L’encapsulation

Qu’est-ce que l’encapsulation ?

L’**encapsulation** est un procédé qui consiste à rassembler les données et les méthodes au sein d’une structure en cachant l’implémentation de l’objet (on met les données et les méthodes dans une capsule).

Ainsi, l’accès aux données est réglementé. L’encapsulation permet donc de garantir l’**intégrité des données** contenues dans l’**objet**. Lorsqu’on veut protéger des informations contre une modification inattendue, on doit se référer au principe d’encapsulation.

L’encapsulation sous Java

En Java, comme dans beaucoup de langages de POO, les classes, les attributs et les méthodes bénéficient de **niveaux d’accessibilité**, qui indiquent dans quelles circonstances on peut accéder à ces éléments.

Principe : cacher les détails de mise en œuvre d’un objet

⇒ pas d’accès direct aux champs de l’extérieur de l’objet

Mise en oeuvre

Chaque entité (classe, champ, méthode ou constructeur) possède un niveau d'encapsulation :

- Définit à partir d'où dans le programme une entité est visible
- Permet de masquer les détails de mise en œuvre d'un objet

Niveaux d'encapsulation en Java

L'encapsulation permet de définir des niveaux de visibilité des éléments de la **classe.** Ces niveaux de visibilité définissent les **droits d'accès aux données** selon que l'on y accède par une méthode de la classe elle-même, d'une **classe fille**, ou bien d'une classe quelconque. Il existe quatre niveaux de visibilité :

- **Visibilité par défaut :** aucun modificateur de visibilité n'est indiqué.

Dans ce cas, pas de visivibilité en dehors du package

- **Visibilité publique :** on peut accéder aux données ou aux méthodes d'une classe définie avec ce niveau de visibilité à partir de n'importe quelle classe : mot clé public. Il s'agit du plus bas niveau de protection des données.

Visibilité privée : l'accès aux données est limité aux méthodes de la classe elle-même : mot clé private. Il s'agit du niveau de protection des données le plus élevé.

- **Visibilité protégée** : l'accès aux données est réservé aux fonctions des classes filles. On reviendra dessus avec l'héritage. (mot clé protected).

Comment déterminer le niveau de visibilité ?

En général :

- Les champs sont privés (inaccessibles en dehors de la classe)
- Les méthodes sont publiques

Méthodes d’accès

Lorsque les attributs sont déclarés en « private », ils ne sont pas accessibles directement de l’extérieur de la classe.

Pour accéder et modifier ces attributs depuis une autre classe, on utilise des méthodes d’accès : des accesseurs (getters) et des modificateurs (setters). Ce procédé présente plusieurs avantages. Par exemple,

- On peut ajouter des contrôles pour éviter que ces attributs ne prennent des valeurs invalides.

- On évite d’exposer la représentation interne de l’objet.

L’encapsulation par l’exemple

- Reprenons l’exercice 2 du TD 8 :
    - Nous allons Créer une classe nommée CompteBancaire avec les attributs suivants, tous définis comme **private** (l’accès à ces attributs n’est possible que par le bais de méthodes spécifiques : getters et setters) :
        - numeroCompte (String) : le numéro du compte.
        - solde (double) : le solde du compte.
    - Ensuite, nous allons créer les méthodes d’accès en lecture et écriture : getters et setters
    - - un **getter** pour le numéro de compte
    - - un **setter** pour modifier le solde (le solde ne doit pas être négatif)
- fonctionnalités du compte :
    - Une méthode deposer(double montant) .
    - Une méthode retirer(double montant) // (on s’assurera que le solde reste positif après un retrait).

```java
public class CompteBancaire {
	private String numeroCompte ;
	private double solde ;
	public CompteBancaire(String num,double leSolde)
	{
		this.numeroCompte=num;
		if (leSolde>=0) {
			this.solde=leSolde;
		}
		else {
			System.out.println("Le solde initial ne peut pas être
			négatif. Solde initialisé à 0.");
			this.solde = 0;
			}
	}
```

```java
// Accesseur en lecture (getter ) pour le numéro de compte
        public String getNumeroCompte() {
                return numeroCompte;
        }


        // Accesseur en lecture (getter ) pour le solde
        public double getSolde() {
                return solde;
        }
        // Accesseur en modification (setter ) pour le solde
        public void setSolde(double nouveauSolde) {
                if (nouveauSolde >= 0) {
                        this.solde = nouveauSolde;
                }
                else {
                        System.out.println("Erreur : le solde ne peut pas être négatif !");
                }
        }
```

```java
// Méthodes pour fonctionnalités du compte
        public void deposer(double montant) {
                if (montant>0) solde+=montant;
                else System.out.println("Erreur : le montant à déposer doit être positif !");
        }
        public void retirer(double montant) {
                if(solde<montant) System.out.println("la provision est insuffisante ");
                else {
                        solde-=montant ;
                        System.out.println("Vous avez retiré "+ montant+" euros");
                }
        }
        public void afficherInfoCompte() {
                System.out.println("Numéro de compte : "+numeroCompte+"\nVotre nouveau solde est
                "+solde);
        }
}
```

Test de la classe compte bancaire

```
public class TestCompteBancaire {


        public static void main(String[] args) {
// Création d'un compte bancaire
                CompteBancaire compte = new CompteBancaire("63047S", 50.50);


// Informations du compte
                compte.afficherInfoCompte();
```

```
// Opérations sur le compte

                compte.deposer(1500.0); // On dépose 1500
                compte.retirer(300.0); // On retire 300
                compte.retirer(2000.0); /* Que se passe-t-il si le montant demandé est plus grand
que le solde ?*/
                compte.setSolde(-100.0); // Peut-on initialiser le solde avec un
réel négatif ?


// Affichage des informations après opérations

                compte.afficherInfoCompte();
        }
}
```