**OOP (Object Oriented Programming)**

Modo di programmare basato su:

* CLASSE 🡪 modello che descrive (caratteristiche, funzionalità 🡪 VARIABILI o ATTRIBUTI o PROPRIETÀ) un’entità, cioè un preciso pezzo di realtà
* ISTANZA o OGGETTO 🡪 ogni elemento che può essere rappresentato tramite il modello

Es. Audi è un’istanza della classe automobile; Drew è un’istanza della classe cane, ...

Sintassi:

* per creare una classe

|  |  |
| --- | --- |
| class NomeClasse {  // all’interno definisco le proprietà di quella specifica classe  public $attributo1;  protected $attributo2;  private $attributo3 = ‘valore’;  } | generalmente il nome viene scritto in PascalCase ed è al singolare  La parola chiave definisce il livello di visibilità di quella proprietà:   * si può accedere e modificare il contenuto sia dall’interno che dall’esterno della classe * si può accedere e modificare il contenuto solo dall’interno della classe stessa o di una classe derivata da essa * si può accedere e modificare il contenuto solo dall’interno della classe   N.B. Se tutti gli oggetti condividono per lo stesso attributo lo stesso valore posso già definirlo |

* le classi possono avere al loro interno anche delle FUNZIONI o METODI

|  |  |
| --- | --- |
| class NomeClasse {  public function nomeFunzione($parametro) {  // codice  }  } | I metodi sono di due tipi:  - SETTER 🡪 per impostare il valore di una proprietà  function setNomeFunzione($parametro) {  $this->attributo = ‘valore’  }  - GETTER 🡪 per ottenere il valore di una proprietà  function getNomeFunzione() {  return $this->attributo  }  N.B. perché effettivamente sia riutilizzabile devo salvarlo in una variabile quando invocato:  Es. $tesla\_price = $tesla->getPrice(); |
| (Type $parametro = valore) | Per ogni parametro può essere specificato il tipo:  - string  - float  - bool  - int  E volendo anche un valore di default |

* per creare un’istanza della classe

|  |  |
| --- | --- |
| $oggetto = new NomeClasse(); |  |

* per accedere agli attributi di un’istanza o per invocare i metodi sull’istanza

|  |  |
| --- | --- |
| $oggetto->attributo = ‘valore’; | $oggetto->nomeFunzione(eventuale parametro); |

* FUNZIONE COSTRUTTORE o MAGIC METHOD

|  |  |
| --- | --- |
| public $attributo;  function \_\_construct($attributo) {  $this->attributo = $attributo;  } | Viene invocata nel momento in cui viene creata l’istanza:  $oggetto = new Classe (valore attributo);  N.B. una volta definita la funzione costruttore l’invocazione DEVE contenere tutti i parametri specificati altrimenti l’oggetto non viene creato) |

Per verificare man mano var\_dump($oggetto);

getFullName() {  
return $this->name . ‘ ’ . $this->lastname  
}

EREDITARIETÀ

🡪 E’ una delle possibili relazioni tra due classi

🡪 Accade quando una classe B riceve TUTTI i metodi e le proprietà di UNA classe A

🡪 La classe B può anche eventualmente aggiungerne altri

🡪 Se alla classe A vengono aggiunte proprietà o classi, automaticamente la classe B le eredita

Sintassi:

* class NomeClasseFiglia extends NomeClasseGenitore { … }

POLIFORMISMO

🡪 Nella classe figlia posso anche modificare il contenuto di un metodo della classe genitore

🡪 La classe figlia eredita anche la funzione costruttore

🡪 Se alla classe figlia vogliamo aggiungere altri parametri come funzione costruttore:

function \_\_construct ($attributoGenitore, $attributoGenitore,… $attributoFiglio) {

// faccio ereditare i parametri del genitore

parent::\_\_construct($attributoGenitore, $attributoGenitore);

// definisco gli attributi esclusivi del figlio

$this->attributoFiglio = $attributoFiglio;

}

hash()

require\_once \_\_DIR\_\_ . ‘nomeFile.php’;