



Le constructeur

Je vous avais montré dans le PDF précédent qu'il y avait moyen d'instancier une classe en mettant directement les attributs en paramètres.

```
$maVoiture = new Voiture("blanche", 1500.0, 7800 );
```

Certains d'entre vous l'avaient testé et s'étaient rendu compte que ça ne fonctionnait pas. Il n'assignait pas « blanche » à couleur, 1500.0 à poids et 7800 à prix. Il nous manquait quelques choses pour faire ça.

Je vous avais dit qu'on y reviendrait pour faire en sorte que ça fonctionne. C'est donc via les constructeurs qu'on pourra faire ce genre de chose.

Avant pour créer un objet et lui assigner des valeurs, on faisait comme ceci :

```
$maVoiture = new Voiture();  
$maVoiture->changerCouleur("Verte");  
$maVoiture->changerPoids(600.95);  
$maVoiture->changerPrix(5000);
```

On devait d'abord créer l'objet maVoiture avec les valeurs par défauts que j'ai fourni dans la classe Voiture. Ensuite assigner les nouvelles valeurs que je voulais pour chaque attribut en passant par les « **Setters** » (les méthodes de classe de modification/mutateur). Imaginons, nous voulons créer 20 voitures, ça fait vite très long, en plus si je rajoute des propriétés/attributs, ça rallonge le code. La solution pour éviter ça, viendra des constructeurs.

Lorsqu'on instancie notre classe, pour créer donc les différents objets, on fera appel à ce qu'on appelle un constructeur. C'est un peu comme une fonction qui a pour but de construire l'objet comme son nom l'indique.

Voici la syntaxe :

```
public function __construct(paramètres){  
    //assignations des valeurs  
}
```

Par défauts le constructeur « **__construct()** » existe sur tous les objets. C'est un constructeur vide, il permet juste de créer l'objet sans lui donner des valeurs externes. C'est grâce à ça qu'on peut l'appeler. Mais nous on veut faire en sorte que quand on crée notre objet voiture, on veut lui passer des paramètres.

```
public function __construct(string $uneCouleur, float $unPoids, int $unPrix){  
    $this->couleur = $uneCouleur;  
    $this->poids = $unPoids;  
    $this->prix = $unPrix;  
}
```

Ici donc le constructeur va recevoir en paramètre 3 valeurs qui correspondent à nos 3 attributs privé. Je précise bien le type pour qu'on ne se trompe pas, on aurait pu ne pas les préciser.

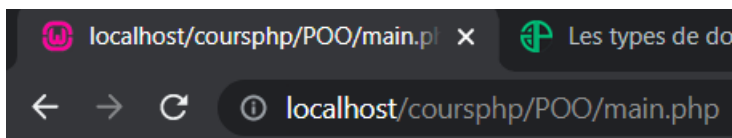
Il faut par contre bien préciser qu'on va modifier les attributs de l'objet courant grâce au mot clé « **\$this** ». L'objet courant que je suis entrain de créer, prend moi l'attribut « **couleur** » et remplace la par « **uneCouleur** » qu'on reçoit en paramètre, c'est bien entendu pareil pour les 2 autres attributs.

Maintenant que notre constructeur est fini, on va enfin pouvoir créer un objet voiture avec juste une ligne de code au lieu de 4 auparavant :

```
$maVoiture3 = new Voiture("Noire",1750.40,10000 );
```

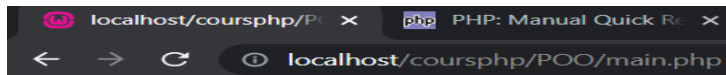
(Exos)

- 1) Affichez les informations de la voiture 3 de cette manière :



La voiture 3
Couleur : Noire
Poids : 1750.4 Kg
Prix : 10000 Euros

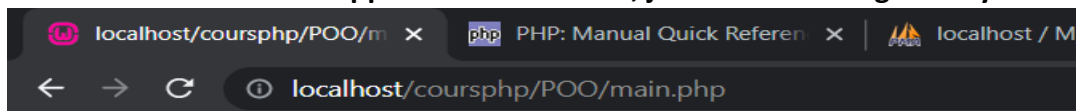
- 2) Créez 2 autres objets voiture avec des valeurs différentes.
- 3) Ajoutez un nouvel attribut/propriété privée, de type string qui s'appelle « marque ». Elle ne reçoit rien par défauts. Créez les Getters et Setters pour cet attribut. Modifiez votre constructeur en ajoutant cet attribut. Pour finir créez 2 voitures avec des valeurs différentes en utilisant ce nouveau constructeur. Affichez-moi ces 2 voitures dans votre navigateur de cette manière.



La voiture 6
Couleur : Jaune
Poids : 1300.5 Kg
Prix : 8000 Euros
Marque : Audi

La voiture 7
Couleur : Brune
Poids : 1900.5 Kg
Prix : 16000 Euros
Marque : BMW

- 4) (Difficile) Créez une méthode « plusChere() » qui reçoit une voiture en paramètre et qui retourne un booléen. Si la voiture courante est plus chère que la voiture en paramètre ça nous retourne vrai sinon faux. Ensuite dans votre main.php en fonction du résultat de l'appelle de la méthode, je veux un message du style :



La voiture Brune est plus chère que la voiture Jaune
Il y a une difference de prix de 8000 Euros

La méthode toString()

Jusqu'à présent quand je vous demandais de m'afficher les données d'une voiture vous faisiez quelque chose comme ça :

```
$maVoiture6 = new Voiture("Jaune",1300.5,8000,"Audi");  
echo "La voiture 6 <br>";  
echo "Couleur : " . $maVoiture6->obtenirCouleur() . "<br>";  
echo "Poids : " . $maVoiture6->obtenirPoids() . " Kg<br>";  
echo "Prix : " . $maVoiture6->obtenirPrix() . " Euros<br>";  
echo "Marque : " . $maVoiture6->getMarque() . "<br>";
```

Et si je vous demandais de créer 20 voitures et de les afficher à chaque fois, ou encore si j'avais 10 attributs au lieu de 4, on ne s'en sortirait plus pour devoir afficher tout ça.

Heureusement on a la possibilité de simplifier notre affichage et de le modeler comme on le souhaite. Ce sera grâce à la méthode « **__toString()** ». On peut la retrouver dans la documentation officielle de PHP.

La méthode `__toString()` détermine comment l'objet doit réagir lorsqu'il est traité comme une chaîne de caractères. Par exemple, ce que « `echo $maVoiture ;` » affichera.

Essayez de faire ça :

```
$maVoiture8 = new Voiture("Grise", 975.8, 6450, "Mercedes");  
echo $maVoiture8;
```

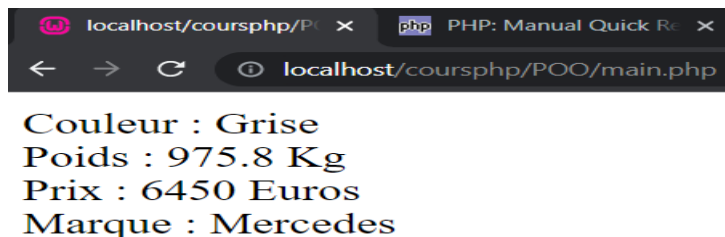
Que remarquez-vous ?

Il vous sort une erreur disant comme quoi on ne peut pas convertir un objet en une chaîne de caractère. Je ne vais pas rentrer dans les détails mais il faut savoir que `__construct()`, `__toString()`, et d'autres, ce sont ce qu'on appelle des méthodes magiques parce qu'elles sont présentes dans tous les objets. On doit juste les remplir en fonction de nos attributs ou méthodes de classe.

On va donc devoir définir comment je veux qu'on affiche mon objet voiture. Voici du coup ce que vous ajouterez à votre classe Voiture :

```
public function __toString() : string{  
    return "Couleur : " . $this->couleur."<br>".  
           "Poids : " . $this->poids." Kg<br>".  
           "Prix : " . $this->prix ." Euros<br>".  
           "Marque : " . $this->marque ."<br>";  
}
```

Maintenant si vous relancer votre navigateur vous verrez qu'on aura juste besoin de faire un `echo` suivi de notre objet et il affichera de cette manière.



```
Couleur : Grise  
Poids : 975.8 Kg  
Prix : 6450 Euros  
Marque : Mercedes
```

On voit bien ici que c'est ce que j'ai mis dans la méthode `__toString()`, donc si on veut changer l'affichage de n'importe quel objet de classe, il suffira de créer une méthode `__toString()` et de retourner une chaîne de caractères avec les attributs de classe.

(Exos)

- 5) (Difficile) Créez une nouvelle classe « Personne » dans un nouveau fichier. Elle aura 3 attributs (prenom, age, sexe). N'oubliez pas de rajouter les Getters et Setters ainsi que le constructeur. Ajoutez une méthode `__toString()`. Dans votre « main » créez 5 objets personnes. Créez une méthode « `estPlusAge(Personne $unePersonne) : bool{}` » qui retourne un booléen pour dire si une personne est plus âgée qu'une autre personne. Affichez les personnes et testé votre méthode sur 2 personnes. (bonus) Essayez d'utiliser une boucle pour afficher les personnes.

Prénom : Julien
Age : 34 ans
Sexe : M

Prénom : Sarah
Age : 27 ans
Sexe : F

Prénom : Mohamed
Age : 15 ans
Sexe : M

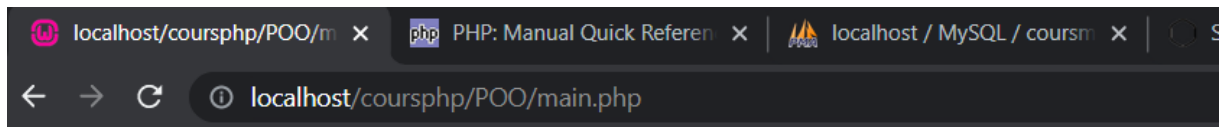
Prénom : Konchita
Age : 40 ans
Sexe : X

Prénom : Bruce
Age : 45 ans
Sexe : M

Konchita a 40 ans et est plus âgée que Sarah qui a 27 ans

- 6) Créez une classe « Personnage » qui à 3 attributs (nom, vie et attaque). Créez le constructeur, les setters, les getters et la méthode `toString()`. Regardez l'image ci-dessous pour voir comment se fait l'affichage dans le `toString()`. Créez une méthode `estVivant()` qui reçoit rien en paramètres et qui vérifie si le personnage courant a encore des points de vie et retourne un booléen. Créez une méthode « `lanceAttaque(Personnage $unPersonnage) : void{}` ». Elle enlève au personnage reçu en paramètres de la vie avec le nombre de point d'attaque du personnage courant (un calcule). Affichez tous les personnages. Ensuite créez une boucle qui va faire en sorte qu'un personnage s'affronte contre un autre personnage et qu'à la fin (c'est-à-dire quand un personnage n'a plus assez de vie) il quitte la boucle et nous affiche quel personnage a gagné et en combien de coups.

Attention Suite sur la page suivante...



Nom : Ryu

Vie : 70

Puissance d'attaque : 24 de dégats

Nom : Ken

Vie : 60

Puissance d'attaque : 20 de dégats

Nom : Sub-Zero

Vie : 80

Puissance d'attaque : 48 de dégats

Nom : Jin Kazama

Vie : 100

Puissance d'attaque : 46 de dégats

Nom : Mario

Vie : 60

Puissance d'attaque : 42 de dégats

Le personnage Jin Kazama a battu le personnage Ryu en 2 coups.
Et il lui reste 76 de points de vie