PHP
PROGRAMMATION
ORIENTÉ OBJET

Sommaire

- Introduction
- PHP POO
- Les Exceptions
- PHP PDO

Introduction

INTRODUCTION

C’est quoi la POO ?

La Programmation Orientée Objet (POO) est un paradigme de programmation basé sur les concepts d'objets et de classes.

- **Objet** :

  Représentation d'une chose matérielle ou immatérielle du réel à laquelle on associe des propriétés et des actions.
  Exemple : une voiture, une personne, un animal, un nombre ou bien un compte bancaire peuvent être vus comme des objets.

- **Attributs :**

  Les « attributs » (aussi appelés « données membres ») sont les caractères propres à un objet.
  Une personne, par exemple, possèdent différents attributs qui lui sont propres comme le nom, le prénom, la couleur des yeux, le sexe, la couleur des cheveux, la taille.

- **Méthodes** :

  Les « méthodes » sont les actions applicables à un objet.
  Un objet personne, par exemple, dispose des actions suivantes : manger, dormir, boire, marcher, courir.

INTRODUCTION

C’est quoi la POO ?

- **Les classes :**

  percevoir une classe comme un moule grâce auquel nous allons créer autant d'objets de même type et de même structure qu'on le désire les objets sont bâtis sur des modèles que l’on appelle des classes

- **Instances :**

  Lorsque l'on crée un objet, on réalise ce que l'on appelle une « instance de la classe ».
  C'est à dire que du moule, on en extrait un nouvel objet qui dispose de ses attributs et de ses méthodes.
  L'objet ainsi créé aura pour type le nom de la classe.

PHP POO

PHP POO

Les Classe

```php
class Personne
{
    // Attributs
    public $nom;
    public $prenom;
    public $dateDeNaissance;
    public $genre;

    // Méthodes
    public function __construct($nom, $prenom, $dateDeNaissance, $genre)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
        $this->dateDeNaissance = $dateDeNaissance;
        $this->genre = $genre;
    }

    public function parler($message)
    {
        return "dit" . $message;
    }
}
```

Remarque :

- par convention, on écrit le nom d'une classe en majuscule et en PascalCase»
- **$this** est un mot-clé utilisé à l'intérieur d'une classe pour faire référence à l'instance actuelle de cette classe.

  Il est utilisé pour accéder aux propriétés et méthodes de l'objet en cours de manipulation.

PHP POO

Les Attributs

- les attributs sont les caractéristiques propres d'un objet.
- Toute personne possède un nom, un prénom, une date de naissance, un sexe... Tous ces éléments caractérisent un être humain.
- Il existe trois niveaux de visibilité (public, private et protected) qui peuvent être appliqués à un attribut.
- Le mot-clé public permet de rendre l'attribut accessible depuis l'extérieur de la classe.
- En POO, un attribut est qu'une variable, une fonction est une méthode
- Deux classes différentes peuvent avoir les même attributs et méthode sans risque de conflit.
- une constante doit être déclarée et initialisée avec sa valeur en même temps

PHP POO

Introduction au typage

Le typage des attributs en programmation orientée objet permet de définir le type de données que peut contenir un attribut d'une classe.

Cela contribue à renforcer la robustesse et la fiabilité du code en spécifiant les types de données attendus pour les attributs.

```
class Personne {
    public string $nom; // Attribut nom de type string (chaine de caractères)
    public int $age;    // Attribut age de type int (entier)
}
```

PHP POO

Le constructeur

- Le constructeur est une méthode particulière appelé méthode magique en PHP **__construct**
- C'est elle qui est appelée implicitement à la création de l'objet (**instanciation**).
- Le programmeur est libre de définir des paramètres obligatoires à passer au constructeur ainsi qu'un groupe d'instructions à exécuter à l'instanciation de la classe.
- Il n'y peut y avoir que **un seul constructeur** par classe en PHP
- Les Paramètres du constructeur peuvent également être typé

```php
public function __construct(string $nom, string $prenom, DateTime $dateDeNaissance, string $genre)
    {
        $this->nom = $nom;
        $this->prenom = $prenom;
        $this->dateDeNaissance = $dateDeNaissance;
        $this->genre = $genre;
    }
```

PHP POO

Les méthodes

- Les méthodes sont les actions que l'on peut déclencher sur un objet.
- Il s'agit en fait de fonctions qui peuvent prendre ou non des paramètres et retourner ou non des valeurs / objets.
- Elles se déclarent de la même manière que des fonctions traditionnelles.
- Au même titre que les attributs, on déclare une méthode avec un niveau de visibilité.
- Remarque : deux classes différentes peuvent avoir les mêmes méthodes sans risque de conflit.

```php
public function parler($message)
{
   return "dit" . $message;
}
```

PHP POO

L'instanciation d'une classe

- L'instanciation d'une classe est la phase de création de l'objet.
- Lorsque l'on instancie une classe, on utilise le mot-clé new suivant du nom de la classe.
- Cette instruction appelle la méthode constructeur (__construct()) qui construit l'objet et effectue la réservation en mémoire.

```php
// Instanciation d'objets de la classe Personne
$personne1 = new Personne("Dupont", "Jean", new DateTime('1990-05-15'), "Homme");
$personne2 = new Personne("Durand", "Chantal", new DateTime('1985-08-20'), "Femme");
```

PHP POO

L'encapsulation

L'Encapsulation :

- Consiste à restreindre l'accès aux attributs et méthodes d'une classe, permettant ainsi de contrôler la manière dont ces éléments sont utilisés ou modifiés depuis l'extérieur de la classe, grâce aux opérateurs d'accessibilités.

- Public : Les attributs ou méthodes publics sont accessibles depuis n'importe où .

```php
class Personne {
    public $nom; // Attribut public
}
$personne = new Personne();
$personne->nom = "Dupont"; // Accès direct à l'attribut public depuis l'extérieur
```

- Private : Les attributs ou méthodes privés ne sont accessibles qu'à l'intérieur de la classe.

```php
class Personne {
    private $dateNaissance; // Attribut privé
}
$personne = new Personne();
$personne->dateNaissance = "1990-01-01"; // Erreur : L'attribut privé ne peut être modifié de l'extérieur
```

- Protected : Les attributs ou méthodes protégés sont similaires aux attributs privés, mais ils sont également accessibles dans les sous-classes (classes héritées).

PHP POO

Les Accesseurs et Mutateurs

Les **accesseurs** (**getters**) et **mutateurs** (**setters**) sont des méthodes utilisées pour accéder et modifier les attributs privés d'une classe respectivement. Ces méthodes permettent un contrôle précis sur la lecture et l'écriture des données d'un objet, tout en maintenant l'encapsulation des données.

- **Getters (Accesseurs) :**

  Les getters sont des méthodes publiques permettant d'accéder aux valeurs des attributs privés depuis l'extérieur de la classe. Ils retournent la valeur de l'attribut, mais ne modifient pas directement l'attribut lui-même

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

```php
echo $personne->getNom(); // Accès à l'attribut privé
```

- **Setters (Mutateurs) :**

  Les setters sont des méthodes publiques utilisées pour modifier les valeurs des attributs privés d'une classe. Ils permettent de définir ou de modifier la valeur d'un attribut tout en appliquant des vérifications ou des traitements spécifiques avant l'assignation.

```php
public function setNom(string $nom): void {
    $this->nom = $nom;
}
```

```php
$personne->setNom("Dupont"); // Modification de l'attribut
```

PHP POO

La méthode __toString

La méthode magique **__toString()** est une méthode spéciale en PHP qui est automatiquement appelée lorsque l'objet est utilisé dans un contexte de chaîne de caractères, tel qu'une concaténation avec un autre texte ou lorsqu'il est passé à la fonction echo.

Utilisation de la méthode magique **__toString()** :

```php
class Personne {
    private string $nom;
    private int $age;

    public function __construct(string $nom, int $age) {
        $this->nom = $nom;
        $this->age = $age;
    }

    public function __toString() {
        return "Nom : " . $this->nom . ", Age : " . $this->age;
    }
}

$personne = new Personne("Alice", 25);
echo $personne; // Affichera le résultat de la méthode __toString()
```

PHP POO

L'Héritage

- L'héritage est cette possibilité pour une classe **d'hériter des attributs et méthodes** d'une classe parent.
- L'héritage est une spécialisation.
- L'héritage **simple est supporté** en PHP. L'héritage **multiple non**.
- La classe enfant hérite donc des attributs et méthodes du parent (mais seuls les attributs **public** et **protected** sont accessibles directement à partir des descendants) et possède elle-même ses propres attributs et méthodes.
- L 'héritage ce fait à l'aide du mot clé **extends**

```
class Employe extends Personne {
    private float $salaire;

    public function __construct(string $nom, string $prenom,  DateTime $dateDeNaissance,
string $genre, float $salaire)
    {
        parent::__construct($nom, $prenom, $dateDeNaissance,$genre);
        $this->salaire = $salaire;
    }

    public function afficherSalaire(): float {
        return $this->salaire;
    }
}
```

PHP POO

L'Héritage

- En programmation orientée objet php, **parent::** est un élément clé permettant d'accéder aux méthodes et propriétés de la classe parente à l'intérieur d'une classe enfant .

```
public function __construct(string $nom, string $prenom,  DateTime
                    $dateDeNaissance, string $genre, float $salaire)
{
    parent::__construct($nom, $prenom, $dateDeNaissance,$genre);
    $this->salaire = $salaire;
}
```

PHP POO

L'Héritage

- En programmation orientée objet, **parent::** est un élément clé permettant d'accéder aux méthodes et propriétés de la classe parente à l'intérieur d'une classe enfant .
- Chaque classe doit posséder un constructeur.
- Pour exécuter le constructeur du parent à partir du constructeur de l'enfant :

```
public function __construct(string $nom, string $prenom,  DateTime
                                $dateDeNaissance, string $genre, float $salaire)
{
  parent::__construct($nom, $prenom, $dateDeNaissance,$genre);
  $this->salaire = $salaire;
}
```

PHP POO

Surcharge des Méthodes

La surcharge des méthodes se produit lorsqu'une classe enfant redéfinit une méthode de sa classe parente. Cela permet à la classe enfant de fournir une implémentation spécifique de la méthode, tout en conservant le même nom et la même signature que la méthode de la classe parente.

```php
class Vehicule {
    public function demarrer() {
        echo "Le véhicule démarre.<br>";
    }
}

class Voiture extends Vehicule {
    public function demarrer() {
        echo "La voiture démarre avec un moteur.<br>";
    }
}
```

PHP POO

L'Abstraction

- L'abstraction en programmation orientée objet consiste à définir des méthodes sans fournir leur implémentation dans la classe où elles sont déclarées.

- **Méthodes abstraites** :
  Une méthode abstraite est déclarée sans contenu dans une **classe abstraite**, et elle doit être implémentée dans toutes les sous-classes qui héritent de cette classe abstraite.

- **Classes abstraites :**
  Une classe abstraite est déclarée avec au moins une méthode abstraite. Elle ne peut pas être instanciée directement mais peut être étendue par d'autres classes.

```php
abstract class Forme {
    abstract public function calculerSurface(): float;

    public function afficherDetails() {
        echo "Ceci est une forme abstraite.<br>";
    }
}
```

```php
class Cercle extends Forme {
    private float $rayon;
    //...
    public function calculerSurface(): float {
        return 3.14 * $this->rayon * $this->rayon;
    }
}
```

PHP POO

Classe et méthode final

En PHP, le mot-clé final est utilisé pour restreindre la modification ou l'héritage de certaines entités, telles que les classes, les méthodes ou les propriétés. Lorsqu'une classe est déclarée comme final, cela signifie qu'elle ne peut pas être étendue par d'autres classes.

Une méthode ne pourra pas être surchargé.

```php
final class Animal {
    public function deplacer() {
        echo "L'animal se déplace.<br>";
    }
}
```

```php
class Chien extends Animal { // Erreur : Impossible d'étendre une classe finale
    // ...
}
```

PHP POO

Les interfaces

Les interfaces en PHP sont des contrats définissant des méthodes sans aucune implémentation. Elles spécifient quelles méthodes une classe doit implémenter sans fournir les détails de l'implémentation.

Une classe peut implémenter plusieurs interfaces, garantissant ainsi l'implémentation de toutes les méthodes spécifiées dans ces interfaces.

```php
interface Animal {
    public function deplacer();
    public function faireDuBruit();
}
```

```php
class Chien implements Animal {
    public function deplacer() {
        echo "Le chien se déplace en courant.<br>";
    }

    public function faireDuBruit() {
        echo "Le chien aboie.<br>";
    }
}
```

Implémentation d'une interface :

Une classe peut implémenter une interface à l'aide du mot-clé **implements**. Elle doit alors fournir une implémentation pour **toutes les méthodes définies** dans cette interface.

Les Exceptions

GESTION DES EXCEPTIONS

La gestion des **exceptions** est essentielle dans la programmation **PHP** pour plusieurs raisons, notamment pour gérer les **erreurs fatales**. Voici pourquoi la gestion des exceptions est importante, en particulier pour les erreurs fatales:

- **Arrêt contrôlé du script** :
  Les erreurs fatales, telles que les erreurs de syntaxe, les erreurs de type et les erreurs de fonctionnement critiques, interrompent généralement l'exécution du script **PHP**.
  En utilisant la gestion des exceptions, vous pouvez attraper ces erreurs et prendre des mesures pour gérer la situation de manière appropriée, plutôt que de simplement interrompre brutalement l'exécution du script.

- **Affichage d'informations utiles** :
  Lorsqu'une exception est levée, vous avez la possibilité de fournir des informations utiles sur l'erreur, telles que le type d'erreur, l'emplacement où elle s'est produite et des détails supplémentaires sur la cause de l'erreur. Cela permet de diagnostiquer et de résoudre plus facilement les problèmes.

- **Continuité de l'application** :
  En attrapant les exceptions et en gérant les erreurs de manière appropriée, vous pouvez permettre à votre application de continuer à fonctionner même lorsque des erreurs se produisent. Cela peut éviter des temps d'arrêt inattendus et assurer une meilleure expérience utilisateur.

GESTION DES EXCEPTIONS

Gérer les exceptions avec try catch

- Le bloc **try**...**catch** est utilisé pour entourer le code qui pourrait générer une exception.
- Dans le bloc **try**, vous placez le code susceptible de déclencher une exception.
- Le bloc **catch** est utilisé pour attraper et gérer les exceptions qui sont levées dans le bloc **try**.
- Vous pouvez spécifier le type d'exception à attraper après le mot-clé **catch**. Si aucune exception n'est spécifiée, le bloc **catch** attrapera toutes les exceptions.

```php
try {
// Code susceptible de générer une exception
$result = 10 / 0;
} catch (DivisionByZeroError $e) {
 echo "Division par zéro !";
}
```

GESTION DES EXCEPTIONS

Gérer les exceptions avec try catch

- En plus des blocs **try** et **catch**, vous pouvez également utiliser un bloc **finally** facultatif.
- Le bloc **finally** est toujours exécuté, que l'exception soit levée ou non. Il est souvent utilisé pour effectuer un nettoyage ou des opérations de clôture.

```
try {
        // Code susceptible de générer une exception
    }
catch (Exception $e) {
        // Gestion de l'exception
    }
finally {
        // Code exécuté après le bloc try-catch, indépendamment de l'exception
    }
```

PHP PDO

PHP PDO

Introduction

Le **PDO** (**PHP Data Objects**) est une **extension PHP** qui offre une interface unifiée pour **accéder à des bases de données**. Elle permet d'interagir avec différentes bases de données en utilisant un même ensemble de fonctions, ce qui rend le code plus portable et sécurisé.

```php
// Paramètres de connexion à la base de données
$host = 'localhost';
$dbname = 'nom_de_la_base_de_donnees';
$user = 'nom_utilisateur';
$password = 'mot_de_passe';

try {
    // Connexion à la base de données
    $connexion = new PDO("mysql:host=$host;dbname=$dbname", $user, $password);

    // Définition des attributs de gestion d'erreurs
    $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    echo "Connexion réussie !";
} catch(PDOException $e) {
    echo "Erreur de connexion : " . $e->getMessage();
}
```

PHP PDO

Effectuer une requête vers la bdd

Pour effectuer une requête **SQL** vers la base de données , une fois la connexion vers la **bdd** établie, la requête peut être exécuter avec la méthode **exec()** de l'objet **PDO**

```php
try {
    $host = "localhost";
    $db = "courspdo";
    // Connexion à la base de données avec PDO
    $connexion = new PDO("mysql:host=$host;dbname=$db", "root", "");
    // Configuration de PDO pour générer des exceptions en cas d'erreur
    $connexion->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
    // Message de succès si la connexion est établie avec succès
    echo "Connexion réussie !";
    // Requête SQL d'insertion de données dans la table persons
    $query = "INSERT INTO persons (fullName, age, email) VALUES ('TOTO', 99, 'toto@test.fr');";
    // Exécution de la requête SQL d'insertion avec la méthode exec()
    $connexion->exec($query);
} catch (PDOException $e) {
    // Gestion des exceptions PDO : affichage du message d'erreur en cas d'échec
    echo "Erreur : " . $e->getMessage();
}
```

PHP PDO

Requête paramétrée et préparée

Pour effectuer une requête **SQL** avec des valeurs en paramètre , il est important de préparer la requête avec la méthode **prepare** de l'objet **PDO**, une requête paramétrée et non préparer est la porte ouverte au injection de code SQL ! Les paramètres peuvent être nommée avec des **:nomDuParametre** , ou par un **?**.

Pour renseigner la valeur du paramètre, on utilise la méthode **bindParam** qui prend le nom du paramètre (**:nomDuParam**) ou le numéro du paramètre **1 , 2 , 3**..(si vous utiliser les **?**) ainsi que la valeur à renseigner. Le méthode **bindParam** permet d 'échapper les caractères SQL et donc d 'éviter les injections SQL.

```php
// Préparation de la requête SQL avec un marqueur de paramètre
  $query = "INSERT INTO persons (fullName , age , email) VALUES (:fullName, :age, :email)";
  $statement = $connexion->prepare($query);
  // Liaison des valeurs des paramètres
  $nom = " John Doe ";
  $age = 99;
  $email = "JohnDoe@test.fr";
  $statement->bindParam(":fullName", $nom);
  $statement->bindParam(":age", $age);
  $statement->bindParam(":email", $email);
  // Exécution de la requête préparée
  $statement->execute();
  echo "Insertion réussie !";
```

PHP PDO

Requête paramétrée et préparée

Pour effectuer une requête **SQL** avec des valeurs en paramètre , il est important de préparer la requête avec la méthode **prepare** de l'objet **PDO**, une requête paramétrée et non préparer est la porte ouverte au injection de code SQL ! Les paramètres peuvent être nommée avec des **:nomDuParametre** , ou par un **?**.

Pour renseigner la valeur du paramètre, on utilise la méthode **bindParam** qui prend le nom du paramètre (**:nomDuParam**) ou le numéro du paramètre **1 , 2 , 3**..(si vous utiliser les **?**) ainsi que la valeur à renseigner. Le méthode **bindParam** permet d 'échapper les caractères SQL et donc d 'éviter les injections SQL.

```php
// Préparation de la requête SQL avec un marqueur de paramètre
  $query = "INSERT INTO persons (fullName , age , email) VALUES (:fullName, :age, :email)";
  $statement = $connexion->prepare($query);
  // Liaison des valeurs des paramètres
  $nom = " John Doe ";
  $age = 99;
  $email = "JohnDoe@test.fr";
  $statement->bindParam(":fullName", $nom);
  $statement->bindParam(":age", $age);
  $statement->bindParam(":email", $email);
  // Exécution de la requête préparée
  $statement->execute();
  echo "Insertion réussie !";
```

PHP PDO

Requête de récupération

Pour effectuer une requête **SQL** de récupération une fois préparer et exécuter, on récupère les valeurs à l'aide de la méthode **fetch()** ou **fecthAll()** si on a plusieurs résultats.

La constante **PDO::FETCH_ASSOC** est un mode de récupération des résultats de la requête. Lorsque cette constante est utilisée avec la méthode **fetchAll()**, chaque ligne de résultat est renvoyée sous forme **de tableau associatif**, où les **clés** du tableau correspondent aux **noms des colonnes** de la table de la base de données.

```php
// Préparation de la requête SQL SELECT
$query = "SELECT * FROM persons";
$statement = $connexion->prepare($query);
// Exécution de la requête
$statement->execute();
// Récupération des résultats de la requête
$persons = $statement->fetchAll(PDO::FETCH_ASSOC);
// Affichage des résultats
foreach ($persons as $person) {
    echo "ID : " . $person['id'] . "<br>";
    echo "Nom complet : " . $person['fullName'] . "<br>";
    echo "Age : " . $person['age'] . "<br>";
    echo "Email : " . $person['email'] . "<hr>";
}
```

PHP PDO

Les transactions

Les **transactions** en base de données sont des séquences d'opérations qui sont exécutées comme une unité indivisible. Elles garantissent l'intégrité des données en assurant que toutes les opérations sont exécutées avec succès ou aucune d'entre elles ne l'est. Cela signifie que si une opération échoue, toutes les opérations précédentes doivent être annulées pour éviter les incohérences dans la base de données.

Pour gérer les transactions avec **PDO**, on utilise les méthodes **beginTransaction()**, **commit()** et **rollback()**. La méthode **beginTransaction()** démarre une nouvelle transaction, **commit()** valide la transaction et **rollback()** annule la transaction en cas d'erreur.

```php
// Début d'une transaction
$connexion->beginTransaction();
try {
    // Opérations de modification de la base de données
    $connexion->exec("INSERT INTO persons (fullName, age, email) VALUES ('John Doe', 30, 'john@example.com')");
    $connexion->exec("UPDATE persons SET age = 31 WHERE fullName = 'John Doe'");
    // Validation de la transaction
    $connexion->commit();
    echo "Transactions effectuées avec succès !";
} catch (PDOException $e) {
    // En cas d'erreur, annulation de la transaction
    $connexion->rollback();
    echo "Erreur de transaction : " . $e->getMessage();
}
```