

# **Le langage Java - Syntaxe**

**CLASSES, INSTANCE, MÉTHODES, ...**

**CLASSE EN JAVA, PAS À PAS**

**UTILISER DES CLASSES: C O=NEW C()**

**UTILISER LES MÉTHODES: O.M()**

**UNE CLASSE AVEC PLUSIEURS CONSTRUCTEURS**

**LA SURCHARGE DES MÉTHODES**

**VARIABLES DE CLASSE**

**VARIABLES D'INSTANCE**

**LES MÉTHODES DE LA CLASSE**

**EXEMPLE AVEC STATIC**

**ET SI L'ON PARLAIT D'HÉRITAGE**

**CLASSE ET MÉTHODES ABSTRAITES**

**L'HÉRITAGE MULTIPLE VS INTERFACE.**

**REGROUPER DES CLASSES DANS UN PACKAGE**

**TYPÉ UN OBJET (CASTING)**

**COPIER DES OBJETS**

**COMPARER DES OBJETS**

# **Une introduction aux objets**

il nous faut examiner ce qu'est un objet.

# une vision animiste!

une vision *animiste*

Définition de *classe*:

Une classe est archétype qui conditionne tous les comportements

Définition *d'objet*:

Un objet est une instance d'une et une seule classe. Un individu qui possède tous les comportements de la classe dont il est dérivé.

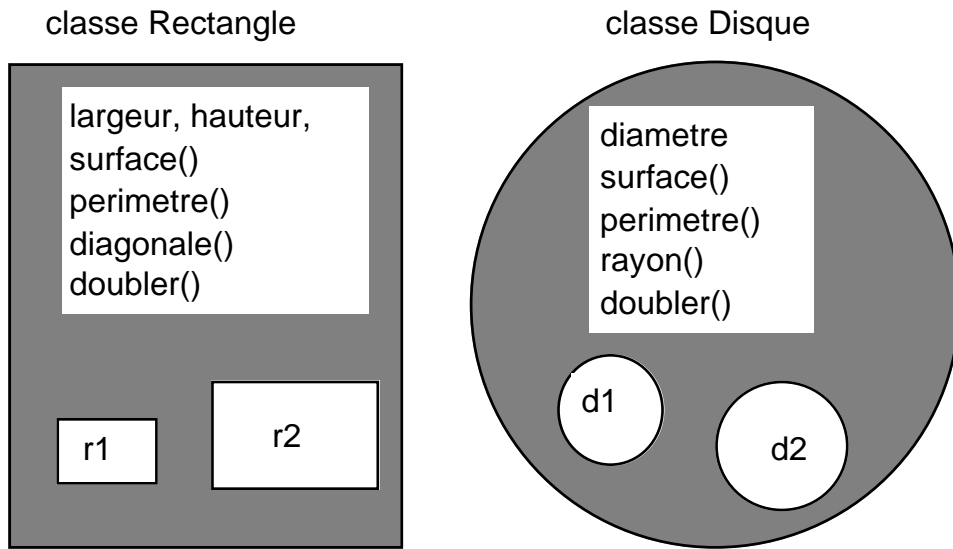
Définition de *méthode*:

Une méthode définit l'action élémentaire que l'on peut effectuer sur un objet. L'ensemble des méthodes définissent le comportement de la classe.

Définition de *message*:

Un message est l'occurrence de la demande d'exécution d'une méthode à un objet.

# classes, instance,méthodes, ...



# Classe en JAVA, pas à pas

## déclarer une classe

```
class Rectangle extends Object{
    ...
}
```

## variables d'instances. *largeur* et *hauteur*

```
class Rectangle extends Object{
    public double largeur, hauteur;
    ...
}
```

## méthodes de Rectangle

```
class Rectangle extends Object{
    ...
    public double perimetre() {
        return 2*(largeur+hauteur);
    }

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

    public double diagonale() {
        return Math.sqrt(largeur*largeur+hauteur*hauteur);
    }

    public void doubler() {largeur*=2; hauteur*=2;}
}
```

# Classe en JAVA, pas à pas

le constructeur (dit comment créer une instance)

```
class Rectangle extends Object{
    ...
    public Rectangle(double initL, double initH){
        largeur=initL;
        hauteur=initH;
    }
    ...
}
```

## Le code complet de la classe Rectangle

```
class Rectangle extends Object{
    public double largeur, hauteur;

    public Rectangle(double initL, double initH){
        largeur=initL;
        hauteur=initH;
    }

    public double perimetre() {return 2*(largeur+hauteur);}
    public double surface() {return largeur*hauteur;}
    public double diagonale() {
        return Math.sqrt(largeur*largeur+hauteur*hauteur);}
    public void doubler() {largeur*=2; hauteur*=2;}
}
```

## Classe Disque

```
class Disque extends Object{
    public double diametre;
    private static final double pi=3.14159;

    public Disque(double initD){
        diametre=initD;
    }
    public double perimetre() {return pi*diametre;}
    public double surface() {return (pi*diametre*diametre)/4;}
    public double rayon() {return diametre/2;}
    public void doubler() {diametre*=2;}
}
```

## Utiliser des classes: C o=new C()

La déclaration d'une instance

- déclarer le type de la variable (sa classe)
- utiliser un constructeur (la variable)

### déclaration de type

```
Disque d1,d2;
```

par un mot clé *new* qui indique l'on va crée une instance (création des variables de l'instance)

```
d1=new Disque(2);  
d2=new Disque(4);
```

contractées en une seule instruction

```
Rectangle r1=new Rectangle(2,4);  
Rectangle r2=new Rectangle(3,4);
```

## Utiliser les méthodes: o.m()

**instance.méthode(paramètres, ...)**

Pour obtenir la diagonale de r1

```
r1.diagonale()
```

écrire un petit programme qui utilise les classes Rectangle et Disque.

noms commençant par une majuscule sont réservés aux identifiants  
des classes

# TestRectDisk1

```
class TestRectDisk1{

    public static void main (String args[]) {

        Rectangle r1=new Rectangle(2,4);
        Rectangle r2=new Rectangle(3,4);

        System.out.println("diagonale de r1: "+r1.di-
agonale());
        System.out.println("p rim tre de r2:
"+r2.perimetre());
        r2.doubler();
        System.out.println("p rim tre de r2:
"+r2.perimetre());

        Disque d1,d2;

        d1=new Disque(2);
        d2=new Disque(4);
        System.out.println("rayon de d1: "+d1.ray-
on());
        System.out.println("p rim tre de d2:
"+d2.perimetre());
        d2.doubler();
        System.out.println("p rim tre de d2:
"+d2.perimetre());

    }
}
```

```
diagonale de r1: 4.47214
p rim tre de r2: 14
p rim tre de r2: 28
rayon de d1: 1
p rim tre de d2: 12.5664
p rim tre de d2: 25.1327
```

## Une classe avec plusieurs constructeurs

offrir plusieurs constructeurs, pour une même classe travailler dans plusieurs formes de représentation

créer un nouveau constructeur qui doit se distinguer par le nombre et/ou le type des paramètres.

un constructeur à la classe Rectangle

**coin supérieur gauche (csgx,csgy)**

**coin inférieur droite (cidx,cidy)**

```
class Rectangle extends Object{
    ...
    public Rectangle(double initL, double initH){
        largeur=initL;
        hauteur=initH;
    }

    public Rectangle(double csgx, double csgy,
                    double cidx, double cidy){
        largeur=Math.abs(csgx-cidx);
        hauteur=Math.abs(csgy-cidy);
    }
    ...
}
```

## TestRectDisk2

```
choisir comment déclarer les instances de Rectangle:
class TestRectDisk2{

    public static void main (String args[]) {

        Rectangle r1=new Rectangle(1,1,3,5);
        Rectangle r2=new Rectangle(3,4);
        ...
    }
}
```

## La surcharge des méthodes

- offrir plusieurs méthodes effectuant une action similaire
- sans devoir leurs donner des noms différents
- doit se distinguer par le nombre et/ou le type des paramètres.

ajouter une méthode doubler() à la classe Rectangle

```
class Rectangle extends Object{
    ...
    public void doubler() {
        largeur*=2;
        hauteur*=2;}

    public void doubler(boolean l, boolean h) {
        if (l) largeur*=2;
        if (h) hauteur*=2;
    }
    ...
}
```

choisir comment doubler les instances de Rectangle

```
class TestRectDisk2{
    public static void main (String args[]) {

        r2.doubler();
        r2.doubler(false,true);
        ...
    }
}
```

# Surcharge

surcharge des identificateurs de méthodes

Le compilateur décide de la bonne méthode en examinant

- les types paramètres
- leur nombre
- la hiérarchie.

autre exemple de surcharge:

```
r2.doubler();  
d2.doubler();
```

```
class B {}  
class SB {} extends B{  
    void f(B x, SB y){ }  
    void f(SB x, B y){ }  
}  
class X{  
    SB s;  
    f(s,s); ???  
    f(s,(B)s); ???  
}
```

# Variables de classe

## **la classe peut posséder des variables propres**

permet de conditionner le comportement de l'ensemble de ses instances

- déclarées statiques (static).
- initialisées au chargement de la classe.

classe disque, la constante pi est déclarée static.

accessibles à l'intérieur de la classe sans être préfixées

à l'extérieur de la classe en étant préfixée par l'objet.

## Variables d'instance

L'instruction suivante imprimera les dimensions de r2:

```
System.out.println("dimension:"+r2.largeur+
                  "/" +r2.hauteur);
r2.hauteur=45;
```

Il est possible de *cache* les variables d'instance pour éviter qu'on les modifie de l'extérieur avec deux modificateurs: *private* et *protected*

## Les méthodes de la classe

Les méthodes de la classe permettent de travailler sur la classe

- ajouter une méthode `modifierEchelle()`
- *static*.
- préfixant par le nom de la classe.

initialisations complexes des variables de la classe

associé un bloc d'instruction

exécuté au chargement de la classe.

```
static { instruction; ...}
```

```
class Rectangle extends Object{  
  
    public static double echelle=1.0;  
    public double largeur, hauteur;  
  
    ...  
}
```

## Exemple avec static

```
public static void modifierEchelle(double e){
    echelle=e;
}

static {
    //initialisation complex
    echelle=0.5;
}

class TestRectDisk3{
    public static void main (String args[]) {

        Rectangle r1=new Rectangle(2,4);
        Rectangle r2=new Rectangle(3,4);

        System.out.println("dim. r2:"+r2.largeur+"/
+r2.hauteur);
        Rectangle.modifierEchelle(4);
        System.out.println("echelle "+Rectangle.echelle);
    }
}
```

Résultat de l'exécution de TestRectDisk3:

```
dim. r2: 3/4
echelle 4
```

## Et si l'on parlait d'héritage

la relation d'héritage qui existe entre classes.

conserver certain acquis d'une autre classe

faire une classe Carre,

sans l'héritage,

redéfinir, les méthodes `perimetre()`, `surface()`, `diagonale()`, ...

**Alors qu'un carré c'est presque un rectangle.**

classe Carre étend le comportement de Rectangle

hérite de lui aussi ces variables d'instance et de ces méthodes.

un constructeur pour la classe Carre,

demande à être construit par la classe Rectangle

désignateur *super* qui permet de faire référence à la classe dont un objet hérite.

méthode `cote()` à la classe Carre.

désignateur *this* qui permet de faire référence à l'objet à qui la méthode est adressée

## TestRectDisk4

dans notre cas un carré particulier qui hérite du comportement d'un rectangle et donc qui possède aussi des variables d'instance largeur et hauteur.

```
class Carre extends Rectangle{

    public Carre(double initC){
        super(initC,initC);
    }

    public double cote() {return this.hauteur;}
}
class TestRectDisk4{
    public static void main (String args[]) {

        Carre c1=new Carre(4);

        System.out.println("dim. c1: "+c1.largeur+"/
"+c1.hauteur);
        System.out.println("surf. c1: "+c1.surface());
        System.out.println("coté de c1 "+c1.cote());
    }
}
```

Résultat de l'exécution de TestRectDisk4:

```
dim. c1: 4/4
surf. c1: 16
coté de c1 4
```

## de la surface au volume

étendre le comportement des rectangles à celui des parallélépipèdes.

- une variable d'instance pour mémoriser la profondeur
- le constructeur
- les méthodes propres à cette classe.

```
class Parallelepipede extends Rectangle{
    public double profondeur;

    public Parallelepipede(double x, double y, double z){
        super(x,y);
        profondeur=z;
    }
    public double volume(){return surface()*profondeur;}
}
```

```
class TestRectDisk5{
    public static void main (String args[]) {

        Parallelepipede p1=new Parallelepipede(2,3,4);

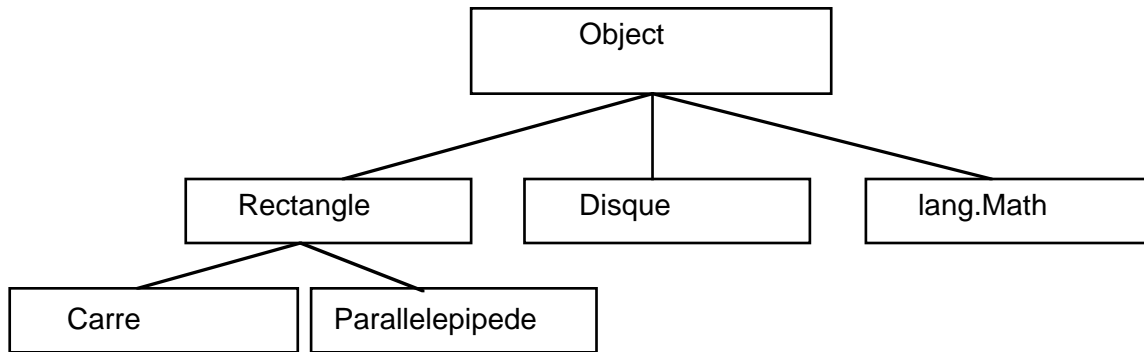
        System.out.println("prof. p1:"+p1.profondeur);
        System.out.println("surf. p1:"+p1.surface());
        System.out.println("vol. p1:"+p1.volume());
    }
}
```

Résultat de l'exécution de TestRectDisk4:

```
prof. p1:4
surf. p1:6
```

# Héritage

l'arbre des liens d'héritage.



toutes les classes ont pour origines la classe Object.  
Les classes ne peuvent étendre qu'une seule classe  
donc une hiérarchie stricte.

## Classe et méthodes abstraites

méthodes en commun entre les classes

constante

*servir de modèle pour ses sous-classes*

les méthodes sont aussi abstraites,

seule est donné la définition de la classe

son implémentation doit être définie dans la sous-classe.

Une classe abstraite ne peut pas avoir d'instance

Le modifier *abstract*

pour spécifier que la classe ou la méthode est abstraite.

## classe abstraite GeometriePlane

avec les deux méthodes abstraites

perimetre()

surface()

une constante pi.

Les classes Rectangle et Disque

étendent cette nouvelle classe

et donne une implémentation pour les méthodes abstraites (en plus de leurs propres méthodes )

```
abstract class GeometriePlane{
    public static final double pi=3.14159;
    public abstract double perimetre();
    public abstract double surface();
}
```

## étendre la classe abstraite

```
class Rectangle extends GeometriePlane{
    ...
    public double perimetre() {
        return 2*(largeur+hauteur);
    }
    public double surface(){
        return largeur*hauteur;
    }
    ...
}
```

```
class Disque extends Object{
    ...
    public double perimetre() {
        return GeometriePlane.pi*diametre;
    }
    public double surface() {
        return (GeometriePlane.pi*diametre*diametre)/4;
    }
}
```

en regardant la structure, nous savons que toutes les sous-classes d'une classe abstraite implémente ses méthodes. Si:

```
class Triangle extends GeometriePlane{...}
```

nous savons que des méthodes périmètre() et surface() sont disponibles sur les instances de triangles.

permet donc de bien structurer le développement.

## **l'héritage multiple vs interface.**

l'héritage simple; un objet ne peut être considéré que d'une seule manière.

considérer un objet comme appartenant à deux classes simultanément.

pas directement possible en Java,

### **un autre concept interface.**

- un ensemble de méthodes (seulement la définition)
- et de constantes.

Une classe implémente une interface (ou plusieurs)

elle s'engage alors à donner une implémentation pour toutes les méthodes définies dans l'interface.

En retour, on peut considérer des instances de cette classe comme des instances de cette interface (bien qu'il n'y a pas de création à proprement parler)

Les interfaces peuvent:

- implémenter des interfaces
- constituer une autre hiérarchie (forêt).

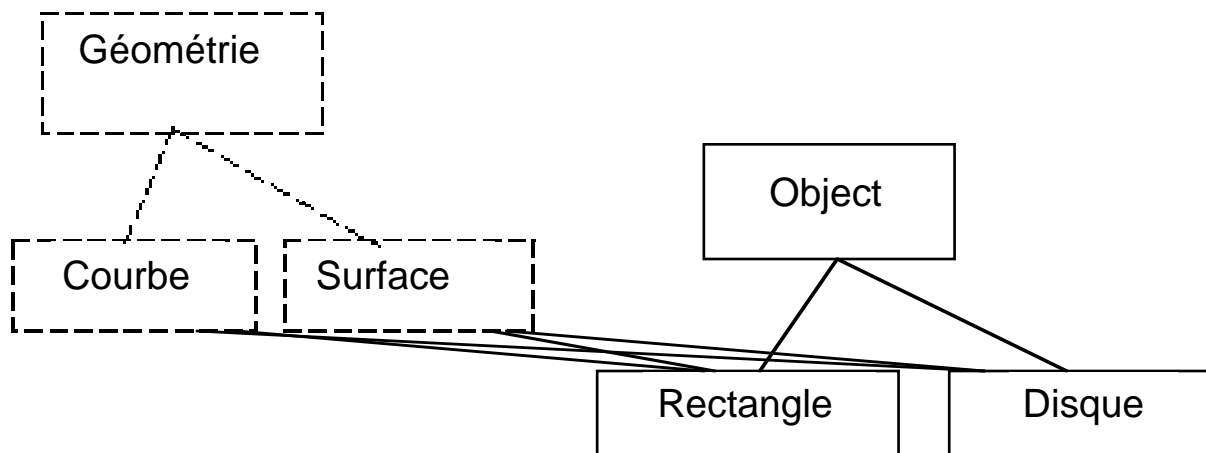
permet de briser l'appartenance à une seule classe.

## interface Geometrie , ...

```
interface Geometrie{  
    public static final double pi=3.14159;  
}
```

```
interface Courbe extends Geometrie{  
    public double longueur();  
    public void doubler();  
}
```

```
interface Surface extends Geometrie{  
    public double surface();  
}
```



## les classes implémentent des interfaces.

```
class Rectangle extends Object
    implements Courbe, Surface{
    public double largeur, hauteur;

    public Rectangle(double initL, double initH){
        largeur=initL;
        hauteur=initH;
    }
    public double longueur(){2*(largeur+hauteur);}
    public void doubler() {largeur*=2; hauteur*=2;}
    public double surface(){return largeur*hauteur;}
    public double diagonale(){
        return Math.sqrt(largeur*largeur+hauteur*hauteur);}
}
class Disque extends Object
    implements Courbe, Surface{
    public double diametre;

    public Disque(double initD){
        diametre=initD;
    }
    public double longueur() {return Geometrie.pi*diametre;}
    public void doubler() {diametre*=2;}
    public double surface(){
        return (Geometrie.pi*diametre*diametre)/4;}
    public double rayon() {return diametre/2;}
}
```

## On parle de polymorphisme

rectangles et disques comme des courbes ou des surfaces.

```
class TestRectDisk7{
    public static void main (String args[]) {
        Courbe c[]=new Courbe[10];
        Surface s[]=new Surface[10];
        Rectangle r[]=new Rectangle[5];
        Disque d[]=new Disque[5];

        // créer des rectangles
        for (int i=0;i<5;i++){
            r[i]=new Rectangle(i+1,i+1);
            c[i]=r[i];
            s[i]=r[i];
        }
        // créer des disques
        for (int i=0;i<5;i++){
            d[i]=new Disque(i+1);
            c[i+5]=d[i];
            s[i+5]=d[i];
        }

        // manipuler comme objets de plusieurs classes
        System.out.println("surf. r[2]: "+r[2].surface());
        System.out.println("surf. s[2]: "+s[2].surface());
        System.out.println("long. r[2]: "+r[2].longueur());
        System.out.println("long. c[2]: "+c[2].longueur());
    }
}
```

## **// considerer comme un tout**

```
//surface de tous les objets
double surfTotal=0;
for (int i=0;i<10;i++)surfTotal+=s[i].surface();
System.out.println("surfTotal: "+surfTotal);
// doubler tous les périmètres
for (int i=0;i<10;i++)c[i].doubler();
//surface de tous les objets
surfTotal=0;
for (int i=0;i<10;i++)surfTotal+=s[i].surface();
System.out.println("surfTotal: "+surfTotal);
}
}
```

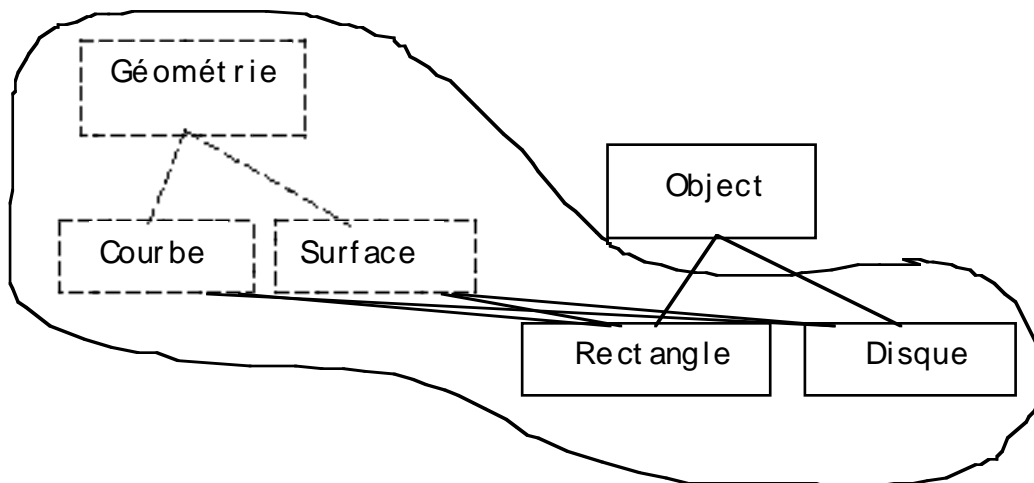
Exécution de TestRectDisk7:

```
surf. r[2]: 9
surf. s[2]: 9
long. r[2]: 12
long. c[2]: 12
surfTotal: 98.1969
surfTotal: 392.787
```

# regrouper des classes dans un package

regroupement physique  
(dans le même répertoire)

schéma de nommage host....répertoire...class



dans un fichier ...

```
package geo;
public interface Geometrie{
    public static final double pi=3.14159;
}
```

dans un autre fichier;

```
import geo.*;
class TestRectDisk7{ ... }
```

## typer un objet (casting)

changer le type de l'objet

(respecter la hiérarchie, implémentation)

### (Class) objet

(Object) o toujours possible!

exemple:

```
class B {}
class SB {} extends B{
    void f(B x, SB y){ }
    void f(SB x, B y){ }
}
class X{
    SB s;
    f(s,s); ???
    f(s,(B)s); ???
}
```

## copier des objets

```
Rectangle r1=new Rectangle(3,4);  
Rectangle r2=new Rectangle(6,8);
```

```
r1=r2  
r2.doubler();
```

que vaut r1.hauteur() ?

```
r1=r2.clone()  
r2.doubler();
```

que vaut r1.hauteur() ?

### Travaille par référence

```
r1  
    hauteur=...  
    largeur=...  
r2
```

## comparer des objets

```
Rectangle r1=new Rectangle(1,2);  
Rectangle r2=new Rectangle(1,2);
```

```
if r1==r2 ; // oui  
else ; // non;
```

```
r1=r2
```

```
if r1==r2 ; // oui  
else ; // non;
```

```
if (r1.equals(r2)) ; // oui  
else ; // non;
```

```
r1  
    hauteur=...    equals()  
==    largeur=...  
r2
```