Support PHP OO

Objectifs:

- Améliorer la maintenance d'un site et favoriser le travail en groupe

- Séparer les langages (PHP, SQL et HTML) pour une plu grande clarté et gestion du code.

Inconvénients:

- N'apporte rien de plus, techniquement, en terme d'affichage, que le PHP Procédural

- Son approche est moins intuitive que le procédural. Ce dernier suit une logique séquentielle, ligne par ligne, alors que le PHP OO fait interagir des objets entre eux

C'est moins évident pour l 'esprit humain. Peu sont à l'aise dès le début avec cette démarche

- Légère perte de performances (compensable par ailleurs)

Avantages:

- Un modèle de projet peut-être récupérable pour un autre lui ressemblant. C'est une base commune qui sera modifiable, améliorable au fur et à mesure.

C'est ce que font les CMS ou un framework comme Symfony. Ils utilisent une base de départ, minimale et optimale pour ensuite la faire évoluer selon le nouveau projet

- Du fait de la séparation des langages, un intégrateur (spécialisé en HTML) aura moins de mal à gérer sa tache de travail que s'il devait coder au milieu de lignes PHP ou SQL. Il pourra désormais travailler dans des fichiers destinés aux vues (affichage) où sera écrit seulement le minimum de code PHP.

De la même manière, un développeur PHP prenant le projet en cours, aura moins de mal à analyser le code PHP s'il est condensé dans une partie du projet, plutôt que réparti dans tous les fichiers.

Favorise donc le travail à plusieurs

- Le principe d'encapsulation va permettre aussi de protéger (masquer) le code.

Ne pourront le modifier que ceux qui en ont le droit (les capacités). Permet de se prémunir d'erreurs involontaires.

La complexité du code (tout le traitement PHP pour exécuter une méthode/fonction) va être rangée dans des fichiers spécifiques, pour ensuite faire appel à elles dans d'autres fichiers, avec du code plus simple, beaucoup plus lisible. Plutôt que d'avoir du traitement PHP dans tous les fichiers, ne sachant pas à quelle ligne il faut aller pour le comprendre et le modifier lors des opérations de maintenance, mises a jour du code. C'est ce que l'on a déjà fait un peu avec les dossiers init.

- Le PHP OO permet aussi de documenter son code, avec une syntaxe et des informations conventionnées, pour permettre aux autres développeurs de récupérer ses mêmes informations qui leur seront nécessaires

- En plus du principe d'encapsulation, on va bénéficier d'autres fonctionnalités telles que l'héritage, l'abstraction, qui nous permettent de travailler plus vite, plus sure, avec aussi une meilleure maintenance.

- Cette nouvelle manière de travailler (séparant les langages, l'encapsulation etc.) sera commune aux autres langages orientés objet. Le temps d'adaptation sera plus rapide car la logique sera la même.

Conclusion:

En procédural, moins de fichiers, mais plus de code dedans. C'est l'inverse en objet. Plus de fichiers car moins de code dedans

Pour un petit projet, site vitrine avec back-office minimaliste, le construire en orienté objet sera concevable. Par contre, dans le cas de projets plus ambitieux qui va nécessiter une équipe de devs, cela sera beaucoup plus délicat. Il faut lui préférer l'orienté objet.

Le PHP OO est une méthodologie de travail, une organisation, destinée non pas à améliorer l'affichage pour l'utilisateur, mais pour optimiser le développement d'un site.

1 Classe / Objet / Instance et visibilité

Je crée un premier dossier 01-classe-objet-instance-visibilite

A l'intérieur un fichier: Panier.class.php

Je ne suis pas obligé de mettre une majuscule à Panier. Cela fonctionnera très bien sans (contrairement à la syntaxe Java). Je ne suis pas non plus obligé d'ajouter après; .class. Ce n'est qu'une convention pour repérer plus vite un fichier qui va contenir une classe

Je code, puis j'explique

```php
<?php

class Panier{

  public $nbProduits;

  public function ajouterProduit(){
```

```
    return "Le produit à bien été ajouté <br>";
  }

  protected function retirerProduit(){
    return "Le produit a bien été retiré <br>";
  }

  private function afficherProduit(){
    return "Voici tous les produits de votre panier<br>";
  }
}
```

En tout premier, je déclare ma classe. Son nom doit être identique au nom du fichier (sans .class), débutant par une majuscule (non obligatoire, c'est une convention)

J'initialise un attribut $nbProduits avec la visibilité public.

S'ensuivent trois méthodes avec trois niveaux de visibilité différents, chacune retournant une chaîne de caractères

En l'état actuelle, j'ai donné les caractéristiques de ma classe. Mais, je ne vais pas pouvoir les exploiter. Si je tente un echo de $nbProduits, j'aurais un undefined nbProduit sur mon navigateur.

Je dois pour cela instancier ma classe. Je dois créer un objet de cette classe.

Je code ceci, à l'extérieur de ma classe

```
$panier = new Panier;

echo '<pre>'; var_dump($panier); echo '</pre>';
echo '<pre>'; print_r(get_class_methods($panier)); echo '</pre>';
```

Le var_dump m'indique que

```
object(Panier)#1 (1) {
  ["nbProduits"]=>
  NULL
}
```

C'est un objet de ma classe Panier. Qu'il a pour identifiant 1. Qu'il a une propriété (nbProduits) à laquelle aucune valeur n'a été affectée (NULL)

Le print_r m'indique que

```
Array
(
    [0] => ajouterProduit
)
```

La méthode ajouterProduit est présente, mais il ne peut voir les deux autres car elles sont protected et private

Si je veux donner une valeur à $nbProduits, je peux écrire cela

```
$panier->nbProduits = 5;
```

Remarque: lorsque je pointe avec la flèche → , je n'écris pas le signe $ devant mon attribut

Je refais un var_dump

```
echo '<pre>'; var_dump($panier); echo '</pre>';
```

Désormais, sa valeur ne sera plus NULL

```
object(Panier)#1 (1) {
  ["nbProduits"]=>
  int(5)
}
```

Si je veux un affichage plus conventionnel du nombre de produit, j'écris ceci

```
echo "Vous avez actuellement dans votre panier " . $panier->nbProduits . " articles
<br>";
```

Remarque; en affectant la valeur de 5 à $nbProduits, je modifie l'objet et non la classe!

De la même manière, je vais exécuter la fonction ajouterProduit() pour afficher son contenu

```
echo $panier->ajouterProduit() . '<br>';
```

Tout cela se fait de manière simple car mon attribut $nbProduits et ma méthode ajouterProduit() ont un niveau de visibilité public.

Un niveau de visibilité public signifie que je peux l'utiliser à l'intérieur de ma classe (pourquoi pas dans la fonction ajouterProduit().

A l'extérieur de ma classe, comme lors de la création de l'objet $panier.

Je pourrais même l'utiliser dans une classe qui hérite de celle ci

```
class Produit extends Panier{
  // code
}
```

La visibilité public est le niveau d'accessibilité le plus permissif, le plus bas.

Les niveaux de visibilité / accessibilité servent à protéger, par exemple, de l'envoi de données.

Ce sera le cas pour envoyer des données en BDD via un formulaire. Je ne pourrais rien envoyer tant que le contenu ne sera pas vérifié.

Je tente la même syntaxe pour retirerProduit()

```
echo $panier->retirerProduit() . '<br>';
```

Fatal error: Uncaught Error: Call to protected method Panier::retirerProduit() from global scope

Erreur fatale car j'essaie d'appeler une méthode à l'accessibilité protected en dehors de ma classe

L’accessibilité protected rend ma fonction disponible à l’intérieur de ma classe

```
class Panier{

  protected function retirerProduit(){
    return "Le produit a bien été retiré <br>";
  }

  public function test(){
    return $this->retirerProduit();
  }

}
```

Elle est aussi accessible dans une classe héritière

```
class Produit extends Panier{
  public function test(){
    return $this->retirerProduit();
  }
}
```

La différence avec public, c’est que protected n’est pas disponible en dehors de la classe

Dernier test avec ma méthode visibilité private

```
echo $panier->afficherProduit();
```

Fatal error: Uncaught Error: Call to private method Panier::afficherProduit() from global

Même chose, erreur fatale

Si je tente de l’utiliser à l’intérieur de ma classe, c’est bon

```
class Panier{

  private function afficherProduit(){
    return "Voici tous les produits de votre panier<br>";
  }

  public function test(){
    return $this->afficherProduit();
  }

}
```

mais pas pour une classe qui hérite

```
class Produit extends Panier{
  public function test(){
    return $this->afficherProduit();
  }
```

```
}
```

On utilise donc un niveau d'accessibilité plutôt qu'un autre selon que l'on veuille donner ou pas le droit de modifier la méthode, d'injecter une valeur etc.

De manière générale, le niveau public n'existe quasiment pas, très rarement. Tout sera en protected ou private.

Je peux créer un nouvel objet de ma classe (en fait autant que je veux)

```
$panier2 = new Panier;
```

Je fais un var_dump

```
echo '<pre>'; var_dump($panier2); echo '</pre>';
```

```
object(Panier)#2 (1) {
  ["nbProduits"]=>
  NULL
}
```

Celui ci a pour id, 2. La propriété $nbProduit n'a toujours aucune valeur qui lui a été affectée (c'est à la propriété de l'objet $panier à laquelle on a affecté la valeur de 5, pas à celle de la classe)

2 Getter / Setter / Constructeur / This

- 2.1 Getter et Setter

Second dossier nommé 02-getter-setter-constructeur-this.

A l'intérieur, un fichier nommé Membre.class.php

Je déclare la classe Membre avec à l'intérieur 2 propriétés, private, $prenom et $nom. Et une méthode, public, setPrenom(), qui attend un argument: $newPrenom

```
<?php

class Membre{

  private $prenom;
  private $nom;

  public function setPrenom($newPrenom){

  }

}
```

Dans la méthode setPrenom(), je code ceci puis j'explique

```
public function setPrenom($newPrenom){

  if(is_string($newPrenom)){
    $this->prenom = $newPrenom;
  }else{
    trigger_error('Cela ne correspond pas car ce n\'est pas une chaîne de caractères', E_USER_ERROR);
  }

}
```

Ma méthode setPrenom est ce que l'on appelle un Setter. Un setter sert établir un protocole. Il sert à donner un cadre à ce que peut accueillir une propriété. Il va servir de contrôleur de champs comme nous faisons auparavant avec if(!isset($_POST['prenom']) etc.

Remarque: setPrenom() n'est pas une fonction prédéfinie. J'aurai pu l'appeler prenom() ou verifPrenom(), cela n'a pas d'importance. Par contre, c'est une convention. Pour qu'un autre développeur comprenne rapidement que c'est un setter, on va débuter son nommage par set, suivi du champs à vérifier.

Cette fonction sera public car elle doit être accessible de l'extérieur de ma classe (sur un formulaire dans une vue). C'est elle qui va faire le travail de contrôle pour les deux propriétés que j'ai protégé en private.

Dans cette méthode, je dis que si le contenu que je reçois dans $newPrenom est bien une chaîne de caractères, alors, son contenu ira affecter la valeur de la propriété de l'objet en cours. Si ce n'est pas une chaîne de caractères, alors je génère un message d'erreur.

Exemple, je crée un objet membre de ma classe Membre et je donne en argument une valeur à setPrenom()

```
$membre = new Membre;
$membre->setPrenom('Aziz');
```

Ma méthode setPrenom() dans la classe Membre va vérifier si le contenu envoyé en argument est bien un string. Si oui, alors avec $this→ ce contenu va affecter la valeur de la propriété $prenom de l'objet en cours.

```
$this->prenom = $newPrenom;
```

Je dois utiliser cette syntaxe et non

```
$prenom = $newPrenom;
```

Cette dernière syntaxe n'aura aucun sens ou utilité. Elle est déclarée dans un espace local (celui de la méthode). Elle n'est pas équivalente à celle déclarée dans l'espace global de ma classe, avec visibilité private.

Je dois donc utiliser $this→ pour pointer vers l'objet en cours.

C'est équivalent à

```
$membre->prenom = 'Aziz';
```

Sauf que cela je ne peux pas le faire

Fatal error: Uncaught Error: Cannot access private property Membre::$prenom

Car j'ai volontairement mis ma propriété en private (pour contrôler ce qui va être injecté en valeur/contenu)

$this→ pointe toujours vers l'objet en cours

Je veux maintenant afficher le prénom de mon objet. Si j'écris ceci

```
echo $membre->prenom;
```

J'aurais à nouveau l'erreur de tout à l'heure.

Fatal error: Uncaught Error: Cannot access private property Membre::$prenom

Je vais devoir passer par un Getter (que je vais coder). C'est qui qui va me permettre de récupérer ce contenu. Un getter ne prend pas d'argument.

```
public function getPrenom(){
  return $this->prenom;
}
```

C'est cette méthode qui va me retourner grace à $this→ qui pointe vers la propriété de l'objet en cours

Et c'est cet echo qui va afficher la valeur de la propriété

```
echo $membre->getPrenom();
```

Je pourrai utiliser ce getter pour un autre cas de figure, pas juste pour afficher, mais pour une requête d'insertion

```
public function insertUser(){
  $ajouterUser = $pdo->prepare("INSERT INTO membre (prenom, nom) VALUES (". $this-
>getPrenom(). ",". $this-getnom() . ")  ");
}
```

Je crée à présent les setter et getter pour la propriété $nom

```
public function setNom($newNom){
```

```
  if(is_string($newNom)){
    $this->nom = $newNom;
  }else{
    echo 'Cela ne correspond pas car ce n\'est pas une chaîne de caractères';
  }

}

public function getNom(){
  return $this->nom;
}
```

Puis j'affiche

```
$membre->setNom('Tobbal');

echo 'Je suis ' . $membre->getPrenom() . ' ' . $membre->getNom() . '<br>';
```

L'équivalent en français de Getter est accesseur. Pour Setter, c'est mutateur

2.2 Constructeur

Dans le même dossier, je crée un nouveau fichier nommé Etudiant.class.php

Dedans, une propriété de visibilité private $prenom

```
<?php

class Etudiant{

  private $prenom;

}
```

Après cela je vais déclarer un constructeur, un setter et un getter. Je code puis j'explique

```
public function __construct($arg){
  echo "Durant l'instanciation, nous avons bien reçu l'information suivante: $arg
<br>";
  $this->setPrenom($arg);
}

public function setPrenom($arg){
  $this->prenom = $arg;
}

public function getPrenom(){
  return $this->prenom;
}
```

J’instancie ma classe Etudiant

```
$etudiant = new Etudiant;
```

Mon navigateur va me générer une erreur.

Fatal error: Uncaught ArgumentCountError: Too few arguments to function Etudiant::__construct(), 0 passed

Il me signale que mon constructeur attend un argument, je n’en ai aucun (lorsque je crée mon objet $etudiant).

En fait, du fait que j’ai déclaré le __construct(), j’aurai du coder ceci lors de l’instanciation de ma classe

```
$etudiant = new Etudiant('Aziz');
```

Un constructeur est une méthode magique qui s’auto-exécute au moment où l’on instancie la classe.

Un constructeur permet une automatisation des taches.

Cette nouvelle syntaxe ci dessus rappelle la syntaxe pour se connecter à sa BDD

```
$pdo = new PDO('mysql:host=localhost');
```

Cela veut juste dire que dans la classe PDO il y a un constructeur, et que obligatoirement si je crée un objet de cette classe, je dois lui donner un argument, sinon, erreur!

Le constructeur grâce à

```
$this->setPrenom($arg);
```

Permet d’injecter au setter la valeur contenue dans $arg (qui sera ensuite affectée à $prenom dans le setter et récupérable avec le getter)

Pour afficher, de manière classique, j’écrirai

```
echo 'Je suis ' . $etudiant->getPrenom() . '<br>';
```

Désormais, avec une syntaxe plus rapide, je peux affecter des valeurs aux propriétés de ma classe.

Un constructeur permet d’automatiser certaines taches. On peut le considérer comme un équivalent du fichier init.php

Remarque: du fait qu’il s’auto-exécute, il ne peut y avoir qu’un seul constructeur par classe

Si j’avais voulu donner un prénom et un nom à mon étudiant, voici le code (en donnant deux arguments au constructeur)

```
<?php
```

```php
class Etudiant{

  private $prenom;
  private $nom;

  public function __construct($newPrenom, $newNom){
    echo "Durant l'instanciation, nous avons bien reçu l'information suivante: 
$newPrenom et  $newNom<br>";
    $this->setPrenom($newPrenom);
    $this->setNom($newNom);
  }

  public function setPrenom($newPrenom){
    $this->prenom = $newPrenom;
  }

  public function getPrenom(){
    return $this->prenom;
  }

  public function setNom($newNom){
    $this->nom = $newNom;
  }

  public function getNom(){
    return $this->nom;
  }

}

$etudiant = new Etudiant('Aziz', 'Tobbal');

echo 'Je suis ' . $etudiant->getPrenom() . ' ' . $etudiant->getNom() . '<br>';
```

3 Constante / Static et Self

Je crée un nouveau dossier nommé 03-constante-static-self avec à l'intérieur un fichier Maison.class.php

Je déclare dedans une première propriété, public, $couleur. Et une seconde, toujours public mais static, $espaceTerrain (cela va nous servir à automatiser certaines taches, comme on va le voir progressivement)

```php
<?php

class Maison{
```

```
  public $couleur = 'blanc';
  public static $espaceTerrain = '500m²';

}
```

Une propriété statique est une propriété qui appartient à la classe. Inversement, $couleur appartient à l'objet.

Concrètement, quelle différence ?

Dans le cas de couleur, par défaut, tous mes objets auront la couleur blanc.

Si je veux modifier cet état, je devrais le faire lors de chaque instanciation

Dans le premier cas, je ne fais que créer un objet de ma classe puis j'affiche son contenu avec un var_dump()

```
$maison = new Maison;
echo '<pre>'; var_dump($maison); echo '</pre>';
```

```
object(Maison)#1 (1) {
  ["couleur"]=>
  string(5) "blanc"
}
```

Dans le second cas, je veux que la couleur ne soit plus blanc mais bleu

```
$maison2 = new Maison;
$maison2->couleur = 'bleu';
echo '<pre>'; var_dump($maison2); echo '</pre>';
```

```
object(Maison)#2 (1) {
  ["couleur"]=>
  string(4) "bleu"
}
```

Troisième cas, semblable au premier, je ne fais que créer une instance de ma classe

```
$maison3 = new Maison;
echo '<pre>'; var_dump($maison3); echo '</pre>';
```

La couleur redevient blanc, par défaut.

```
object(Maison)#3 (1) {
  ["couleur"]=>
  string(5) "blanc"
}
```

Cela veut dire, que si au fur et a mesure de mon code je ne veux plus de la couleur blanc, je devrais pour chaque nouvel objet crée, le préciser comme lors du second exemple.

Remarque: ça ne serait pas pertinent de changer la valeur d'origine, car je la perdrai pour mes premiers objets, alors que eux ont besoin de couleur = blanc.

Dans le cas d'une propriété statique, cela ne sera pas le cas. Dans la mesure où elle appartient à la classe et non l'objet, je pourrai avec une syntaxe appropriée modifier sa valeur (qui écrasera la précédente), qui impactera tous les nouveaux objets, sans modifier celle des objets précédents.

Nous allons voir cela progressivement.

Voici la syntaxe pour pouvoir afficher la valeur de mon attribut statique

```
echo "L'espace terrain par défaut est de " . Maison::$espaceTerrain . "<br>";
```

Avec une propriété statique, je n'ai pas besoin d'instancier ma classe. Elle appartient à la classe et non à un des objets.

Cette syntaxe rappelle PDO::FETCH_ASSOC. PDO qui est une classe, et pour faire appel à la méthode FETCH_ASSOC, je n'avais pas besoin d'instancier PDO. Ici, je retrouve la même syntaxe.

Pour modifier la valeur d'une propriété statique

```
Maison::$espaceTerrain = '1000m²';
echo "L'espace terrain par défaut est de " . Maison::$espaceTerrain . "<br>";
```

A présent, j'ajoute une nouvelle propriété, private cette fois, toujours static

```
public $couleur = 'blanc';
public static $espaceTerrain = '500m²';
private static $nbPieces = 7;
```

Cette fois, j'ai besoin d'afficher son contenu. Du fait qu'elle soit private, je vais devoir coder son Getter !

```
public static function getNbPieces(){
  return self::$nbPieces;
}
```

Cette méthode, public (car j'aurai besoin de l'afficher en dehors de la classe, dans une vue) sera aussi static (même si ce n'est pas obligé).

Mais pour une raison logique, si le getter est déclaré pour une propriété qui appartient à la classe, alors la méthode appartiendra aussi à la classe et non à l'objet.

Cette fois, je n'utilise pas $this→ car ce dernier pointe vers l'objet en cours. Mais, comme $nbPieces appartient à la classe et non à une instance de la classe, alors, je vais devoir utiliser une autre syntaxe => self:: (en faisant un hover sur self dans le code, je peux voir qu'il représente la classe Maison)

Voici la syntaxe pour afficher le nombre de pièces

```
echo "Le nombre de pièces par défaut est: " . Maison::getNbPieces() . "<br>";
```

Si elle n'est pas static, c'est cette syntaxe qu'il faudra utiliser

```
echo $maison3->getNbPieces();
```

Mais pour des raisons de cohérence, si la propriété est statique, alors son getter le sera aussi

---

Quelques syntaxes pour tester notre script

```
echo $maison3->espaceTerrain;
```

Va générer une erreur car je tente d'utiliser dans mon objet une propriété qui ne lui appartient pas

```
echo Maison::$couleur;
```

Va aussi générer une erreur car je tente d'appeler via ma classe une propriété qui ne lui appartient pas non plus

Autre test, qui a priori ne devrait pas non plus fonctionner

```
echo $maison3->getNbPieces();
```

Ne va pourtant pas générer d'erreur, alors que cela aurait du (appeler via l'objet une méthode qui ne lui appartient pas)

Dernier test, toujours normalement mauvais

```
echo $maison3::$espaceTerrain . '<br>';
```

Fonctionne aussi, même si c'est incohérent.

C'est deux derniers exemples montrent la permissivité de PHP.

Remarque: Lorsque l'on passe par la classe pour appeler une propriété, il faut le signe $

Inversement, si c'est un objet, il ne faudra pas le signe $

En procédural, pour déclarer une constante il fallait ecrire

```
define('URL','sa valeur');
```

En PHP OO, il faut écrire

```
const HAUTEUR = '10m';
```

On utilise le mot clé const + son nom en Majuscules (convention).

Cette constante est automatiquement la propriété de la classe.

Pour l'afficher, il me faudra donc passer par la classe

```
echo 'Hauteur sous-plafond: ' . Maison::HAUTEUR . '<br>';
```

Conclusion: comme le constructeur, faire appel à une propriété static d'automatiser certaines taches

C'est ce qui déterminera notre choix au moment de sa déclaration.

Pour mieux saisir cette notion d'automaticité, regarder ce tutoriel de Grafikart

Cette notion de static, et savoir quand l'utiliser ou non n'est pas évidente. C'est la pratique qui progressivement permettra de mieux se décider.

4 Héritage

Nouveau dossier 04-heritage avec à l'intérieur un fichier Personnage.class.php

Cette classe va avoir une méthode, protected, nommée deplacement()

```php
<?php

class Personnage{

  protected function deplacement(){
    return 'je me déplace très vite';
  }

}
```

J'ajoute une méthode public, saut()

```php
public function saut(){
  return 'je saute très haut';
}
```

Dans ce même fichier, en dessous, je déclare une nouvelle classe nommée Mario

```php
class Mario{

}
```

La classe Mario a en fait les mêmes caractéristiques que la classe Personnage.

Elle possède les deux mêmes méthodes.

Pour ne pas avoir à les coder à nouveau, je peux utiliser un mot clé, extends, qui va me permettre d'en hériter

Remarque: une classe ne peut hériter que d'une seule autre classe, pas plusieurs

```php
class Mario extends Personnage{

}
```

Je lui ajoute une méthode propre à elle quiSuisJe()

```
class Mario extends Personnage{

  public function quiSuisJe(){
    // code
  }

}
```

Enfin, j'instancie cette classe Marion, puis je fais un print_r() pour vérifier si j'hérite bien de la classe Personnage

```
$mario = new Mario;
echo '<pre>'; print_r(get_class_methods($mario)); echo '</pre>';
```

J'hérite bien, je peux donc les utiliser dans ma méthode quiSuisJe()

```
public function quiSuisJe(){
  return "Je suis Mario, " . $this->deplacement() . ' et ' . $this->saut() .
'<br>';
}
```

Puis j'affiche

```
echo $mario->quiSuisJe();
```

Ça fonctionne bien et je remarque aussi que si la méthode deplacement n'apparaît pas dans le print_r() du fait qu'elle soit protected, j'en hérite quand même et je peux l'exécuter. Si elle avait été private, je n' en aurais pas été hérité

J'ajoute une nouvelle classe dans ce fichier, qui aussi hérite de Personnage

```
class Bowser extends Personnage{

  public function quiSuisJe(){
      return 'Je suis Bowser et ' . $this->deplacement() . '<br>';
  }

}
```

Je vais déclarer la fonction saut() (qui existe dans la classe Personnage), en lui modifiant son comportement, et je vais voir le résultat

```
public function saut(){
  return 'je ne saute pas très haut';
}
```

Je récupère en fait le nouveau résultat. La précédente valeur a été écrasée. On dira que j'ai surchargé la méthode. C'est un besoin que je pourrai avoir.

J'hérite de tout, mais je peux l'ajuster selon un cas précis

Second exemple d'héritage

Dans le même dossier je crée un fichier nommé heritage_sens.php.

Avec ceci à l'intérieur

```php
<?php

class A{

  public function test1(){
    return 'j\'affiche test1';
  }

}

class B extends A{

  public function test2(){
    return 'j\'affiche test2';
  }

}

class C extends B{

  public function test3(){
    return 'j\'affiche test3';
  }

}
```

Je veux savoir si C va hériter de A ?

Pour cela je vais instancier ma classe C et faire un get_class_methods() de l'objet $c

```php
$c = new C;
echo '<pre>'; print_r(get_class_methods($c)); echo '</pre>';
echo $c->test1() . '<br>';
echo $c->test2() . '<br>';
echo $c->test3() . '<br>';
```

Ça fonctionne, même s'il n'y a pas de lien direct dans le code. Par répercussion, C hérite de A

5 Exercice

Créer dossier nommé 05-exercice. A l'intérieur un fichier nommé exercice.php

Dans ce fichier, déclarer une première classe Voiture. Elle aura une propriété nommée

$litresEssence. Coder son setter (pour ajouter de l'essence dans le véhicule) et getter (pour afficher le nombre de litres)

Dans ce même fichier, déclarer une seconde classe nommée Pompe. Elle aura les mêmes propriétés et méthodes, avec une méthode en plus, donnerLitresEssence() qui servira à donner de l'essence à la voiture

```
/*

-----------------------------
|          Voiture           |
-----------------------------
|        $litresEssence      |
-----------------------------
|       setLitresEssence     |
-----------------------------
|      getLitresEssence      |
-----------------------------



-----------------------------
|          Pompe             |
-----------------------------
|        $litresEssence      |
-----------------------------
|       setLitresEssence     |
-----------------------------
|      getLitresEssence      |
-----------------------------
|     donnerLitresEssence    |
-----------------------------

1 Créer voiture1
2 Lui donner de l'essence (5 litres)
3 Afficher cette quantité
4 Créer pompe1
5 Lui donner de l'essence (500 litres)
6 Afficher cette quantité
7 La pompe donne 50 litres à la voiture
8 Afficher la nouvelle quantité d'essence dans la voiture
9 Afficher la nouvelle quantité d'essence dans la pompe
10 Faire en sorte que le réservoir de la voiture ne puisse contenir plus de 50
litres

*/
```

Voici le code jusqu'à l'étape 6

```php
<?php

class Voiture{

  private $litresEssence;

  public function setLitresEssence($litres){
    $this->litresEssence = $litres;
  }

  public function getLitresEssence(){
    return $this->litresEssence;
  }

}

class Pompe extends Voiture{

  public function donnerLitresEssence(){

  }

}

$voiture1 = new Voiture;
$voiture1->setLitresEssence(5);
echo 'Mon véhicule a dans son réservoir ' . $voiture1->getLitresEssence() . ' litres d\'essence<br>';

$pompe1 = new Pompe;
$pompe1->setLitresEssence(500);
echo 'Ma pompe a dans son réservoir ' . $pompe1->getLitresEssence() . ' litres d\'essence<br>';
```

A partir de maintenant, étape 7, je vais devoir relier mon objet pompe avec mon objet voiture

PHP me permet d'imposer un objet de ma classe Voiture à la classe Pompe (indépendamment du fait qu'elle en hérite...faire le test sans le extends) en donnant ceci en argument à la méthode donnerLitresEssence()

```php
public function donnerLitresEssence(Voiture $vehicule){

}
```

Remarque, j'ai écris $vehicule, mais j'aurais pu ecrire $v ou autre

En dehors de ma classe, je relie l'objet pompe à l'objet voiture crée

```php
$pompe1->donnerLitresEssence($voiture1);
```

$voiture1 est donné en argument à la méthode. C'est cet objet de ma class Voiture que je viens d'imposer à la classe Pompe.

On appelle cela une injection de dépendance

Je peux vérifier avec un var_dump() ce que contient voiture1

```php
public function donnerLitresEssence(Voiture $vehicule){
  echo '<pre>'; var_dump($vehicule); echo '</pre>';
}
```

je récupère bien dans $vehicule ce que contient $voiture1. Maintenant que je sais que les deux objets sont reliés, je vais mettre en place le calcul qui permet de donner 50 litres d'essence à la voiture, sans dépasser cette quantité (50 litres)

```php
public function donnerLitresEssence(Voiture $vehicule){
    $vehicule->setLitresEssence($vehicule->getLitresEssence() + (50 - $vehicule-
>getLitresEssence()));
    echo '<pre>'; var_dump($vehicule); echo '</pre>';
  }
```

Tout d'abord, je pointe avec mon objet sur son setter pour pouvoir donner/modifier la quantité d'essence. Ensuite, je dois vérifier quelle quantité d'essence existe déjà dans mon réservoir

Je peux l'avoir grace au getter qui me la récupère. Une fois connue, j'envoie 50 litres, moins la quantité déjà existante.

Voici le calcul pour le restant dans le réservoir de la pompe.

Par contre je dois l'écrire avant le calcul qui donne l'essence à la voiture (sinon cela donnera 50 litres envoyés – 50 litres dans le réservoir...je dois avoir la quantité initiale dans le réservoir, donc je l'écris avant)

```php
public function donnerLitresEssence(Voiture $vehicule){
    $this->setLitresEssence($this->getLitresEssence() - (50 - $vehicule-
>getLitresEssence()));

    $vehicule->setLitresEssence($vehicule->getLitresEssence() + (50 - $vehicule-
>getLitresEssence()));
    echo '<pre>'; var_dump($vehicule); echo '</pre>';
  }
```

$this→ représente l'objet en cours pompe et je pointe vers setLitresEssence pour donner/modifier la quantité. Je récupère avec le getLitresEssence la quantité de départ, à laquelle je soustrais 50 litres, moins ce qu'il y avait déjà dans le réservoir du véhicule.

6 Surcharge / Abstraction / Finalisation / Trait

Nouveau dossier, 06-surcharge-abstraction-finalisation-trait.

6-1 Surcharge

A l'intérieur, un fichier nommé surcharge.php, avec une classe A qui contient, qlq propriétés et une méthode basique.

```php
<?php

class A{

  protected $nombre1;
  protected $nombre2;
  protected $nombre3;

  public function calcul(){
    return 10;
  }

}
```

J'ajoute une seconde classe B, qui hérite de A

```php
class B extends A{


}
```

J'hérite donc bien sur de la totalité des propriétés et méthodes de la classe A

Mais, concernant les spécificités de la classe B, j'ai besoin de modifier la méthode dont elle hérite

J'ai besoin dans un premier temps de récupérer la valeur 10 retournée par la méthode calcul

Voici la syntaxe

```php
class B extends A{

  public function calcul(){
    $nb = parent::calcul();
  }

}
```

Dans un premier temps, pour modifier la méthode calcul() de la classe A, je dois la redéclarer dans la classe B

Je peux reprendre le même nom, ou alors totalement le modifier. De toutes les manières, comme je vais la modifier, la surcharger, la redéfinir, cela devient une nouvelle méthode (cela pourrait être calcul2(), nouveauCalcul() etc...)

A l'intérieur je déclare une variable, $nb, qui va récupérer la valeur 10.

```
public function calcul(){

  $nb = parent::calcul();

}
```

Pour affecter la valeur de 10 à $nb, je vais devoir appeler la méthode calcul() (celle de la classe A, et je ne pourrai donc pas la nommer autrement) en utilisant parent::

A présent que j'ai récupéré la valeur retournée par la méthode parent, je peux améliorer ma nouvelle méthode pour répondre aux besoins de la classe qui hérite

```
public function calcul(){

  $nb = parent::calcul();

  if($nb < 100 )
    return '$nombre est inférieur à 100';
  else
    return '$nombre est supérieur à 100';
}
```

J'instancie ma classe B et je fais un print_r de l'objet pour voir ce que je récupère comme propriété et/ou méthodes via l'héritage

```
Array
(
    [0] => calcul
)
```

Dans le array retourné par get_class_methods() je récupère bien ma méthode calcul().

Pour etre bien sur que ce n'est pas celle déclarée directement dans la classe B, je modifie son nom

```
public function calcul2(){

  $nb = parent::calcul();

  if($nb < 100 )
    return '$nombre est inférieur à 100';
  else
    return '$nombre est supérieur à 100';
}
```

Le print_r() me retourne désormais cet array

```
Array
(
    [0] => calcul2
    [1] => calcul
)
```

J'ai bien deux méthodes. Maintenant que je suis sur que mon héritage fonctionne, je reviens à ma syntaxe d'origine (même nom pour les deux méthodes)

```php
public function calcul(){

  $nb = parent::calcul();

  if($nb < 100 )
    return '$nombre est inférieur à 100';
  else
    return '$nombre est supérieur à 100';
}
```

J'affiche maintenant ce que me retourne $nb, avec les contraintes de la surcharge (condition)

```php
echo $b->calcul();
```

6-2 Abstraction

Nouveau fichier, dans le même dossier, abstraction.php

Je déclare un classe abstraite

```php
<?php

abstract class Joueur{


}
```

Une classe abstraite permet d'imposer une contrainte à un développeur qui voudra utiliser mon code

Dans cet exemple, je vais simuler un site de jeu en ligne et je code les caractéristiques de la classe joueur (en m'intéressant aux méthodes, pas aux propriétés)

```php
abstract class Joueur{

  private $pseudo;

  private $age;

  public function seConnecter(){
    return $this->etreMajeur();
  }
```

```
  abstract function etreMajeur();

  abstract function devise();

}
```

Une première méthode, public, seConnecter() qui aura besoin d'une méthode etreMajeur() pour s'exécuter.

Je déclare ensuite une première méthode, abstraite, etreMajeur(), mais sans la développer, sans coder les instructions.

Je ne le peux pas. Car selon les pays, l'age pour etre majeur n'est pas le même

Idem pour la suivante devise(). Je ne peux établir une devise, ne sachant pas à l'avance qu'elle sera la devise du joueur.

Remarque: c'est du fait que j'ai déclaré dans la classe deux méthodes abstraites que ma classe devient abstraite

Je code donc tout ce que je peux coder, mais pour le reste, je déclare des méthodes abstraites, vides, qui devront etre codées par la suite par les développeurs qui voudront reprendre mon code (c'est le cas pour Symfony, WordPress … tous les modèles prêts à l'emploi, mais qui nécessiteront d'être complétés par la suite)

Ce développeur aura une classe bien spécifique, par exemple, la classe JoueurFr qui héritera de la classe globale Joueur

```
class JoueurFr extends Joueur{


}
```

Et à partir de là, le développeur aura cette charge d'implémenter les deux méthodes abstraites.

C'est une contrainte positive, pour que le code fonctionne bien, s'il ne le fait pas, il aura une erreur PHP pour le lui rappeler

Fatal error: Class JoueurFr contains 2 abstract methods and must therefore be declared abstract or implement the remaining methods

Je dois donc les déclarer et coder les instructions

```
class JoueurFr extends Joueur{

  public function etreMajeur(){
    return 18;
  }

  public function devise(){
```

```
    return '€';
  }

}
```

Remarque: on ne peut pas instancier une classe abstraite, du fait qu'elle contienne des méthodes abstraites. On ne pourra instancier que les classes qui héritent

```
$joueurFr = new JoueurFr;
echo 'En France, l\'age légal pour jouer en ligne est de ' . $joueurFr-
>etreMajeur() . '<br>';
echo 'Pour jouer en ligne en France, on doit utiliser les ' . $joueurFr->devise() ;
```

Pour les USA, le développeur devra redéfinir les paramètres

```
class JoueurUs extends Joueur{

  public function etreMajeur(){
    return 21;
  }

  public function devise(){
    return '$';
  }

}
```

6-3 Finalisation

Créer fichier finalisation.php dans le même dossier et créer une classe final

```
<?php

final class Application{


}
```

Une classe finale est déclarée ainsi pour empêcher qu'une autre classe en hérite (et que l'on modifie à l'intérieur une méthode, par exemple).

Le développeur qui verra le mot clé final saura qu'il ne faut pas tenter de la modifier.

Je code ceci à l'intérieur

```
final class Application{

  public function executerApplication(){
    return 'Je fonctionne';
  }
```

```
}
```

Elle est instanciable

```
$app = new Application;
echo $app->executerApplication() . '<br>';
```

Autre possibilité, je peux avoir une classe «normale», avec à l'intérieur une final function

```
class Application2{

  final function lancerApplication(){
    return 'Je fonctionne';
  }

}
```

Cela permettra qu'une autre classe en hérite, par contre la méthode sera non modifiable.

```
class Extension extends Application2{

}

$ext = new Extension;
echo '<pre>'; print_r(get_class_methods($ext)); echo '<pre>';
```

Le print_r() m'indique que j'hérite bien de la méthode

```
Array
(
    [0] => lancerApplication
)
```

Je ne peux par contre la surcharger

```
class Extension extends Application2{
  public function lancerApplication(){
    return 'je fonctionne autrement';
  }
}
```

Fatal error: Cannot override final method Application2::lancerApplication()

Remarque: une classe finale, par définition, ne pourra etre abstraite. Cette dernière n'étant pas instanciable et ayant vocation a etre parent des classes qui en hérite.

6-4 Trait

Nouveau fichier, trait-php

Une classe ne permet d'hériter que d'une seule autre classe à la fois.

Le trait va nous permettre de contourner cet obstacle, car une classe pourra désormais hériter de différentes méthodes de différents traits.

Au lieu d'écrire class, j'écris trait, avec un T majuscule avant le nom du trait.

```php
<?php

trait TPanier{

  public $nbProduits;

  public function afficheProduits(){
    return "J'affiche tous les produits <br>";
  }

}
```

Je déclare un second trait

```php
trait TMembre{

  public function afficheMembres(){
    return "J'affiche tous les membres <br>";
  }
}
```

Je déclare à présent une classe Site, et voici la syntaxe pour hériter / importer des méthodes issues de traits différents

```php
class Site{

  use TPanier, TMembre;

}
```

J'instancie ma classe pour vérifier avec un var_dump et print_r son contenu

Cela fonctionne bien, je récupère bien tout le contenu des propriétés et méthodes

Remarque: un trait ne peut etre instancié comme une classe

Fatal error: Uncaught Error: Cannot instantiate trait TMembre

Noter que, je peux hériter d'une classe (une seule) et en même temps hériter de traits

```php
class Produit{

  public $prix;

}
```

```php
class Site extends Produit{

  use TPanier, TMembre;

}
```

Conclusion: les traits permettent à une classe d'hériter de plusieurs traits (et donc de leur caractéristiques). Je ne peux par contre créer un objet issu de ces derniers

La question qu'il faudra se poser, c'est ais-je besoin d'une classe dans laquelle je mets toutes ces méthodes (code de peut-être 600 lignes), puis une classe va en hériter ?

Ou, pour des besoins de lisibilité, clarté, maintenance, il ne vaut mieux pas déclarer plusieurs traits, puis les exporter

6-5 Exercice

Créer un fichier exerciceVehicule.php

A l'intérieur, une classe et ses trois méthodes. Voici le code

```php
class Vehicule{

  public function demarrer(){
    return 'je démarre';
  }

  public function carburant(){
    return;
  }

  public function nombreTestsObligatoires(){
    return 100;
  }

}
```

Énoncé de l'exercice

- La classe véhicule ne peut etre instanciée

- La méthode demarrer() devra etre non-modifiable. Les classes qui en hériteront devront posséder la même méthode

- Je déclare une classe Renault qui en hérite et doit fonctionner au diesel

- Je déclare une classe Peugeot qui en hérite et doit fonctionner à l'essence

- La classe Renault doit effectuer 30 tests supplémentaires par rapport à un véhicule de base

- La classe Peugeot doit effectuer 70 tests supplémentaires par rapport à un véhicule de base

- Effectuer tous les affichages nécessaires pour vérifier les caractéristiques des classes Renault et Peugeot

Voici le code pour la correction

```php
<?php

abstract class Vehicule{

  final public function demarrer(){
    return 'je démarre';
  }

  abstract public function carburant();

  public function nombreTestsObligatoires(){
    return 100;
  }

}

class Renault extends Vehicule{

  public function carburant(){
    return 'je fonctionne au diesel';
  }

  public function nombreTestsObligatoires(){
    return 30 + parent::nombreTestsObligatoires();
  }

}

class Peugeot extends Vehicule{

  public function carburant(){
    return 'je fonctionne à l\'essence';
  }

  public function nombreTestsObligatoires(){
    return 70 + parent::nombreTestsObligatoires();
  }

}

$renault = new Renault;
echo '<pre>'; var_dump($renault); echo '</pre>';
echo '<pre>'; print_r(get_class_methods($renault)); echo '</pre>';
```

```
echo 'Je suis une Renault, ' . $renault->demarrer() . ', ' . $renault-
>carburant() . ' et je dois effectuer ' . $renault->nombreTestsObligatoires() . '
tests obligatoires avant de pouvoir etre proposée à la vente <br>';

$peugeot = new Peugeot;
echo '<pre>'; var_dump($peugeot); echo '</pre>';
echo '<pre>'; print_r(get_class_methods($peugeot)); echo '</pre>';

echo 'Je suis une Peugeot, ' . $peugeot->demarrer() . ', ' . $peugeot-
>carburant() . ' et je dois effectuer ' . $peugeot->nombreTestsObligatoires() . '
tests obligatoires avant de pouvoir etre proposée à la vente <br>';
```

7 Classes existantes (TRY / CATCH)

Création d'un nouveau dossier, 07-classes-existantes, avec un fichier exceptions.php

A l'intérieur, pas de classe, mais une fonction qui attend deux arguments; $tab et $element, fonction qui va nous permettre d'aller chercher des éléments dans un tableau

```
<?php

function recherche($tab, $element){


}
```

A l'intérieur de cette fonction deux conditions, je code puis j'explique

```
function recherche($tab, $element){

  if(!is_array($tab))
    throw new Exception('Ce n\'est pas un tableau');

  if(sizeof($tab) == 0)
    throw new Exception('Le tableau ne contient pas de données');

  $position = array_search($element, $tab);
  return $position;

}
```

Je vérifie en premier si l'argument entré n'est pas un tableau. Dans ce cas, j'envoie un message d'erreur (à l'attention du développeur, grâce à la fonction prédéfinie Exception).

La syntaxe throw permet d'envoyer ce message dans le bloc catch

En second, je vérifie la taille de mon tableau. Si elle est égale à 0, alors, nouveau message d'erreur

Enfin, si c'est bien un tableau, et qu'il contient des données, alors j'entre deux arguments pour array_search(), le premier l'élément à trouver, le second, le nom du tableau dans lequel je le cherche.

Je retourne sa position /son indice grace à array_search()

A présent je déclare deux tableaux, l'un vide, l'autre non.

```
$tab = array();
$tabPersonnages = ['Mario', 'Luigi', 'Toad', 'Peach', 'Bowser', 'Yoshi'];
```

Je fais un print_r de mon second tableau, pour avoir les indices.

```
echo '<pre>'; print_r($tabPersonnages); echo '<pre>';
```

Je déclare à la suite un bloc Try Catch.

C'est qlq chose de très utile pour personnaliser les messages d'erreur, plutôt que d'avoir le rendu des messages d'erreurs de PHP.

Dans le bloc Try, je teste mon code. S'il y a une erreur dedans, il va stopper toute la suite du code (pas de raison de continuer).

Le bloc Catch va me récupérer mon message d'erreur

Cette technique s'utilise fréquemment, notamment dans un init.php pour la connexion à la BDD

Si la connexion à la BDD ne se fait pas, j'aurais mon propre message d'erreur plutôt que celui de PHP

Je code

```
try{

}

catch(Exception $erreur){

}
```

Dans le catch j'envoie deux arguments, la fonction prédéfinie Exception (qui centralise les erreurs en cas de code erroné) en plus d'une variable ($erreur) qui va contenir (entre autres) le message de l'Exception codé en amont

Dans le bloc try je teste ceci

```
try{
  echo 'Le personnage Toad possède l\'indice ' . recherche($tabPersonnages, 'Toad')
. '<br>';
}
```

dans ce cas de figure, comme $tabPersonnages est bien un tableau et qu'il contient des données, je ne rentre dans aucune condition, il exécute le array_search() en reprenant les arguments déjà fournis.

Il peut donc mon donner l'indice de mon personnage recherché

Je tente une autre recherche, volontairement génératrice d'erreurs, avec le tableau vide

```
echo recherche($tab, 'Toad') . '<br>';
```

Rien ne s'affiche pour l'instant, c'est normal.

Je fais deux print_r de $erreur, pour voir son contenu ainsi que les méthodes disponibles.

Je récupère bien mon message d'erreur, preuve que j'ai bien été reversé dans le catch. J'ai aussi maintenant accès a plusieurs méthodes, telles que getFile, getLine qui me permettront de personnaliser le message d'erreur et me débuguer plus vite

Je peux a présent générer un message plus clair

```
catch(Exception $erreur){
  // echo '<pre>'; print_r($erreur); echo '<pre>';
  // echo '<pre>'; print_r(get_class_methods($erreur)); echo '<pre>';

  echo 'Erreur: ' . $erreur->getMessage() . '.<br>Cela ne respecte pas votre code à la ligne ' . $erreur->getLine() . ', du fichier ' . $erreur->getFile() . '<br>';
}
```

L'exception avec le try catch permet de personnaliser un message plus pertinent et agréable visuellement, en plus de stopper le code automatiquement.

Try et Catch fonctionnent ensemble. Ils sont une aide supplémentaire à l'attention du développeur.

Second exemple, avec une simulation de connexion à la BDD

Je déclare (plus bas) un nouveau try & catch, avec cette fois non pas Exception en argument, mais PDOException (spécifique à PDO)

```
try{

}

catch(PDOException $erreur){

}
```

Je code le try, avec en plus un message qui confirme la connexion réussie

```
try{
  $pdo = new PDO('mysql:host=localhost; dbname=boutique', 'root', '');
  echo 'Connexion établie';
}
```

Si ça ne fonctionne pas, je mets pour l'instant dans mon catch les deux print_r

```
catch(PDOException $erreur){
  echo '<pre>'; print_r($erreur); echo '<pre>';
  echo '<pre>'; print_r(get_class_methods($erreur)); echo '<pre>';
}
```

Si je fais une erreur, je tombe bien dans le catch

Je personnalise à présent mon message

```
catch(PDOException $erreur){
  // echo '<pre>'; print_r($erreur); echo '<pre>';
  // echo '<pre>'; print_r(get_class_methods($erreur)); echo '<pre>';
  echo 'Erreur: ' . $erreur->getMessage() . '.<br>Problème pour se connecter à la
BDD à la ligne ' . $erreur->getLine() . ', du fichier ' . $erreur->getFile() .
'<br>';
}
```

8 Namespace

Nouveau dossier 08-namespace, avec à l'intérieur 2 fichiers, namespace-commerce.php et namespace-general.php.

Un namespace est un espace de rangement. De manière générale, on ne déclare qu'un namespace par fichier

Comme je ne vais faire que des tests, je vais en déclarer plusieurs dans un même fichier (je code dans namespace-commerce.php)

Il existe deux manières de déclarer un namespace

```
<?php

namespace Commerce1{

// code


}
```

Mais la plus usuelle est celle ci

```
<?php

namespace Commerce1;
```

Je déclare à l'intérieur une classe avec une propriété

```
class Commande{
```

```
  public $nbCommandes = 3;

}
```

Juste après, je déclare un second namespace, avec encore à l'intérieur une classe qui contient une propriété

```
namespace Commerce2;

class Produit{

  public $nbProduits = 22;

}
```

A la suite, nouveau namespace, avec deux classes

```
namespace Commerce3;

class Panier{

  public $nbProduitsPanier = 2;

}

class Produit{

  public $nbProduits = 12;

}
```

J'ai deux classes Produit, mais qu'elles sont dans deux namespace différents, cela ne pose pas de problème

Je code à présent dans namespace-general.php

Je déclare aussi un namespace, puis je fais un require_once de mon autre fichier, et à partir de là, j'importe les namespaces qu'il contient

```
<?php

namespace General;

require_once('namespace-commerce.php');

use Commerce1, Commerce2, Commerce3;
```

Avec la constante magique __NAMESPACE__ ; je peux vérifier dans quel namespace je me situe

```
echo __NAMESPACE__ ;
```

Je me connecte à une BDD

```
$pdo = new PDO('mysql:host=localhost;dbname=boutique', 'root', '');
```

PHP me signale une erreur. Il ne reconnaît pas la classe PDO dans cet espace General.

Du fait que j'ai déclaré un namespace, je ne suis plus dans l'espace global, mais dans celui de mon namespace.

Pour parer à cette erreur, je vais devoir utiliser un \ devant PDO

```
$pdo = new \PDO('mysql:host=localhost;dbname=boutique', 'root', '');
```

L'erreur disparaît. Le \ me permet de sortir de cet espace pour retourner dans le global

---

A présent je vais utiliser mes imports pour instancier la classe Commande

```
$commande = new Commande;
```

J'ai à nouveau une erreur. Je ne peux pas le faire de cette manière

Je dois écrire (rappeler son namespace)

```
$commande = new Commerce1\Commande;
```

Je vais un var_dump de mon objet pour vérifier que c'est bon

```
echo '<pre>'; var_dump($commande);echo '<pre>';
```

J'instancie mes trois autres classes-existantes

```
$produit = new Commerce2\Produit;
echo '<pre>'; var_dump($produit);echo '<pre>';

$panier = new Commerce3\Panier;
echo '<pre>'; var_dump($panier);echo '<pre>';

$produit = new Commerce3\Produit;
echo '<pre>'; var_dump($produit);echo '<pre>';
```

Remarque: concrètement, un namespace pourra me servir pour naviguer dans mon projet pour atteindre les différents fichiers. En procédural, pour passer d'un ficher à un autre, positionné dans un autre dossier (par exemple le fichier style.css situé dans le dossier css, je devais ecrire css/style.css)
En POO, je ne peux pas utiliser le nom du dossier...c'est le namespace qui va le remplacer

9 Autoload

Nouveau dossier 09-autoload, à l'intérieur 6 fichiers

- A.class.php

- B.class.php

- C.class.php

- D.class.php

- autoload.php

- accueil.php

L'autoload permet d'automatiser l'importation des fichiers. Il va remplacer le require_once

Je code dans A.class.php

```php
<?php

class A{

  public function __construct(){
    echo 'Instanciation de A';
  }
}
```

Je fais la même chose dans B, C et D

Si je veux récupérer le contenu de ces fichiers dans accueil.php, je vais devoir faire plusieurs require_once. Et encore plus s'il doit réceptionner encore plus de fichiers. Un autoload va remédier à cela en automatisant cette tâche

Je code ceci dans autoload.php; une méthode qui attend un argument

```php
<?php

function inclusionAuto($nomDeClasse){

}
```

J'utilise à présent la fonction prédéfinie spl_autoload_register()

C'est une fonction qui s'exécute à chaque fois qu'elle voit le mot clé new. De plus, elle renvoie automatiquement, c'est sa programmation, tout ce qui suit le mot new (c'est a dire le nom de la classe) en argument de la fonction qu'elle a elle même en argument

Je lui donne en paramètre la fonction inclusionAuto (syntaxe telle que si c'était une chaîne de caractères)

```php
spl_autoload_register('inclusionAuto');
```

Je teste en instanciant la classe A et je vérifie avec un var_dump

```php
<?php

function inclusionAuto($nomDeClasse){
  echo '<pre>'; var_dump($nomDeClasse); echo '</pre>';
  echo 'j\'entre bien dans cette fonction';
}

spl_autoload_register('inclusionAuto');

$a = new A;
```

Il me récupère bien le nom de la classe (ne pas faire attention à l'erreur, elle est normale, il n'a pas accès à la classe A. L'important était de vérifier que la fonction s'exécutait bien, tout en récupérant le nom de la classe.

Si je veux supprimer l'erreur, je fais un require_once au dessus

```php
<?php

function inclusionAuto($nomDeClasse){
  require_once($nomDeClasse. '.class.php');
  echo "require_once($nomDeClasse.class.php) <br>";

  var_dump($nomDeClasse);
  echo '<br>j\'entre bien dans cette fonction';
}

spl_autoload_register('inclusionAuto');

$a = new A;
```

Désormais, je peux récupérer tout le contenu de mes classes dans n'importe que fichier. Je n'ai plus qu'un seul require_once à mettre en haut de mon fichier, et pas pour tous les fichiers dont j'aurais besoin

Je code ceci dans accueil.php

```php
<?php

require_once('autoload.php');

$a = new A;
$b = new B;
$c = new C;
$d = new D;

echo'<pre>'; var_dump($a); echo '</pre>';
echo'<pre>'; var_dump($b); echo '</pre>';
echo'<pre>'; var_dump($c); echo '</pre>';
echo'<pre>'; var_dump($d); echo '</pre>';
```

10 MVC (Model View Controller)

Nouveau dossier 10-mvc. Dedans un fichier panier.php.

Je code en procédural le processus de création d'un panier

```php
<?php

session_start();

if(isset($_GET['action']) && $_GET['action'] == 'create' ||
isset($_SESSION['panier'])){
  $_SESSION['panier'] = array(26,27,28);
  echo "Produits présents dans le panier: " . implode(' - ', $_SESSION['panier']) .
'<hr>';
  echo '<a href="?action=delete">Vider le panier</a>';
}else{
  echo '<a href="?action=create">Créer le panier</a>';
}

if(isset($_GET['action']) && $_GET['action'] == 'delete'){
  unset($_SESSION['panier']);
}
```

J'ouvre un session.

Si j'ai reçu dans l'URL une action «create» ou si la session panier existe déjà, j'affiche les indices des produits contenus dans mon panier avec en plus un lien pour vider le panier

Si ce n'est pas le cas, je mets un lien, créer le panier

C'est effectivement ce cas de figure qui apparaît. Je clique sur ce lien, puis apparaît le texte avec les indices des produits. Si je veux vider le panier, je vais devoir cliquer deux fois dessus, car la première fois ne suffira pas, même si le panier a été vidé, l'affichage reste jusqu'à cliquer une seconde fois sur le lien

En fait, c'est du a un mauvais positionnement de mon code dans mon traitement PHP.

Le code se recharge et re-exécute une seconde fois le script, même si le panier a été vidé dès la première. Il faudra cliquer une seconde fois pour confirmer que le panier n'existe plus et voir le lien créer réapparaître

Il aurait fallu positionner le if qui encadre le unset($_SESSION['panier']) en haut du ficher, après le session_start()

Ce code positionné en premier, c'est lui qui sera lu en premier lorsqu'on cliquera sur vide le panier

Le procédural est souvent confronté a ce type de mauvais organisation du code

PHP OO va résoudre ce problème d'organisation en s'appuyant sur une architecture de type MVC, Model View Controller, qui va séparer les langages entre eux.

Dans Model (dossier), on va traiter toutes les requêtes SQL. Chaque requête sera codée dans une class, avec ses attributs et méthodes

Dans View, seront regroupées tous mes affichages (tous mes templates), tout le traitement HTML

Controller va servir de pivot entre Model et View. Selon l'action du user, il va récupérer une requête SQL, créer un objet de la classe, pour l'envoyer dans le bon template

Concrètement aussi, l'intégrateur de l'équipe saura qu'il devra aller travailler dans le dossier view sans se préoccuper du reste du code

11 CRUD

Nouveau dossier 11-crud.

Arborescence de mon projet

CRUD

- dossier App
  - fichier config.xml
- dossier controller
  - fichier Controller/php
- dossier model
  - fichier EntityRepository.php
- dossier view
  - fichiers les vues au fur et a mesure
- fichier index.php (le renommer en index1.php pour éviter l'effet porte d'entrée, tant que le projet n'est pas terminé)
- fichier autoload.php

11-1 BDD

Je crée une nouvelle BDD que je nomme oo_entreprise et j'importe celle existante

11-2 config.xml

Xml est un langage de balises, comme html, sauf que l'on peut déclarer ses propres balises

Ce fichier va me servir à ne pas coder en dur une connexion à la base de données comme on pouvait le faire avant

Je code dans config.xml

```xml
<?xml version="1.0" ?>

<config>

</config>
```

A l'intérieur de <config> je vais declarer toutes les infos dont j'aurais besoin pour la connexion

```xml
<?xml version="1.0" ?>

<config>
  <host>localhost</host>
  <user>root</user>
  <password></password>
  <dbname>oo_entreprise</dbname>
  <table>employe</table>
</config>
```

11-3 Autoload

Je mets en place à présent mon autoload.php

Pour tester si mon autoload va fonctionner, je vais coder dans Controller.php.

Je déclare rapidement ceci à l'intérieur, juste un message d'affichage pour vérifier ensuite que tout est ok

```php
<?php

namespace controller;

class Controller{

  public function __construct(){
    echo "instanciation de la class ok";
  }

}
```

Je reviens dans autoload.php, et je teste spl_autoload_register()

Je dis que a chaque fois que l'autoload voit passer le mot new, il va dans la classe Autoload pour exécuter la méthode inclusionAuto()

```php
<?php




spl_autoload_register(array('Autoload', 'className'));
```

Je mets le tout dans un array car cette fois ma méthode est intégrée dans une classe (travailler plus proprement quel ors du test autoload)

Pour tester, je vais devoir instancier ma classe Controller

Sous spl_autoload_register() j'écris

```
$controller = new controller\Controller;
```

Je rentre d'abord dans le namespace controller, puis dans la classe Controller

Maintenant que l' autoload a vu le mot new, il doit aller dans la classe Autoload (que je vais coder) pour exécuter la méthode inclusionAuto (que je vais coder dedans) et donner en argument tout ce qui suit le mot new (c'est automatique)

```
<?php

class Autoload{
//                                controller/Controller
  public static function inclusionAuto($className){

  }

}


spl_autoload_register(array('Autoload', 'inclusionAuto'));

$controller = new controller\Controller;
```

Cette méthode je la mets en static pour qu'elle appartienne à la classe et non à un éventuel objet qui pourrait etre instancié de la classe Autoload.

$classname sera l'argument qui réceptionnera tout ce qui suit le mot clé new

Dans ma méthode, je fais un require_once suivi de la constante magique __DIR__ qui va me récupérer automatiquement le chemin du fichier dans lequel je suis (appelé via l'argument entré en paramètre)

```
//                              controller/Controller
public static function inclusionAuto($className){
  require_once __DIR__ ;
}
```

Je peux tester plus bas dans mon code ce que fais __DIR__ (je dois mettre en commentaire momentanément l'instanciation de Controller)

```
// $controller = new controller\Controller;
```

```
echo __DIR__ ;
```

Elle me renvoie bien le chemin dans lequel se trouve mon fichier autoload.php

D:\xampp\htdocs\personnel\poles_back\php-oo\11-crud

__DIR__ va me permettre de récupérer mon chemin de manière dynamique, que je sois en local comme en ligne

Je complète mon require_once()

```
//                              controller/Controller
public static function inclusionAuto($className){
  require_once __DIR__ . '/' . str_replace('\\', '/', $className . '.php') ;
}
```

Après __DIR__, je concatène avec un slash pour séparer du dossier suivant. Ensuite je suis obligé d'utiliser str_replace pour remplacer les anti slash en slash (double anti slash, le premier servant de caractère d'échappement, sinon, le code est perturbé...faire le test avec un seul \ )

Remarque: si je garde les anti slash, cela pourrait fonctionner aussi, mais dans le doute, je garde le str_replace

Voici la totalité du code de ce fichiers

```
<?php

class Autoload{
  //                              controller/Controller
  public static function inclusionAuto($className){
    require_once __DIR__ . '/' . str_replace('\\', '/', $className . '.php') ;
    echo __DIR__ . '/' . str_replace('\\', '/', $className . '.php <br>') ;
  }

}


spl_autoload_register(array('Autoload', 'inclusionAuto'));

$controller = new controller\Controller;
```

Pour vérifier si tout fonctionne, je code ceci dans mon index.php

```
<?php

require_once('autoload.php');

$controller = new controller\Controller;

echo'<pre>'; var_dump($controller); echo '</pre>';
```

Si c'est bon, je vais à présent mettre divers affichage en commentaires dans autoload.php pour ne conserver que l'essentiel

```php
<?php

class Autoload{
  //                              controller/Controller
  public static function inclusionAuto($className){
    require_once __DIR__ . '/' . str_replace('\\', '/', $className . '.php') ;
    // echo __DIR__ . '/' . str_replace('\\', '/', $className . '.php <br>') ;
  }

}


spl_autoload_register(array('Autoload', 'inclusionAuto'));

// $controller = new controller\Controller;
```

11-4 EntityRepository.php

Dans mon Controller je dois accéder a tout ce qui est codé dans le dossier model; les classes, les requêtes ou la connexion à la BDD

Dans Controller.php, je mets en commentaire le contenu du __construct(), je n'en ai plus besoin

```php
public function __construct(){
  // echo "instanciation de la class ok<hr>";
}
```

Au dessus de ce __construct(), je déclare une propriété private, $dbEntityRepository, qui stockera les données envoyées par le model. Ces données ensuite seront dispatchées par le controller vers les views concernées

```php
private $dbEntityRepository;

public function __construct(){
  // echo "instanciation de la class ok<hr>";
}
```

Et $dbEntityRepository pourra nous servir ensuite dans le constructeur pour récupérer la connexion à la BDD.

En l'état actuel, ce n'est pas possible, je vais devoir d'abord coder dans EntityRepository.php

Je donne a ce fichier un namespace, model, qui devra etre similaire au nom du dossier dans lequel il est (attention à la casse, c'est important pour l'autoload.php)

Je déclare la classe, attention là aussi à la casse (similaire au nom du fichier → autoload.php)

```php
<?php
```

```php
namespace model;

class EntityRepository{
```

Cette classe utilise un nom assez commun, que l'on retrouvera par exemple dans le code généré par Symfony lorsque l'on créera un projet via ce Framework.

Entity, représentant une entité, en fait une table SQL. Repository représentant les classes qui permettent de faire une requête de sélection.

A l'intérieur de cette classe je déclare une private $pdo, qui stockera un objet issu de la classe PDO, représentant la connexion à la BDD.

Une public $table, qui représentera la table sur laquelle on fera une requête de sélection

Enfin, une public function getPdo(), pour construire la connexion à la BDD

```php
class EntityRepository{

  private $pdo;
  public $table;

  public function getPdo(){


  }

}
```

Je code à l'intérieur de cette méthode, puis j'explique

```php
public function getPdo(){

  if(!$this->pdo){

  }
  return $this->pdo;

}
```

Si l'objet courant ne pointe pas vers $pdo, c'est que la connexion n'est pas encore faite; alors je vais me connecter (pas encore codé)

Si par contre $this pointe déjà sur $pdo, alors je retourne le résultat cette connexion pour la récupérer

Voici le code pour se connecter (dans le if)

Tout d'abord, par mesure de précaution, je code cette connexion dans un try/catch .

Pour le catch, deux arguments: Exception (précédé d'un anti slash pour sortir du namespace model) et $erreur pour récupérer les méthodes qui serviront à générer un message d'erreur adapté

```
if(!$this->pdo){
  try{

  }
  catch(\Exception $erreur){

  }
}
return $this->pdo;
```

Dans le bloc Try, je vais récupérer les informations contenues dans config.xml pour faire la connexion

Je code puis j'explique (le try...pour le catch, je ne fais qu'ajouter une message plus adapté)

```
try{
  $xml = simplexml_load_file('../app/config.xml');
  echo '<pre>'; print_r($xml);echo '</pre>';
}
catch(\Exception $erreur){
  echo 'Impossible de récupérer le contenu du fichier. Erreur: ' . $erreur-
>getMessage() . '.<br>Il y a une erreur à la ligne ' . $erreur->getLine() . ', du
fichier ' . $erreur->getFile() . '<br>';
}
```

Je déclare une variable $xml pour stocker ce que va extraire la fonction prédéfinie simplexml_load_file(). Je lui donne le chemin pour atteindre le fichier visé, d'abord ../ pour sortir du dossier model, puis le dossier app/ puis le nom de mon fichier

Je fais un print_r pour vérifier le contenu récupérer

Pour afficher le résultat de ce print_r, je dois instancier ma classe EntityRepository (en dehors de ma classe, en bas du fichier)

```
$et = new EntityRepository;
$et->getPdo();
```

Le résultat est satisfaisant, j'ai bien récupéré les infos insérées en début de projet

```
SimpleXMLElement Object
(
    [host] => localhost
    [user] => root
    [password] => SimpleXMLElement Object
        (
        )

    [dbname] => oo_entreprise
    [table] => employe
)
```

A présent, dans le bloc Try, j'affecte à ma public $table, le nom de la table trouvé dans le config.xml, que je récupère grace à $xml→table

```
try{
  $xml = simplexml_load_file('../app/config.xml');
  echo '<pre>'; print_r($xml);echo '</pre>';
  $this->table = $xml->table;
}
```

Maintenant que j'ai récupéré diverses infos nécessaires, je me connecte à ma BDD.

A l'intérieur du bloc Try, je déclare un autre Try / Catch, celui qui va tester la connexion, avec un /PDOException dans le Catch

```
try{
  $xml = simplexml_load_file('../app/config.xml');
  echo '<pre>'; print_r($xml);echo '</pre>';

  $this->table = $xml->table;

  try{

  }catch(\PDOException $erreur){

  }
}
```

Je code la connexion plus j'ajoute un message adapté en cas d'erreur (notamment penser à vérifier le config.xml)

```
try{
  $xml = simplexml_load_file('../app/config.xml');
  echo '<pre>'; print_r($xml);echo '</pre>';

  $this->table = $xml->table;

  try{
    $this->pdo = new \PDO("mysql:host=$xml->host;dbname=$xml->dbname", "$xml-
>user", "$xml->password", array(\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION, \
PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8')
);
    echo "Connexion établie";
  }catch(\PDOException $erreur){
    echo 'Erreur: ' . $erreur->getMessage() . '. Le problème vient peut-être des
données renseignées dans le config.xml. Vérifiez les !
<br>Problème pour se connecter à la BDD à la ligne ' . $erreur->getLine() . ', du
fichier ' . $erreur->getFile() . '<br>';
  }
}
```

Maintenant que tout fonctionne, je mets en commentaire tout ce qui m'a servi pour les tests...print_r, message connexion, instanciation de la classe. Il faut aussi enlever les ../ avant app/

Cette syntaxe ne concerne que le test a partir de ce fichier

Voici le fichier définitif

```php
<?php

namespace model;

class EntityRepository{

  private $pdo;
  public $table;

  public function getPdo(){

    if(!$this->pdo){
      try{
        $xml = simplexml_load_file('app/config.xml');
        // echo '<pre>'; print_r($xml);echo '</pre>';

        $this->table = $xml->table;

        try{
          $this->pdo = new \PDO("mysql:host=$xml->host;dbname=$xml->dbname", "$xml->user", "$xml->password", array(\PDO::ATTR_ERRMODE => \PDO::ERRMODE_EXCEPTION, \PDO::MYSQL_ATTR_INIT_COMMAND => 'SET NAMES utf8'));
          // echo "Connexion établie";
        }catch(\PDOException $erreur){
          echo 'Erreur: ' . $erreur->getMessage() . '. Le problème vient peut-etre des données renseignées dans le config.xml. Vérifiez les !<br>Problème pour se connecter à la BDD à la ligne ' . $erreur->getLine() . ', du fichier ' . $erreur->getFile() . '<br>';
        }
      }
      catch(\Exception $erreur){
        echo 'Impossible de récupérer le contenu du fichier. Erreur: ' . $erreur->getMessage() . '.<br>Il y a une erreur à la ligne ' . $erreur->getLine() . ', du fichier ' . $erreur->getFile() . '<br>';
      }
    }
    return $this->pdo;

  }

}

// $et = new EntityRepository;
// $et->getPdo();
```

Je reviens à présent coder dans mon Controller.php

Dans le constructeur de ce dernier, je vais devoir disposer d'un accès a tout ce qui aura pu etre codé dans EntityRepository.php

Je vais utiliser pour cela $dbEntityRepository

```php
class Controller{

  private $dbEntityRepository;

  public function __construct(){
    // echo "instanciation de la class ok<hr>";

    $this->dbEntityRepository = new \model\EntityRepository;
  }

}
```

De la même manière que $pdo avait stocké les informations de l'objet instancié de la classe PDO (les infos liées à la connexion). Cette dernière va stocker les informations de l'objet instancié de la classe EntityRepository

Remarque: au moment où j'instancierai ma classe Controller, automatiquement mon constructeur instanciera la classe EntityRepository

11-5 Index.php

Pour tester tout cela, on va vérifier dans index.php

J'avais déjà ce code (pas besoin de use car le require_once autoload fait déjà ce travail)

```php
<?php

require_once('autoload.php');

$controller = new controller\Controller;

echo'<pre>'; var_dump($controller); echo '</pre>';
```

Ce var_dump m'envoie de nouvelles informations à présent.

```
object(controller\Controller)#1 (1) {
  ["dbEntityRepository":"controller\Controller":private]=>
  object(model\EntityRepository)#2 (2) {
    ["pdo":"model\EntityRepository":private]=>
    NULL
    ["table"]=>
    NULL
  }
}
```

Le new Controller a obligé l'autoload à instancier ma classe controller, et du fait que celle ci a un constructeur, il s'est auto exécuté et instancié ma classe EntityRepository

Test intéressant à faire: dans autoload.php, dé-commenter le echo __DIR__ et ajouter devant ce dernier, «require_once» en le concaténant à ce qui suit

```
echo "require_once" . __DIR__ . '/' . str_replace('\\', '/', $className . '.php
<br>') ;
```

Rafraîchissez la page index.php

Je n'ai pas un mais deux echos qui apparaissent, car deux classes ont bien été instanciées

require_onceD:\xampp\htdocs\personnel\poles_back\php-oo\11-crud/controller/Controller.php
require_onceD:\xampp\htdocs\personnel\poles_back\php-oo\11-crud/model/EntityRepository.php

Je re-commente ce le echo dans autoload.php, je n'en ai plus besoin et je retourne dans Controller.php

Je vais coder à présent une condition pour gérer l'action du user.

Veut-il ajouter un employé ? Le modifier, voir sa fiche, le retirer de la BDD ?

Après le constructeur, je déclare une méthode handleRequest (existante dans Symfony) avec à l'intérieur un try / catch. Dans ce dernier, je récupère un message copié/collé que j'adapterai ensuite

```
public function handleRequest(){

  try{

  }
  catch(\Exception $erreur){
    echo 'Impossible de récupérer le contenu du fichier. Erreur: ' . $erreur-
>getMessage() . '.<br>Il y a une erreur à la ligne ' . $erreur->getLine() . ', du
fichier ' . $erreur->getFile() . '<br>';
  }

}
```

Au dessus du try / catch je vais coder ma condition, selon l'action récupérée dans l'URL (envoyée selon le clic du USER)

Je code puis explique

```
$choixUser = isset($_GET['choixUser']) ? $_GET['choixUser'] : NULL;

try{

}
catch(\Exception $erreur){
  echo 'Impossible de récupérer le contenu du fichier. Erreur: ' . $erreur-
>getMessage() . '.<br>Il y a une erreur à la ligne ' . $erreur->getLine() . ', du
fichier ' . $erreur->getFile() . '<br>';
}
```

La syntaxe de cette condition est en ternaire

Si je récupère dans l'URL choixUser=qlqchose, alors je stocke la valeur de ce qlqchose dans une variable nommée $choixUser.

Je pourrais donc ainsi y stocker son choix: Créer, Ajouter ou Supprimer

Autre /else je stocke dans cette même variable NULL.

Je code à présent ceci dans le bloc try

```
try{

  if($choixUser == 'add' || $choixUser == 'update')
    $this->save($choixUser);
  elseif($choixUser == 'select')
    $this->select();
  elseif($choixUser == 'delete')
    $this->delete();
  else
    $this->selectAll();

}
```

Si l'action sélectionnée est add (un employé) ou update (la fiche d'un employé), j'irais chercher la méthode save() (que je n'ai pas encore coder)

Si c'est l'action select (pour ne voir la fiche que d'un seul employé), j'irais pointer vers la méthode select()

Si c'est l'action delete() (supprimer la fiche d'un employé), j'irais pointer vers la méthode delete()

Dans tous les autres cas /else j'affiche tous les employés (méthode selectAll())

Cette méthode est la première que je vais coder, en dessous de handleRequest(). C'est celle qui s'affiche si l'internaute n'a cliqué sur rien

Je code toutes les méthodes (sans donner le contenu final, juste un echo)

```
public function selectAll(){

  echo "Affichage de tous les employés";

}
public function save(){

  echo "Affichage d'un formulaire d'ajout ou modification";

}
public function select(){

  echo "Affichage d'un seul employé'";

}
```

```php
public function delete(){

  echo "Suppression de la fiche d'un employé";

}
```

Pour vérifier cet affichage sur la page index, je fais appel à la méthode handleRequest() . Le var_dump est désormais commenté

```php
<?php

require_once('autoload.php');

$controller = new controller\Controller;

// echo'<pre>'; var_dump($controller); echo '</pre>';

$controller->handleRequest();
```

Je récupère bien cet affichage, car rien n'a transité via l'URL

Affichage de tous les employés

Faire le test en ajoutant à la main (car on ne peut pas faire autrement pour l'instant) dans l'URL, après index.php, ?choixUser=add

Cela fonctionne aussi, mon index affiche

Méthode save | Affichage d'un formulaire d'ajout ou modification

---

En fait, ce n'est pas véritablement le fichier où j'aurai du développer ces méthodes. Ce n'était que pour rapidement tester. Le Controller ne fera que les récupérer (comme il a récupéré getPdo() pour ensuite l'afficher sur une vue

Je dois les coder dans EntityRepository.php

Je fais donc un copié collé de toutes ces méthodes vers cet autre fichier (et je les garde dans celui ci pour encore des tests).

Je les colle après la méthode getPdo()

Je vais à présent leur donner leur véritables instructions, tout en les rebaptisant (permettra d' identifier plus facilement plus tard leur provenance)

Je commence comme tout à l'heure par selectAllEntityRepo(), et je vais devoir dans un premier temps récupérer la connexion à la BDD en pointant vers getPdo()

```php
public function selectAllEntityRepo(){

  $data = $this->getPdo()->query("SELECT * FROM $this->table");

}
```

je pointe vers la méthode avec $this . Je fais un query puis SELECT * FROM ma table employe. Je conserve ce résultat dans une variable nommée $data

Une fois faite ma requête de sélection, je passe au fetchAll() et je retourne le résultat

```
public function selectAllEntityRepo(){

  $data = $this->getPdo()->query("SELECT * FROM $this->table");
  $afficheTousEmployes = $data->fetchAll(\PDO::FETCH_ASSOC);
  return $afficheTousEmployes;
}
```

Dans mon Controller, je peux récupérer le résultat de ce fetchAll grace à l'instance de mon EntityRepository contenue dans (en haut de mon fichier)

```
private $dbEntityRepository;
```

Je code dans ma méthode selectAll()

```
public function selectAll(){
    echo "Méthode selectAll | Affiche tous les employés";
    $resultat = $this->dbEntityRepository->selectAllEntityRepo();
    echo '<pre>'; print_r($resultat); echo '</pre>';
  }
```

Je pointe avec $this vers cette propriété pour récupérer ce que retourne la méthode selectAllEntityRepo, et je mets ce résultat dans une variable.

Variable que je passe dans un var_dump pour afficher son contenu (dans index.php)

11-6 Création d'un layout

Création dans le dossier view d'un fichier nommé layout.php

Faire un doctype + installer Bootstrap avec nav etc.

```
<!DOCTYPE html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Crud POO</title>

  <!-- css BS -->
  <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-
1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3"
crossorigin="anonymous">
```

```
</head>
<body>

  <header>

  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container-fluid">
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-
bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-
label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarNav">
        <ul class="navbar-nav">
          <li class="nav-item">
          <a class="nav-link active" aria-current="page" href="#">Accueil</a>
          </li>
          <li class="nav-item">
          <a class="nav-link" href="#">Ajouter un employé</a>
          </li>
        </ul>
      </div>
    </div>
  </nav>

  </header>

  <main class="container">

    <h1 class="text-center my-5">Affichage des employés</h1>

  </main>

  <footer>

  </footer>

  <!-- bundle BS -->
  <script
src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"
integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p"
crossorigin="anonymous"></script>

</body>
</html>
```

Ce layout va devenir la base de toutes mes pages.

Il est le gabarit par défaut pour tout le site

Je l'appellerai pour toutes mes pages, en lui ajoutant, selon mes pages, un template. Template que je pourrai ajouter dans une des parties de mon layout. Ce layout accueillera en plus d'un template, affichage spécial pour cette page, du contenu. Ce seront les données particulières de cette page

11-6 Méthode Render

Créer une méthode render() dans Controller.php, en dessous de la méthode handleRequest

Elle va prendre trois arguments (en fait les trois éléments qui s'afficheront sur une seule page)

```
public function render($layout, $template, $parameters = array()){
  extract($parameters);
}
```

Extract est une fonction prédéfinie qui permet d'extraire chaque indice d'un tableau sous forme de variable. Exemple, avec mon tableau parameters, je n'aurai pas à ecrire $parameters['employes'], mais directement $employes.

Ensuite, je vais mettre en place un processus de mise en mémoire progressive pour stocker mes trois éléments, puis les afficher

Je code puis j'explique

```
public function render($layout, $template, $parameters = array()){

  extract($parameters);

  ob_start();
  require_once("view/$template");
  $content = ob_get_clean();

  ob_start();
  require_once("view/$layout");

  return ob_end_flush();

}
```

Je commence le processus de mise en tampon, avec ob_start(), pour garder en mémoire des données. On temporise en fait la sortie d'affichage

Je fais un require_once du template qui m'intéresse. Ce require est important car il va faire appel au fichier template pour pouvoir ensuite l'insérer dans son layout gabarit

Je continue le processus de mise en mémoire avec ob_get_clean(), en stockant les données (le template en fait, grace au require_once) dans une variable $content

Je recommence ce processus pour le layout, sauf ce dernier n'aura pas besoin d'être stocké dans une variable car il sera envoyé directement sur le navigateur avec la nav etc...

Je termine avec ob_end_flush qui va libérer l'affichage et tout faire apparaitre sur le navigateur

---

Je vais utiliser cette méthode render() pour donner un meilleur affichage a selectAll (sur index.php)

Je vais pour cela créer un fichier affichage-employes.php (ça sera son template particulier)dans le dossier view

A présent, je vais totalement modifier le script de selectAll (mettre tout en commentaires à l'intérieur

Je code puis j'explique

```
public function selectAll(){
  // echo "Méthode selectAll | Affiche tous les employés";
  // $resultat = $this->dbEntityRepository->selectAllEntityRepo();
  // echo '<pre>'; print_r($resultat); echo '</pre>';

  $this->render('layout.php', 'affichage-employes.php',[
    'title' => 'Affichage des employés',
    'data' => $this->dbEntityRepository->selectAllEntityRepo()
  ]);
}
```

J'appelle ma méthode render() dans selectAll().

Le premier argument sera le layout que je veux utiliser. En second, le template voulu pour cet affichage. En dernier, dans un tableau, les paramètres que je veux voir affichés, c'est a dire le titre (que je renseigne) . Puis quelles infos, et là je reprend ma syntaxe précédente en allant pointer vers dbEntityRepository qui me permettra d'aller récupérer la bonne requête pour afficher tous les employés

Désormais, c'est le layout dans view qu va s'afficher pour la page index.php

---

J'affiche les données de ma requête selectAllEntityRepo dans leur template

Je peux juste tenter dans un premier temps de faire un print_r

Je code ceci dans affichage-employes.php

```
<?php

echo '<pre>'; print_r($data);echo '</pre>';
```

J'ajoute ceci dans le main de layout, car c'est la bas que je veux afficher les données

```
<main class="container">

  <?= $content ?>
```

```
</main>
```

Le title dans data va aller alimenter le h1 dans main

```
<main class="container">

  <h1 class="text-center my-5"><?= $title ?></h1>

  <?= $content ?>

</main>
```

11-7 Affichage dans le template selectAll

Je vais à présent mettre en place l'affichage conventionnel de tous les employes, à la place du print_r

Dans affichage-employes.php, je mets en commentaires le print_r, puis je code ceci dans une balise <table>

```
<?php

// echo '<pre>'; print_r($data);echo '</pre>';
?>

<table class="table table-dark text-center my-5">
  <?php foreach($data as $dataEmployes): ?>
    <tr>
      <td><?= implode('</td><td>', $dataEmployes) ?></td>
    </tr>
  <?php endforeach; ?>
</table>
```

Il me manque néanmoins les entêtes

Pour cela, je vais retourner dans EntityRepository et créer une nouvelle méthode

Tout en bas de mon fichier, après deleteEntityRepo()

```
public function getFields(){
  $data = $this->getPdo()->query("DESC " . $this->table);
  $afficheEntetes = $data->fetchAll(\PDO::FETCH_ASSOC);
  return $afficheEntetes;
}
```

Cette fonction va donc me permettre de récupérer les entêtes.

Je fais appel à getPdo, pour ensuite faire un query sur DESC...pour description (et pas descendant, order by) et je récupère ce résultat dans $data

Sur $data je fais un fetchAll et je récupère ça dans une nouvelle variable sur laquelle je fais un return

Cette méthode, je vais la récupérer dans mon Controller.php, dans ma méthode selectAll()

```
public function selectAll(){
  // echo "Méthode selectAll | Affiche tous les employés";
  // $resultat = $this->dbEntityRepository->selectAllEntityRepo();
  // echo '<pre>'; print_r($resultat); echo '</pre>';

  $this->render('layout.php', 'affichage-employes.php',[
    'title' => 'Affichage de tous les employés',
    'data' => $this->dbEntityRepository->selectAllEntityRepo(),
    'fields' => $this->dbEntityRepository->getFields()
  ]);
}
```

Une fois cela codé, je peux aller faire un print_r dans affichage-employes.php pour voir ce que contient $fields

```
<?php

// echo '<pre>'; print_r($data);echo '</pre>';
echo '<pre>'; print_r($fields);echo '</pre>';
?>
```

Je récupère bien mes champs d'entêtes. Je vais améliorer ma table

Je mets en commentaires le print_r et je modifie le script ainsi

```
<table class="table table-dark text-center my-5">
  <thead>
  <tr>
    <?php foreach($fields as $values): ?>
    <th><?= $values['Field'] ?></th>
    <?php endforeach; ?>
  </tr>
  </thead>
  <tbody>
  <?php foreach($data as $dataEmployes): ?>
    <tr>
      <td><?= implode('</td><td>', $dataEmployes) ?></td>
    </tr>
  <?php endforeach; ?>
  </tbody>
</table>
```

J'ajoute trois <th> pour modifier, voir et supprimer

```
<thead>
```

```
<tr>
  <?php foreach($fields as $values): ?>
  <th><?= $values['Field'] ?></th>
  <?php endforeach; ?>
  <th>Voir</th>
  <th>Modif</th>
  <th>Supp</th>
</tr>
</thead>
```

J'ajoute un lien pour les icones de BS dans le header de mon layout.php

```
<!-- links pour les icon bootstrap -->
 <link rel="stylesheet"
href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.4.0/font/bootstrap-icons.css">
```

Pour ajouter ceux dont j'ai besoin pour chaque employé dans <tbody>

```
<tbody>
<?php foreach($data as $dataEmployes): ?>
  <tr>
    <td><?= implode('</td><td>', $dataEmployes) ?></td>
    <td><i class="bi bi-eye"></i></td>
    <td><i class="bi bi-pencil"></i></td>
    <td><i class="bi bi-trash"></i></td>
  </tr>
<?php endforeach; ?>
</tbody>
```

J' ajoute une balise <a> autour de chaque icône (et je fais un peu de mise en forme BS pour chacun)

```
<tr>
  <td><?= implode('</td><td>', $dataEmployes) ?></td>
  <td><a href="?choixUser=select" class="btn btn-success"><i class="bi bi-
eye"></i></a></td>
  <td><a href="?choixUser=add" class="btn btn-warning"><i class="bi bi-
pencil"></i></a></td>
  <td><a href="?choixUser=delete" class="btn btn-danger"><i class="bi bi-
trash"></i></a></td>
</tr>
```

A présent, pour chaque action, je vais en plus de renseigner ce à quoi est égal choixUser, je vais devoir renseigner l'id pour les cibler individuellement

Je pourrai ecrire choixUser=select&id=<?= $dataemployes['idEmploye']?>

Sauf que ceci ne sera pas un code très générique.

Je devrai a chaque fois venir ici pour modifier cette information.

Je vais retourner dans Controller.php et ajouter un nouveau paramètre/indice pour la méthode selectAll

```
'id' => 'id' . ucfirst($this->dbEntityRepository->table)
```

Dans la mesure ou cet id s'appelle idEmploye, je concatène le mot id avec le nom de la table, et ucfirst me permet d'avoir un E majuscule

Voici le code pour selectAll

```
public function selectAll(){
  // echo "Méthode selectAll | Affiche tous les employés";
  // $resultat = $this->dbEntityRepository->selectAllEntityRepo();
  // echo '<pre>'; print_r($resultat); echo '</pre>';

  $this->render('layout.php', 'affichage-employes.php',[
    'title' => 'Affichage de tous les employés',
    'data' => $this->dbEntityRepository->selectAllEntityRepo(),
    'fields' => $this->dbEntityRepository->getFields(),
    'id' => 'id' . ucfirst($this->dbEntityRepository->table)
  ]);
}
```

11-8 Méthode selectEntityRepo

A présent je vais coder la méthode selectEntityRepo() (dans EntityRepository.php) qui va me permettre d'afficher un seul employé

Pour l'instant, j'ai ceci dans ma méthode select (dans Controller.php)

```
public function select(){

  echo "Méthode select | Affichage d'un seul employé'";

}
```

Si je clique sur l'icone œil, cela fonctionne bien, mon message s'affiche bien

Je vais la développer dans EntityRepository.php

Elle demande un argument ($id). Je le récupérerais dans l'URL

```
public function selectEntityRepo($id){

    $data = $this->getPdo()->query("SELECT * FROM $this->table WHERE id" .
ucfirst($this->table) . " = " . (int) $id);

  }
```

Je pointe avec $this vers ma méthode getPdo(). Je fais un query SELECT sur ma table de manière dynamique. Je cible un idEmploye particulier, celui qui aura le même que celui de l'URL

Je fais ensuite mon fetch etc...

```
public function selectEntityRepo($id){

  $data = $this->getPdo()->query("SELECT * FROM $this->table WHERE id" .
ucfirst($this->table) . " = " . (int) $id);
  $afficheUnEmploye = $data->fetchAll(\PDO::FETCH_ASSOC);
  return $afficheUnEmploye;

}
```

Une fois cela codé, je vais préparer son template dans view. Je le nomme detail-employe.php

Maintenant je vais coder dans Controller.php, ma méthode select dans laquelle je ferai appel a la méthode render()

Je dois dans un premier temps pour qu'elle fonctionne, récupérer l'id envoyé dans l'URL pour injecter sa valeur dans une variable qui servira ensuite d'argument pour selectEntityRepo()

```
public function select(){

  $id = (isset($_GET['id'])) ? $_GET['id'] : NULL ;

}
```

Ceci fait, j'appelle ma méthode render en lui fournissant toutes les données dont elle a besoin

```
public function select(){

  $id = (isset($_GET['id'])) ? $_GET['id'] : NULL ;

  $this->render('layout.php', 'detail-employe.php',[
    'title' => "Affichage de l'employé n° $id",
    'data' => $this->dbEntityRepository->selectEntityRepo($id),
  ]);

}
```

Je fais un print_r dans mon template et je clique ensuite sur l'icone œil. Ca fonctionne bien