OOP

Che cos’è la Programmazione ad Oggetti?

- Innanzitutto la OOP(Object Oriented Programming) è una metodologia, approccio, paradigma, idea ecc. di sviluppo software: Praticamente non è obbligatorio utilizzare questa metodologia, ma a fine capitolo capirete voi stesso il motivo per il quale dovrete sposare questo approccio.
- Due dei Principi fondamentali della OOP sono:
  - **Astrazione**: ossia la capacita di creare del codice in grado di astrarre la risoluzione di una tipologia di problemi (e non uno solo!);
  - **Incapsulamento**: Possibilità di isolare il comportamento rispetto all'intero programma.

Che cos'è una Classe?

Una classe serve a **Classificare**

- Il paradigma OOP si basa sul concetto di Classe, ossia: un'astrazione di una entità contenente sia codice (funzione) che dati (variabili);
- La classe è un tipo di dato definito dall'utente, utilizzato per modellare un oggetto.
- Una classe definisce le azioni (metodi) di una particolare entità e le relative proprietà (Attributi).

Esempio del perché si rende necessario l'utilizzo di una classe:

```
//Tekken
$player1 = [
  'firstname' ⇒ 'Jin',
  'lastname' ⇒ 'Kazama',
  'overall' ⇒ 80
];

$player2 = [
```

```
  'firstname' => 'Bryan',
  'lastname' => 'Fury',
  'overall' => 81
];


var_dump($player1['firstname']); //Jin
var_dump($player2['lastname']); //Fury
```

Immaginiamo ora di dover creare altri 40 personaggio giocabili. Per ogni player dovremmo creare un array preoccupandoci di inserire le stesse chiavi, senza sbagliare o alterarne qualcuna.

Come posso ovviare a questo problema qui? **Con la creazione di una Classe Player.**

- Classe al singolare;
- Rigorosamente in Inglese;
- Lettera Maiuscola;
- Attributi, Costruttore e Metodi;
- Visibilità:
    - Public
    - Private
    - Protected:

```
 //Attributi

class Player
{

  public $firstname;
  public $lastname;
  public $overall;

}
```

```
$player1 = new Player;

print_r($player1);
```

Il paradigma ad oggetti ci permette di creare una descrizione univoca per ogni persona. Questa descrizione è detta Classe.

Nuove Keyword:

- **New**: Con la keyword new stiamo istruendo il nostro programma a creare un nuovo oggetto di classe Persona. Questo oggetto avrà 3 attributi: nome, cognome e forza.
- **This**: Esempio pratico con Astrazione del **$this** ($this : Quando si crea un nuovo oggetto di una determinata classe, viene creato un token identificativo per ogni oggetto creato. Ciò vuol dire che $this è un riferimento univoco all'oggetto.)

```
class Player
{

  public $firstname;
  public $lastname;
  public $overall;


  public function __construct($a,$b,$c)
  {
    var_dump($a);
    var_dump($this->firstname);
    die();
  }
}

$player1 = new Player('Jin','Kazama',78);
```

```
print_r($player1 );
```

Costruttore:

```
$player1 = new Player($a,$b,$c);

class Player
{

  public $firstname;
  public $lastname;
  public $overall;


    public function __construct($a, $b, $c){//costruttore
        $this→firstname = $a;
    $this→lastname = $b;
    $this→overall = $c;
    }
}


$player = new Player('Jin', 'Kazama', 89);

print_r($player);

// Accedere al singolo nome

print_r($player→firstname);
```

Metodi:

```
class Player
{
```

```
  public $firstname;
  public $lastname;
  public $overall;


    public function __construct($a, $b, $c){
        $this->firstname = $a;
    $this->lastname = $b;
    $this->overall = $c;
    }

  public function selectPlayer()
  {
    echo "Hai scelto  $this->firstname  $this->lastname \n";
  }
}

$player1 = new Player('Jin', 'Kazama', 80);
$player2 = new Player('Bryan', 'Fury', 81);

$player1->selectPlayer();
```

Attributi Statici

I metodi e le proprietà statiche possono essere richiamate senza istanziare un oggetto di quella classe.

```
class Player
{

  public $firstname;
  public $lastname;
  public $overall;
  public static $count = 0;


  public function __construct($a, $b, $c)
  {
```

```
    $this->firstname = $a;
    $this->lastname = $b;
    $this->overall = $c;
  }

  public function selectPlayer()
  {
    echo "Hai scelto  $this->firstname  $this->lastname \n";
  }

  public function fight($opposite)
  {
    if ($this->overall > $opposite->overall) {
      self::$count++;
    }
  }
}

$player1 = new Player('Jin', 'Kazama', 85);
$player2 = new Player('Bryan', 'Fury', 81);
$player3 = new Player('Yoshimitsu', '', 79);
$player4 = new Player('Eddie', 'Gordo', 80);

$player1->selectPlayer();


$player1->fight($player2);
$player1->fight($player3);
$player1->fight($player4);

echo 'Vittorie: ' . Player::$count;
```

Metodi Statici

Una classe può implementare anche metodi statici che, così come gli attributi statici, possono essere richiamati senza istanziare un oggetto di quella classe.

```
class Player
{
```

```
  public $firstname;
  public $lastname;
  public $overall;
  public static $count = 0;


  public function __construct($a, $b, $c)
  {
    $this->firstname = $a;
    $this->lastname = $b;
    $this->overall = $c;
  }

  public function selectPlayer()
  {
    echo 'Hai scelto ' . $this->firstname . ' ' . $this->lastname . "\n";
  }

  public static function counter()
  {
    echo "Vittorie: " . self::$count++ . "\n";
  }


  public function fight($opposite)
  {
    if ($this->overall > $opposite->overall) {
      self::$count++;
    }
  }
}

$player1 = new Player('Jin', 'Kazama', 85);
$player2 = new Player('Bryan', 'Fury', 81);
$player3 = new Player('Yoshimitsu', '', 79);
$player4 = new Player('Eddie', 'Gordo', 80);
```

```
$player1->selectPlayer();


$player1->fight($player2);
$player1->fight($player3);
$player1->fight($player4);

Player::counter();
```