eVOGUE

Liam TARDIEU

www.evogue.fr

SUPPORT DE COURS / PHP – ORIENTE OBJET

Sommaire

- Sommaire 2
- Présentation 5
  - Historique 5
  - Une autre technique de développement, Pourquoi ? 5
  - A quels besoins repond la programmation orientée objet ? 6
  - Avantages de la programmation orientée objet 7
  - Inconvénients de la programmation orientée objet 8
  - Unified Modeling Language 8
  - Langage php et programmation orientée objet 9
- Classes et Objets 10
  - Qu'est-ce qu'un objet ? 10
  - Qu'est-ce qu'une classe 10
  - Différence entre une classe et un objet 11
  - Application concrete de la classe sur l'objet 12
- Création d'un objet 13
  - Introduction 13
  - Instanciation 13
  - Inférence 13
  - Transformation 14
  - Classe par défaut 14
- Encapsulation 15
  - Définition 15
  - Pourquoi ? 15
  - Concevoir pour réutiliser...comment faire ? 15
  - Intérêt de l'encapsulation 16
  - Niveau de visibilité 17
- Manipulation d'objets 18
  - La pseudo-variable $this 18
  - Accesseur (getter) / Mutateurs (setter) 18
- Manipulation de classes 20
  - Constantes 20
  - Eléments static 20
  - Appartenance à la classe ou a l'objet ? 21
  - Opérateur de résolution de portée 22
  - Différence entre self:: et $this-> 22

- Héritage 23
  - Concept 23
  - Fonctionnement 23
  - Relation et notion d'héritage 23
  - Effectuer un héritage, dans quels cas ? 24
  - Surcharger une méthode 25
- Abstraction 26
  - Définition 26
  - Principe et utilisation 26
  - Classes abstraites 26
  - Methodes abstraites 27
  - Informations 27
- Finalisation 28
  - Définition 28
  - Classes finales 28
  - Methodes finales 28
- Méthodes magiques 29
  - Introduction 29
  - __construct 29
  - __destruct 30
  - __set 30
  - __get 30
  - __call 31
  - __tostring 31
  - __isset 32
  - __sleep 32
  - __wakeup 32
  - __unset 32
  - __invoke 33
  - __setstate 33
  - __clone 33
- Comparaison 35
  - Les opérateurs 35
- Interfaces 36
  - Introduction 36
  - Principe et utilisation 37
  - Différence héritage et implementation 38
- Traits 39
  - Introduction 39
  - Utilisation 39
  - Fonctionnement 39

- Design Pattern 40
  - Introduction 40
  - Type : Création 40
  - Type : Structure 40
  - Type : Comportement 41
  - Présentation du pattern : Singleton 41
- Namespace 44
  - Introduction 44
  - Avantages 44
  - Exemple et Utilisation 45
- Exceptions 46
  - Introduction 46
  - Fonctionnement 47
  - La classe exception 47
  - Déroulement 49
- Php Data Objects 50
  - Introduction 50
  - Requêtes 51
  - Requêtes préparées 52
  - Passage d'arguments 52
  - Constantes pré-définies 53
  - Récupération des données 53
  - Marqueur 54
  - Différence entre PDO et PDOStatement 54
  - La classe pdo 55
  - La classe pdostatement 56
  - Gestion des erreurs 57
  - Choix du connecteur de base de données 58
- Methodes Pratiques 59
  - Divers 59
  - spl_autoload_register 59

➢ Présentation

HISTORIQUE

La notion d'objet a été introduite avec le langage de programmation Simula, créé à Oslo entre 1962 et 1967 dans le but de faciliter la programmation de logiciels de simulation.
Avec ce langage de programmation, les caractéristiques et les comportements des objets à simuler sont décrits dans le code source.
Le langage de programmation orientée objet Smalltalk a été créé par le centre de recherche Xerox en 1972.
La programmation orientée objet est devenue populaire en 1983 avec la sortie du langage de programmation C++, un langage orienté objet, dont l'utilisation ressemble volontairement au populaire langage C.

UNE AUTRE TECHNIQUE DE DEVELOPPEMENT, POURQUOI ?

Au cours des 35 dernières années, les concepteurs de matériel informatique sont passés des machines de la taille d'un hangar à des ordinateurs portables légers basés sur de minuscules microprocesseurs.
Au cours des mêmes années, les développeurs de logiciels sont passés de l'écriture de programmes en assembleur et en COBOL à l'écriture de programmes encore plus grands en C et C++. Nous pourrons parler de progrès (bien que cela soit discutable), mais il est clair que le monde du logiciel ne progresse pas aussi vite que celui du matériel. Qu'ont donc les développeurs de matériel que les développeurs de logiciels n'ont pas ?
La réponse est donnée par les composants. Si les ingénieurs en matériel électronique devaient partir d'un tas de sable à chaque fois qu'ils conçoivent un nouveau dispositif, si leur première étape devait toujours consister à extraire le silicium pour fabriquer des circuits intégrés, ils ne progresseraient pas bien vite. Or, un concepteur de matériel construit toujours un système à partir de composants préparés, chacun chargé d'une fonction particulière et fournissant un ensemble de services à travers des interfaces définies. La tâche des concepteurs de matériel est considérablement simplifiée par le travail de leurs prédécesseurs.
La réutilisation est aussi une voie vers la création de meilleurs logiciels. Aujourd'hui encore, les développeurs de logiciels en sont toujours à partir d'une certaine forme de sable et à suivre les mêmes étapes que les centaines de programmeurs qui les ont précédés. Le résultat est souvent excellent, mais il pourrait être amélioré. La création de nouvelles applications à partir de composants existants, déjà testés, a toutes chances de produire un code plus fiable. De plus, elle peut se révéler nettement plus rapide et plus économique, ce qui n'est pas moins important.