Cours 07 - Programmation orientée objet

https://github.com/heig-vd-progserv1-course

Support de cours · Présentation (web) · Présentation (PDF)

L. Delafontaine, avec l'aide de GitHub Copilot.

Retrouvez plus de détails dans le support de cours

Cette présentation est un résumé du support de cours. Pour plus de détails, consultez le support de cours.

Objectifs (1/2)

- Lister les concepts clés de la POO.
- Expliquer les avantages et les désavantages de la POO.
- Créer des classes et des objets en PHP.
- Définir des attributs et des méthodes dans une classe.
- Utiliser l'encapsulation pour protéger les données des objets.

Objectifs (2/2)

- Définir des constructeurs et des destructeurs pour initialiser et nettoyer les objets.
- Utiliser des constantes dans les classes.

Buts de la programmation orientée objet (POO)

- Paradigme de programmation basé sur des objets (= façon de penser/représenter l'information).
- Permet de créer des programmes modulaires et réutilisables.
- Permet de modéliser des entités du monde réel plus faciles à maintenir.

Concepts clés de la POO (1/2)

- **Classes** : modèles qui définissent les propriétés et les comportements des objets.
- **Objets** : instances (= création en mémoire) des classes qui représentent des entités concrètes.
- **Attributs** : variables qui stockent l'état des objets.
- **Méthodes** : fonctions qui définissent les comportements des objets.
- **Encapsulation** : protection des données des objets en limitant l'accès direct.
- **Constructeurs et destructeurs** : méthodes spéciales pour initialiser et nettoyer les objets.

Concepts clés de la POO (2/2)

Il existe d'autres concepts, mais ils ne seront pas abordés dans ce cours :

- Héritage/polymorphisme
- Interfaces
- Namespaces
- Exceptions
- Etc.

Avantages de la POO

- **Lisibilité** : le code est organisé en classes et objets, ce qui le rend plus facile à comprendre.
- **Réutilisabilité** : les classes peuvent être réutilisées, ce qui réduit la duplication de code.
- **Maintenabilité** : les modifications apportées à une classe n'affectent pas les autres classes, ce qui facilite la maintenance du code.

Désavantages de la POO

- **Complexité** : la POO peut être plus complexe que la programmation procédurale, ce qui peut rendre le code plus difficile à comprendre pour les débutants.
- **Performance** : la POO peut être moins performante que la programmation procédurale, car elle nécessite plus de ressources pour créer et gérer des objets.

La POO en PHP

- La POO est prise en charge par PHP depuis la version 5.
- PHP propose toutes les fonctionnalités de la POO (classes, objets, attributs, méthodes, encapsulation, constructeurs et les destructeurs).
- Explorons certains de ces concepts en PHP.

Classes (1/2)

- Une classe est un modèle qui définit les propriétés et les comportements des objets.
- Dans PHP, une classe est définie à l'aide du mot-clé `class`.
- Par convention, les noms de classes commencent sont en Pascal case (par exemple, `JeSuisUneClasse`).

Classes (2/2)

```php
<?php
class Person {
    public $name;
    public $age;
}
```

```java
// Équivalent en Java
public class Person {
    public String name;
    public int age;
}
```

Instanciation d'objets (1/2)

- Un objet est une instance d'une classe.
- On crée un objet en utilisant le mot-clé `new` suivi du nom de la classe suivi de parenthèses (`()`).
- Par convention, les noms d'objets sont écrits en Camel case (par exemple, `$jeSuisUnObjet`).

Instanciation d'objets (2/2)

```php
$person1 = new Person();
$person2 = new Person();
```

```java
// Équivalent en Java
Person person1 = new Person();
Person person2 = new Person();
```

Attributs (1/2)

- Les attributs sont des variables qui stockent l'état des objets.
- Accédées selon leur visibilité :
  - `public` : accessibles de partout.
  - `protected` : accessibles dans la classe et ses sous-classes.
  - `private` : accessibles uniquement dans la classe.
- Accessible via l'opérateur `->`.

Attributs (2/2)

```php
<?php
class Person {
    public $name;
    public $age;
}

$person = new Person();

$person->name = "Alice";
$person->age = 30;

echo $person->name . "<br>";
echo $person->age;
```

```java
// Équivalent en Java
public class Person {
    public String name;
    public int age;
}

Person person = new Person();

person.name = "Alice";
person.age = 30;

System.out.println(person.name);
System.out.println(person.age);
```

Méthodes (1/3)

- Fonctions qui définissent les comportements des objets.
- Définies dans la classe avec leur visibilité (`public` , `protected` , `private`).
- Accédées via l'opérateur `->` .
- Le mot-clé `$this` permet de faire référence à l'objet courant.

Méthodes (2/3)

```php
<?php
class Person {
    public $name;
    public $age;

    public function greet() {
        return "Hi, my name is " .$this->name . " and I am " . $this->age . " years old.";
    }
}

$person = new Person();

$person->name = "Alice";
$person->age = 30;

echo $person->greet();
```

Méthodes (3/3)

```
// Équivalent en Java
public class Person {
    public String name;
    public int age;

    public String greet() {
        return "Hi, my name is " + this.name + " and I am " + this.age + " years old.";
    }
}

Person person = new Person();

person.name = "Alice";
person.age = 30;

System.out.println(person.greet());
```

Encapsulation (1/3)

- Protection des données des objets en limitant l'accès direct (avec `public`, `protected` et `private`).
- Permet de contrôler l'accès aux données et de garantir l'intégrité des objets.
- Utilisation de méthodes pour accéder (appelées *"getters"*) et modifier les attributs (appelées *"setters"* en anglais).

Encapsulation (2/3)

```php
<?php
class Person {
    private $name; // Attribut privé
    private $age; // Attribut privé

    public function setName($name) {
        if (strlen($name) < 3) {
            return "Name must be at least 3 characters long.";
        }

        $this->name = $name;
    }
```

```
    public function getName() {
        return $this->name;
    }

    public function setAge($age) {
        if ($age < 0) {
            return "Age cannot be negative.";
        }

        $this->age = $age;
    }

    public function getAge() {
        return $this->age;
    }
}
```

```php
$person = new Person();

$error = $person->setName("Alice");

if (!empty($error)) {
        echo $error . "<br>";
}

$error = $person->setAge(30);

if (!empty($error)) {
        echo $error . "<br>";
}

echo $person->getName() . "<br>"; // Affiche "Alice"
echo $person->getAge() . "<br>";  // Affiche 30
```

```php
$error = $person->setName("AS");

if (!empty($error)) {
    echo $error . "<br>";
}

$error = $person->setAge(-1);

if (!empty($error)) {
    echo $error . "<br>";
}

$person->name = "Bob"; // Erreur : l'attribut est privé
```

Encapsulation (3/3)

```
// Équivalent en Java
public class Person {
    private String name; // Attribut privé
    private int age; // Attribut privé

    public String setName(String name) {
        if (name.length() < 3) {
            return "Name must be at least 3 characters long.";
        }

        this.name = name;

        return null;
    }
```

```
    public String getName() {
        return this.name;
    }

    public String setAge(int age) {
        if (age < 0) {
            return "Age cannot be negative.";
        }

        this.age = age;

        return null;
    }

    public int getAge() {
        return this.age;
    }
}
```

```
    public String getName() {
        return this.name;
    }

    public String setAge(int age) {
        if (age < 0) {
            return "Age cannot be negative.";
        }

        this.age = age;

        return null;
    }

    public int getAge() {
        return this.age;
    }
}
```

```java
Person person = new Person();

String error = person.setName("Alice");

if (error != null) {
    System.out.println(error);
}

error = person.setAge(30);

if (error != null) {
    System.out.println(error);
}

System.out.println(person.getName()); // Affiche "Alice"
System.out.println(person.getAge());  // Affiche 30
```

```
error = person.setName("AS");

if (error != null) {
    System.out.println(error);
}

error = person.setAge(-1);

if (error != null) {
    System.out.println(error);
}

person.name = "Bob"; // Erreur : l'attribut est privé
```

Constructeurs et destructeurs (1/3)

- **Constructeur** : méthode spéciale appelée lors de la création d'un objet. Permet d'initialiser les attributs de l'objet.
- **Destructeur** : méthode spéciale appelée lors de la destruction d'un objet pour libérer les ressources (rarement utilisé en PHP).

Constructeurs et destructeurs (2/3)

```php
<?php
class Person {
    private $name;
    private $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

    public function __destruct() {
        echo $this->name . " is being destroyed.<br>";
    }

    public function greet() {
        return "Hi, my name is " . $this->name . " and I am " . $this->age . " years old.";
    }
}
```

```php
$alice = new Person("Alice", 30);
$bob = new Person("Bob", 25);
$evelyn = new Person("Evelyn", 40);

echo $alice->greet() . "<br>";
echo $bob->greet() . "<br>";
echo $evelyn->greet() . "<br>";

// L'objet `$bob` a maintenant la valeur `null`.
// L'objet est donc détruit et le destructeur est appelé.
$bob = null;

// L'objet `$evelyn` référence maintenant le même objet que `$alice`.
// L'objet `$evelyn` n'est plus utilisé et est donc détruit.
$evelyn = $alice;

// L'objet `$alice` sera automatiquement détruit à la fin du script.
// Son destructeur sera appelé.
```

Constructeurs et destructeurs (3/3)

Pas de destructeur explicite comme en PHP.

```java
// Équivalent en Java
public class Person {
    private String name;
    private int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String greet() {
        return "Hi, my name is " + this.name + " and I am " + this.age + " years old.";
    }
}
```

```java
Person alice = new Person("Alice", 30);
Person bob = new Person("Bob", 25);
Person evelyn = new Person("Evelyn", 40);

System.out.println(alice.greet());
System.out.println(bob.greet());
System.out.println(evelyn.greet());

// En Java, il n'y a pas de destructeur explicite.
// Mais tout objet avec une valeur `null` est éligible pour le garbage collector.
bob = null;

// L'objet `evelyn` référence maintenant le même objet que `alice`.
// L'objet `evelyn` n'est plus utilisé et sera éligible pour le garbage collector.
evelyn = alice;

// L'objet `alice` sera éligible pour le garbage collector à la fin du programme.
```

Constantes (1/3)

- Les constantes sont des valeurs qui ne changent pas pendant l'exécution du programme.
- Elles peuvent être définies dans une classe.
- Accessibles via l'opérateur `::` (opérateur de résolution de portée).
- En majuscules par convention (par exemple, `MA_CONSTANTE`).

Constantes (2/3)

```php
<?php
class Person {
    const ROLE_MANAGER = 'Manager';
    const ROLE_DEVELOPER = 'Developer';
    const ROLE_DESIGNER = 'Designer';
    const ROLE_EMPLOYEE = 'Employee';

    private $name;
    private $role;

    public function __construct($name, $role) {
        $this->name = $name;
        $this->role = $role;
    }

    public function greet() {
        return "Hi, my name is " . $this->name . ". I work as a " . $this->role . " at my company.";
    }
}
```

```php
$alice = new Person("Alice", Person::ROLE_DEVELOPER);
$bob = new Person("Bob", Person::ROLE_MANAGER);
$evelyn = new Person("Evelyn", Person::ROLE_DESIGNER);

// Affiche "Hi, my name is Alice. I work as a Developer at my company."
echo $alice->greet() . "<br>";

// Affiche "Hi, my name is Bob. I work as a Manager at my company."
echo $bob->greet() . "<br>";

// Affiche "Hi, my name is Evelyn. I work as a Designer at my company."
echo $evelyn->greet();
```

Constantes (3/3)

```java
// Équivalent en Java
public class Person {
    public static final String ROLE_MANAGER = "Manager";
    public static final String ROLE_DEVELOPER = "Developer";
    public static final String ROLE_DESIGNER = "Designer";
    public static final String ROLE_EMPLOYEE = "Employee";

    private String name;
    private String role;

    public Person(String name, String role) {
        this.name = name;
        this.role = role;
    }

    public String greet() {
        return "Hi, my name is " + this.name + ". I work as a " + this.role + " at my company.";
    }
}
```

```java
Person alice = new Person("Alice", Person.ROLE_DEVELOPER);
Person bob = new Person("Bob", Person.ROLE_MANAGER);
Person evelyn = new Person("Evelyn", Person.ROLE_DESIGNER);

// Affiche "Hi, my name is Alice. I work as a Developer at my company."
System.out.println(alice.greet());

// Affiche "Hi, my name is Bob. I work as a Manager at my company."
System.out.println(bob.greet());

// Affiche "Hi, my name is Evelyn. I work as a Designer at my company."
System.out.println(evelyn.greet());
```

Conclusion

- La POO permet de créer des programmes modulaires et réutilisables.
- Améliore la lisibilité, la réutilisabilité et la maintenabilité du code.
- PHP prend en charge la POO et propose toutes les fonctionnalités nécessaires pour créer des classes et des objets.

Questions

Est-ce que vous avez des questions ?

Feedback

Le formulaire de feedback vous **permet de partager votre retour** sur l'unité d'enseignement *"ProgServ1"*.

Il ne prend **que quelques minutes** et est **anonyme**. Vous pouvez aussi y **demander un/des cours d'appui**.

Les résultats seront discutés au prochain cours. **Merci beaucoup !**

À vous de jouer !

- (Re)lire le support de cours.
- Réaliser le mini-projet.
- Faire les exercices.
- Poser des questions si nécessaire.
- Partager vos retours à l'aide du formulaire de feedback.

Entraidez-vous si vous avez des difficultés !

Sources (1/2)

- Illustration principale par Richard Jacobs sur Unsplash
- Illustration par Aline de Nadai sur Unsplash
- Illustration par Eric Prouzet sur Unsplash
- Illustration par Thomas Le sur Unsplash
- Illustration par Ussama Azam sur Unsplash
- Illustration par Feliphe Schiarolli sur Unsplash
- Illustration par Kenny Eliason sur Unsplash
- Illustration par Pearse O'Halloran sur Unsplash

Sources (2/2)

- Illustration par Birmingham Museums Trust sur Unsplash
- Illustration par Erol Ahmed sur Unsplash
- Illustration par Scott Blake sur Unsplash
- Illustration par Lluvia Morales sur Unsplash
- Illustration par Nikita Kachanovsky sur Unsplash