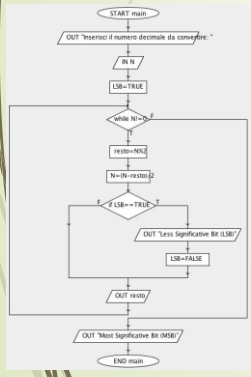
Object Oriented Programming

Prof. Ing. Loris Penserini
elpense@gmail.com

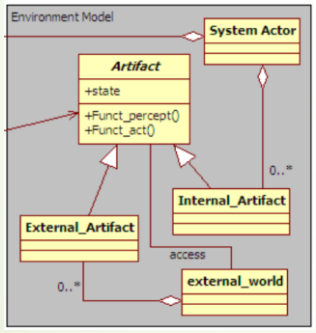
Paradigmi di programmazione…

Alcuni principali paradigmi di programmazione e livelli di astrazione del pensiero computazionale.

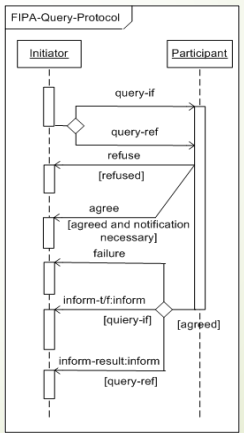
Flow-Chart (Structured Prog.)



UML (Object Oriented Prog.)



Agent-UML (Agent Oriented Prog.)



Scratch/Blockly (Block based Prog.)

Modellare il Mondo Reale

Ciascun paradigma di programmazione fornisce allo sviluppatore un differente approccio concettuale per implementare il pensiero computazionale, cioè la definizione della strategia algoritmica utile per affrontare e risolvere un problema del mondo reale.

La OOP è attualmente il paradigma di sviluppo del SW più utilizzato, per questo molti linguaggi del Web che in passato nascevano non ad oggetti ora lo sono diventati, come il C → C++, il PHP dopo la ver. 5, mentre altri sono nati direttamente OO come JAVA (JSP e Servlet).

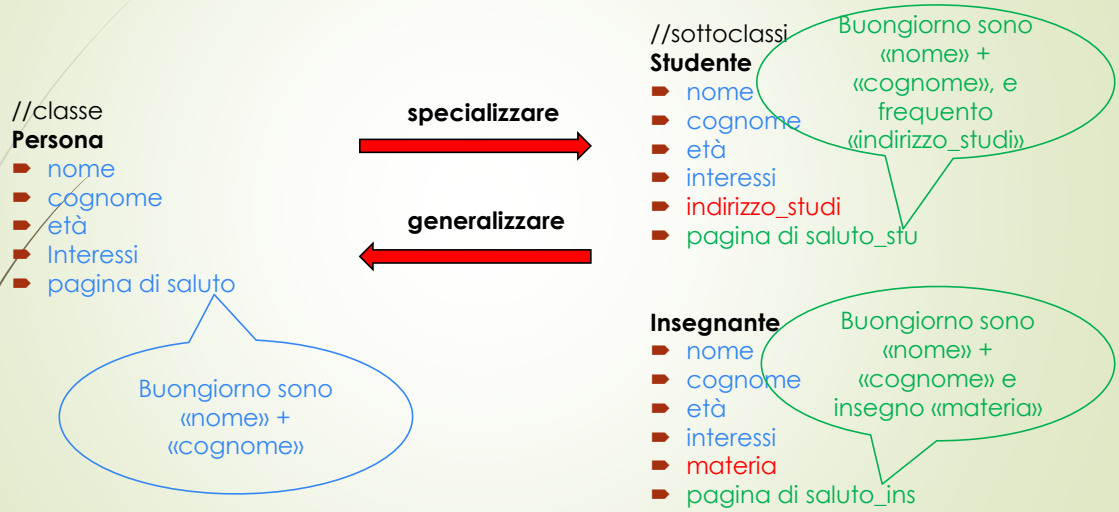
In ogni caso, per creare pagine Web dinamiche, il protocollo standard rimane sempre il **Common Gateway Interface (CGI)**, cioè, indipendentemente dal linguaggio di programmazione usato, si lascia aperta la possibilità di eseguire codice remoto (su un server) invocandolo da un client Web (browser).

Perché OOP per lo sviluppo di SW

Ecco alcuni vantaggi di pensare ad un algoritmo in termini di Oggetti:

- Astrarre dalla complessità del codice per concentrarsi maggiormente sulle similitudini tra gli oggetti software e gli oggetti del mondo reale che si vogliono modellare.
- Pensare al comportamento (behavior) di un oggetto software come al suo analogo reale: causalità e relativi effetti.
- Notevole aumento della possibilità di riuso del codice, con conseguente aumento della produttività di qualità:
  - Maggiore stabilità delle applicazioni;
  - Facilità nell'aggiornare mantenere il codice

Concetto di «classe» in OOP



Cosa è un «oggetto»?

Nella OOP il concetto di «oggetto» è :

- Un processo in esecuzione in memoria
- Una applicazione che fa uso di funzioni/metodi appartenenti a classi diverse (librerie diverse)
- Una applicazione alla quale si può dare uno stato iniziale che ne determina il comportamento (behavior) iniziale.
- Una applicazione che interagisce con il mondo esterno determinando il comportamento che più si adatta per quello scenario

Differenza tra «classe» e «oggetto»

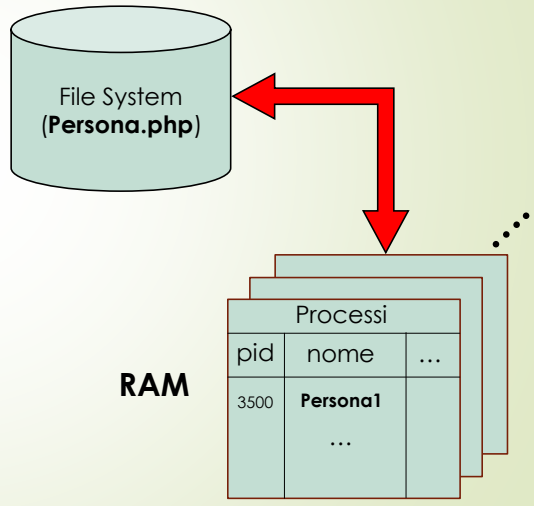
DESIGN time

//classe: è un file
Persona
- nome
- cognome
- età
- Interessi
- pagina di benvenuto

//oggetto: è un processo
Persona1
- nome → Mario
- Cognome → Rossi
- Età → 50
- Interessi → robotica
- pagina di benvenuto → Ciao Mario Rossi

RUN time

Memoria di massa



Proprietà fondamentali in OOP

Tutti i linguaggi OOP forniscono sempre queste tre proprietà:

- **Ereditarietà**
- **Incapsulamento**
- **Polimorfismo**

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 <u>Studente</u> da quale blocco di codice si inizia?

Project work

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»

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**

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**.

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 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

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 Project work

```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**)

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 __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();
?>
```


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

ITS- Turismo Marche

Esercitazione

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)**.

GRAZIE!