

Faculté des Sciences  
Département de Mathématiques et  
d'informatique  
Laboratoire de Recherches en Informatique

## Cours & Exercices POO : Le langage Java

SMI-S5

Mme I. ARRASSEN

5/12/2024

Cours et Exercices Java

1

## Plan du cours

- Classes et objets
- L'héritage
- Les tableaux
- Les chaînes de caractères
- La gestion de exceptions

5/12/2024

Cours et Exercices Java

2

# Chapitre 1 : Classes et Objets

5/12/2024

Cours et Exercices Java

3

## Plan chapitre 1: Classes et Objets

- *Notion de Classe, d'Objet*
- *Éléments de conception de classe*
- *La notion de constructeur*
- *L'attribut final*
- *Champs et Méthodes de classes*
- *Surdéfinition(surcharge) de Méthodes*
- *Autoréférence : le mot clé this*
- *Les paquetages*

5/12/2024

Cours et Exercices Java

4

### Notion d'Objet

- Un objet est une variable améliorée: il stocke les données
- On peut effectuer des requêtes sur cet objet en envoyant un message à cet objet. Ceci est équivalent à un appel de fonction.
- Chaque objet a un type précis: c'est-à-dire chaque objet est une instance (variable) d'une classe (type)
- Un programme est un ensemble d'objets qui s'envoient des messages.

5/12/2024

Cours et Exercices Java

5

### Notion de Classe

- L'entité de base de tout code Java est la classe.
- Tout se trouve dans une classe.
- Pas de déclarations/code en dehors du corps d'une classe.
- Une classe contient :
  - des déclarations de variables globales, appelées "attributs",
  - et des méthodes (équivalents à des fonctions).
- Le code se trouve exclusivement dans le corps des méthodes.

5/12/2024

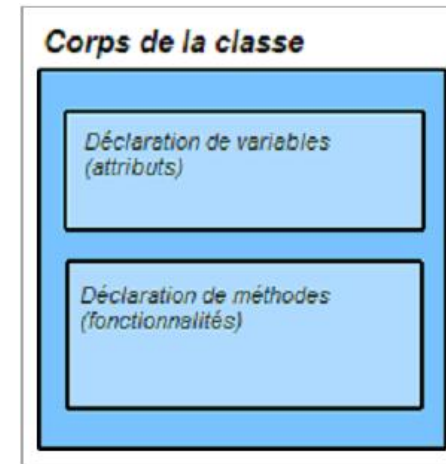
Cours et Exercices Java

6

## Notion de Classe, d'Objet

- Dans la définition de la classe, on met deux types d'éléments:
  - des données membres de la classe (champs) qui sont des objets de n'importe quel type.
  - des fonctions membres de la classe (méthodes).

## Notion de Classe, d'Objet



## Notion de Classes, d'Objets

### Déclaration et Création d'objets

- Pour rappel, déclarer une variable, c'est lui donner un nom et dire de quel type elle est.
- Pour les variables de type objet, le type précise le nom de la classe d'objets correspondant.

•→ Exemples d'illustrations

5/12/2024

Cours et Exercices Java

9

## Notion de Classe, d'Objet

### Exemples :

- une **Renault clio** , **rouge**, est un objet (instance) de la classe **Voiture**.
- **Mohammed**, **22 ans**, célibataire, est un objet (instance) de la classe **Etudiant**.
- Un **tigre de neige** est un objet (instance) de la classe **Animal**
- ...

5/12/2024

Cours et Exercices Java

10

## Notion de Classe, d'Objet

### Déclaration d'une variable de type objet

#### Exemple :

Déclaration d'une variable de type objet dont le nom est – a –

et qui pourra contenir un objet de type Point, c'est-à-dire un point du plan.

**Point a;**

## Notion de Classe, d'Objet

### Création

**a= new Point(2,3);**

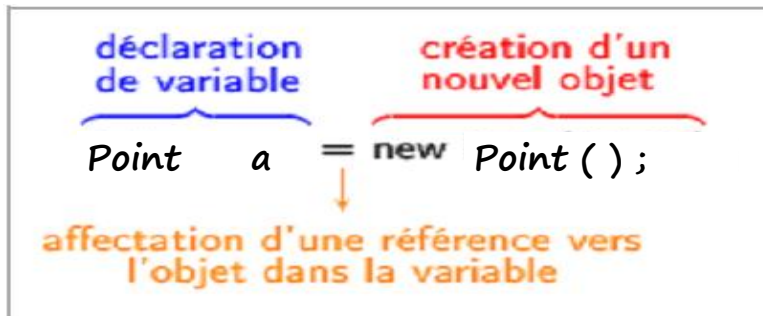
Création d'un objet d'une classe/instanciation d'une classe.

Le nouvel objet créé est appelé instance de la classe **Point**.

Ensuite, il y a l'affectation de l'objet nouvellement créé à la variable a.

## Notion de Classe, d'Objet

on peut faire la déclaration et initialiser la variable en une seule instruction



5/12/2024

Cours et Exercices Java

13

## Notion de Classe, d'Objet

### Exercice 1:

```

public class Point {
    public static void main(String args[]){
        Point a;
        a=new Point();
        a.x=2;
        a.y=3;
        System.out.println( " abscise= " + a.x + "
ordonne "+ a.y);
    }
    private int x,y;
}
  
```

5/12/2024

Cours et Exercices Java

14

### De quoi a-t-on besoin pour écrire, compiler et exécuter un programme java ?

- Une plate forme Java  
**j2sdk1.4.1\_07** : Java 2 Standard Développement Kit (Sun Microsystems)  
Éditeur de texte : **Bloc Note**
- Ou bien une plate forme de développement avec un environnement graphique **Eclipse Java Development Tools 2.1(IBM)**

5/12/2024

Cours et Exercices Java

15

### Compiler et Exécuter Point.java

- **Lancer l'invité de commandes**
- **Allez dans le dossier**  
**C:\j2sdk1.4.1\_07\bin**
- **javac Point.java**
- **java Point**

5/12/2024

Cours et Exercices Java

16



### Exercice 2: la classe point.java

```
class Point {
    public void initialise(int abs, int ord){// méthode d'altération
        x= abs;
        y=ord;
    }

    public void affiche(){//méthode d'accès
        System.out.println("abscisse=" + x + "ordonné" + y );
    }

    private int x;
    private int y;
}
```

5/12/2024

Cours et Exercices Java

17

### Exercice 2: la classe testpoint.java

```
public class TestPoint {
    public static void main(String[] args){
        Point a;
        a=new Point();
        a.initialise(2,3);
        a.affiche();
    }
}
```

5/12/2024

Cours et Exercices Java

18

## Compiler et Exécuter les fichiers \*.java

- Lancer l'invité de commandes
- Allez dans le dossier  
C:\j2sdk1.4.1\_07\bin

- javac Point.java
- javac TestPoint.java
- java TestPoint

5/12/2024

Cours et Exercices Java

19

## Ajouter une nouvelle méthode

Ajouter la méthode d'altération suivante:

```
public void deplace(int dx,int dy)
{ x+=dx;
  y+=dy;
}
```

Appeler la méthode dans testpoint.java

```
a.deplace(1,3);
```

Enregistrer votre fichier.

Exécuter les commandes suivantes :

- javac point.java
- javac testpoint.java
- java testpoint

5/12/2024

Cours et Exercices Java

20

## Encapsulation et modificateur de visibilité

le mot réservé **private** : qui apparait dans leurs déclarations, permet de modifier la visibilité de ces variables.

Dès que l'on possède une instance de la classe, on peut utiliser les méthodes/les attributs de la classe avec l'opérateur point: '.'

le point '.' permet d'accéder à n'importe quel membre d'une classe à partir d'une instance de la classe.

5/12/2024

Cours et Exercices Java

21

## Encapsulation et Modificateur de visibilité

On pourrait donc écrire un programme qui affiche la valeur des coordonnées du point d'un plan en utilisant l'opérateur point '.' et en spécifiant le nom de la variable d'instance, à savoir x et y.

Mais le programme ne va pas compiler, on peut lire l'erreur de compilation suivante à la console :

**The field Point.x is not visible**  
**The field Point.y is not visible**

la variable x n'est pas visible depuis cette méthode

5/12/2024

Cours et Exercices Java

22

## Encapsulation et modificateur de visibilité

- En effet, le modificateur de visibilité **private** restreint la visibilité de la variable à la classe dans laquelle elle est déclarée.
- Il n'y a donc que depuis le corps de la classe Point que l'on pourra accéder aux variables d'instances de la classe Point en utilisant une instance de la classe Point et l'opérateur point (.)

5/12/2024

Cours et Exercices Java

23

## Notion de Classe, d'Objet

### Notion de portée

Le concept de portée fixe simultanément la visibilité et la durée de vie des noms définis dans cette portée.

La portée est fixée par les accolades { }.

### Exemple:

```
{
    int n=12; // seul n est accessible
    {
        int m=96; // n et m tous les deux sont accessibles
    }
    // seul n est accessible
    // m est hors de portée
}
```

Cours et Exercices Java

## Notion de Classe, d'Objet

Attention: Ceci n'est pas permis en Java

```
{
  int n=12;
  {
    int m=96; // illégale en Java, valable en C, C++
  }
}
```

Cours et Exercices Java

## Notion de Classe, d'Objet

Portée des objet: Considérons une classe nommée A

```
{
  A a=new A();
} // fin de portée
```

- La référence *a* disparaît à la fin de la portée,
- l'objet qui été référencé par *a* existe toujours, mais on n'a aucun contrôle sur lui (il reste inaccessible).
- Les objets n'ont pas la même durée de vie que les types primitifs.

Cours et Exercices Java

## Notion de Classe, d'Objet

- Il n'existe aucun opérateur explicite pour détruire l'objet dont on n'a pas besoin,
- Il existe un mécanisme de gestion automatique de la mémoire connu sous le nom de ramasse miettes (en anglais Garbage Collector). Son principe est le suivant:
  - A tout instant on connaît le nombre de références à un objet donné.
  - Lorsqu'il n'existe plus aucune référence sur un objet, on est certains que le programme ne peut pas y accéder, donc l'objet devient candidat au ramasse miette
  - Il est nécessaire pour éviter de saturer la mémoire

Cours et Exercices Java

## Stockage en mémoire

Les objets et les valeurs de type primitif ne sont pas stockés de la même façon en mémoire

- Variable de type primitif:
  - Réserve de mémoire pour la valeur

```
int i = 45;
double d = 7.12;
boolean b = true;
```

Cours et Exercices Java

## Stockage en mémoire

### • Variable de type objet (2 étapes):

- Réserve de mémoire pour une référence vers les variables d'instance.

*Rectangle r;*

- Réserve de mémoire pour les variables d'instance.

*r = new Rectangle(1.0, 2.0);*

### • Situation en mémoire ...

Cours et Exercices Java

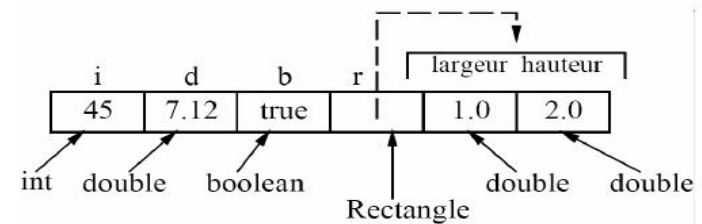
## Stockage en mémoire

### • Exemples d'affichage:

```
System.out.println(i); // 45
```

```
System.out.println(r); // 80ca178
```

```
System.out.println(r.largeur); // 1.0
```

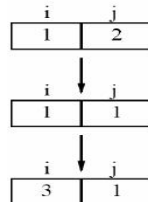


Cours et Exercices Java

## Stockage en mémoire

### Variables de type primitif

```
int i = 1;
int j = 2;
System.out.println(j);           // 2
j = i;                           // copie valeur
System.out.println(j);           // 1
i = 3;
System.out.println(j);           // 1
```



Cours et Exercices Java

## Variables de type objet

```
Rectangle r1 = new Rectangle(1.0, 2.0);
Rectangle r2 = new Rectangle(3.0, 4.0);
```

```
System.out.println(r2.largeur); // 3.0
r2 = r1; // copie référence
```

```
System.out.println(r2.largeur); // 1.0
r1.largeur = 5.0;
```

```
System.out.println(r2.largeur); // 5.0 !!!
```

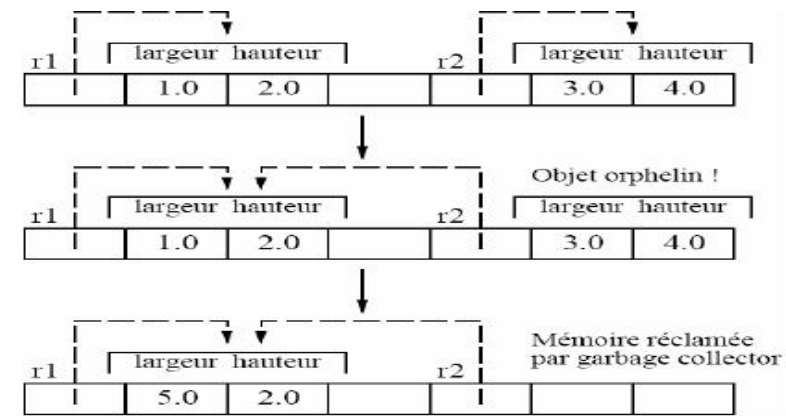
5/12/2024

Cours et Exercices Java

32



### Variables de type objet



5/12/2024

Cours et Exercices Java

33

### Éléments de conception de classe

Une bonne conception OO s'appuie sur la notion de contrat :

-> une classe est caractérisée par un ensemble de service définis par :

- Les entêtes de ses méthodes publiques
- Le comportement de ces méthodes

Les champs et les méthodes privés, le corps des méthodes public ne sont pas sensé être connu de l'utilisateur => l'implémentation de la classe

5/12/2024

Cours et Exercices Java

34

## Éléments de conception de classe

Une classe comporte différents types de méthodes:

- les constructeurs
- **Les méthodes d'accès** (en anglais accessor) : fournissent des informations relatives à l'état d'un objet, c'est-à-dire aux valeurs de certains de ses champs(généralement privés) sans les modifier. (getXXXXX)
- **Les méthodes d'altération** (en anglais mutator) qui modifient l'état d'un objet, donc les valeurs de certains de ses champs. (setXXXXX)

5/12/2024

Cours et Exercices Java

35

## La notion de constructeur

- La notion de constructeur permet **d'automatiser le mécanisme d'initialisation** d'un objet.
- Un constructeur est une **méthode sans valeur de retour portant le même nom de la classe**.

5/12/2024

Cours et Exercices Java

36

### Exemple : point.java avec constructeur

```
class point
{
    public point(int abs, int ord) { //constructeur
        x= abs;
        y=ord;
    }
    public void affiche() {
        System.out.println("abscisse=" + x + "ordonné"+y );
    }
    private int x;
    private int y;
}
}
```

5/12/2024

Cours et Exercices Java

37

### Suite de l'exemple

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

        point a;
        a=new point(2,3);
        a.affiche();
    }
}
```

5/12/2024

Cours et Exercices Java

38

Un autre exemple

```

public class Etudiant {
    private String nom, prenom; // Variables d'instance
    private int codeNatEtudiant;
    // Constructeur
    public Etudiant(String n, String p) {
        nom = n;
        prenom = p;
    }
    public void setCNE (int cne) {
        codeNatEtudiant = cne;
    }
}

```

Questions:

1.Écrire une méthode main qui utilise la classe Etudiant

5/12/2024

Cours et Exercices Java

39

Un autre exemple

```

1 public class Die
2 {
3     // Les variables d'instance
4     private int nbFaces;
5     private int visibleFace;
6
7     // Constructeur
8     public Die()
9     {
10         nbFaces = 6;
11         visibleFace = (int) (Math.random() * nbFaces) + 1;;
12     }
13
14     // Les méthodes
15 }

```

Questions:

1.Écrire une méthode main qui utilise la classe Die.

5/12/2024

Cours et Exercices Java

40

## Règles concernant les constructeurs

1. Aucun type ne doit figurer devant son nom
2. Si une classe ne dispose pas d'un constructeur -> instancier les objets comme s'il existait un constructeur:  
`a= new point();`
3. Un constructeur ne peut pas être appelé depuis un objet, comme une autre méthode:  
`a.point(2,3);` // interdit
4. Un constructeur peut appeler un autre constructeur (surdéfinition).
5. Un constructeur peut être privé (intérêt lorsque il existe un autre constructeur dans la classe)

5/12/2024

Cours et Exercices Java

41

## Construction et initialisation d'objets

- Initialisation par défaut des champs d'un objet  
 Un objet est créé, ses champs sont initialisés à une valeur par défaut null
- Initialisation explicite des champs d'un objet  

```
class A
{ public A (..){.... .....} //constructeur de A
....
private int n=10;
private int p;
}
• A a = new A{.....}
```

5/12/2024

Cours et Exercices Java

42

### Initialisation par défaut des champs d'un objet

Type du champ	Valeur par défaut
boolean	false
char	caractère de code nul
int, byte, short, int long	0
float, double	0.f ou 0.
class	null

5/12/2024

Cours et Exercices Java

43

### Appel du constructeur

- Le corps du constructeur n'est exécuté qu'après :
  - l'initialisation par défaut et
  - l'initialisation explicite.

5/12/2024

Cours et Exercices Java

44

## Appel du constructeur

### Exercice 4

```
public class Init
{ public static void main(String args[])
  { A a = new A(); //ici a.n vaut 5
    a.affiche();
  }
}

class A
{ public A(){
  System.out.println("n= "+n); //ici n vaut 20
  n=5;
}
  public void affiche(){
    System.out.println("n= "+n);}
  private int n=20;
}
```

5/12/2024

Cours et Exercices Java

45

## L'attribut final

**Définition :** Un champs peut être déclaré avec l'attribut *final*. C'est à dire qu'il n'est initialisé qu'une seule fois.

### Exercice 5

```
public class Init
{ public static void main(String args[])
  { A a = new A();
    a.affiche();
  }
}

class A
{ public A(){
  n=5;
}
  public void affiche()
  { System.out.println("n= "+n);}
}
  private final int n=20;
```

5/12/2024

Cours et Exercices Java

46

## L'attribut final

### Exercice 5

```
public class Init
{   public static void main(String args[])
    {   A a = new A();
        a.affiche();
    }
}
class A
{   public A(){
    n=5;
    }
    public void affiche(){
        System.out.println("n= "+n);}

    Private final int n=20;
}
```

5/12/2024

Cours et Exercices Java

47

## Champs et Méthodes de classes

### Définition :

- Un champs de classe est un champ statique qui n'existe qu'en un seul exemplaire pour toute les instances des objet de la classes.
- Une méthode de classe est une méthode statique qui peut être appelé indépendamment de tout objet de la classe (ex: méthode main)

5/12/2024

Cours et Exercices Java

48



## Champs et Méthodes de classes

### Exemple: champs de classe

```
class A
{
static int n;
float y;
}
```

L'écriture : A a1=new A(), a2=new A();  
Les notations a1.n, a2.n, A.n sont équivalentes

5/12/2024

Cours et Exercices Java

49

## Champs et Méthodes de classes

### Exemple: méthodes de classe

```
class A
{
static int n;
float y;
Public static void f{.....}
}
```

.....  
A.f();//appelle la méthode de classe f de la classe A  
a.f();// reste autorisé mais déconseillé.

5/12/2024

Cours et Exercices Java

50

## Surdéfinition(Surcharge) de Méthodes

### Définition :

plusieurs méthodes peuvent avoir:

- Le même nom.
- Le nombre et le type d'argument permet au compilateur d'effectuer un choix.

5/12/2024

Cours et Exercices Java

51

## Surdéfinition(Surcharge) de Méthodes

### Exercice:

```
class point{
    public point (int abs, int ord)
        {x=abs; y=ord;}
    public void deplace (int dx, int dy)
        {x+=dx; y+=dy;}
    public void deplace (int dx)
        {x+=dx;}
    public void deplace (short dx)
        {x+=dx;}

    private int x, y;
}
```

5/12/2024

Cours et Exercices Java

52

## Surdéfinition(Surcharge) de Méthodes

Exercice (suite):

```
public class surdef1
{public static void main(String arg[])
Point a = new point(2,3);
```

```
//appelle deplace (int,int)
a.deplace(1,1);
a.affiche();
```

```
//appelle deplace (int)
a.deplace(2);
```

```
//appelle deplace (short)
short p = 3;
a.deplace(p);
```

```
//appelle deplace (short) après conversion de b en short
byte b=2;
a.deplace(b);}
}
```

5/12/2024

Cours et Exercices Java

53

## Surdéfinition(Surcharge) de Méthodes

En cas d'ambiguïté:

Supposons que notre classe Point est dotée des deux méthodes déplace suivantes:

```
public void deplace (int dx, byte dy)
{ x+=dx; y+=dy; }
```

```
public void deplace (byte dx, int dy)
{ x+=dx;      }
```

5/12/2024

Cours et Exercices Java

54

### Surdéfinition(Surcharge) de Méthodes

Considérons les instructions :

```
Point a = new Point();
```

```
int n; byte b;
```

```
a.deplace (n,b);
```

*//ok, appel de deplace(int, byte)*

```
a.deplace (b,n);
```

*// ok, appel de deplace(byte, int)*

```
a.deplace (b,b);
```

*//erreur : ambiguïté, cet appel est refusé  
par le compilateur*

5/12/2024

Cours et Exercices Java

55

### Surdéfinition(Surcharge) de Méthodes

```
class Additionneur {
    static int somme(int a, int b) // 1
    {
        return (a+b);
    }
    static int somme(int a, int b, int c) // 2
    {
        return (a+b+c);
    }
    static float somme(float a, float b) // 3
    {
        return (a+b);
    }
    static float somme(int a, int b) // interdit à cause de 1
    {
        return ((float)a+(float)b);
    }
}
```

5/12/2024

Cours et Exercices Java

56

## Surdéfinition(Surcharge) de Méthodes

### Règles générales

À la rencontre d'un appel donné, le compilateur recherche toutes les méthodes acceptables et il choisit la meilleure si elle existe. Pour qu'une méthode soit acceptable, il faut :

- Qu'elle dispose du nombre d'arguments voulus.
- Que le type de chaque argument effectif soit compatible par affectation avec le type de l'argument muet correspondant.
- Qu'elle soit accessible

5/12/2024

Cours et Exercices Java

57

## Autoréférence : le mot clé this

### Exemple d'utilisation de this

```
public boolean coincide (point pt)
{
    return ((pt.x==this.x) && (pt.y==this.y));
}
```

### Un autre exemple d'utilisation de this

```
public point(int x, int y)
{ this.x=x;
  this.y=y;
}
```

5/12/2024

Cours et Exercices Java

58

## Affectation et comparaison d'objets

*L'affectation dans le concept orientée objet porte sur les références et non sur les objets eux-mêmes.*

### Exemple

*On dispose d'une classe Point.java possédant un constructeur à deux arguments entiers :*

*Point a,b;*

*a= new Point(3,5);*

*b=new Point(2,5);*

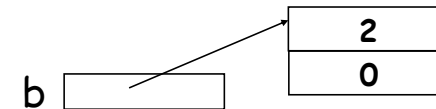
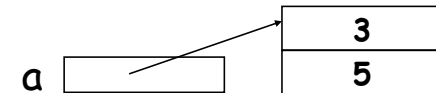
*Après exécution on aboutit à la situation suivante:*

5/12/2024

Cours et Exercices Java

59

## Affectation et comparaison d'objets



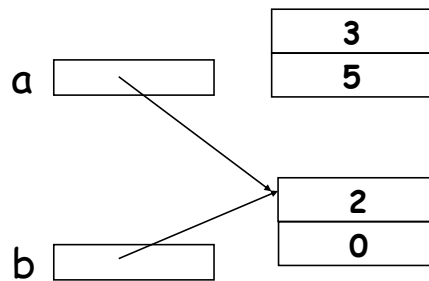
5/12/2024

Cours et Exercices Java

60

## Affectation et comparaison d'objets

Après exécution de l'instruction :  $a=b$



*a et b désignent le même objet, et non pas deux objets de même valeur.*

5/12/2024

Cours et Exercices Java

61

## Les paquetages

Définition: Regroupement logique sous un identificateur commun d'un ensemble de classes. cette notion est proche de la notion de bibliothèque que l'on rencontre dans d'autres langages .

Le but est donc une organisation logique des classes

Attribution d'une classe à un paquetage

Mettre en début du fichier source :

```
package xxxxxx;
```

xxxxxx représente le nom du package

5/12/2024

Cours et Exercices Java

62

## Les paquetages

### Utilisation d'une classe d'un paquetage

#### *En citant le nom du package*

```
mesclasses.point p=new Mesclasses.point(2,3);
```

#### *En important une classe*

```
import Mesclasses.point, Mesclasses.cercles;
```

#### *En important un package*

```
import Mesclasses.*
```

5/12/2024

Cours et Exercices Java

63

## Les paquetages

### Les paquetage standards(détaillés dans le 1<sup>er</sup> chapitre)

Les nombreuses classes standards avec lesquelles java est fourni sont structurées en paquetage. On aura l'occasion d'utiliser certains d'entre eux par la suite.

```
java.awt
java.awt.event,
javax.swing
```

5/12/2024

Cours et Exercices Java

64



### Les paquetages et les droits d'accès

Avec le mot clé public, la classe est accessible à toutes les autres classes( moyennant éventuellement le recours à une instruction import)

Sans le mot clé public la classe n'est accessible qu'aux classes du même paquetage.

5/12/2024

Cours et Exercices Java

65

### Remarques

1. Ne pas confondre le droit d'accès à une classe avec le droit d'accès à un membre d'une classe, même si certains des mots clés utilisés sont communs. ainsi , le mot clé private a un sens pour un membre et non pas pour une classe
2. Dans le chapitre consacré à l'héritage nous verrons qu'il existe un autre droit d'accès appelé (protected)

5/12/2024

Cours et Exercices Java

66

### Les paquetages : Exercices

*Création de package dans eclipse.*

*Utilisation de la classe point et testpoint définis précédemment.*

## Chapitre 2 :

## L'héritage

## Plan chapitre 2 : L'héritage

- Accès d'une classe dérivée aux membres de sa classe de base
- Construction et initialisation des objets dérivés
- Dérivations successives
- Redéfinition et Surdéfinition de membres
- Le polymorphisme
- Le mot clé super
- Les classes abstraites
- Les interfaces

5/12/2024

Cours et Exercices Java

69

## L'héritage

Exemple : Un cas concret:

La classe Voiture représente toutes sortes de voitures possibles

- On pourrait définir un camion comme une voiture très longue, très haute, etc.
- Mais un camion a des spécificités vis-à-vis des voitures : remorque, cargaison, boîte noire, etc.
- On pourrait créer une classe Camion qui ressemble à la classe Voiture
- Mais on ne veut pas réécrire tout ce qu'elles ont en commun

5/12/2024

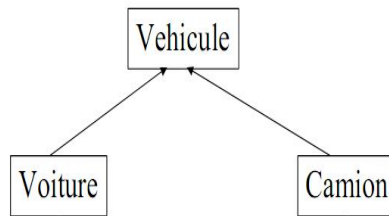
Cours et Exercices Java

70

## L'héritage

### Solution :

- La classe Vehicule contient tout ce qu'il y a de commun à Camion et Voiture
- Camion ne contient que ce qu'il y a de spécifique aux camions



5/12/2024

Cours et Exercices Java

71

## L'héritage

- La notion d'héritage constitue l'un des fondements de la programmation OO.
- L'héritage est un mécanisme qui permet à une classe d'hériter l'ensemble du comportement et des attributs d'une autre classe.
- Permet de définir une nouvelle classe, dite classe dérivée, à partir d'une classe existante dite classe de base (super classe).
- Cette nouvelle classe hérite d'emblée des fonctionnalités de la classe de base (champs et méthodes) qu'elle pourra modifier ou compléter à volonté.

5/12/2024

Cours et Exercices Java

72

## L'héritage

### Exemple : classe de base Point

```
class Point {
    public initialise(int abs, int ord)
    {
        x=abs;
        y=ord;
    }
    public void affiche()
    {
        System.out.println(x + << >>+y);
    }
    Private int x,y;
}
```

5/12/2024

Cours et Exercices Java

73

## L'héritage

### classe dérivée Pointcol

```
class Pointcol extends Point {
    public void colore (byte couleur)
    { this.couleur=couleur;
    }
    private byte couleur;
}
```

**Règle:** un objet de type pointcol peut faire appel :

- Aux méthodes public de pointcol
- Aux méthodes public de point (initialise , affiche...)

5/12/2024

Cours et Exercices Java

74

### Accès d'une classe dérivée aux membres de sa classe de base

1. Une classe dérivée n'accède pas aux membres privés de sa classe de base.

Exemple: ajouter une méthode affichec() dans pointcol

```
public void affichec()
{
    System.out.println( "je suis en:" + x + "et en" + y + );
    //interdit car x, y sont privés

    System.out.println( "ma couleur est : " + couleur);
}
```

5/12/2024

Cours et Exercices Java

75

### Accès d'une classe dérivée aux membres de sa classe de base

2. Une méthode d'une classe dérivée a accès aux membres publics de sa classe de base.

```
public void affichec()
{
    affiche( );
    System.out.println( "ma couleur est : " + couleur);
}
```

5/12/2024

Cours et Exercices Java

76

### Construction et initialisation des objets dérivés

1. Le constructeur de la classe dérivée doit prendre en charge l'intégralité de la construction de l'objet.

Le constructeur de pointcol doit:

- Initialiser le champs couleur.
- Appeler le constructeur de Point pour initialiser les champs x et y;

5/12/2024

Cours et Exercices Java

77

### Construction et initialisation des objets dérivés

2. Si un constructeur d'une classe dérivée appelle un constructeur d'une classe de base, il doit obligatoirement s'agir de la première instruction du constructeur et ce dernier est désigné par le mot clé **super**.

5/12/2024

Cours et Exercices Java

78

## Construction et initialisation des objets dérivés

### Exemple

```
class Point
{ public Point(int x, int y) //constructeur de la classe point
  { ... }
  private int x, y;
}

class PointCol extends Point
{ public PointCol (int x, int y, byte couleur) //constructeur
  //de la classe pointcol
  { super(x,y);
    this.couleur=couleur;}
  private byte couleur;
}
```

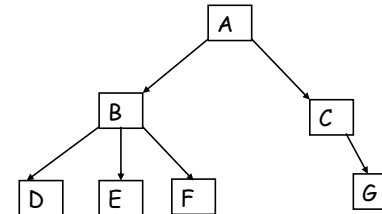
5/12/2024

Cours et Exercices Java

79

## Dérivations successives

- D'une même classe peuvent être dérivée plusieurs classes différentes.
- Les notions de classe de base et de classe dérivée sont relatives puisqu'une classe dérivée peut à son tour, servir de classe de base pour une autre.



D dérivée de B  
 B dérivée de A  
 A super classe de B  
 B super classe de D  
 A superclasse de D

5/12/2024

Cours et Exercices Java

80



## Redéfinition et Surdéfinition de membres

### Redéfinition :

- Une classe dérivée peut fournir une nouvelle définition d'une méthode d'une classe de base.
- La méthode porte le même nom, même nombre d'arguments et même type d'arguments.

### Exemple :

```
public void affiche()
{
    super.affiche(); // appel de la méthode affiche de la
                    // super classe.
    System.out.println(" et ma couleur est " + couleur);
}
```

5/12/2024

Cours et Exercices Java

81

## Redéfinition et Surdéfinition de membres

### Empêcher la redéfinition

Une classe peut protéger une méthode afin d'éviter qu'elle ne soit redéfinie dans ses sous-classes. En Java, on va simplement ajouter le modificateur `final` dans l'entête de la méthode.

```
public final void getMessage()
{
    // Corps de la méthode
}
```

5/12/2024

Cours et Exercices Java

82

## Redéfinition et Surdéfinition de membres

Surdéfinition : une classe dérivée peut surdéfinir une méthode d'une classe de base.

Exemple :

```
class A
{ public void f(int n) {.....}
....
}
```

```
class B extends A
{ public void f( float x) {....}
...
}
```

5/12/2024

Cours et Exercices Java

83

## Redéfinition et Surdéfinition de membres

Contraintes portant sur la redéfinition

Dans la surdéfinition :

on est pas obligé de respecter le type de la valeur de retour .

Dans le redéfinition :

java impose non seulement l'identité des signatures mais également celle du type de la valeur de retour.

5/12/2024

Cours et Exercices Java

84

## Le polymorphisme

Un objet de type Voiture peut utiliser toutes les méthodes de la classe Vehicule:

- Il doit disposer d' une valeur pour tous les attributs de la classe Vehicule.
- A tout moment, une méthode qui utilise un objet de type Vehicule peut manipuler un objet de type Voiture en guise de Vehicule.
- Cette dernière propriété est le polymorphisme.

5/12/2024

Cours et Exercices Java

85

## Le polymorphisme

```
class Vehicule {
// Vehicule() {}
}
```

```
class Voiture extends Vehicule {
int nbPortes;
double longueur;
```

```
Voiture(double lg, int nbP){
longueur = lg;
nbPortes = nbP;
}
}
```

5/12/2024

Cours et Exercices Java

86

```

class Garagiste {
    public boolean garer(Vehicule v)
    {
        v.demarrer();
        for (int pl=0; pl<nbPlaces; ++pl)
        {
            if (place[pl].estLibre())
            {
                v.amener(place[pl]);
                v.arreter();
                return true;
            }
        }
        System.out.println(<< Aucune place libre >>);
        return false;
    }
}

```

5/12/2024

Cours et Exercices Java

87

## Le polymorphisme

Le polymorphisme en Java se traduit par :

1. La compatibilité par affectation entre un type classe et un type ascendant ( la classe de base).
2. La ligature dynamique des méthodes.

5/12/2024

Cours et Exercices Java

88

## Le polymorphisme

Exemple :

```
class Point
{
    public Point(int x, int y) {...}
    public void affiche () {...}
}

class Pointcol extends Point
{
    public Pointcol(int x, int y, byte couleur) {...}
    public void affiche () {...}
}
```

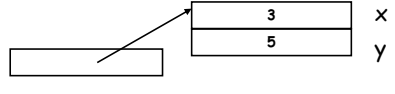
5/12/2024

Cours et Exercices Java

89


## Le polymorphisme

Point p;  
p = new Point (3,5);



(Point p)

p = new Pointcol (4,8, (byte) 2);  
//p de type Point contient  
//la référence a un objet de type Pointcol



(Point p)

⇒ Conversion implicite (légale) d'une référence à un type classe Pointcol en une référence à un type ascendant de Point.

5/12/2024

Cours et Exercices Java

90

## Le polymorphisme

```
Point p;  
p = new Pointcol (4, 8, 2);  
p.affiche(); //appelle la méthode affiche de la classe  
             Pointcol
```

L'instruction `p.affiche()`; appelle la méthode `affiche` de la classe `Pointcol`.

⇒ L'appel se fonde sur le type effectif de l'objet référencé par `p` au moment de l'appel et non sur le type de la variable `p`.

⇒ Le polymorphisme permet d'obtenir un comportement adapté à chaque type d'objet. Sans avoir besoin de tester sa nature.

## Cast (transtypage): conversions de classes

- Le « cast » est le fait de forcer le compilateur à considérer un objet comme étant d'un type qui n'est pas le type déclaré ou réel de l'objet
- En Java, les seuls casts autorisés entre classes sont les casts entre classe de base et classes dérivées

## Cast (transtypage): conversions de classes

### Syntaxe

- Pour caster un objet *o* en classe *A* :  
 (A) *o*;
- Exemple :  
 Point *a* = new Point();  
 PointCol *r* = (PointCol) *a*;  
 Point *b*;  
 (PointCol) *b*=*r*;

## UpCast : classe de base → classe dérivée

- *Upcast* : un objet est considéré comme une instance d'une des classes de base de sa classe réelle.
- Il est toujours possible de faire un *upcast* car tout objet peut être considéré comme une instance d'une classe de base.
- Le *upcast* est souvent implicite.

Cours et Exercices Java

### DownCast : classe dérivée → classe de base

- *Downcast* : un objet de la classe dérivée est considéré comme étant d'une classe de base de sa classe de déclaration
- Toujours accepté par le compilateur
- Mais peut provoquer une erreur à l'exécution ; à l'exécution il sera vérifié que l'objet est bien de la classe dérivée
- Un *downcast* doit toujours être explicite

Cours et Exercices Java

### Le mot clé super

- Le mot clé **super** permet d'accéder à une méthode d'une classe de base, alors que cette dernière était redéfinie dans la classe dérivée.

5/12/2024

Cours et Exercices Java

96



### Le mot clé super

- `super()` permet d'appeler le constructeur de la classe mère.
- C'est la première chose à faire dans la construction d'une sous-classe.
- Appeler le constructeur de la classe mère garantit que l'on peut initialiser les arguments de la classe mère.
- On passe les paramètres nécessaires
- Si l'on n'indique pas `super()`, il y a un appel du constructeur sans paramètres de la classe mère.
- Le mot clé `super` permet d'accéder à une méthode d'une classe de base, alors que cette dernière était redéfinie dans la classe dérivée.

5/12/2024

Cours et Exercices Java

97

### Le mot clé super

#### Exemple :

```
class A{
    void f() { System.out.println(" appel f de A ") ; }
}
class B extends A{
    void f() { System.out.println(" appel f de B ") ; }
    public void test() {
        A a=new A();
        a.f();
        super.f();
        this.f(); }
}
public class Super{
    public static void main(String args[])
    B b= new B();
    b.test();
}
}
```

5/12/2024

Cours et Exercices Java

98

### Le mot clé super

#### Un autre Exemple :

```
class Mot {
    Mot(String chaine) { ... }
    Mot renvoyerPluriel();
}

class Verbe extends Mot {
    int groupe;
    boolean transitif;

    Verbe(String chaine, int gr, boolean trans)
    {
        super(chaine);
        setGroupe(gr);
        setTransitif(trans);
    }
}
```

5/12/2024

Cours et Exercices Java

99

### La super - classe Object

En java toute classe simple dérive de la classe Object.

Lorsqu'on définit une classe point

```
class Point
{.....}
```

Tout se passe comme si :

```
class Point extends Object
{.....}
```

5/12/2024

Cours et Exercices Java

100

### La super - classe Object

*Utilisation d'une référence de type Objet:*

```
Point p=new Point(.....);
Pointcol pc=new Pointcol(..... );
Fleur f =new Fleur(....);
Objet o;
.....
o=p;    //ok
o=pc;   //ok
o=f;    //ok
```

5/12/2024

Cours et Exercices Java

101

### Les méthodes de la classe Object

*Tout objet à accès aux méthodes publiques définies dans la classe Object. Certaines des méthodes de cette classe sont très intéressantes et méritent notre attention.*

*il s'agit simplement d'une surcharge des méthodes se trouvant dans la classe Object.*

*La classe Object rassemble donc des méthodes que tout objet aura étant donné que toutes les classes héritent directement ou indirectement de la classe Object. Cette classe possède des méthodes très générales.*

5/12/2024

Cours et Exercices Java

102

## Les méthodes de la classe Object

### 1. public String toString();

- Cette méthode permet d'obtenir une représentation de l'objet sous forme d'une chaîne de caractère.
- Cette méthode fournit une chaîne contenant :  
Le nom de la classe concernée  
L'adresse de l'objet en Héxadécimal (précédée de @).
- `toString()` peut être redéfinie par exemple dans la classe `Point` pour fournir une chaîne contenant les coordonnées de `Point`.

5/12/2024

Cours et Exercices Java

103

## Les méthodes de la classe Object

### Exemple :

```
class Point
{ public point(int abs, int ord)
  { x=abs;y=ord;}
  private int x,y;
}
public class ToString1{
  public static void main(String args[])
  { point a = new point (1,2);
    point a = new point (5,6);
    System.out.println("a="+ a.toString());
    System.out.println("a="+ a.toString());
  }
}
```

### Résultat :

```
a= Point@fc17aedf
b= Point@fc1baedf
```

5/12/2024

Cours et Exercices Java

104

### Les méthodes de la classe Object

public boolean equals (Object o);

Cette méthode permet de comparer l'adresse d'un objet avec l'adresse de l'objet courant pour savoir s'ils sont égaux ou non.

```
Object o1= new Point(1;2);
Object o2= new Point(1;2);
```

`o1.equals(o2)` a pour valeur `false`

On peut surdéfinir cette méthode dans n'importe quelle classe.

5/12/2024

Cours et Exercices Java

105

### Les méthodes de la classe Object

protected Object clone();

Si `x` désigne un objet `obj1`, l'exécution de `x.clone()` renvoie un second objet `obj2`, qui est une copie de `obj1` si `obj1` est ensuite modifié, `obj2` n'est pas affecté par ce changement.

Pour permettre le clonage d'une classe, il faut implémenter dans la classe l'interface `Cloneable`.

La première chose que fait la méthode `clone()` de la classe `Object`, quand elle est appelée, est de tester si la classe implémente `Cloneable`. Si ce n'est pas le cas, elle lève l'exception `CloneNotSupportedException`.

5/12/2024

Cours et Exercices Java

106

### Les classe abstraites

- Une classe abstraite est une classe qui ne permet pas d'instancier des objets.
- Elle ne peut servir que de classe de base pour une dérivation.

Elle se déclare ainsi:

```
abstract class A
{.....
}
```

5/12/2024

Cours et Exercices Java

107

### Les classe abstraites

On peut trouver dans une classe abstraite, des méthodes et des champs, dont héritera toute classe dérivée.

On peut aussi trouver des méthodes dites abstraites

abstract class A

```
{ public void f(){.....} //f défini dans A
```

```
    public abstract void g (int n)
    {
        // g n'est pas défini dans A on fournit juste son en tête
    }
}
```

5/12/2024

Cours et Exercices Java

108

### Les classes abstraites

`A a;     // ok`

`A=new A(.....); //interdit`

*On doit définir une classe B dérivée de A et définir la méthode abstraite g.*

```
Class B extends A()
{ public void g(int n){.....}
}
```

`A a=new B(.....); // ok`

5/12/2024

Cours et Exercices Java

109

### Intérêt des classes abstraites

*On peut placer dans une classe abstraite toutes les fonctionnalités dont on souhaite disposer pour toute ses descendantes.*

Soit:

- sous forme d'implémentation complète de méthodes (non abstraites) et de champs lorsqu'ils sont communs à toutes ses descendantes.
- Soit sous forme d'interface de méthodes abstraites dont on est alors sûr qu'elles existeront dans toute classe dérivée instanciable.

5/12/2024

Cours et Exercices Java

110

## Les interfaces

*Si on considère une classe abstraite n'implémentant aucune méthode et aucun champ, on aboutit à la notion d'interface.*

*Une interface : définit les entêtes d'un certain nombre de méthodes, ainsi que des constantes.*

- ⇒ Une classe pourra implémenter plusieurs interfaces (alors qu'une classe ne pouvait dériver que d'une seule classe abstraite).
- ⇒ La notion d'interface se superpose à la notion de dérivation et non s'y substitue.
- ⇒ Les interfaces pouvant se dériver.
- ⇒ On pourra utiliser des variables de type interface.

5/12/2024

Cours et Exercices Java

111

## Mise en œuvre d'une interface

⇒ Définition d'une interface:

*On définit une interface comme une classe:*

```
public interface I
{
    void f(int n);
    void g();
}
```

5/12/2024

Cours et Exercices Java

112



## Mise en œuvre d'une interface

### ⇒ Implémentation d'une interface

Dans la définition d'une classe on peut préciser qu'elle implémente une interface donnée en utilisant le mot clé **implements**.

```
class A implements I
{
// A doit re(définir) les méthodes f et g prévues dans
// l'interface I
}
```

5/12/2024

Cours et Exercices Java

113

## Le modificateur de visibilité protected

Donner une visibilité publique à une variable d'instance viole le principe d'encapsulation qui dit que le monde extérieur ne devrait pas avoir accès directement à l'état d'un objet.

Il existe donc un autre modificateur de visibilité représenté par le mot réservé **protected** qui joue le même rôle que **private** mais qui autorise l'héritage.

Un membre déclaré comme **protected** ne sera donc visible par personne du monde extérieur sauf par les classes dérivées.

5/12/2024

Cours et Exercices Java

114

### Le modificateur de visibilité protected

- Un membre déclaré comme **protected** est moins fort qu'un membre privé, par ailleurs, les membres **protected** sont de plus visibles dans toutes les autres classes du même package.
- Lorsque l'on doit choisir le modificateur pour un membre, il est donc préférable de commencer par **private**. On passe ensuite successivement à **protected** puis à **public** en cas de soucis.

5/12/2024

Cours et Exercices Java

115

### Le modificateur de visibilité protected

La classe `Book` est donnée dans le listing suivant:

```
public class Book
{
    protected int nbPages = 2000;
    public void printPageNumber()
    {
        System.out.println ("Le livre compte " + nbPages + "
pages.");
    }
}
```

Question : Ecrire une classe `Dictionnary` qui hérite de la classe `Book`

5/12/2024

Cours et Exercices Java

116

### Le modificateur de visibilité protected

```
public class Dictionary extends Book
{
    private int nbDefs = 15000;
    public void printStats()
    {
        System.out.print("Le dictionnaire compte
" + nbDefs + " définitions, ");
        System.out.println("soit en moyenne" +
(nbDefs / nbPages) + " définitions par page");
    }
}
```

5/12/2024

Cours et Exercices Java

117

### En Résumé : Spécificateurs d'accès:

Il existe 3 mots clés pour spécifier l'accès au sein d'une classe: public, private et protected.

- **public**: veut dire tout le monde
- **private**: veut dire que personne ne peut accéder à ces définitions à part les méthodes interne de ce type.
- **protected**: se comporte comme **private** avec moins de restriction. Une classe dérivée a un accès aux membres protected mais pas aux membres private
- Accès par défaut, lorsqu'aucun de ces spécificateurs n'est mentionné. Ca correspond à l'accès au paquetage

Cours et Exercices Java

Exemple:

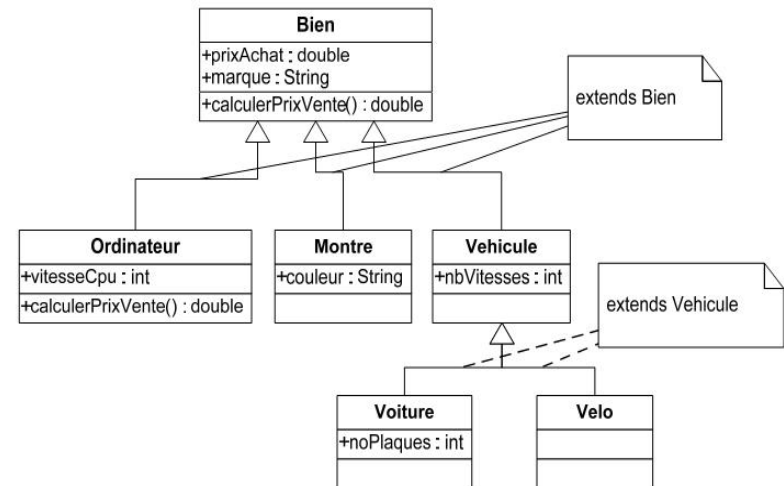
```

class A {
    public int x;
    private int y;
    public void initialise (int i, int j){
        x=i;
        y=j;
    }
}

public class TestA{ // fichier source de nom TestA.java
    public static void main (String args[]) {
        A a;
        a=new A(); // On peut aussi déclarer A a=new A();
        a.initialise(1,3); // x vaut 1 et y vaut 3
        a.x=2; // x vaut maintenant 2. On peut accéder à x car il est
        public
        a.y=3; // ne compile pas car y est privée
    }
}

```

Cours et Exercices Java

L'opérateur instanceof

### L'opérateur instanceof

- L'opérateur logique instanceof permet de tester le type d'un objet:

```
Velo vel = new Velo(...);
```

```
boolean b;
```

```
b = (vel instanceof Velo); // true
```

```
b = (vel instanceof Vehicule); // true
```

```
b = (vel instanceof Bien); // true
```

```
b = (vel instanceof Voiture); // false
```

- Il est donc permis d'affecter un vélo à une variable de type

Vehicule ou Bien:

```
Velo vel = new Velo(...);
```

```
Vehicule veh = vel; // OK
```

```
Bien b = vel; // OK
```

```
Voiture voit = vel; // Erreur
```

## Chapitre 3 : Les tableaux

## Plan chapitre 3 : les tableaux

- Déclaration et création de tableaux
- Utilisation d'un tableau
- Tableau en argument ou en retour
- Les tableaux à plusieurs indices

5/12/2024

Cours et Exercices Java

123

## Les tableaux

### Déclaration et création de tableaux

```
int t[ ] ;
```

//t est destiné à contenir la référence à un tableau d'entiers. pas de dimension et pas de valeur.

```
t= new int[5] ;
```

//allouer l'emplacement nécessaire à un tableau de 5 éléments. Par défauts les éléments sont initialisé à une valeur nulle ( 0 pour int).

5/12/2024

Cours et Exercices Java

124

## Les tableaux

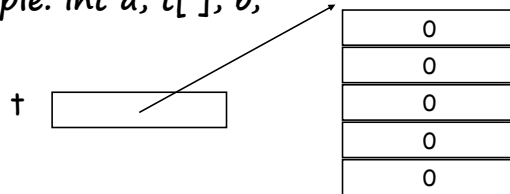
La déclaration d'une référence à un tableau précise le type des éléments du tableau;

`int t[ ];` ou

`int [ ] t1,t2;`

La première forme permet de mélanger les tableaux de type `int` et les variables de type `int`.

Exemple: `int a, t[ ], b;`



5/12/2024

Cours et Exercices Java

125

## Les tableaux

Remarque :

Une déclaration de tableau ne doit pas préciser de dimensions

`int t[5];` // est une erreur

5/12/2024

Cours et Exercices Java

126

## Les tableaux

Création d'un tableau par l'opérateur new

```
int n=Clavier.lireInt( );
```

```
int t[ ]=new int [n];
```

Après exécution *la taille de l'objet tableau ne change pas*, mais la référence t peut changer

Création d'un tableau en utilisant un initialiseur

```
int n,p;
```

```
int t[ ]={1,n,n+p,2*p,12}
```

On crée un tableau de 5 entiers ayant les valeurs des expressions mentionnées et on place la référence dans t.

5/12/2024

Cours et Exercices Java

127

## Les tableaux

Utilisation d'un tableau

- Accès individuel à chacun de ses éléments
- Accès global à l'ensemble du tableau

5/12/2024

Cours et Exercices Java

128



## Les tableaux

### Accès individuel à chacun de ses éléments

- `int t [ ]=new int [5];`
- `t [0 ]=15;`
- `t [2 ]++;`
- `System.out.println(t [4 ]);`

5/12/2024

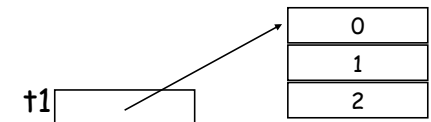
Cours et Exercices Java

129

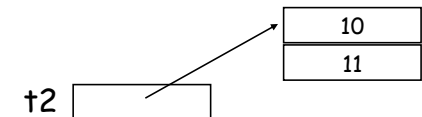
## Les tableaux

### Accès global à l'ensemble du tableau

```
int [ ] t1=new int [3];
for( int i=0 ; i<3 ; i++)
t1[ i ] = i;
```



```
int [ ] t2=new int [2];
for( int i=0; i<2 ; i++)
t2[ i ] = i+10;
```



5/12/2024

Cours et Exercices Java

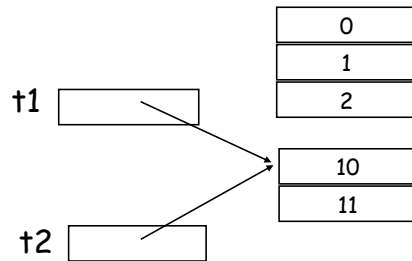
130

## Les tableaux

Accès global à l'ensemble du tableau

`t1 = t2;`

Après exécution la situation devient:



5/12/2024

Cours et Exercices Java

131

## Les tableaux

La taille d'un tableau : length

- `int t [ ] = new int [5];`
- `System.out.println(t.length);`  
// affiche 5
- `int t [ ] = new int [3];`
- `System.out.println(t.length);`  
// affiche 3

5/12/2024

Cours et Exercices Java

132

Exemple de tableau d'Objets

```

Point [] tp ;
//Déclaration d'une référence à un tableau de type Point.

tp = new Point[3] ;
//Création d'un tableau de taille 3 dont les éléments sont de
// type Point

tp[0] = new Point (1, 2) ;
//Création d'un objet de type Point de valeur (1,2) dont la
// référence est placée dans tp[0].

tp[1] = new Point (4, 5) ;

tp[2] = new Point (8, 9) ;

```

5/12/2024

Cours et Exercices Java

133

Exercice : qu'affiche le programme suivant?

```

public class TabPoint
{ public static void main (String args[])
  { Point [] tp ;
    tp = new Point[3] ;
    tp[0] = new Point (1, 2) ;
    tp[1] = new Point (4, 5) ;
    tp[2] = new Point (8, 9) ;
    for (int i=0 ; i<tp.length ; i++)
      tp[i].affiche() ;
  } }
class Point
{ public Point(int x, int y)
  { this.x = x ; this.y = y ;
  }
  public void affiche ()
  { System.out.println ("Point : " + x + " , " + y) ;
  }
  private int x, y ; }

```

5/12/2024

Cours et Exercices Java

134

## Les tableaux

### Cas particulier des tableaux de de caractères

```
char [ ] tc ;
int [ ] ti ;
.....
system.out.println(tc) ;
// on obtient bien les valeurs des caractères de
  tc

system.out.println(ti) ;
// on n'obtient pas les valeurs de entiers de ti
```

5/12/2024

Cours et Exercices Java

135

## Les tableaux

### Tableaux transmis en argument

- Lorsqu'on transmet un Tableaux en argument d'une méthode, on transmet en fait *une copie de la référence au tableau*. La méthode agit directement sur le tableau concerné.

### Exercice:

Ecrire une classe avec deux méthodes statiques permettant:

1. D'afficher les valeurs des éléments d'un tableau d'entiers.
2. De mettre à zéro les éléments d'un tableau d'entier.

5/12/2024

Cours et Exercices Java

136

```

public class TabArg
{ public static void main (String args[])
  { int t[] = { 1, 3, 5, 7 };
    System.out.print ("t avant : ") ;
    Util.affiche (t) ;
    Util.raz (t) ;
    System.out.print ("\nt apres : ") ;
    Util.affiche (t) ;
  }
}
class Util
{ static void raz (int t[])
  { for (int i=0 ; i<t.length ; i++)
    t[i] = 0 ;
  }
  static void affiche (int t[])
  { for (int i=0 ; i<t.length ; i++)
    System.out.print (t[i] + " ") ;
  }
}

```

### Solution

### Résultats d'exécution

- *t avant : 1 3 5 7*
- *t après : 0 0 0 0*

## Les tableaux

### Tableaux transmis en retour

Même réflexion s'applique aux tableaux fournis en valeur de retour.

Exemple:

```
public static int[ ] suite (int n)
{   int [ ] res = new int [n];
    for(int i=0; i<n ; i++)
        res[i]=i+1;
    return res;
}
```

5/12/2024

Cours et Exercices Java

139

## Les tableaux

### Les tableaux à plusieurs indices

Java offre la notion de tableaux de tableaux, qui est plus riche que celle de tableaux à plusieurs indices.

5/12/2024

Cours et Exercices Java

140

## Les tableaux

Déclaration:

```
int t [][];   int [] t [];  int [] [] t;
```

*t est une référence à un tableau, dans lequel chaque élément est lui-même une référence à un tableau d'entiers.*

```
t = { new int[3] , new int[2] } ;
```

*L'évaluation de cette instruction crée un tableau de 3 entiers et un tableau de 2 entiers*

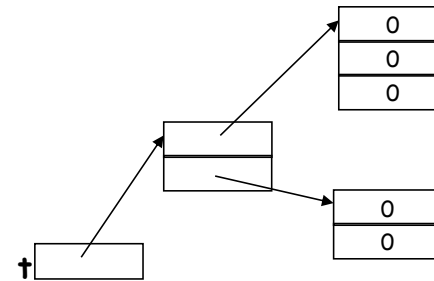
5/12/2024

Cours et Exercices Java

141

## Les tableaux

*On aura la situation suivante:*



*t[0] : référence au premier tableau de 3 entiers*

*t[0][1] : le deuxième élément du premier tableau*

*t[1] : référence au deuxième tableau de 2 entiers*

*t[1][0] : le premier élément du deuxième tableau*

5/12/2024

Cours et Exercices Java

142

## Les tableaux

### Exercice :

*Ecrire une classe Util qui utilise deux méthodes statiques permettant:*

- 1. Afficher les valeurs des éléments d'un tableau d'entiers à deux indices, ligne par ligne (méthode affiche).*
- 2. Mettre à zéro les éléments d'un tableau d'entiers à deux indices (méthode raz).*

5/12/2024

Cours et Exercices Java

143

# Chapitre 4 : Les chaînes de caractères

5/12/2024

Cours et Exercices Java

144



## Plan chapitre 4 : les chaînes de caractères

- Fonctionnalités de base de la classe String
- Recherche dans une chaîne
- Comparaisons de chaînes
- Modifications de chaînes
- Tableaux de chaînes
- Conversions entre chaînes et types primitifs

5/12/2024

Cours et Exercices Java

145

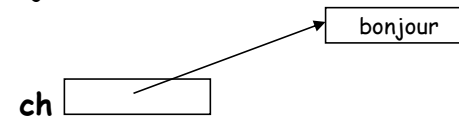
## Les chaînes de caractères

### Fonctionnalités de base de la classe String

**String ch;**

Déclare que ch est une référence à un objet de type String

**ch = "bonjour " ;**



5/12/2024

Cours et Exercices Java

146

## Les chaînes de caractères

### Fonctionnalités de base de la classe String

```
String ch1= new String ( );  
//ch1 contient la référence à une chaîne vide
```

```
String ch2= new String ("hello");  
//ch2 contient la référence à une chaîne contenant la  
suite "hello".
```

```
String ch3= new String (ch2);  
//ch3 contient la référence à une chaîne contenant la  
suite "hello".
```

5/12/2024

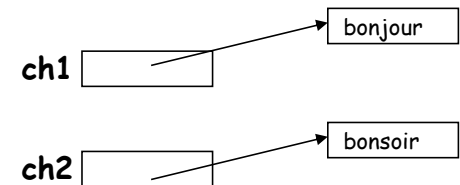
Cours et Exercices Java

147

## Les chaînes de caractères

Un objet de type String n'est pas modifiable  
Il n'existera aucune méthode permettant de  
modifier la valeur d'un objet de type String.

Exemple :  
String ch1,ch2, ch;  
ch1="bonjour";  
ch2="bonsoir";



5/12/2024

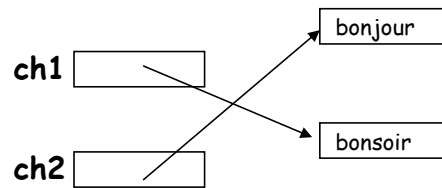
Cours et Exercices Java

148

## Les chaînes de caractères

Soit les instructions suivantes:

```
ch=ch1;
ch1=ch2;
ch2=ch;
```



### Conclusion:

Les deux objets de type chaîne n'ont pas été modifiés, mais les références `ch1` et `ch2` l'ont été.

5/12/2024

Cours et Exercices Java

149

## Les chaînes de caractères

Affichage d'une chaîne à l'écran

```
System.out.println("bonjour");
// affiche une constante chaîne équivalente à
String ch;
```

```
....
System.out.println(ch);
```

Lecture d'une chaîne au clavier.

```
String ch;
```

```
....
ch = Clavier.lireString();//crée un objet de type
String contenant la chaîne lue au clavier.
```

5/12/2024

Cours et Exercices Java

150

## Les chaînes de caractères

### Longueur d'une chaîne

La méthode `length()` permet d'obtenir la longueur d'une chaîne = le nombre de caractères qu'elle contient.

#### Exemple:

```
String ch="bonjour";
int n=ch.length( ); // n contient 7
ch="hello";          // n contient 5
ch="";               // n contient 0
```

5/12/2024

Cours et Exercices Java

151

## Les chaînes de caractères

### Accès aux caractères d'une chaîne : `charAt`

La méthode `charAt( )` de la classe `String` permet d'accéder à un caractère de rang donné d'une chaîne (le premier caractère porte le rang 0).

#### Exemple :

```
String ch = "bonjour ";
```

```
ch.charAt(0)    correspond au caractère b.
ch.charAt(2)    correspond au caractère n.
```

5/12/2024

Cours et Exercices Java

152

## Les chaînes de caractères

### Exercice :

Écrire une classe `MotCol` qui permet de lire une chaîne de caractères au clavier et l'affiche verticalement.

### Exemple:

Donner un mot : bonjour

Voici votre mot en colonne:

b  
o  
n  
j  
o  
u  
r

5/12/2024

Cours et Exercices Java

153

## Les chaînes de caractères

### Concaténation de chaînes

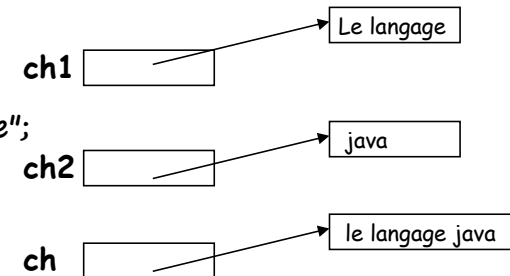
L'opérateur `+` est défini lorsque ses deux opérandes sont des chaînes. Il fournit en résultat une nouvelle chaîne formée de la concaténation des deux autres.

### Exemple :

`String ch1= "le langage";`

`String ch2= "java";`

`String ch=ch1+ch2;`



5/12/2024

Cours et Exercices Java

154

### Les chaînes de caractères

#### Conversions des opérandes de l'opérateur +

java autorise le mélange de chaînes et expressions d'un type primitif.

```
int n=26;
```

```
System.out.println(" n= " + n);
```

```
//affiche n=26;
```

#### L'opérateur +=

```
String ch= " bonjour " ;
```

```
ch+= " monsieur " ;
```

```
// affiche bonjour monsieur
```

5/12/2024

Cours et Exercices Java

155

### Les chaînes de caractères

#### Ecriture des constantes chaînes

On peut utiliser les caractères \n et \t

Le code Unicode du caractère, exprimé en hexadécimal sous la forme \uxxxx

xxxx représente 4 chiffres hexadécimaux

5/12/2024

Cours et Exercices Java

156

## Les chaînes de caractères

### Recherche dans une chaîne

La méthode `indexOf( )` permet de donner:

- La position du caractère
- La valeur `-1` sinon

### Exercice

Écrire un programme complet qui permet d'utiliser la méthode `indexOf( )` pour compter le nombre de caractères « e » présents dans un mot entré au clavier

5/12/2024

Cours et Exercices Java

157

## Les chaînes de caractères

### Comparaisons de chaînes

1. Les opérateurs `==` et `!=`

### Exemple

```
String ch1="bonjour";
String ch2="bonjour";
```

```
if ( ch1 == ch2 )
System.out.println("égales");
else
System.out.println("non égales");
```

5/12/2024

Cours et Exercices Java

158

## Les chaînes de caractères

### Comparaisons de chaînes

#### 2. La méthode equals( ) de la classe String

Permet de comparer 2 chaînes.

```
String ch1= "hello " ;
```

```
String ch2="bonjour";
```

```
ch1.equals(ch2); //expression fausse
```

```
ch1.equals("hello " ); //expression vrai
```

La méthode `equalsIgnoreCase( )` effectue la même comparaison, mais sans distinguer les majuscules des minuscules.

5/12/2024

Cours et Exercices Java

159

## Les chaînes de caractères

### Comparaisons de chaînes

#### 3. La méthode compareTo( )

Permet d'effectuer des comparaisons lexicographiques de chaînes pour savoir laquelle des deux chaînes apparaît avant une autre en se fondant sur l'ordre des caractères.

Utilisation: `ch1.compareTo(ch2);`

Elle fournit:

- Un entier négatif si `ch1` arrive avant `ch2`;
- Un entier nul si `ch1` et `ch2` sont égales;
- Un entier positif si `ch1` arrive après `ch2`;

5/12/2024

Cours et Exercices Java

160



## Les chaînes de caractères

### Modifications de chaînes

#### 1. Remplacement de caractères

`replace()` crée une chaîne en remplaçant toutes les occurrences d'un caractère donné par un autre.

```
String ch="bonjour";
String ch1=ch.replace('o','a');
```

// ch n'est pas modifié, ch1 contient banjaour

5/12/2024

Cours et Exercices Java

161

## Les chaînes de caractères

### Modifications de chaînes

#### 2. Extraction de sous chaîne

`substring()` permet de créer une nouvelle chaîne en extrayant de la chaîne courante.

`String substring` (int beginIndex, int endIndex)

`String substring` (int beginIndex)

#### Exemple :

```
System.out.println("abc");
String cde = "cde";
System.out.println("abc" + cde);
String c = "abc".substring(2,3);
String d = cde.substring(1, 2);
```

5/12/2024

Cours et Exercices Java

162

## Les chaînes de caractères

### Modifications de chaînes

#### 3. Passage en majuscule ou en minuscule

`toLowerCase()` crée une nouvelle chaîne en remplaçant toutes les majuscules par leur équivalent en minuscules

`toUpperCase()` crée une nouvelle chaîne en remplaçant toutes les minuscules par leur équivalent en majuscules

```
String ch= "LaNGaGe_3 ";
String ch1=ch.toLowerCase();
//ch non modifié,ch1 contient "langage_3 "
String ch2=ch.toUpperCase();
//ch non modifié,ch2 contient "LANGAGE_3 "
```

5/12/2024

Cours et Exercices Java

163

## Les chaînes de caractères

### Modifications de chaînes

#### 4. Suppression des séparateurs de début et de fin

`trim()` crée une nouvelle chaîne en supprimant les éventuels séparateurs de début et de fin (espace, tabulations, fin de ligne)

```
String ch= " \n\t séparateurs avant, pendant \t\n,
et après\n\t " );
```

```
String ch1=ch.trim();
//ch1 contient "séparateurs avant, pendant \t\n,
et après ";
```

5/12/2024

Cours et Exercices Java

164

## Les chaînes de caractères

### Tableaux de chaînes

#### Exercice

Ecrire une classe *TriCh* qui effectue un tri lexicographique des chaînes contenues dans un tableau

NB: utiliser le tableau suivant :

```
String tabCh[ ]=(" java ", " c ", " pascal ", " c++ ", "
ada ", " basic ", " fortran ");
```

5/12/2024

Cours et Exercices Java

165

## Les chaînes de caractères

### 1. Conversion d'un type primitif en une chaîne

```
int n=427;
```

```
String ch=String.valueOf(n);
```

//fournit une chaîne obtenue par formatage de la valeur contenue dans n, soit ici "427 "

#### Exercice :

Ecrire un programme *ConvCh* qui permet de lire des entiers au clavier et les convertir en chaîne(on interrompt lorsque utilisateur fournit un entier nul)

5/12/2024

Cours et Exercices Java

166

## Les chaînes de caractères

### 2. Conversion d' une chaîne en un type primitif

Pour convertir une chaîne en un entier de type int, on utilisera la méthode statique `parseInt` de la classe `Integer`, comme ceci:

```
String ch= " 3587";
int n=Integer.parseInt(ch);
```

D'une manière générale on dispose de :

```
Byte.parseByte,
Short.parseShort,
Integer.parseInt,
Long.parseLong,
Float.parseFloat,
Double.parseDouble,
```

5/12/2024

Cours et Exercices Java

167

## Les chaînes de caractères

### Conversions entre chaînes et tableaux de caractères

On peut construire une chaîne à partir d'un tableau

```
char mot [ ] =( 'b','o','n','j','o','u','r');
//tableau de 7 caractères
String ch=new String (mot);
Ch contient la chaîne : "bonjour" ;
```

5/12/2024

Cours et Exercices Java

168

## Les chaînes de caractères

### Les arguments de la ligne de commande

L'entête de la méthode main se présentait comme suit:

```
public static void main(string args[ ] )
```

La méthode reçoit un argument du type tableau de chaînes destiné à contenir les éventuels arguments fournis au programme lors de son lancement.

5/12/2024

Cours et Exercices Java

169

## Les chaînes de caractères

### Les arguments de la ligne de commande

Exemple de programme récupérant les arguments de la ligne de commande

```
public class ArgMain
{ public static void main(string args[])

{int nbArgs=args.length;

if (nbArgs==0) system.out.println(" pas d'arguments");

{ for (int i=0 ; i< nbArgs ; i++)
system.out.println("argument n:" + i+1 + "=" + args[i]);
}
}
}
```

5/12/2024

Cours et Exercices Java

170

### Les chaînes de caractères

#### Site Internet:

Sur le site de SUN suivant vous trouverez un inventaire des méthodes définies dans la classe String .

<http://java.sun.com/javase/6/docs/api/java/lang/String.html>

- *Test de connaissance*

1. Donner la définition de l'héritage.
2. Quelles sont les règles de l'héritage?
3. Donner un exemple concrétisant cette notion.
4. Définir le polymorphisme.
5. Donner un exemple.
6. Définir une interface.
7. Donner un exemple complet d'une classe utilisant une interface.
8. Donner un exemple d'un tableau d'objets

## Les packages

### **Le package java.lang**

- Ce package de base contient les classes fondamentales tel que *Object*, *Class*, *Math*, *System*, *String*, *StringBuffer*, *Thread*, les wrappers etc ...
- Il contient également plusieurs classes qui permettent de demander des actions au système d'exploitation sur laquelle la machine virtuelle tourne, par exemple les classes *ClassLoader*, *Runtime*, *SecurityManager*.
- Ce package est tellement fondamental qu'il est implicitement importé dans tous les fichiers sources par le compilateur.

### **La présentation rapide du package awt java**

- AWT est une collection de classes pour la réalisation d'applications graphiques ou GUI (Graphic User Interface)
- AWT se compose de plusieurs packages dont les principaux sont:
  - *java.awt* : c'est le package de base de la bibliothèque AWT
  - *java.awt.images* : ce package permet la gestion des images
  - *java.awt.event* : ce package permet la gestion des événements utilisateurs
  - *java.awt.font* : ce package permet d'utiliser les polices de caractères
  - *java.awt.dnd* : ce package permet l'utilisation du cliquer/glisser



# Chapitre 5 : La gestion des exceptions

5/12/2024

Cours et Exercices Java

177

## Plan chapitre 5 : La gestion des exceptions

*Définition**Levée d' exception**Traitement d'exception**Classes et sous-classes d'exception**Ordre des blocs catch**Sous-classes et clause throws**Relancer une exception**Catégorie d'objet lancé**Classes dérivées de Throwable**Créer une classe d'exception**Ordre des blocs catch*

### Définition

- Une **exception** est un signal qui se déclenche en cas de problème. Les exceptions permettent de gérer les cas d'erreur et de rétablir une situation stable (ce qui veut dire, dans certains cas, quitter l'application proprement). La gestion des exceptions se décompose en deux phases :
  - La levée d'exceptions
  - Le traitement d'exceptions.
- En Java, une exception est représentée par une classe. Toutes les exceptions dérivent de la classe *Exception* qui dérive de la classe *Throwable*.

### Levée d' exception

- Une exception est levée grâce à l'instruction *throw* :

```
if (k<0)
  throw new EstNegatifException("Message");
```

Une exception peut être traitée directement par la méthode dans laquelle elle est levée, mais elle peut également être envoyée à la méthode appelante grâce à l'instruction *throws* (à ne pas confondre avec *throw*)

### Levée d' exception

```
public void maMethode(int entier) throws
    IOException {
//code de la méthode
}
```

Dans cet exemple, si une exception de type *IOException* est levée durant l'exécution de *maMethode*, l'exception sera envoyée à la méthode appelant *maMethode*, qui devra la traiter.

### Traitement d'exception

- Le traitement des exceptions se fait à l'aide de la séquence d'instructions *try...catch...finally*.
  - L'instruction *try* indique une instruction (ou plus un bloc d'instructions) susceptible de lever des exceptions.
  - L'instruction *catch* indique le traitement pour un type particulier d'exceptions.
  - L'instruction *finally*, qui est optionnelle, sert à définir un bloc de code à exécuter dans tous les cas, exception est levée ou non.

**Exemple :**

```

public String lire(String nomDeFichier) throws IOException {
    try {
        // cette ligne est susceptible de lever une exception de type FileNotFoundException
        FileReader lecteur = new FileReader(nomDeFichier);
        char[] buf = new char[100];
        // Cette ligne est susceptible de lever une exception de type IOException
        lecteur.read(buf,0,100);
        return new String(buf);
    }
    catch (FileNotFoundException fnfe) {
        fnfe.printStackTrace();
        // Indique l'exception sur le flux d'erreur standard
    }
    finally {
        System.err.println("Fin de méthode");
    }
}

```

**Suite exemple**

- Le bloc **catch**  
(FileNotFoundException fnfe)  
capture toute exception du type  
FileNotFoundException (cette classe  
dérive de la classe IOException).
- Le bloc **finally** est exécuté quelque  
soit ce qui se passe (exception ou  
non).

### Suite exemple

- Toute autre exception non capturée (telle `IOException`) est transmise à la méthode appelante, et doit toujours être déclarée pour la méthode, en utilisant le mot clé **throws**, sauf les exceptions dérivant de la classe `RuntimeException`.
- S'il n'y avait pas cette exception à la règle, il faudrait déclarer **throws** `ArrayIndexOutOfBoundsException` chaque fois qu'une méthode utilise un tableau, ou **throws** `ArithmeticException` chaque fois qu'une expression est utilisée, par exemple.

### Classes et sous-classes d'exception

- L'héritage entre les classes d'exceptions peut conduire à des erreurs de programmation. En effet, une instance d'une sous-classe est également considérée comme une instance de la classe de base.

### Ordre des blocs catch

- L'ordre des blocs catch est important : il faut placer les sous-classes avant leur classe de base. Dans le cas contraire le compilateur génère l'erreur `exception classe_exception has already been caught`.
- Exemple d'ordre incorrect :

```
try{
  FileReader lecteur = new FileReader(nomDeFichier);
}
catch(IOException ioex) // capture IOException et ses sous-classes{
  System.err.println("IOException caught :");
  ioex.printStackTrace();
}

catch(FileNotFoundException fnfex)
// <-- erreur ici // FileNotFoundException déjà capturé par
  catch(IOException ioex)
{
  System.err.println("FileNotFoundException caught :");
  fnfex.printStackTrace();
}
```

### Ordre des blocs catch

- L'ordre correct est le suivant :

```
try{
  FileReader lecteur = new FileReader(nomDeFichier);
}

catch(FileNotFoundException fnfex) {
  System.err.println("FileNotFoundException caught :");
  fnfex.printStackTrace();
}

catch(IOException ioex)
// capture IOException et ses autres sous-classes{
  System.err.println("IOException caught :");
  ioex.printStackTrace();
}
```

### Sous-classes et clause throws

- Une autre source de problèmes avec les sous-classes d'exception est la clause throws.
- Ce problème n'est pas détecté à la compilation.

### Exemple

```
public String lire(String nomDeFichier) throws
    FileNotFoundException {
```

```
    try {
```

```
        FileReader lecteur = new FileReader(nomDeFichier);
        char[] buf = new char[100];
        lecteur.read(buf,0,100);
        return new String(buf);
    }
```

```
    catch (IOException ioe) { // capture IOException et ses sous-
                               classes
        ioe.printStackTrace();
    }
}
```

Cette méthode ne lancera jamais d'exception de type `FileNotFoundException` car cette sous-classe de `IOException` est déjà capturée.

### Relancer une exception

- Relancer une exception consiste simplement à utiliser l'instruction **throw** avec l'objet exception que l'on a capturé.

### Relancer une exception

- Exemple:

```
public String lire(String nomDeFichier) throws IOException {
    try {
        FileReader lecteur = new FileReader(nomDeFichier);
        char[] buf = new char[100];
        lecteur.read(buf,0,100);
        return new String(buf);
    }

    catch (IOException ioe) // capture IOException et ses sous-classes {
        // ... traitement partiel de l'exception ...
        throw ioe; <-- relance l'exception
    }
}
```



### Catégorie d'objet lancé

- Ici on traite des exceptions, mais en fait tout objet dont la classe dérive de la classe **Throwable** peut être utilisé avec les mots-clés **throw**, **throws** et **catch**.

### Classes dérivées de Throwable

Il existe deux principales sous-classes de la classe **Throwable** :

- Exception signale une erreur dans l'application,
- Error signale une erreur plus grave, souvent au niveau de la machine virtuelle (manque de ressource, mauvais format de classe, ...).

### Créer une classe d'exception

- Il est également possible d'étendre une classe d'exception pour spécialiser un type d'erreur, ajouter une information dans l'objet exception, ...

### Exemple :

```
public class HttpException extends Exception {  
    private int code;  
    public HttpException(int code,String message) {  
        super(""+code+" "+message);  
        this.code=code;  
    }  
    public int getHttpCode() {  
        return code;  
    }  
}
```

**Exemple :**

Une instance de cette classe peut ensuite être lancée de la manière suivante :

```
public void download(URL url) throws HttpException {  
    ... throw new HttpException ( 404, "File not found" );  
}
```

et capturée comme suit :

```
try {  
    download( ... );  
}  
catch(HttpException http_ex) {  
    System.err.println("Erreur "+http_ex.getHttpCode());  
}
```