

Illustration

```
public class Additionneur {
    private int resultat = 0;
    public void calcule(int nb1, int nb2) {
        this.resultat = nb1 + nb2;
    }
    public int getResultat() {
        return this.resultat;
    }
}

Additionneur add = new Additionneur();
add.calcule(5, 3);
System.out.println(add.getResultat());
add.resultat = 12; // !!! interdit !!!, évite corruption résultat
System.out.println(add.getResultat());
```

- ▶ **resultat** ne doit pas pouvoir être modifié directement, il doit correspondre au résultat de l'addition. "Contrat" de la classe.

- ▶ gestion de compte en banque, classe `Compte` avec attribut `solde` exprimant le solde

```
Compte compte = new Compte();
```

1. **solde** est "public" dès que l'on a la référence de `compte` : si un "programmeur client" programme un module d'affichage de solde :

```
System.out.println(compte.solde);
compte.solde = 1000000; // ben tiens!
Si l'on décide dans un second temps d'encapsuler compte :
tous les programmes clients doivent être modifiés!
```

2. **solde** est "private" et encapsulé :
 - ↪ utilisation dès le départ de `compte.getSolde()` et `compte.setSolde(...)`
 - Le "programmeur créateur" peut modifier le comportement sans impact :
 - ▶ contrôle lors de la modification
 - ▶ suppression de l'attribut `solde` remplacé par `credit` et `debit`
 - ▶ etc.

```
private Auteur monAuteur;

public void setMonAuteur(Auteur auteur) {
    this.monAuteur = auteur;
}

public Auteur getMonAuteur() {
    return this.monAuteur;
}
```

```
public class Livre {
    // les attributs de la classe livre
    private Auteur auteur;
    private String titre;
    private int annee;
    private String texte;
    // constructeur
    public Livre(Auteur unAuteur, String titre, int annee, String texte) {
        this.auteur = unAuteur;
        this.titre = titre;
        this.annee = annee;
        this.texte = texte;
    }
    // les méthodes de la classe Livre
    public Auteur getAuteur() {
        return this.auteur;
    }
    public void setAuteur(Auteur auteur) {
        this.auteur = auteur;
    }
}
```

Exploitation

(en dehors de la class `Livre`)

```
Livre leLivre = new Livre("JRR Tolkien", "Le Seigneur des Anneaux", 1954);
leLivre.affiche();
System.out.println(leLivre.auteur); // !!! interdit !!!
System.out.println(leLivre.getAuteur());
leLivre.auteur = new Auteur("un autre"); // !!! interdit !!!
leLivre.texte = "Quand M. Bilbon Sacquet, ..."; // !!! interdit !!!
leLivre.setAuteur(new Auteur("un autre"));
```

Vive le public

Ce qui comptent ce sont les fonctionnalités proposées par une classe, **son interface publique**

Considérons une application dans laquelle on manipule des objets **Disque**.

L'important est ce que l'on peut faire avec ces disques :
`surface() :float,` `perimetre() :float,` `getRayon() :float,`
`getCentre() :Point,` `appartient(p :Point) :boolean,` `etc.`

Quel état pour ces objets ?

Mais attention

Les références étant copiées, il y a **partage de référence** entre le paramètre formel et le paramètre effectif!

```
public class TestPassageParCopie {
    public void changeRayonDisque(Disque disque) {
        disque.setRayon(5);
        System.out.println("dans méthode ->" +disque);
    }
    public static void main(String[] args) {
        TestPassageParCopie test = new TestPassageParCopie();
        Disque d = new Disque(3);
        System.out.println("avant -> " +d);
        test.changeRayonDisque(d);
        System.out.println("après -> " +d);
    }
}

trace d'exécution : java TestPassageParCopie
avant -> 3
dans méthode -> 5
après -> 5
```

Problème de l'égalité

- ▶ identificateur d'objet = information sur comment trouver l'objet référencé
- ▶ comparer 2 identificateurs = vérifier si cette information est la même
- ▶ Rien à voir avec le contenu des objets référencés
- ▶ identificateur = pointeur, donc on retrouve la problématique de l'égalité de pointeur.

```
String ch1 = new String("Le Seigneur des Anneaux");
String ch2 = new String("Le Seigneur des Anneaux");
```

- ▶ 2 références différentes sur 2 objets **différents**
- ▶ `ch1 == ch2` \implies `false`
- ▶ pour comparer les états des instances on utilise la méthode `equals`
`ch1.equals(ch2)` \implies `true`
- ▶ La méthode `equals`, doit être définie et adaptée pour chaque classe. Par défaut, elle fait comme `==`!
- ▶ `String ch3 = ch1;`
 \hookrightarrow range dans `ch3` l'information stockée dans `ch1`
`ch1 == ch3` \implies `true`
`ch1.equals(ch3)` \implies `true`

En Java tout est objet ?

Oui... sauf les types primitifs

| type primitif | ex | Size | minimum | maximum | classe "Wrapper" |
|---------------|-------------------|---------|---|--|------------------|
| boolean | <code>true</code> | - | - | - | Boolean |
| char | <code>'x'</code> | 16 bits | Unicode 0 (\u0000) | Unicode 2 ¹⁶ - 1 (\uFFFF) | Character |
| byte | | 8 bits | -128 | +127 | Byte |
| short | | 16 bits | -2 ¹⁵ | +2 ¹⁵ - 1 | Short |
| int | | 32 bits | -2 ³¹ | +2 ³¹ - 1 | Integer |
| long | | 64 bits | -2 ⁶³ | +2 ⁶³ - 1 | Long |
| float | | 32 bits | -10 ³⁸ ... - 10 ⁻³⁸ | 10 ⁻³⁸ ... + 10 ³⁸ | Float |
| double | | 64 bits | -10 ³⁰⁸ ... - 10 ⁻³⁰⁸ | 10 ⁻³⁰⁸ ... + 10 ³⁰⁸ | Double |
| void | | - | - | - | Void |

justificatif? facilité d'utilisation et de manipulation

Variables primitives

déclaration de variable primitive : pas de new (pas d'objet!)

```
int i;                    boolean fini = true;
```

variable primitive = espace mémoire réservé
 taille fixe (objet pas de taille fixe (référence si!) y compris même classe)

- ▶ variable de **type primitif** contient la **valeur** de la **variable**
- ▶ variable **référence d'objet** contient l'information sur **comment trouver l'objet**

Retour sur égalité

Type primitif : variable contient valeur

donc

```
int i = 5;
int j = 5;
```

`i == j` \implies `true`

`==` ne regarde que le contenu des variables

nul n'est parfait...

```
int biggest = Integer.MAX_VALUE;  
int biggerThanBiggest = biggest+1;  
System.out.println("biggest = "+biggest);  
System.out.println("biggerThanBiggest = "+biggerThanBiggest);
```

```
+-----  
| biggest = 2147483647  
| biggerThanBiggest = -2147483648  
+-----
```