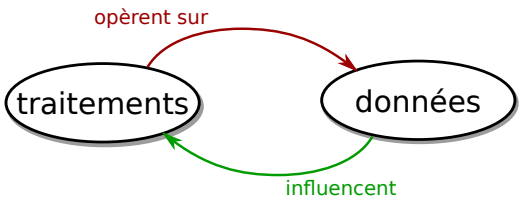
Programmation impérative/procédurale (rappel)

Dans les programmes que vous avez écrits jusqu'à maintenant, les notions

- de variables/types de **données**
- et de **traitement** de ces données

étaient séparées :



Programmation procédurale : exemple

```
class Geometrie {

  public static void main(String[] args) {
    double largeur = 3.0;
    double hauteur = 4.0;

    System.out.println("Surface du rectangle : "
                       + surface(largeur, hauteur));
  }

  static double surface(double largeur,
                        double hauteur) {
    return (largeur * hauteur);
  }
}
```

Objets : quatre concepts de base

Un des objectifs principaux de la notion d'**objet** :

organiser des programmes complexes

grâce aux notions :

- d'encapsulation
- d'abstraction
- d'héritage
- et de polymorphisme

Notions d'encapsulation

Principe d'encapsulation :
regrouper dans le même objet informatique («concept»), les données et les traitements qui lui sont *spécifiques* :

- **attributs** : les données incluses dans un objet
- **méthodes** : les fonctions (= traitements) définies dans un objet

☞ Les objets sont définis par leurs attributs et leurs méthodes.

Notion d'abstraction

Pour être véritablement intéressant, un objet doit permettre un certain degré d'**abstraction**.

Le processus d'abstraction consiste à identifier pour un ensemble d'éléments :

- des caractéristiques communes à tous les éléments
- des mécanismes communs à tous les éléments

☞ description **générique** de l'ensemble considéré :
se focaliser sur l'essentiel, cacher les détails.

Notion d'abstraction : exemple

Exemple : Rectangles

- la notion d'« *objet rectangle* » n'est intéressante que si l'on peut lui associer des propriétés et/ou mécanismes *généraux* (valables pour l'ensemble des rectangles)
- Les notions de *largeur* et *hauteur* sont des propriétés générales des rectangles (**attributs**),
- Le mécanisme permettant de calculer la surface d'un rectangle (surface = largeur × hauteur) est commun à tous les rectangles (**méthodes**)

Abstraction et Encapsulation

En plus du regroupement des données et des traitements relatifs à une entité, l'encapsulation permet en effet de définir **deux niveaux** de perception des objets :

- niveau *externe* : partie « *visible* » (par les programmeurs-utilisateurs) :
  - l'**interface** : *entête* de quelques méthodes bien choisies
  - ☞ résultat du processus d'*abstraction*
- niveau *interne* : (détails d')**implémentation**
  - **corps** :
    - méthodes et attributs accessibles uniquement depuis l'intérieur de l'objet (ou d'objets similaires)
    - définition de toutes les méthodes de l'objet

Exemple d'interface

L'interface d'une voiture

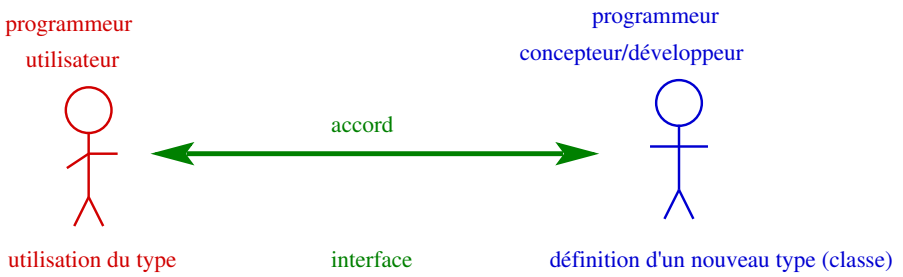
- Volant, accélérateur, pédale de frein, etc.
- Tout ce qu'il faut savoir pour la conduire
  (mais pas la réparer ! ni comprendre comment ça marche)
- L'interface ne change pas, même si l'on change de moteur...
  ...et même si on change de voiture (dans une certaine mesure) :
  abstraction de la notion de voiture (en tant qu'« objet à conduire »)

Encapsulation et Interface

Il y a donc deux facettes à l'encapsulation :

1. regroupement de tout ce qui caractérise l'objet : données (attributs) **et** traitements (méthodes)
2. isolement et dissimulation des détails d'implémentation
   Interface = ce que le programmeur-utilisateur (hors de l'objet) peut utiliser
   - ☞ Concentration sur les attributs et les méthodes concernant l'objet (*abstraction*)

Les « 3 facettes » d'une classe



Pourquoi abstraire/encapsuler ?

1. L'intérêt de regrouper les traitements et les données conceptuellement reliées est de permettre une *meilleure visibilité* et une meilleure cohérence au programme, d'offrir une plus grande modularité.

```
double largeur = 3.0;
double hauteur = 4.0;

System.out.print("Surface : ");
System.out.println(surface(largeur,
                    hauteur));
```

```
Rectangle rect = new Rectangle(3.0, 4.0);
System.out.print("Surface : ");
System.out.println(rect.surface()):
```

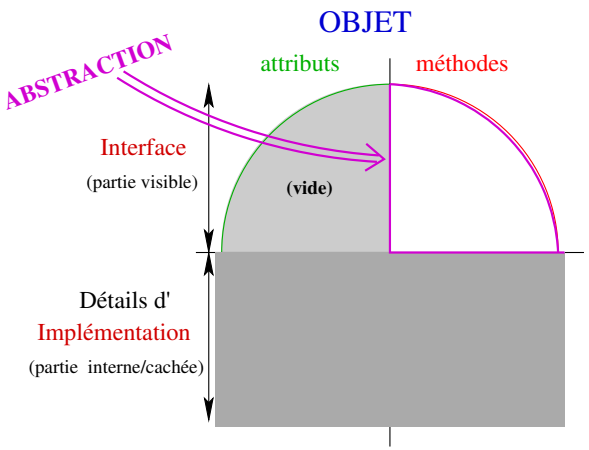
Pourquoi abstraire/encapsuler ? (2)

2. L'intérêt de séparer les niveaux *interne* et *externe* est de donner un **cadre** plus **rigoureux** à l'utilisation des objets utilisés dans un programme

   Les objets ne peuvent être utilisés qu'au travers de leurs interfaces (niveau externe)
   et donc les éventuelles **modifications** de la structure interne restent **invisibles** à l'extérieur

Règle : les attributs d'un objet ne doivent pas être accessibles depuis l'extérieur, mais uniquement par des méthodes.

Encapsulation / Abstraction : Résumé

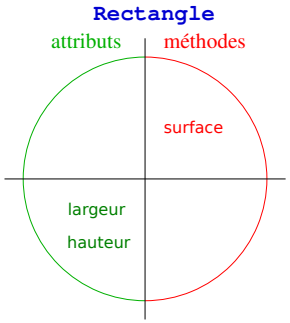


Classes et Instances, Types et Variables

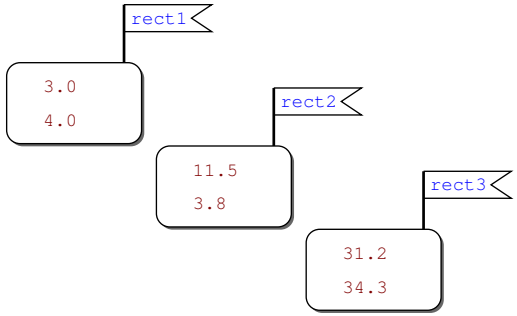
En programmation Objet :

- le résultat des processus d'encapsulation et d'abstraction s'appelle une **classe**
  classe = catégorie d'objets
- une classe définit un **type**
- une réalisation particulière d'une classe s'appelle une **instance**
  instance = **objet**
- un objet est une **variable**

Classes et Instances, Types et Variables (illustration)



classe
type (abstraction)
existence conceptuelle
(écriture du programme)



objets/instances
variables en mémoire
existence concrète
(exécution du programme)