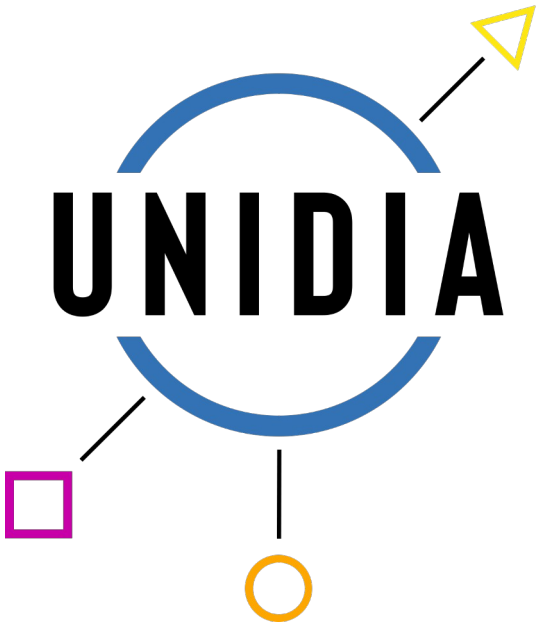



# PHP Objet

# Sommaire

# Sommaire (à terminer)

- Les classes et objets
  - Espaces de nom
  - Classes
  - Le mot class
  - Propriétés
  - Constantes de classe
  - Méthodes
  - L’opérateur nullsafe
  - Le mot clé $this
  - Constructeurs
  - Le mot clé new
  - Destructeurs
  - Accesseurs
  - Afficher un objet
  - Auto-chargement des classes

# Sommaire

- Hiérarchie de classes
  - Héritage
  - Classe abstraite
  - Polymorphisme

# Les classes et les objets

# Les classes et objets

- Espaces de noms

  Dossiers virtuels permettant à encapsuler (c'est-à-dire à isoler) certains éléments de certains autres.

  Les espaces de noms vont notamment permettre d'éliminer les conflits possibles entre deux éléments de même nom.

  - Déclaration :

```php
namespace App\Models;

class Joueur {
```

  - Utilisation :

```php
use App\Models\Joueur;
```

# Les classes et objets

- Classes :
  - Déclaration : class NomClasse { }

```php
class Joueur {
    // déclaration des attributs
    private string $login;
    private string $mdp;
    private int $age;

    // déclaration des méthodes
    public function connecter(): bool {
        return true;
    }

}
```

Nomenclature d'une classe : Commence par une lettre ou un underscore, suivi de n'importe quel(s) chiffre(s) ou lettre(s) ou d'underscores.

# Les classes et objets

- Le mot « class »
  - Utilisation : nomClass::class

```php
$toto = Joueur::class::getNew("toto");
```

Retourne le nom d'une classe. Utile pour obtenir l'espace de nom.

Peut être utilisé sur un objet :

```php
echo $toto::class;
```

# Les classes et objets

- Propriétés

  Variables d'une classe. Appelées ainsi dans la documentation officielle de PHP

  - Déclaration : Modificateur ?type nomPropriete = valeur
    - Modificateur
      - Visibilité :
        - Public : accessible depuis l'instance d'objet
        - Private : accessible à l'intérieur de la classe mais pas depuis l'instance d'objet
        - Protected : accessible à l'intérieur de la classe parente et enfante mais depuis l'instance d'objet
      - Statique : accessible directement dans la classe sans passer par une instance d'objet
      - Readonly : empêche la modification après l'initialisation
    - Type : déclaration de type
    - ? : opérateur nullsafe. Permet de définir que la propriété peut être nulle
    - NomPropriété : déclaration classique
    - Valeur : Cette déclaration peut comprendre une initialisation, mais celle-ci doit être une valeur constante

```php
private string $mdp;
private int $age = 0;
```

# Les classes et objets

- Constantes de classe

  Identiques et non modifiables.

  La visibilité par défaut des constantes de classe est public.

  - Déclaration : const MACONSTANTE = « valeur »
  - Utilisation
    - Au sein de la classe : self::MACONSTANTE
    - A l'extérieur de la classe : nomClasse::MACONSTANTE

# Les classes et objets

- Méthodes

  Fonctions d'une classe.

  - Déclaration : Modificateur function nomMéthode(params) : retour
    - Modificateur :
      - Visibilité :
        - Public : accessible depuis l'instance d'objet
        - Private : accessible à l'intérieur de la classe mais pas depuis l'instance d'objet
        - Protected : accessible à l'intérieur de la classe parente et enfante mais depuis l'instance d'objet
      - Statique : accessible directement dans la classe sans passer par une instance d'objet
      - Readonly : empêche la modification après l'initialisation
    - NomMéthode : déclaration classique
    - Params :
      - Paramètres typés : (int $age)
      - Paramètres par référence : (int &$age)
      - Paramètres à nombre variable (mixed ...$values). Il faudra traiter le paramètre comme un tableau
      - Valeur par défaut des paramètres (int $age = 18)
    - Retour : type de valeur retournée. Pour une procédure, on utilisera le mot clé void.

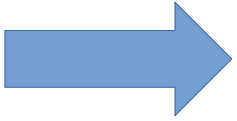
# Les classes et objets

- L’opérateur nullsafe

  Remplace les vérifications conditionnelles de nul : lorsque l'évaluation d'un élément de la chaîne échoue, l'exécution de la chaîne complète est terminée et la chaîne entière évaluée à null.

  Exemple :

```php
$country =  null;

if ($session !== null) {
  $user = $session->user;

  if ($user !== null) {
    $address = $user->getAddress();

    if ($address !== null) {
      $country = $address->country;
    }
  }
}
```



```php
$country = $session?->user?->getAddress()?->country;
```

# Les classes et objets

- Le mot clé « $this »

  Permet d'utiliser les méthodes et les propriétés au sein d'une classe.

# Les classes et objets

- Constructeurs

  Cette méthode est appelée à chaque création d'une nouvelle instance de l'objet afin de permettre son initialisation.

  - Déclaration :

    function __construct(mixed ...$values = ""): void

# Les classes et objets

- Constructeurs

  Depuis la version 8, il est possible de déclarer les attributs et de les initialiser dans le constructeur.

```php
public function __construct(
    // déclaration des attributs
    private ?string $login = null,
    private ?string $mdp = null,
    private ?int $age = null
) {}
```

# Les classes et objets

- Le mot clé « new »
  - Instanciation d'une classe

    Si une chaîne de caractères contenant un nom de classe est utilisée avec new, une nouvelle instance de cette classe sera créée.

```php
$monJoueur = new Joueur();

// Ceci peut également être réalisé avec une variable :
$classeJoueur = 'Joueur';
$monJoueur = new $classeJoueur(); // new Joueur()

$monJoueur = new (Joueur::class);
```

# Les classes et objets

- Le mot clé « new »
  - Création d'objet via une méthode statique

```php
static public function getNew($login)
{
    $joueur = new static;
    $joueur->login = $login;
    return $joueur;
}
```

  - Appel d'une méthode d'un objet nouvellement créé

```php
var_dump((new Joueur())->connecter());
```

# Les classes et objets

- Destructeurs

  La méthode destructeur est appelée dès qu'il n'y a plus de référence sur un objet donné, ou dans n'importe quel ordre pendant la séquence d'arrêt.

  - Déclaration :

```php
public function __destruct() {
    echo "destruct";
}
```

# Les classes et objets

- Accesseurs

  Il est possible d'implémenter des méthodes magiques c'est à dire automatiquement appelées pour accéder ou modifier les propriétés d'un objet :

  - __get() : lire des données depuis des propriétés inaccessibles (protégées ou privées)
  - Implémentation :

```php
public function __get($name)
{
    return match ($name) {
        "login" => $this->login,
        "mdp" => "******",
        "age" => $this->age,
        "connecte" => (bool) $this->connecte,
        default => null
    };

}
```

Utilisation :

```php
$j = new Joueur("Elodie");
echo "login :".$j->login.".";
```

# Les classes et objets

- Accesseurs
  - __set() : écrire des données vers des propriétés inaccessibles (protégées ou privées)
  - Implémentation :

```php
public function __set($name, $value) {

    return match ($name) {

        "mdp" => $this->mdp = password_hash($value, PASSWORD_BCRYPT ),
        default => $this->$name = $value
    };
}
```

Utilisation :

```php
$j = new Joueur("Elodie", "mdp");
$j->mdp = "new mdp";
```

# Les classes et objets

- Afficher un objet

  Il est possible de renvoyer un objet sous forme de chaîne de caractère grâce à la méthode __toString() :

  - Implémentation :

```php
public function __toString() {
    return "Hello ".$this->login." avec comme mot de passe ".$this->mdp;
}
```

  - Utilisation :

```php
$j = new Joueur("Elodie", "mdp");
echo $j;
```

# Les classes et objets

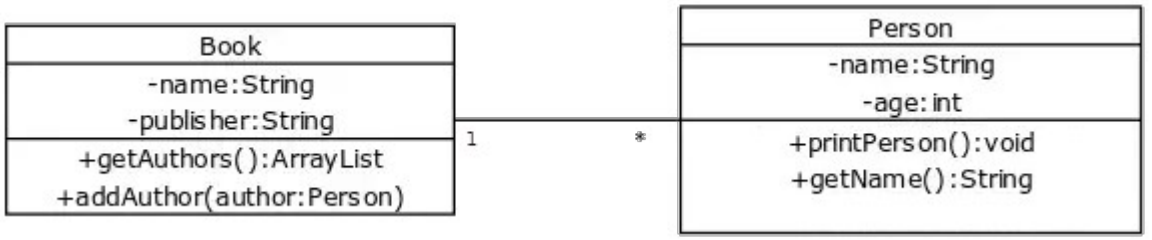
- Auto-chargement de classes
  - spl_autoload_register(). Fonction qui permet de n'inclure que les classes dont on a besoin dans un script. Elle va être appeler automatiquement dès qu'on va essayer d'instancier une classe.

```php
spl_autoload_register(function($classe){
    require 'classes/' .$classe. '.class.php';
});

$pierre = new Admin('Pierre', 'abcdef', 'Sud');
$mathilde = new Admin('Math', 123456, 'Nord');
$florian = new Abonne('Flo', 'flotri', 'Est');
```

# Association entre classes

# Association entre classes

- Relation 1..*
  - UML



# Association entre classes

- Relation 1..*
  - PHP
    - Classe Book

```php
class Book {

  public function __construct(
    // déclaration des attributs
    private ?string $name = null,
    private ?string $publisher = null,
    private ?array $authors = array()
  ) {}

  public function __toString() : string {
    return $this->name . " de : ".implode($this->authors);
  }

  public function addAuthor(Person $author) : void {
    $this->authors[] = $author;
  }

  // ...
}
```

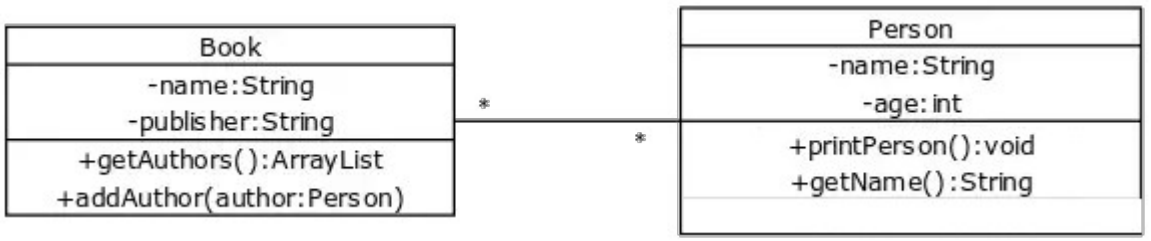
# Association entre classes

- Relation 1..*
  - PHP
    - Classe Person

```php
class Person {

  public function __construct(
    // declaration des attributs
    private ?string $name = null,
    private int $age = 0,
    private ?Book $book = null
  ) {}

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

  // ...
}
```

# Association entre classes

- Relation *..*
  - UML



# Association entre classes

- Relation *..*
  - PHP
    - Classe Book

```php
class Book {

  public function __construct(
    // déclaration des attributs
    private ?string $name = null,
    private ?string $publisher = null,
    private ?array $authors = array()
  ) {}

  public function __toString() : string {
    return $this->name . " de : ".implode($this->authors);
  }

  public function addAuthor(Person $author) : void {
    $this->authors[] = $author;
  }

  // ...
}
```

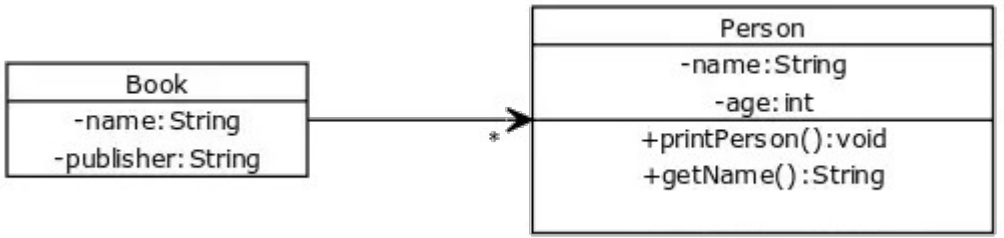
# Association entre classes

- Relation *..*
  - PHP
    - Classe Person

```php
class Person {

  public function __construct(
    // déclaration des attributs
    private ?string $name = null,
    private int $age = 0,
    private ?array $books = array()
  ) {}

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

  // ...
}
```

# Association entre classes

- Relation unidirectionnelle



# Association entre classes

- Relation unidirectionnelle
  - PHP
    - Classe Book

```php
class Book {

  public function __construct(
    // déclaration des attributs
    private ?string $name = null,
    private ?string $publisher = null,
    private ?array $authors = array()
  ) {}

  public function __toString() : string {
    return $this->name . " de : ".implode($this->authors);
  }

  public function addAuthor(Person $author) : void {
    $this->authors[] = $author;
  }

  // ...
}
```

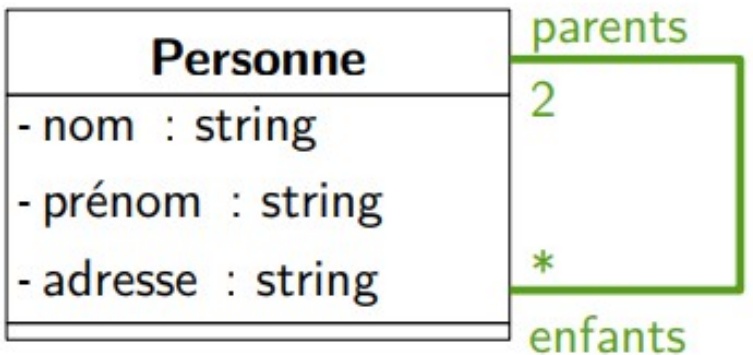
# Association entre classes

- Relation unidirectionnelle
  - PHP
    - Classe Person

```php
class Person {

  public function __construct(
    // déclaration des attributs
    private ?string $name = null,
    private int $age = 0
  ) {}

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

  // ...
}
```

# Association entre classes

- Association réflexive
  - UML



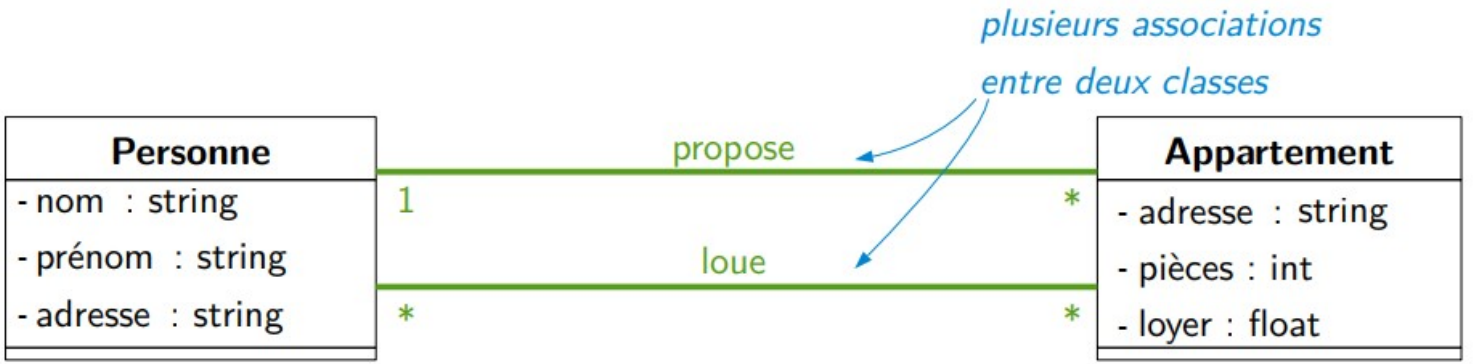
# Association entre classes

- Association réflexive
  - PHP

```php
class Personne {

  public function __construct(
    // déclaration des attributs
    private ?string $nom = null,
    private ?string $prenom = null,
    private ?string $adresse = null,
    private Personne $parent1 = null,
    private Personne $parent2 = null,
    private ?array $les_enfants = array()
  ) {}

}
```

# Association entre classes

- Associations multiples
  - UML



# Association entre classes

- Associations multiples
  - PHP
    - Classe Personne

```php
class Personne {

  public function __construct(
    // déclaration des attributs
    private ?string $nom = null,
    private ?string $prenom = null,
    private ?string $adresse = null,
    private ?array $appartementsLoues = array(),
    private ?array $appartementsPropose = array()
  ) {}

}
```

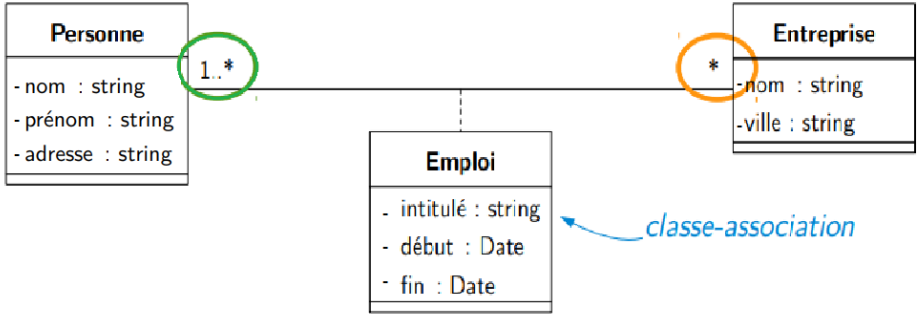
# Association entre classes

- Associations multiples
  - PHP
    - Classe Appartement

```php
class Appartement {

  public function __construct(
    // déclaration des attributs
    private ?string $adresse = null,
    private int $piece = 1,
    private float $loyer = 0.0,
    private ?array $locataires = array(),
    private ?Person $proprietaire = null
    ) {}

}
```

# Association entre classes

- Classe d'association
  - UML



# Association entre classes

- Classe d'association
  - PHP
    - Classe Personne

```php
class Personne {

    public function __construct(
      // déclaration des attributs
      private ?string $nom = null,
      private ?string $prenom = null,
      private ?string $adresse = null,
      private ?array $emplois = array()
    ) {}

}
```

# Association entre classes

- Classe d'association
  - PHP
    - Classe Entreprise

```php
class Entreprise {

    public function __construct(
      // déclaration des attributs
      private ?string $nom = null,
      private ?string $ville = null,
      private ?array $emplois = array();
    ){}
}
```

# Association entre classes

- Classe d'association
  - PHP
    - Classe Emploi

```php
class Emploi {

    public function __construct(
        // déclaration des attributs
        private date $debut,
        private date $fin,
        private Entreprise $entreprise,
        private Personne $employe
    ){}
}
```

# Association entre classes

- Agrégation
  - UML

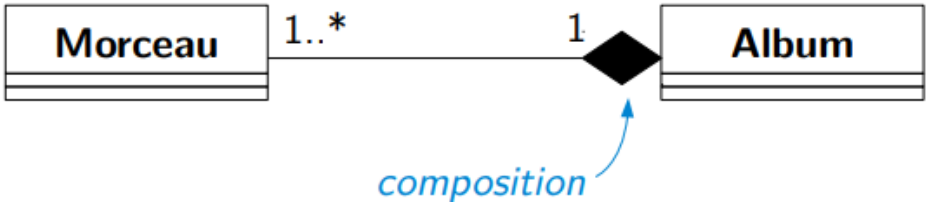
# Association entre classes

- Agrégation
  - PHP

```php
class ListeLecture {
    public function __construct(
        // déclaration des attributs
        private array $morceau
    ){}

}

class Morceau {}
```

# Association entre classes

- Composition
  - UML



# Association entre classes

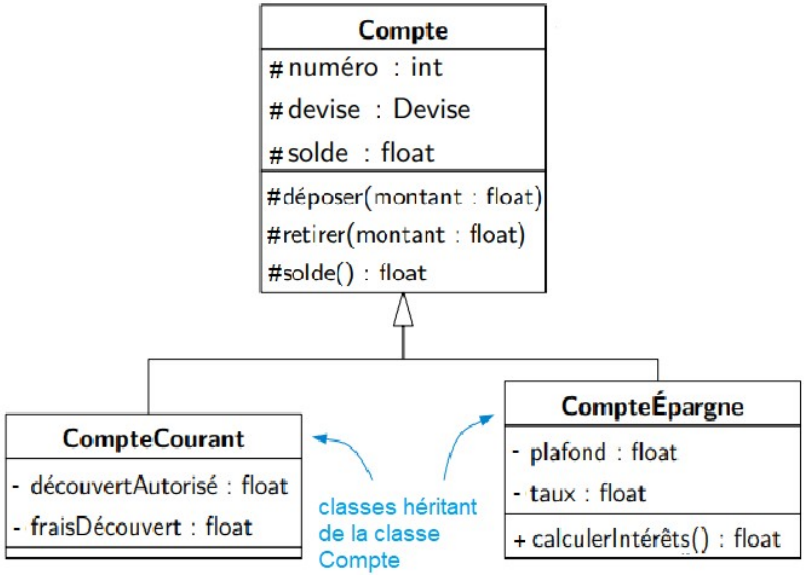
- Composition
  - PHP

```php
class Album {
    public function __construct(
        // déclaration des attributs
        private ?array $morceau = array()
    ){}

}

class Morceau {}
```

# Hiérarchie de classes

# Hiérarchie de classes

- Héritage simple
  - UML



# Hiérarchie de classes

- Héritage simple
  - PHP
    - Classe Compte

```php
class Compte {

    public function __construct(
        // déclaration des attributs
        protected int $numero,
        protected string $devise,
        protected float $solde,
    ){}
}
```

# Hiérarchie de classes

- Héritage simple
  - PHP
    - Classe CompteCourant

```php
class CompteCourant extends Compte {


    public function __construct(
        // declaration des attributs
        int $numero,
        string $devise,
        float $solde,
        private float $decourvertAutorise,
        private float $fraisDecouvert,

    ){
        // parent:: permet d'utiliser une methode de la classe parente
        parent::__construct( $numero, $devise, $solde);

    }
}
```

# Hiérarchie de classes

- Héritage simple
  - PHP
    - Classe CompteEpargne

```php
class CompteEpargne extends Compte {
    public function __construct(
        // declaration des attributs
        int $numero,
        string $devise,
        float $solde,
        private float $plafond,
        private float $taux,

    ){
        // parent:: permet d'utiliser une méthode de la classe parente
        parent::__construct( $numero, $devise, $solde);

    }

}
```

# Hiérarchie de classe

- Traits

  PHP : héritage simple.

  Une classe fille hérite d'une classe mère.

  Solution pour l'héritage multiple :

  - Les traits
    - semblable à une classe,
    - non instanciable,
    - utilisation de méthodes de classe et de classes sans besoin d'héritage.

# Hiérarchie de classe

- Traits
  - Déclaration

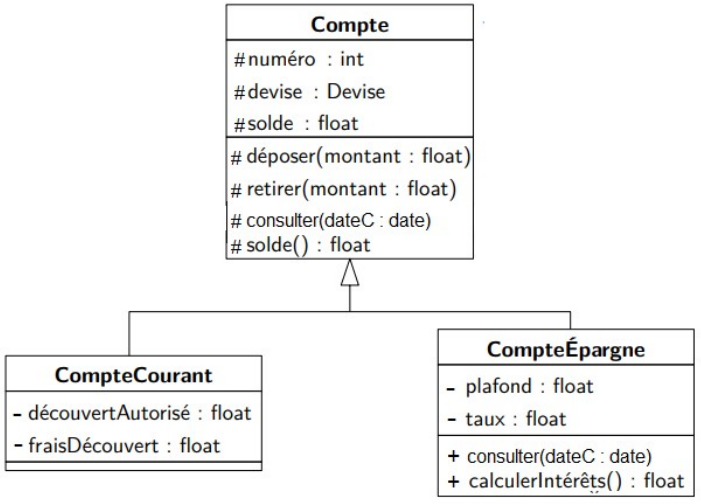
```php
trait School {
  public function learn() {
    echo "I am learning PHP at WayToLearnX!";
  }
}
class Person {
  use School;
}
```

  - Utilisation

```php
$person = new Person();
$person->learn();
```

# Hiérarchie de classes

- Polymorphisme
  - UML



# Hiérarchie de classes

- Polymorphisme
  - PHP
    - Classe Compte

```php
class Compte {

    public function __construct(
        // déclaration des attributs
        protected int $numero,
        protected string $devise,
        protected float $solde
    ){}

    protected function consulter(Date $dateC) {
        // un comportement
    }
}
```

# Hiérarchie de classes

- Polymorphisme
  - PHP
    - Classe CompteCourant

```php
class CompteCourant extends Compte {

    public function __construct(
        // déclaration des attributs
        int $numero,
        string $devise,
        float $solde,
        private float $decourvertAutorise,
        private float $fraisDecouvert

    ){
        // parent:: permet d'utiliser une méthode de la classe parente
        parent::__construct( $numero, $devise, $solde);

    }

    public function consulter(Date $dateC) {
        // un comportement
    }

}
```

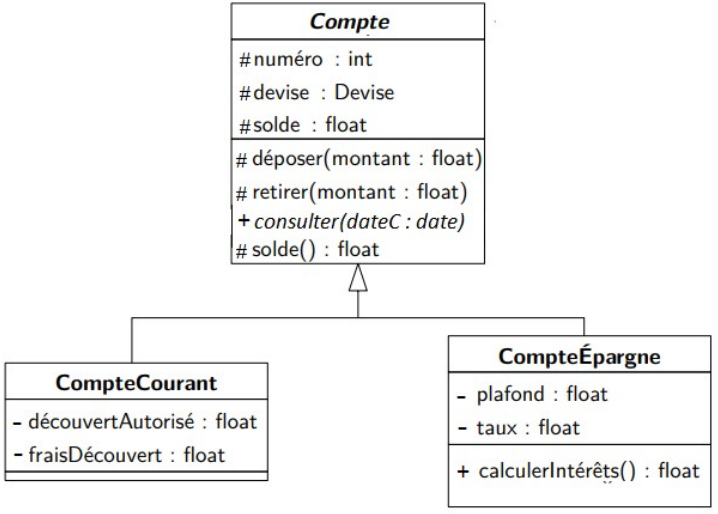
# Hiérarchie de classes

- Polymorphisme
  - PHP
    - Classe CompteEpargne

```php
class CompteEpargne extends Compte {
    public function __construct(
        // declaration des attributs
        int $numero,
        string $devise,
        float $solde,
        private float $plafond,
        private float $taux,

    ){
        // parent:: permet d'utiliser une méthode de la classe parente
        parent::__construct( $numero, $devise, $solde);

    }

}
```

# Hiérarchie de classes

- Classe abstraite
  - UML



# Hiérarchie de classes

- Classe abstraite
  - PHP
    - Classe Compte

```php
abstract class Compte {

    public function __construct(
        // déclaration des attributs
        protected int $numero,
        protected string $devise,
        protected float $solde
    ){}

    abstract public function consulter(Date $dateC) ;
}
```

# Hiérarchie de classes

- Classe abstraite
  - PHP
    - Classe CompteCourant

```php
class CompteCourant extends Compte {

    public function __construct(
        // déclaration des attributs
        int $numero,
        string $devise,
        float $solde,
        private float $decourvertAutorise,
        private float $fraisDecouvert

    ){
        // parent:: permet d'utiliser une méthode de la classe parente
        parent::__construct( $numero, $devise, $solde);

    }

    public function consulter(Date $dateC){
        // un comportement
    }

}
```

# Hiérarchie de classes

- Classe abstraite
  - PHP
    - Classe CompteEpargne

```php
class CompteEpargne extends Compte {
    public function __construct(
        // declaration des attributs
        int $numero,
        string $devise,
        float $solde,
        private float $plafond,
        private float $taux,

    ){
        // parent:: permet d'utiliser une méthode de la classe parente
        parent::__construct( $numero, $devise, $solde);

    }

    public function consulter(Date $dateC){
        // un comportement
    }

}
```

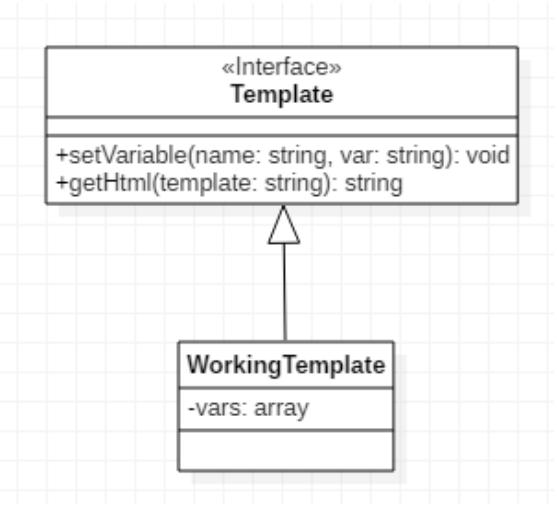
# Hiérarchie de classes

- Le mot clé « final »

  Empêche les classes enfants de redéfinir une méthode ou constante en préfixant la définition avec final. Si la classe elle-même est définie comme finale, elle ne pourra pas être étendue.

# Hiérarchie de classes

- Interface
  - UML



# Hiérarchie de classes

- Interface
  - PHP
    - Interface Template

```php
// Declaration de l'interface 'Template'
interface Template
{
    public function setVariable(string $name, string $var):void;
    public function getHtml(string $template):string;
}
```

# Hiérarchie de classes

- Interface
  - PHP
    - Classe WorkingTemplate

```php
// Implementation de l'interface
// Ceci va fonctionner
class WorkingTemplate implements Template
{
    private $vars = [];

    public function setVariable(string $name, string $var): void
    {
        $this->vars[$name] = $var;
    }

    public function getHtml(string $template): string
    {
        foreach($this->vars as $name => $value) {
            $template = str_replace('{' . $name . '}', $value, $template);
        }

        return $template;
    }
}
```