

Chapitre 4

Compléments
exemple

- Visibilité
- Utilisation de *this*
- Exemple : la classe Compte

Sous-classes

- Classes dérivées
- Classes abstraites
- Interfaces
- Exemples

Affectations, copies, passages de paramètres

Visibilité des membres d'une classe

- **public** : visible sans restriction
- **protected** : visible dans le paquetage et dans les classes dérivées
- **friendly** (par défaut) : visible dans le paquetage
- **private** : visible uniquement dans la classe

Il est conseillé de déclarer les attributs **private**, ou à la rigueur **protected**, sauf - si l'on a des raisons particulières de faire autrement
- pour définir des constantes

Pour que l'accès à un attribut *xxx* non visible, on peut définir les méthodes

- **getXxx()** permettant la consultation mais pas la modification
- **setXxx(valeurs)** permettant une modification, mais assurant une cohérence que la modification directe de l'attribut ne garantirait peut-être pas

- **Attention !** : inutile de définir plus de méthodes **get...** et **set...** que nécessaire

Visibilité des classes

public ou **friendly** (par défaut)

une seule classe publique par fichier qui doit avoir le même nom que la classe publique avec l'extension `.java`

Utilisation de *this*

this est utilisé pour désigner *cet* objet

Il est aussi utilisé, dans un constructeur, pour désigner un autre constructeur

Exemples :

si on a défini le constructeur

```
public Cla(Cla1 c1, Cla2 c2) { ... }
```

on peut définir un autre constructeur par

```
public Cla(Cla1 c) { this(c, valeurParDefaut); }
```

si on a défini le constructeur

```
public Cla(Cla1 c) { ... }
```

on peut définir un autre constructeur par

```
public Cla(Cla1 c1, Cla2 c2) { this(c1);... }
```

Attributs de classe

Contrairement aux attributs d'instances qui mémorisent les caractéristiques des objets d'une classe, les **attributs de classes** sont utilisés pour définir des constantes ou des valeurs par défaut ou l'ensemble des instances de cette classe.

De plus un attribut déclaré **final** n'est pas modifiable.

Exemples :

```
/* constante utilisable de l'extérieur, non modifiable */
```

```
public static final int C = 100;
```

```
/* constante non utilisable de l'extérieur, non modifiable */
```

```
private static final int D = 50;
```

```
/* attribut de classe par défaut, modifiable dans la classe */
```

```
private static double valeurParDefaut = 1000;
```

```
/* liste des instances de la classe */
```

```
private static Vector tousLesRépertoires
```

Pour retrouver une instance de classe à partir d'un identifiant (par exemple un nom)

- Mémoriser la liste des instances créées dans un *Vector* ou une *Hashtable* dans le programme principal ou dans un **attribut de la classe** considérée (Exemple : *tousLesRepertoires*)
- Créer une classe pour implémenter les ensembles d'instances de la classe et les mémoriser dans un **attribut d'instance** (Exemple : la classe *Banque* et l'attribut *lesComptes*)
- *Sauvegarder* les instances de la classe sur disque avec un identifiant et *charger* une instance à partir de son identifiant.

Exemple : la classe Compte

Programmation Java 2003/2004 - D.Pastre

Chap.4.7

Première version, très simple

```
public class Compte0 {  
    private String titulaire;           // attributs d'instance  
    private int numero; // ou protected  
    private double solde;  
    private Vector releve = new Vector(); // ou protected  
    private static int nbComptes=0;    // attribut de classe  
  
    public Compte0(String nomTitulaire) {    // 1er constructeur  
        Compte0.nbComptes++; // ou this.nbComptes++; // ou nbComptes++;  
        this.numero = nbComptes;  
        this.titulaire = nomTitulaire;}  
  
    public Compte0(String nomTitulaire, double versementInitial) {  
        this(nomTitulaire);           // 2ème constructeur  
        this.crediter(versementInitial);}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.8

```

public void crediter(double montant) {
    if (montant > 0) this.enregistrer(montant); else ... ;}
public void enregistrer(double montant) {
    this.solde += montant;
    this.releve.addElement(new Double(montant)); // ou ... (""+montant);}
public void debiter(double montant) {
    if (montant > 0)
        if (this.isDebitPossible(montant))this.enregistrer(-montant); else ... ;
    else ... ;}
public boolean isDebitPossible(double montant) {
    return montant < this.solde;}
public void virer(double montant, Compte0 c) {
    if (montant > 0) if (isDebitPossible(montant)) {this.debiter(montant);
                                                    c.crediter(montant);}
    else ... ;
    else ... ;}

```

Programmation Java 2003/2004 - D.Pastre

Chap.4.9

```

public int getNumero() {return this.numero;}
public String getTitulaire() {return this.titulaire;}
public double getSolde() {return this.solde;}
public String getReleve() {
    return this.toString()+ "relevé : "+this.releve.toString();}

public String RIB() {
    return "compte numéro : "+ this.numero
        + "\ntitulaire : " + this.titulaire;}

public String toString() {
    String s = "\n=====n"
        + this.RIB() + "\nsolde : " + this.solde;
    s+="\n=====n";
    return s; }
}

```

Programmation Java 2003/2004 - D.Pastre

Chap.4.10

Deuxième version, plus complète

```
public class Compte {  
    ...  
    // attribut d'instance  
    private double decouvertAutorise;  
    // attribut de classe  
    private static double decouvertAutoriseParDefaut = 100;  
  
    public Compte(String nomTitulaire) { // premier constructeur  
        ...  
        this.decouvertAutorise = Compte.decouvertAutoriseParDefaut;  
        // ou this.decouvertAutorise = this.decouvertAutoriseParDefaut;  
        // ou this.decouvertAutorise = decouvertAutoriseParDefaut;  
        // ou setDecouvertAutorise(decouvertAutorise, decouvertAutoriseParDefaut);  
    }  
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.11

```
public static final int ESPECES = 0; // constantes  
public static final int CHEQUE = 1;  
public static final int CB = 2;  
public static final int VIREMENT = 3;  
  
// deux autres constructeurs  
public Compte(String nomTitulaire, double versementInitial,  
              int moyenPaiement, double decouvertAutorise) {  
    this(nomTitulaire);  
    this.crediter(versementInitial, moyenPaiement);  
    this.decouvertAutorise = decouvertAutorise;}  
  
public Compte(String nomTitulaire, double versementInitial,  
              int moyenPaiement) {  
    this(nomTitulaire, versementInitial, moyenPaiement,  
        Compte.decouvertAutoriseParDefaut);} }
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.12

```

public double getDecouvertAutorise() {return this.decouvertAutorise;}

public void setDecouvertAutorise(double decouvertAutorise) {
    if (decouvertAutorise >0) this.decouvertAutorise = decouvertAutorise;}

public static double getDecouvertAutoriseParDefault() {
    return Compte.decouvertAutoriseParDefault;}

public static void setDecouvertAutoriseParDefault(
    double decouvertAutoriseParDefault) {
    if (decouvertAutoriseParDefault >0)
        Compte.decouvertAutoriseParDefault
            = decouvertAutoriseParDefault;}

```

```

// attributs
private static final String[] lesMoyensPaiement =
    {"espèces ", "chèque ", " CB ", "virement"};
protected static final DecimalFormat df =
    new DecimalFormat("###0.00");

public void crediter(double montant, int moyenPaiement) {
    if (montant > 0) this.enregistrer(montant, moyenPaiement); else ... ;}

public void enregistrer(double montant, int moyenPaiement) {
    this.solde += montant;
    String date = new Date().toString();
    date = date.substring(4,16)+date.substring(23,28);
    this.releve.addElement(date+" "+lesMoyensPaiement[moyenPaiement]+
        ((montant >0)?" crédit +":" débit ") +Compte.df.format(montant));}

```

```

public void debiter(double montant, int moyenPaielement) {
    if (montant > 0) if (this.isDebitPossible(montant))
        this.enregistrer(-montant, moyenPaielement);
        else ... ;
    else ... ;}

public boolean isDebitPossible(double montant) {
    return montant < this.solde+this.decouvertAutorise;}

public void virer(double montant, Compte c) {
    if (montant > 0) if (isDebitPossible(montant)) {
        this.debiter(montant, Compte.VIREMENT);
        c.crediter(montant, Compte.VIREMENT);}
        else ... ;
    else ... ; }

```

Programmation Java 2003/2004 - D.Pastre

Chap.4.15

```

public String getReleve() {
    StringBuffer sb = new StringBuffer(this.toString()+"relevé : \n");
    for (int i=0; i<this.releve.size();i++)
        sb.append(this.releve.elementAt(i)+"\n");
    return sb.toString();
}

public String toString() {
    String s = "\n===== \n"
        + this.RIB() + "\nsolde : " + Compte.df.format(this.solde)
        + "\ndéouvert autorisé : "
            + Compte.df.format(this.decouvertAutorise);
    if (this.solde < 0) s += "\nvous êtes en découvert";
    s+="\n===== \n";
    return s;
}

```

Programmation Java 2003/2004 - D.Pastre

Chap.4.16

Avec la classe Banque

```
public class Banque {  
    private String nom; private String code;  
    private Vector lesComptes;  
  
    public Banque(String nom, String code) {  
        this.nom=nom; this.code = code;  
        this.lesComptes = new Vector();}  
  
    public String getNom() { return this.nom;}  
    public String getCode() { return this.code;}  
  
    public int nouveauNumeroCompte(Compte compte) {  
        this.lesComptes.add(compte);  
        return this.lesComptes.size();}  
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.17

```
public CompteEnBanque getCompte(String titulaire) {  
    for (int i=0; i<this.lesComptes.size(); i++) {  
        CompteEnBanque c =  
            (CompteEnBanque) this.lesComptes.elementAt(i);  
        if (c.getTitulaire().equals(titulaire)) return(c);  
    }  
    return null;  
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.18

```

public class CompteEnBanque {
    ...
    private Banque banque;
    ...
    public CompteEnBanque (Banque banque, String nomTitulaire) {
        this.banque=banque;
        this.numero = banque.nouveauNumeroCompte(this);
        ...}
    public CompteEnBanque (Banque banque, String nomTitulaire, ...) {
        this(banque, nomTitulaire, ...);
        ... }

    public String RIB() {
        return "banque "+this.banque.getNom()+"\ncompte numéro : "
            + this.banque.getCode()+"_"+this.numero + "\ntitulaire : "
            + this.titulaire;}
    }

```

Programmation Java 2003/2004 - D.Pastre

Chap.4.19

Sauvegardes

```

public class CompteAvecSauvegarde implements Serializable {
    /** Enregistre ce compte dans un fichier de nom le titulaire du compte,
     *  et le nombre de comptes ouverts dans un fichier de nom 'nbComptes'.
     *  En cas d'échec, affiche un message. */
    public void sauvegarder(){ ... }
    /** S'il existe un fichier de nom 'titulaire',
     *    renvoie le compte enregistré dans celui-ci
     *    sinon crée un nouveau compte pour 'titulaire'.
     *  S'il existe un fichier de nom "nbComptes" contenant un entier
     *    plus grand que la constante nbComptes, nbComptes sera
     *    remplacé par cette valeur.
     *  En cas d'échec affiche un message. */
    public static CompteAvecSauvegarde charger(String titulaire) { ... }
}

```

Programmation Java 2003/2004 - D.Pastre

Chap.4.20

```
public class TestCompteAvecSauvegarde {  
    public static void main(String[] args) {  
        ...  
        String nom = Clavier.readString("nom du titulaire : ");  
        CompteAvecSauvegarde c = CompteAvecSauvegarde.charger(nom);  
        ...  
        c.sauvegarder();  
    }  
}
```

Sous-classes

Exemples

Classes dérivées

- Une **sous-classe** (ou classe **dérivée**) **hérite** des attributs et méthodes de sa classe **de base** ou classe **mère**.
- On peut **ajouter** des attributs et des fonctions (constructeurs et méthodes).
- On peut **redéfinir** des méthodes ("**surcharge**").
- On ne peut pas redéfinir des attributs.

- Pour faire appel à un constructeur ou une méthode de la classe mère, on utilise le mot-clef **super**
 - **super(...)** appelle le constructeur de la classe de base
 - dans **super.f(...)**, *super* remplace *this* et c'est la fonction *f* de la classe de base qui sera appliquée à *cet* objet

une **sous-classe** correspond à

- une **spécialisation**

Exemples : vertébré / mammifère,
matrice / matrice carrée / matrice symétrique,
Number / *Integer*
 / *Double*
 / ...
Vector / VecteurEntiers
Liste / ListeEntiers
parallélogramme / rectangle / carre
 / losange

ou à

- un **enrichissement**

Exemples : point/point coloré
Random / RandomEtendu
Compte / ...

Méthodes et classes abstraites

- Une méthode est dite **abstraite** si elle n'est définie que dans une (des) sous-classe(s).
Elle doit être déclarée dans la classe de base.
- Une classe est dite abstraite si elle a au moins une méthode abstraite

Exemples : *Number* / ...

Figure / Rectangle
/ Cercle

La classe **Number**

```
package java.lang;
public abstract class Number
    byte byteValue()
        Returns the value of the specified number as a byte.
    short shortValue()
        Returns the value of the specified number as a short.
    abstract int intValue()
        Returns the value of the specified number as an int.
    abstract long longValue()
        Returns the value of the specified number as a long.
    abstract float floatValue()
        Returns the value of the specified number as a float.
    abstract double doubleValue()
        Returns the value of the specified number as a double.
```

La classe **VecteurdEntiers** spécialisation de *Vector*

```
/** implémente des vecteurs d'entiers (Integer) */
public class VecteurdEntiers extends Vector {
    int taille = 0; // attribut non défini dans Vector

    /** crée un vecteur d'entiers vide */
    public VecteurdEntiers() {super();} // facultatif

    /** ajoute un entier (chaine ou Integer) à ce vecteur d'entiers */
    public void addElement(Object o) { // redéfinition de la méthode de Vector
        Integer I;
        if (o instanceof Integer) I = o;
        else if (o instanceof String) I = (Integer.valueOf((String) o)); else return;
        super.addElement(I); // appel de la méthode de Vector
        this.taille++;
    }
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.27

```
/** renvoie la somme des éléments de ce vecteur d'entiers */
public int somme() {
    int s=0;
    for (int i=0;i<this.taille;i++) // au lieu de for (int i=0;i<this.size();i++)
        s += ((Integer) this.elementAt(i)).intValue();
    return s;
}

/** renvoie un vecteur d'entiers somme de ce vecteur d'entiers
 * et du vecteur d'entiers spécifié */
public VecteurdEntiers somme(VecteurdEntiers v) { ... }

/** renvoie une représentation sous forme de chaine de ce vecteur
 * d'entiers et de sa taille */
public String toString() { // redéfinition de la méthode de Vector
    // pour faire apparaitre la taille
    return "vecteur de "+this.taille+" entiers "+super.toString();}
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.28

La classe **RandomEtendu** enrichissement de *Random*

```
/** enrichissement de la classe Random */
public class RandomEtendu extends Random {

    /** crée un nouveau générateur de nombres aléatoires */
    public RandomEtendu() {super();} // facultatif

    /** renvoie le prochain entier pseudoaléatoire compris entre les
     *  deux valeurs spécifiées */
    public int nextInt(int min, int max) {
        return nextInt(max-min+1)+min;
    }
}
```

Autres exemples (Exercices)

- Définir une sous-classe **VecteurTrie** de la classe **VecteurEntiers**, dans laquelle la méthode *addElement* insèrera les nombres à leur place.
- Définir la classe **Compte** comme sous-classe de *Compte0* puis les classes **CompteEnBanque** et **CompteAvecSauvegarde** comme sous-classes de *Compte*

Interfaces

Une interface est une classe abstraite dont

- tous les membres sont publics
- toutes les méthodes sont abstraites
- tous les attributs sont *static* et *final*

Exemples : *Enumeration*

Figure / Rectangle
/ Cercle

Une interface spécifie les méthodes qu'une classe doit avoir,
sans indiquer comment les réaliser

L'interface Enumeration

Méthodes

boolean **hasMoreElements()**;

Tests if this enumeration contains more elements

Object **nextElement()**;

Returns the next element of this enumeration if this enumeration
object has at least one more element to provide

Dans la classe *Vector* (resp. *Hashtable*) on a la méthode

Enumeration **elements()**;

Returns an enumeration of the components (resp. values) of this
Vector (resp. *Hashtable*)

Exemples d'utilisation

Pour imprimer tous les éléments d'un *Vector* v :

```
for (Enumeration e = v.elements(); e.hasMoreElements();)  
    System.out.println(e.nextElement()+" ");
```

Pour imprimer tous les éléments d'une *Hashtable* h

```
for (Enumeration e = h.elements();e.hasMoreElements();)  
    System.out.print(e.nextElement()+" ");
```

Pour énumérer les clés de la *Hashtable*

```
for (Enumeration e = h.keys();e.hasMoreElements();)  
    System.out.print(e.nextElement()+" ");
```

Pour énumérer les éléments rangés dans la *Hashtable*

```
for (Enumeration e = h.keys();e.hasMoreElements();)  
    System.out.print(h.get(e.nextElement())+" ");
```

Pour trouver la clé d'un object obj

```
boolean trouve=false;  
for (Enumeration e = h.keys();e.hasMoreElements() && !trouve;) {  
    Object cle = e.nextElement();  
    if (h.get(cle).equals(obj)) {  
        trouve = true;  
        System.out.println("la cle de "+obj+" est "+cle);  
    }  
}
```

Définition de classes implémentant l'interface *Enumeration*

```
/** classe implémentation l'interface Enumeration pour énumérer
 * les éléments d'un tableau d'entiers */
import java.util.Enumeration;
public class TabEnumeration implements Enumeration {
    private int[] tab;
    private int indice ;
    public TabEnumeration(int [] tab) {this.tab=tab; this.indice = 0;}
    public boolean hasMoreElements() {
        return this.indice < this.tab.length;}
    public Object nextElement() {
        return new Integer(tab[this.indice++]);}
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.35

```
/** classe implémentation l'interface Enumeration pour énumérer
 * les éléments d'un tableau de tableaux */
public class Tab2Enumeration implements Enumeration {
    private int[][] tab;
    private int i,j ;
    public Tab2Enumeration(int [][] tab) {
        this.tab=tab;this.i = this.j = 0;}
    public boolean hasMoreElements() {
        return (this.i<this.tab.length);}
    public Object nextElement() {
        int val = this.tab[i][j];
        if (this.j < this.tab[i].length-1) this.j++;
        else {this.i++;this.j=0;}
        return new Integer(val);
    }
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.36

Utilisation de ces énumérations

```
int [] tab = {5,9,8,6,7};  
for (Enumeration e= new TabEnumeration(tab);e.hasMoreElements();)  
    System.out.print(e.nextElement()+" ");  
  
int [][] tab2 = {{1,2,3,4,5},{10,11},{100,101,102}};  
for (Enumeration e= new Tab2Enumeration(tab2);e.hasMoreElements();)  
    System.out.print(e.nextElement()+" ");  
}  
}
```

Exercices

- Reprendre les exercices des chapitres précédents et remplacer les balayages par des énumérations.

Affectations

Copies

Passages de paramètres

Affectations et copies

Une variable de type élémentaire fait référence à son contenu.

Une affectation correspond donc à une copie.

Une variable de type tableau ou objet fait référence à son adresse.

Une affectation copie l'adresse mais non le tableau ou l'objet.

(Il s'agit en fait d'un pointeur (comme en C) mais implicite.)

Passages de paramètres

Les passages de paramètres se font par valeur.

Pour les tableaux et les objets, comme les paramètres sont en réalité des pointeurs, on peut dire que les passages se font par référence.

Exemples

// variables de types int

```
int n=5; int m=n; System.out.println("n="+n+" m="+m);
```

```
n=n+1; System.out.println("n="+n+" m="+m);
```

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

n=5 m=5
n=6 m=5
n=6 m=-5

```
// tableaux
```

```
int [] tab, tab1;
```

```
tab = new int[3];
```

```
for (int i=0;i<3;i++) tab[i]=i+1;
```

```
// affectation
```

```
tab1=tab;
```

```
System.out.println("tab="+toString(tab)+"tab1="+toString(tab1));
```

```
tab[0] = - tab[0]; tab1[1] = - tab1[1];
```

```
System.out.println("tab="+toString(tab)+"tab1="+toString(tab1));
```

```
/** Renvoie une chaine de caractères  
représentant le tableau spécifié. */
```

```
public static String toString(int[] tab) {  
    ...  
}
```

```
affectation
```

```
tab=1 2 3 tab1=1 2 3
```

```
tab=-1 -2 3 tab1=-1 -2 3
```

```
// copie ou clone
```

```
for (int i=0;i<3;i++) tab[i]=i+10;
```

```
tab1 = copie(tab);
```

```
// ou
```

```
tab1 = (int[]) tab.clone();
```

clone() est une méthode de la classe Object
transtypage nécessaire

```
System.out.println("tab="+toString(tab)+" tab1="+toString(tab1));
```

```
tab[0] = - tab[0]; tab1[1] = - tab1[1];
```

```
System.out.println("tab="+toString(tab)+" tab1="+toString(tab1));
```

```
/** Renvoie une copie du tableau spécifié. */
```

```
public static int[] copie(int[] tab) {
```

```
    int[] t = new int[tab.length];
```

```
    for (int i=0;i<tab.length;i++) t[i]=tab[i];
```

```
    return t;
```

```
}
```

```
tab=10 11 12 tab1=10 11 12
```

```
tab=-10 11 12 tab1=10 -11 12
```

```
// Vector
```

```
Vector v, v1;
```

```
// affectation
```

```
v = new Vector();
```

```
for (int i=0;i<3;i++) v.addElement(new Integer(i+10)); v1=v;
```

```
System.out.println("v="+v+" v1="+v1);
```

```
v.addElement(new Integer(20));v1.addElement(new Integer(30));
```

```
System.out.println("v="+v+" v1="+v1);
```

```
v=[10, 11, 12] v1=[10, 11, 12]
```

```
v=[10, 11, 12, 20, 30] v1=[10, 11, 12, 20, 30]
```

```
// copie ou clone
```

```
v = new Vector();
```

```
for (int i=0;i<3;i++) v.addElement(new Integer(i+10));
```

```
v1 = copie(v);
```

```
// ou
```

```
v1 = (Vector) v.clone();
```

clone() est une méthode de la classe Object
transtypage nécessaire

```
System.out.println("v="+v+" v1="+v1);
```

```
v.addElement(new Integer(20));v1.addElement(new Integer(30));
```

```
System.out.println("v="+v+" v1="+v1);
```

```
/** Renvoie une copie du Vector spécifié. */
```

```
public static Vector copie(Vector v) {
```

```
    Vector vc = new Vector();
```

```
    for (int i=0;i<v.size();i++) vc.addElement(v.elementAt(i));
```

```
    return vc;
```

```
}
```

```
v=[10, 11, 12] v1=[10, 11, 12]
```

```
v=[10, 11, 12, 20] v1=[10, 11, 12, 30]
```

```
// Classe Cla
Cla c,c1;
// affectation
c = new Cla(10,11); c1 = c;
System.out.println("c="+c+" c1="+c1);
c.setAttribut1(20);c1.setAttribut2(30);
System.out.println("c="+c+" c1="+c1);
```

```
c=Cla-10-11 c1=Cla-10-11
c=Cla-20-30 c1=Cla-20-30
```

```
public class Cla {
    private int attribut1, attribut2;
    public Cla(int a, int b) {this.attribut1=a; this.attribut2=b;}
    public String toString() { return "Cla-"+attribut1+"-"+attribut2;}
    public void setAttribut1(int val) { this.attribut1=val;}
    public void setAttribut2(int val) { this.attribut2=val;}
}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.45

```
// copie ou clone
c = new Cla(10,11);
c1 = c.copie();
// ou
c1 = (Cla) c.clone();
System.out.println("c="+c+" c1="+c1);
c.setAttribut1(0);c1.setAttribut2(1);
System.out.println("c="+c+" c1="+c1);
```

```
c=Cla-10-11 c1=Cla-10-11
c=Cla-0-11 c1=Cla-10-1
```

```
/** Renvoie une copie de cette instance de Cla. */
public Cla copie() {Cla c= new Cla(this.attribut1, this.attribut2);
    return c;}

/** Redéfinit la méthode clone() de la classe Object. */
public Object clone() {Cla c= new Cla(this.attribut1, this.attribut2);
    return c;}
```

Programmation Java 2003/2004 - D.Pastre

Chap.4.46

Passage de paramètres

```
// variable de type int
int a=5;
System.out.println("a="+a);
int b= f(a);
System.out.println("a="+a+" b="+b);
```

```
a=5
a=5 b=-5
```

```
/** Renvoie l'opposé de 'a' modifié. */
public static int f(int a) {a=-a;return a;}
```

```
// tableau
int[] tab = new int[1];
tab[0]=10;
System.out.println("tab[0]="+tab[0]);
p(tab);
System.out.println("tab[0]="+tab[0]);
```

```
/** Ajoute 1 au premier élément du tableau spécifié. */
public static void p(int[] tab) {tab[0]=tab[0]+1;}
```

```
tab[0]=10
tab[0]=11
```



```
// Vector
Vector v = new Vector();
v.addElement("5");
System.out.println("v="+v);
p("12",v);
System.out.println("v="+v);
```

```
/** Ajoute la chaine spécifiée au Vector spécifié. */
public static void p(String s, Vector v) {v.addElement(s);}
```

```
v=[5]
v=[5, 12]
```

```
// instance de la classe Cla
Cla c = new Cla(10,12);
System.out.println("c="+c);
p(20,c);
System.out.println("c="+c);
```

```
/** Donne la valeur spécifiée à l'attribut1 de l'instance de Cla
 * spécifiée. */
public static void p(int n,Cla c) {c.setAttribut1(n);}
```

```
c=Cla-10-12
c=Cla-20-12
```