JACK FELLERS

php

LE GUIDE COMPLET DE LA PROGRAMMATION ET DU LANGAGE DES OBJETS EN 7 JOURS



WEBHAWK

PHP

JACK FELLERS

Cher lecteur, chère lectrice, pour te remercier de la confiance que tu m'as témoignée en achetant mon livre, voici un **cadeau pour** toi, un guide pour fortifier encore plus tes connaissances en programmation Web !

Scannez le code ou cliquez sur le lien pour l'échanger en moins d'une minute :

Lien alternatif pour le code QR :

https://webhawk.tech/optin-it/

Bonne lecture !

TABLE DES MATIÈRES

Avant-propos

1. Php
2. Installation
3. Code PHP
4. Types de données
5. Cycles
6. Fonctions
7. Objets
8. Exceptions
9. Référence de l'objet
10. Méthodes et classes
11. Classes avancées
12. Classes et interfaces abstraites

Conclusions

AVANT-PROPOS

Au fil des ans, de nombreuses langues se sont succédé, certaines pendant une courte période, d'autres plus durablement. PHP est certainement un langage considérablement ancien, dont la première version remonte à 1994. Au fil des ans, plusieurs autres langages ont tenté de le remplacer. En effet, certains s'en sont approchés, d'autres ont échoué. Cependant, PHP est un langage qui a continué à évoluer avec son temps. En fait, il est reconnu pour avoir facilité le développement des programmeurs Web, en ajoutant des fonctionnalités nouvelles et passionnantes qui ont été optimisées de façon excellente.

Le PHP n'est pas un langage difficile, mais il n'est pas non plus extrêmement facile, car chaque site Web est distinct. Il est alors impossible de prendre un script, de le placer sur un site Web et de s'attendre à ce qu'il fonctionne. En fait, l'objectif est d'assister les concepteurs de sites Web ayant peu ou pas de connaissances en programmation à acquérir la confiance requise afin de plonger dans le code et, par la suite, de l'adapter à leurs propres besoins.

1

PHP

Officiellement, notez que PHP est l'abréviation de *PHP Hypertext Preprocessor*.

Ce livre a pour objectif de vous permettre à programmer avec PHP et, ce faisant, de vous faire comprendre ce qui rend les programmeurs PHP si heureux et la raison de leur utilisation. PHP est un langage de script qui donne vie aux sites Web de plusieurs façons énumérées ci-dessous :

- Télécharger des fichiers via une page Web ;
- Générer des vignettes à partir d'images plus grandes ;
- Lecture et écriture dans des fichiers ;
- Utiliser une base de données afin d'afficher et de stocker des informations ;
- Rendre les sites Web indexables ;
- Affichez et mettez à jour les informations de façon dynamique ;
- Et bien plus encore.

PHP est simple à apprendre. Il est indépendant de la plateforme, de sorte que le même code fonctionne sous Windows, Mac OS X, Linux et tout autre logiciel que vous devez développer.

Notez que PHP est un logiciel libre, il est alors gratuit. Il s'agit de l'une des technologies les plus utilisées afin de créer des sites Web dynamiques, mais l'un de ses principaux objectifs était de créer un livre qui recueille des informations à partir d'un formulaire en ligne et les affiche sur une page Web.

Le PHP est utilisé afin de créer du contenu dynamique par plus de 80 % des 10 millions de sites Web qu'il analyse de manière régulière. En fait, c'est le langage qui pilote les systèmes de gestion de contenu CMS les plus populaires comme Drupal, Joomla ! et WordPress. En outre, il est aussi utilisé par Facebook et Wikipédia.

Cela suffirait à donner confiance à un langage, mais l'un des grands attraits de PHP, toutefois, est qu'il reste fidèle à ses racines. En effet, le créateur original de PHP, Rasmus Lerdorf, l'a décrit comme : « *un langage de script très convivial pour les programmeurs, adapté aux personnes ayant peu ou pas d'expérience en programmation ainsi qu'aux développeurs web expérimentés qui doivent résoudre des problèmes rapidement*. » Vous pouvez débuter à écrire des scripts intéressants sans devoir apprendre beaucoup de théorie, mais vous savez que la même technologie peut aussi être utilisée de façon industrielle ou pour d'importants projets considérablement plus grands.

Un bref aperçu du monde de PHP

PHP a été conçu à l'origine pour être intégré dans le code HTML d'une page Web, et c'est ainsi qu'il est fréquemment utilisé. Par exemple, si vous voulez afficher l'année en cours dans un avis de droit d'auteur, vous êtes en mesure de l'insérer dans le pied de page comme suit :

 &copy ; < ? php **echo date**('Y') ; ?> **Copyright **

Sur un serveur Web compatible avec PHP, le code entre les balises < ? php et ?> est automatiquement traité, et affiche l'année en cours de la façon démontrée ci-dessous :



Ce n'est qu'un exemple simple, mais il démontre quelques des avantages de l'utilisation de PHP :

- Chaque utilisateur verra l'année correcte parce que c'est le serveur qui la définit ;
- La date est calculée par le serveur Web, elle n'est alors pas influencée par l'horloge de l'ordinateur de l'utilisateur, qui peut être mal réglée.

Bien qu'il soit pratique d'incorporer le code PHP dans le HTML de cette façon, cela est répétitif et peut provoquer des erreurs. Il peut aussi rendre complexe la gestion de vos pages Web, notamment quand vous commencez à utiliser un code PHP plus difficile. Par conséquent, il est logique de stocker le code dans des fichiers différents, et ensuite d'utiliser PHP afin de créer vos pages à partir des composants distincts.

Il est important de noter que ces fichiers séparés sont en mesure de contenir uniquement du PHP, uniquement du HTML ou une combinaison des deux, créant ainsi de véritables composants réutilisables.

À titre d'exemple, vous pouvez placer le menu de navigation de votre site Web dans un fichier à inclure et utiliser PHP afin de l'inclure dans chaque page. Chacune des fois que vous devez apporter des changements

au menu, vous ne changez qu'un seul fichier et les changements sont automatiquement répercutés sur chaque page qui inclut le menu. Imaginez le temps que vous pouvez gagner sur un site Web de nombreuses centaines de pages !

Avec une page HTML normale, le contenu est défini par le développeur Web au stade de la conception et téléchargé sur le serveur Web. Quand quelqu'un visite la page, le serveur Web envoie tout simplement le HTML et d'autres ressources, comme des images et des feuilles de style. Il s'agit d'une architecture vraiment facile : la demande provient du navigateur et le contenu, défini par le programmeur, est renvoyé par le serveur. Quand vous créez des pages Web avec PHP, ce n'est pas si facile, car quand un site Web basé sur PHP est visité, la séquence d'événements (énumérée ci-dessous) est déclenchée :

1. Le navigateur envoie une requête au serveur Web ;
2. Le serveur Web transmet la demande au moteur PHP, qui est intégré au serveur ;
3. Le moteur PHP traite le code. Dans de nombreux cas, les requêtes sont aussi effectuées sur une base de données ;
4. Le serveur envoie la page complète au navigateur.

De plus, notez que ce processus ne prend en général que quelques fractions de seconde, de sorte que le visiteur d'un site Web PHP ne remarquera probablement aucun retard. Effectivement, comme chaque page est construite individuellement, les sites PHP sont en mesure de répondre à l'entrée de l'utilisateur en affichant un contenu spécifique, par exemple, après s'être connecté au site.

PHP est un langage côté serveur et le code PHP reste sur le serveur Web. Après son traitement, le serveur n'envoie que la sortie du script au navigateur du visiteur. Il s'agit normalement de HTML, mais PHP peut

aussi être employé afin de générer d'autres langages Web, comme JSON (JavaScript Object Notation).

De plus, remarquez que PHP vous permet d'introduire une logique dans vos pages Web en fonction d'alternatives. Quelques décisions sont prises à partir des informations que PHP collecte sur le serveur comme la date, l'heure, le jour de la semaine, les informations contenues dans l'URL de la page, et ainsi de suite. Supposons que nous créions un site Web avec les programmes TV du jour, nous sommes certains que si nous sommes mercredi, il montrera les programmes TV du mercredi. D'autres fois, les décisions sont basées sur les entrées des utilisateurs, que PHP extrait des formulaires en ligne. En effet, si vous vous êtes inscrit sur un site, des informations personnalisées, comme le prénom, le nom, la photo de profil, et ainsi de suite, seront affichées. Cette méthode est sûre parce que le site Web ne reçoit que la sortie d'un script, mais ne sait pas à quoi ressemble le script.

Mais discutons un peu de sécurité.

Un exemple est de comparer le PHP est à l'électricité ou bien les couteaux de cuisine dans votre maison : manipulés correctement, ils sont très sûrs ; manipulés de façon irresponsable, ils peuvent causer beaucoup de dégâts. PHP n'est pas dangereux, et il n'est pas nécessaire de devenir un expert en sécurité afin de l'utiliser. L'important est de comprendre le principe de base de la sécurité de PHP. En effet, il faut toujours vérifier les entrées de l'utilisateur avant de les traiter. La majorité des risques de sécurité peuvent être éliminés avec un effort minimal. La meilleure manière de se protéger est de comprendre le code que vous utilisez et de se mettre à la place d'un utilisateur malveillant qui souhaiterait voler des données sur votre serveur.

Les outils

Pour être précis, aucun logiciel spécial n'est nécessaire afin d'écrire des scripts PHP.

En fait, le code PHP est un simple texte et peut être créé dans n'importe quel éditeur de texte, comme Notepad sous Windows ou TextEdit sous Mac OS X. Cela dit, la programmation sera considérablement plus simple si vous utilisez un logiciel doté de fonctions conçues afin d'accélérer le processus de développement. Vous pouvez compter sur une gamme vraiment large, tant gratuite que payante.

Grâce à ces outils, s'il y a une erreur dans votre code, vous serez en mesure de vous en rendre compte bien avant d'arriver au navigateur, mais directement dans la phase de programmation et non dans la phase de test. Vous devez choisir un éditeur qui présente les caractéristiques énumérées ci-dessous :

- Coloration de la syntaxe PHP : le code est mis en évidence dans des couleurs distinctes en fonction de son rôle. En effet, si votre code a une couleur rouge ou inattendue, vous avez fort probablement commis une erreur ;
- Contrôle syntaxique PHP : les contrôleurs syntaxiques surveillent le code lorsqu'il est tapé et mettent en évidence à l'instant même les erreurs, ce qui aide d'économiser beaucoup de temps et de frustration ;
- Numérotation des lignes : trouver rapidement une ligne spécifique simplifie énormément le dépannage ;
- Conseils pour le code PHP : PHP possède tellement de fonctions intégrées qu'il peut être difficile de se rappeler la manière de les utiliser, même pour un utilisateur expérimenté. Plusieurs éditeurs de scripts présentent automatiquement des conseils avec des rappels sur le fonctionnement ou les paramètres d'entrée ;

- Une fonction afin d'équilibrer les parenthèses : les parenthèses (()), les crochets ([]) et les accolades ({}) doivent toujours être dans des paires assorties. Il est simple d'oublier de fermer une paire, toutefois, les éditeurs de script vous permettent de résoudre ce souci.

Il est important de savoir que même si vous ne prévoyez pas de développer beaucoup de code en PHP, vous devriez envisager d'utiliser un éditeur dédié si votre programme de développement Web ne prend pas en charge le contrôle syntaxique. Les éditeurs discutés ci-dessous disposent de l'ensemble des fonctionnalités essentielles, comme la vérification de la syntaxe et les suggestions de code. Notez qu'il ne s'agit pas d'une liste exhaustive, mais plutôt d'une expérience personnelle.

- Atom : c'est un IDE très personnalisable, il existe un plugin pour tout, y compris pour PHP ;
- Komodo : IDE gratuit et open source pour PHP et un certain nombre d'autres langages informatiques populaires. Notez qu'il est disponible pour Windows, Mac OS X et Linux ;
- PhpStorm : est un programme d'édition PHP dédié et propose un très bon support pour HTML, CSS et JavaScript. C'est actuellement mon programme favori pour le développement avec PHP ;
- Sublime : Si vous êtes un fan de Sublime Text, il se trouve des plug-ins pour la coloration de la syntaxe PHP, la vérification de la syntaxe et la documentation ;
- Zend Studio : Si vous êtes très sérieux au sujet du développement PHP, Zend Studio est l'environnement de développement intégré (IDE) le plus complet pour PHP. Zend Studio fonctionne sous Windows, Mac OS X et Linux. Dans le

passé, il était dispendieux, mais le prix pour les développeurs individuels est maintenant beaucoup moins élevé.

Maintenant, nous sommes prêts à explorer et débuter le développement.

2

INSTALLATION

Maintenant que vous avez choisi d'utiliser PHP afin d'enrichir vos pages Web, vous devez vous assurer que vous avez tout ce qu'il vous faut afin de commencer. Bien que vous puissiez tout tester sur votre serveur distant, il est en général plus pratique de tester les pages PHP sur votre ordinateur local : tout ce que vous devez installer est gratuit. Dans ce chapitre, j'expliquerai les options distinctes pour Windows et Mac OS X. Les composants nécessaires sont généralement installés par défaut sous Linux.

La manière la plus facile de savoir si votre site Web prend en charge PHP est de le demander à votre hébergeur, mais PHP est en général pris en charge par défaut. L'autre façon de le savoir est de télécharger une page PHP sur votre site Web et de constater si elle fonctionne. Même si vous savez déjà que votre site prend en charge PHP, faites le test ci-dessous afin de comprendre quelle version est en cours d'exécution :

1. Ouvrez un éditeur de texte, par exemple Notepad ou TextEdit, et tapez le code ci-dessous dans une page blanche : < ? php **echo** phpversion () ; ?>
2. Enregistrez le fichier sous le nom de *phpversion.php*. Il est essentiel de s'assurer que le système d'exploitation utilisé

n’ajoute pas l’extension . *txt* après . *php*. Notez que les utilisateurs de Mac doivent aussi s’assurer que TextEdit n’enregistre pas le fichier en RTF (Rich Text Format) ;

3. Téléchargez *phpversion.php* sur votre site Web comme vous le feriez pour une page HTML, et ensuite tapez l’URL dans un navigateur. En supposant que vous avez téléchargé le fichier au niveau supérieur de votre site, l’URL ressemblera à ce qui suit : http://www.tuosito.it/phpversion.php. Si vous observez un nombre en trois parties sur l’écran, comme 5.6.1, PHP est activé. Ce numéro montre quelle version de PHP est exécutée sur votre serveur. Vous devez avoir au moins la version 5.4.0 pour les fonctions de ce livre électronique ;
4. Si vous recevez un message d’erreur, PHP est peut-être pris en charge, mais vous avez commis une erreur en tapant le code dans le fichier ;
5. Si vous n’observez que le code que vous avez saisi sans aucune version de PHP, cela veut dire que PHP n’est pas pris en charge.

Si votre serveur utilise PHP 5.3 ou bien une version antérieure, veuillez le contacter afin de l’informer que vous souhaitez la dernière version de PHP.

Contrairement aux pages Web normales, vous ne pouvez pas simplement double-cliquer sur les pages PHP et les présenter dans votre navigateur. Ces pages nécessitent d’être analysées ou bien traitées via un serveur Web supportant PHP. Si votre hébergeur prend en charge PHP, vous pouvez télécharger vos fichiers sur votre site Web et les tester à distance. Toutefois, vous devez télécharger le fichier chaque fois que vous apportez une modification. Si vous voulez travailler tout de suite avec PHP, utilisez votre site Web comme banc d’essai. Vous découvrirez sous peu la nécessité d’un environnement de test PHP local, alors analysons la façon de procéder.

Afin de tester les pages PHP sur votre ordinateur local, vous devez installer les éléments ci-dessous :

- Un serveur Web : logiciel qui démontre des pages Web qui peuvent résider sur votre propre ordinateur ;
- PHP ;
- MySQL et un frontal pour MySQL appelé *phpMyAdmin*, qui sont nécessaires afin de travailler avec une base de données.

Pendant plusieurs années, j'ai conseillé d'installer chaque composant d'un environnement de test PHP séparément, au lieu d'utiliser un paquet contenant *Apache*, *PHP*, *MySQL* et *phpMyAdmin* en une seule fois. Au tout début, ils étaient simples à installer, mais il était pratiquement impossible de les désinstaller ou bien de les mettre à jour. Les packages tout-en-un actuellement disponibles sont formidables et je n'ai aucune hésitation à les conseiller ; en fait, sur mes ordinateurs, j'utilise **XAMPP** pour Windows et **MAMP** pour Mac OS X. D'autres forfaits sont disponibles, quel que soit celui que vous choisissez.

Windows

Par défaut, la majorité des ordinateurs Windows cachent l'extension à trois ou bien quatre lettres, comme *.doc* ou *.html*, de sorte que tout ce que vous observez dans les boîtes de dialogue ainsi que dans l'explorateur de fichiers de Windows cache son extension. Windows 8 affiche l'extension de fichier pour les fichiers PHP, mais il est utile d'activer l'affichage de l'extension du nom pour tous les fichiers. Utilisez ces instructions afin d'activer l'affichage des extensions de fichiers dans Windows 8/10 :

1. Ouvrez l'Explorateur de fichiers ;

2. Sélectionnez View afin de développer le ruban qui se trouve en haut de la fenêtre de l'explorateur de fichiers ;
3. Cochez la case « Extensions de nom de fichier ».

En visualisant l'extension du fichier, vous êtes plus sûr : vous êtes en mesure de savoir si un document d'apparence innocente est en fait un virus.

La plupart des installations PHP fonctionnent sur le serveur Web Apache. Les deux sont open source et fonctionnent bien ensemble. Toutefois, Windows a son propre serveur Web, Internet Information Services (IIS), qui supporte aussi PHP. Notez que Microsoft a travaillé en étroite collaboration avec l'équipe de développement de PHP pour améliorer les performances de PHP sur IIS pour qu'elles correspondent à celles d'Apache. Donc, lequel choisir ? La réponse dépend du fait que vous développez des pages Web en employant ASP ou ASP.NET ou bien que vous ayez l'intention de le faire. À moins que vous n'ayez besoin de IIS pour ASP ou ASP.NET, je conseille d'installer Apache, en utilisant XAMPP ou bien l'un des autres paquets tout-en-un. Notez que si vous devez utiliser IIS, la manière la plus pratique d'installer PHP est d'utiliser le *Microsoft Web Platform Installer* (Web PI).

Il est important de savoir qu'il existe trois paquets populaires pour Windows qui installent Apache, PHP, MySQL, phpMyAdmin et plusieurs autres outils sur votre ordinateur en une seule fois : XAMPP, WampServer et EasyPHP. Le processus d'installation ne prend généralement que quelques minutes. Une fois le paquet installé, il peut être nécessaire de changer quelques paramètres, comme nous l'analyserons sous peu.

MacOS X

Le serveur Web Apache et PHP sont tous deux préinstallés sur Mac OS X, mais ne sont pas activés par défaut. Plutôt que d'utiliser les versions

préinstallées, je conseille d'utiliser **MAMP,** qui installe Apache, PHP, MySQL, phpMyAdmin et plusieurs autres outils en une seule opération. Afin d'éviter les conflits avec les versions préinstallées d'Apache et de PHP, MAMP place l'ensemble des applications dans un dossier dédié sur le disque dur, ce qui permet de tout désinstaller simplement en faisant glisser de façon simple le dossier MAMP vers la corbeille si nécessaire.

Avant de débuter, vous devez vous assurer que vous êtes connecté à votre ordinateur avec des privilèges d'administrateur.

- Téléchargez l'image MAMP gratuite ;
- Quand le téléchargement est terminé, lancez l'image disque et continuez en acceptant le contrat de licence. Continuer afin de poursuivre l'installation de l'image disque ;
- Suivez les instructions illustrées ;
- Vérifiez que MAMP a été installé dans le dossier Applications.

MAMP utilise par défaut des ports non standard pour Apache et MySQL. À moins que vous n'utilisiez de nombreuses installations d'Apache et de MySQL, vous deviez changer les paramètres des ports comme suit :

1. Double-cliquez sur l'icône MAMP sous Applications / MAMP. Votre navigateur par défaut devrait démarrer et vous afficher la page de bienvenue de MAMP. Remarquez que l'URL dans la barre d'adresse du navigateur débute par localhost:8888. Cela signifie que le système est exécuté localement et sur le port 8888, cela veut dire qu'il indique qu'Apache attend des requêtes sur le port non standard 8888 ;
2. Ouvrez le panneau de configuration de MAMP, qui doit être exécuté sur le bureau. Les petites lumières vertes à côté du

serveur Apache et du serveur MySQL signifient que les deux serveurs fonctionnent ;
3. Cliquez sur l'icône *Préférences* et sélectionnez l'onglet *Ports* en haut du panneau. Vous y observez les ports sur lesquels Apache et MySQL fonctionnent, à savoir 8888 et 8889 ;
4. Cliquez sur 'Set Web and MySQL ports to 80 and 3306' afin de définir la configuration standard ;
5. Cliquez sur OK et entrez le mot de passe du Mac quand vous y êtes invité afin que MAMP puisse redémarrer les deux serveurs ;
6. Quand les deux voyants sont à nouveau verts, cliquez sur "Ouvrir la page de bienvenue" dans le panneau de configuration de MAMP. Cela rechargera la page d'accueil de MAMP dans votre navigateur. Cette fois, l'URL ne doit pas comporter de deux-points suivis d'un chiffre après localhost parce qu'Apache écoute maintenant sur le port par défaut.

Où sont les fichiers ?

Vous devez créer les fichiers à un endroit où le serveur Web peut les traiter. En général, cela veut dire que les fichiers doivent se trouver à un niveau élevé du serveur ou dans un sous-répertoire de ce niveau. Le chemin par défaut pour les paramètres les plus courants est comme suit :

- XAMPP : C:\xampp\htdocs
- WampServer : C:\wamp\ww
- EasyPHP : C:\EasyPHP\ww
- IIS : C:\inetpub\wwwroot
- MAMP : Macintosh HD:Applications:MAMP:htdocs

Il est important de savoir qu'afin d'afficher une page PHP, elle doit être chargée dans un navigateur à l'aide d'une URL. L'URL de la page principale du serveur Web dans l'environnement local est http://localhost/.

Vérifier l'installation

Après avoir installé PHP, il est excellent de vérifier ses paramètres de configuration. En plus des fonctionnalités principales, PHP dispose beaucoup d'extensions optionnelles. Tant le paquet tout-en-un que Microsoft Web PI installent l'ensemble des extensions nécessaires. Cependant, quelques des paramètres de configuration de base peuvent être légèrement distincts. Afin d'éviter tout souci imprévu, veuillez vérifier que votre configuration PHP correspond aux paramètres conseillés suivants.

PHP a une commande intégrée, phpinfo(), qui montre les détails de la configuration de PHP sur le serveur. La quantité de détails produits par cette commande peut paraître être une énorme surcharge d'informations, mais elle est inestimable afin de déterminer pourquoi quelque chose fonctionne parfaitement sur votre ordinateur local, mais pas sur votre site Web. Généralement, le souci vient du serveur distant qui a désactivé une fonction ou n'a pas installé une extension optionnelle. Les paquets tout-en-un simplifient l'exécution de phpinfo() :

- XAMPP : cliquez sur le lien phpinfo dans le menu à gauche de l'écran de bienvenue de XAMPP ;
- MAMP : cliquez sur phpinfo dans le menu principal en haut de la page de démarrage de MAMP ;
- WampServer : ouvrez le menu WampServer et cliquez sur Localhost. Le lien pour phpinfo() se situe sous Outils.

La page que vous trouverez ressemblera à ce qui suit :



PHP Version 7.3.18

Build Date	May 12 2020 10:48:27
Compiler	MSVC15 (Visual C++ 2017)
Architecture	x64
Configure Command	cscript /nologo /e:jscript configure.js "--enable-snapshot-build" "--enable-debug-pack" "--with-pdo-oci=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--with-oci8-12c=c:\php-snap-build\deps_aux\oracle\x64\instantclient_12_1\sdk,shared" "--enable-object-out-dir=../obj/" "--enable-com-dotnet=shared" "--without-analyzer" "--with-pgo"
Server API	Apache 2.0 Handler
Virtual Directory Support	enabled
Configuration File (php.ini) Path	C:\WINDOWS
Loaded Configuration File	C:\xampp2020\php\php.ini
Scan this dir for additional .ini files	(none)
Additional .ini files parsed	(none)
PHP API	20180731
PHP Extension	20180731
Zend Extension	320180731
Zend Extension Build	API320180731,TS,VC15
PHP Extension Build	API20180731,TS,VC15
Debug Build	no
Thread Safety	enabled
Thread API	Windows Threads
Zend Signal Handling	disabled
Zend Memory Manager	enabled
Zend Multibyte Support	provided by mbstring
IPv6 Support	enabled
DTrace Support	disabled
Registered PHP Streams	php, file, glob, data, http, ftp, zip, compress.zlib, compress.bzip2, https, ftps, phar

Vous devez aussi exécuter phpinfo() sur votre serveur distant afin de vérifier les fonctionnalités activées. Si les extensions énumérées ne sont pas prises en charge, certains codes de ce livre électronique ne fonctionneront pas quand vous téléchargerez vos fichiers sur votre site Web. Si vous trouvez une configuration différente, vous devrez changer le fichier php.ini. Il s'agit d'un fichier vraiment long, ce qui a tendance à perturber les nouveaux venus dans le domaine de la programmation, mais il n'y a pas lieu de s'inquiéter. Il s'agit d'un texte facile et l'une des raisons de sa longueur est qu'il contient plusieurs commentaires expliquant les options distinctes. Cela dit, c'est une excellente idée de faire une copie de sauvegarde avant de changer le php.ini en cas d'erreur. Après l'avoir ouvert, vous trouverez des lignes débutant par un point-virgule (;) qui sont des commentaires. Les lignes que vous devez changer ne débutent pas par un point-virgule. La majorité des directives sont précédées d'un ou de nombreux exemples indiquant comment les définir. Veillez à ne pas changer par erreur l'un des exemples commentés. Pour les directives qui utilisent On ou Off, il suffit de changer la valeur par celle qui est conseillée. Par exemple, si vous devez activer l'affichage des messages d'erreur, changez la ligne ci-dessous :

display_errors = Off

en le changeant en ceci :

display_errors = On

Afin de définir le niveau de signalement des erreurs, nous devons utiliser des constantes PHP, écrites en majuscules et en minuscules. La directive doit ressembler à ce qui suit :

error_reporting = E_ALL

Après avoir changé le fichier php.ini, enregistrez le fichier et redémarrez Apache ou IIS pour que les changements prennent effet. Si le serveur Web ne démarre pas, vous devez vérifier le fichier journal des erreurs du serveur. Nous sommes en mesure le trouver dans les endroits énumérés ci-dessous :

- XAMPP : Dans le panneau de contrôle XAMPP, cliquez sur le bouton Logs à côté d'Apache, et ensuite sélectionnez Apache error.log.
- MAMP : sous Applications : MAMP : Logs, double-cliquez sur apache_error.log afin de l'ouvrir dans la Console.
- WampServer : dans le menu WampServer, sélectionnez Apache puis Apache Error Log.
- EasyPHP : cliquez avec le bouton droit de la souris sur l'icône EasyPHP dans la barre des tâches et sélectionnez Log Files et ensuite Apache ;
- IIS : le chemin par défaut des fichiers journaux est C:\inetpub\logs.

L'entrée la plus nouvelle du journal des erreurs devrait donner une indication de ce qui a empêché le serveur de redémarrer. Utilisez ces informations afin de corriger les modifications apportées au fichier php.ini. Si le souci persiste, utilisez le fichier de sauvegarde que vous avez enregistré avant les modifications. Conseillez et vérifiez soigneusement les changements que vous apportez au fichier.

3

CODE PHP

À première vue, le code PHP peut sembler assez difficile, mais une fois que vous aurez compris les bases, vous constaterez que sa structure est remarquablement facile. Si vous avez déjà travaillé avec un autre langage de programmation, comme JavaScript ou jQuery, vous constaterez qu'ils ont plusieurs points communs.

Chaque page PHP doit avoir les éléments ci-dessous :

- L'extension de fichier correcte, généralement . *php :*
- Balises PHP d'ouverture et de fermeture entourant chaque bloc de code PHP.

Une page PHP typique utilise tout ou partie des éléments ci-dessous :

- Des déclarations conditionnelles pour la prise de décision ;
- Fonctions ou objets afin d'effectuer des tâches prédéfinies ;
- Boucle afin d'effectuer des tâches de manière répétée ;
- Les matrices peuvent contenir de nombreuses valeurs ;
- Variables servant de substituts à des valeurs inconnues ou changeantes.

Analysons rapidement chacun de ces éléments, en débutant par le nom du fichier et les balises d'ouverture et de fermeture.

PHP est un langage côté serveur, cela signifie que le serveur Web traite votre code PHP et n'envoie que les résultats, en général sous forme de HTML, au navigateur. Comme l'ensemble des actions se fait sur le serveur, vous devez indiquer que vos pages contiennent du code PHP. Cela implique deux étapes faciles, à noter :

- Attribue à chaque page une extension de nom de fichier PHP ; la valeur par défaut est . *php* ;
- Placez tout le code PHP dans des balises PHP.

La balise d'ouverture est < ? php et la balise de fermeture est ?>. Si vous placez les balises sur la même ligne que le code environnant, il n'est pas obligé qu'il y ait un espace avant la balise d'ouverture ou bien après la balise de fermeture, cependant il doit y avoir un espace après le php dans la balise d'ouverture comme ceci :

** C'est du HTML avec du code PHP intégré** < ? php // PHP code**? >.**

Quand vous insérez plus d'une ligne de PHP, il est excellent de placer les balises d'ouverture et de fermeture sur des lignes séparées pour plus de clarté. Analysez l'exemple ci-dessous

```
< ? php
// ligne de code PHP
// une autre ligne de code PHP
?>
```

Il se peut que vous rencontriez < ? comme version courte de la balise d'ouverture, mais cette balise ne fonctionne pas sur l'ensemble des serveurs. Utilisez < ? php pour qu'il n'y ait pas de soucis de compatibilité entre les serveurs.

PHP est un langage intégré aux pages Web, ce qui veut dire que vous pouvez insérer des blocs de code PHP dans des pages Web normales. Quand une personne visite votre site et demande une page PHP, le serveur l'envoie au moteur PHP, qui lit la page de haut en bas à la recherche de balises PHP. Le HTML n'est pas changé, mais chaque fois que le moteur PHP rencontre une balise < ? php, il débute à traiter le code et continue jusqu'à ce qu'il atteigne la balise de fermeture ?>. Si le code PHP produit quelque chose dans la sortie, il sera inséré à cet endroit de la page.

Variables

PHP utilise aussi des variables, ce qui veut dire un nom que vous donnez à quelque chose qui peut modifier ou que vous ne connaissez pas à l'avance. Les variables en PHP débutent toujours par $ (un signe de dollar). Bien que le concept de variable semble abstrait, nous employons des variables tout le temps dans la vie de tous les jours. Quand vous rencontrez quelqu'un pour la première fois, l'une des premières choses que vous posez est "Quel est votre nom ?" Peu importe que la personne que vous venez de rencontrer se nomme Antonio, Filippo ou bien Gerry, le mot "nom" reste constant. De même, avec votre compte bancaire, l'argent entre et sort tout le temps, mais le montant disponible est toujours indiqué comme le solde. Donc, "nom" et "solde" sont des variables quotidiennes. Mettez un signe de dollar devant eux et vous avez deux variables PHP prêtes, comme l'exemple ci-dessous :

Nom

Chaleur

Vous pouvez choisir ce que vous souhaitez comme nom de variable, à condition de respecter les règles ci-dessous :

- Les variables débutent toujours par un signe de dollar ($) ;

- Le premier caractère après le signe dollar ne peut pas être un chiffre ;
- Aucun espace ni signe de ponctuation n'est autorisé, à l'exception du caractère de soulignement (_).
- Les noms de variables sont sensibles à la casse : $start et $start ne sont pas identiques.

Quand vous choisissez des noms pour des variables, il est logique de choisir une chose qui vous affiche à quoi elles servent. Les variables que vous avez vues jusqu'à maintenant sont de excellents exemples. Étant donné que vous ne pouvez pas utiliser d'espaces dans les noms de variables, il est judicieux de mettre en majuscules la première lettre du deuxième mot ou des mots suivants quand vous les combinez (ce que nous nommons la notation camelCase ou camel). Vous pouvez aussi utiliser un trait de soulignement ($initial_notation) nommé notation snake ou snake_case.

Ne tentez pas de gagner du temps en utilisant des variables vraiment courtes, car l'utilisation de $s, $p, $n et $b rend le code plus difficile à saisir et donc à écrire. Plus essentiel encore, cela rend les erreurs plus difficiles à détecter. Comme toujours, il y a des exceptions à une règle. Par convention, $i, $j et $k sont fréquemment utilisés afin de garder la trace du nombre de fois qu'une boucle a été exécutée et $ e est utilisé dans la vérification des erreurs.

Les variables obtiennent leurs valeurs de diverses sources, notamment les valeurs énumérées ci-dessous :

- Le résultat d'un calcul ;
- Inclusion directe dans le code PHP ;
- Saisie par l'utilisateur via des formulaires en ligne ;
- Une base de données ;

- Une source externe, comme un flux de nouvelles ou un fichier XML.

Quelle que soit l'origine de la valeur, elle est toujours associée à un signe égal (=), comme ce qui suit :

$variable = valeur ;

La variable se situe à gauche du signe égal et la valeur à droite. Comme il attribue une valeur, le signe égal est nommé opérateur d'affectation.

Le PHP est écrit comme une série de commandes ou d'instructions. Chaque instruction affiche généralement au moteur PHP d'effectuer une certaine action, et doit toujours être suivie d'un point-virgule, comme ce qui suit :

< ? php

Faites-le ;

maintenant fais ça ;

?>

Comme pour l'ensemble des règles, il existe une exception : vous pouvez omettre le point-virgule s'il n'y a qu'une seule instruction dans le bloc de code. Toutefois, je vous le déconseille, car, contrairement à JavaScript, PHP ne suppose pas automatiquement qu'il doit y avoir un point-virgule à la fin d'une ligne. Cela a un effet secondaire intriguant : vous pouvez répartir de longues instructions sur de nombreuses lignes et organiser votre code afin d'en simplifier la lecture. PHP, comme HTML, ignore les espaces dans le code. Au lieu de cela, il s'appuie sur les points-virgules afin de savoir où se termine une commande et où débute la suivante.

Tableaux

Comme d'autres langages de calcul, PHP permet de stocker de nombreuses valeurs dans un type spécial de variable nommé tableau. Imaginez un tableau comme une liste de courses. Bien que chaque élément puisse être distinct, vous pouvez vous y référer collectivement par un seul nom. Imaginez une variable $liste d'achats qui se réfère collectivement à cinq articles : vin, poisson, pain, raisin et fromage.

Les éléments individuels, ou éléments de tableau, sont changés par un nombre entre crochets qui suit tout de suite le nom de la variable. PHP attribue le numéro automatiquement, mais il est essentiel de noter que la numérotation débute toujours par 0. Donc, le premier élément du tableau, vin dans notre exemple, est désigné par $listExpenditure[0], et non par $listExpenditure[1] comme on pourrait s'y attendre. Pour la même raison, bien qu'il y ait cinq éléments, le dernier (fromage) est trouvé avec $listExpenditure[4]. Le nombre est donné comme la clé ou l'index du tableau et ce type de tableau est nommé un **tableau indexé**.

PHP emploie un autre type de tableau dans lequel la clé est un mot (ou toute combinaison de lettres et de chiffres). Par exemple, un tableau peut ressembler à ce qui suit :

```
$article['title']='Éponge de cuisine' ;
$article['producer']='Company Goofy' ;
$article['quantity'] = '100' ;
$article['id'] = '9238-1-442-036-2' ;
```

Ce type de tableau est appelé **tableau associatif**. Il est important de noter que la clé du tableau est entourée de guillemets (ils peuvent être simples ou doubles, cela n'a pas d'importance) et ne doit pas contenir d'espace ou bien de ponctuation.

Les tableaux sont une partie essentielle et utile de PHP, ils sont même largement utilisés. De plus, les tableaux sont largement utilisés dans les bases de données afin de récupérer les résultats de recherche stockés dans une série de tableaux.

PHP contient de nombreux tableaux intégrés qui sont automatiquement remplis d'informations utiles. Nous les appelons des **tableaux superglobaux** et ils débutent tous par un signe dollar suivi d'un trait de soulignement. Les deux que vous observerez souvent sont $_POST et $_GET. Ils possèdent des informations transmises de formulaires en postes via le protocole de transfert hypertexte (HTTP). Les tableaux superglobaux sont tous des tableaux associatifs et les clés de $_POST et $_GET sont automatiquement dérivées des noms des éléments de formulaire ou des variables dans une chaîne de requête à la fin d'une URL. Imaginons que vous détenez un champ de saisie de texte nommé "adresse" dans un formulaire ; PHP crée automatiquement un élément de tableau nommé $_POST['adresse'] quand le formulaire est soumis par la méthode post ou $_GET['adresse'] si vous employez la méthode GET. $_POST['adresse'] possède la valeur saisie par un visiteur dans le champ de texte, ce qui vous aide à l'afficher à l'écran, de l'entrer dans une base de données, de l'envoyer dans la boîte de réception, ou de faire ce que vous souhaitez.

Valeurs spéciales

Bien que le texte doive être mis entre guillemets, il existe trois valeurs spéciales - true, false et null - qui ne doivent jamais être mises entre guillemets, sauf si vous voulez les traiter comme du texte (ou des chaînes). Les deux premiers signifient ce que vous attendez ; le dernier, null, signifie "rien" ou "aucune valeur". Ces valeurs sont vraiment utiles à PHP afin de prendre des décisions, par exemple, si la valeur d'une variable est vraie, exécuter une commande.

Jetez un coup d'œil au code ci-dessous :

$variable = false ;

Elle fait précisément ce que vous attendez : elle rend $variable fausse. Maintenant, regardez ça :

$variable = 'false' ;

Cela fait précisément le contraire de ce à quoi on pourrait s'attendre : cela rend $variable vrai ! Pourquoi ? Parce que les guillemets autour de false le transforment en chaîne de caractères et PHP traite les chaînes de caractères comme des vrais. L'autre chose à remarquer en ce qui concerne ces valeurs est qu'elles ne sont pas sensibles à la casse. Les exemples énumérés ci-dessous sont tous valables :

$variable = VRAI ;

$variable = tRuE ;

$variable = true ;

Instructions conditionnelles

La vie est remplie de décisions, et le PHP aussi. Si vous avez l'expérience d'autres langages de programmation, vous connaissez déjà une énorme partie de ce sous-chapitre. En effet, le processus de décision en PHP utilise des instructions conditionnelles et la plus fréquente utilise if, qui est vraiment proche du langage courant. Dans la vie réelle, vous pouvez être confronté à la décision ci-dessous : s'il fait chaud, je vais à la plage.

En pseudo-code PHP, la même décision est similaire à ce qui est ci-dessous :

if (it is hot) {

Je vais à la plage ;

}

La condition à tester est placée entre parenthèses rondes alors que l'action est placée entre crochets. Il s'agit du modèle de décision de base :

if (condition is true) {

// code à exécuter si la condition est vraie

}

Le code à l'intérieur des accolades n'est exécuté que si la condition est vraie. Si elle est fausse, PHP ignore tout ce qui se situe à l'intérieur des accolades et passe à la section suivante du code. Dans le code précédent, nous avons aussi inséré un commentaire, débutant par les symboles //.

Quelquefois, l'instruction if est tout ce dont vous avez besoin, mais fréquemment, vous voulez qu'une action soit déclenchée si la condition n'est pas remplie. Afin de ce faire, utilisez if comme indiqué ci-dessous :

```
if (condition is true) {
// code à exécuter si la condition est vraie
} else {
// code à exécuter si la condition est fausse
}
```

Si vous souhaitez plus d'alternatives, vous pouvez ajouter d'autres déclarations conditionnelles comme celle illustrée ci-dessous :

```
if (condition is true) {
// code à exécuter si la première condition est vraie
} elseif (la deuxième condition est vraie) {
// code à exécuter si la première condition échoue, mais que la seconde est vraie.
} else {
// code si les deux conditions sont fausses
}
```

Il est important de noter que les instructions conditionnelles ne sont utiles que pour vérifier que la condition testée est vraie ; effectivement, si elle n'est pas vraie, elle doit être fausse. Les conditions dépendent fréquemment de la comparaison de deux valeurs. Cette valeur est-elle supérieure à celle-là ? Sont-ils égaux ? Et ainsi de suite.

Afin de vérifier l'égalité, PHP emploie deux signes égaux (==), comme ce qui suit :

```
if ($profile == 'administrator') {
```

// envoyer à la page d'administration
} else {
// envoyer à la page de l'utilisateur
}

De même, vous pouvez employer les symboles supérieur à (>), inférieur à (<), supérieur ou égal à (>=) ou inférieur à (=<), autre que (!=) afin de comparer des valeurs.

Souvent, la comparaison de deux valeurs ne suffit pas. PHP vous aide à définir une série de conditions en employant des opérateurs logiques afin de spécifier si toutes ou seulement certaines doivent être satisfaites.

Les opérateurs logiques les plus essentiels en PHP sont présentés dans le tableau ci-dessous :

&&
Et
$a && $b
Est égal à vrai si $a et $b sont tous deux vrais.
||
Ou
$a || $b
Est égal à true si $a ou $b sont true ; sinon, false
!
Non
!$a
Est égal à vrai si $a n'est pas vrai

Afficher les résultats

Bien sûr, vous pouvez présenter les résultats dans votre page Web et il y a deux manières de le faire en PHP : utiliser echo ou print. Il existe plusieurs différences subtiles entre les deux, mais vous pouvez les considérer comme

équivalentes. Vous pouvez employer echo avec des variables, des nombres et des chaînes de caractères ; il suffit de le placer devant ce que vous souhaitez afficher, comme ce qui suit :

```
$name = 'Philip' ;
echo $name ; // afficher Philippe
echo 5 ; // afficher 5
echo 'David' ; // afficher David
```

Quand vous utilisez echo et print avec une variable, ils ne fonctionnent qu'avec les variables qui possèdent une seule valeur. Vous ne pouvez pas les utiliser afin d'afficher le contenu d'un tableau ou une extraction d'une base de données. Afin de ce faire, des boucles sont nécessaires : vous utilisez echo ou print dans la boucle afin d'afficher chaque élément individuellement. Nous observerons certains exemples de ce type dans les prochains chapitres.

Vous pouvez observer des scripts qui utilisent des parenthèses avec echo et print, comme ce qui suit :

```
echo ('David') ; // afficher David
```

Les parenthèses ne font aucune différence et, à moins que vous n'aimiez les écrire pour le plaisir, ne les utilisez pas.

Quand vous voulez afficher la valeur d'une seule variable ou expression, vous pouvez utiliser une balise PHP abrégée de la façon suivante :

```
<p>Mon nom est < ? = $name ; ?>. </p>
```

Cela produit précisément le même résultat que celui-ci :

```
<p>Mon nom est < ? php echo $name ; ?>. </p>
```

Comme il s’agit d’un raccourci, aucun autre code ne peut se situer dans le même bloc PHP, mais il est, en fait, particulièrement utile afin d’intégrer des résultats de bases de données dans une page Web. Il est évident que la valeur de la variable doit être définie dans un bloc PHP précédent avant que ce type d’affichage puisse être utilisé.

4

TYPES DE DONNÉES

PHP est ce que nous appelons un langage **faiblement typé,** en pratique, contrairement à d'autres langages de programmation (par exemple Java ou C #), il faut noter que PHP ne se soucie pas du type de données que vous stockez dans une variable.

La majorité du temps, c'est énormément pratique, même s'il faut faire attention à la saisie de l'utilisateur. Vous pouvez vous attendre à ce qu'un utilisateur entre un nombre dans un champ de formulaire, mais PHP ne retournera pas d'erreur si l'utilisateur entre un mot. Voilà la raison pourquoi il est important de vérifier soigneusement la saisie de l'utilisateur. Bien que PHP soit faiblement typé, il utilise les huit types de données ci-dessous :

- **Entier** : un nombre entier, par exemple 1, 25, 42 ou 2006. Les nombres entiers ne doivent pas contenir de virgule ou de ponctuation comme séparateurs. Vous pouvez aussi utiliser des nombres hexadécimaux, qui doivent être précédés de 0x (par exemple, 0xFFFF, 0x0000).
- **Flottant** : un nombre qui possède un point décimal, comme 9,99, 98,6 ou 2,1. Remarquez que PHP ne supporte pas l'utilisation de la virgule comme point décimal, ce qui est courant dans plusieurs pays européens, vous devez alors utiliser

un point. Comme pour les nombres entiers, les nombres à virgule flottante n'ont pas besoin de séparateurs pour les milliers.

- Chaîne : une chaîne est un texte de n'importe quelle longueur. En effet, il peut être court, avec zéro caractère (une chaîne vide) et n'a pas de limite supérieure.
- **Booléen** : ce type n'a que les deux valeurs suivantes : vrai ou faux.
- **Tableau** : un tableau est une variable qui peut stocker de nombreuses valeurs, mais qui peut aussi ne contenir aucune valeur (un tableau vide). Les tableaux peuvent contenir tout type de données, y compris d'autres tableaux. Un tableau de tableaux est nommé un tableau multidimensionnel.
- Objet : un objet est un type de données complexe qui est en mesure de stocker et de manipuler des valeurs.
- **Ressource** : Quand PHP se connecte à une source de données externe, comme un fichier ou une base de données, il stocke une référence à celle-ci en tant que ressource.
- **NULL** : Il s'agit d'un type de données spécial mentionnant qu'une variable n'a pas de valeur.

Un effet secondaire crucial du typage faible de PHP est que si vous enfermez un entier ou un nombre à virgule flottante entre guillemets, PHP le convertit automatiquement d'une chaîne de caractères en un nombre, vous permettant d'effectuer des calculs sans avoir besoin d'une manipulation particulière. À cet égard, il est distinct de JavaScript et peut avoir des conséquences inattendues. Quand PHP voit le signe plus (+), il suppose que vous souhaitez effectuer une addition, et tente donc de convertir les chaînes de caractères en nombres entiers ou flottants, comme dans l'exemple illustré ci-dessous :

```
$fruits = '2 pommes' ;
$vegetables = '2 carottes' ;
echo $fruits + $vegetables ; // afficher 4
```

PHP analyse que $fruits et $vegetables débutent tous deux par un chiffre, il extrait alors le chiffre et ignore le reste. Toutefois, si la chaîne ne débute pas par un chiffre, PHP la convertit en 0, comme l'illustre l'exemple ci-dessous :

```
$fruits = '2 pommes' ;
$vegetables = 'et 2 carottes' ;
echo $fruits + $vegetables ; // afficher 2
```

La typographie faible est un travers et un plaisir, mais elle rend PHP vraiment simple pour les débutants. Souvenez-vous qu'il est souvent nécessaire de vérifier qu'une variable contient le bon type de données avant de l'utiliser.

5

CYCLES

Pendant que et faire... pendant que

Une boucle est une section de code qui est répétée jusqu'à ce qu'une certaine condition soit remplie. Les boucles sont fréquemment contrôlées en définissant une variable qui compte le nombre d'itérations. En augmentant la variable d'une unité toutes les fois, la boucle s'arrête quand la variable atteint un nombre prédéfini. Les boucles sont aussi contrôlées en exécutant chaque élément d'un tableau. Quand il n'y a plus d'éléments à traiter, la boucle s'arrête. Les boucles possèdent couramment des instructions conditionnelles. Donc, bien que leur structure soit extrêmement facile, elles peuvent être utilisées afin de créer un code qui traite les données de façon complexe.

Le type de boucle le plus facile est nommé boucle while. Sa structure de base ressemble à ce qui suit :

```
while (condition is true) {
faire quelque chose ;
}
```

Le code ci-dessous présente l'ensemble des nombres entre 1 et 100 dans un navigateur. Elle débute par donner la valeur 1 à une variable ($i), et par la suite, utilise cette variable comme un compteur pour contrôler la boucle, en plus d'afficher le nombre actuel à l'écran.

```
$i = 1 ; // mettre le compteur à 1
while ($i <= 100) {
echo '$i < br>' ;
$i++ ; // augmente le compteur de 1
}
```

Une variante de la boucle while utilise le mot clé do et suit le schéma de base affiché ci-dessous :

```
faire {
code à exécuter
} while (condition à tester) ;
```

Il faut noter que la différence entre une boucle do...while et une boucle while est que le code contenu dans le bloc do est exécuté au moins une fois, même si la condition n'est jamais vraie. Le code ci-dessous présente la valeur de $i une fois, même si elle est supérieure au maximum attendu.

```
$i = 1000 ;
faire {
echo '$i < br>' ;
$i++ ; // augmente le compteur de 1
} while ($i <= 100) ;
```

Le danger avec do...while est d'oublier de définir une condition qui mène à la fin de la boucle ou de définir une condition impossible. C'est ce que nous appelons une boucle sans fin qui bloque l'ordinateur ou bien fait planter le navigateur.

Pour et foreach

Il est important de savoir que la boucle for est moins susceptible de générer une boucle infinie parce qu'il est nécessaire de déclarer l'ensemble des conditions de la boucle dans la première ligne. La boucle for utilise le modèle de base illustré ci-dessous :

```
for (initialisation ; condition ; code à exécuter après chaque itération) {
code à exécuter
}
```

Le code ci-dessous fait précisément la même chose que la boucle while précédente, en présentant tous les chiffres de 1 à 100 :

```
for ($i = 1 ; $i <= 100 ; $i ++) {
echo '$i < br>' ;
}
```

Les trois expressions entre parenthèses contrôlent l'action de la boucle (remarquez qu'elles sont séparées par des points-virgules et non des virgules) :

1. La première expression est exécutée avant le début de la boucle. En ce qui concerne ce cas, il fixe la valeur initiale de la variable compteur $i à 1.
2. La deuxième expression définit la condition qui détermine la durée d'exécution de la boucle. En fait, il peut s'agir d'un nombre fixe, d'une variable ou d'une expression qui calcule une valeur.
3. La troisième expression est exécutée à la fin de chaque itération de la boucle. Alors, dans ce cas, il augmente $i de 1, mais notez que rien ne vous empêche d'augmenter de 4, 10 ou 50. Par exemple, si vous remplacez $i++ par $i+=10 dans cet exemple, vous obtiendrez 1, 11, 21, 31 et ainsi de suite.

De plus, le dernier type de boucle en PHP est utilisé avec les tableaux et les objets. Il s'affiche sous deux formes distinctes, qui utilisent toutes deux des variables temporaires afin de traiter chaque élément. Si vous n'avez besoin d'utiliser que la valeur de l'élément, la boucle foreach prend la forme ci-dessous :

```
foreach (variableName as element) {
faire quelque chose avec le
}
```

L’exemple ci-dessous crée une boucle sur le tableau $listExpenditure et présente le nom de chaque élément :

```
$listExpenditure = ['vin', 'poisson', 'pain', 'raisin', 'fromage'] ;
foreach ($ listExpenditure as $item) {
echo $item. '<br>' ;
}
```

Bien que l’exemple ci-dessus emploie un tableau indexé, il est aussi possible d’utiliser la forme facile de la boucle foreach avec un tableau associatif. La forme alternative de la boucle foreach permet d’accéder à la fois à la clé et à la valeur de chaque élément. Elle prend cette forme légèrement distincte :

```
foreach (variableName as key => value) {
faire quelque chose avec la clé et la valeur }
```

Afin de mettre fin prématurément à une boucle quand une certaine condition est remplie, insérez le mot clé break dans une instruction conditionnelle. Dès que le script est interrompu, la boucle est quittée.

Afin d’ignorer l’évaluation du code dans une boucle quand une certaine condition est remplie, utilisez le mot-clé continue. Plutôt que de sortir de la boucle, il retourne au début de la boucle et traite l’élément ci-dessous. Par exemple, la boucle ci-dessous saute l’élément actuel si $photo n’a pas de valeur :

```
foreach ($image as $photo) {
if (empty ($photo)) continue ;
// code pour afficher une photo
}
```

6

FONCTIONS

Les fonctions proposent un moyen pratique d'effectuer des tâches souvent réalisées. En plus du grand nombre de fonctions intégrées, PHP vous donne la chance de créer vos propres fonctions. Notez que le principal avantage est que vous ne devez écrire le code qu'une seule fois, plutôt que de devoir le réécrire chaque fois que vous en avez besoin. Cela aide non seulement d'accélérer le développement, mais aussi de faciliter la lecture et la maintenance du code. S'il y a un souci avec le code de votre fonction, vous pouvez le mettre à jour à un seul endroit au lieu de chercher partout. En outre, les fonctions accélèrent en général le traitement de vos pages.

Il est simple de créer ses propres fonctions en PHP. Il suffit d'enfermer un bloc de code dans une paire de crochets et d'utiliser le mot-clé function afin d'appeler la nouvelle fonction. Le nom de la fonction est toujours suivi d'une paire de parenthèses. L'exemple ci-dessous démontre la structure de base d'une fonction personnalisée :

```
function greeting() {
echo 'Hello!' ;
}
```

Vous pouvez invoquer cette fonction en tapant tout simplement hello() ; dans un bloc de code PHP et vous observerez à l'écran Hello ! Ce type de

fonction effectue toujours précisément la même opération. Afin que les fonctions puissent réagir aux circonstances, des valeurs doivent leur être transmises comme arguments.

Imaginons que vous souhaitiez adapter la fonction pour qu'elle présente le nom d'une personne. Vous pouvez le faire en insérant une variable entre parenthèses dans la déclaration de la fonction. La même variable est par la suite utilisée dans la fonction pour afficher la valeur transmise à la fonction. Afin de transmettre plus d'un argument à une fonction, séparez les variables par des virgules entre des crochets ouverts. Voici ci-dessous à quoi ressemble la fonction maintenant :

```
function greet($name) {
echo "Bonjour $name !";
}
```

Donc, vous êtes maintenant en mesure d'utiliser cette fonction dans une page pour présenter la valeur de toute variable passée à greet(). Par exemple, si vous avez un formulaire en ligne qui enregistre le nom d'une personne dans une variable nommée $visitor et qu'Antonio visite votre site, vous pouvez le saluer de façon personnalisée en invoquant la fonction greet($visitor) ; dans votre page.

Il est aussi essentiel de comprendre que les fonctions créent un environnement séparé qui est comme une boîte noire. Généralement, ce qui se passe à l'intérieur de la fonction n'a aucun impact sur le reste du script, sauf si elle renvoie une valeur. Les variables d'une fonction restent exclusives à la fonction. L'exemple devrait démontrer ce que je veux expliquer :

```
function duplicate ($number) {
$number *= 2 ;
echo '$number is '. $number. <br>' ;
}
$number = 4 ;
```

```
dupliquer($numéro) ;
echo '$number is '. $number ;
```

Les quatre premières lignes définissent une fonction nommée duplicate(), qui accepte un nombre, le double et le présente à l'écran. Le reste du script attribue la valeur 4 à $number. Il passe par la suite $number comme argument à duplicate(). La fonction traite $number et présente 8. À la fin de la fonction, $number est présenté à l'écran par la fonction echo. Cette fois, la valeur sera 4 et non 8 parce que la valeur n'a pas été renvoyée par la fonction, mais seulement utilisée pour l'affichage à l'écran.

Cela illustre que $number dans le script principal n'a aucun rapport avec la variable portant le même nom dans la fonction. C'est ce que nous appelons la portée ou l'étendue de la variable. Même si la valeur de la variable modifie à l'intérieur d'une fonction, les variables portant le même nom à l'extérieur ne sont pas affectées. Afin d'éviter toute confusion, il est excellent d'utiliser dans le reste du script des noms de variables distincts de ceux utilisés dans les fonctions. Ce n'est pas toujours possible, il est alors utile de savoir que les fonctions sont de petites boîtes noires et n'ont généralement aucun impact direct sur les valeurs des variables dans le reste du script. Un autre aspect fondamental est qu'une fonction ne peut généralement pas accéder aux valeurs du script externe, à moins qu'elles ne soient transmises à la fonction en tant qu'arguments.

Il existe plus d'une manière pour une fonction de changer la valeur d'une variable donnée en entrée, mais la méthode la plus essentielle consiste à utiliser le mot clé return et à affecter le résultat à la même variable ou à une autre. Nous sommes en mesure le démontrer en modifiant la fonction duplicate() de la façon illustrée ci-dessous :

```
function duplicate ($number) {
retourner $number *= 2 ;
}
$num = 4 ;
```

```
$double = duplicate($num) ;
echo '$num est : '. $num. <br>' ;
echo '$raddoppiato est : '. $raddoppiato ;
```

Cette fois, j'ai utilisé des noms distincts pour les variables afin de ne pas les mélanger. J'ai aussi assigné le résultat de duplicate($num) à une nouvelle variable. L'avantage de cette méthode est que nous disposons à la fois de la valeur originale et du résultat du calcul. Il n'est pas toujours nécessaire de conserver la valeur d'origine, mais cela peut quelquefois s'avérer vraiment utile.

Bien que les fonctions ne changent généralement pas la valeur des variables passées en argument, il arrive que nous voulions changer la valeur originale au lieu d'avoir une valeur de retour. Afin de ce faire, lors de la définition de la fonction, faites précéder le paramètre que vous voulez changer d'un e (&), comme ceci :

```
function duplicate(&$number) {
$number *= 2 ;
}
```

Remarquez que cette version de la fonction duplicate() n'imprime pas la valeur de la variable à l'écran et ne renvoie pas la valeur du calcul. Comme le paramètre entre parenthèses est précédé d'un e commercial (&), la valeur originale d'une variable passée en argument à la fonction sera changée. C'est ce qu'on appelle le passage par référence.

Le code illustré ci-dessous en affiche l'effet :

```
$num = 4 ;
echo '$num est : ' . $num . '< br>' ;
dupliquer($num) ;
echo '$num est maintenant : ' . $num ;
```

Si votre fonction se situe sur la même page que celle où elle est utilisée, l'endroit où vous la déclarez n'a pas d'importance ; il peut se situer avant ou après son utilisation. Il est cependant logique de déclarer les fonctions

ensemble, en haut ou en bas d'une page. Cela les rend plus simples à repérer et à entretenir. Il est mieux de stocker les fonctions utilisées dans plus d'une page dans un fichier externe inclus dans chaque page. L'inclusion de fichiers externes se fait via include et request. Quand les fonctions sont enregistrées dans des fichiers externes, il est requis d'inclure le fichier externe avant d'employer une fonction.

Exiger et inclure

La possibilité d'inclure le contenu d'un fichier dans un autre est l'une des fonctionnalités les plus intéressantes de PHP et l'une des plus simples à mettre en œuvre. La majorité des pages d'un site Web partagent des éléments communs, comme les en-têtes, les pieds de page et les menus de navigation. Vous pouvez changer l'apparence de ces éléments sur l'ensemble du site en changeant les règles de style dans une feuille de style externe. Mais les CSS n'ont qu'une capacité limitée à modifier le contenu des éléments de la page. Si vous voulez ajouter un nouvel élément à votre menu, vous devez changer le code HTML de chaque page qui le présente.

Il est important de savoir que PHP prend en charge les inclusions côté serveur (SSI), ce qui signifie un fichier externe contenant du code dynamique ou du HTML (ou les deux) que vous voulez intégrer dans plusieurs pages. PHP intègre le contenu dans chaque page Web sur le serveur, et comme chaque page utilise le même fichier externe, vous pouvez mettre à jour un menu ou un autre élément commun en changeant et en téléchargeant un seul fichier, ce qui permet de gagner du temps considérablement.

La possibilité d'inclure du code provenant de divers fichiers est un élément crucial de PHP. Il suffit d'employer l'une des commandes d'inclusion de PHP et d'indiquer au serveur où trouver le fichier.

De plus, PHP dispose de quatre commandes qui peuvent être employées afin d'inclure du code provenant d'un fichier externe. Les commandes sont énumérées ci-dessous :

- comprend
- inclure_once
- exiger
- exiger_once

Ils font pratiquement tous la même chose, donc pourquoi en avoir quatre à disposition ? La distinction essentielle est que include essaie de poursuivre le traitement d'un script, même si le fichier externe est manquant, tandis que require est employé dans le sens inverse : si le fichier est manquant, le moteur PHP arrête le traitement et génère une erreur. En d'autres mots, cela veut dire que vous devez utiliser include si votre page reste utilisable sans le contenu du fichier externe. Utilisez require si votre page dépend d'un fichier externe.

De plus, les deux autres commandes, include_once et require_once, fonctionnent de la même façon, mais empêchent le même fichier d'être inclus de nombreuses fois dans une page, ce qui est particulièrement important quand nous incluons des fichiers qui définissent des fonctions ou des classes. Essayer de définir une fonction ou une classe plus d'une fois dans un script provoque une erreur irrécupérable. Par conséquent, l'utilisation de include_once ou request_once garantit que les fonctions et les classes ne sont définies qu'une seule fois, même si le script essaie d'inclure le fichier externe de nombreuses fois.

Afin d'inclure un fichier externe, vous devez utiliser l'une des quatre commandes suivies du chemin d'accès au fichier sous forme de chaîne : en d'autres mots, le chemin d'accès au fichier doit être entre guillemets (simple ou double, peu importe). Le chemin du fichier peut être absolu ou

relatif au document actuel. Par exemple, cela fonctionnera de l'une des façons illustrées ci-dessous (pour autant que le fichier de destination existe) :

inclut 'includes/menu.php' ;

inclut 'C:/xampp/htdocs/phpsols/includes/menu.php' ;

inclut '/Applications/MAMP/htdocs/phpsols/includes/menu.php' ;

Quand vous utilisez un chemin d'accès relatif, il est conseillé d'utiliser ./ afin d'indiquer que le chemin d'accès débute dans le dossier actuel. Il est alors plus efficace de réécrire le premier exemple de cette façon :

inclut './includes/menu.php' ;

Ce qui ne fonctionne pas, c'est l'utilisation d'un chemin de fichier relatif à la racine du site, comme ce qui suit :

inclut '/includes/menu.php' ;

Voici la façon d'inclure un pied de page commun à plusieurs pages :

< ? php **include** './includes/footer.php' ; ?>

Quand le moteur PHP détecte une commande d'inclusion, il arrête le traitement PHP au début du fichier externe et le reprend à la fin. Il est important de savoir que si le fichier externe doit utiliser du code PHP, ce code doit être entouré de balises PHP. Comme le fichier externe est traité comme une partie du fichier PHP qui l'inclut, un fichier inclus peut contenir n'importe quelle extension.

Quelques développeurs utilisent . inc comme extension de nom de fichier afin d'indiquer clairement que le fichier doit être inclus dans un autre fichier. Cependant, la majorité des serveurs traitent les fichiers . inc comme du texte brut. Cela présente un risque pour la sécurité si le fichier possède des informations confidentielles, comme le nom d'utilisateur et le mot de passe de la base de données. Si le fichier est stocké dans le dossier racine de votre site Web, toute personne qui trouve le nom du fichier peut

tout simplement taper l'URL dans la barre d'adresse du navigateur et ce dernier présentera l'ensemble des données sensibles !

Cependant, tout fichier portant l'extension . php est automatiquement envoyé au moteur PHP pour être analysé avant d'être envoyé au navigateur. Tant que vos informations sensibles se situent dans un bloc de code PHP et dans un fichier portant l'extension . php, elles ne seront pas exposées. Voilà la raison pourquoi quelques développeurs utilisent . inc.php comme include pour avoir une double extension. La partie . inc vous rappelle qu'il s'agit d'un fichier d'inclusion, mais les serveurs ne sont intéressés que par le . php à la fin, qui garantit que tout le code PHP est analysé correctement.

Pendant longtemps, j'ai suivi la convention consistant à utiliser . inc.php concernant les fichiers d'inclusion. Toutefois comme je stocke l'ensemble de mes fichiers include dans un dossier séparé nommé include, j'ai choisi que la double extension était superflue et je n'utilise plus que . php. La convention que vous choisissez vous appartient, mais l'utilisation de . inc est la moins sûre.

Pour s'assurer qu'une page existe, vous pouvez vérifier via les fonctions que PHP offre nativement. Il est important de savoir que si une page n'a aucun sens sans le fichier include, vous devez rediriger l'utilisateur vers une page d'erreur si le fichier include est absent.

Une manière de le faire est de générer une exception, comme le montre l'exemple ci-dessous :

```
< ? php
$file = './includes/menu.php' ;
if (file_exists($file) && is_readable($file)) {
inclut $file ;
} else {
lancer une nouvelle exception ("$file not found") ;
}
?>
```

Une caractéristique utile des fichiers include de PHP est qu'ils peuvent être placés n'importe où, tant que la page avec la commande include sait où les repérer. Les fichiers à inclure ne doivent même pas se trouver à la racine de votre serveur Web, ce qui veut dire que vous êtes en mesure de protéger des fichiers contenant des informations confidentielles, comme des mots de passe, dans un répertoire (dossier) privé auquel un navigateur ne peut accéder. Alors, notez que si votre société d'hébergement donne une zone de stockage en dehors de la racine de votre serveur, vous devriez considérer d'y placer vos fichiers afin de les inclure.

Mais pourquoi n'est-il pas possible d'insérer une référence à la racine du site ? Quand vous cliquez sur un lien afin d'aller sur une autre page, le chemin d'accès dans la balise <a> affiche au navigateur comment passer de la page actuelle à la suivante. La majorité des outils de création de sites Web spécifient le chemin relatif au document actuel. Si la page de destination se situe dans le même dossier, seul le nom du fichier est utilisé. S'il s'agit d'un niveau supérieur à la page en cours, le nom du fichier est précédé de ../. C'est ce que nous appelons le chemin ou le lien relatif au document.

De plus, l'autre type de lien débute toujours par une barre oblique, indiquant la racine du site. L'avantage d'un chemin relatif vers la racine du site est que, quelle que soit la profondeur de la page actuelle dans la hiérarchie du site, la barre oblique (/) garantit que le serveur Web fera la recherche à partir du niveau supérieur du site. Bien que les liens relatifs à la racine du site soient beaucoup plus simples à lire, PHP ne peut pas les gérer.

Il est possible de convertir un chemin relatif à la racine du site en un chemin absolu en concaténant la variable globale $_SERVER ['DOCUMENT_ROOT'] au début du chemin, comme ce qui suit :

inclut $_SERVER['DOCUMENT_ROOT'].'/includes/namefile.php' ;

La majorité des serveurs prennent en charge $_SERVER['DOCUMENT_ROOT'], toutefois il est nécessaire de vérifier

la section des variables PHP dans les détails de configuration présentés par phpinfo() afin de vous assurer que cela fonctionne.

7

OBJETS

Vous avez peut-être entendu dire que la majorité des langages de programmation offrent le meilleur d'eux-mêmes avant d'être implémentés. Cela peut être vrai en ce qui concerne des langages faits pour des spécifications ambitieuses, mais PHP n'en fait définitivement pas partie. Ses humbles débuts sont démontrés par la signification originale de l'acronyme PHP : Page d'accueil personnelle. Il est né comme une façon facile d'ajouter du contenu dynamique aux pages HTML et est devenu un langage de programmation de plus en plus complet. L'orientation objet ne faisait pas partie du langage à l'origine, mais elle a progressivement pris de l'importance. De nos jours, il donne la majorité des fonctionnalités que les programmeurs attendent d'un langage orienté objet, tout en conservant son dynamisme et en nous laissant choisir notre style préféré de programmation. C'est pourquoi PHP est un outil essentiel dans votre boîte à outils.

Cela vous donne la chance de vous concentrer davantage sur l'optimisation de la conception et de sa mise en œuvre, au lieu de vous battre pour répondre aux exigences de la langue. À ce jour, la majorité des fonctionnalités qui existaient depuis longtemps dans Java et d'autres langages orientés objet ont été implémentées dans PHP. L'utilisation de ces nouvelles fonctionnalités est généralement facultative ; par conséquent, vous n'êtes pas contraint d'adopter un style de programmation plus proche

de Java si vous êtes habitué au style PHP et voulez le conserver. Nous débuterons ce chapitre en analysant les bases des objets, puis nous nous intéresserons à l'une des fonctionnalités les plus utiles de PHP : la gestion des exceptions. Par la suite, nous nous assurerons que nous comprenons la caractéristique la plus fondamentale du modèle objet de PHP : le fait que les objets sont manipulés par référence. Finalement, nous observerons la façon de manipuler magiquement les appels de méthodes en utilisant l'une des fonctionnalités les plus avancées de PHP.

Notez qu'il existe deux clés afin de comprendre le fonctionnement des objets et des classes. La première est de savoir la manière d'écrire une classe et d'utiliser les constructions de langage qui supportent la programmation orientée objet. L'autre sujet, moins simple et plus avancé, consiste à saisir la manière de faire interagir les objets de façon à atteindre l'objectif principal de la programmation orientée objet : un code maintenable, cela signifie la conception orientée objet. L'idée ici est de débuter par un peu de théorie et de pratiquer afin de la consolider avant de passer à des idées plus avancées. Toutefois, avant de débuter, une brève explication de la raison pour laquelle nous souhaitons comparer PHP à Java.

Comparaison avec Java

Les raisons de la comparaison sont pratiques. En effet, il n'y a pas de jugement de valeur implicite. Il n'y a aucune intention de faire une compétition entre les deux langages ou d'insinuer que Java est la seule ou la meilleure alternative à PHP. L'idée est plutôt d'apprendre quelque chose de la comparaison et de s'assurer que les détails sont excellents. La majorité de la syntaxe orientée objet de PHP est similaire à celle de Java. Comme plusieurs différences sont subtiles, il est facile de confondre les deux langues. Par exemple, le constructeur de l'interface est pratiquement identique dans les deux langages, mais en PHP, contrairement à Java, le

constructeur peut être spécifié dans l'interface comme les autres méthodes. La comparaison de deux langages de programmation similaires met en évidence et clarifie les détails spécifiques de chacun. Et comme Java est indéniablement un langage extrêmement populaire, il est probable que de plusieurs développeurs utilisent les deux langages et passent de l'un à l'autre.

Il est important de noter qu'il y a des développeurs PHP qui programment en Java occasionnellement ou souvent, et il est probable que quelques lecteurs soient des programmeurs qui ne connaissent pas PHP, mais détiennent une certaine expérience de Java.

Objets et classes

Les aspects cruciaux, les objets et les classes, sont documentés dans le manuel PHP, mais nous allons les analyser brièvement et, nous le souhaitons, obtenir une perspective légèrement distincte et nouvelle. D'après le manuel, une classe est "une collection de variables et de fonctions qui utilisent ces variables". Cette phrase peut être laconique, bien qu'il soit tout à fait possible d'avoir une classe sans variables. Les fonctions sont nommées méthodes dans la terminologie correcte de l'orientation objet. On pourrait dire qu'une classe est comme une maison. Les méthodes sont des pièces et la déclaration de classe représente les murs extérieurs de la maison. Diverses activités ont lieu dans des pièces distinctives : vous êtes en mesure de cuisiner dans la cuisine, dormir dans la chambre, et ainsi de suite. De même, chaque méthode d'une classe effectue un travail spécifique. Les murs ont un sens parce qu'ils protègent le code à l'intérieur de la maison du code à l'extérieur et vice versa. Si toutes les variables sont globales, vous ne pouvez jamais changer la manière dont une variable est utilisée sans risquer de créer un bug ailleurs dans le programme. Les fonctions en PHP permettent de protéger les variables en les rendant locales

à la fonction. Les classes étendent toujours ce concept en introduisant des variables qui appartiennent à un objet pour qu'elles puissent être employées dans de nombreuses méthodes sans être globales. Nous appelons cela l'encapsulation.

Analysons un exemple facile. Il est difficile de trouver des exemples orientés objet qui soient faciles et réalistes. Créons alors un scénario : nous devons créer une application Web qui produit des "Hello world !". Nous créons une classe qui génère du code HTML pour un message "Hello World".

```
class HelloWorld {
public $world = 'World' ;
function getHtml() {
return "<html><body> ".
"Bonjour, ".$this->monde." !".
"</body></html> " ;
}
}
```

Afin d'utiliser cette classe, vous devez faire quelque chose comme ce qui suit :

```
$greetings = new HelloWorld ; // Créer l'objet
echo $greetings-> getHtml() ; // Afficher le message d'accueil
```

Il est important de savoir que la classe génère le document HTML en concaténant des chaînes constantes et en saisissant le nom qu'elle a stocké dans la variable. Cette variable est nommée variable d'instance. Une variable d'instance appartient à l'objet et est disponible comme $this-> world dans n'importe quelle méthode de la classe. Le mot clé public déclare

la variable et la rend globalement disponible. De plus, l'utilisation de variables publiques n'est pas nécessairement une excellente pratique en PHP, mais elle permet de garder les choses faciles tant que l'on expérimente. Afin d'utiliser la variable d'instance dans la méthode getHtml(), faites-y référence en tant que $this->world. Notez que cette variable n'est utilisée que dans une seule méthode, mais les variables d'instance ne deviennent vraiment utiles que quand elles sont employées dans plus d'une méthode.

Constructeurs : création et initialisation

L'application "hello world" est un succès retentissant, mais malheureusement, plusieurs jours plus tard, votre patron vous annonce que le programme n'est pas compatible avec la devise de l'entreprise, "l'excellence universelle". Il suffit donc de permettre de spécifier le nom de la planète pendant la création de l'objet. Analysez l'exemple ci-dessous :

```
$greetings = new HelloWorld('Epsilon Eridani II') ;
echo $greetings-> getHtml() ;
```

Le mot-clé new crée une nouvelle instance de la classe. Il exécute aussi une méthode nommée constructeur que nous pouvons utiliser afin d'initialiser et de configurer l'objet. En PHP, les constructeurs sont nommés construct(). Alors maintenant nous pouvons utiliser le constructeur afin de définir le nom d'utilisateur :

```
class HelloWorld {
public $world ;
function __construct($world) {
$this->monde = $monde ;
}
```

```
function getHtml() {
return "<html><body> ".
"Bonjour ".$this->monde." !".
"</body></html> " ;
}
}
```

Les variables d'instance permettent à diverses méthodes de partager des variables même si elles ne sont pas globales. Alors, si vous possédez une ancienne application PHP qui utilise librement des variables globales, une astuce utile consiste à les transformer en variables d'instance dans une classe. Un système BBS peut contenir une fonction display_messages() avec la déclaration de variable globale ci-dessous :

```
global $db, $strings, $mode ;
```

$db est un objet représentant la connexion à la base de données, $strings est une collection de chaînes de caractères indépendante du langage, et $mode est un mode d'affichage (avec ou sans threads). Imaginons que nous souhaitions remanier cette application. Une possibilité est de les faire appartenir à une classe, par exemple MessageView. Les variables seraient donc des variables d'instance comme illustrés ci-dessous :

```
class MessageView {
public $db ;
public $strings ;
public $mode ;
function __construct($db, $strings, $mode) {
$this-> db = $db ;
$this->strings = $strings ;
```

```
$this->mode = $mode ;
}
function display_messages() {
$result = $this-> db->query('SELECT * FROM messages') ;
//etc...
}
}
```

Cet exemple démontre aussi la façon une variable d'instance peut posséder un autre objet, dans ce cas un objet représentant la connexion à la base de données. En PHP, cela veut dire que l'objet possède une référence à l'autre objet. Les références d'objets seront analysées plus loin dans ce chapitre. La classe MessageView a trois variables d'instance. $strings et $mode sont représentés comme des attributs. Comme $db est un objet, et peut-être un peu difficile, il peut être compris comme une classe séparée.

Héritage

Conceptuellement, l'héritage dans la programmation orientée objet est une manière d'exprimer les relations entre les catégories. Techniquement, l'héritage est une façon par lequel une classe est en mesure d'obtenir tout ou partie des caractéristiques d'une autre classe de façon facile. L'alternative est de créer une instance de l'autre classe et de l'utiliser, mais cela implique considérablement plus de travail. Une classe hérite des caractéristiques d'une autre classe tout simplement en ajoutant le mot clé extends à la déclaration de la classe. Observons ce qui se passe si nous créons une classe vide qui étend une autre classe :

```
class NewsArticle extends Document {
}
```

Il existe maintenant ce que nous appelons une relation parent-enfant entre les deux classes. En effet, la classe NewsArticle est la classe enfant ; la classe Document est le parent. Le résultat pratique de ce que nous avons fait est que la classe NewsArticle fonctionne exactement comme la classe Document. Si nous avions copié et collé l'ensemble de la classe Document et changé son nom, cela aurait eu le même effet. La différence ne réside pas dans la manière dont le code fonctionne, mais dans le fait que nous ne devons pas dupliquer le code. En fait, nous réutilisons la classe Document, et c'est une excellente idée.

C'est vraiment bien, sauf pour le fait évident que nous n'avons en général pas besoin de deux classes qui fonctionnent précisément de la même façon. Cela a plus de sens si la classe enfant possède une implémentation. De plus, nous pouvons ajouter de nouvelles méthodes et données ou écraser des méthodes. Par exemple, la classe NewsArticle peut avoir des variables, par exemple $newsSource ou $byline, qui ne sont pas présentes dans la classe Document. Continuons avec notre exemple précédent. Encore une fois, ce n'est pas vraiment réaliste, mais imaginons que nous souhaitions une classe plus générale : une classe qui puisse représenter n'importe quel document HTML, et pas seulement ceux contenant des messages de bienvenue. Débutons alors par insérer le document HTML de base :

```
class HtmlDocument {
function getHtml() {
return "<html><body> ".$this-> getContent().
"</body></html> " ;
}
function getContent() { return " ; }
}
```

La méthode getHtml() insère le résultat de la méthode getContent() entre les balises de début et de fin du document HTML et renvoie le résultat. La fonction getContent() est assez inutile parce qu'elle renvoie une chaîne vide, mais HtmlDocument est juste notre classe parente. Notez que nous pouvons ajouter une classe enfant qui fait quelque chose de plus similaire à ce que nous avons exécuté précédemment :

```
class HelloWorld extends HtmlDocument {
public $world ;
function __construct($world ) {
$this->name = $world ;
}
function getContent() {
return "Bonjour, ".$this->monde." !";
}
}
```

En effet, la méthode getContent() de la classe HelloWorld écrase maintenant la méthode getContent() de sa classe mère. De plus, la méthode getHtml() fonctionne comme si nous l'avions copiée de la classe HtmlDocument vers la classe HelloWorld. En effet, cette classe fait alors le même travail que notre classe HelloWorld précédente, mais la méthode getHtml() est désormais dans la classe parente. Cela veut dire que nous pouvons créer une autre classe qui étend la classe HtmlDocument et place quelque chose d'autre à l'intérieur du document. Il est important de noter que l'héritage n'est pas seulement un privilège des méthodes ordinaires, les constructeurs peuvent aussi en bénéficier. Une partie du travail requise à la construction d'un objet peut être commune à des classes similaires et

d’autres peuvent être différentes. Remarquez qu’en PHP, un nouveau style de constructeur a été introduit, ce qui rend les choses plus faciles. Plutôt que d’utiliser un constructeur portant le même nom que la classe, nous pouvons utiliser la méthode spéciale __construct(), comme le démontre l’exemple ci-dessous :

```
class Document {
protégé $title ;
protégé $text ;
function __construct($title,$text) {
$this->titre = $titre ;
$this->text = $text ;
}
}
```

Il est ainsi légèrement plus simple d’hériter du comportement des constructeurs que des constructeurs de l’ancien style :

```
classe NewsArticle extends Document{
privé $introduction ;
function __construct($title,$text,$introduction) {
parent::__construct($title,$text) ;
$this->introduction = $introduction ;
}
}
```

L’instruction parent::construct appelle le constructeur de la classe Document. Une autre question est de savoir s’il est utile d’hériter du comportement du constructeur. Nous pourrions courir le risque de rendre le remaniement plus difficile. Effectivement, dans l’exemple illustré, il serait

préférable de déplacer toute la construction vers la classe fille et de la dupliquer dans les autres classes filles pour rendre le code plus lisible et plus facile à changer. Cette petite quantité de duplication n'est pas susceptible de nuire à la lisibilité ou à la maintenabilité de votre code. Jusqu'à maintenant, nous avons étudié le fonctionnement des objets dans des circonstances normales. Plus tôt, nous avons remarqué qu'une classe est comme une maison avec des pièces. Maintenant, nous voulons savoir ce qu'il faut faire en cas d'incendie. Nous souhaitons être en mesure de sortir rapidement, mais en toute sécurité. Afin de rendre cela possible, les langages orientés objet (dont PHP à partir de la version 5) utilisent une fonctionnalité nommée exceptions.

8

EXCEPTIONS

La manière la plus facile de gérer une erreur en PHP 4 était d'utiliser la méthode die() en cas d'erreur. Martin Fowler l'appelle "l'équivalent logiciel du suicide au cas où vous auriez manqué un vol". Toutefois, il ajoute que "si le coût de l'arrêt du programme est faible et que l'utilisateur est tolérant, l'abandon d'un programme ne pose aucun souci". En PHP, comme dans plusieurs autres langages, nous détenons une alternative au suicide : lancer une exception. C'est l'équivalent logiciel de sauter par la fenêtre d'une maison ou d'un bâtiment en espérant qu'il y ait des pompiers ou quelqu'un avec un tapis avant que vous ne touchiez le sol. Remarquez que si nous ne traitons pas l'exception en utilisant catch à un moment spécifique dans le code qui appelle cette classe (directement ou indirectement), le programme s'arrêtera et imprimera un message avec une trace de la pile d'appel. Alors, l'effet immédiat est approximativement le même que die(), mais si nous choisissons plus tard que nous voulons gérer l'erreur, nous pouvons le faire plus facilement. Les mécanismes d'utilisation des exceptions sont une chose ; les utiliser de manière judicieuse et judicieuse en est une autre. Cette section n'illustrera pas l'ensemble des détails des exceptions. En effet, elle se concentrera plutôt sur les manières judicieuses d'utiliser les exceptions et les aspects du traitement des exceptions de PHP qui sont les plus utiles pour les supporter.

Plus de détails sur les aspects techniques des exceptions PHP peuvent être trouvés dans le manuel PHP en ligne. En ce qui concerne un examen approfondi de la façon d'utiliser les exceptions, consultez le chapitre consacré à ce sujet dans le livre Object Design de Rebecca Wirfs-Brock et Alan McKean. Dans cette section, nous allons débuter par découvrir le fonctionnement des exceptions. Par la suite, nous analyserons comment et quand il est approprié d'utiliser les exceptions et quand il pourrait être préférable d'utiliser les retours à l'ancienne. Nous observerons comment créer nos propres classes d'exceptions et ferons de notre mieux pour trouver la manière de remplacer les erreurs PHP intégrées par des exceptions. Enfin, nous observerons la façon d'éviter une utilisation excessive des exceptions.

La construction du langage de programmation connue sous le nom d'exception est une façon de communiquer des conditions d'erreur ou d'exception entre différentes parties du programme. Par exemple, nous pouvons avoir le nom de la base de données utilisée par l'application dans une variable d'environnement nommée DB_NAME. Sans traitement des exceptions, nous pourrions permettre à une méthode de récupérer ce nom comme ci-dessous :

```
public function getDatabaseName() {
si ( ! array_key_exists('DB_NAME',$_ENV))
die('La variable d'environnement DB_NAME n'est pas définie') ;
retourner $_ENV['DB_NAME'] ;
}
```

Voici alors la version suicide, qui utilise die() au lieu de exit pour rendre le parallèle clair, bien que les deux aient précisément le même effet. Mais utiliser une exception à la place est vraiment facile :

```
public function getDatabaseName() {
si ( ! array_key_exists('DB_NAME',$_ENV))
```

lance une nouvelle exception (
"La variable d'environnement DB_NAME n'est pas définie") ;
retourner $_ENV['DB_NAME'] ;
}

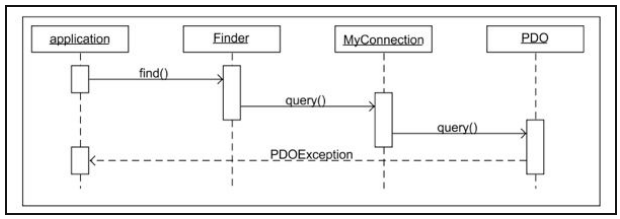
Si nous ne faisons rien pour attraper l'exception, cela a le même effet que die() : l'application s'arrête. Toutefois, il fait une chose supplémentaire : il imprime une trace de la pile qui peut être utile pour le débogage. Vous pouvez détenir une trace de la pile sans utiliser d'exceptions grâce aux fonctions debug_backtrace() et debug_print_backtrace(). Toutefois, lancer une exception est une façon encore plus facile.

Erreur fatale : exception non attrapée 'Exception' avec message
La variable d'environnement DB_NAME n'est pas définie" in /path/exception.php:6
Trace de la pile :
#0 /path/exception.php(6) : Config::getDatabaseName()
#1 /path/exception.php(12) : Config-> getDatabaseName()
#2 {main}
lancé dans /path/exception.php à la ligne 6

Si nous ne souhaitons pas que les utilisateurs voient le rapport d'erreur technique (et nous ne le faisons en général pas pour des raisons de sécurité), il est plus sûr d'avoir utilisé des exceptions plutôt que die(). Si nous avons utilisé die() à de nombreux endroits, nous devrons probablement trouver toutes les occurrences et les remplacer une par une. Si nous utilisons des exceptions, il suffit de les attraper à un seul endroit, par exemple au niveau le plus élevé de l'application. Nous pourrions enregistrer le message et

rediriger l'utilisateur vers une page qui affiche tout simplement qu'une erreur s'est produite :

```
$config = new Config ;
essayez {
$config-> getDatabaseName() ;
}
catch(Exception $e) {
$logger->log($e-> getMessage()) ;
header("Emplacement : unrecoverable.php") ;
}
```



L'image est un diagramme de séquence UML montrant le fonctionnement des exceptions. Les détails du diagramme, les noms des classes et des méthodes, ne sont pas essentiels. L'essentiel est ce qui suite : quand l'objet PDO génère une exception, au lieu de retourner à MyConnection, il contourne à la fois MyConnection et Finder, puisque ni l'un ni l'autre n'a de blocs try et catch. Toutefois, l'application attrape l'exception et la traite. Si vous faites pivoter le diagramme de 90 degrés dans le sens inverse des aiguilles d'une montre, il sera orienté de façon à correspondre à l'analogie du bâtiment : le flux de messages illustre les escaliers, passe les étages un par un, saute par la fenêtre et est attrapé dans le réseau au rez-de-chaussée : par l'application. Il y a un avantage fondamental que les exceptions partagent avec die() : elles interrompent la suite du traitement. La raison pour laquelle cela est utile est qu'en général, quand une exception est

générée, le reste de ce qui se passe dans la méthode actuelle n'a pas de sens. Un exemple modeste est la méthode getDatabaseName() que nous venons d'observer : renvoyer le nom de la base de données à partir de la fonction n'a aucun sens puisqu'il n'y a pas de nom de base de données à renvoyer. Aucun des programmes PHP n'est exécuté entre le moment où l'exception est générée et le moment où l'exception est attrapée.

La signification syntaxique d'un mot clé en langage de programmation est souvent distincte de la signification conceptuelle du mot. Cela s'applique aussi en cas d'exceptions. Le mot "exception" veut dire quelque chose qui se produit rarement. Dans la conception de logiciels, il existe une distinction entre les erreurs et les exceptions. Une erreur est en général quelque chose qui est fatal ou qui paralyse la capacité du programme à effectuer son travail ; une exception est une situation peu commune, mais récupérable. En pratique, les exceptions (au sens syntaxique du terme) sont vraiment utiles afin de gérer des erreurs comme celle que nous avons vue. Si le nom de la base de données n'est pas disponible et que l'application en est totalement dépendante, la possibilité de prévoir des exceptions afin d'empêcher la poursuite du traitement à ce moment-là est appropriée et utile. Essayer d'exécuter des requêtes SQL avec une base de données inexistante et tenter de traiter des données inexistantes ne peut que générer d'autres erreurs potentiellement déroutantes. Le code de gestion des erreurs peut être contre-productif : si vous en créez trop, le programme sera moins lisible et il sera plus difficile de détecter les bogues. Si nous détenons une excellente couverture en termes de tests unitaires, la gestion des erreurs est surtout nécessaire aux limites de notre application : ses interfaces avec d'autres systèmes et le reste du monde. Même si votre logiciel est peuplé exclusivement d'objets bien formés, vous devez patrouiller les limites. L'une des limites est constituée par les ressources données par le système d'exploitation, les fichiers, le réseau et les bases de données. Vous le savez

probablement déjà, mais résumons certaines erreurs typiques dans une application Web PHP :

- Erreurs dues à des informations de configuration incorrectes, comme le mot de passe requis pour se connecter à une base de données ;
- Erreurs de syntaxe SQL ou XML ;
- Fichiers essentiels qui ne peuvent pas être lus.

Toutefois, il se peut que nous devions également patrouiller l'autre frontière : celle d'un autre logiciel qui utilise notre programme. Les contrôles de sécurité et la validation des entrées de l'utilisateur ont en général lieu aux niveaux les plus élevés de l'application ; la réponse est plus susceptible d'être un message direct à l'utilisateur qu'une exception. Cependant, qu'en est-il si nous créons un logiciel de bibliothèque de bas niveau qui est utilisé par d'autres ? Vérifier les entrées et s'assurer qu'il n'y a pas de valeurs absurdes peut faire gagner du temps considérable pendant le débogage.

Un exemple possible est un paquet qui communique des informations statistiques sur les données stockées dans une base de données. Généralement, les clients de cette bibliothèque devront donner l'heure de début et l'heure de fin des statistiques. Que se passe-t-il si l'heure de début ou l'heure de fin, ou les deux, sont NULL, 0 ou une autre valeur inappropriée ? S'ils sont tous deux nuls et que nous les interprétons comme le 1er janvier 1970, nous retournerons très probablement un ensemble de données vide. De plus, trouver la raison de l'ensemble de données est vide peut prendre des heures.

En supposant que nous soyons uniquement responsables du générateur de statistiques, nous souhaitons nous assurer qu'il dispose d'une couverture de test ; par conséquent, nous examinons les interfaces pour les erreurs et

les entrées invalides. Cela concerne les erreurs. Pour les exceptions au sens conceptuel du terme, pour les situations rares, mais récupérables, il peut fréquemment être plus utile d'utiliser la logique conditionnelle ordinaire et les codes de retour afin de rétablir tout de suite l'erreur, au lieu de générer une exception. Il existe de nombreuses manières de procéder. La meilleure solution consiste en général pour la classe appelante à demander à la classe que nous appelons les informations qui lui permettront de choisir si une situation d'exception existe en premier lieu. Une autre solution consiste à renvoyer un code d'erreur. Une autre solution encore consiste à ignorer purement et tout simplement le souci. Par exemple, s'il manque un fichier parmi un ensemble de fichiers à traiter, l'utilisateur final peut préférer un résultat incomplet à un message d'erreur.

Création d'une exception

La classe Exception est intégrée à PHP lui-même et est toujours disponible. En créant des classes enfant qui étendent la classe Exception, nous pouvons définir nos propres types d'exception. Cela peut être aussi facile que cela :

```
class ConfigurationException extends Exception {}
```

Comme nous l'avons vu précédemment, une classe qui étend une autre classe, mais ne détient aucune implémentation fonctionne exactement comme la classe originale. Donc pourquoi souhaitions-nous faire ça ? Parce que les exceptions sont légèrement différentes des classes ordinaires. Afin de distinguer les divers types d'exceptions, il est d'usage d'utiliser différentes classes. Et comme la clause catch vous permet de spécifier la classe d'exception, vous pouvez l'utiliser afin d'attraper différentes exceptions à divers endroits. Par exemple, si vous utilisez une classe d'exception pour l'échec de la connexion à une base de données et une

autre pour les erreurs de syntaxe SQL, vous pouvez attraper les deux à des endroits différents dans votre code. D'autre part, c'est une mauvaise idée de créer de grandes hiérarchies de classes d'exceptions.

WirfsBrock et McKean conseillent un maximum de cinq ou sept classes d'exceptions diverses pour la simple raison qu'il est difficile de se souvenir de trop de classes. Au lieu de cela, il est possible d'utiliser des codes d'erreur afin de distinguer les différents sous-types au sein d'une classe d'exception. Une manière lisible et sûre de le faire est d'utiliser la capacité de la classe Exception à stocker le code d'erreur avec les constantes de la classe. Si nous souhaitons une ConfigException capable de signaler à la fois les erreurs de connexion à la base de données et les erreurs de syntaxe SQL, nous pouvons définir la classe Exception comme ce qui suit :

```
class ConfigException extends Exception {
const SQL_SYNTAX_ERROR = 1 ;
const DB_CONNECTION_ERROR = 2 ;
}
```

Quand nous lançons l'exception, nous sommes en mesure de spécifier à la fois le message d'erreur et le code d'erreur, puisque ceux-ci sont acceptés par le constructeur de la classe Exception.

```
lancez une nouvelle ConfigException(
"Impossible de se connecter à la base de données $dbname',
ConfigException::DB_CONNECTION_ERROR) ;
```

Et quand nous attrapons l'exception, nous pouvons tester le code d'erreur et agir en conséquence :

```
catch(ConfigException $e) {
switch ($e-> getCode()) {
```

```
cas ConfigException::DB_CONNECTION_ERROR :
echo 'Erreur de connexion' ;
pause ;
cas ConfigException::SQL_SYNTAX_ERROR :
echo 'Erreur SQL' ;
pause ;
}
}
```

Dans la vie réelle, bien sûr, nous ferions quelque chose de plus sophistiqué que de simplement imprimer la chaîne à l'écran. Toutefois, que faire si nous ne souhaitons traiter qu'un seul de nos sous-types d'exception ici et traiter l'autre type ailleurs ? C'est facile : nous sommes en mesure de le relancer afin qu'il soit attrapé par une méthode ou un objet divers :

```
cas ConfigException::SQL_SYNTAX_ERROR :
lancez $e ;
pause ;
```

C'est une excellente idée de nommer les classes d'exception en fonction de ce qui a mal tourné, au lieu d'utiliser un nom faisant référence à l'endroit où le souci s'est produit. La classe ConfigException dans les exemples ci-dessus est destinée à transmettre l'idée qu'il s'agit d'exceptions qui sont en général causées par des erreurs de configuration ou des bogues dans l'application.

Après avoir utilisé les exceptions, il est un peu irritant que les erreurs PHP soient signalées comme des erreurs de style PHP 4 au lieu d'exceptions, surtout si vous portez une ancienne application. Notez qu'il est heureusement possible d'établir un "pont" entre l'ancien système de

traitement des erreurs et le nouveau. Bien que cela ne permette pas de détecter l'ensemble des erreurs (les erreurs d'exécution irréversibles, comme l'appel d'une méthode inexistante sur un objet, ne seront pas signalées), la gestion des erreurs sera plus cohérente. Les premières choses requises sont une classe d'exception afin de distinguer les erreurs PHP des autres exceptions et un simple gestionnaire d'erreur pour recevoir une erreur PHP et lancer une exception à la place :

```
class ErrorFromPHPException extends Exception {}
function PHPErrorHandler($errno, $errstr, $errfile, $errline) {
lancer une nouvelle ErrorFromPHPException($errstr,$errno) ;
}
```

Nous sommes désormais en mesure de configurer le gestionnaire d'erreurs. Si nous tentons d'ouvrir un fichier inexistant, nous obtiendrons une exception au lieu de la bonne vieille erreur :

```
$oldHandler = set_error_handler('PHPErrorHandler') ;
fopen('/tmp/non-existent', 'r') ;
```

Et si, pour une raison quelconque, nous souhaitons revenir à la façon ordinaire de traiter ces erreurs, nous pouvons le faire :

```
set_error_handler($oldHandler) ;
```

Nous souhaitons éviter d'encombrer notre code avec trop de gestion des erreurs, et les exceptions nous aident à le faire parce que les instructions catch peuvent ne pas satisfaire aux conditions de gestion des erreurs qui doivent tester les codes de retour de chaque appel de méthode. Mais même

avec des exceptions, il n'y a aucune raison de vérifier tous les soucis possibles. Comme le mentionnent Wirfs-Brock et McKean :

Les collaborations défensives, c'est-à-dire la conception d'objets qui prennent des précautions avant et après avoir appelé un collaborateur, sont coûteuses et sujettes à des erreurs. Ces responsabilités ne doivent pas être confiées à tous les objets.

Heureusement, PHP ne vous force jamais à vérifier quoi que ce soit. La gestion des exceptions est l'une des fonctionnalités les plus essentielles introduites en PHP. Un changement encore plus crucial était la nouvelle manière de gérer les références d'objets.

9

RÉFÉRENCE DE L'OBJET

Quand la police recherche un criminel recherché ou une personne disparue, il est utile de disposer d'une photographie de l'individu. Une excellente photographie peut faciliter la reconnaissance d'une personne, mais elle n'illustre qu'un moment particulier. Les gens changent de vêtements, se maquillent, se coupent les cheveux, se rasent, se laissent pousser la barbe, mettent des lunettes de soleil et ont même recours à la chirurgie plastique. Tôt ou tard, il devient complexe de reconnaître la personne sur la photo. Plus évident encore et plus essentiel est le fait que faire quelque chose à la photographie n'affectera pas la personne. Mettre l'image dans une cellule de prison est inutile. À moins que vous ne croyiez aux rites vaudous, vous devez vivre avec le fait que l'image et la personne sont alors des limites à ce que vous pouvez faire si vous n'avez que la photographie à votre disposition. Ce n'est jamais la même chose que d'avoir la personne présente.

De plus, la gestion des objets PHP est similaire. Avant PHP 5, une copie d'un objet était créée à chaque fois qu'une tâche était exécutée ou qu'un objet était retourné par une fonction ou une méthode. Donc, vous détenez un instantané qui ressemble faussement à l'original, mais qui est en fait un objet divers et qui ne reflète ni ne provoque de changements dans l'original. Cela crée quelques problèmes, similaires à l'analogie de l'instantané. Dans

la programmation orientée objet, un objet représente en général une entité, réelle ou abstraite, qui ne peut pas être tout simplement modifiée par procuration. Changer une copie d'un document ne servira à rien si l'original est celui qui est enregistré dans la base de données. La modification d'un objet représentant le titre d'une page HTML ne servira à rien si l'original est celui qui s'affiche dans le navigateur. Cependant, contrairement à une photographie, la copie d'un objet a toute la masse et le poids de l'original. Si l'objet original détient deux mégaoctets de données, la copie possède aussi deux mégaoctets de données, de sorte que vous avez maintenant quatre mégaoctets en tout. Donc, la copie d'objets fait que le programme consomme plus de mémoire que nécessaire.

Voilà pourquoi la manipulation d'objets dans ce style en PHP est universellement reconnue comme un aspect négatif. En effet, cela semblait être une excellente idée au moment où elle a été mise en œuvre, mais, en réalité, ce n'était pas le cas. Les personnes qui ont développé PHP ne voulaient pas passionnément ce genre de comportement d'objet. C'était juste plus simple étant donné la façon dont PHP était implémenté. L'orientation objet n'était pas vraiment utilisée en PHP dans le passé, mais dès qu'elle a été réévaluée, la programmation orientée objet en PHP est devenue assez populaire. Finalement, il est devenu évident que la manière dont PHP gérait les objets était un handicap, il est alors devenu une priorité urgente de modifier le comportement par défaut des objets en PHP afin d'utiliser les références plutôt que des copies.

Cela a débuté avec PHP 5 et s'est propagé dans les versions suivantes. L'orientation objet en PHP fonctionne désormais de la même façon que dans la majorité des autres langages orientés objet. PHP 4 a aussi des références, mais elles sont distinctes des références d'objets dans la majorité des langages orientés objet. Ils peuvent être utilisés, et l'ont été, pour la programmation orientée objet en PHP 4, mais il est difficile de saisir

la façon qu'ils fonctionnent et quelquefois ils font des choses auxquelles on ne s'attend pas. Leur comportement est contre-intuitif.

Les objets PHP (à partir de la version 5) se comportent la majorité du temps de façon utile et naturelle. La variable détient une référence à l'objet et seule la référence est copiée. Comme nous l'avons mentionné, les références d'objets contribuent à améliorer les performances en empêchant la copie d'objets et la consommation excessive d'espace mémoire. Dans les applications PHP 4,de considérables efforts ont été faits afin d'éviter cette surcharge en copiant explicitement les objets par référence. Cela a du sens si vous avez beaucoup d'objets ou s'ils sont très grands. Tentez de télécharger un objet PEAR DB et vous observerez ce que je veux expliquer par grands objets. D'autre part, si vous gardez un design simple, cela vous aidera aussi à garder vos objets plus petits.

De plus, en PHP 5, ces efforts ne sont plus requis ; avoir des objets représentés par des références a aussi des avantages pour la conception orientée objet. Il facilite la construction et la manipulation de structures d'objets complexes. Imaginons qu'un objet $dog se trouve à l'intérieur de l'objet $doghouse, par la suite que vous changiez l'objet $dog et que vous souhaitez que cela se reflète dans $doghouse. Les interfaces graphiques présentent en général ce type de structure complexe. En programmation Web, nous travaillons avec des documents HTML, mais imaginons que nous représentions les éléments d'un document HTML comme des objets. Nous serions en mesure de faire quelque chose comme ce qui suit :

```
$checkbox = new Checkbox ;
$form = nouveau formulaire ;
$document = nouveau Document ;
$document->add($form) ;
$form-> add($checkbox) ;
```

Maintenant, que se passe-t-il si nous modifions l'un des éléments internes ?

De plus, en PHP 5, cela va changer et quand nous générerons du code HTML à partir de l'objet Document, il aura une case à cocher sélectionnée. C'est évidemment ce que nous souhaitons et cela montre ce que je veux expliquer quand je mentionne que le comportement des objets PHP est le plus fréquemment intuitif, utile et naturel. Les références d'objets peuvent être merveilleusement intuitives la majorité du temps, mais d'autres fois, nous souhaitons activement que les objets soient copiés au lieu d'être transmis par référence. C'est le cas des types d'objets nommés objets de valeur. Si nous représentons les dates, les sommes d'argent et autres comme des objets, il sera plus naturel de les copier parce qu'ils n'ont pas d'identité. Afin de copier des objets en PHP, utilisez le mot clé clone.

10

MÉTHODES ET CLASSES

En PHP, il se trouve une fonctionnalité nommée "appels de méthodes surchargeables". Essentiellement, la fonction nous donne la chance d'intercepter, de rediriger et de redéfinir les appels de méthode. C'est comme voler le courrier de quelqu'un et l'ouvrir. Nous sommes en mesure par la suite de l'envoyer à quelqu'un d'autre, changer son contenu ou même le jeter à la poubelle. Cela veut dire que nous sommes en mesure de changer la façon habituelle dont les méthodes répondent et même répondre à des méthodes inexistantes. Nous débuterons ce chapitre en clarifiant le nom de cette fonctionnalité et comment elle est liée à l'idée d'intercepter les appels de méthode. Nous analyserons par la suite certains exemples d'utilisation : la surcharge de méthodes à la Java et un mécanisme généralement d'enregistrement des appels de méthodes. Finalement, nous nous pencherons sur un sujet connexe : la façon d'utiliser la surcharge pour contrôler ce qui se passe lorsqu'une classe est instanciée. Le terme "surcharge de méthodes" peut être un peu bizarre, car il a une signification spécifique dans d'autres langues. En Java et en C++, la surcharge de méthodes consiste à écrire diverses méthodes portant le même nom, mais avec des nombres ou des types d'arguments différents, la méthode exécutée dépendant des arguments donnés. Ceci est tout particulièrement utile dans les langages à typage statique (comme Java et C++). Sans la surcharge des

méthodes, deux méthodes avec des noms divers pourraient être nécessaires juste afin de gérer des arguments de types différents (par exemple, une date spécifiée comme une chaîne ou bien un horodatage numérique). En PHP, c'est quelque chose de plus général. Vous êtes en mesure de surcharger les appels de méthode, toutefois vous devez définir vous-même la surcharge. Cela fonctionne comme ce qui suit : si vous tentez d'appeler une méthode qui n'est pas définie, PHP appellera une méthode appelée __call() à la place. Vous êtes en mesure de faire ce que vous souhaitez avec l'appel à la méthode "failed". Vous êtes en mesure d'exécuter une autre méthode, éventuellement sur un autre objet, ou donner un message d'erreur différent. Vous pouvez aussi ne rien faire ; cela permettra à PHP d'ignorer les appels de méthode échoués plutôt que de générer une erreur irrécupérable.

De plus, cela peut être utile de temps en temps, mais en général, il faut se douter de tout ce qui réduit le niveau de vérification des erreurs et donne la chance aux bogues de passer inaperçus. Notez que ce comportement n'est pas une surcharge de méthode, mais il donne la chance de définir une surcharge de méthode ; ainsi, il rend les appels de méthode surchargeables. Le terme de surcharge veut dire qu'un même élément (dans ce cas, un nom de méthode) peut avoir des significations distinctes selon le contexte. De plus, puisque __call() nous donne la chance de contrôler le contexte et de réagir en fonction de celui-ci, la surcharge des méthodes est l'une des choses que nous pouvons faire avec elle. Il est quelquefois pratique de pouvoir appeler la même méthode avec un nombre variable d'arguments. PHP rend cela possible grâce à sa capacité à définir des arguments optionnels avec des valeurs prédéfinies. Toutefois, il est quelquefois nécessaire que la méthode ait un comportement sensiblement distinct selon la liste des arguments. Dans les langages qui n'ont pas de surcharge de méthode, cela veut dire d'ajouter une logique conditionnelle au début de la méthode. Si vous êtes en mesure d'utiliser la surcharge de méthode à la place, vous pouvez sauter la logique conditionnelle et votre code sera plus

propre et plus élégant. En effet, vous pouvez implémenter ce type de surcharge de méthode en utilisant __call() en PHP 5. Observons un exemple. Imaginons que nous allons réutiliser le comportement de surcharge, alors plaçons-le dans une classe parent abstraite :

```
abstract class OverloadableObject {
function __call($name,$args) {
$method = $name. "_".count($args) ;
si ( ! method_exists($this,$method)) {
lance une nouvelle exception ("Call to undefined method".
get_class($this). "::$method") ;
}
return call_user_func_array(array($this,$method),$args) ;
}
}
```

La majorité des comportements de cette classe sont définis par une seule ligne. Si une méthode non définie est nommée, la méthode __call() génère un nouveau nom pour la méthode, composé de la méthode originale et du nombre d'arguments, séparés par un trait de soulignement. Il appelle par la suite la méthode avec le nouveau nom généré, en passant les arguments originaux. Toutefois, si nous souhaitons créer une méthode surchargée nommée multiply qui peut être appelée avec un ou deux arguments et qui multipliera dans les deux cas, nous créons deux méthodes nommées respectivement multiply_2 et multiply_3 :

```
class Multiplier extends OverloadableObject {
function multiply_2($one,$two) {
retourner $one * $two ;
}
function multiply_3($one,$two,$three) {
```

```
retourner $one * $two * $three ;
}
}
```

Afin de l'utiliser, il suffit d'appeler la méthode avec deux ou trois arguments :

```
$multi = nouveau Multiplicateur ;
echo $multi-> multiply(5,6)."\n" ;
echo $multi-> multiply(5,6,3)."\n" ;
```

Ce n'est toujours pas précisément la même chose que la surcharge des méthodes en Java et en C++ parce que nous vérifions uniquement le nombre d'arguments, et non leurs types. Toutefois, nous pourrions aussi utiliser des informations sur le type. D'un autre côté, comme nous l'avons vu, rendre le comportement dépendant des types d'arguments est moins essentiel en PHP que dans les langages typés statiquement. Nous avons analysé le fonctionnement des appels de méthodes surchargeables. Pour un exemple de leur utilisation, observons la façon ils peuvent être utilisés afin d'enregistrer les appels de méthode.

La programmation orientée aspect est une façon relativement nouvelle de faire quelques choses qui ne sont pas autant élégantes dans la simple programmation orientée objet. Analysons, par exemple, le souci de l'enregistrement du début et de la fin de tous les appels de méthode dans une application. Afin de faire cela en POO, nous devons ajouter du code à chaque méthode. Nous sommes en mesure de nous efforcer de rendre ce code supplémentaire minimal, mais il ajoutera certainement un encombrement substantiel à nos classes. La journalisation est en général le type de souci abordé par la programmation orientée aspect. Ces problèmes,

nommés problèmes transversaux, touchent divers modules ou sous-systèmes et sont complexes à isoler dans des classes distinctes. Un autre exemple serait de vérifier si l'utilisateur actuel est autorisé ou non à utiliser une méthode. La programmation orientée aspect est en général réalisée en définissant des aspects, des constructions de classe qui sont insérées dans le code lors d'un processus de génération de code. Ici, nous allons faire quelque chose de considérablement plus simple et plus primitif en utilisant __call() en PHP. Nous employons la classe PEAR Log et contrôlons le processus de journalisation à partir de la méthode __call() d'une classe mère, comme affiché :

```
class LoggingClass {
function __call($method,$args) {
$method = "_$method" ;
si ( ! method_exists($this,$method))
lance une nouvelle exception ("Appel à une méthode non définie").
. get_class($this). "::$method") ;
$log = Log::singleton('file', '/tmp/user.log',
'Méthodes', NULL, LOG_INFO) ;
$log->log("Just starting method $method") ;
$return = call_user_func_array(array($this,$method),$args) ;
$log->log("Vient de terminer la méthode $method") ;
retourner $return ;
}
}
```

Il est important de noter que cette situation est similaire à notre exemple de surcharge de méthode, dans la mesure où la méthode réelle a un nom légèrement distinct de celui que nous appelons depuis le code client. La méthode que nous appelons depuis le code client n'existe pas, alors __call()

l'intercepte, enregistre le début, appelle la méthode réelle et enregistre la fin. Afin de l'utiliser, il est nécessaire d'étendre LoggingClass et donner aux méthodes des noms débutant par un trait de soulignement. Il n'y a aucune raison pour que ce soit un trait de soulignement. De plus, notez que vous pouvez utiliser tout ce qui rend les noms uniques. Voici ci-dessous une classe simplifiée afin de gérer les dates et les heures :

```
class DateAndTime extends LoggingClass {
privé $timestamp ;
function __construct($timestamp=FALSE) {
$this-> init($timestamp) ;
}
protected function _init($timestamp) {
$this->timestamp = $timestamp ? $timestamp : time() ;
}
function getTimestamp() { return $this->timestamp ; }
protected function _before(DateAndTime $other) {
retourner $this->timestamp < $other-> getTimestamp() ;
}
}
```

Les méthodes init() et before() seront enregistrées tandis que la méthode getTimestamp() ne sera pas enregistrée, car le nom ne débute pas par un caractère de soulignement. J'ai ajouté la méthode init() afin de donner aussi la chance à la journalisation de la construction de l'objet. Remarque que la méthode __call() n'est normalement pas activée lors de la construction. En effet, ce n'est pas surprenant, puisqu'une classe n'est pas obligée d'avoir un constructeur. Vous aurez observé que les méthodes sont déclarées protégées, ce qui veut dire qu'elles ne peuvent pas être appelées par le code client, sauf via le mécanisme __call(). De plus, ils sont protégés plutôt que privés, car

la méthode __call() se trouve dans une classe mère. Maintenant, essayons la classe et analysons ce qui se passe. Nous créons deux objets DateAndTime distincts, et par la suite, nous les comparons :

```
$now = nouveau DateAndTime ;
$nexthour = new DateAndTime(time() + 3600) ;
print_r(array($now,$nexthour)) ;
if ( $now->before($nexthour) ) {
echo 'OK
}
```

Les appels de méthode sont enregistrés de la façon illustrée ci-dessous :

04 15:20:08 Méthodes [info] Démarrage de la méthode _init

Juillet 04 15:20:08 Méthodes [info] Vient de terminer la méthode _init

04 juillet 15:20:08 Méthodes [info] Démarrage de la méthode _init

Juillet 04 15:20:08 Méthodes [info] Vient de terminer la méthode _init

Juillet 04 15:20:08 Méthodes [info] Démarrage de la méthode _before

Juillet

Il est important de savoir que nous sommes loin de la programmation orientée aspect (POA) dans un langage POA spécialisé. De plus, dans la pratique, si vous souhaitez enregistrer les appels de méthode, vous êtes en mesure de chercher un outil de profilage. Toutefois, il semble y avoir un potentiel pour des applications utiles. Les appels de méthodes surchargeables sont une sorte de magie qui nous donne la chance de définir ce qui se passera chaque fois qu'une méthode, quelle qu'elle soit, sera

appelée. Effectivement, l'autoloading de classe est un concept similaire. Il donne la chance de définir ce qui se passe chaque fois que nous tentons d'employer une classe non définie, n'importe quelle classe non définie. PHP a une façon d'éviter d'inclure les classes manuellement en automatisant le processus de chargement des classes. Vous êtes en mesure de définir une fonction appelée __autoload() qui sera exécutée chaque fois que vous essayerez de créer une instance d'une classe non définie. Cette fonction peut ensuite inclure le fichier de classe approprié. Le code ci-dessous illustre un exemple légèrement plus sophistiqué :

```
function __autoload($className) {
include_once __autoloadFilename($className) ;
}
function __autoloadFilename($className) {
return str_replace('_','/',$className).". php" ;
}
```

La fonction __autoloadFilename() génère le nom du fichier à inclure. Une fonction distincte pour cela faciliterait les tests. Nous sommes en mesure d'effectuer un test sur la fonction __autoloadFilename() et vérifier que la valeur renvoyée est correcte. Vérifier qu'un fichier a été inclus est plus complexe que de simplement vérifier la valeur retournée.

La fonction str_replace remplace l'ensemble des underscores par des slashs. Alors, si le nom de la classe est HTML_Form, la fonction __autoload() inclura le fichier HTML/Form.php. Cela facilite le tri des classes dans divers répertoires de la façon standard de PEAR. Si vous avez de vraiment petites classes (et vous en avez déjà observé dans ce livre), vous êtes en mesure de trouver pratique de conserver plus d'une classe dans un fichier. Notez que vous pouvez combiner cela avec le chargement automatique en créant un lien dans le système de fichiers. Imaginons que

vous possédez une classe Template et une classe Redirect et existent toutes deux dans un fichier nommé Template.php. Sous Linux ou UNIX, vous pourriez faire ce qui suit :

ln -s Template.php Redirect.php

Maintenant, si vous employez la classe Redirect, la fonction __autoload() inclura le fichier Redirect.php, qui semble être un lien vers Template.php, qui détient à son tour la classe Redirect.

La programmation orientée objet en PHP est une manière naturelle de travailler, surtout avec les améliorations introduites dans la version 5. Quelques caractéristiques sont communes à presque tous les langages orientés objet. Vous avez la chance de définir des classes qui vous permettent de créer des objets avec le comportement que vous voulez. En effet, vous pouvez utiliser des constructeurs afin de contrôler ce qui se passe lorsque l'objet est créé. De plus, vous pouvez employer l'héritage afin de créer des variations sur une classe. Les exceptions permettent une gestion des erreurs considérablement plus flexible et plus lisible. Effectivement, le fait d'être en mesure de manipuler les objets par référence rend la vie beaucoup plus facile en PHP, surtout quand il s'agit de structures difficiles orientées objet. La possibilité d'appeler une méthode sur le résultat d'un appel de méthode est pratique dans les mêmes circonstances. La possibilité d'intercepter les appels de méthode et d'accéder aux variables d'instance nous donne la chance de résoudre divers soucis de façon beaucoup plus élégante. Nous pouvons désormais faire le premier pas vers la programmation orientée aspect, en utilisant la surcharge afin d'insérer du code avant ou après l'ensemble des appels de méthode (ou une sélection d'entre eux) sans devoir tout dupliquer. De plus, nous passons progressivement de la syntaxe de programmation à la conception d'applications. Dans le chapitre suivant, nous observerons certaines des

fonctionnalités de PHP qui servent d'outils pour la conception orientée objet. Il s'agit notamment des restrictions de visibilité, des méthodes de classe, des classes abstraites et des interfaces.

11

CLASSES AVANCÉES

Des haches de pierre aux compagnies aériennes, les objets, réels et tangibles, sont omniprésents dans la technologie. De cette perspective, il n'est pas surprenant que la technologie logicielle en soit venue à dépendre d'objets virtuels. Les cours, en revanche, c'est autre chose. Nommer, classer des éléments en catégories ou classes, est inhérent au langage naturel, mais discuter de catégories de choses et du processus de dénomination est étranger à la technologie physique. En effet, les classes sont nées de la philosophie et des mathématiques, en commençant par les Grecs anciens. Il est important de savoir que la combinaison des deux est extraordinairement puissante. Dans la technologie moderne, la physique et les mathématiques abstraites sont appliquées à l'activité concrète de création de choses. De plus, la programmation orientée objet répète ce schéma. En effet, notez que l'abstraction conceptuelle des classes et l'exploitation directe des objets individuels se rejoignent, créant une forte synergie. En outre, les classes et les objets ont à la fois une expression syntaxique pratique dans le langage et des significations conceptuelles, abstraites et sémantiques. Dans ce chapitre, nous nous concentrerons sur la façon d'utiliser les classes, et tout particulièrement sur les fonctionnalités les plus utiles. Nous débutons par étudier les restrictions de visibilité : la façon d'améliorer l'encapsulation en empêchant que tout ce qui existe à

l'intérieur de la classe soit accessible de l'extérieur. Nous analyserons par la suite la façon d'utiliser la classe comme un conteneur pour les méthodes, les variables et les constantes qui appartiennent à la classe elle-même plutôt que comme une instance d'un objet. Maintenant, nous passons à une autre caractéristique restrictive : les classes et méthodes abstraites, qui peuvent aider à structurer l'héritage des classes. Finalement, nous observerons comment fonctionnent les indications de type de classe, et pour terminer, nous analyserons le fonctionnement et le rôle des interfaces en PHP.

Visibilité

Un principe fondamental du paradigme orienté objet est l'encapsulation. Un objet encapsule des données et des comportements qui sont naturellement liés les uns aux autres. L'action peut avoir lieu à l'intérieur de l'objet sans que le reste du monde s'en occupe. Dans le chapitre précédent, nous avons comparé une classe à une maison. En effet, l'encapsulation, c'est comme avoir de la nourriture dans le réfrigérateur, pour ne pas avoir à sortir à chaque fois que vous voulez manger. Ou, de manière probablement plus appropriée, quand nous planifions, la majorité du temps nous ne devons pas nous préoccuper de ce qui se passe à l'intérieur des murs de la maison. Nous n'avons pas besoin de nourrir la classe de l'extérieur. Si la nourriture est constituée de données stockées dans des variables d'instance, les méthodes de la classe peuvent la consommer sans aucune aide supplémentaire de notre part. Afin de prendre en charge l'encapsulation, plusieurs langages orientés objet ont des fonctionnalités qui nous aident à contrôler la visibilité de ce qui se trouve à l'intérieur de l'objet. Les méthodes et les variables à l'intérieur des objets peuvent être rendues invisibles à l'extérieur de l'objet en les déclarant privées. Une manière un peu moins restrictive de le faire est d'utiliser protected. PHP détient des fonctions et des variables membres privées et protégées. Il s'agit en fait de

méthodes privées et protégées, et non de fonctions,car elles existent constamment à l'intérieur d'une classe, mais la syntaxe pour définir une méthode emploie le mot-clé function. Les méthodes et les variables privées ne sont disponibles qu'au sein d'une même classe. Effectivement, les méthodes et les variables protégées sont disponibles au sein d'une même classe et à partir des classes parent et enfant (ou plus précisément, ancêtre et descendant). De plus, une méthode est marquée comme publique, privée ou protégée en ajoutant un mot clé avant le mot fonction comme ci-dessous :

```
public function getEmail() {}
protected function doLoad() {}
private function matchString($string) {}
```

Les restrictions de visibilité sont employées de façon distincte pour les méthodes et les variables d'instance (et les variables de classe), bien que la syntaxe soit similaire. Dans cette section, nous introduirons en premier lieu les méthodes, et par la suite les variables. Nous analysons pourquoi et comment utiliser les méthodes privées et protégées, ensuite pourquoi il est recommandé de conserver toutes les variables d'instance privées. Nous tentons d'employer l'interception au lieu des méthodes d'accès. Finalement (et ironiquement), nous introduisons le concept de classes et de méthodes finales.

Les fonctionnalités permettant de changer la visibilité sont souvent absentes ou peu visibles dans les langages typés dynamiquement. En effet, c'est logique, car ces langages ont tendance à donner la chance aux programmeurs de faire ce qui leur semble utile sans plusieurs contraintes. D'autre part, la possibilité de contrôler la visibilité des méthodes et des variables d'instance peut être perçue comme une extension naturelle de la possibilité de restreindre la portée d'une variable ordinaire dans le code. Effectivement, restreindre la visibilité des variables d'instance n'est en

général pas un problème, puisque nous pouvons toujours donner une méthode pour y accéder. Toutefois, limiter la visibilité des méthodes comporte le risque de devenir trop restrictif, d'après les personnes qui emploieront ces méthodes. Il peut être tentant de les utiliser afin de s'assurer que votre classe est utilisée d'une façon spécifique, que seule l'API "officielle" d'une classe est employée. Le souci est qu'il est vraiment complexe de savoir à l'avance quelles méthodes seront utiles quand vous, et surtout quelqu'un d'autre, débuterez à réutiliser une classe. Nous emploierons des méthodes privées et protégées dans les exemples de ce livre, mais l'hypothèse de base est qu'une méthode privée ou protégée peut être rendue publique à tout moment. Une manière de considérer les méthodes privées et protégées est de les voir comme une sorte de documentation, une aide à la lisibilité. Elles donnent la chance de voir plus facilement comment les méthodes sont employées et vous empêchent de les utiliser de façon incorrecte, mais elles ne dictent pas nécessairement la façon dont elles doivent être utilisées à l'avenir. La visibilité est légèrement distincte en PHP et en Java. Java possède un concept de paquetage qui affecte la visibilité. Par défaut, une méthode ou une variable d'instance est visible par toute classe du même paquetage. Les méthodes et les variables protégées sont disponibles pour les classes filles et descendantes et pour les autres classes du même paquetage. Cependant, en PHP, la visibilité par défaut est publique, ou plutôt, si nous n'indiquons pas de visibilité, tout est visible publiquement. PHP n'a pas non plus la capacité de Java à rendre les classes privées et protégées.

Nous avons parlé de la raison de l'utilisation de restrictions de visibilité pour les méthodes. En supposant alors que nous souhaitons les utiliser, quand et comment les appliquerions-nous spécifiquement ? Nous traiterons d'abord des méthodes privées, et par la suite des méthodes protégées. Les méthodes privées sont fréquemment des méthodes utilitaires employées à de nombreuses reprises dans une classe (mais pas dans une autre classe) ou

des méthodes employées uniquement à un endroit. Il peut sembler bizarre d'avoir une méthode différente pour quelque chose qui ne se produit qu'à un seul endroit, mais la raison en est généralement la lisibilité : mettre un morceau de code dans une méthode différente qui a un nom révélateur d'intention. Le code illustré ci-dessous est un exemple simplifié de validation de l'utilisateur. Un administrateur a changé un compte d'utilisateur existant et a soumis le formulaire. Si l'administrateur n'a pas changé le nom de l'utilisateur, il met à jour le compte utilisateur existant. L'administrateur est en mesure de créer un nouveau compte utilisateur en changeant le nom de l'utilisateur, mais le nom ne doit pas entrer en conflit avec un compte utilisateur existant, sinon ce compte sera écrasé ou dupliqué. Afin de rendre le code plus lisible, il se trouve une méthode distincte afin de tester chacune de ces situations où la soumission du formulaire est acceptée (nameUnchanged() et nameNotInDB()).

```
class UserValidator {
function validateFullUser($user) {
si ($this-> nameUnchanged($user) ||
$this-> nameNotInDB()) return TRUE ;
retourner FALSE ;
}
private function nameUnchanged($user) {
return $_POST['username'] == $user-> getUsername() ;
}
private function nameNotInDB() {
// Interroge la base de données, retourne TRUE s'il n'y a pas d'utilisateur.
// avec un nom correspondant à $_POST['username'])
}
}
```

$user est un objet utilisateur représentant l'utilisateur existant en cours de modification ; en d'autres mots, l'objet dont les propriétés ont été présentées dans le formulaire.

De plus, les méthodes protégées en PHP sont disponibles au sein d'une même classe et à partir de classes ancestrales ou descendantes, c'est-à-dire lorsque la classe utilisant la méthode hérite de la classe contenant la méthode ou le contraire. De plus, les possibilités d'utilisation de méthodes protégées apparaissent quand une classe enfant emploie une méthode d'une classe parent. Elle est aussi utile dans le cas inverse, quand la classe mère utilise une méthode de la classe fille. Ce point est un peu plus complexe à saisir, mais reste important. Nous analyserons sous peu certains exemples complets.

privé ou protégé ?

Techniquement, les mots-clés private et protected fonctionnent précisément de la même façon avec les méthodes et les variables d'instance. Toutefois, en pratique, il y a une différence entre les variables et les méthodes, car vous êtes en mesure d'utiliser une méthode afin d'obtenir une variable, mais pas l'inverse. Cela veut dire que vous pouvez toujours garder les variables membres privées ou protégées tant que vous fournissez des méthodes afin d'obtenir et de définir la valeur de la variable :

```
class Document {
privé $title ;
//...
//...
function getTitle { return $this->title ; }
function setTitle($arg) { $this->title = $arg }
}
```

De cette manière, vous n'empêchez personne de faire quoi que ce soit, vous contrôlez tout simplement la manière dont il le fait. Voilà pourquoi il n'y a presque jamais de problème à garder les variables membres privées. Et parce que ce n'est pas un souci, c'est en général considéré comme une excellente pratique, du moins dans les langages qui n'ont aucune façon d'intercepter l'accès à une variable d'instance pour que sa signification puisse être changée au besoin. Comme nous l'avons mentionné, une variable membre privée est une variable à laquelle on ne peut accéder que directement depuis la classe. Généralement, l'idéal est que la variable ne soit employée qu'à l'intérieur de la classe. Si vous avez la chance d'éviter de l'utiliser de l'extérieur, c'est un signe que vous suivez le principe "dites, ne demandez pas". De plus, si vous devez accéder à la valeur depuis l'extérieur de la classe, utilisez les accesseurs, également appelés méthodes getter et setter, pour obtenir et définir la valeur. Certains vous diront que c'est la seule manière de procéder : les variables membres ne doivent jamais être publiques. Toutefois, trouver une raison satisfaisante pour laquelle il devrait en être ainsi n'est pas nécessairement simple. Ne pas exposer la variable du tout est une excellente idée, mais quand vous devez l'exposer, pourquoi devez-vous utiliser des accesseurs ? Quelques des raisonnements ne sont pas entièrement convaincants. Certains vous diront, par exemple, que si vous possédez une variable code postal, vous devrez probablement la valider avant de la définir. C'est alors une excellente idée de rendre la variable privée et d'avoir une méthode setter, setZipCode(), qui se charge de la valider au préalable. De cette manière, personne ne peut définir une valeur invalide. Quelque chose comme ce qui suit :

```
classe Adresse {
private zipCode ;
function setZipCode($zip) {
// Effectuer une validation
$this-> zipCode = $zip ;
```

```
}
}
```

Ceci est convaincant pour le cas particulier d'un code postal. Toutefois, souvent, il faut simplement stocker la valeur comme elle est. Donc, pourquoi utiliser des accesseurs quand aucun traitement n'est nécessaire ? Certainement, cela pourrait être nécessaire à l'avenir, mais nous ne le savons pas. Que se passe-t-il alors si nous gardons la variable publique jusqu'à ce qu'un traitement supplémentaire soit nécessaire ? Dans ce cas, toutes les occurrences de la variable doivent être transformées en appels d'accès. De plus, ;e seul souci est que nous ne possédons aucune méthode de savoir où la variable a été utilisée. Il se peut que nous ne les trouvions pas tous et qu'ils apparaissent comme des bogues gênants. Voilà pourquoi il est préférable d'utiliser les accesseurs dès le début, c'est-à-dire à partir du moment où il est réellement nécessaire d'accéder à la variable. Il n'y a aucune raison d'ajouter des fonctions d'accès pour l'ensemble des variables et il n'y a aucune raison d'ajouter une méthode setter pour une variable qui peut être en lecture seule. Généralement, les getters et setters doivent satisfaire aux exigences actuelles, et non à celles d'un futur hypothétique. L'utilisation des accesseurs est aussi vraiment répandue en PHP, vous pouvez traiter une variable comme si elle était privée et vous assurer que tous les accès depuis l'extérieur de la classe passent par les accesseurs. PHP rend la vie un peu plus simple si vous employez des variables publiques et que vous souhaitez par la suite les rendre privées. Une fois que vous avez déclaré la variable privée, PHP renverra un message à chaque fois que vous exécuterez un code qui essaie de l'utiliser depuis l'extérieur de la classe. Toutefois, il y a une autre possibilité. En effet, vous pouvez employer la surcharge afin de transformer ce qui ressemble à un accès à une variable

depuis l'extérieur en un appel d'accès. Alors, par exemple, quand vous exécutez ce qui suit:

$email = $message->text ;

PHP s'exécutera :

$message-> getText() ;

Mot clé final

Le mot clé final vous donne la chance d'empêcher les classes enfants d'étendre une classe en surchargeant une classe ou une méthode. Voici ci-dessous un exemple qui illustre la facilité de la restriction imposée par une classe finale :

```
final class AccessControl { }
class MyAccessControl extends AccessControl { }
```

Cela produit le message d'erreur illustré ci-dessous:

La classe bar ne peut pas hériter de la classe finale (AccessControl)...

Une méthode finale est une méthode qui n'est pas autorisée à être surchargée dans une classe enfant. Une méthode finale pourrait ressembler à ce qui suit :

```
class AccessControl {
public final function encryptPassword(){}
}
```

Cela est maintenant interdit :

```
class MyAccessControl extends AccessControl {
public function encryptPassword() {}
}
```

Quand les classes et méthodes finales sont-elles utiles ? Il n'est pas simple de répondre à cette question. La majorité des livres sur la conception orientée objet ignorent tout simplement la finale. Il se trouve une certaine divergence d'opinion sur cette question. Quelques personnes disent qu'il faut employer final afin d'éviter une mauvaise conception : si vous croyez qu'hériter d'une classe (ou surcharger une méthode) serait une mauvaise idée, rendez-la finale. D'autres se demandent si cela est réalistement possible parce que cela implique de tenter de deviner à l'avance quelles extensions d'une classe seront nécessaires. Les exemples ci-dessus suggèrent qu'il peut y avoir des situations dans lesquelles l'utilisation de la finale serait logique pour des raisons de sécurité. Une utilisation possible et plus spécifique de final se produit quand une méthode ou une classe est marquée comme étant dépréciée. Si une méthode ou une classe ne doit pas être utilisée du tout, il semble raisonnable d'empêcher son utilisation en la surchargeant ou en l'étendant. En Java, final est aussi utilisé avec une signification distincte : afin de définir des constantes de classe. PHP utilise const à la place. Nous avons mentionné des restrictions de visibilité appliquées aux méthodes et aux variables dans les instances d'objets. Toutefois, les méthodes et les variables d'instance peuvent aussi appartenir à la classe elle-même.

Classes sans objets

Une classe donne une maison virtuelle pour les instances de l'objet appartenant à la classe. Il peut aussi stocker des informations indépendantes des instances. Par exemple, si nous possédons une classe de produit et que nous créons des instances de produit à partir d'une table dans une base de données, le nom de la table appartient logiquement à la classe plutôt qu'à une instance spécifique. De plus, vous pouvez avoir la nécessité de faire quelque chose avant même d'avoir créé des instances. Par exemple, il peut être nécessaire de lire les données requises afin de créer une instance à partir d'une base de données. En effet, ce comportement, lu dans la base de données, est lié à la classe, mais ne peut être exécuté par une instance de la classe. De plus, ce comportement peut être intégré dans une méthode de classe : une méthode qui peut être appelée en utilisant uniquement le nom de la classe au lieu d'une variable représentant une instance :

$product = Product::find($productCode) ;

Nous utilisons le double point (: :) quand nous souhaitons accéder à une méthode, une variable ou une constante appartenant à une classe plutôt qu'à un objet. Il y a toujours une alternative. Au lieu d'utiliser les méthodes, les variables et les constantes de la classe, nous pourrions créer une autre classe (ou de nombreuses classes) dont les instances donnent les informations et le comportement appartenant à la classe. Par exemple :

$finder = nouveau ProductFinder ;
$product = $finder-> find($productCode) ;

Cette solution probablement plus flexible, mais elle est aussi plus difficile : il y a une classe supplémentaire et une ligne de code supplémentaire pour le client. Comme dans toute chose, il y a encore des avantages et des inconvénients.

Les méthodes de classe sont des méthodes qui ne sont pas exécutées sur une instance d'objet spécifique. Elles sont définies dans la classe, toutefois, fonctionnent comme de simples fonctions, mais vous devez utiliser le nom de la classe quand vous les appelez. Le mot clé pour les méthodes et les variables de classe est static. Cette terminologie provient de C++ et Java et est d'usage courant. Alors, même si la "méthode de classe" peut être plus appropriée et descriptive, elle est communément nommée "méthode statique". En PHP, la méthode statique typique est définie en utilisant la fonction statique ou la fonction publique statique :

static public function encryptPassword($password) {}

Disons que nous possédons une classe Utilisateur avec une méthode insert() afin d'enregistrer l'objet utilisateur dans la base de données. Il détient aussi une méthode encryptPassword() qui accepte un mot de passe non crypté comme argument et renvoie un mot de passe crypté. Alors, afin de créer un nouvel objet utilisateur et le sauvegarder dans la base de données, vous devez faire ce qui suit :

$user = new User(/* Arguments incluant le nom d'utilisateur, etc. */) ;

$user-> insert() ;

De plus, afin de crypter un mot de passe, vous devez faire ce qui suit :

$password = User::encryptPassword($password) ;

Au sein de la classe User, nous sommes en mesure d'utiliser self afin de faire référence à la classe :

$password = self::encryptPassword($password) ;

Cela a précisément le même effet qu'une simple fonction :

```
$password = encryptPassword($password) ;
```

La fonction elle-même peut être identique, mais dans un cas elle existe dans une définition de classe, dans l'autre non. Si vous avez d'un ensemble de fonctions procédurales qui semblent être liées, vous pouvez les placer dans une classe uniquement à des fins de tri. Ce que vous avez donc est en fait une bibliothèque de fonctions ; le fait que vous employiez le mot clé class pour la définir ne la rend pas très orientée objet, comme vous ne créez pas d'instance d'un quelconque objet. En PHP, vous pouvez déclarer des méthodes de classe en employant le mot clé static ce qui suit :

```
static public function encryptPassword($password) {
retourner md5($password) ;
}
```

Le mot-clé static est similaire à private et protected en ce sens qu'il documente l'utilisation prévue de la méthode et empêche toute mauvaise utilisation involontaire. Si une méthode est définie comme statique, vous ne pouvez rien faire d'utile avec la variable $this. Vous ne devriez alors pas tenter d'exécuter quelque chose comme ça ce qui suit :

```
static public function encryptPassword($password) {
return $this->format(md5($password)) ;
}
```

Si j'avais mentionné quelque chose de similaire, PHP aurait généré une erreur.

Il existe de nombreuses utilisations des méthodes de classe. Parmi les plus courantes, énumérons :

- Méthodes de création et de fabrication
- Méthodes de recherche
- Code de procédure
- Substitution de constantes

De plus, les méthodes de création et les méthodes de fabrique sont des méthodes qui créent et renvoient des instances d'objets. Ils sont souvent employés quand la création ordinaire à l'aide du nouveau mot-clé devient insuffisante. Les méthodes de recherche (aussi nommées finder), qui permettent de trouver un objet dans une base de données ou un autre référentiel, peuvent être considérées comme un cas particulier des méthodes de création, comme elles renvoient une nouvelle instance de l'objet. Dans certains cas, il est possible d'exécuter un fragment de code procédural aussi efficacement qu'avec une méthode orientée objet. Quelques calculs ou conversions simples en sont un exemple ; il est quelquefois essentiel d'inclure du code procédural dans une classe au lieu d'utiliser de simples fonctions. La raison de l'inclusion dans une classe peut être d'éviter les collisions de noms avec d'autres fonctions ou, car elles se trouvent à une classe autrement basée sur des méthodes d'instance. Le fait que les méthodes statiques puissent être utilisées pour toutes ces choses ne prouve pas qu'elles doivent toujours être utilisées. Les manières statiques ont l'avantage de la simplicité, mais sont complexes à remplacer à la volée ; si une méthode appartient à une instance d'objet, elle est potentiellement linkable. Nous pouvons remplacer l'instance de l'objet par une autre pour

modifier le comportement de façon significative sans modifier le code client ou la classe d'origine. Analysons notre exemple précédent :

$finder = nouveau ProductFinder ;
$product = $finder-> find($productCode) ;

Si nous remplaçons l'outil de recherche de produits par une autre classe (par exemple, si nous souhaitons donner les informations sur les produits à partir d'un service Web), l'ancienne classe ProductFinder et la deuxième ligne de l'exemple peuvent rester les mêmes ; l'outil de recherche est enfichable. D'autre part, en employant la méthode statique :

$product = Product::find($productCode) ;

Dans ce cas, le comportement est intégré dans la classe Produit et ne peut être modifié sans changer cette ligne de code. Ce n'est pas un énorme souci si cela ne se produit qu'une fois, mais si le nom de la classe est utilisé à de nombreuses reprises, c'est une autre affaire. Ce souci peut devenir en particulièr important dans les tests unitaires : nous sommes en mesure de vouloir remplacer la méthode find() par une autre, convenablement changée, qui renvoie des données de test fixes au lieu des données réelles de la base de données.

Les variables de classe sont des variables qui appartiennent à une classe. Afin d'en définir un, vous employez le mot clé static comme pour les méthodes de classe :

classe Personne {
static private $DBTABLE = 'Persons' ;
}

Le nom de la table est désormais disponible pour toutes les instances et est toujours le même pour toutes les instances. Nous sommes en mesure d'y accéder en employant le mot clé self :

$select = "SELECT * FROM ".self::$DBTABLE ;

Dans cet exemple, nous avons déclaré la variable privée, pour qu'elle ne soit pas accessible depuis l'extérieur de la classe. Mais si nous le rendons public, nous pouvons y faire référence de cette façon :

$select = "SELECT * FROM ".Person::$DBTABLE ;

Mais quand est-il approprié d'utiliser une variable de classe ? Dans ce cas spécifique, nous aurions pu utiliser une constante de classe à la place. Ou nous aurions pu utiliser une variable d'instance, initialisée de la même façon. De cette manière, toutes les instances auraient eu le nom de la table disponible. Nous aurions quand même pu l'employer dans les méthodes d'instance de la classe :

$select = "SELECT * FROM ".$this->$DBTABLE ;

Mais elle ne serait pas disponible pour les méthodes de la classe et ne serait pas disponible en dehors de la classe sans avoir préalablement créé une instance de la classe Personne. Tout cela veut dire que l'une des utilisations typiques des variables et des constantes de classe est ce type de données : noms de tables, fragments SQL, autres types de syntaxe (expressions régulières, format printf(), format strftime(), et ainsi de suite). Une autre manière de procéder est de considérer le fait qu'avoir de nombreuses variables globales dans un programme est toujours une mauvaise idée. Si elles sont présentes, une méthode simple d'améliorer la situation (pas

nécessairement idéal, mais tout est relatif) est tout simplement de les rassembler dans une ou plusieurs classes en les remplaçant par des variables de classe publiques. Alors, pour une Config analysez ce qui suit :

```
classe Config {
public static $DBPASSWORD = 'secret' ;
public static $DBUSER = 'developer' ;
public static $DBHOST = 'localhost' ;
//...
}
```

Nous pouvons désormais nous connecter à un serveur MySQL de cette façon :

```
mysqli_connect(Config::$DBHOST,Config::$DBUSER,
Config::$DBPASSWORD) ;
```

J'ai délibérément mis une majuscule aux noms des variables afin de souligner leur similitude avec les variables et constantes globales.

Les constantes de classe sont similaires aux variables de classe, mais il se trouve certaines différences cruciales :

- Comme leur nom l'indique, ils ne peuvent pas être changés
- Ils sont toujours publics
- Il y a des restrictions sur ce que vous pouvez mettre dedans.
- Même si la manière dont vous les utilisez est similaire, la manière dont vous les définissez est complètement différente.

Au lieu du mot clé static, les constantes de classe sont définies à l'aide du mot clé const :

```
classe Personne {
const DBTABLE = 'Persons' ;
}
```

Nous pouvons désormais accéder à la constante en utilisant self::DBTABLE à l'intérieur de la classe et Person::DBTABLE à l'extérieur. Dans ce cas, la constante peut sembler avoir tous les avantages d'une variable. Le nom de la table ne modifiera pas lors de l'exécution du programme ; il ne semble alors pas y avoir de raison d'utiliser une variable. De plus, les constantes ne peuvent pas être écrasées accidentellement. Toutefois, il y a une raison pour laquelle nous pouvons souhaiter utiliser une variable de toute façon : les tests. Nous pourrions souhaiter employer un tableau de test pour le test ; remplacer la variable de classe au début du test est une méthode facile de le faire. D'autre part, le fait qu'une constante ne puisse pas être changée peut être excellent pour la sécurité parce qu'elle ne sera jamais changée à des fins malveillantes.

Les constantes de classe sont tout particulier utiles pour les énumérations. Si une variable ne peut avoir qu'un ensemble fixe de valeurs, vous êtes en mesure de coder toutes les valeurs fixes en tant que constantes et faire en sorte que la variable prenne toujours l'une d'entre elles. Analysons l'exemple d'un système d'autorisation vraiment simple. Le système d'autorisation comporte trois rôles fixes ou catégories d'utilisateurs : utilisateur, webmaster et administrateur. Nous pourrions représenter les rôles comme de faciles chaînes de caractères. Nous aurions une variable $role dont la valeur pourrait être 'utilisateur', 'webmaster' ou 'administrateur'. Par la suite, afin de vérifier que l'utilisateur actuel a des privilèges d'administrateur, nous pourrions faire quelque chose comme ce qui suit :

```
< ? php if ($role == 'amdinistrator') : ?>
```

Modifier
< ? php **endif** ; ?>

Le seul souci est que le mot "administrateur" est mal orthographié et que le test ne fonctionne pas. Il s'agit d'un bogue, mais il peut être évité en utilisant des constantes. En PHP, il existe une manière plus propre de faire cela, nommé constantes de classe :

```
class Role {
const REGULAR = 1 ;
const WEBMASTER = 2 ;
const ADMINISTRATOR = 3 ;
//...
}
```

Désormais, nous pouvons exécuter ceci à la place :

< ? php **if** ($role == Role::ADMINISTRATOR) : ?>

Nous ne nous en tirerons pas non plus avec des fautes d'orthographe ici ; l'utilisation d'une constante de classe non définie est une erreur irréversible. Toutefois, par rapport aux constantes globales, c'est plus simple à comprendre, notamment car nous savons où la constante est définie (dans la classe Role), il nous suffit de la regarder. Toutefois, utiliser des constantes de classe depuis l'extérieur d'une classe n'est pas nécessairement la meilleure manière de procéder. Laisser la tâche de tester le rôle à un objet pourrait être la meilleure chose à faire.

< ? php **if** ($role-> isAdministrator()) : ?>

Cela donne la chance de cacher d'autres informations au code client. C'est un exemple d'un principe nommé "dire, ne pas demander". Généralement, il est préférable de laisser un objet travailler sur ses propres données au lieu de demander les données et de les traiter. Dans le deuxième exemple, nous avons utilisé la constante en dehors de la classe Role. Si nous devions l'utiliser à l'intérieur de la classe afin de décider du comportement de l'objet, nous pourrions débuter à envisager une autre option : utiliser l'héritage pour différencier le comportement des différentes classes. Nous aurions donc des sous-classes de rôle qui pourraient s'appeler AdministratorRole, WebmasterRole et UserRole.

De plus, les constantes de classe conviennent tant qu'elles sont prêtes à faire ce dont nous avons besoin, mais leurs limites tendent à devenir évidentes dès le début. La valeur d'une constante ne peut être fixée que quand elle est définie et ne peut être définie dans les méthodes d'une classe. Vous ne pouvez attribuer que des valeurs faciles à une constante ; il n'y a aucune façon de lui attribuer un objet. En fait, vous n'êtes même pas en mesure d'utiliser la concaténation de chaînes de caractères quand vous définissez une constante. Comme pour la majorité des limitations syntaxiques, il est toujours possible que celles-ci soient changées au moment où vous lirez ce document. Et comme mentionné, il n'y a aucune méthode de remplacer la constante à des fins de test. Pour tout l'ensemble des raisons, nous devons savoir ce qu'il faut faire quand nous devons remplacer une constante de classe.

Le remplacement le plus facile et le plus évident d'une constante de classe est une variable de classe, en général publique. Comme les variables peuvent être changées après avoir été définies, nous pouvons le faire à l'intérieur d'une méthode ou d'une fonction, ce qui nous offre la possibilité de lui attribuer un objet ou le résultat de tout type de traitement. Toutefois, faire en sorte que cela se produise est légèrement compliqué. Nous pouvons le faire dans le constructeur de l'objet, mais donc la variable ne sera pas

disponible avant que nous ayons créé la première instance de la classe. Certainement, si possible, nous pourrions en créer un tout de suite après la déclaration de la classe. Ou plus simplement, nous pourrions avoir une méthode de classe afin d'initialiser les variables de la classe et l'exécuter. Si nous utilisons deux bases de données MySQL différentes, rbac et cms, nous pourrions rendre une connexion disponible pour chacune d'entre elles de cette façon :

```
class Connections {
public statique $RBAC ;
public static $CMS ;
public function init() {
self::$RBAC =
nouveau mysqli('localhost', 'user', 'password', 'rbac') ;
self::$CMS =
nouveau mysqli('localhost', 'user', 'password', 'cms') ;
}
}
Connexions::init() ;
```

C'est probablement laid et inélégant, mais au moins ça marche. Mais désormais, nous pouvons aussi rendre les variables privées et ajouter des méthodes d'accès statiques :

```
public static function getRbac() { return self::$RBAC ; }
public static function getCms() { return self::$CMS ; }
```

Une méthode de classe en lecture seule est souvent un substitut parfaitement valable pour une constante. De plus, ils peuvent avoir l'air

pratiquement similaires. Vous pouvez remplacer Person::DBTABLE par Person::DBTABLE() :

```
public static function DBTABLE() { return 'Persons' ; }
```

Il est facile et fonctionne même dans les anciennes versions de PHP. Dans le cadre d'une méthode, nous n'avons pas de limites à ce que nous pouvons faire. Par exemple, si nous souhaitons réutiliser une longue instruction SQL qui peut être plus facilement formatée en utilisant la concaténation, nous pouvons le faire comme ce qui suit :

```
class UserMapper {
public static function sqlSelect($id) {
"SELECT user_id,email,password,firstname,lastname,".
"nom d'utilisateur,rôle+0 comme roleID FROM Users WHERE id = $id" ;
}
}
```

Les variables de classe et les constantes ont été introduites en PHP 5, observons-les en détail.

12

CLASSES ET INTERFACES ABSTRAITES

Les classes abstraites, qui sont une autre caractéristique de PHP, sont composées d'un aspect conceptuel et d'un aspect pratique. Comme nous privilégions l'aspect pratique, analysons ce que fait réellement une classe abstraite. Nous allons analyser le fonctionnement de base des classes et des méthodes abstraites, et par la suite nous observerons la façon les appliquer à une classe de l'exemple "Hello world" du chapitre précédent. Rendre une classe abstraite est simple, il suffit d'utiliser abstract class au lieu de juste class. Après avoir fait cela, nous ne sommes plus autorisés à créer une instance de la classe. Alors, tu ne devrais pas le faire :

classe abstraite Foo {}
$foo = nouveau Foo ;

Dans ce cas, vous recevrez ce message illustré ci-dessous :

Impossible d'instancier la classe abstraite Foo

Donc, quel est l'intérêt d'avoir une classe abstraite ? Elle est utile, car une autre classe, qui n'est pas abstraite, cela veut dire une classe concrète, peut

en hériter. Une méthode abstraite n'est en fait que la déclaration d'une signature de méthode qui peut être utilisée par des classes filles.

Fonction abstraite protégée createAdapter(DomElement $element) ;

Cette prétendue méthode ne fait rien ; elle prétend tout simplement être essentielle. Il ne s'agit en fait que de la signature d'une méthode, mais nous l'appelons malgré tout une méthode. Techniquement, la relation entre les méthodes abstraites et les classes abstraites est que si une méthode abstraite est déclarée, la classe qui la détient doit aussi être déclarée abstraite. En d'autres mots, une classe concrète ne peut pas avoir de méthodes abstraites, mais une classe abstraite peut avoir des méthodes concrètes. Dans notre exemple d'héritage du chapitre précédent, nous avions une classe HtmlDocument et une classe enfant 'Hello world'. Si vous employez la classe HtmlDocument seule, une page HTML vide sera renvoyée. Il n'y a donc guère d'intérêt à l'utiliser, sauf indirectement via ses enfants. En d'autres mots, il n'y a aucun mal à la déclarer abstraite. Pendant que nous y sommes, nous pourrions aussi déclarer la méthode getContent() abstraite.

```
abstract class HtmlDocument {
public function getHtml() {
return "<html><body> ".$this-> getContent().
"</body></html> " ;
}
Fonction publique abstraite getContent() ;
}
```

La méthode abstraite nécessite simplement les classes enfant à implémenter la méthode getContent(). Nous n'avions pas besoin du comportement

précédent de cette méthode – renvoyer une chaîne vide – comme tout ce que nous avons est un document HTML vide, qui n'est pas la chose la plus excitante que nous puissions présenter dans un navigateur Web. Au lieu de cela, nous attendons de toutes les classes filles qu'elles implémentent cette méthode avec une méthode qui renvoie du texte.

Qu'avons-nous obtenu en faisant cela ? Nous avons évité deux erreurs possibles : créer une instance de la classe HtmlDocument elle-même et oublier d'implémenter la méthode getContent() dans une classe enfant. En outre, nous avons rendu le code un peu plus clair. Le mot clé abstract class affiche à quelqu'un qui le lit pour la première fois qu'il se trouve des classes enfant de HtmlDocument. L'intention de la classe, à savoir être une classe parentale sans travail indépendant à effectuer, est plus précise. L'un des objectifs des classes abstraites est de prendre en charge les suggestions de type qui ne sont pas vraiment spécifiques, mais proposons ci-dessous un exemple afin de clarifier.

Si vous commandez une pizza et qu'on vous livre une encyclopédie, vous restez sur votre faim. Pire, c'est un signe que vous avez un mauvais numéro de téléphone et qu'il n'y a pas eu d'excellente communication avec les personnes qui ont pris votre commande. Ou probablement que le coursier avait la mauvaise adresse. De même, si une méthode ou bien une fonction reçoit le mauvais type d'entrée, c'est fréquemment le signe d'un bogue grave. Une fonction qui effectue des calculs mathématiques ne fonctionnera pas avec des chaînes de caractères ou des objets PEAR DB. Et si vous lui transmettez des objets PEAR DB, vous avez très probablement commis une erreur.

De plus, les langages typés statiques vous obligent à spécifier le type de chaque argument pour chaque méthode. Compte tenu du fait que les soucis liés aux valeurs d'entrée sont relativement rares, cela peut sembler excessif. D'autre part, un seul bogue peut causer un nombre considérable de problèmes. Par conséquent, plusieurs programmeurs ont tenté de mettre en

œuvre diverses solutions alternatives pour la vérification de type dans les langages typés dynamiquement. Toutefois, elles ont tendance à être incomplètes et lourdes. Observons en premier lieu comment fonctionnent les indications de type, et par la suite, dans quels cas elles sont utiles. PHP a une solution incomplète, mais pas encombrante. PHP peut contrôler les types d'arguments de méthode pour vous, afin que vous n'ayez pas à écrire de code conditionnel explicite dans ce but. Notez qu'au moment de la rédaction de cet article, cela n'est vrai que si les arguments sont des objets ou bien des tableaux. Comme nous l'avons mentionné, les langages à typage statique comme Java exigent que vous spécifiiez le type de chaque argument dans une méthode. Une méthode Java débutera comme ce qui suit:

public void addDocument**(**Document Document**) {**

Dans ce cas, void est le type retourné, ce qui veut dire que la méthode ne retournera rien, et Document veut dire que l'argument unique doit être un objet Document ou un objet appartenant à une sous-classe de Document. PHP vous donne la chance de faire quelque chose de similaire avec tout argument qui est un objet ou un tableau. Alors, vous pouvez écrire ce qui suit :

public function addDocument(Document $document) {}

Contrairement à Java, PHP ne vous montrera pas que vous avez commis une erreur avant que vous n'exécutiez le code. Ce qui précède est équivalent à ce qui suit :

```
public function addDocument($document) {
if ( !($document instanceof Document)) {
die('Argument 1 doit être une instance de Document') ;
```

```
}
}
```

Ainsi, si NewsArticle est une sous-classe de Document, vous serez en mesure de passer un objet NewsArticle à la méthode, mais pas une simple chaîne ou un objet User. L'excellente nouvelle avec l'indice de type est que vous recevez un avertissement à l'avance si vous avez commis une erreur. En effet, sans l'indication du type, vous ne recevrez pas de message d'erreur jusqu'à ce que vous tentiez d'utiliser l'objet ou la valeur de manière inappropriée, par exemple en appelant la mauvaise méthode. Assurez-vous que les arguments des méthodes sont valides peut augmenter la fiabilité et faciliter le débogage en détectant les erreurs à l'avance. Toutefois, il s'agit d'une arme à double tranchant : si vous ajoutez du code de vérification des erreurs là où il n'est pas requis, il encombrera votre code et le rendra moins lisible. Cela peut à son tour rendre votre code moins fiable parce qu'il devient plus complexe de trouver les bogues et les failles de sécurité. C'est pourquoi, généralement, il vaut mieux éviter un trop grand nombre de tests, sauf s'il s'agit d'une interface dont vous savez qu'elle pourrait être mal utilisée. Une excellente couverture de test réduit le besoin de vérification. D'autre part, si vous écrivez une classe qui doit être utilisée par des personnes que vous ne connaissez probablement pas, la vérification des types d'arguments est plus que nécessaire.

De plus, les indications de type impliquent peu de code et améliorent la lisibilité au lieu de la diminuer. Dans un code bien structuré, orienté objet, avec plusieurs petites classes, la chose la plus complexe à comprendre peut être l'interaction entre les classes, et le fait de connaître les types d'arguments donne la chance de démêler plus simplement les relations (l'alternative est d'utiliser des commentaires). Cependant, le véritable inconvénient des suggestions de type est la dépendance. Chaque indication

de type est une dépendance de la classe ou de l'interface à laquelle l'indication fait référence. Par conséquent, même si les indications de type peuvent simplifier la recherche de bogues, elles rendent aussi plus complexe la modification du code. En effet, si vous changez le nom d'une classe, il se peut que vous deviez changer l'ensemble des indications de type pour cette classe. Par exemple, nous pouvons avoir une classe appelée Date et nous retrouver dans la situation illustrée ci-dessous :

$this-> setDate(Date $date) ;

Désormais, si nous modifions le nom de la classe Date en DateMidnight, nous devrons changer l'ensemble des indications de type. Les programmeurs qui ont l'habitude de gérer ce genre de choses en Java vous diront probablement qu'il faut taper des indices sur une interface afin d'éviter cela, mais trouver l'interface dont vous avez besoin est loin d'être trivial. Il est important de savoir que les indications de type sont plus susceptibles d'être utiles dans les constructeurs que dans les autres méthodes. Souvent, un constructeur accepte un objet comme argument, stocke l'objet dans une variable d'instance et l'utilise ultérieurement. Plusieurs choses peuvent se produire entre le moment où l'objet est inséré et celui où il est utilisé ; en vérifiant le type quand il est transmis au constructeur, nous pouvons détecter une erreur éventuelle beaucoup plus tôt et il peut être plus simple de trouver le bogue. Les indications de type en PHP ne vous permettent de vérifier que les objets et les tableaux, et non les types de données simples. À moins que vous n'écriviez du code orienté objet plutôt sophistiqué, la majorité des arguments de méthode sont susceptibles d'être de simples chaînes de caractères et des nombres. D'autre part, nous serions en mesure d'utiliser les indications de type pour les types de données simples en enveloppant (avec wrapping) l'élément d'une classe. Nous serions aussi en mesure de créer des classes String et Integer qui n'ont

d’autre but que d’indiquer le type du contenu. Il serait peut-être plus utile d’introduire une classe spécialisée dont le nom afficherait la signification des données. Supposons une méthode query() typique pour un objet de connexion à une base de données. Nous pourrions l’utiliser de la manière suivante :

```
$db->query('SELECT * FROM Log') ;
```

Si nous souhaitons faire quelque chose afin d’empêcher la possibilité de passer une chaîne ou un nombre non pertinent à cette méthode, nous serions en mesure d’introduire une indication de type nécessitant un objet SqlStatement :

```
class DbConnection {
public function query(SqlStatement $sql) {}
}
```

L’objet SqlStatement peut être créé à partir de la chaîne de caractères d’une façon facile :

```
$db->query(new SqlStatement('SELECT * FROM Log')) ;
```

Dans sa forme la plus facile, l’objet SqlStatement ne serait qu’une simple enveloppe pour la chaîne de caractères. De plus, afin qu’elle soit utile dans plus que ce cas spécifique, nous pourrions avoir une classe enveloppe générale et l’étendre afin de créer la classe SqlStatement :

```
abstract class StringHolder {
Protégé $string ;
public function __construct($string) {
$this->chaîne = $chaîne ;
```

```
}
public function getString() {
retourner $this->string ;
}
}
classe SqlStatement extends StringHolder {}
```

Il est important de noter que je ne préconise pas cette approche, j'en souligne tout simplement la possibilité. Il est clair que cela serait excessif en ce qui concerne un usage général, mais il peut y avoir des circonstances dans lesquelles la vérification du type est suffisamment essentielle pour que ce type particulier de complexité en vaille la peine. L'utilisation d'indications de type peut être problématique si vous êtes contraint d'employer un nom de classe spécifique parce que changer le nom de la classe nécessite de changer l'ensemble des indications de type. Une manière d'obtenir plus de liberté dans l'utilisation des indications de type est d'employer des interfaces.

Interfaces

Le mot interface a une signification sémantique et, dans quelques langages de programmation, également syntaxique. Dans la programmation orientée objet, le mot interface affiche en général l'ensemble des messages auxquels un objet est en mesure de répondre, l'ensemble des opérations qu'il peut effectuer. C'est le langage vraiment restreint et limité que l'objet comprend. D'une perspective syntaxique, les interfaces sont une façon de déclarer formellement ce petit langage. De plus, vous êtes en mesure de définir quelles classes d'objets répondent à ce petit langage particulier. Si vous ne donnez pas aux objets désignés la capacité de comprendre le langage, le

compilateur renverra des erreurs. Mais quelle est leur utilité réelle et à quelles fins ? Analysons. Nous allons débuter par analyser le fonctionnement des interfaces et parler de leur nécessité en PHP. Par la suite, nous étudierons certaines utilisations des interfaces : rendre la conception plus claire et améliorer les indications de type. Finalement, nous observerons comment les interfaces en PHP sont distinctes des interfaces en Java.

Techniquement, une interface est une construction similaire à une classe qui déclare un certain nombre de méthodes. En fait, elle est similaire à une classe abstraite qui ne détient que des méthodes abstraites :

interface Template {
public function __construct($path) ;
Fonction publique execute() ;
Fonction publique set($name,$value) ;
fonction publique getContext() ;
}

Une classe basée sur cette interface peut être déclarée à l'aide du mot clé implements, analysez ce qui suit :

classe SmartyTemplate implémente Template {}

Cela veut dire que la classe SmartyTemplate doit contenir toutes les méthodes de l'interface et que les signatures des méthodes doivent être identiques. Donc, set() doit avoir deux arguments. La distinction entre cette classe et une classe abstraite qui ne contient que des méthodes abstraites est qu'une classe ne peut qu'étendre une autre classe, mais peut implémenter plus d'une interface :

classe DateRange extends Range implements TransposableRange, ComparableRange {}

Avons-nous aussi besoin d'interfaces en PHP, ou ne sont-elles que des formalités inutiles et prétentieuses qui nuisent aux performances ? Ont-ils une valeur concrète et pratique ? Analysons en premier lieu la question des performances. Les interfaces n'affecteront pas énormément les performances, à moins que nous ne placions l'interface dans un fichier séparé. L'ouverture d'un fichier prend toujours du temps. Nous devons alors être conscients de cela. Ce que font les interfaces, dans un sens pratique, ce n'est presque rien. En fait, notez que si nous n'utilisons pas les indices de type de classe et que notre code est correct, les interfaces n'ont aucun effet sur l'exécution du code. Les interfaces sont principalement une façon de rendre les choses explicites et d'éviter certaines erreurs. Cependant, même si les interfaces ne sont pas strictement obligées, elles peuvent avoir une certaine valeur. Ils rendent certaines de nos conceptions plus explicites et peuvent éviter des erreurs triviales. Mais à l'heure où nous écrivons ces lignes, les interfaces en PHP n'existent que depuis trop peu de temps afin que nous puissions porter un jugement définitif sur leur utilité et leur inutilité. Ce que nous pouvons faire, c'est explorer quelques avantages et inconvénients possibles, quelques points de vue et quelques pièges.

De plus, une interface est pratiquement la même chose qu'une classe abstraite sans méthodes implémentées, mais une classe ne peut avoir qu'une seule classe parente, mais implémenter un nombre quelconque d'interfaces. Les langages typés dynamiquement n'ont traditionnellement pas de classes et d'interfaces abstraites. Cela a un sens pratique, mais d'une perspective conceptuelle, cela laisse à désirer. Environ tous les langages de programmation modernes ont des classes et d'héritage. Vous pouvez alors posséder des classes qui ont une partie de leur implémentation en commun et qui héritent de leur parent. En général, ces classes ont à la fois cette relation pratique, l'implémentation commune, et une relation conceptuelle. Par exemple, si nous avons une classe Document qui est le parent de la classe Message et de la classe NewsArticle, celles-ci sont précisément liées

sur le plan conceptuel. Si l'on supprime la mise en œuvre commune (il existe en général différentes façons de procéder), la relation conceptuelle demeure. Il semble raisonnable qu'un langage nous permette d'exprimer cette relation même s'il n'existe pas d'implémentation commune. Si des classes abstraites, même sans méthodes concrètes, et des interfaces sont utilisées afin d'exprimer les similitudes entre les classes et la manière dont elles fonctionnent, cela rend ces aspects conceptuels ou de conception explicites et cela peut simplifier la compréhension de la manière dont le code est structuré. Comme pour les classes mères, les interfaces peuvent être utilisées afin de donner des indications sur le type de classe. Cela veut dire que la méthode suivante acceptera un objet appartenant à une classe qui implémente l'interface Template ou qui étend une classe appelée Template :

```
public function display(Template $template) {}
```

Si une suggestion de type était trop spécifique, elle pourrait être trop restrictive. Par exemple, la méthode display() pourrait être en mesure d'utiliser des objets qui ne sont pas des modèles. Par exemple, il pourrait être en mesure d'accepter un objet Redirect qui effectuerait une redirection HTTP plutôt que de générer du code HTML comme le font habituellement les modèles. Il peut être utile de disposer d'un genre de liberté quant au caractère restrictif de la suggestion de type. Une classe ne peut pas étendre plus d'une classe, mais les interfaces sont illimitées ; cela nous donne la chance d'exprimer des relations abstraites en dehors des hiérarchies d'héritage. Afin d'exprimer la similitude entre un objet Redirect et un objet Template, le fait qu'ils sont interchangeables dans quelques contextes, nous avons la chance d'utiliser une interface nommée quelque chose comme Response :

```
interface Response {
Fonction publique execute() ;
```

Fonction publique display() ;

}

De plus, une classe de modèle spécifique pourrait étendre une classe de modèle et mettre en œuvre l'interface Response :

classe FormTemplate extends Template implements Response {}

En PHP, nous avons énormément de contrôle sur le degré de restriction des indications de type. À un extrême, nous pouvons tout simplement les laisser de côté et, à moins que l'objet que nous passons à la méthode ne soit incompatible, nous n'aurons aucun souci. À l'autre extrême, nous pouvons baser l'indication de type sur une interface ou une classe parente qui a les mêmes indications de type. Notez que la manière la moins restrictive de les utiliser sans les ignorer complètement est d'employer une interface sans méthodes, une interface qui communique un certain aspect ou une certaine qualité d'un type de données. Nous aurions pu définir l'interface Response de la façon illustrée ci-dessous :

interface Response {}

En programmation Java, ce type d'interface est parfois nommé interface de balises. Toutefois, il se trouve une distinction essentielle entre Java et PHP : quand vous nommez une interface comme type d'argument d'une méthode en Java, le compilateur ne vous permettra pas d'appeler des méthodes sur cet objet qui ne sont pas définies dans l'interface. En effet, l'indice de type PHP n'est qu'une simple vérification que l'objet est bien du bon type ; vous êtes en mesure d'appeler n'importe quelle méthode, aussi inappropriée soit-elle :

public function display(Template $template) {

$template-> nonExistentMethod() ;

```
}
```

Certainement, cette méthode échouera dès que PHP tentera d'exécuter nonExistentMethod(), mais l'indice de type n'a rien à voir avec cela. Il n'y a pas de relation directe entre l'indication de type et l'appel de méthode. Il n'y en a qu'une faible, indirecte, comme nous pouvons être certains d'éviter ce type d'erreur si nous faisons attention à ne pas utiliser de méthodes qui ne sont pas définies dans l'interface Template. De toute manière, comme il n'y a pas de contrôle de ce type en PHP, nous pouvons toujours utiliser une interface de balises plutôt qu'une "vraie" interface.

Une distinction notable entre les interfaces Java et PHP a déjà été discutée : les indications de type PHP ne limitent pas les méthodes qui peuvent être appelées. En dehors de cela, il existe certaines différences mineures dans la manière dont les interfaces fonctionnent en Java et en PHP. Une différence est qu'une classe en PHP ne peut pas implémenter deux interfaces différentes si elles détiennent la même méthode. Java n'a pas cette restriction. De plus, contrairement à Java, PHP permet d'inclure le constructeur dans l'interface. Il n'est en général pas utile de définir le constructeur comme faisant partie de l'interface, comme la signature du constructeur est fréquemment ce qui doit varier entre des classes similaires. Mais il est essentiel de savoir que vous pouvez le faire et qu'il peut y avoir des situations où cela serait utile. Il existe aussi certaines similitudes qui ne sont pas forcément évidentes. Effectivement, en Java et en PHP, une interface peut étendre une autre interface en employant le mot clé extends. Et dans les deux cas, vous pouvez ajouter le mot clé abstract aux méthodes de l'interface, mais ce n'est pas requis, comme toutes les déclarations de méthodes dans une interface sont de toute manière abstraites.

Comme il y a tant de concepts, faisons un petit résumé. La programmation orientée objet de base prévoit une organisation aux deux

niveaux suivants : des classes et des méthodes dans lesquelles peuvent coexister des données et divers blocs de code. L'héritage est une façon simple pour les classes de partager du code. Les restrictions de visibilité définissent ce qui est visible en dehors de la classe et ce qui ne l'est pas. De plus, les méthodes et variables privées et protégées contribuent à rendre le code orienté objet plus lisible et à encapsuler le contenu d'un objet. Les interfaces, les classes abstraites et les indices de type ne sont pas strictement requis en PHP, mais ils peuvent nous fournir plus d'espace afin d'exprimer les types et les abstractions. Ces caractéristiques sont destinées à faciliter le développement de structures et de conceptions complexes.

De plus, les mérites relatifs du code orienté objet par rapport au code procédural sont quelquefois débattus même par les gourous de la programmation orientée objet. La programmation orientée objet est une bonne idée, mais pas toujours. Sauf dans quelques cas, il n'y a aucune raison de rendre "Hello World" plus difficile que ce script PHP de base :

echo 'Hello world !

Il est important de savoir qu'en Java, vous devez écrire une classe afin de faire quelque chose d'aussi facile que d'imprimer "Hello world" ; en PHP, vous êtes libre d'ignorer la programmation orientée objet aussi longtemps que vous le voulez. En pratique, vous pouvez aussi procéder de la sorte en Java et dans des langages similaires, comme vous êtes libre d'écrire une classe qui ne détient que des blocs de code procédural, ou même une longue méthode procédurale. Il n'est pas orienté objet, dans quelque sens que ce soit, et est en fait l'équivalent de simples scripts et fonctions PHP. Le code supplémentaire requis à l'écriture d'une classe juste pour produire "Hello world" est gaspillé. En effet, c'est un bagage inutile. Mais l'orientation objet n'a pas été inventée afin de résoudre des problèmes faciles ; son utilité est de simplifier la compréhension et la résolution de problèmes difficiles.

Quelque part entre l’exemple du "Hello world" et l’application commerciale au sens large, il se trouve un seuil à partir duquel la programmation orientée objet devient plus efficace que la programmation procédurale à long terme. Je mentionne "à long terme", car le principal avantage de la POO est de faciliter la maintenance des applications. Bien qu’elle exige un peu plus d’efforts au tout début, elle peut être remboursée par un travail moindre lors de l’ajout de nouvelles fonctionnalités et de la correction de bogues. J’ai tendance à croire que ce seuil est bas. Je trouve que les cours d’écriture m’aident même quand le programme est relativement petit et facile. La programmation orientée objet donne la chance de décomposer plus simplement le programme en parties et, comme il faut nommer les parties (les classes), il est plus facile d’observer ce qu’elles font. Vous pouvez aussi le faire avec des fonctions simples, mais à mesure que les fonctions se multiplient, il devient plus difficile d’en garder la trace. En fin de compte, vous devez le découvrir par vous-même. Effectivement, si vous estimez que la conception procédurale est plus simple et plus simple à comprendre et que la rendre orientée objet la complique inutilement, restez-en à la conception procédurale. Ne le rendez pas orienté objet juste, car cela semble intéressant.

CONCLUSIONS

Il est sûr que dans ce livre électronique, nous n'avons pas couvert tous les cas et toutes les combinaisons possibles que vous pourriez trouver en programmant en PHP. Nous avons analysé les bases, en partant d'une théorie solide sur laquelle nous avons construit certains exemples afin de simplifier la compréhension. Si vous avez appris l'ensemble des techniques décrites ici, vous êtes en bonne voie pour devenir un excellent développeur PHP et, avec un peu plus d'efforts, vous entrerez dans le niveau avancé.

Ne vous souciez pas si vous avez eu du mal à comprendre, il suffit de relire les chapitres et tout deviendra plus clair. Souvenez-vous que plus vous vous entraînez, plus tout devient simple.

Si vous vous dites : "*Comment puis-je me rappeler de tout cela ?* La réponse est que vous ne devez pas vous souvenir de tout. En effet, nous sommes à l'ère du numérique, donc beaucoup d'entre vous auront eu des doutes sur la manière d'employer une boucle et il est facile de commettre des erreurs si vous êtes un débutant. N'ayez pas honte de vérifier les choses. Ce qui rend la programmation pour le Web simple n'est pas une connaissance encyclopédique des fonctions et des classes PHP, mais une théorie solide du fonctionnement des instructions conditionnelles, des boucles et des autres structures qui contrôlent le flux d'un script.

Quand vous êtes en mesure d’analyser vos projets sur le plan du "si ceci se produit, que doit-il se passer ensuite ?", vous êtes en fait déjà au niveau supérieur et c’est vous qui vérifiez le code. Je consulte souvent le manuel PHP en ligne et pour moi c’est comme un dictionnaire, la majorité du temps, je veux juste vérifier que je me souviens correctement. Il arrive fréquemment que quelque chose attire mon attention et ouvre de nouveaux horizons, probablement une nouvelle fonction qui donne la chance de gagner du temps et de gagner quelques lignes de code. Je vous remercie sincèrement d’avoir lu ce livre électronique et je souhaite que ce ne soit que le point de départ pour construire de magnifiques sites Web, avec de nombreuses fonctions en PHP, en exploitant tout son potentiel.