("m")."<br>";
//echo $Studentessa1->miPresento("F")."<br>";
echo "<br><br>Utilizzo della logica precedente: <br>";
echo "SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
echo "<br><br>SALUTO DI STUDENTESSA_1: <br>".$Studentessa1->getPagBenvenutoStud();
?>
```

Esempio con TRAIT (index.php)

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

/* STUDENTE (maschio)
 */
$nome1 = "Loris";
$cognome1 = "Penserini";
$eta1 = "50";
$interessi1 = "DRONI";
//OVERLOADING di __consti
$Studente1 = new Studente
$Studente1->setIndStudio

/* STUDENTESSA (femmina)
 */
$nome2 = "Maria";
$cognome2 = "Bianchi";
$eta2 = "25";
$interessi2 = "Pallavolo"
//OVERLOADING di __construct() di Studente
$Studentessa1 = new Studente($nome2, $cognome2, $eta2, $interessi2, "F");
$Studentessa1->setIndStudio("Sistemi Informativi Aziendali");

echo "Sfruttiamo il TRAIT 'gender.php': <br>";
echo $Studente1->getPagGender("F")."<br>";
echo $Studente1->getPagGender("m")."<br>";

//echo $Studente1->miPresento("m")."<br>";
//echo $Studentessa1->miPresento("F")."<br>";
echo "<br><br>Utilizzo della logica precedente: <br>";
echo "SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
echo "<br><br>SALUTO DI STUDENTESSA_1: <br>".$Studentessa1->getPagBenvenutoStud();
?>
```

output

Sfruttiamo il TRAIT 'gender.php':
sono una studentessa
sono uno studente

Utilizzo della logica precedente:
SALUTO DI STUDENTE_1:
Buongiorno sono Loris Penserini, uno studente che frequenta Informatica Avanzata

SALUTO DI STUDENTESSA_1:
Buongiorno sono Maria Bianchi, una studentessa che frequenta Sistemi Informativi Aziendali

Project Work 10

Nel prossimo progetto il programmatore desidera sfruttare due TRAIT: **gender.php** (quello precedente) e **miPresento.php** (da realizzare), che senza perturbare (modificare) i metodi e proprietà delle classi **Studente** e **Persona** producano lo stesso effetto desiderato, per esempio l'output seguente:

Output desiderato

Sfruttiamo i TRAIT 'gender.php' e 'miPresento.php':

Buongiorno sono Loris Penserini, sono uno studente, e frequento l'indirizzo di studi di Informatica Avanzata

Buongiorno sono Maria Bianchi, sono una studentessa, e frequento l'indirizzo di studi di Sistemi Informativi Aziendali

Alcune tracce di una possibile soluzione sono state lasciate nel file «index.php» precedente …

Possibile soluzione con TRAIT

TRAIT: **Gender.php**

```php
trait gender {

    public function getPagGender($sesso) {
        if($sesso === "M" || $sesso === "m" || $sesso === ""){
            return "Sono uno studente";

        }elseif($sesso === "F" || $sesso === "f"){
            return "Sono una studentessa";
        }
    }
}
```

TRAIT: **miPresento.php**

```php
trait miPresento {
    //put your code here
    public function miPresento($sesso){
        $saluto = parent::getPagBenvenuto(); //Richiama un metodo di Persona
        $ruolo = $this->getPagGender($sesso); //Richiama un metodo in altro TRAIT
        $indirizzo = $this->getIndStudio(); //Richiama un metodo della classe che lo usa

        return $saluto.", ".$ruolo.", e frequento l'indirizzo di studi di ".$indirizzo."<br>";
    }
}
```

Possibile soluzione con TRAIT (index.php)

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

/* STUDENTE (maschio)
 */
$nome1 = "Loris";
$cognome1 = "Penserini";
$eta1 = "50";
$interessi1 = "DRONI";
//OVERLOADING di __construct() di Studente
$Studente1 = new Studente($nome1, $cognome1, $eta1, $interessi1);
$Studente1->setIndStudio("Informatica Avanzata");

/* STUDENTESSA (femmina)
 */
$nome2 = "Maria";
$cognome2 = "Bianchi";
$eta2 = "25";
$interessi2 = "Pallavolo";
//OVERLOADING di __construct() di Studente
$Studentessa1 = new Studente($nome2, $cognome2, $eta2, $interessi2, "F");
$Studentessa1->setIndStudio("Sistemi Informativi Aziendali");
/*
echo "Sfruttiamo il TRAIT 'gender.php': <br>";
echo $Studente1->getPagGender("F")."<br>";
echo $Studente1->getPagGender("m")."<br>";

echo "<br><br>Utilizzo della logica precedente: <br>";
echo "SALUTO DI STUDENTE_1: <br>".$Studente1->getPagBenvenutoStud();
echo "<br><br>SALUTO DI STUDENTESSA_1: <br>".$Studentessa1->getPagBenvenutoStud();
*/
echo "Sfruttiamo i TRAIT 'gender.php' e 'miPresento.php': <br><br>";
echo $Studente1->miPresento("m")."<br>";
echo $Studentessa1->miPresento("F")."<br>";
?>
```

Creati gli oggetti *Studente1* e *Studentessa1*, sarà sufficiente utilizzare solamente il metodo del TRAIT ***miPresento***: **miPresento($gender)**.

Project Work 11

Nel PW-10, provate a cambiare la visibilità del metodo «getPagGender», del trait «gender», da «**public**» a «**private**».

Cosa accade? Tentate una risposta prima di eseguire il codice…

Project Work 11

Nel PW-10, provate a cambiare la visibilità del metodo «getPagGender», del trait «gender», da «**public**» a «**private**».

Cosa accade? Tentate una risposta prima di eseguire il codice…

SOLUZ.

L'utilizzo di un trait all'interno della classe è come fare un copia-incolla, sciogliendo anche i vincoli che invece sussistono in un legame di ereditarietà.

Per esempio, variabili/metodi «private» possono essere tranquillamente utilizzati dalla classe che fa uso del trait. Al contrario, estendendo una classe con metodi «private» questi non verrebbero ereditati. Da qui potete intuire come un uso «semplicistico» dei trait potrebbe portare ad un codice poco chiaro (non ad oggetti) e, quindi, indurre ad errori di logica.

La keyword «final»

Questo elemento del linguaggio serve a modificare le modalità di utilizzo di metodi e proprietà di una classe; cioè, quando utilizzato per dichiarare metodi e/o costanti non ne consente la modifica.

In particolare, per i metodi dichiarati «final» non è consentito fare OVERRIDING. Mentre applicare «final» per le costati, queste non possono più essere modificate.

Project work 12

Utilizzando il PW-7, dove si richiede di fare overriding del metodo *getPagBenvenuto()* della classe «Persona», provate a definire quel metodo come segue:

modifica

```
42      /* metodo che restituisce la pagina di saluto
43       * utilizziamo la keyword "final" quindi non si potrà fare overriding
44      */
        protected final function getPagBenvenuto() {
46          $this->saluto = "Buongiorno sono ".$this->nome." ". $this->cognome;
47          return $this->saluto;
48      }
```

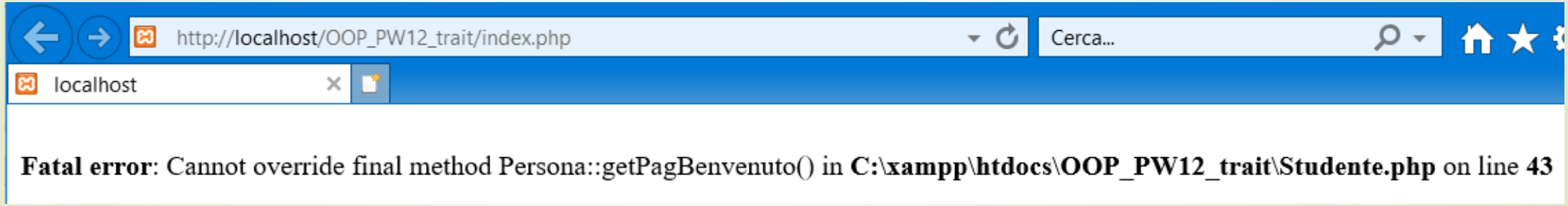
Project work 12

Utilizzando il PW-7, dove si richiede di fare overriding del metodo *getPagBenvenuto()* della classe «Persona», provate a definire quel metodo come segue:

modifica

```
42      /* metodo che restituisce la pagina di saluto
43       * utilizziamo la keyword "final" quindi non si potrà fare overriding
44      */
        protected final function getPagBenvenuto() {
46          $this->saluto = "Buongiorno sono ".$this->nome." ". $this->cognome;
47          return $this->saluto;
48      }
```

output



GRAZIE!