Programmation orientée objet

Cours 3 – Encapsulation et accessibilité

Constructeurs et injection de dépendance

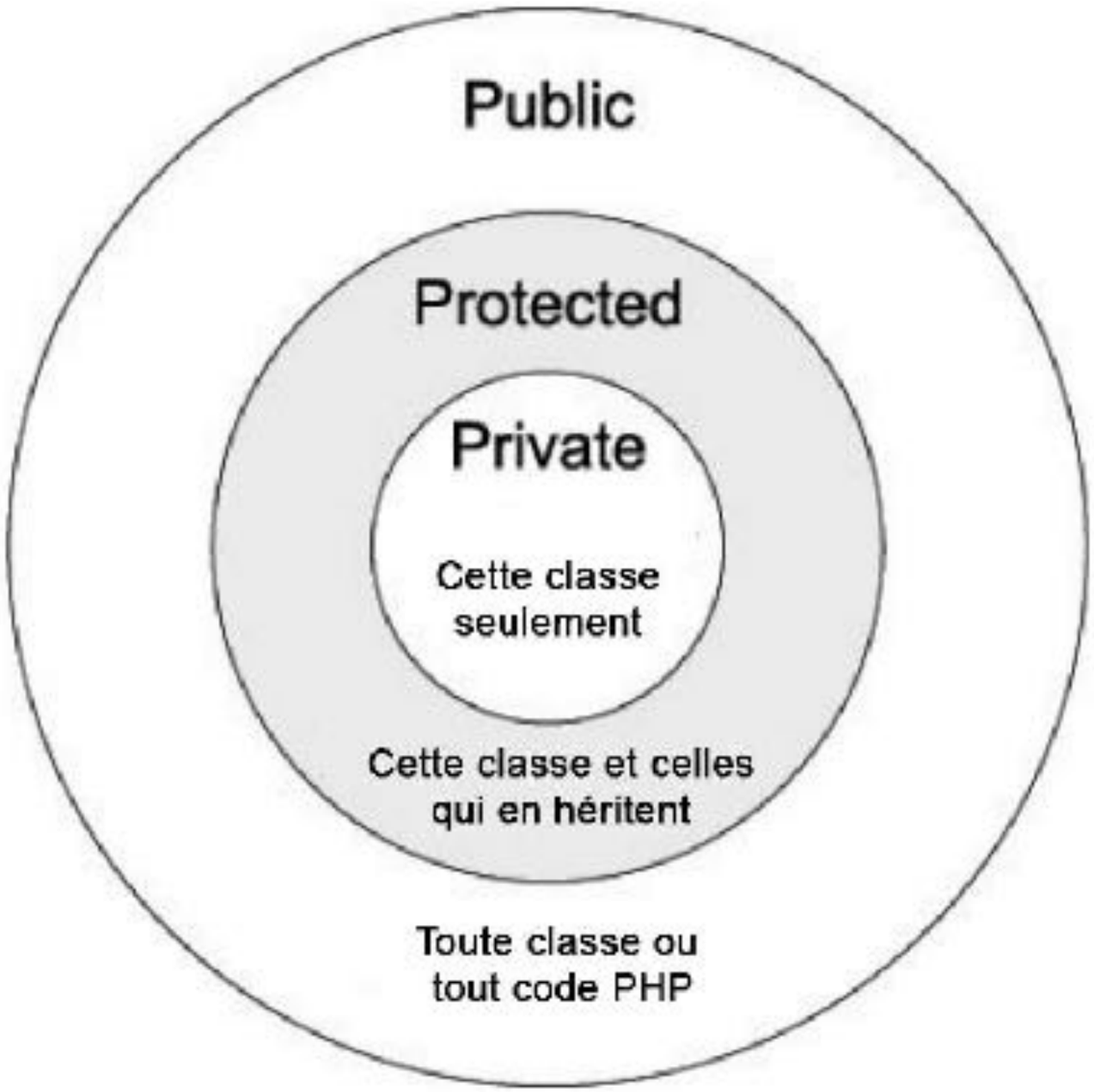
Principe d’encapsulation

- Comme les données sont très importantes dans le concept d’objet, il a été décidé qu’il serait possible pour un programmeur de restreindre l’accès aux données contenues dans l’objet.
- Pourquoi? Pour préserver l’intégrité de ces données.
  - En effet, il est important pour un programmeur de pouvoir décider de la nature des données qui seront contenues dans chacun des attributs d’un objet.
  - En encapsulant les données, le programmeur va fournir des méthodes d’accès aux attributs qui vont devoir être utilisées pour modifier ces derniers.
  - Il devient donc impossible d’affecter n’importe quelle valeur aux attributs d’un objet, puisqu’ils deviennent protégés par des méthodes dans lesquelles le programmeur peut valider la valeur à affecter à l’attribut et soulever une erreur si le contenu n’est pas valide.

Accessibilité

- En pratique, l'encapsulation des attributs est faite via les modificateurs d'accès, qui sont des mots-clés permettant de spécifier l'accès donné aux programmeurs qui utilisent la classe à chacun des attributs.
- Ces modificateurs d'accès peuvent être utilisés sur les attributs (pour protéger l'encapsulation), mais aussi sur les méthodes (pour protéger l'utilisation de certaines fonctions).
- Il existe trois mots-clés différents :
  - Private : l'attribut ou la méthode ne peut être utilisée qu'à l'intérieur de la classe.
  - Public : l'attribut ou la méthode peut être utilisée à l'extérieur ou à l'intérieur de la classe.
  - Protected : l'attribut ou la méthode ne peut être utilisée qu'à l'intérieur de la classe ou à l'intérieur d'une classe qui hérite de cette classe (nous verrons l'héritage plus tard).

Accessibilité



Modificateurs d’accès (suite)

- La pratique courante est de mettre tous les attributs private et de fournir des méthodes qui seront *public* pour permettre d’affecter une valeur à cet attribut ou d’accéder à cette valeur.
  - On les appelle des méthodes d’accès, ou des “getters” et des “setters”.
  - Ils comportent généralement une logique de validation pour empêcher qu’une valeur illogique soit mise dans un attribut.
- Voir exemple

Initialisation des objets

- Lorsque l'on crée un objet, on désire généralement affecter des valeurs à ses attributs.
- Plutôt que de créer l'objet pour ensuite appeler les méthodes qui nous permettent d'affecter des valeurs à ses attributs, il est préférable d'utiliser ce que l'on appelle des *constructeurs*.
- Le constructeur n'est jamais obligatoire, mais il est très utile pour faire toutes les opérations qui ne seront fait qu'à l'initialisation de l'objet.

Constructeurs et méthodes magiques

- Depuis PHP 5, il est conseillé de toujours utiliser le nom de fonction __construct() pour chaque constructeur que vous ajoutez à vos classes.
- Le constructeur est la méthode qui sera appelée automatiquement lorsque vous créez un nouvel objet avec le mot clé new.

  Ex :

```
function __construct( $nom, $prix, $fabricant, $description)  {
    //ne pas faire l'affectation directement parce que cela
    //briserait le concept d'encapsulation
    //$this->nom = $nom …. Aucune validation!
    $this->setNom($nom);
    $this->setPrix($prix);
    $this->setFabricant($fabricant);
    $this->setDescription($description);
}
```

Constructeurs par défaut et méthodes magiques

- Un constructeur peut aussi être défini pour être le constructeur par défaut (celui qui est appelé lorsque l'on envoie aucun paramètres lors de l'utilisation de *new)*.
- Il suffit de mettre tous les paramètres d'un constructeur optionnels (c'est-à-dire de donner des valeurs par défaut à tous les paramètres).
- Voir exemple

Destructeurs

- Il existe aussi la possibilité de déclarer des destructeurs, c'est-à-dire une méthode qui sera appelée automatiquement lorsqu'un objet a terminé son existence.
- Si je déclare un objet dans une fonction, par exemple, il sera détruit lorsque l'appel de la fonction sera terminé.
- Voir exemple

Injection de dépendances

- Il est possible pour un attribut d'une classe d'être aussi un objet!
- Par exemple, la classe Livre pourrait contenir un attribut Auteur qui serait de la classe Auteur!
- Mais alors, comment construire un objet Livre? Faut-il lui envoyer tous les attributs d'un Auteur en plus de ceux qui lui sont propres?
    - Voir exemple.
- Dans ce cas, il est préférable de découpler le plus possible les deux classes en faisant ce que l'on appelle de l'injection de dépendance.
    - Cela consiste à envoyer au constructeur de la classe conteneur une instance de la classe contenue.
    - On peut d'ailleurs renforcer ce comportement avec le *type hinting*, qui permet de forcer une fonction/méthode à recevoir un paramètre d'un certain type.
    - Voir exemple.