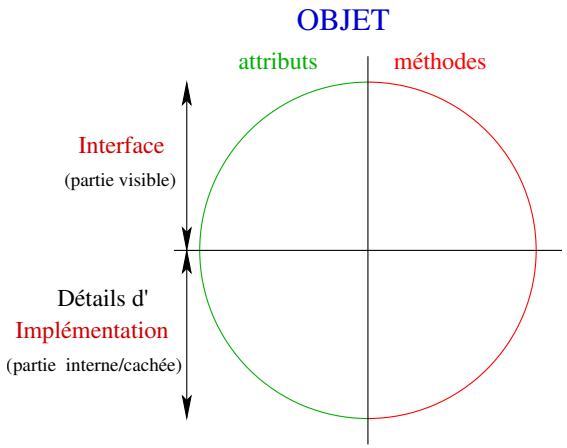
Encapsulation / Abstraction



Encapsulation et interface

Tout ce qu'il n'est pas nécessaire de connaître à l'extérieur d'un objet devrait être dans le corps de l'objet et identifié par le mot clé `private` :

```
class Rectangle {
  private double hauteur;
  private double largeur;
  double surface();
}
```

Attribut d'instance **privée** = inaccessible depuis l'extérieur de la classe.
C'est également *valable pour les méthodes*.

Erreur de compilation si référence à un(e) attribut/méthode d'instance privée :

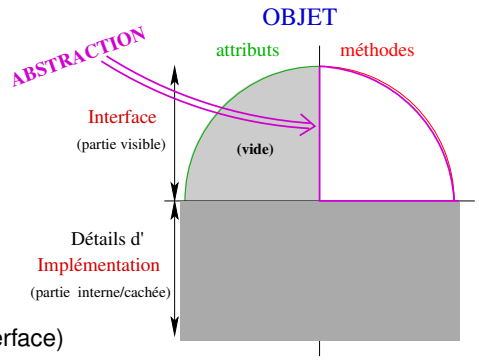
```
error: hauteur has private access in Rectangle
```

Note : Si aucun droit d'accès n'est précisé il s'agit des droits d'accès par défaut («friendly»).

Encapsulation et interface (2)

À l'inverse, l'interface, qui est accessible de l'extérieur, se déclare avec le mot-clé `public:`

```
class Rectangle {
    public double surface() { ... }
    ...
}
```



Dans la plupart des cas :

- **Privé** :
  - Tous les attributs
  - La plupart des méthodes
- **Public** :
  - Quelques méthodes bien choisies (interface)

Droit d'accès `public` et droit par défaut

Les programmes Java sont habituellement organisés au moyen de la notion de paquetage (`package`) (voir les compléments)
[Si vous ne spécifiez rien, vous travaillez dans le `package` **par défaut**.]

Si aucun droit d'accès n'est spécifié, alors l'attribut/la méthode est publiquement accessible par toutes les autres classes **du même package**, mais *pas en dehors* de celui-ci !

☞ Il est recommandé de mettre explicitement `public` devant tout membre que vous estimez devoir appartenir à l'interface de la classe.

« Accesseurs » et « manipulateurs »

Tous les attributs sont privés ?

- Et si on a besoin de les utiliser depuis l'extérieur de la classe ?!

  Par exemple, comment « manipuler » la largeur et la hauteur d'un rectangle ?

« Accesseurs » et « manipulateurs »

Si le programmeur *le juge utile*, il **inclut les méthodes publiques nécessaires** ...

1. Accesseurs (« méthodes get » ou « getters ») :
   - Consultation
   - Retour de la valeur d'une variable d'instance précise

```
double getHauteur() { return hauteur; }
double getLargeur() { return largeur; }
```

2. Manipulateurs (« méthodes set » ou « setters ») :
   - Modification (i.e. « action »)
   - Affectation de l'argument à une variable d'instance précise

```
void setHauteur(double h) { hauteur = h; }
void setLargeur(double l) { largeur = l; }
```

Notre programme (4/4)

```
class Exemple
{
  public static void main (String[] args)
  {
    Rectangle rect1 = new Rectangle();

    rect1.setHauteur(3.0);
    rect1.setLargeur(4.0);

    System.out.println("hauteur : "
                       + rect1.getHauteur());
  }
}
```

```
class Rectangle
{
  public double surface()
    { return hauteur * largeur; }

  public double getHauteur()
    { return hauteur; }
  public double getLargeur()
    { return largeur; }

  public void setHauteur(double h)
    { hauteur = h; }
  public void setLargeur(double l)
    { largeur = l; }

  private double hauteur;
  private double largeur;
}
```

« Accesseurs », « manipulateurs » et encapsulation

Fastidieux d'écrire des « Accesseurs »/« manipulateurs » alors que l'on pourrait tout laisser en `public` ?

```
class Rectangle
{
  public double largeur;
  public double hauteur;
  public String label;
}
```

mais dans ce cas ...

```
Rectangle rect = new Rectangle();
rect.hauteur = -36;
System.out.print(rect.label.length());
```

Masquage (shadowing)

masquage = un identificateur « cache » un autre identificateur

Situation typique en POO : un paramètre cache un attribut

```
void setHauteur(double hauteur) {
  hauteur = hauteur;  // Hmm.... pas terrible !
}
```

Masquage et `this`

Si, dans une méthode, un attribut est **masqué** alors la valeur de l'attribut peut quand même être référencée à l'aide du mot réservé `this`.

`this` est une **référence à l'instance courante**

`this` ≃ « moi »

Syntaxe pour spécifier un attribut *en cas d'ambiguïté* :
`this.nomAttribut`

Exemple :

```
void setHauteur(double hauteur) {
  this.hauteur = hauteur;  // Ah, là ça marche !
}
```

L'utilisation de `this` est obligatoire dans les situations de **masquage** (mais évitez ces situations !)

Portée des attributs (résumé)

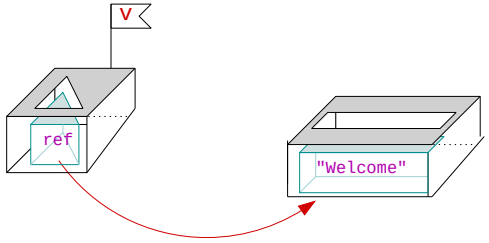
La portée des attributs dans la définition des méthodes est résumée par le schéma suivant :

```
class MaClasse {
  private int x;
  private int y;

  public void uneMethode( int x ) {
    ... y ...
    ... x ...
    ... this.x ...
  }
}
```

Objets en mémoire

Attention : Comme nous l'avons déjà vu pour la classe prédéfinie `String` (et pour les tableaux), les objets, contrairement aux entités de types élémentaires, sont manipulés via des **références** :



Il est impératif de s'en rappeler lorsque l'on :

- *compare* deux objets
- *affecte* un objet à un autre
- *affiche* un objet

La constante `null`

La constante prédéfinie `null` peut être affectée à n'importe quel objet de n'importe quelle classe.

Affectée à une variable, elle indique que celle-ci ne référence aucun objet :

```
Rectangle  rect = null; // la variable rect
                        // ne reference aucun objet
```

Avant de commencer à utiliser un objet, il est souvent judicieux de tester s'il existe vraiment, ce qui peut se faire par des tournures telles que :

```
if (rect == null) {...}
if (rect != null) {...}
```