PHP

OOP

Dans ce module

- OOP ?
- OOP : Encapsulation
- Propriété : attributs et constantes
- Propriété : fonctions
- Propriété : utilisation interne/externe
- Typage fort des attributs (PHP7.4)
- Méthodes magiques
- Le constructeur
- Paramètres de classe
- Pratique

OOP ? (Object-Oriented Programming)

- **Problème** : Objet distinct (ex: un utilisateur, un article en vente, un achat) et répétitions de structure de données et de leur comportement (ex: données des articles en vente et comportement en cas de ventes).
- **Solution** : Un moule (Class) pour créer des objets (Object)

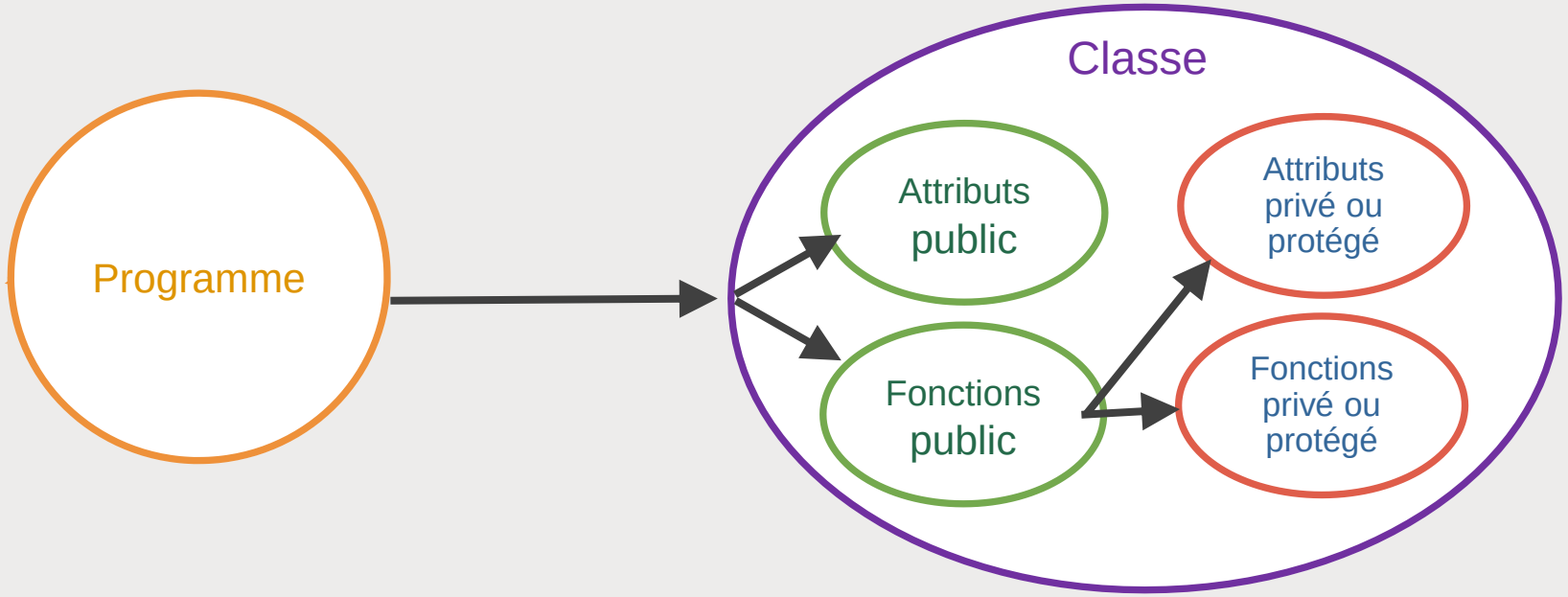
OOP ? (Object-Oriented Programming)

- En informatique, une classe est un type complexe à part entière.
- Comme pour les fonctions, une classe doit porter un nom (première lettre majuscule par convention) et peut avoir des paramètres.
- Mais contrairement au fonction, une classe est un type plus complexe regroupant des propriétés comme suit :
  - *Une propriété peut être une variable ou une fonction (On parlera alors de **variables de classe** et **fonctions de classe**).*
  - *Une propriété doit avoir un "droit d'accès" (**scope**) : public, privé ou protégé*
  - *Une propriété a 1 état parmi 2 statuts : **dynamique ou statique***

```php
class Animal {
  //
}

$cat = new Animal();
$dog = new Animal();
```

OOP : Encapsulation



Propriété : variables/attributs de classe

```php
class Animal {
    public $name; // Public
    public $type; // Public
    protected $age; // Protected
    private $gender; // Private
}
```

```php
$animal = new Animal();
$animal->name = 'Jango';
$animal->type = 'dog';
$animal->age = 3; // ! ACCES ERROR !
$animal->gender = 'male'; // ! ACCES ERROR !
```

```php
$dog = new Animal();
$dog->type = 'dog';

$cat = new Animal();
$cat->type = 'cat';

echo $dog->type; // dog
echo $cat->type; // cat
```

Propriété : constantes et attributs statiques

```php
class Animal {
    const TYPES = ['dog', 'cat']; // Constant

    public static $familly = ’Mammifère'; // Static

    public $type; // Public
}
```

```php
$dog = new Animal();
$dog->type = 'dog';
echo $dog->type; // dog
echo $dog::$familly; // Mammifère
var_dump($dog::TYPES); // ['dog', 'cat']

$cat = new Animal();
$cat->type = 'cat';
echo $cat->type; // cat
echo $cat::$familly; // Mammifère
var_dump($cat::TYPES); // ['dog', 'cat']
```

```php
echo Animal->type; // ! SYNTAX ERROR !
echo Animal::$familly // Mammifère
echo Animal::TYPES // ['dog', 'cat']
```

```php
$dog = new Animal();
echo $dog::$familly; // Mammifère

$cat = new Animal();
echo $cat::$familly; // Mammifère

Animal::$familly = 'Animal';
echo $dog::$familly; // Animal
echo $cat::$familly; // Animal

Animal::TYPES = ['rabbit']; // ! ERROR !
```

Propriété : valeur par défaut des attributs

```php
class Animal {
    public $type;
}
```

```php
$animal = new Animal();

var_dump($animal->type); // null

$animal->type = 'dog';
var_dump($animal->type); // (string) dog
```

```php
class Animal {
    public $type = 'animal';
}
```

```php
$animal = new Animal();

var_dump($animal->type); // (string) animal

$animal->type = 'dog';
var_dump($animal->type); // (string) dog
```

Propriété : fonctions et fonctions statiques

```php
class Test {
    public static function imStatic()
    {
        echo 'Static';
    }

    public function hello()
    {
        echo 'Hello !';
    }

    protected function secret()
    {
         echo 'SECRET';
    }
}
```

```php
hello(); // ! ERROR Not found  !

Test::imStatic();

$test = new Test();
$test::imStatic();
$test->hello();
$test->secret(); // ! ACCESS ERROR !
```

Propriété : utilisation interne des attributs

```php
class Animal {
    protected $type = 'animal';

    public function getType()
    {
        return $this->type;
    }

    public function setType($type)
    {
        $this->type = $type;
    }
}
```

→ On appelle cela un "**getter**" (fonction d'obtention d'attribut)

→ On appelle cela un "**setter**" (fonction d'attribution d'attribut)

```php
$animal = new Animal();

$animal->type = 'cat'; // ! ACCESS ERREUR !
echo $animal->type; // ! ACCESS ERREUR !

$animal->setType('cat');
echo $animal->getType();
```

Propriété : utilisation des constantes et attributs statiques

```php
class Order {
  const VAT = [20, 10, 5.5];
  const DEFAULT_VAT = 20;
  protected static $lastVat;

  public static function lastVat()
  {
    return self::$lastVat;
  }

  public static function priceTTC($price, $vat = null)
  {
    $vat = self::goodVat($vat);

    return $price * (1 + $vat/100);
  }

  protected static function goodVat($vat)
  {
    return self::$lastVat = in_array($vat, self::VAT)
      ? $vat
      : self::DEFAULT_VAT;
  }
}
```

```php
echo Order::priceTTC(94.79, 5.5); // 100
echo Order::lastVat(); // 5.5

echo Order::priceTTC(83.33); // 100
echo Order::lastVat(); // 20

$test = new Test();
echo $test::lastVat(); // 20
$test::priceTTC(94.79, 5.5); // 100
echo $test::lastVat(); // 5.5
```

Propriété : Utilités des attributs et fonctions - Exemple

```php
class Animal {
    const TYPES = ['dog', 'cat', 'rabbit'];
    protected $type;

    public function getType()
    {
        return is_null($this->type) ? '' : ucfirst($this->type);
    }

    public function setType($type)
    {
        if (in_array($type, self::TYPES)) {
            $this->type = $type;
            return true;
        }

        return false;
    }
}
```

Propriété : Attributs : Résumé

- Les attributs ont les fonctionnalités suivante :
  - ***Un nom***
  - ***Une valeur*** *("null" par défaut si elle n'est pas définie)*
  - *Un accès* ***(Scope)*** *: "public" (tous), "protected" ou "private" (usage interne à la classe)*
  - ***Un statut :***
    - **dynamique** (par défaut) : différent pour chaque instance/objet créée. Attribut utilisable/modifiable uniquement avec une classe instancié (`$cat = new Animal();`).
    - **statique** (avec l'instruction "static") : partagé entre toutes les instances/objets créées. Utilisable/Modifiable sens instancier la classe (`Animal::$my_var_static;`) ou depuis une classe instancié (`$cat = new Animal();  $cat::$my_var_static;`).
    - **constante** (avec l'instruction "const") : partagé comme un attribut statique mais non modifiable (→ constante).

Propriété : Fonctions : Résumé

- Les fonctions ont les fonctionnalités suivante :
  - ***Un nom***
  - ***Un contenu : { du code }***
  - *Un accès **(Scope)** : "public" (tous), "protected" ou "private" (usage interne à la classe)*
  - ***Un statut :***
    - **dynamique** (par défaut) : utilisable uniquement avec une classe instancié (`$cat = new Animal();`).
    - **statique** (avec l'instruction "static") : partagé entre toutes les instances/objets créées. Utilisable sens instancier la classe (`Animal::my_method_static();`) ou depuis une classe instancié (`$cat = new Animal();  $cat::my_static_static();`).

Typage fort des attributs (PHP 7.4+)

```php
class Animal {
    public string $name;
    public int $age;
}
```

Méthodes magiques

- Les Classes possèdent des méthodes "magique" (fonctions) qui permettent d'ajouter/modifier les comportements de ces dernières dans différents cas ou à certains moment.
- Elles portent toutes un nom préfini par PHP et sont automatique appeler par PHP dans des cas définis si elles existent dans la classe.
- Elles n'existent pas par défaut dans une classe. Il suffit de créer une fonction (= méthode) portant le nom d'une méthode magique pour en créer une. On peut alors ajouter/modifier le comportement dans les cas où PHP appel la méthode magique.
- Elles permettent les comportements les plus complexes d'une classe et sont par conséquent très utilisées par les Frameworks (automatiquement).
- Le méthodes magiques doivent être définies en "public".

→ PHP Magic Method !

Méthode magique : Constructeur

- Le constructeur est une méthode magique appelée lors de l'initialisé/instanciation/création d'un objet de la classe. Elle permet alors de réaliser toutes actions répétitifs voulues au démarrage :
  - *attribution de valeur à des attributs de la classe dont les données sont chargées ailleurs (ex: fichier de configuration php/json à part).*
  - *créer un système de droit d'accès.*
  - *...*
- Il existe aussi son opposé : **le destructeur** qui s'exécute lorsque l'instance de la classe est détruite (unset($objet), fin de script PHP).

```php
class Animal {
    public function __construct()
    {
        // Mon code se lançant à chaque "new Animal()"
    }
}

$animal = new Animal(); // PHP cherche et appel __construct()
```

Paramètres de classe

- Tout comme les fonctions, les classes peuvent avoir des paramètres.
- On les définis dans les paramètres du constructeur d'une classe. Leur comportement est donc identique aux paramètres d'une fonction classique.
- On les utilises lors de l'instanciation de la classe.
- Ils vont servir à initialisé les valeurs d'attributs de la classe ou toutes données de configuration cette dernière (ex: new PDO($dsn, 'user', 'password');).

```php
class Animal {
    protected $name;
    protected $age;

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

```php
$animal = new Animal('Jango');
$animal = new Animal('Jango', 5);
$animal = new Animal(); // ERREUR: $name required
```

Pratique

- Dans un premier temps, recopiez et testez quelques exemples du cours pour vous entrainer.

- Ensuite, le but de l'exercice est de créer plusieurs classes PHP pour un petit e-commerce.
  - Chaque classe représentera un élément de donné distinct des autres.
  - Chaque classe devra être dans fichier distinct qui portera le même nom que la classe.
  - On appellera plus tard ces classes des « Model » (soit le « M » de MVC que l'on étudiera par après).
  - On améliorera les classes dans la seconde partie du cours OOP.

- Dans un fichier "Product.php" créez une classe "Product" et dans un fichier "Order.php" créez une classe "Order".
- Vous testerez l'appel des classes, c'est-à-dire la création de l'instance d'un objet représentant la classe, directement dans le fichier PHP de la classe pour le moment.

Pratique

- Composition de la classe Product :

- Les attributs:
  - (public) label (nom/titre) (à attribuer lors de la création de la classe (=constructeur))
  - (public) description (à attribuer lors de la création de la classe (=constructeur))
  - (public) brand (à attribuer lors de la création de la classe (=constructeur))
  - (protected) priceTTC (prix TTC)
  - (protected) priceHT (prix HT)
  - (protected) vat (TVA en valeur pourcentage, ex: 20, 10, 5.5)
  - (protected) quantity (représente la quantité totale d'un produit dans l'e-commerce)

- Les fonctions/méthodes :
  - Ajoutez des getter/setter pour priceTTC, priceHT, vat et quantity
  - (public) calculPriceHT() (Retourne le prix HT à partir des attributs priceTTC et vat + attribution de la valeur manquante de priceHT).
  - (public) calculPriceTTC() (Retourne le prix TTC à partir des attributs priceHT et vat + attribution de la valeur manquante de priceTTC).
  - (public) totalPriceTTC() (Avec une quantité en paramètre) (Retourne le prix TTC (attribut de la classe) multiplié par une quantité passée en paramètre).

Pratique

- Composition de la classe Order :

- Les attributs:
  - (protected) ref (référence de commande)
  - (protected) userId (ID base de données de l'acheteur pour un usage futur)
  - (protected) priceTTC (prix TTC)
  - (protected) priceHT (prix HT)
  - (protected) totalVat (le montant de la tva, soit priceTTC – priceHT)

- Les fonctions/méthodes :
  - Ajoutez un getter (mais pas de setter) pour tous les attributs
  - (public) generateRef() (cette méthode sans paramètre va créer une référence de commande, soit un nombre entre 1 000 000 et 9 999 999 et l'affecter à l'attribut ref).
  - (public) addProductPriceToOrder(Product $product, int $quantity)
    - ou $product est un objet instancié de la classe Product
    - cette méthode devra utiliser les méthodes de l'objet Product pour ajouter le prix du produit multiplié par la quantité au prix de la commande

Pratique

- Créer une nouvelle classe « ProductsController » dans un sous-dossier « controllers ». Cette Classe va gérer la majorité des actions, liées aux Produits, entre les pages du site et la Classe Product.
- Les méthodes de cette classe seront appelées depuis d'autres script PHP. Cette classe n'est pas à utiliser directement et ne doit en réalité pas être accessible directement depuis le navigateur.

- Les fonctions/méthodes :
  - (public) index()
    - Cette méthode affichera une page contenant toutes la liste des produits.
  - (public) create()
    - Cette méthode affichera la page contenant le formulaire de création de produits.
  - (public) store()
    - Cette méthode devra insérer un produit dans une table « products » d'une base de données.
    - Les données viendront d'un formulaire HTML en utilisant « $_POST » de PHP.
  - (public) delete()
    - Cette méthode devra supprimer un produit de la table « products ».
    - Vous êtes libre de choisir le processus de d'identification du produit à supprimer.

PHP

OOP

Fin du module