La Programmation Orientée Objet en PHP

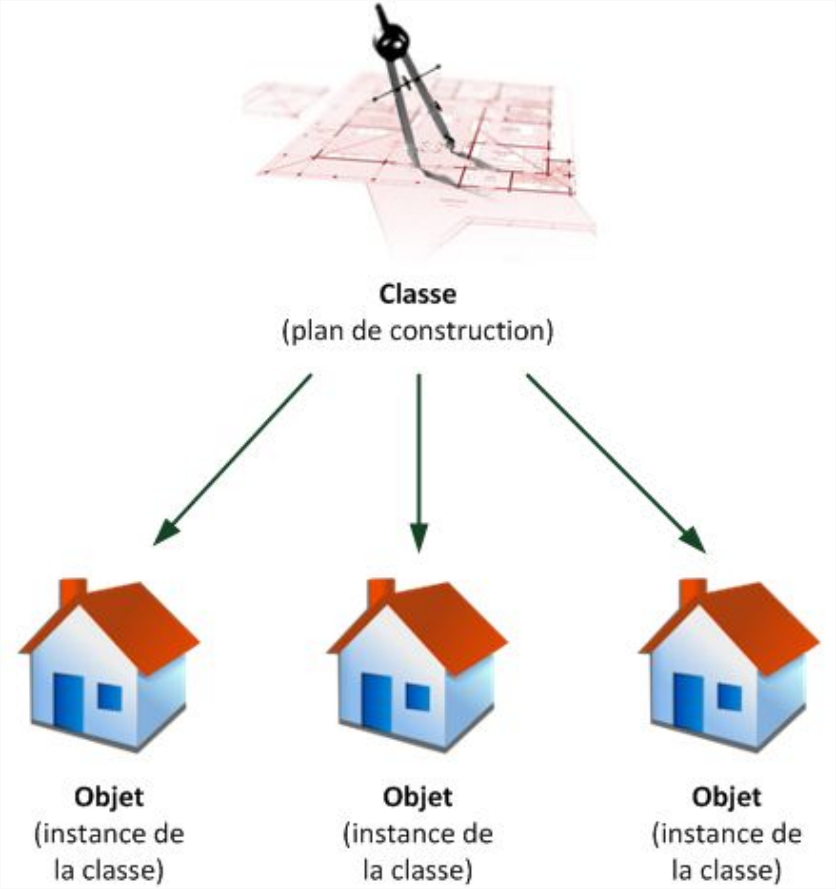
Classe et objet

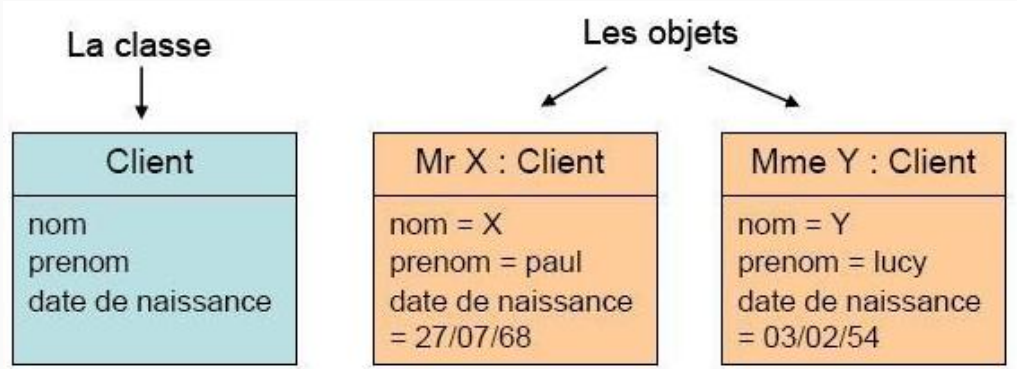
Une classe, c'est un ensemble de variables et de fonctions (attributs et méthodes).

Les classes forment la structure des données et des actions et utilisent ces informations pour créer des objets.

Un objet, c'est une instance de la classe pour pouvoir l'utiliser.

Classe et objet





Classe et objet

- La syntaxe pour créer une classe: déclarer une classe en utilisant le mot-clé **class**, suivi du nom de la classe et d'un ensemble d'accolades.
- Après avoir créé la classe, une nouvelle classe peut être instanciée et stockée dans une variable en utilisant le nouveau mot-clé: **new**

Classe et objet

```php
<?php

class User {
    // Mettre les attributs et méthodes ici
}

$me  = new User();

var_dump($me);

?>
```

propriété et méthode d'une classe

```php
<?php

class Utilisateur {

    public $prenom = 'Bamba';

    public function presentation() {
        echo 'je suis un objet de la classe utilisateur';
    }

}

$me  = new Utilisateur();

echo $me->prenom;
$me->presentation();
```

propriété et méthode d’une classe

La POO permet aux objets de se référencer en utilisant **$this**. Lorsque vous travaillez dans une méthode, utilisez $this de la même manière que vous utiliseriez le nom de l'objet en dehors de la classe.

propriété et méthode d'une classe

```php
class Utilisateur {

    public $prenom = 'Bamba';

    public function changerDePrenom($nouveauPrenom) {
        $this->prenom = $nouveauPrenom;
    }

    public function retounerPrenom() {
        return $this->prenom;
    }

}

$me  = new Utilisateur();

$me->changerDePrenom('Ouz');
echo $me->retounerPrenom();
```

La visibilité public / private

La visibilité permet de définir comment une propriété ou une méthode pourra être utilisée.

- **public**, permet d'indiquer que la propriété ou la méthode sera accessible à l'intérieur mais aussi à l'extérieur de la classe
- **private**, permet d'indiquer que la propriété ou la méthode sera accessible à l'intérieur de la classe seulement
- **protected**, permet d'indiquer que la propriété ou la méthode sera accessible à l'intérieur de la classe et des classes héritées

getters et setters

Pour lire ou modifier un attribut, on utilise des getters et des setters.

```php
class Utilisateur {

    private $prenom;
    private $nom;

    public function setPrenom($prenom) {
        $this->prenom = $prenom;
    }

    public function getPrenom() {
        return $this->prenom;
    }

    public function setNom($nom) {
        $this->nom = $nom;
    }

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

Le constructeur

Le constructeur d'une classe a pour rôle principal d'initialiser l'objet en cours de création, c'est-à-dire d'initialiser la valeur des attributs (soit en assignant directement des valeurs spécifiques, soit en appelant diverses méthodes).

Le constructeur

```php
public function __construct($prenom, $nom) {
    echo 'La classe "', __CLASS__, '" est initialisée!<br />';
    $this->setPrenom($prenom);
    $this->setNom($nom);
}
```

```php
$me  = new Utilisateur("Nanko", "Diouf");
echo $me->getPrenom().' '.$me->getNom();
```

Les constantes de classe

Il est possible de définir des valeurs constantes à l'intérieur d'une classe, qui ne seront pas modifiables. Les constantes diffèrent des variables normales du fait que l'on n'utilise pas le symbole $ pour les déclarer ou les utiliser. La visibilité par défaut des constantes de classe est *public*.

les constantes de classe

```php
private $prenom;
private $nom;
private $role;
const DEFAULT_ROLE = 'USER_SIMPLE';
public function __construct($prenom, $nom, $role) {
    $this->setPrenom($prenom);
    $this->setNom($nom);
    $this->setRole($role);
}

public function afficheDefaultRole() {
    echo self::DEFAULT_ROLE.'<br>';
}
```

```php
$me  = new Utilisateur("Nanko", "Diouf", Utilisateur::DEFAULT_ROLE);
$me->afficheDefaultRole();
echo $me->getPrenom().' '.$me->getNom().' '.$me->getRole();
```

les attributs et méthodes statiques

les méthodes statiques sont des méthodes qui sont faites pour agir sur une classe et non sur un objet.

les attributs et méthodes statique

```php
private $prenom;
private $nom;
private static $compteur = 0;

public function __construct($prenom, $nom) {
    $this->setPrenom($prenom);
    $this->setNom($nom);
    self::$compteur++;
}

public static function getCompteur() {
return self::$compteur;
}
```

```php
$user1  = new Utilisateur("Nanko", "Diouf");
$user2  = new Utilisateur("Diamil", "Cisse");

echo Utilisateur::getCompteur();
```

heritage

Les classes peuvent hériter des méthodes et des propriétés d'une autre classe à l'aide du mot-clé extends

Heritage

```php
<?php

class Personne {
    protected $nom;
    protected $prenom;
}

class Etudiant extends Personne {
    private $numeroMatricule;
}


class Professeur extends Personne {
    private $specialites;
}


?>
```

Heritage

```php
class Personne {
    protected $prenom;
    protected $nom;

    public function __construct($prenom, $nom) {
        $this->prenom = $prenom;
        $this->nom = $nom;

    }

    public function afficheNomComplet() {
        echo $this->prenom." ".$this->nom;
    }
}

class Etudiant extends Personne {
    private $numeroMatricule;

    public function __construct($prenom, $nom) {
        parent::__construct($prenom, $nom);
    }
}
```

Heritage

```php
$personne = new Personne("Lamine", "Ngom");
$etudiant = new Etudiant("Bamba", "Tounkara");

$personne->afficheNomComplet();
$etudiant->afficheNomComplet();
```

Heritage (surcharge de méthodes)

```php
class Personne {
    protected $prenom;
    protected $nom;

    public function __construct($prenom, $nom) {
        $this->prenom = $prenom;
        $this->nom = $nom;

    }

    public function afficheNomComplet() {
        echo $this->prenom." ".$this->nom;
    }
}

class Etudiant extends Personne {
    private $numeroMatricule;

    public function __construct($prenom, $nom, $numeroMatricule) {
        parent::__construct($prenom, $nom);
        $this->numeroMatricule = $numeroMatricule;
    }

    public function afficheNomComplet() {
        echo $this->prenom." ".$this->nom." ".$this->numeroMatricule;
    }
}
```

Heritage(surcharge de methode)

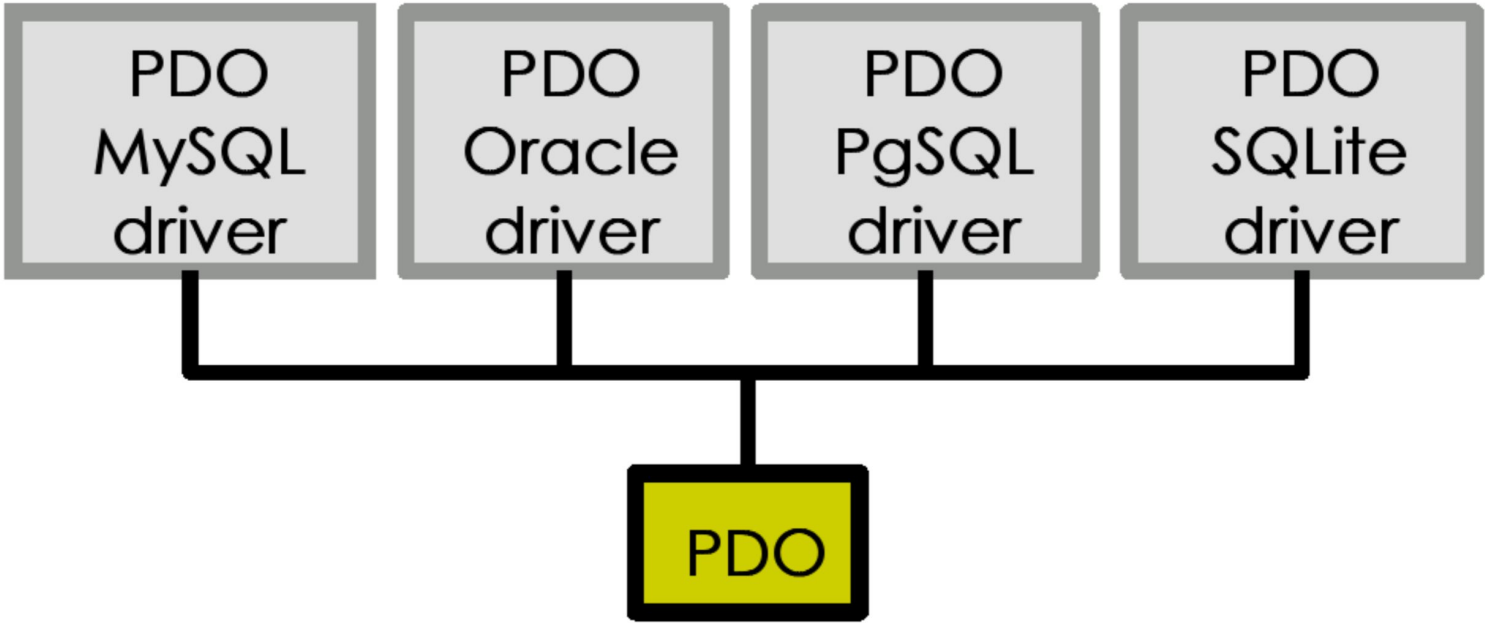
```php
$personne = new Personne("Lamine", "Ngom");
$etudiant = new Etudiant("Bamba", "Tounkara", "2012059I4");

$personne->afficheNomComplet();
$etudiant->afficheNomComplet();
```

PHP Connect to MySQL

PDO est une interface d'accès aux bases de données.

PDO fonctionnera sur 12 systèmes de base de données différents (Oracle, PostgreSQL….), alors que MySQLi ne fonctionnera qu'avec les bases de données MySQL.

les requêtes préparés



les requêtes préparés

- La classe PDO gère l'accès aux SGBD ainsi que les fonctionnalités de base :
  a. Connexions
  b. Lancement des requêtes
- La classe PDOStatement gère une liste de résultats.

- La classe PDOException est une classe d'exception personnalisée interne pour PDO.

PHP connect to mysql

```php
<?php
$servername = "localhost";
$username = "root";
$password = "azerty99";

try {
    $conn = new PDO("mysql:host=$servername;dbname=tdsi_2018", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

    echo "Connection réussie";
}
catch(PDOException $e){
    echo "echec connection: " . $e->getMessage();
}

?>
```

Les requêtes préparées

les requêtes préparées

Le système de *requêtes préparées* a l'avantage d'être beaucoup plus sûr mais aussi plus rapide pour la base de données si la requête est exécutée plusieurs fois. C'est ce que je préconise d'utiliser si vous voulez adapter une requête en fonction d'une ou plusieurs variables.

Avec les requêtes paramétrées il n'est plus nécessaire de se protéger des attaques par injection SQL car le serveur sait ce qu'il attend.

les requêtes préparés

```php
try {
    $conn = new PDO("mysql:host=$servername;dbname=tdsi_2018", $username, $password);
    $conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
}catch(PDOException $e){
    die("echec connection: " . $e->getMessage());
}

$req = $conn->prepare('SELECT * FROM proverbes WHERE user_id = ?');
$req->execute(array($_GET['user_id']));

while ($row = $req->fetch()){
    echo $row['id']." ".$row['titre']." ".$row['description']."<br>";
}
```

Insertion

```php
$titre = 'Olof Ndiaye néna';
$description = 'dou lébi néneu';
$user_id = 4;
$req = $conn->prepare('INSERT INTO proverbes(titre, description, user_id)
VALUES(:titre, :description, :user_id)');

$req->execute(array(

    'titre' => $titre,

    'description' => $description,

    'user_id' => $user_id

    ));
```