



CORSO FULL STACK WEB DEVELOPER



OOP

Scopriamo la
Programmazione Orientata agli Oggetti



OOP - A cosa serve

Gestire grandi progetti richiede grandi quantità di codice che, come abbiamo visto, cresce velocemente di complessità.

L'approccio OOP introduce regole per promuovere le seguenti pratiche:

- 1 Modularità
- 2 Riutilizzo del Codice
- 3 Isolamento
- 4 Estensibilità

**La OOP è un paradigma di programmazione e
non è una esclusiva di PHP.**

**Tanti altri linguaggi offrono gli strumenti per
seguire questo paradigma
(C++, Java, Python, JavaScript, ...)**



OOP - Le Classi

Alla base della OOP c'è il concetto di **Classe**.

Una classe è un **modello** che descrive un preciso **pezzo di realtà**.

Immaginiamola quindi come un progetto, da attuare integralmente, per costruire un certo oggetto.

Alcuni esempi:

- Un'auto
- Un animale
- Un utente di un sistema di contenuti



OOP - Le istanze o oggetti

Una Classe, nel momento in cui viene implementata, genera una istanza (o oggetto).

Le istanze sono le concretizzazioni di quel modello.

Alcuni esempi:

- La classe Auto ha 2 istanze:
 - l'auto Alfa Romeo
 - l'auto Lancia
- La classe Utente ha 3 istanze:
 - Pippo
 - Pluto
 - Paperino

Abbiamo già usato la parola oggetto in JS, in quel caso era una contrazione di oggetto letterale, molto simile al concetto di array associativo.

Vi viene in mente un oggetto/istanza che abbiamo già usato?



Classi in PHP

Come si crea una classe?

- keyword **class**
- **nome classe**: il nome di una classe si indica in **PascalCase**

Come si generano gli oggetti/istanze di quella classe?

- keyword **new**
- **nome classe**: richiamiamo la classe con le **parentesi tonde**, proprio come si invoca una funzione

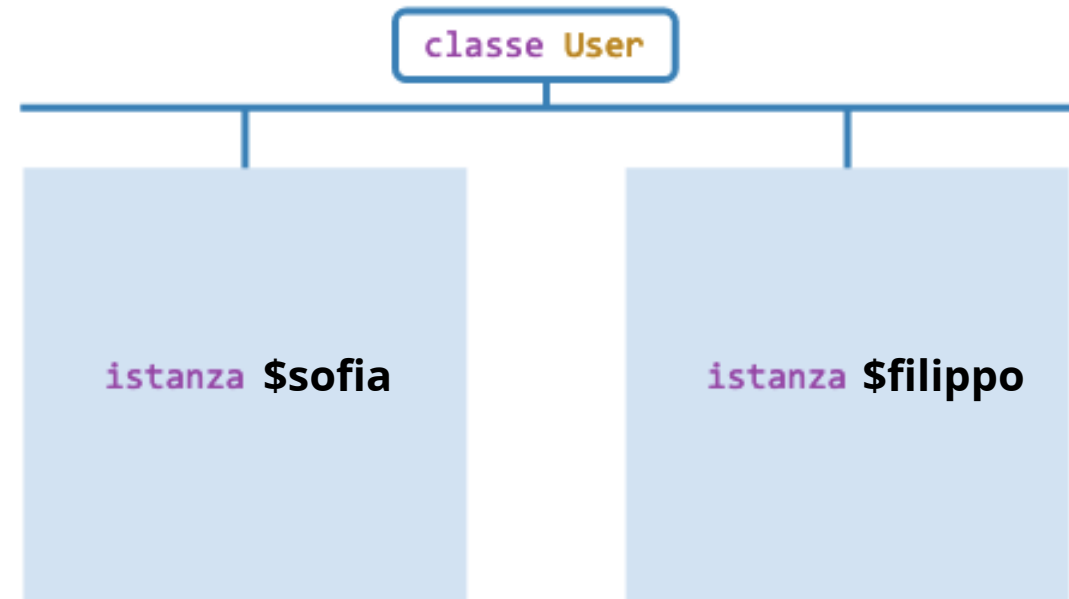
```
1 1 class User {  
2  
3 }  
4  
5  
6 $sofia = new User();  
7 $filippo = new User();
```




Classi in PHP

Ogni istanza è come una copia della classe di partenza.

Hanno in comune le caratteristiche espresse nella classe, ma sono mondi separati.





Variabili (o attributi) nelle classi

Le classi possono avere delle **variabili**, dette **attributi**.

Per accedere agli attributi di un'istanza si usa l'operatore **->**

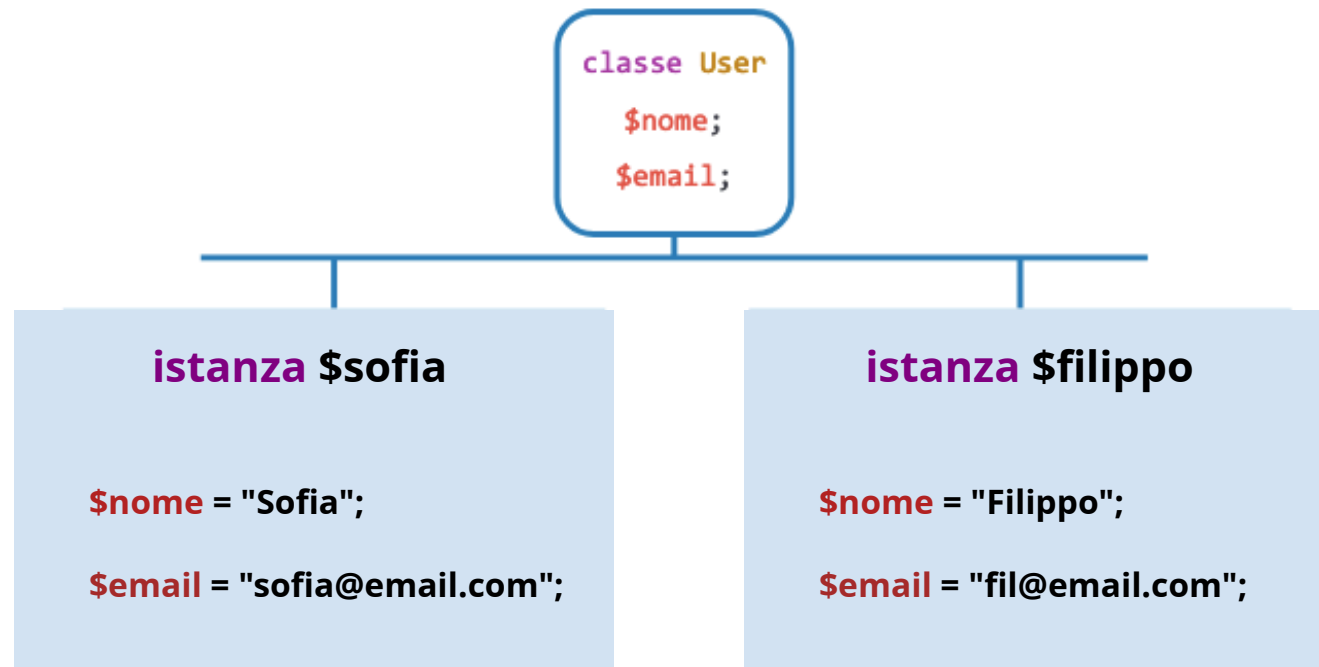
```
1 1 class User {  
2     public $nome;  
3     public $email;  
4 }  
5  
6 $sofia = new User();  
7 $sofia->nome = "Sofia";  
8 $sofia->email = "sofia@email.it";
```



Abbiamo già usato questo operatore, quando?



Le istanze, ambienti diversi





Metodi nelle Classi

Le Classi possono avere delle **funzioni**, dette **metodi**, pertinenti a ciò che rappresentano.

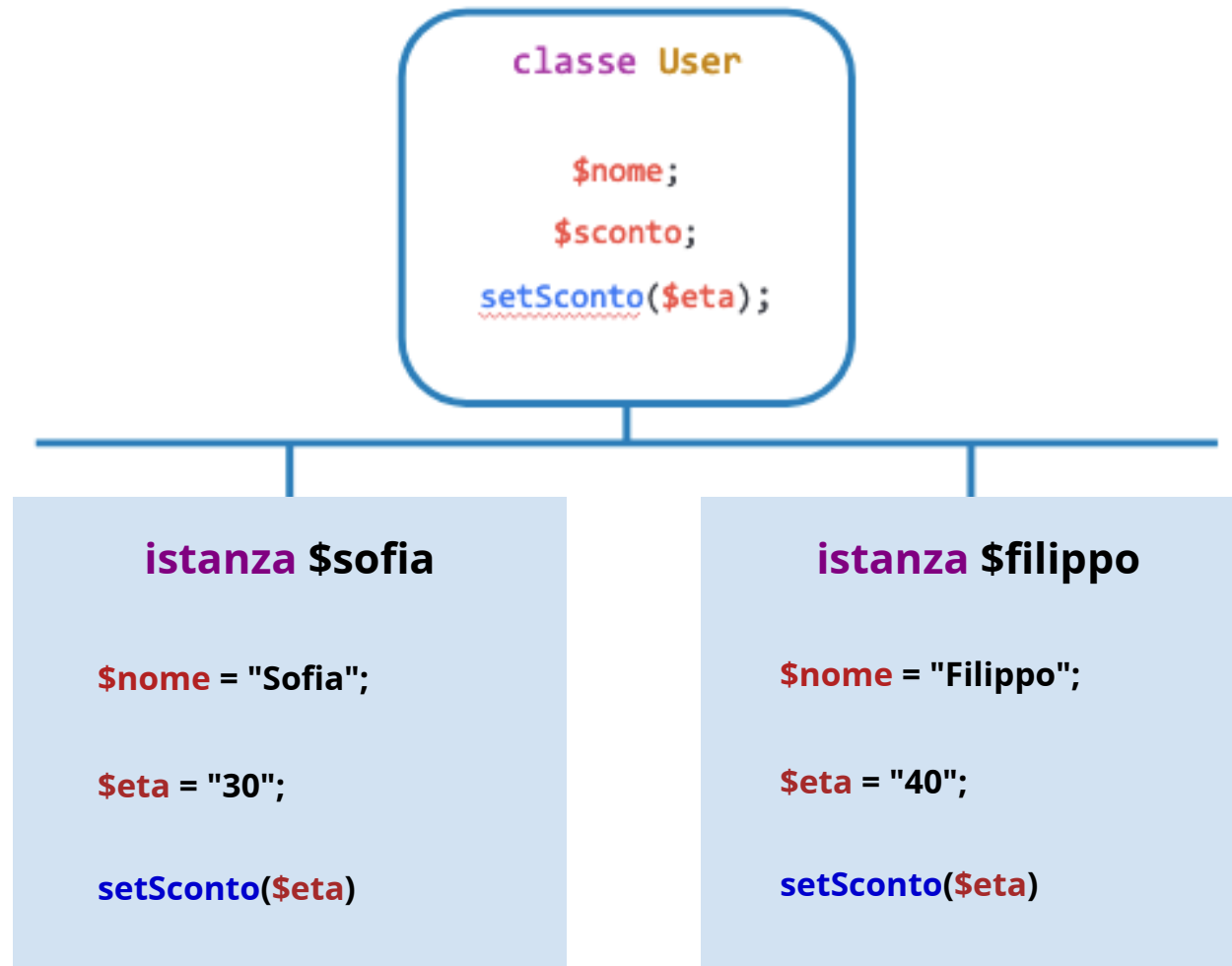
Ogni istanza avrà a disposizione tutti i metodi definiti nella classe.

Per richiamare i metodi su un'istanza si usa l'operatore **->**

```
1 1 class User {
2   public $nome;
3   public $sconto = 0;
4
5   public function setSconto($eta) {
6       if($eta > 65) {
7           $this->sconto = 40;
8       }
9   }
10
11  public function getSconto() {
12      return $this->sconto;
13  }
14 }
15
16 $filippo = new User();
17 $filippo->setSconto(40);
18 $sconto_filippo = $filippo->getSconto(); //0
```



Le istanze, ambienti diversi



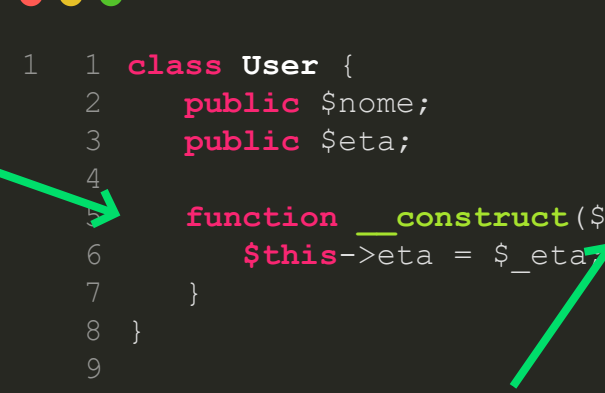


Costruttori

Ogni Classe può avere un particolare tipo di funzione, chiamata **costruttore**, che viene invocata quando si usa l'operatore new.

Il costruttore permette di eseguire azioni nel momento in cui viene creata l'istanza della classe.

Se prevediamo un argomento per il costruttore, dobbiamo fornire il relativo valore al momento della creazione dell'istanza.



```
1 1 class User {
2   public $nome;
3   public $eta;
4
5   function __construct($_eta) {
6       $this->eta = $_eta;
7   }
8 }
9
10 $filippo = new User(40);
11 echo $filippo->eta; // 40
```



Costruttore

Il costruttore viene dichiarato solamente nella Classe.

La *manifestazione* del costruttore nell'istanza sono proprio le parentesi tonde.

```
1 1 class User {  
2   public $nome;  
3   public $eta;  
4  
5   function __construct($_eta) {  
6       $this->eta = $_eta;  
7   }  
8 }  
9  
10 $filippo = new User(40);
```




Composizione



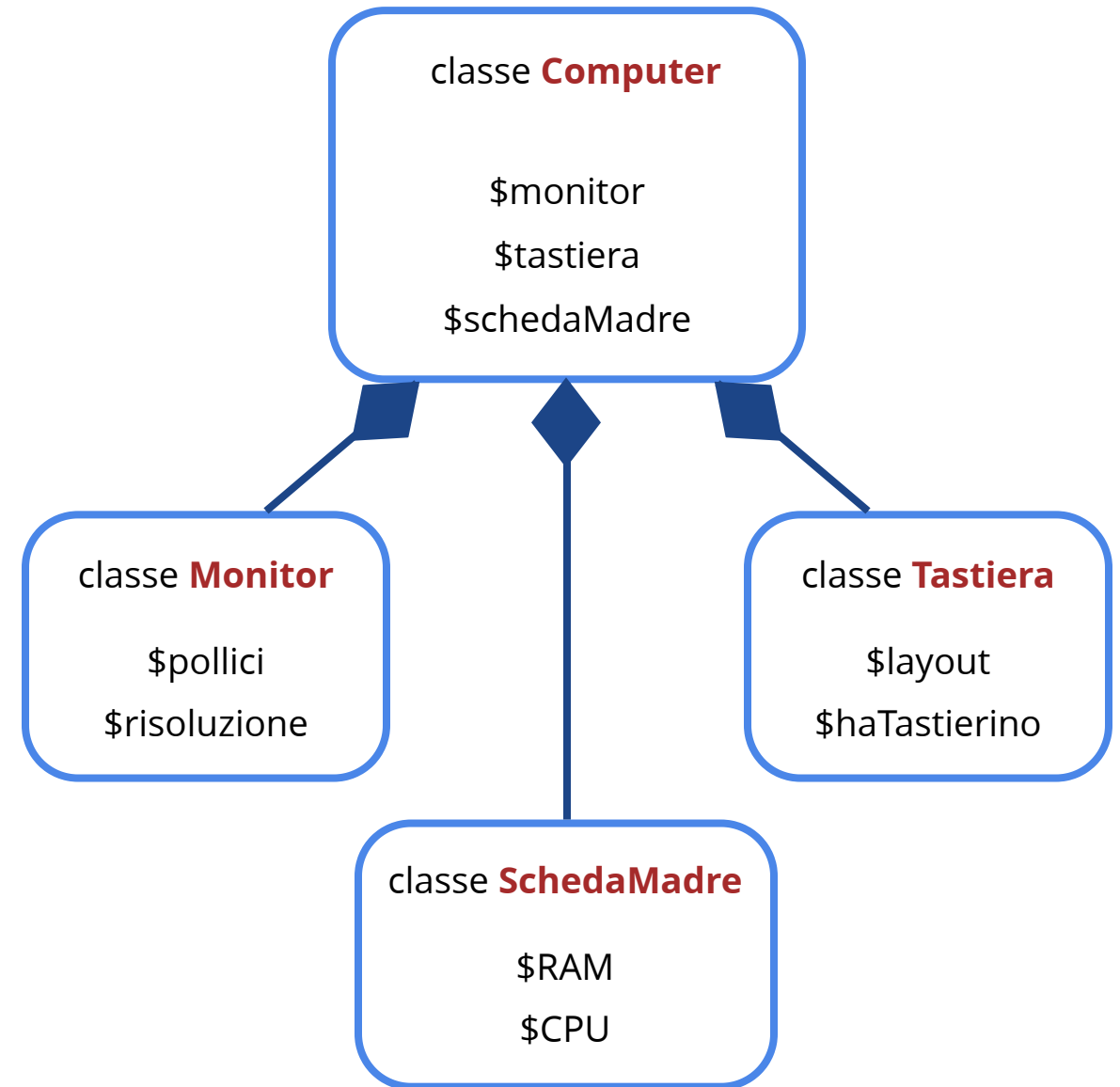
Composizione: cos'è?

La composizione è uno dei concetti fondamentali della programmazione orientata agli oggetti.

Tramite la composizione è possibile **utilizzare una classe per costruire un'altra classe più complessa**, creando una relazione di tipo **ha-un** (*has-a*) tra le due classi.

Pensiamo ad esempio ad un computer portatile: questo è composto da un monitor, una tastiera, una scheda madre, ecc.

Ogni elemento che compone il pc è un oggetto a sé stante ma tutti fanno parte di una classe più grande e complessa che è appunto il computer.





Composizione: come si fa?

Tornando all'esempio di prima, un utente, oltre ad avere un nome e un'età, ha anche un indirizzo.

L'indirizzo a sua volta è composto da: via, città e CAP.

Possiamo quindi definire la classe Address con le sue caratteristiche e poi inserire dentro alla classe User un'istanza della classe Address che rappresenta l'indirizzo specifico di quel particolare utente.

```
1 1 class Address
2 {
3     public $street;
4     public $city;
5     public $postcode;
6
7     public function __construct($street, $city, $postcode)
8     {
9         $this->street = $street;
10        $this->city = $city;
11        $this->postcode = $postcode;
12    }
13 }
```

```
1 1 class User
2 {
3     public $nome;
4     public $eta;
5     public $address;
6
7     function __construct($nome, $eta, Address $address)
8     {
9         $this->nome = $nome;
10        $this->eta = $eta;
11        $this->address = $address;
12    }
13 }
```



Composizione: come si usa?

Abbiamo quindi definito la relazione ha-un tra la classe User e la classe Address.

Questo significa che la classe **User** ha bisogno di un'istanza della classe **Address** per essere istanziata.

```
1 1
2 $filippo = new User('Filippo', 50, new Address('Via del codice 404', 'Milano', '20200'));
```

```
1 1 class Address
2 {
3     public $street;
4     public $city;
5     public $postcode;
6
7     public function __construct($street, $city, $postcode)
8     {
9         $this->street = $street;
10        $this->city = $city;
11        $this->postcode = $postcode;
12    }
13 }
```

```
1 1 class User
2 {
3     public $nome;
4     public $eta;
5     public $address;
6
7     function __construct($nome, $eta, Address $address)
8     {
9         $this->nome = $nome;
10        $this->eta = $eta;
11        $this->address = $address;
12    }
13 }
```



Nullsafe Operator

Lavorando con la composizione, per poter accedere alle proprietà dell'istanza contenuta nella classe, è necessario verificare che questa sia effettivamente stata definita, in altre parole, che non sia *null*.

Dalla versione 8 di PHP è stato introdotto l'operatore *Nullsafe*, il quale permette di semplificare il controllo sulle proprietà di un oggetto: prima di accedere alle proprietà di un oggetto, verifica che questo sia diverso da *null*, altrimenti restituisce *null* a sua volta.

```
1 1 <?php
2
3 // Questa riga di codice:
4 $result = $user->address->city;
5
6 // Corrisponde al controllo fatto in questo blocco:
7 if (is_null($user->address)) {
8     $result = null;
9 } else {
10     $result = $user->address->city;
11 }
```



Proprietà e metodi statici



Proprietà statiche

In alcune occasioni, ci possono essere delle proprietà che non dipendono dall'istanza, ma sono invece **comuni a tutte le istanze** della classe.

Le **proprietà statiche** sono **condivise** da tutte le istanze della classe.

Nel nostro esempio significa che tutti gli indirizzi avranno come nazione la *stessa* stringa "Italy".

Per definire una proprietà come statica si usa la keyword **static**.

```
1 1 class Address
2 {
3     public $street;
4     public $city;
5     public $postcode;
6     public static $country = "Italy";
7
8     public function __construct($street, $city, $postcode)
9     {
10         $this->street = $street;
11         $this->city = $city;
12         $this->postcode = $postcode;
13     }
14 }
```

Docs: https://www.w3schools.com/php/php_oop_static_properties.asp



Proprietà statiche

Come accedo ad una proprietà statica?

Per accedere alle proprietà statiche è sufficiente usare:

- il nome della classe (es.: **Address**)
- l'operatore **::**
- il nome della proprietà (es.: **\$country**)

```
1 1 class Address
2 {
3     public $street;
4     public $city;
5     public $postcode;
6     public static $country = "Italy";
7
8     public function __construct($street, $city, $postcode)
9     {
10         $this->street = $street;
11         $this->city = $city;
12         $this->postcode = $postcode;
13     }
14 }
```

```
1 1 echo Address::$country;
```



Cosa ci restituisce questo echo?



Proprietà statiche

Come accedo ad una proprietà statica dall'interno della classe?

Abbiamo visto come accedere al valore di una proprietà statica dall'esterno, ma come facciamo ad accedervi da dentro un metodo della nostra classe?

- parola chiave **self**
- l'operatore **::**
- il nome della proprietà (es.: **\$country**)

```
1 1 class Address
2 {
3     public static $country = "Italy";
4
5     // ... costruttore e proprietà varie
6
7     public function getStaticCountry() {
8         return self::$country;
9     }
10 }
```

```
1 1 $address = new Address('Corso del Popolo, 1', 'Roma', '00123');
2 echo $address->getStaticCountry(); // Italy
```



Metodi statici

I metodi statici si utilizzano quando per svolgere determinate operazioni non si ha bisogno di un'istanza della classe.

Questi metodi, infatti, non si richiamano a partire da un oggetto, bensì dalla classe. Di conseguenza, al loro interno non si può utilizzare la parola chiave `$this`.

Per creare un metodo statico si utilizza la parola chiave **static**.

```
1 1 class Greetings
2  {
3
4      public static function sayHelloWorld()
5      {
6          return "Hello World"
7      }
8  }
```

Docs: https://www.w3schools.com/php/php_oop_static_methods.asp



Metodi statici

Come utilizzo un metodo statico?

Per richiamare un metodo statico è sufficiente usare:

- il nome della classe (es.: **Greetings**)
- l'operatore ::
- il nome del metodo (es.: **sayHelloWorld**)

```
1 1 class Greetings
2  {
3
4  public static function sayHelloWorld()
5  {
6      return "Hello World"
7  }
8 }
```

```
1 1 echo Greetings::sayHelloWorld(); // Hello World
```



LIVE CODING



ESERCITAZIONE