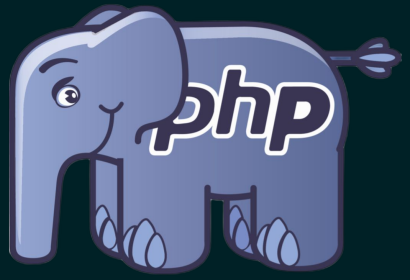
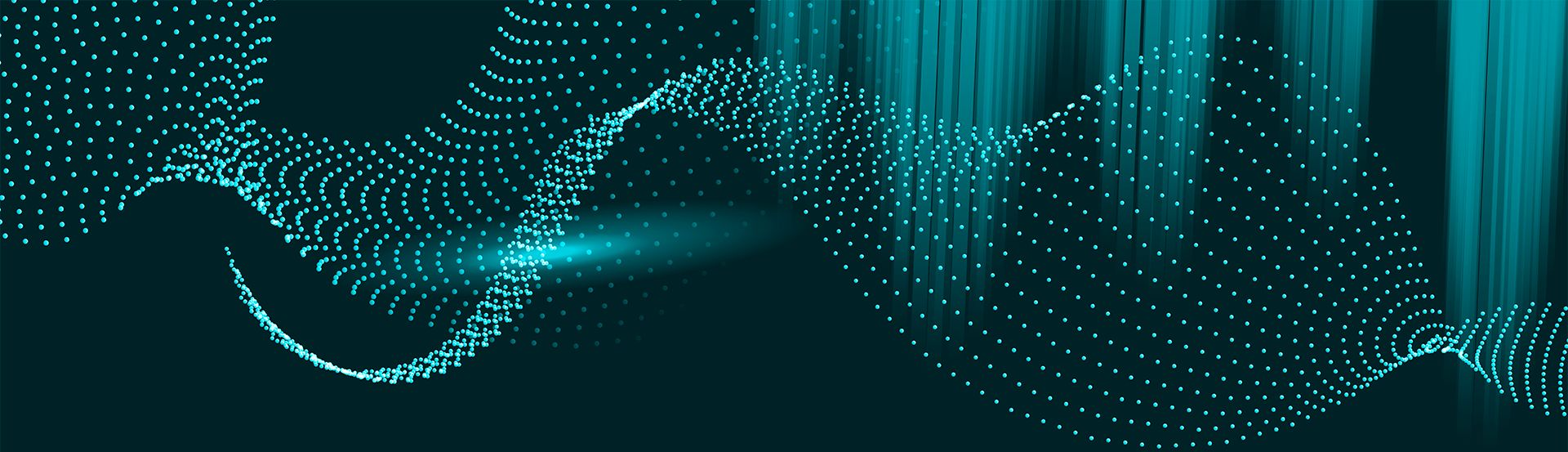
Semestre 3
Back-End

Semaine 16 - PHP Orienté Objet



Par Pierre-Alexandre Lacaze - pa.lacaze1@gmail.com



01 L’orienté Objet

Qu’est ce que c’est ?

Définition et concept

- Paradigme de programmation
  - Repose sur l'organisation du code via des objets

- Structurer le code en petites entités indépendantes
  - Meilleure organisation
  - Réutilisation accrue
  - Evolution plus facile

Les Objectifs

Organisation

Modularité

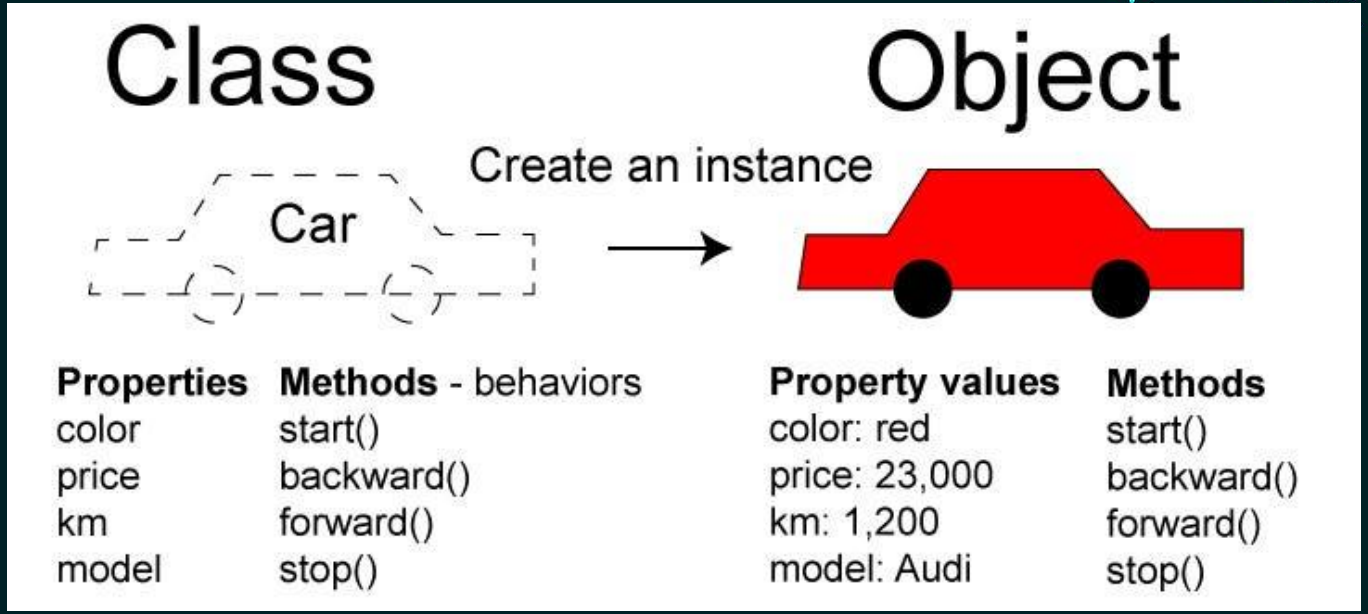
Réutilisation

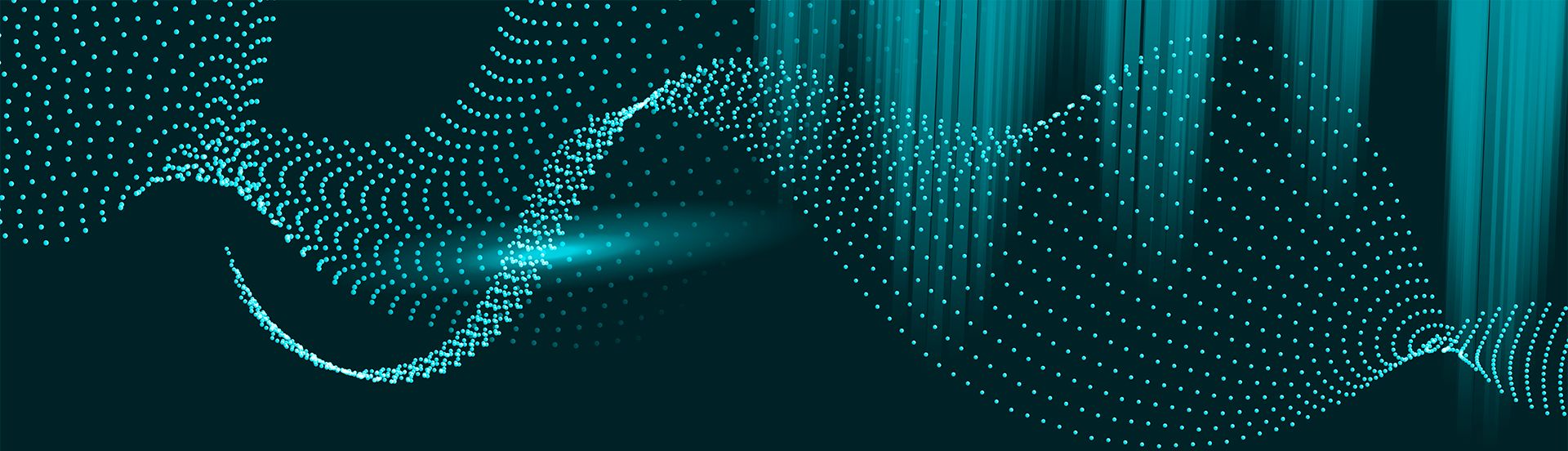
Evolution

Les principes

- Les Classes
  - Modélisation / Plan
  - Définissent les caractéristiques et le comportement d'un objet
  - Regroupent les propriétés (données) et méthodes (comportements)
- Les Objets
  - Instance / Concrétisation de la classe
  - Possèdent des valeurs propres
  - Indépendants les uns des autres
- Les Méthodes
  - Méthodes = fonctions
  - Décrivent le comportement de l'objet
  - Peuvent agir sur les propriétés ou exécuter une action
- Les Propriétés
  - Propriétés = variables
  - Utilisées pour stocker des informations sur l'objet
  - Chaque objet est indépendant et a des caractéristiques propres

Les principes en image





02 PHP OO

Ses spécificités

Evolution de PHP vers la POO

- A l'origine seulement procédural
- Test de la POO dans PHP 4
- Révolution de la POO avec PHP 5 en 2004

Présence des concepts de base

- Création de classe

```php
class Utilisateur {
    public $nom;
    public $email;

    public function afficherInfos() {
        echo "Nom : " . $this->nom . ", Email : " . $this->email;
    }
}
```

- Instantiation d'un objet

```php
$utilisateur1 = new Utilisateur();
$utilisateur1->nom = "Pierra";
$utilisateur1->email = "pierra@mail.com";
$utilisateur1->afficherInfos(); // Affiche : Nom : Pierra, Email : pierra@mail.com
```

Présence des concepts de base

- Encapsulation

```php
class Utilisateur {
    private $motDePasse;

    public function setMotDePasse($mdp) {
        $this->motDePasse = password_hash($mdp, PASSWORD_BCRYPT);
    }

    public function getMotDePasse() {
        return $this->motDePasse;
    }
}
```

Présence des concepts de base

- Heritage

```php
class Animal {
    public $nom;

    public function dormir() {
        echo "$this->nom dort.";
    }
}

class Chien extends Animal {
    public function aboyer() {
        echo "$this->nom aboie.";
    }
}

$chien = new Chien();
$chien->nom = "Pilou";
$chien->aboyer(); // Affiche : Pilou aboie.
```

Présence des concepts de base

- Interface

```php
interface Authentifiable {
    public function login($email, $motDePasse);
}

class Utilisateur implements Authentifiable {
    public function login($email, $motDePasse) {
        // Logique d'authentification
    }
}
```

Présence des concepts de base

- Polymorphisme

```php
class Animal {
    public function parler() {
        return "L'animal fait un bruit.";
    }
}

class Chien extends Animal {
    public function parler() {
        return "Le chien aboie.";
    }
}

class Chat extends Animal {
    public function parler() {
        return "Le chat miaule.";
    }
}

function faireParler(Animal $animal) {
    echo $animal->parler();
}

$chien = new Chien();
$chat = new Chat();

faireParler($chien); // Le chien aboie.
faireParler($chat);  // Le chat miaule.
```

Particularité en PHP

- Méthodes magiques

```php
class Utilisateur {
    public $nom;
    public $email;

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

$utilisateur = new Utilisateur("Bob", "bob@mail.com");
echo $utilisateur->nom; // Affiche : Bob
```

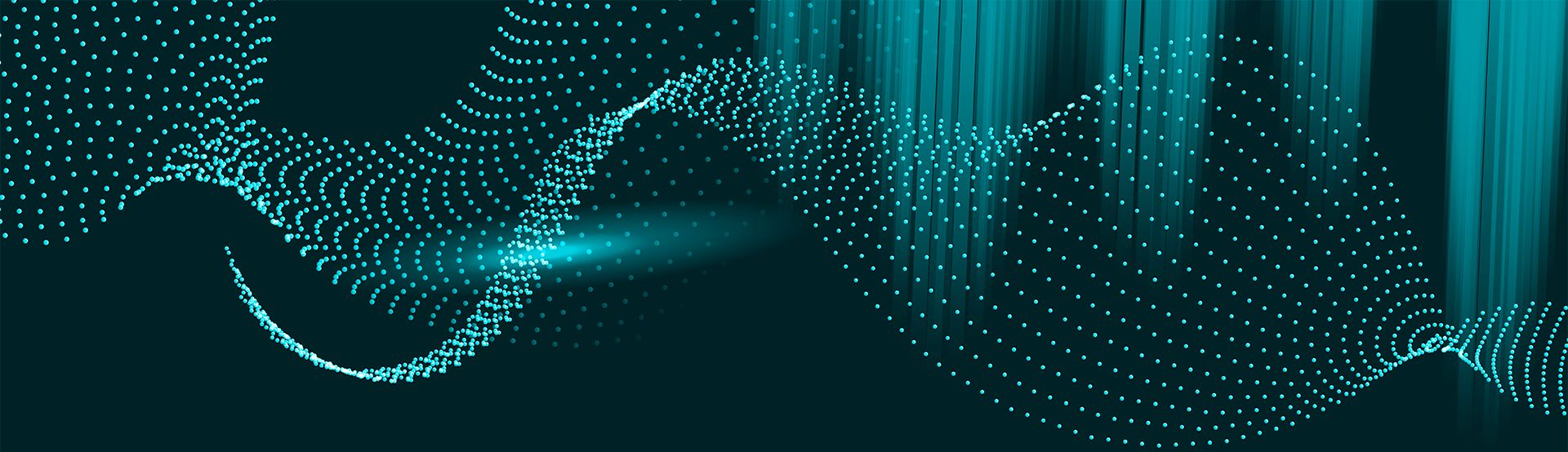
Particularité en PHP

- Trait

```php
// Trait pour l'upload d'images
trait ImageUploader {
    public function uploadImage($image) {
        // Simuler l'upload d'image
        echo "Image " . $image . " uploadée avec succès!<br>";
    }
}

// Classe Product avec la possibilité d'uploader une image de produit
class Product {
    use ImageUploader;

    public function uploadProductImage($image) {
        $this->uploadImage($image);
    }
}

// Utilisation
$product = new Product();
$product->uploadProductImage("produit.jpg");
?>
```

Particularité en PHP

- Quelques autres particularités
  - Interfaces avec Méthodes Privées
  - Visibilité des propriétés statics
  - Gestion des fichiers PHAR
  - Variabilité du Typage
  - Late Static Binding



03

Structure et pratiques

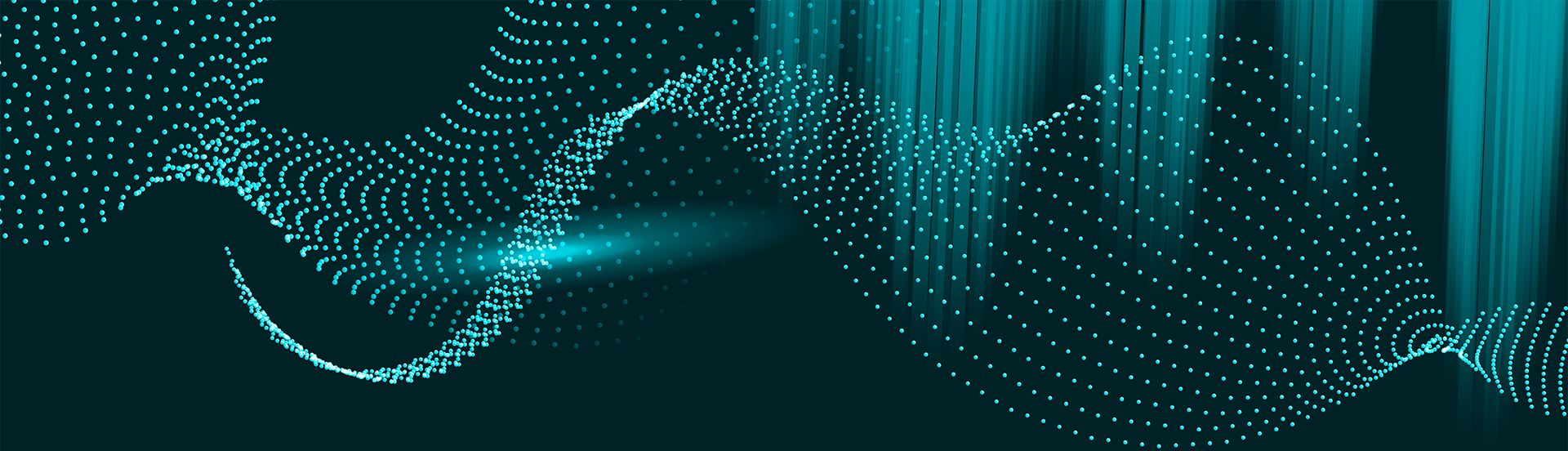
Comment organiser son code et ses fichiers

Structurer un projet POO en PHP

- Séparation des responsabilités
  - 1 Classe = 1 Responsabilité
- Organisation des fichiers
  - 1 Classe = 1 Fichier
  - Séparation des classes par rôles
    - Classes avec action BDD = Modèles
    - Classes avec contrôle de flux = Contrôleurs
  - Séparation des fichiers html
- Autoloading
- Réutilisation et modularité

Convention de nommage et bonnes pratiques

- Noms de classes = PascalCase
  - User, Product, Order
- Noms des méthodes et variables = camelCase
  - ajouterProduit(), getPrix(), $quantiteDispo
- Noms des fichiers nommés au nom de la classe
  - Classe Produit = Produit.php ou Produit.class.php
- Commenter le code
- Ne pas trop complexifier les classes
- Utiliser des namespaces



04 Modélisation UML

Réflexion sur la structure système - Diagramme de Classes

UML, son utilisation

- UML = Unified Modeling Language
- Nous sert à représenter des concepts, des structures logicielles
- Non dépendant d'un langage de programmation
- 2 Catégories de diagrammes : Structure et Comportement

Visualisation

Communication

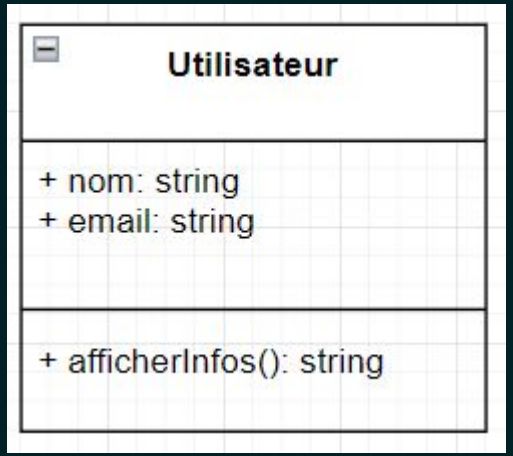
Documentation

Le Diagramme de classe

- Le plus utilisé
- Le plus important
- La structure statique d’un système orienté objet
- Décrit les classes, leurs attributs, leurs méthodes et les relations entre elles
- Du Diagramme de classe découle facilement la modélisation de la BDD

Une classe en UML

- 1 cadre -> 3 lignes
  - Nom de la classe
  - Propriétés de la classe
  - Méthodes de la classe



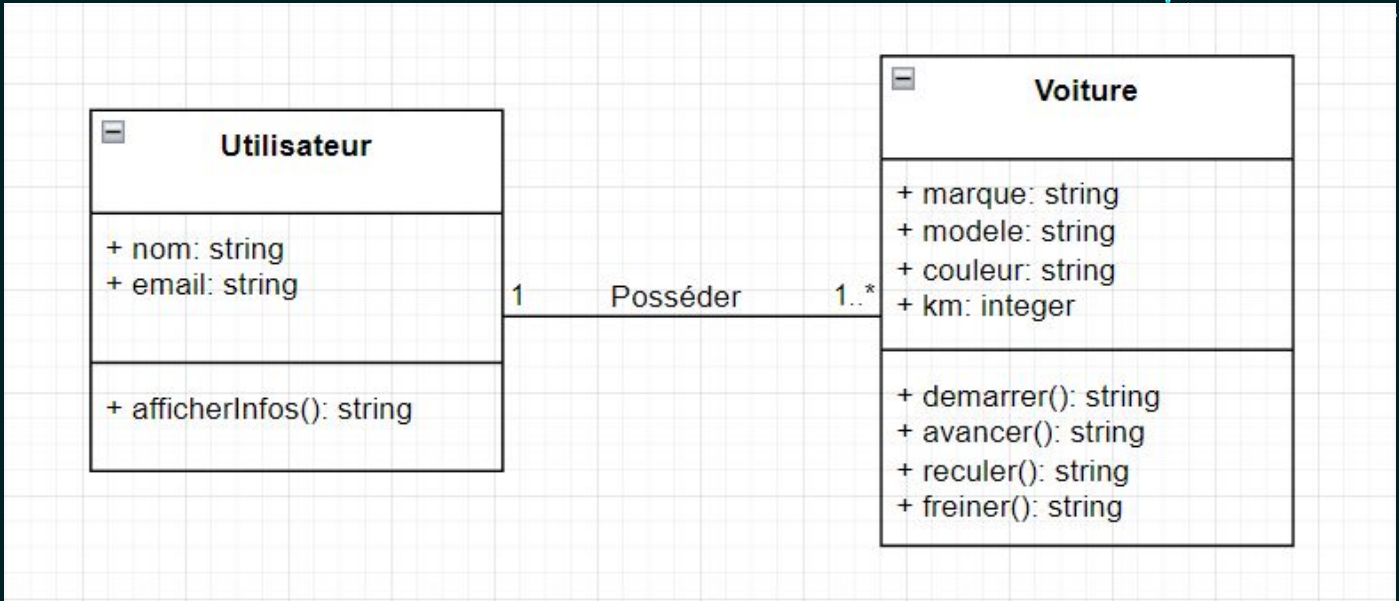
- Visibilité :
  - + public
  - # protected
  - -

Les relations entre les classes

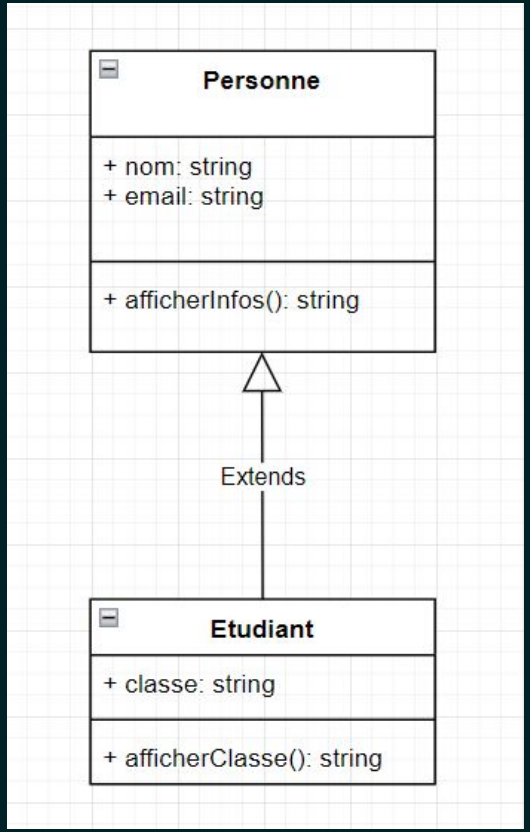
- Association : Relation normale entre deux classes
  - Ex : Une personne a plusieurs voitures
- Héritage : Relation classe Parent, classe Enfant
  - Ex : Un étudiant est une personne
- Composition : Relation pour dire que certaines classes en composent une autre
  - Ex : Une voiture est composé d'un moteur, de roues, etc.
- Agrégation : Similaire à la composition mais les classes existent indépendamment
  - Ex : Une classe est composée de plusieurs élèves

- Entre ces relations on définit des multiplicités pour éclaircir le contexte, cela nous aidera pour la conception de la BDD

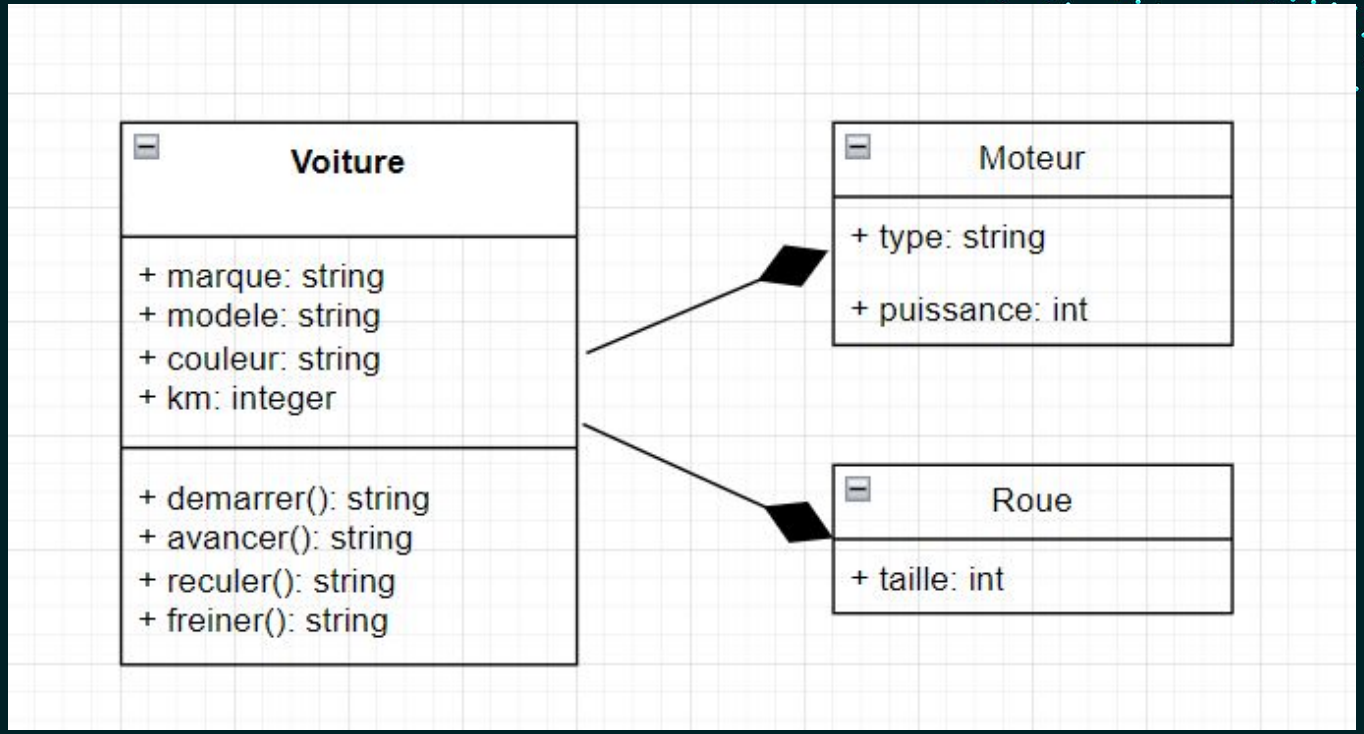
Association



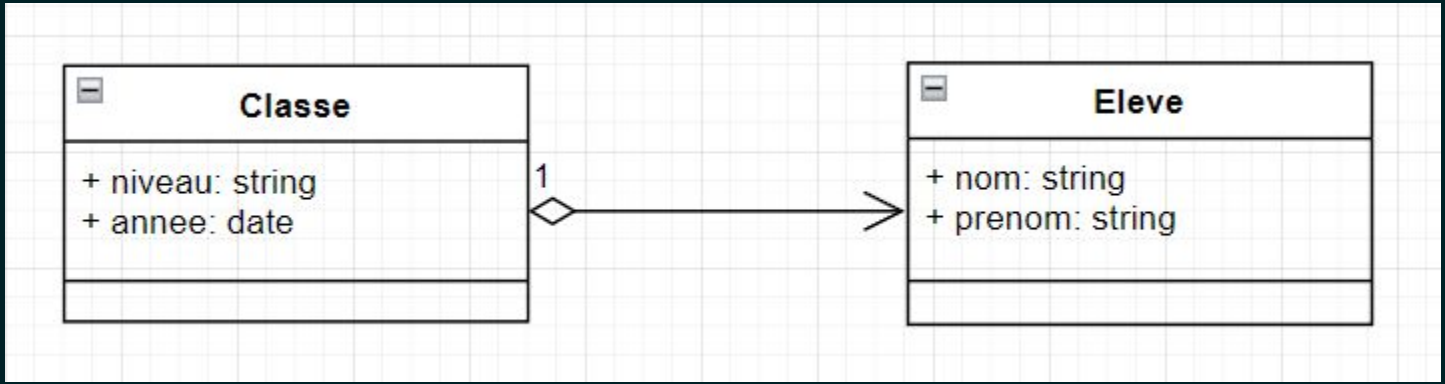
Héritage


Personne
+ nom: string
+ email: string
+ afficherInfos(): string
Extends
Etudiant
+ classe: string
+ afficherClasse(): string

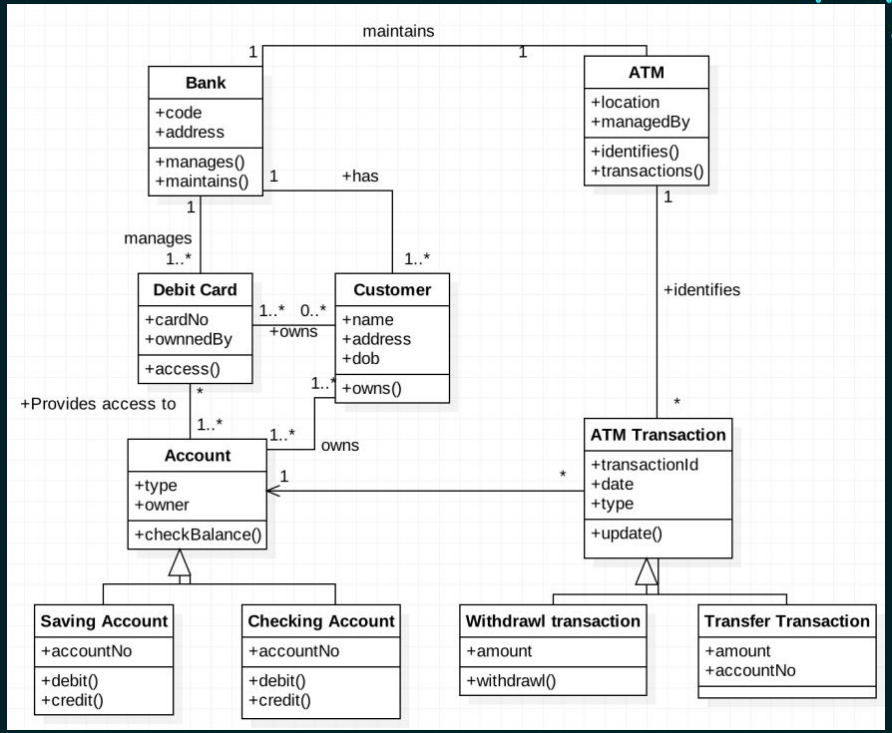
Composition



Agrégation

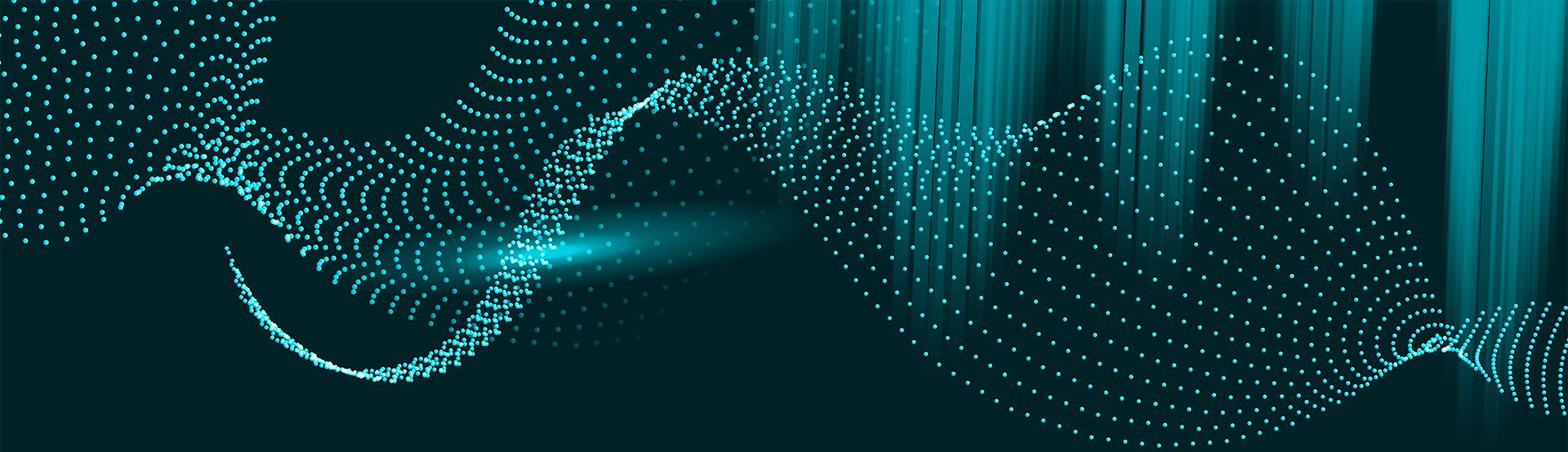

Classe
+ niveau: string
+ annee: date
1
Eleve
+ nom: string
+ prenom: string

Exemple d’un diagramme complet



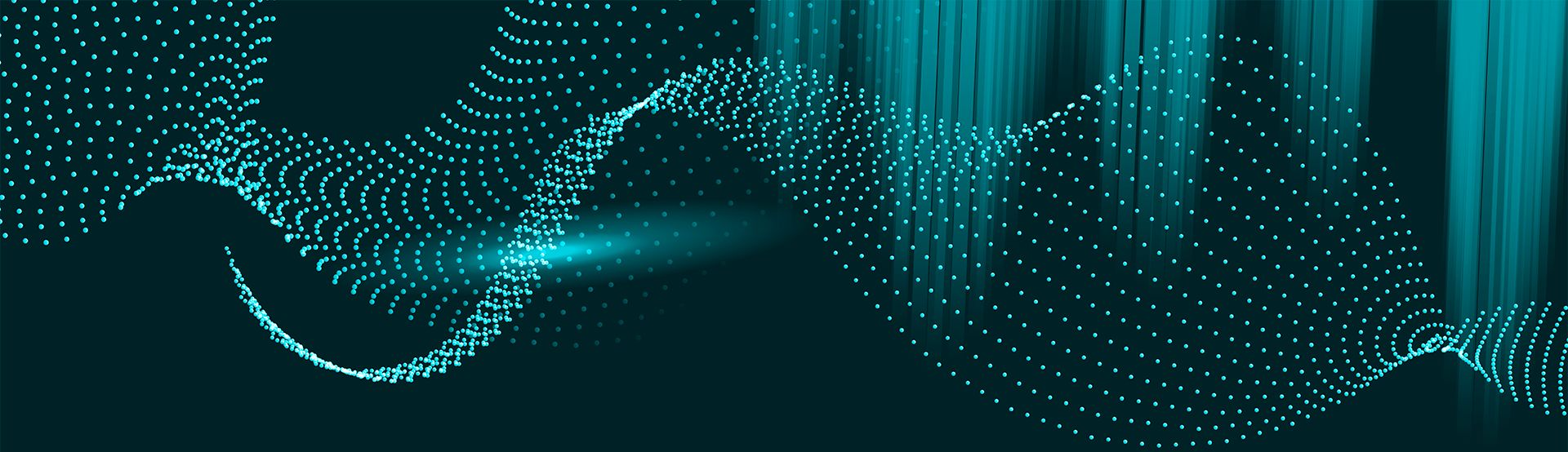
Réaliser un bon diagramme de classe UML

1. Comprendre le domaine du problème
2. Identifier les classes principales
3. Définir les attributs et méthodes
4. Etablir les relations entre les classes
5. Utiliser des noms clairs et significatifs
6. Gérer l'héritage intelligemment
7. Penser à l'encapsulation
8. Modéliser les comportements clés
9. Eviter les diagrammes trop complexes
10. Utiliser des outils de modélisation (diagrams.net)
11. Vérification et révision



05 Pratique

Voir les fichiers de code



06 Le patron MVC

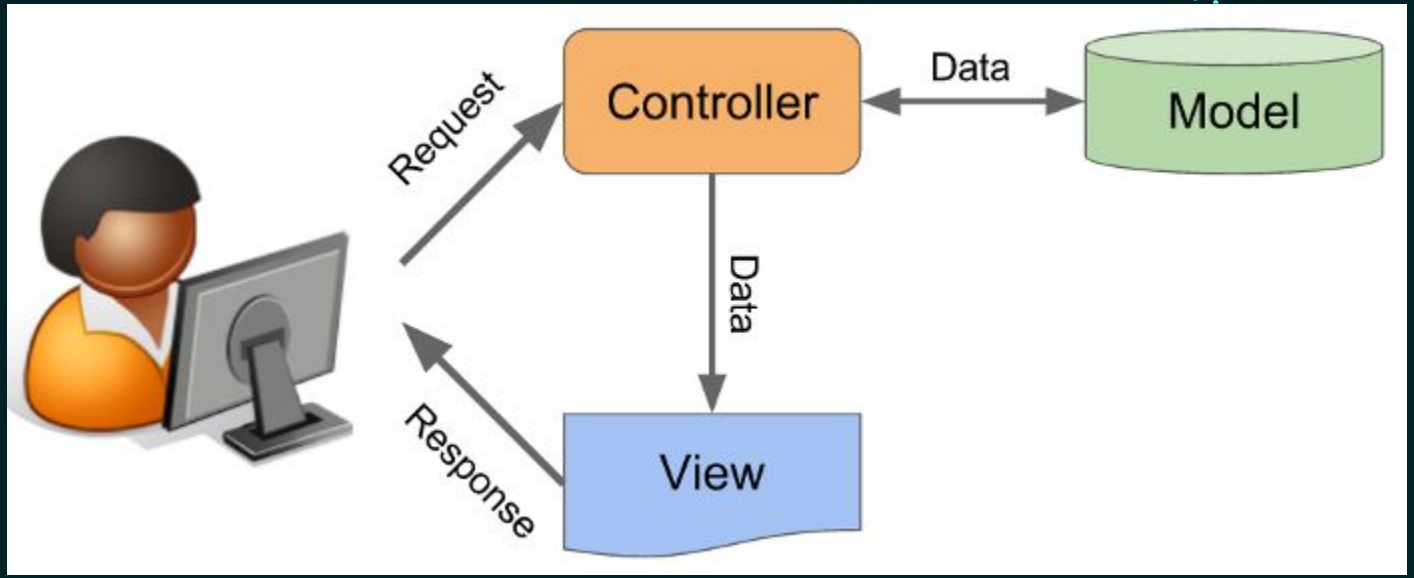
Modèle - Vue - Contrôleur
Model - View - Controller

Le Modèle-Vue-Contrôleur

- Patron d'architecture constamment utilisé dans le web

1. Modèle (Model) = partie qui gère les données = interaction BDD
2. Vue (View) = gère la présentation des données = HTML/CSS
3. Contrôleur (Controller) = Intermédiaire gérant les requêtes de l'utilisateur = Décisionnaire de l'entièreté des interactions PHP

Le Flux MVC



Exemple MVC : Arborescence

```
/app
    /controllers
        UserController.php
    /models
        User.php
    /views
        user-list.php
/index.php
```

Exemple MVC : Modèle

- Interagit sur les Users de la BDD

```php
class User {
    private $db;

    public function __construct($db) {
        $this->db = $db;
    }

    public function getAllUsers() {
        $stmt = $this->db->query('SELECT * FROM users');
        return $stmt->fetchAll(PDO::FETCH_ASSOC);
    }
}
```

Exemple MVC : Contrôleur

- Interagit avec le Modèle et renvoie les infos à la Vue

```php
require_once '../models/User.php';

class UserController {
    private $userModel;

    public function __construct($db) {
        $this->userModel = new User($db);
    }

    public function listUsers() {
        $users = $this->userModel->getAllUsers();
        require '../views/user-list.php';
    }
}
```

Exemple MVC : Vue

- Formate les données transmises par le Contrôleur

```php
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User List</title>
</head>
<body>
    <h1>Liste des utilisateurs</h1>
    <ul>
        <?php foreach ($users as $user): ?>
            <li><?php echo $user['name']; ?> (<?php echo $user['email']; ?>)</li>
        <?php endforeach; ?>
    </ul>
</body>
</html>
```

Exemple MVC : Point d’entrée

- Index.php reçoit les requêtes client et renvoie vers le contrôleur

```php
// Connexion à la base de données via PDO
$db = new PDO('mysql:host=localhost;dbname=testdb', 'root', '');

// Inclure le contrôleur
require 'app/controllers/UserController.php';

// Initialiser le contrôleur
$controller = new UserController($db);

// Appeler la méthode pour afficher la liste des utilisateurs
$controller->listUsers();
```

Bonnes pratiques et Avantages du MVC

- Séparation stricte des responsabilités / préoccupations
- Modularité
- Réutilisabilité

LE MVC UNE PORTE VERS DE NOUVELLES POSSIBILITÉS