Traits

Si può ereditare da due classi?

PHP è un linguaggio ad ereditarietà singola

Potrebbe sorgere l'idea di ereditare da più classi, tipo così:

```
1 1 class Employee extends User and Person
```

In PHP **non è possibile** perché è un *single inheritance language*.

Da PHP5.4 in poi però esiste un modo per riutilizzare il codice in più classi!

Traits!

Ancora più riciclo

I **trait** possono essere usati
per creare proprietà da applicare a più classi.

Come si crea una trait?

keyword **trait**

nome trait: il nome di una trait si indica in **PascalCase**.

```php
trait Position {
    public $lat;
    public $lng;

    public function getCoordinates() {
        return "$lat, $lng";
    }
}
```

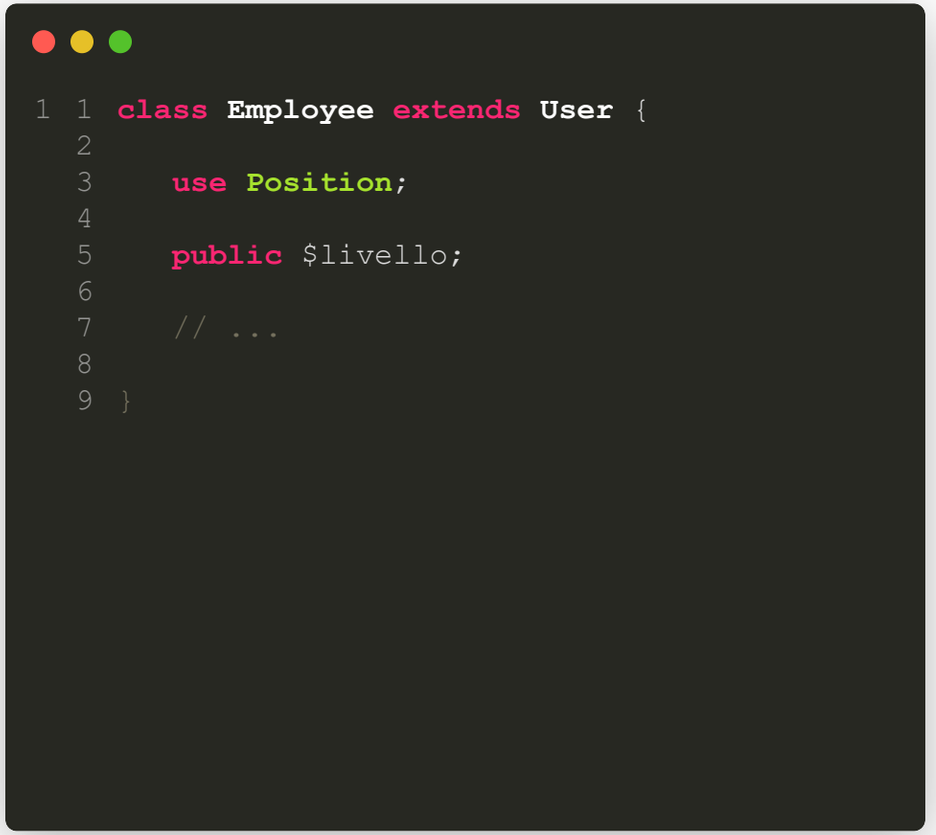
Traits!

Ancora più riciclo

Come si collega un trait ad una classe?

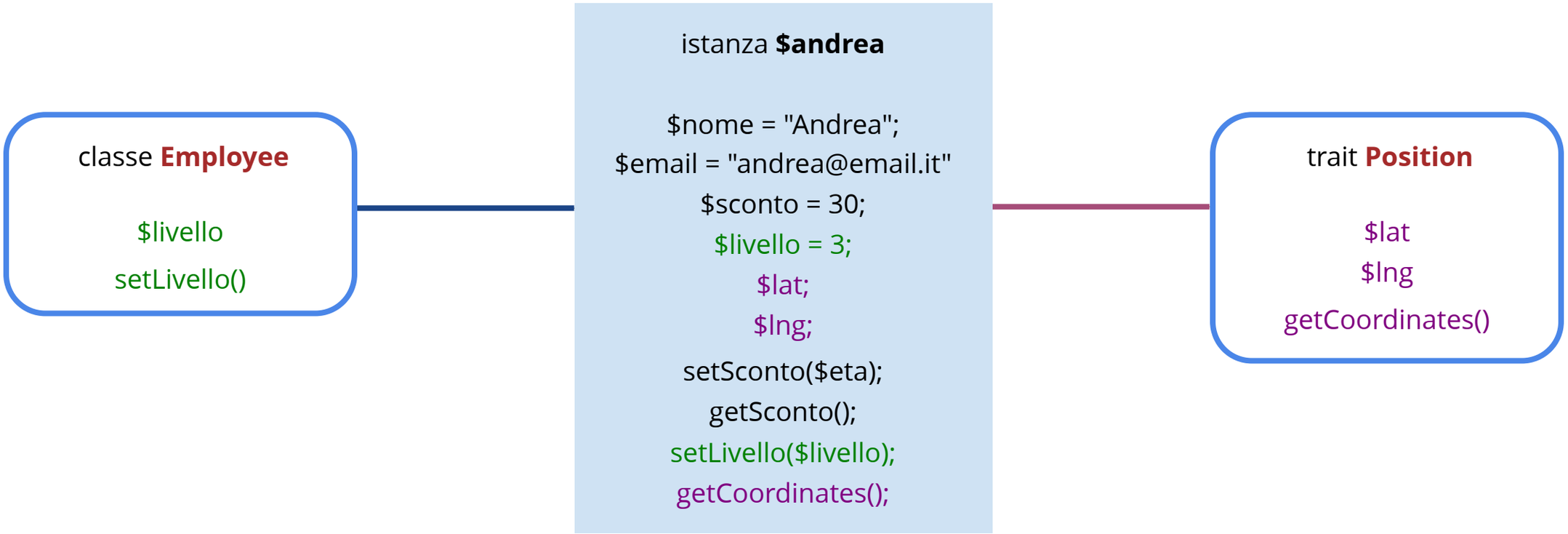
keyword **use**

nome trait: richiamiamo il trait per nome.



```php
class Employee extends User {

    use Position;

    public $livello;

    // ...

}
```

Traits!


classe **Employee**
$livello
setLivello()
istanza **$andrea**
$nome = "Andrea";
$email = "andrea@email.it"
$sconto = 30;
$livello = 3;
$lat;
$lng;
setSconto($eta);
getSconto();
setLivello($livello);
getCoordinates();
trait **Position**
$lat
$lng
getCoordinates()

OOPS!
È avvenuta un’eccezione!

Exceptions

In **PHP** le eccezioni, e quindi la gestione degli errori, **hanno un ruolo molto importante.**

PHP cerca di separare nettamente ciò che sono le **logiche applicative** dagli **errori / eccezioni**.

Da **PHP 5** in poi abbiamo la possibilità di lanciare eccezioni personalizzate così da avere sempre sotto controllo il punto in cui **un errore si verifica**.

Exceptions

Come si dichiara un’eccezione?

keyword **throw**

nuova istanza di: **Exception**

argomento di Exception: **'stringa'**

```
1  1  throw new Exception('Is not a number');
```

Usare un’Eccezione

In una funzione, andiamo quindi a lanciare una nuova eccezione.

Supponiamo che la nostra funzione voglia moltiplicare un **numero * 5**.

Se il valore passato non fosse un numero sarebbe un problema, quindi solo in quel caso lanciamo un’eccezione.

```
1  1  function multiplication($int) {
   2   if (!is_int($int)) {
   3     throw new Exception('Is not a number');
   4   }
   5   return $int*5;
   6  }
```

Basta die()!
Quando creiamo dei controlli per la produzione lanciamo delle eccezioni in caso di errore.

try and catch

Ok, ma come faccio ad intercettare un'eccezione?

Ora abbiamo la nostra funzione PHP che nel caso riscontri un errore lancia un'eccezione.

Come possiamo quindi chiamare la nostra funzione **senza rischiare di essere bloccati dalla nostra eccezione?**

Ci viene in aiuto il **try** and **catch**.

```php
try {
    echo multiplication('ciao');
} catch (Exception $e) {
    echo 'Eccezione: ' . $e->getMessage();
}
```

Try interroga la nostra funzione e **catch** si occupa di intercettare il nostro messaggio in caso di errore.

try and catch

In questo caso cosa comparirà a schermo?

```
1 1 try {
  2  echo multiplication('ciao');
  3 } catch (Exception $e) {
  4  echo 'Eccezione: ' . $e->getMessage();
  5 }
```

try and catch

Abbiamo quindi lanciato la nostra prima eccezione inserendola in una nostra funzione.

Nel **catch** utilizziamo la classe **Exception** a cui associamo la variabile **$e**.
La classe **Exception** ha diversi metodi

(tanti a dire il vero, per quello spesso sarà impossibile fare var_dump() di $e),

tra cui quello che abbiamo utilizzato noi, cioè **getMessage()** che ritornerà il messaggio, da noi definito, **Is not a number**.

LIVE CODING

ESERCITAZIONE