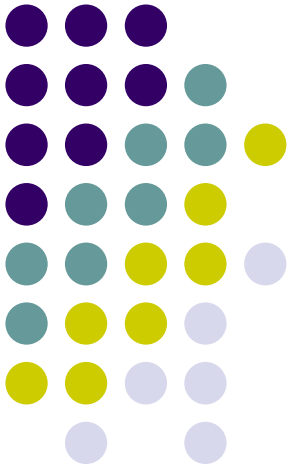
PHP

POO

Héritage



Franck LAMY – 2023/2024 – BTS SIO2

Objectifs

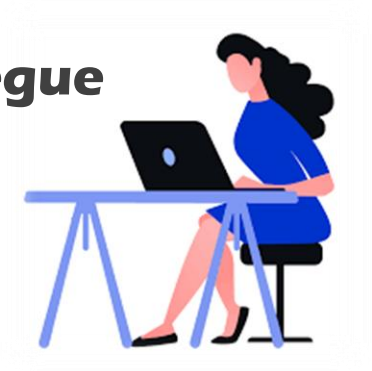
- **Introduire la notion d'héritage**
- **Présenter et mettre en œuvre les principes de l'héritage**

L'équipe de développement

Votre chef de projet

Un collègue

Une collègue



C'est vous !

Développeur débutant

Nouveau projet

Nous allons démarrer un nouveau projet. Et tu fais partie de l'équipe !
***Dans un premier temps, je souhaiterais que tu implémentes la classe* EMPLOYE**

Employe
-prenom : string -nom : string -age : int
+presenter

OK BOSS !
Je vous fais ça dans la journée !

La classe Employe

Employe
-prenom : string -nom : string -age : int
+presenter

Quelque jours plus tard...

Le projet évolue ! Je souhaite que l'on intègre le concept de Patron dans l'application.
Cela ne devrait pas te poser trop de problème !

OK BOSS !
Mais qu'est-ce qui caractérise un patron ?

La classe Patron

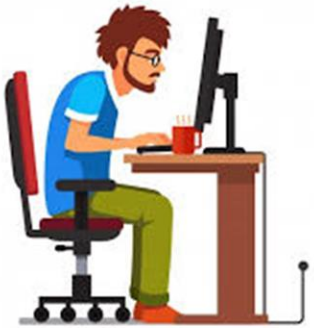
Voici la classe Patron !

Patron

-prenom
-nom
-age
-voiture

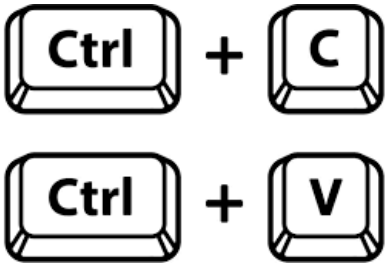
presenter
deplacer

OK ! Je vous fais ça rapidement !



Après réflexion…

Employe
-prenom -nom -age
presenter

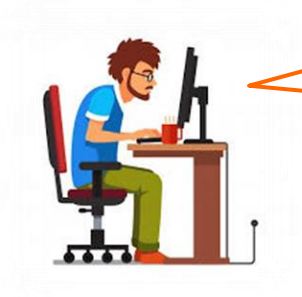


Patron
-prenom -nom -age -voiture
presenter deplacer

Après quelques heures de travail...

```php
class Patron
{
    private string $nom;
    private string $prenom;
    private int $age;
    private string $voiture;

    public function __construct(string $nom, string $prenom, int $age, string $voiture)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
        $this->age = $age;
        $this->voiture = $voiture;
    }

    // Getters et Setters

    public function presenter()
    {
        dump("Bonjour, je m'appelle $this->prenom $this->nom et j'ai $this->age ans");
    }

    public function deplacer() {
        dump("Je me déplace en $this->voiture");
    }
}
```

Le moment de la présentation...



Voila BOSS ! J'ai terminé l'implémentation de la classe Patron ! TROP FACILE !!!!!!!

C'est ça ta classe Patron ! Tu ne respectes vraiment AUCUN PRINCIPE !

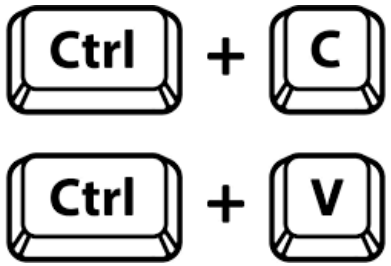
Quel principe ?

DRY: Don`t Repeat Yourself

Duplication de code ...

Tu as fait la seule chose que tout développeur doit FUIRE !

DUPLICATION DE CODE !



Copier/Coller du code ...

Je te montre !

```php
class Patron
{
    private string $nom;
    private string $prenom;
    private int $age;
    private string $voiture;

    public function __construct(string $nom, string $prenom, int $age, string $voiture)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
        $this->age = $age;
        $this->voiture = $voiture;
    }

    // Getters et Setters

    public function presenter()
    {
        dump("Bonjour, je m'appelle $this->prenom $this->nom et j'ai $this->age ans");
    }

    public function deplacer() {
        dump("Je me déplace en $this->voiture");
    }
}
```

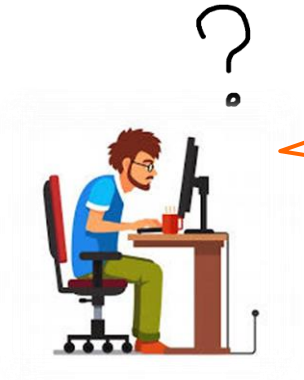
repeat

repeat

repeat

OUI ! J'ai effectivement dupliqué le code !

Mais alors …



Comment aurais-je pu éviter cette duplication de code

Ton professeur ne t'as donc jamais enseigné la notion d'HERITAGE ?

Euh non ! Je n'en ai aucun souvenir

Le professeur un an avant ...

Nous allons aujourd'hui abordé un nouveau concept de la POO

l'héritage

Votre prof

C'est pas moi !

C'est vous !

Etudiant en BTS

Un bon chef de projet ...

*Je vais donc t'expliquer en quoi consiste l'**héritage** et les avantages à l'utiliser dans tes futurs développement !*

Héritage

Mécanisme en POO permettant de créer une classe à partir d'une autre classe

Réutiliser le code d'une classe existante afin d'éviter la duplication de code

Héritage

On va donc **créer** la classe **Patron** **à partir** de la classe **Employe**

La classe Patron hérite de la classe Employe

Tous les **attributs** et les **méthodes** de la classe **Employe** vont **être hérités** dans la classe **Patron**

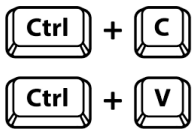


Diagramme de classes

Employe

-prenom
-nom
-age

+presenter()

Hérite de

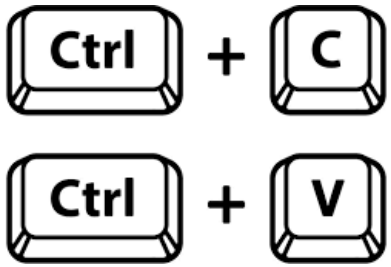
Patron

-voiture

+deplacer()

-prenom
-nom
-age

+presenter



Héritage en PHP

Employe

-prenom
-nom
-age

+presenter()

Hérite de

Patron

-voiture

+deplacer()

```
class Patron extends Employe
{
   ...
}
```

DUPLICATION DE CODE

Un peu de vocabulaire

Employe

-prenom
-nom
-age

+presenter()

Hérite de

Patron

-voiture

+deplacer()

La classe Patron **spécialise** le classe Employe

C'est une **spécialisation** de la classe Employe

La classe Patron **étend** la classe Employe

C'est une **extension** de la classe Employe

Spécificités de la classe Patron

Un peu de vocabulaire

Employe

-prenom
-nom
-age

+presenter()

Hérite de

Patron

-voiture

+deplacer()

La classe **Employe** **généralise** la classe **Patron**

C'est une **généralisation** de la classe **Patron**

Spécificités de la classe **Patron**

Un peu de vocabulaire

Un patron

EST UN employé

avec des **spécificités**

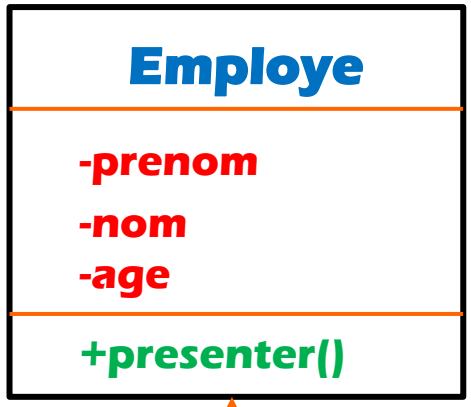
Attributs spécifiques

Il possède une voiture

Il se déplace

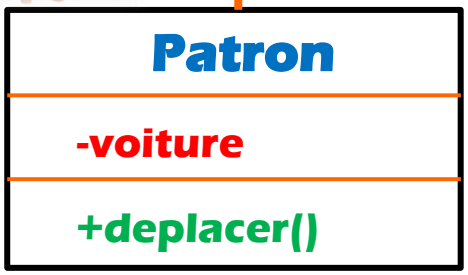
Méthodes spécifiques

Un peu de vocabulaire



Super-classe
Classe parente
Classe mère

Hérite de



Sous-classe
Classe fille

Cas du constructeur

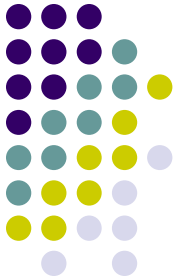
```php
public function __construct(string $nom, string $prenom, int $age, string $voiture)
{
    $this->nom = $nom;
    $this->prenom = $prenom;
    $this->age = $age;
    $this->voiture = $voiture;
}
```

repeat

Code du constructeur de la classe Employe : classe parente

```php
public function __construct(string $nom, string $prenom, int $age, string $voiture)
{
    parent::__construct($nom,$prenom,$age);
    $this->voiture = $voiture;
}
```

Test



\tests\PatronTest.php

```php
<?php

require "vendor/autoload.php";

use App\Patron;

$patron = new Patron("MARTIN","Jean",40,"BMW");
$patron->presenter(); // méthode héritée de Employe
$patron->deplacer();
```

Redéfinition de méthode

Ils se présentent de la même manière !

On souhaite que le patron se présente différemment !

Redéfinition de méthode

Pour ce faire, on va **spécialiser** dans la classe **Patron** la manière dont un patron va se présenter

Spécialiser une méthode >>

Redéfinir une méthode héritée, dans une sous-classe, afin d'adapter son implémentation

Redéfinition de méthode

La méthode héritée de

```php
public function presenter()
{
    dump("Bonjour, je m'appelle $this->prenom $this->nom et j'ai $this->age ans");
}
```

La méthode redéfinie

```php
public function presenter()
{
    dump("Bonjour, je suis $this->prenom $this->nom, j'ai $this->age ans et je suis le boss !");
}
```

Redéfinir une méthode implique que l'on conserve la même **signature**

Redéfinition de méthode

```
$patron = new Patron("MARTIN","Jean",40,"BMW");
$patron->presenter();
```

C'est toujours la méthode redéfinie que sera appelée sur une instance de la sous-classe

Test

```php
<?php

require "vendor/autoload.php";

use App\Patron;

$patron = new Patron("MARTIN","Jean",40,"BMW");
$patron->presenter(); // méthode redéfinie
$patron->deplacer();
```

Les attributs hérités

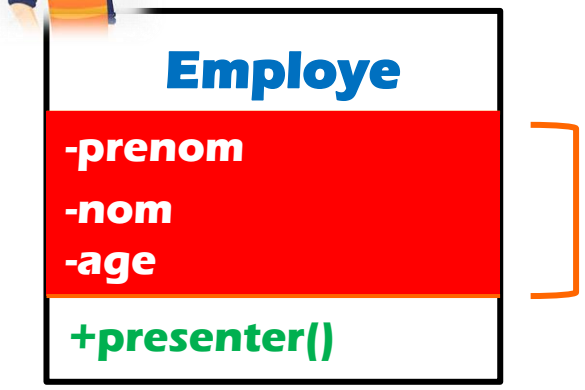

```
Notice: Undefined property: App\Patron::$prenom
```

```
Notice: Undefined property: App\Patron::$nom
```

```
Notice: Undefined property: App\Patron::$age
```

Problème lié à l'accessibilité des attributs hérités

Accessibilité des attributs



Les attributs sont uniquement accessibles dans les méthodes de la classe Employe et donc inaccessibles dans n'importe quelle autre classe

Accessibilité des attributs

Employe

-prenom
-nom
-age

presenter

Hérite de

-prenom
-nom
-age

Patron

-voiture

deplacer

Les attributs sont bien hérités mais inaccessibles

Solution 1

Pour **accéder** dans une **sous-classe** aux **attributs hérités**, on peut utiliser les **getters/setters hérités** de la classe **super-classe**

```
public function presenter()
{
   dump("Bonjour, je suis {$this->getPrenom()} {$this->getNom()}, j'ai {$this->getAge()} ans et je suis le patron");
}
```

La syntaxe {getter} est spécifique à la fonction dump()

Solution 2

Modifier l'**accessibilité** des **attributs** dans la **super-classe** afin d'y **accéder directement dans la sous-classes**

Employe

#prenom
#nom
#age

presenter

statut

PROTECTED

```
class Employe {
    protected string $nom;
    protected string $prenom;
    protected int $age;
    ...
}
```

Solution 2

```
class Employe {
    protected string $nom;
    protected string $prenom;
    protected int $age;
    ...
}
```

```
public function presenter()
{
    dump("Bonjour, je suis $this->prenom $this->nom, j'ai $this->age ans et je suis le patron");
}
```

Accessibilité des attributs

PRIVATE — Accessible uniquement dans les méthodes de la classe

PROTECTED — Accessible uniquement dans les méthodes de la classe et de ses sous-classes

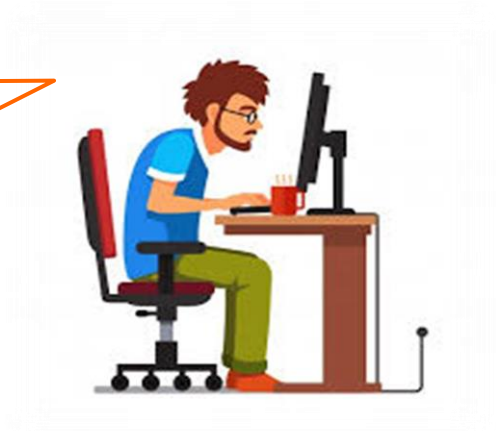
PUBLIC — Accessible dans les méthodes de toutes les classes

Principe d'encapsulation

Au final…

Voila ! J'espère que tu as compris la notion d'héritage et les avantages que cela procurent au niveau du code !

Oui BOSS, j'ai bien compris ! En effet c'est un concept puissant permettant d'éviter la duplication de code !



Quelques jours plus tard…

Le projet évolue encore ! Tu dois maintenant intégrer dans le projet le concept de chef de service! Tu sais comment procéder !!!!!!!!!

ChefService
-prenom -nom -age -service
+presenter()

Pas de soucis BOSS ! Je vous fais ça en 10' !

Quelques précisions …

Un chef de service se présente de la manière suivante :

Bonjour, je suis Alain DURAND, j'ai 38 ans et je suis le chef du service Informatique