

TP PHP POO

# COMPTE RENDU

LIEN SITE

[HTTPS://POOACADEMY.ENZOTANG.FR/](https://pooacademy.enzotang.fr/)

LIEN GITHUB

[HTTPS://GITHUB.COM/ENZOTNG/POO-ACADEMY](https://github.com/ENZOTNG/POO-ACADEMY)

**PRÉSENTÉ PAR  
ENZO TANG**

**PRÉSENTÉ À  
M. BEN AMOR**

**DATE DE REMISE  
15/04/2023**

The background is a solid dark blue. In the top-left corner, there are several overlapping geometric shapes: a dark blue rounded rectangle, a light blue rounded rectangle, and a small dark blue circle. In the bottom-right corner, there are more overlapping shapes: a dark blue rounded rectangle, a light blue rounded rectangle, and a small dark blue circle. The text "EXERCICE 1" is centered in the middle of the page in a bold, white, sans-serif font.

# EXERCICE 1

# EXERCICE 1

03

Dans cet exercice, nous avons créé une classe Ville qui représente une ville avec les propriétés nom et département. La classe a également une méthode afficher qui affiche "La ville X est dans le département Y".

Voici le code de l'exercice :

```
<?php
class Ville
{
    public $nom;
    public $departement;

    public function afficher()
    {
        echo '<p>La ville de ' . $this->nom . ' est dans le département : ' . $this->departement . '</p>';
    }
}

$ville1 = new Ville();
$ville1->nom = "Nantes";
$ville1->departement = "Loire Atlantique";
$ville1->afficher();

$ville2 = new Ville();
$ville2->nom = "Lyon";
$ville2->departement = "Rhône";
$ville2->afficher();
?>
```

Nous avons créé deux objets Ville, \$ville1 et \$ville2, et avons affecté leurs propriétés en utilisant la notation fléchée. Ensuite, nous avons appelé la méthode afficher() pour chaque objet, qui affiche le nom de la ville et son département.

Voici le résultat affiché à l'écran :

## Exercice 1

La ville de Nantes est dans le département : Loire Atlantique

La ville de Lyon est dans le département : Rhône

Abstract geometric shapes in the top-left corner, including a dark blue rounded rectangle, a light blue circle, and a dark blue circle.

# EXERCICE 2

Abstract geometric shapes in the bottom-right corner, including a dark blue rounded rectangle, a light blue circle, and a dark blue circle.

# EXERCICE 2

05

Le but de cet exercice est de modifier la classe Ville précédente en ajoutant un constructeur. Le constructeur permet de simplifier la création d'un objet et l'affectation de ses propriétés.

Le code de la classe VilleConstructeur est le suivant :

```
class VilleConstructeur
{
    public $nomConstructeur;
    public $departementConstructeur;

    public function __construct($nomConstructeur, $departementConstructeur)
    {
        $this->nomConstructeur = $nomConstructeur;
        $this->departementConstructeur = $departementConstructeur;
    }

    public function afficher()
    {
        echo '<p>La ville de ' . $this->nomConstructeur . ' est dans le département : ' . $this->departementConstructeur . '</p>';
    }
}
```

Dans le constructeur de la classe, on peut voir que les deux propriétés sont initialisées directement avec les paramètres passés à la fonction. La méthode afficher est également présente, comme dans la classe précédente.

Ensuite, on crée deux objets de type VilleConstructeur, un pour Nantes et un pour Lyon, en passant les propriétés en paramètres du constructeur :

```
$ville_constructeur1 = new VilleConstructeur("Nantes", "Loire Atlantique");
$ville_constructeur1->afficher();

$ville_constructeur2 = new VilleConstructeur("Lyon", "Rhône");
$ville_constructeur2->afficher();
```

On peut ensuite appeler la méthode `afficher` pour chaque objet pour afficher la phrase correspondante.

Voici le résultat affiché à l'écran :

```
Exercice 2
```

```
La ville de Nantes est dans le département : Loire Atlantique
```

```
La ville de Lyon est dans le département : Rhône
```

On peut voir que le constructeur permet de simplifier la création d'objets et d'affecter directement leurs propriétés lors de l'instanciation. Cela permet également d'éviter la répétition de code pour chaque objet créé, puisque l'initialisation des propriétés est effectuée directement dans le constructeur.



# EXERCICE 3

# EXERCICE 3

08

Pour l'exercice 3, nous devons créer une classe représentant une personne avec les propriétés nom, prénom et adresse. La classe doit également avoir un constructeur et un destructeur, une méthode `getPersonne()` qui renvoie les coordonnées complètes de la personne, ainsi qu'une méthode `setAdresse()` qui permet de modifier l'adresse de la personne. Nous allons créer des objets `personne` et utiliser toutes les méthodes de la classe.

Voici le code correspondant à cet exercice :

```
class Personne {
    public $nomPers;
    public $prenomPers;
    public $adressePers;
    public $message;

    public function __construct($nom, $prenom, $adresse) {
        $this->nomPers = $nom;
        $this->prenomPers = $prenom;
        $this->adressePers = $adresse;
    }

    public function __destruct() {
        $this->message = "L'objet a été détruit.";
    }

    public function getPersonne() {
        return "Nom : " . $this->nomPers . ", Prénom : " . $this->prenomPers . ", Adresse : " . $this->adressePers;
    }

    public function setAdresse($nouvelleAdresse) {
        $this->adressePers = $nouvelleAdresse;
    }
}

$pers1 = new Personne("Tang", "Enzo", "3 Rue du Commerce");
echo '<p>' . $pers1->getPersonne() . '</p>';
$pers1->setAdresse("5 Avenue de la Liberté");
echo '<p>' . $pers1->getPersonne() . '</p>';
if (isset($pers1->message)) {
    echo '<p>' . $pers1->message . '</p>';
}
```



# EXERCICE 3

09

Dans ce code, nous avons créé la classe **Personne** avec les propriétés **\$nomPers**, **\$prenomPers** et **\$adressePers**. Nous avons également créé un constructeur qui prend en paramètres les valeurs initiales de ces propriétés et les affecte aux attributs correspondants. Nous avons également créé un destructeur qui affiche un message lorsque l'objet est détruit.

La méthode **getPersonne()** retourne une chaîne de caractères contenant toutes les coordonnées de la personne, et la méthode **setAdresse()** permet de modifier l'adresse de la personne.

Ensuite, nous avons créé un objet **\$pers1** de type **Personne** avec les valeurs initiales "Tang", "Enzo" et "3 Rue du Commerce". Nous avons affiché les coordonnées de cette personne en utilisant la méthode **getPersonne()**, puis nous avons modifié son adresse en utilisant la méthode **setAdresse()** et réaffiché ses coordonnées avec la nouvelle adresse.

Voici le résultat affiché à l'écran :

## Exercice 3

Nom : Tang, Prénom : Enzo, Adresse : 3 Rue du Commerce

Nom : Tang, Prénom : Enzo, Adresse : 5 Avenue de la Liberté



# EXERCICE 4

# EXERCICE 4

11

L'objectif de cet exercice était de créer une classe appelée "Form" représentant un formulaire HTML, avec des méthodes pour ajouter des éléments tels que des zones de texte et des boutons d'envoi. La méthode "getform" devait retourner tout le code HTML de création du formulaire.

Le code de la classe "Form" a été écrit de la manière suivante :

```
class Form
{
    protected $html;

    public function __construct()
    {
        $this->html = '<form>';
        $this->html .= '<fieldset>';
    }

    public function settext($name, $placeholder)
    {
        $this->html .= '<input type="text" name="' . $name . '" placeholder="' . $placeholder . '">';
    }

    public function setsubmit($value)
    {
        $this->html .= '<input class="boutonGeneral" type="submit" value="' . $value . '">';
    }

    public function getform()
    {
        $this->html .= '</fieldset>';
        $this->html .= '</form>';
        return $this->html;
    }
}
```

# EXERCICE 4

12

La méthode "settext" permet d'ajouter une zone de texte, en spécifiant les attributs "name" et "placeholder". La méthode "setsubmit" permet d'ajouter un bouton d'envoi, en spécifiant la valeur du bouton. La méthode "getform" retourne tout le code HTML de création du formulaire.

Nous avons créé un objet de la classe "Form" avec l'instruction `$form1 = new Form();`.

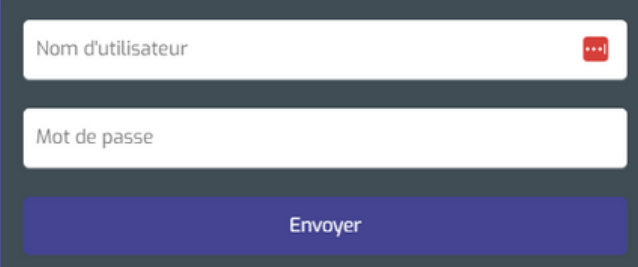
Nous avons ensuite ajouté deux zones de texte et un bouton d'envoi avec les instructions suivantes :

```
$form1->settext("nom", "Nom d'utilisateur");  
$form1->settext("mdp", "Mot de passe");  
$form1->setsubmit("Envoyer");
```

Nous avons enfin affiché le formulaire en appelant la méthode "getform" et en utilisant la fonction "echo" : `echo $form1->getform();`.

Voici le résultat affiché à l'écran :

Nous pouvons voir que le formulaire a été créé avec succès, avec deux zones de texte et un bouton d'envoi. Le code HTML généré correspond bien aux spécifications de l'exercice.



The screenshot shows a web form with a dark blue border. It contains two white text input fields. The first field is labeled "Nom d'utilisateur" and has a red eye icon on the right. The second field is labeled "Mot de passe". Below the input fields is a dark blue button with the text "Envoyer" in white.

The background is a solid dark blue. In the top-left and bottom-right corners, there are abstract geometric shapes in a lighter blue and white. These shapes include rounded rectangles, circles, and lines, some of which are overlapping. The text 'EXERCICE 5' is centered in the middle of the page in a white, bold, sans-serif font.

# EXERCICE 5

Le but de cet exercice était de créer une classe Form2 qui hérite de la classe Form précédente. La nouvelle classe doit ajouter deux nouvelles méthodes : setradio() pour ajouter des boutons radio et setcheckbox() pour ajouter des cases à cocher. Les paramètres de ces méthodes doivent correspondre aux attributs des éléments HTML correspondants. La méthode getform() doit également être modifiée pour inclure les boutons radio et les cases à cocher dans le formulaire HTML.

```
class Form2 extends Form
{
    public function setradio($name, $value, $label)
    {
        $this->html .= '<label><input type="radio" name="' . $name . '" value="' . $value . '"> ' . $label . '</label><br>';
    }

    public function setcheckbox($name, $value, $label)
    {
        $this->html .= '<label><input type="checkbox" name="' . $name . '" value="' . $value . '"> ' . $label . '</label><br>';
    }
}

$form2 = new Form2();
$form2->settext("nom", "Nom d'utilisateur");
$form2->settext("mdp", "Mot de passe");
$form2->setradio("genre", "homme", "Homme");
$form2->setradio("genre", "femme", "Femme");
$form2->setcheckbox("newsletter", "1", "S'abonner à la newsletter");
$form2->setsubmit("Envoyer");
echo $form2->getform();
```

La classe Form2 hérite de la classe Form précédente, donc elle contient tous les éléments définis dans la classe Form. Les nouvelles méthodes setradio() et setcheckbox() sont ajoutées à la classe Form2 pour permettre l'ajout de boutons radio et de cases à cocher. Ces méthodes prennent les mêmes paramètres que les méthodes correspondantes dans la classe Form.

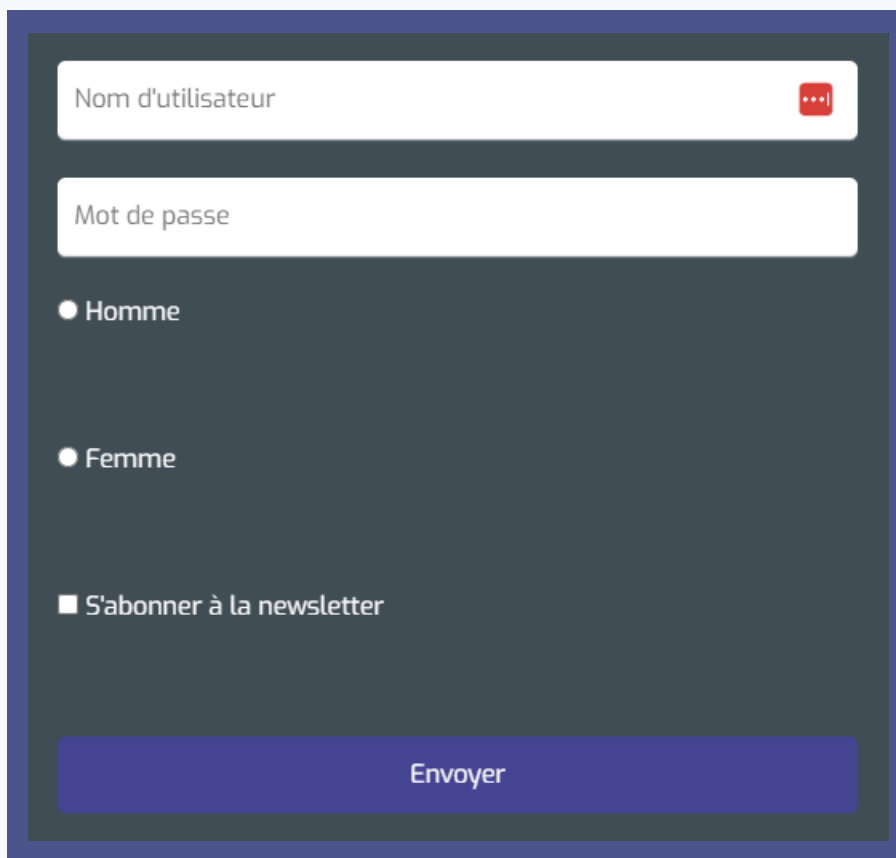
La méthode getform() de la classe Form2 est modifiée pour inclure les boutons radio et les cases à cocher dans le formulaire HTML. La méthode getform() de la classe Form2 appelle la méthode getform() de la classe parent (Form) pour obtenir le formulaire HTML de base, puis ajoute les boutons radio et les cases à cocher.

# EXERCICE 5

15

Dans le script principal, un objet de la classe Form2 est créé et les méthodes appropriées sont appelées pour ajouter des zones de texte, des boutons radio et des cases à cocher. La méthode setsubmit() est ensuite appelée pour ajouter un bouton d'envoi. Enfin, la méthode getform() est appelée pour afficher le formulaire HTML complet.

Voici le résultat affiché à l'écran :



The image shows a web form with a dark blue border. Inside, there is a light gray background. The form contains the following elements:

- A text input field labeled "Nom d'utilisateur" with a red "..." icon on the right.
- A text input field labeled "Mot de passe".
- Two radio buttons: "Homme" (selected) and "Femme".
- A checkbox labeled "S'abonner à la newsletter".
- A large blue button labeled "Envoyer" at the bottom.



# SYNTHÈSE



Pour le TP de Programmation Orientée Objet, j'ai effectué plusieurs exercices qui m'ont permis de mieux comprendre les concepts clés de ce paradigme de programmation.

Comme vous avez pu le constater, pour chaque exercice j'ai inclus une capture d'écran du code et du résultat obtenu, ainsi qu'une explication détaillée du processus.

En ce qui concerne la synthèse de ce que j'ai compris de la programmation orientée objet, j'ai découvert que cette approche consiste à modéliser les données et les actions sous forme d'objets, qui interagissent entre eux pour accomplir des tâches. Les objets sont caractérisés par des attributs et des méthodes, qui définissent leur état et leur comportement respectivement.

En ce qui concerne le lien entre la programmation orientée objet et le modèle MVC (Modèle-Vue-Contrôleur), j'ai compris que ce dernier est un modèle de conception qui permet de séparer les données, l'interface utilisateur et la logique de contrôle en trois composants distincts. Cette approche permet de faciliter la maintenance et l'évolutivité des applications, en réduisant la dépendance entre les différents composants.

Dans le cadre du TP, nous avons pu expérimenter les concepts de la POO en créant des classes pour représenter des objets concrets (villes, personnes, formulaires), ainsi qu'en utilisant l'héritage pour éviter la duplication de code. Nous avons également vu comment utiliser la POO pour créer des formulaires personnalisés en utilisant une classe Form de base et en la spécialisant avec une classe Form2 qui ajoute des fonctionnalités supplémentaires.

En conclusion, l'approche orientée objet permet également de mieux organiser le code en utilisant des classes, ce qui facilite la compréhension et la réutilisation du code.