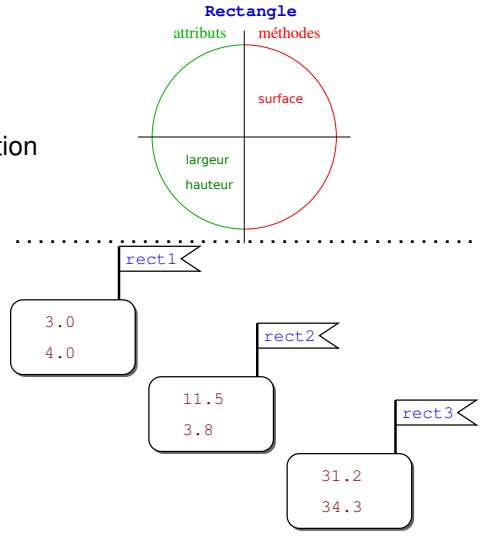
Classes et Instances, Types et Variables

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**



Les classes en Java

En Java une **classe** se déclare par le mot-clé `class`.

Exemple : `class Rectangle { ... }`

Ceci définit un nouveau **type** du langage.

La déclaration d'une **instance** d'une classe se fait de façon similaire à la déclaration d'une **variable** :

`nomClasse nomInstance ;`

Exemple :

`Rectangle rect1;`

déclare une instance `rect1` de la classe `Rectangle`.

Où déclarer les classes ?

1. Déclaration de plusieurs classes dans le même fichier : la classe `Rectangle` et la classe qui l'utilise `Dessin` déclarées dans `Dessin.java`

```
class Dessin
{
   public static void main (String[] args)
   {
         Rectangle r;
         // ....
   }
}
class Rectangle
{
   double largeur;
   double hauteur;
   //...
}
```

```
> javac Dessin.java
```

produit les fichiers :

```
Dessin.class
Rectangle.class
```

Le compilateur crée un fichier `.class` pour chaque classe

Où déclarer les classes? (2)

2. Déclaration de chaque classe dans un fichier à part :
   - `Rectangle` déclarée dans `Rectangle.java`
   - `Dessin` déclarée dans `Dessin.java`
   - Compilation de *chaque* fichier nécessaire

```
> javac Rectangle.java
> javac Dessin.java
```

produit les fichiers:

```
Rectangle.class
Dessin.class
```

Seule la classe contenant main est exécutable:

```
> java Rectangle
Exception in thread "main" java.lang.NoSuchMethodError: main
```

Déclaration des attributs

La syntaxe de la déclaration des attributs est la suivante :

type nomAttribut ;

Exemple :
les attributs `hauteur` et `largeur`, de type `double`,
de la classe `Rectangle` pourront être déclarés par :

```
class Rectangle {
    double hauteur;
    double largeur;
}
```

Accès aux attributs

L'accès aux valeurs des attributs d'une instance de nom `nomInstance`
se fait via la notation pointée :

nomInstance.nomAttribut

Exemple :
la valeur de l'attribut `hauteur` d'une instance `rect1`
de la classe `Rectangle` sera référencée par l'expression :

```
rect1.hauteur
```

Notre programme (2/4)

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

    rect1.hauteur = 3.0;
    rect1.largeur = 4.0;

    System.out.println("hauteur : " + rect1.hauteur);
  }
}
class Rectangle
{
  double hauteur;
  double largeur;
}
```

Déclaration-intialisation d'une instance

L'instruction :

```
nomClasse instance = new nomClasse();
```

crée une instance de type `nomClasse` et initialise tous ses attributs avec des valeurs par défaut :

`int`	`0`
`double`	`0.0`
`boolean`	`false`
objets	`null`

Exemple :

```
Rectangle rect = new Rectangle();
```

Déclaration des méthodes

La syntaxe de la définition des méthodes d'une classe est la syntaxe normale de définition des fonctions :

```
typeRetour nomMethode (typeParam1 nomParam1, ...)
{
  // corps de la méthode
  ...
}
```

mais elles sont simplement mises *dans* la classe elle-même.

Exemple : la méthode `surface()` de la classe `Rectangle` :

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

mais où sont passés les paramètres ?

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

Portée des attributs

Les attributs d'une classe constituent des variables *directement accessibles* dans toutes les méthodes de la classe (*i.e.* des « variables *globales à la classe* »).

On parle de « *portée de classe* ».

Il n'est donc **pas nécessaire de les passer comme arguments des méthodes**.

Par exemple, dans toutes les méthodes de la classe `Rectangle`, l'identificateur `hauteur` (resp. `largeur`) fait *a priori* référence à la valeur de l'attribut `hauteur` (resp. `largeur`) de l'instance concernée (par l'appel de la méthode en question)

Déclaration des méthodes

Les méthodes sont donc :

- des fonctions propres à la classe
- qui ont donc accès aux attributs de la classe

☞ Il ne faut donc **pas** passer les attributs comme arguments aux méthodes de la classe !

Exemple :

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

Paramètres des méthodes

Mais ce n'est pas parce qu'on n'a pas besoin de passer les attributs de la classe comme arguments aux méthodes de cette classe, que les méthodes n'ont *jamais* de paramètres.

Les méthodes **peuvent avoir des paramètres** : ceux qui sont nécessaires (et donc *extérieurs à l'instance*) pour exécuter la méthode en question !

Exemple :

```
class FigureColoree {
  // ...
  void colorie(Couleur c)  { /* ... */ }
  // ...
}

FigureColoree uneFigure;
Couleur rouge;
// ...
uneFigure.colorie(rouge);
// ...
```

Appels aux méthodes

L'appel aux méthodes définies pour une instance de nom `nomInstance` se fait à l'aide d'expressions de la forme :

`nomInstance.nomMethode(valArg1, ...)`

Exemple : la méthode

`void surface();`

définie pour la classe `Rectangle` peut être appelée pour une instance `rect1` de cette classe par :

`rect1.surface()`

Autres exemples :

```
uneFigure.colorie(ROUGE);
i < tableau.size()
```

Notre programme (3/4)

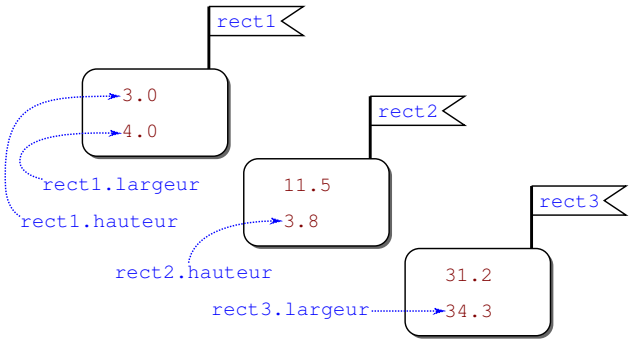
```
class Exemple
{
  public static void main (String[] args)
  {
    Rectangle rect1 = new Rectangle();

    rect1.hauteur = 3.0;
    rect1.largeur = 4.0;

    System.out.println("surface : " + rect1.surface());
  }
}
class Rectangle
{
  double hauteur;
  double largeur;
  double surface() {
    return hauteur * largeur;
  }
}
```

Résumé : Accès aux attributs et méthodes

Chaque instance a ses propres attributs : aucun risque de confusion d'une instance à une autre.



`rect1.surface()` : méthode `surface` de la classe `Rectangle` s'appliquant à `rect1`