Secteur Tertiaire Informatique

Filière « Etude et développement »

Séquence « Développer des pages Web en lien avec une base de données »

Programmation orientée objet en PHP

Apprentissage

Mise en pratique

Evaluation

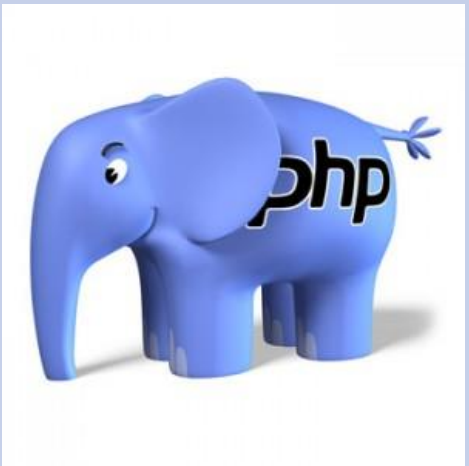

TABLE DES MATIERES

Préambule

Objectifs

Méthodologie

1. LA CLASSE VOITURE

Il s'agit ici de développer un script PHP représentant (de manière simplifiée !) une voiture automobile en s'inspirant des éléments mentionnés dans les supports d'initiation au langage PHP orienté objet.

L'application minimale envisagée ici se limite donc à deux scripts : la définition de la classe Voiture d'une part, et un script procédural utilisant cette classe.

Comme on souhaite que cette classe reste indépendante de son usage par un script particulier, elle n'effectue aucun 'affichage' direct (pas de `echo`) ; les messages générés seront stockés dans un attribut accessible par le getter correspondant, et c'est bien le programme principal qui saura les récupérer et les restituer comme bon lui semble à l'utilisateur.

1.1 SCRIPT ET BLOC DE CLASSE

Créer un script PHP '`Voiture.php`' dans un dossier publié par votre serveur Apache/PHP.

Saisir un 'cartouche' avec la documentation générale de la classe.

Saisir la structure PHP de classe de base.

Enregistrer.

1.2 ATTRIBUTS PRIVES

Définir les différents attributs privés nécessaires pour représenter les données caractéristiques d'un véhicule. On se limitera à :

- Immatriculation – `string`
- Couleur – `string`
- Poids – `int`
- Puissance – `int`
- Capacité du réservoir – `float`
- Niveau d'essence – `float`
- Nombre de place – `int`
- Assuré – `bool`
- Message au tableau de bord – `string`

1.3 CONSTRUCTEUR

Définir le constructeur de la classe qui permet de définir dès l'instanciation d'un véhicule ce qui ne pourra changer :

- Immatriculation
- Couleur
- Poids
- Puissance
- Capacité du réservoir
- Nombre de place

Le réservoir est livré avec 5 litres de carburant et le véhicule n'est pas assuré à la livraison mais le tableau de bord peut déjà afficher un message d'accueil ; initialiser les attributs en conséquence dans ce constructeur.

1.4 GETTERS

Définir des fonctions getters standards (sans contrôle) pour chacun des attributs.

1.5 SETTER

Pour notre voiture, seul l'indicateur d'assurance peut être affecté directement de l'extérieur. Ecrire le setter correspondant :

- Mise à jour de l'indicateur booléen d'assurance ;
- Génération d'un message pour le tableau de bord.

2. TEST INTERMEDIAIRE

Créer le script PHP principal dans le même dossier et saisir uniquement :

- L'entête générale HTML de la page à afficher ;
- L'instruction PHP permettant de fusionner le script définissant la classe Voiture ;
- Le bas de page HTML.

Enregistrer le tout et vérifier la bonne construction et l'absence d'erreurs de syntaxe en lançant l'exécution du script PHP principal.

Corriger au besoin.

Ajouter dans le programme principal l'instanciation de la voiture de vos rêves en précisant les caractéristiques requises par le constructeur. Poursuivre le développement en assurant le véhicule (c'est obligatoire !).

Ajouter une instruction `var_dump()` pour afficher le contenu de l'objet instancié. Tester et mettre au point si nécessaire.

3. METHODES DE SERVICE

3.1 REPEINDRE LA VOITURE

Définir la fonction `Repeindre()` qui admet en paramètre le libellé de la nouvelle couleur.

Si cette nouvelle couleur est identique à la couleur courante, générer simplement un message remerciant pour ce rafraîchissement ; s'il s'agit d'une couleur différente, la mémoriser dans l'attribut correspondant et générer un autre message de remerciement. Cette méthode pourrait aussi bien retourner un code d'erreur, par exemple un booléen `true` si tout va bien ou `false` sinon.

On peut (doit) aussi tester dès le début d'une fonction que le ou les paramètres requis ont bien été reçus. Pour éviter toute erreur à l'exécution, il est nécessaire de déclarer les paramètres optionnels dans la fonction et de tester leur existence grâce à la fonction intégrée `isset()`.

Appliquer cette construction pour tester le paramètre de couleur de peinture.

Dans le programme principal, invoquer cette méthode `Repeindre()` pour repeindre la voiture de vos rêves et afficher de suite un message HTML clair à l'utilisateur en récupérant le message mis à disposition par l'objet voiture.

NB : après chacun des compléments à venir, procéder de même pour tester l'utilisation de la nouvelle fonction ; ne jamais effectuer 'd'affichage direct' par l'objet lui-même et 'habiller' le message récupéré dans une structure HTML qui convient ; tester à fond en observant tout à la fois les affichages et le code source HTML reçu.

NB : les tests sur le passage des paramètres est assez lourd ; on peut pour cet exercice se limiter à n'effectuer les contrôles que pour cette fonction `Repeindre()`.

3.2 FAIRE L'APPOINT D'ESSENCE

Définir la fonction `Mettre_essence()` qui admet en paramètre une quantité de carburant à ajouter au réservoir.

Dans cette fonction, effectuer tout d'abord un test pour savoir si cette quantité est compatible avec le niveau d'essence courant du véhicule ; si tout va bien, ajouter le carburant et générer un message de feed-back, sinon, refuser cet appoint de carburant et générer un message d'erreur.

Cette fonction doit retourner au programme appelant le nouveau niveau de carburant.

Tester en ajoutant successivement plusieurs quantités de carburant par le programme principal.

3.3 SE DEPLACER

C'est bien le but de notre véhicule, alors allons-y !

Définir la méthode `Se_deplacer()` qui admet en paramètres la distance parcourue en km et la vitesse moyenne du déplacement.

En fonction de ces deux paramètres, calculer tout d'abord la consommation de carburant selon la règle de gestion (très simplifiée) :

- Consommation de 10 l aux 100 km en ville, soit à une vitesse moyenne inférieure à 50 km/h ;
- Consommation de 5 l aux 100 km en sur route, soit à une vitesse moyenne comprise entre 50 et 90 km/h ;
- Consommation de 8 l aux 100 km en sur autoroute, soit à une vitesse moyenne comprise entre 90 et 130 km/h ;
- Consommation de 12 l aux 100 km pour une vitesse moyenne supérieure à 130 km/h (et on ne parle pas des retraits de points de permis…) ;

NB : ce calcul élémentaire de consommation pourrait bien faire l'objet d'une nouvelle méthode privée qui serait alors réutilisable par la classe elle-même…

D'après cette consommation nécessaire de carburant, déterminer si ce trajet est possible en fonction du niveau d'essence courant. Si ce n'est pas le cas, générer un message d'erreur. Et si tout va bien, déduire la consommation calculée et générer un message indiquant cette consommation.

Tester en invoquant plusieurs fois cette méthode pour des trajets divers successifs depuis le programme principal.

3.4 METHODE __TOSTRING()

Un objet doit toujours pouvoir se restituer 'en clair', aussi bien pour l'utilisateur que pour des besoins de service (l'interpréteur PHP effectue parfois des conversions à l'insu de votre plein gré…). Il est donc de bonne pratique de définir une fonction de ce type dans chaque classe. En PHP, il existe un nom de fonction réservé : `__toString()`

Dans notre cas, cette méthode retournera un libellé rappelant simplement l'immatriculation, la puissance et la couleur du véhicule. Cela peut être l'occasion de construire un beau message à base de chaîne formatés par la fonction `sprinf()`…

Dans la programme principal, il n'est même plus nécessaire d'invoquer explicitement la méthode `__toString()` car elle fait partie de ces 'méthodes magiques' appelées automatiquement par l'interpréteur ; on peut aussi bien écrire :

```
echo $maVoiture ;
```

ou

```
echo $maVoiture->__toString() ;
```

4. ET POUR ALLER PLUS LOIN

Sur la base de cet exercice, il est encore possible de :

- Instancier plusieurs objets voiture dans le programme principal et les manipuler par leur référence ;
- Créer des classes dérivées comme des Fourgon ou PoidsLourd qui peuvent être enrichies de membres supplémentaires (volume du fourgon, nombre d'essieux du camion…) ;
- Créer une nouvelle classe indépendante (Personne) mise en relation avec les véhicules par échange de références ;
- … et bien plus encore…

CREDITS

ŒUVRE COLLECTIVE DE l'AFPA
Sous le pilotage de la DIIP et du centre d'ingénierie sectoriel Tertiaire-Services

Equipe de conception (IF, formateur, mediatiseur)
B. Hézard – formateur
M. Fayolle - formateur
Chantal Perrachon – Ingénieure de formation