OOP - Cosa abbiamo ottenuto

Siamo stati obbligati a ragionare per classi, individuando una entità da modellare, inserendo nella classe tutte le sue proprietà (le variabili) e i suoi metodi (le funzioni).

Il codice risultante è in un luogo coerente, facile da trovare, facilmente modificabile e ripetuto una sola volta per ogni istanza della Classe.

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

Possiamo ottenere di più?

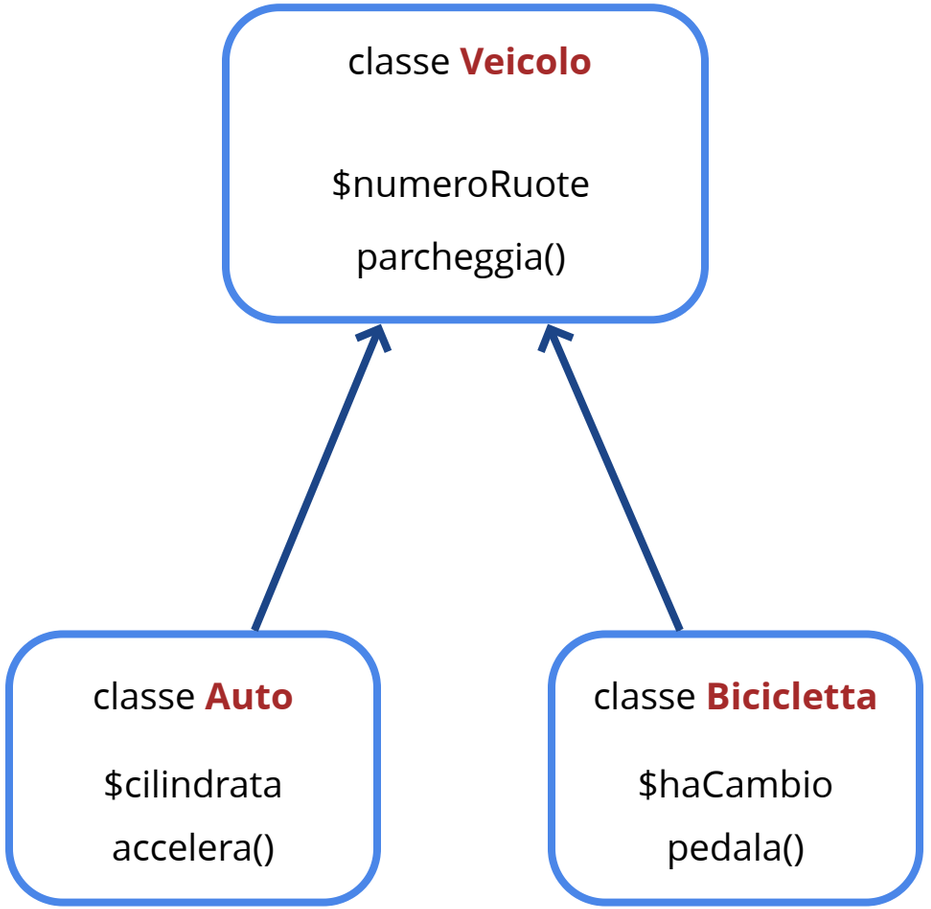
Ereditarietà

Ereditarietà: cos'è?

L'ereditarietà è un caposaldo della programmazione orientata agli oggetti.

Abbiamo visto che con la composizione possiamo creare una relazione *ha-un* tra due classi.
Tramite l'ereditarietà possiamo creare una relazione di tipo ***è-un*** (*is-a*) tra due classi.

Grazie all'ereditarietà possiamo definire una classe a partire da una classe più generica ed estenderne le funzionalità. La classe figlia **eredita** quindi tutte le proprietà e i metodi della classe genitore e può aggiungerne di propri o modificare il comportamento di quelli esistenti.



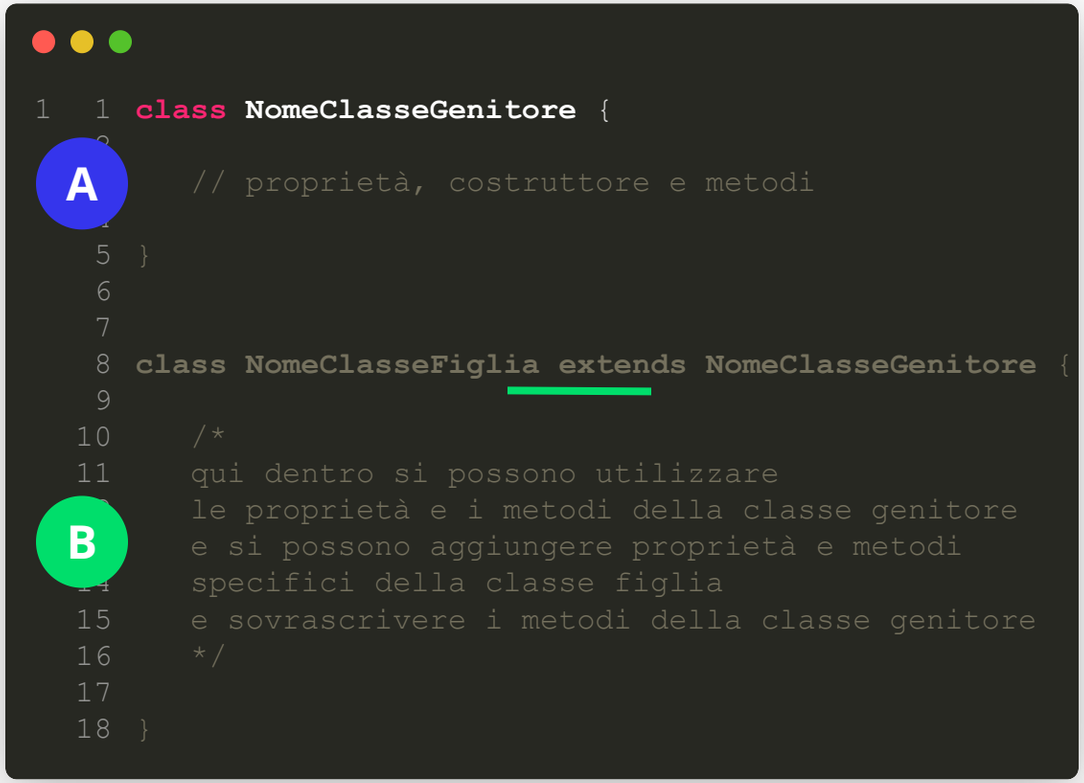
Ereditarietà in PHP

Una **classe figlia B** eredita da una **classe genitore A** quando riceve tutti i metodi e le proprietà di **A** e ne aggiunge altri che meglio specificano le caratteristiche di **B**.

La classe figlia B può anche modificare il funzionamento dei metodi della classe genitore A, andando a ridefinirne il corpo.

Come si scrive questa relazione?

Utilizzando la keyword **extends**



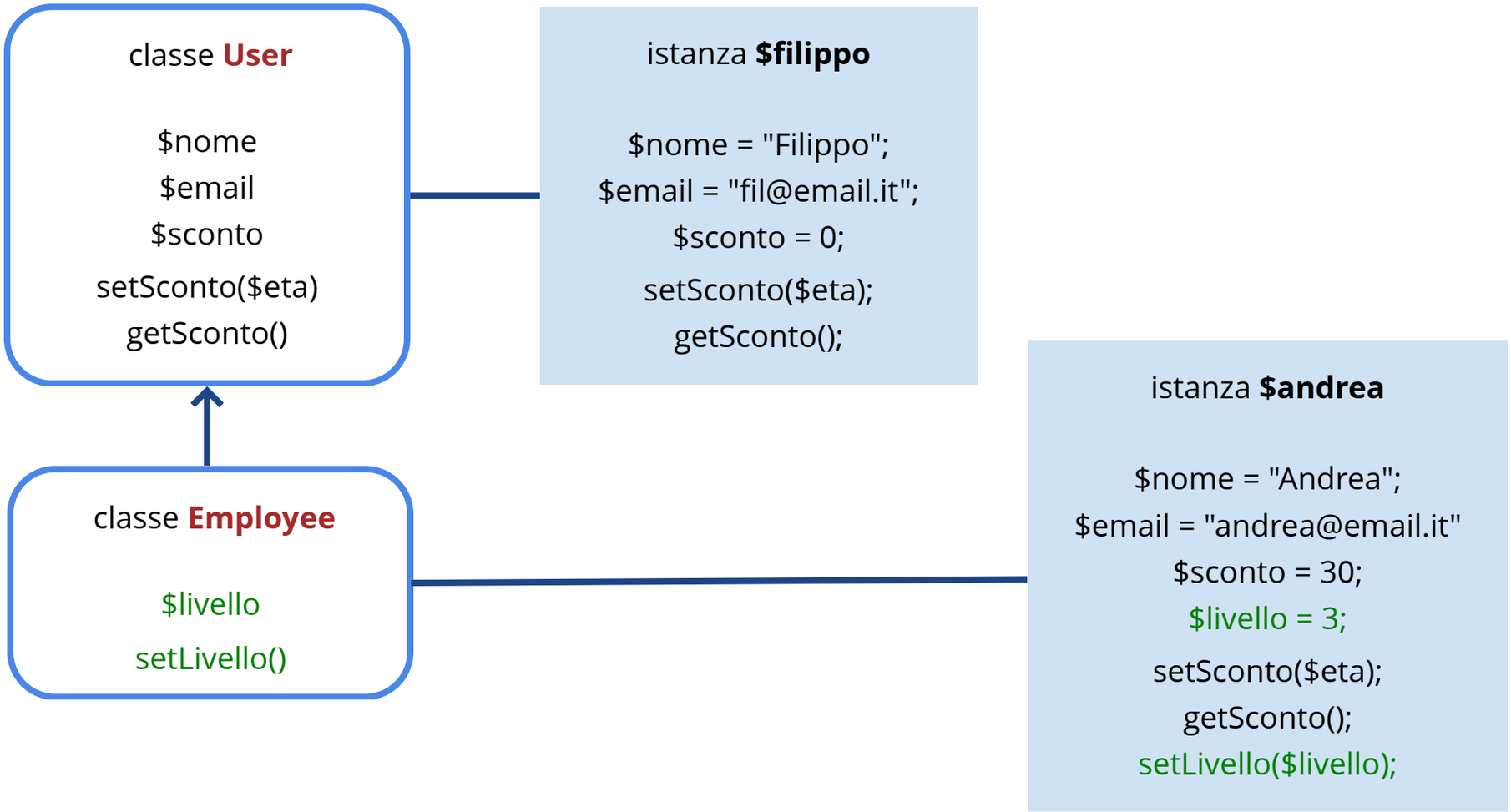
Ereditarietà in PHP

La classe **Employee estende** la classe **User** implementandone tutte le caratteristiche e aggiungendone di nuove (il livello).

```php
class User {
    public $nome;
    public $email;
    public $address;
    public $sconto = 0;

    public function setSconto($eta) {
        if($eta > 65) {
            $this->sconto = 40;
        }
    }

    public function getSconto() {
        return $this->sconto;
    }
}
```

```php
class Employee extends User {
    public $livello;

    public function setLivello($livello) {
        $this->livello = $livello;
    }
}
```

Le istanze, ambienti diversi



Ereditarietà: Polimorfismo

Che cosa succede se abbiamo nella classe figlia
una funzione con lo stesso nome e con un codice diverso all'interno?

```php
class User {
    public $nome;
    public $email;
    public $sconto = 0;

    public function setSconto($eta) {
        if($eta > 65) {
            $this->sconto = 15;
        }
    }

    // ...
}
```

```php
class Employee extends User {
    public $livello;

    public function setLivello($livello) {
        $this->livello = $livello;
    }

    public function setSconto($eta) {
        if($eta > 65) {
            $this->sconto = 50;
        } else {
            $this->sconto = $this->livello * 10;
        }
    }
}
```

Ereditarietà: Polimorfismo

Chiameremo il metodo con lo stesso nome,

ma nell'istanza della classe figlia avremo un risultato diverso.

Classe **User**

istanza **$filippo**

```php
$pippo = new User()
$pippo->nome = "Filippo";
$pippo->setSconto(40);
$pippo->getSconto(); // 0
```

Classe **Employee**

istanza **$andrea**

```php
$paperino = new Employee()
$paperino->nome = "Andrea";
$paperino->setLivello(3);
$paperino->setSconto(40);
$paperino->getSconto(); // 30
```

Visibilità

Per migliorare i concetti di **isolamento**, possiamo restringere l'accesso a metodi e variabili d'istanza secondo determinate condizioni.

1 public

I metodi e le variabili public saranno accessibili da qualsiasi file o metodo abbia accesso all'istanza.

2 protected

Potranno essere utilizzati all'interno della classe o dalle classi che derivano da essa, ma non dall'esterno della classe.

3 private

Potranno essere utilizzati solo all'interno della classe dove sono stati dichiarati

LIVE CODING