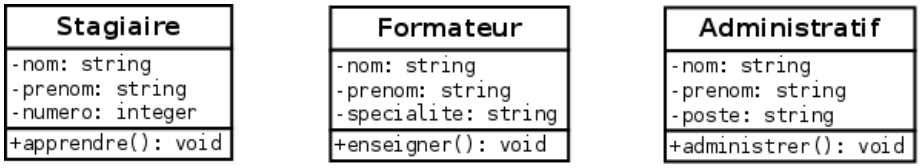
Programmation orientée objet en PHP

Introduction

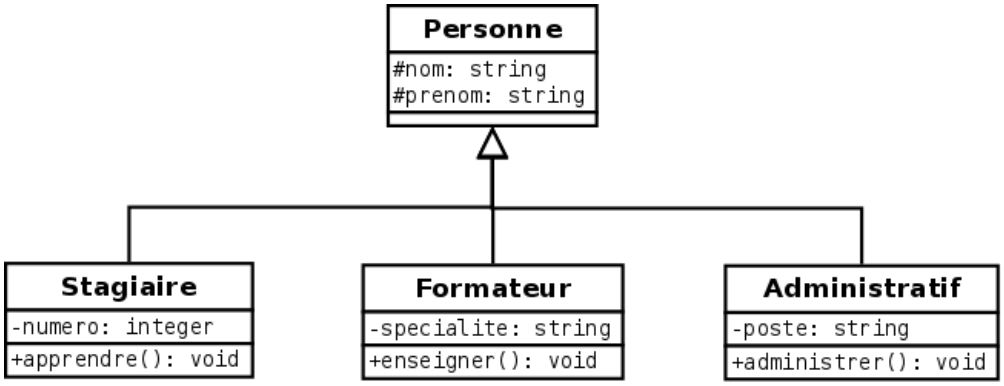
Nous allons créer trois classes :

- une classe Stagiaire possédant les attributs nom, prénom et numéro ainsi qu'une méthode apprendre()
- une classe Formateur possédant les attributs nom, prénom et spécialité ainsi qu'une méthode enseigner()
- une classe Administratif possédant les attributs nom, prénom et poste ainsi qu'une méthode administrer().



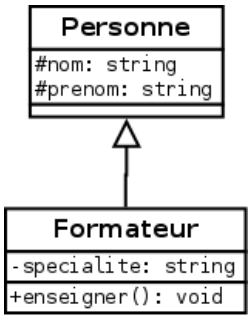
Le Stagiaire, le Formateur et l'Administratif ont tous des attributs communs : le nom et le prénom. Puisque ces trois éléments ont en commun d'être tous des personnes, et comme une personne possède un nom et un prénom, il est normal que le Stagiaire, le Formateur et l'Administratif possède un nom et un prénom. Mais le Stagiaire, le Formateur et l'Administratif ne sont pas de "simples personnes", ils possèdent en plus de la personne des attributs et des compétences propres, ce sont en fait des personnes "spécialisées" : ces entités possèdent un rôle spécial et des caractéristiques propres. En plus du nom et du prénom, le Stagiaire possède un attribut numéro et une méthode apprendre, le Formateur un attribut spécialité et une méthode enseigner, l'Administratif un attribut poste et une méthode administrer. Chacun a son rôle, mais tous sont des personnes, et possèdent donc un nom et un prénom.
Nous pourrions réunir en une classe les attributs et méthodes communes aux entités Stagiaire, Formateur et Administratif : une classe Personne, par exemple, qui posséderait un attribut nom et un attribut prénom. Puis faire en sorte que les classes Stagiaire, Formateur et Administratif récupèrent les attributs et méthodes de la classe Personne. Cela nous éviterait de dupliquer les attributs nom et prénom dans chaque classe. Nous pourrions dire alors que la classe Personne est la classe "mère" ou super-classe des classes Stagiaire, Formateur et Administratif et que les classes Stagiaire, Formateur et Administratif héritent de la classe Personne. La classe Personne contient les attributs, et les méthodes, communs au trois classes Stagiaire, Formateur et Administratif.
La relation qui relie la superclasse, ici Personne, à ses classes filles, ici Stagiaire, Formateur et Administratif, est une relation d'héritage, ou encore une relation is-a ("est un"). Le Formateur est une Personne, il hérite donc de la classe Personne ; de même pour les classes Stagiaire et Administratif. Nous pouvons aussi dire que la classe fille (ou sous-classe) est une spécialisation de la super-classe.

La relation d'héritage permet à la classe fille de posséder les attributs et méthodes accessibles de la classe mère.



La relation d'héritage est représentée par la flèche. Le caractère "#" devant les attributs nom et prénom de Personne signifie que ces attributs ont une visibilité "protected" : ils sont accessibles à la classe elle-même et à ses classes filles. Rien n'empêche une super-classe de posséder des attributs et méthodes privées, mais ces attributs et méthodes ne seront pas accessibles aux sous-classes. Les attributs et méthodes publics sont

accessibles à tous, donc aux sous-classes.



Ce diagramme signifie que la classe Formateur hérite de la classe Personne, que la classe Formateur est une sous-classe ou classe fille de Personne, que la classe Personne est la super-classe ou la classe mère de la classe Personne.

En Java, pour indiquer qu'une classe B hérite d'une classe A, on utilise le mot clé extends dans la déclaration de la classe B.

Personne.php (minimaliste : avec des attributs publics et sans constructeur personnalisé pour l'instant)

```php
<?php

// Personne.php
class Personne {

    public $nom;
    public $prenom;

}
```

Formateur.php (minimaliste aussi : avec un attribut public et sans constructeur personnalisé pour l'instant)

```php
<?php

// Formateur.php
class Formateur extends Personne {

    public $specialite;

    public function enseigner() {
        return 'j\'enseigne';
    }

}
```

Le mot clé extends signifie que Formateur hérite de Personne, ce qui signifie que Formateur possède les attributs accessibles définis dans Personne, c'est-à-dire le nom et le prénom. À partir d'une référence à un objet de type Formateur, nous allons donc pouvoir accéder aux attributs nom, prénom et spécialité.

```php
<?php

// lanceur.php
include './Personne.php';
include './Formateur.php';

$formateur = new Formateur();
$formateur->nom = 'Sparrow';
$formateur->prenom = 'Jack';
$formateur->specialite = 'Développement Logiciel';
echo "nom : $formateur->nom prénom : $formateur->prenom spécialité : $formateur-
>specialite";
```

Dans ce lanceur nous avons créé un objet de type Formateur via le constructeur par défaut et nous avons accédé

directement en lecture et écriture à ses attributs. Nous n'avons donc pas respecté le principe de l'encapsulation (accès aux attributs de l'objet uniquement via des méthodes de l'objet).
Nous allons modifier notre classe Personne pour protéger ses attributs tout en les rendant accessibles à ses classes filles : nous donnerons aux attributs la visibilité protected. Et nous créerons des getters et setters sur ces attributs.

```php
<?php

// Personne.php
class Personne {

    protected $nom;
    protected $prenom;

    public function getNom() {
        return $this->nom;
    }

    public function getPrenom() {
        return $this->prenom;
    }

    public function setNom($nom) {
        $this->nom = $nom;
    }

    public function setPrenom($prenom) {
        $this->prenom = $prenom;
    }

}
```

Nous modifions aussi la classe Formateur pour protéger ces attributs avec une visibilité private et des getters et setters :

```php
<?php

// Formateur.php
class Formateur extends Personne {

    private $specialite;

    public function getSpecialite() {
        return $this->specialite;
    }

    public function setSpecialite($specialite) {
        $this->specialite = $specialite;
    }

    public function enseigner() {
        return 'j\'enseigne';
    }

}
```

Nous allons aussi devoir modifier notre lanceur puisque les attributs ne sont plus accessibles directement. Il faudra maintenant utiliser les getters et setters.

```php
<?php

// lanceur.php
include './Personne.php';
include './Formateur.php';
```

```php

$formateur = new Formateur();
$formateur->setNom('Sparrow');
$formateur->setPrenom('Jack');
$formateur->setSpecialite('Développement Logiciel');
echo "nom : {$formateur->getNom()} prénom : {$formateur->getPrenom()} "
 . "spécialité : {$formateur->getSpecialite()}";
```

Dans les cours précédents nous avons vu que pour initialiser les valeurs des attributs il était très pratique d'utiliser un constructeur personnalisé. Pour initialiser les attributs nom, prénom et spécialité de Formateur nous pourrions donc créer un constructeur.

```php
<?php

// Formateur.php
class Formateur extends Personne {

    private $specialite;

    function __construct($nom, $prenom, $specialite) {
        $this->nom = $nom;
        $this->prenom = $prenom;
        $this->specialite = $specialite;
    }

    public function getSpecialite() {
        return $this->specialite;
    }

    public function setSpecialite($specialite) {
        $this->specialite = $specialite;
    }

    public function enseigner() {
        return 'j\'enseigne';
    }

}
```

Notre lanceur devient :

```php
<?php

// lanceur.php
include './Personne.php';
include './Formateur.php';

$formateur = new Formateur('Sparrow', 'Jack', 'Développement Logiciel');
echo "nom : {$formateur->getNom()} prénom : {$formateur->getPrenom()} "
 . "spécialité : {$formateur->getSpecialite()}";
```

Les attributs nom et prénom sont déclarés dans la classe Personne. Il serait donc intéressant de créer un constructeur personnalisé (ou surchargé) dans la classe Personne qui initialiserait les attributs nom et prénom :

```php
<?php

// Personne.php
class Personne {

    protected $nom;
    protected $prenom;

    function __construct($nom, $prenom) {
        $this->nom = $nom;
```

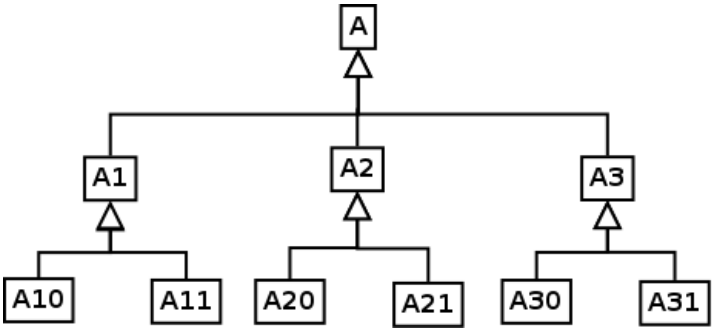
```
        $this->prenom = $prenom;
    }

    public function getNom() {
        return $this->nom;
    }

    public function getPrenom() {
        return $this->prenom;
    }

    public function setNom($nom) {
        $this->nom = $nom;
    }

    public function setPrenom($prenom) {
        $this->prenom = $prenom;
    }

}
```

Nous pourrions donc créer une instance de Personne et l'initialiser avec ce constructeur. Et nous pourrions aussi ré-utiliser ce constructeur dans la classe Formateur : dans le constructeur de la classe fille (Formateur) nous allons appeler le constructeur de la classe mère (Personne). Pour cela nous allons utiliser le mot clé parent et l'opérateur "::".

```
<?php

// Formateur.php
class Formateur extends Personne {

    private $specialite;

    function __construct($nom, $prenom, $specialite) {
        parent::__construct($nom, $prenom);
        $this->specialite = $specialite;
    }

    public function getSpecialite() {
        return $this->specialite;
    }

    public function setSpecialite($specialite) {
        $this->specialite = $specialite;
    }

    public function enseigner() {
        return 'j\'enseigne';
    }

}
```

Le lanceur n'est pas modifié.

En PHP on ne peut hériter que d'une seule classe, mais rien n'empêche une classe d'hériter d'une classe qui elle-même hérite d'une classe, formant ainsi une hiérarchie de classes assez importante :



Les classes A1, A2, A3 héritent directement de A, et possèdent donc les attributs et méthodes accessibles de A.
Les classes A10 et A11 héritent directement de A1, et possèdent donc les attributs et méthodes accessibles de A1, et donc de A puisque A1 hérite de A.

Fin

NAVIGATION

Accueil
- Ma page
  - Pages du site
  - Mon profil
  - Cours actuel
    - POO PHP
      - Participants
      - Généralités
      - La programmation orientée objet : premiers pas
      - L'héritage
      - **Introduction**
      - T.P. héritage
      - Redéfinition de méthodes
      - T.P. redéfinition
      - Les classes abstraites
      - T.P. véhicule
      - Le mot clé final
      - T.P. employés
      - Les interfaces
      - Le typage
      - Les namespaces
      - Les exceptions
      - Les bases de données avec PDO
      - Les tests avec PHPUnit
      - Petite application version 2
      - Petitie application version 3
  - Mes cours

ADMINISTRATION

Administration du cours

Réglages de mon profil

Connecté sous le nom « Arnaud Lemais » (Déconnexion)
POO PHP