

PROGRAMMATION ORIENTÉE OBJETS

ENCAPSULATION

PREMIER PILIER DE LA POO

ENCASULATION

- L'encapsulation est possible grâce aux **spécificateurs d'accès**
 - mots-clés spécifiques appliqués aux classes, attributs et méthodes
 - `private`, `protected`, `public`
- Ils permettent de spécifier ce qui est accessible ou pas
- Autrement dit, il **restreignent la visibilité** des fonctionnalités d'une classe

LES SPÉCIFICATEURS D'ACCÈS

- `private`: visible dans la classe uniquement
- *absence* de spécificateur: *package private* (visible dans tout le package)
- `protected`: visible dans le package et les sous-classes de la classe
- `public`: visible partout

ENCAPSULATION

- En POO, on va interdire l'accès aux attributs d'une classe depuis l'extérieur
- Afin que l'état de l'objet soit complètement sous le contrôle de l'objet lui-même
- ex.: classe `CompteBancaire`
 - Pensez aux conséquences si n'importe quel code extérieur peut modifier directement, sans aucun contrôle, le solde d'un compte...

SPÉCIFICATEURS D'ACCÈS

```
public class Arbre {  
    private double taille;  
    private double diametreTronc;  
    private TypeArbre type;  
  
    public Arbre(double taille, double diametreTronc, TypeArbre type) {  
        this.taille = taille;  
        this.diametreTronc = diametreTronc;  
        this.type = type;  
    }  
  
    public void grandir() {  
        taille = taille + 10;  
        diametreTronc = diametreTronc + 0.5;  
    }  
}
```

ACCÈS EXTÉRIEUR AUX ATTRIBUTS ?

```
1 public class AppArbre {  
2     public static void main(String[] args) {  
3         // Instanciation  
4         Arbre peuplier = new Arbre();  
5         // Affichage de la taille  
6         System.out.println("Taille : " + peuplier.taille);  
7         // Erreur ! Impossible d'accéder à 'taille' (privé)  
8     }  
9 }
```

ACCÈS EXTÉRIEUR AUX ATTRIBUTS

- On veut accéder à un attribut depuis une classe cliente
- Mais l'attribut est privé (encapsulation)
- Au lieu de modifier son spécificateur d'accès, on va fournir une méthode (publique) qui va permettre un accès (indirect)
- Une telle méthode s'appelle un *getter*

GETTER - CONVENTION DE NOMMAGE

- Un *getter* permet l'accès à un attribut en **lecture** depuis une classe cliente
- Par convention, un getter a un nom:
 - préfixé par `get`
 - suivi du nom de l'attribut concerné (avec une majuscule)
 - ex: `getCouleur` pour un *getter* de l'attribut `couleur`

GETTER - CARACTÉRISTIQUES

- Un *getter*:
 - est `public` (ou au moins *package-private*)
 - ne prend (en général) aucun paramètre
 - a un type de retour identique au type de l'attribut
 - retourne l'attribut
 - ex: `public String getCouleur()` (si couleur est de type `String`)

GETTER

```
public class Arbre {  
    private double taille;  
    private double diametreTronc;  
    private TypeArbre type;  
  
    // Getter pour l'attribut 'taille'  
    public double getTaille() {  
        return taille;  
    }  
  
    // Autres méthodes omises...  
}
```

ACCÈS GRÂCE AU GETTER

```
1 public class AppArbre {  
2     public static void main(String[] args) {  
3         // Instanciation  
4         Arbre peuplier = new Arbre();  
5         // Affichage de la taille  
6         System.out.println("Taille : " + peuplier.getTaille());  
7         // Accès depuis le getter public : OK  
8     }  
9 }
```

ACCÈS EXTÉRIEUR EN ÉCRITURE

- On veut accéder à un attribut depuis une classe cliente, mais cette fois en **écriture**
- On va fournir une méthode (publique) qui va permettre un accès en écriture (indirect)
- Une telle méthode s'appelle un ***setter***

SETTER - CONVENTION DE NOMMAGE

- Un *setter* permet l'accès à un attribut en écriture depuis une classe cliente
- Par convention, un setter a un nom:
 - préfixé par set
 - suivi du nom de l'attribut concerné (avec une majuscule)
 - ex: setCouleur pour un *setter* de l'attribut couleur

SETTER - CARACTÉRISTIQUES

- Un *setter*:
 - est `public` (ou au moins *package-private*)
 - prend en paramètre la nouvelle valeur
 - a un type de retour `void`
 - possède une affectation: `attribut = laNouvelleValeur`
 - ex: `public void setCouleur(String valeur)` (si `couleur` est de type `String`)

SETTER

```
public class Arbre {  
    private double taille;  
    private double diametreTronc;  
    private TypeArbre type;  
  
    // Setter pour l'attribut 'taille'  
    public void setTaille(double valeur) {  
        taille = valeur;  
    }  
  
    // Autres méthodes omises...  
}
```


ACCÈS GRÂCE AU SETTER

```
1 public class AppArbre {  
2     public static void main(String[] args) {  
3         // Instanciation  
4         Arbre peuplier = new Arbre();  
5         // Modification de la taille  
6         peuplier.setTaille(200);  
7         System.out.println("Taille : " + peuplier.getTaille());  
8         // Quelle taille fait ici le peuplier ?  
9     }  
10 }
```

GET/SET - POUR QUOI FAIRE ?

- En général, le principe est de contrôler l'état de l'objet
- **On veut garder l'objet dans un état stable**
- Java refusera qu'on mette la valeur d'un nombre entier à « bleu »
 - ça n'a pas de sens
 - ça poserait un problème quand on fera une opération avec ce nombre devenu « instable »
- il en va de même pour les classes que vous créez
- C'est un des bénéfices principaux de l'**encapsulation**

ENCAPSULATION - INTÉRÊTS

- Dans la classe `Arbre`, le concepteur ne veut pas que l'on puisse changer le type d'un arbre une fois que l'arbre a été instancié
 - pourquoi un peuplier deviendrait soudain un bouleau?
- L'état de l'arbre pourrait alors devenir instable
- donc \Rightarrow type de l'arbre `private` et pas de *setter*
- En revanche il semble légitime qu'un client demande le type de l'arbre
 - on fournira alors un *getter*
 - mais seulement si on en a vraiment besoin

IMMUABILITÉ

- Lorsqu'une valeur ne change pas, on dit qu'elle est **immuable**
 - par exemple, le type de l'arbre est immuable
- Une classe est immuable lorsque tous ses attributs le sont
- Concevoir des classes complètement immuables est souvent une bonne pratique (moins de bugs)

EXERCICE - CLASSE "EMPLOYEE"

- Reprendre la classe `Employee` en appliquant les principes d'encapsulation
- Notamment, le code précédent d'affichage des infos des employés ne fonctionne plus