

L3 Informatique - 2024-2025

Partie POO

Atelier 1 – Classes et Objets



Objectifs de ce cours

- Découvrir les concepts fondamentaux de la programmation orientée objet
- Apprendre à programmer des classes simples
- Savoir utiliser des bibliothèques de classes



CH1 – Classes et Objets

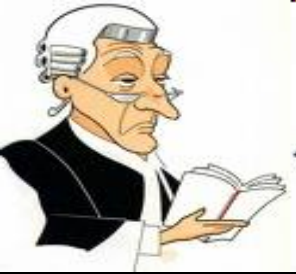


- ▶ Notion intuitive d'Objet
- ▶ Définitions de classe
- ▶ Création d'instances (ou objets)
- ▶ Déclaration et invocation de méthodes
- ▶ Principe d'encapsulation et visibilité
- ▶ Définition de constructeur
- ▶ Attributs et Méthodes de classe
- ▶ Surcharge de constructeurs et de méthodes
- ▶ Spécificités des objets
- ▶ Tableaux statiques en java
- ▶ Tableaux dynamiques (ArrayList)



Qu'est ce que la POO?

- Historique et Origines
- Structure d'un programme OO
- Principes de base
- Atouts de la POO
- Langages



Principes de base de la POO

Abstraction



*Objets et Classes,
Mécanisme d'instanciation*

Encapsulation



Attributs, méthodes, visibilité

Hiérarchie d'héritage



Polymorphisme

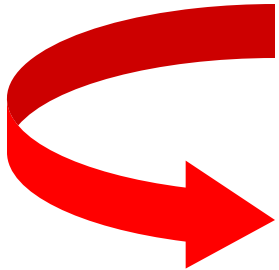


C'est l'objectif de ce cours!!

Atouts de la POO



- Code plus robuste (résistant aux changements)
- Code plus clair car plus proche de la pensée humaine :
 - Représentation naturelle des données complexes



- Maintenance facilitée
- Evolutivité
- Réutilisabilité
- Réduction des temps de développement

Naissance de java ...

JAVA – (1995)

James Gosling (SUN)

- Langage entièrement orienté objet
- Technologie portable (byte-code) adaptée à Internet
- Langage simple et fiable
- Syntaxe familière proche du C/C++ mais
 - Sans pointeur
 - Sans allocation et désallocation explicite de la mémoire



ORACLE®

Rachat en
2009

Présentation de JAVA

JAVA est une "plateforme"

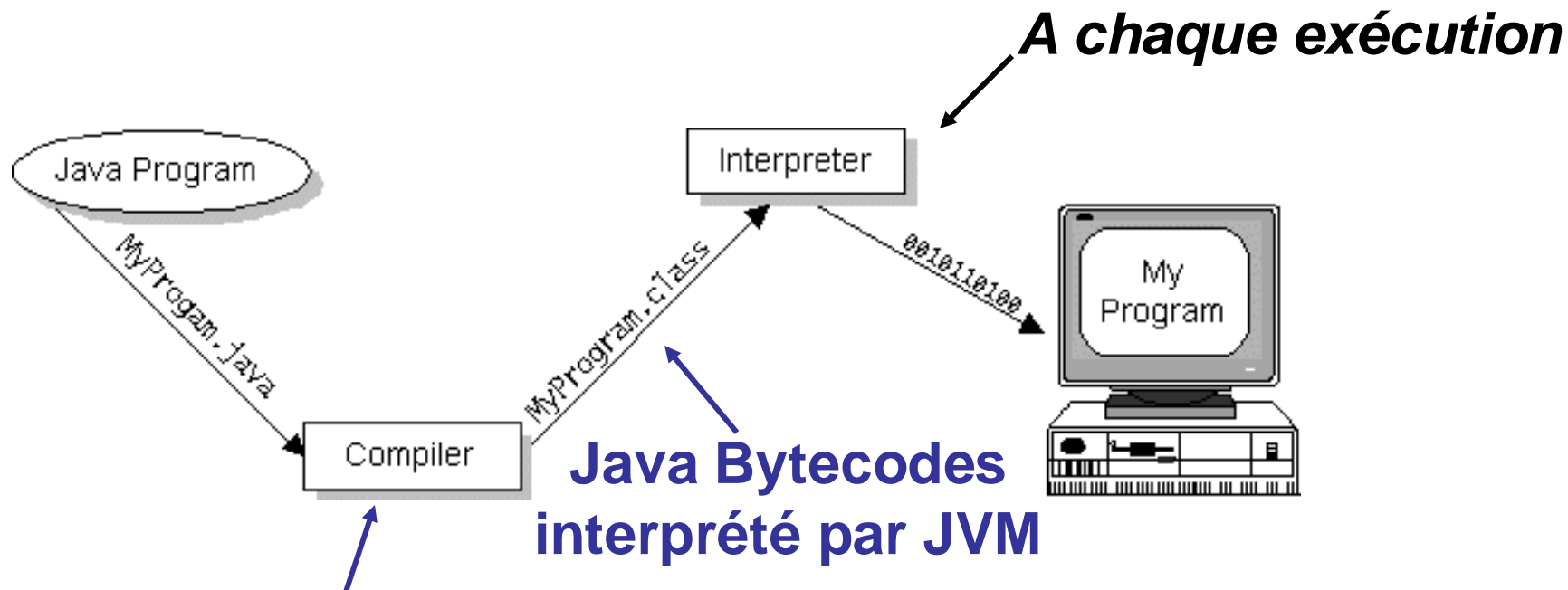
- Un langage de programmation orienté objets
 - Un ensemble de classes standards réparties dans différents **packages** (**API**)
 - Un ensemble d'outils (le **JDK**,...)
- Un environnement d'exécution :
la **machine virtuelle** (JVM)





Présentation de JAVA

- Java est **compilé** et **interprété**



Faite une seule fois

« **compile once, run everywhere** »

Présentation de JAVA

Le JDK

- Environnement de développement fourni par Oracle : Java Development Kit
- Il contient :
 - les classes de base de l'API java (plusieurs centaines),
 - la documentation au format HTML
 - le compilateur : **javac**
 - la JVM (machine virtuelle) : java
 - le visualiseur d'applets : appletviewer
 - le générateur de documentation : javadoc
 - etc.



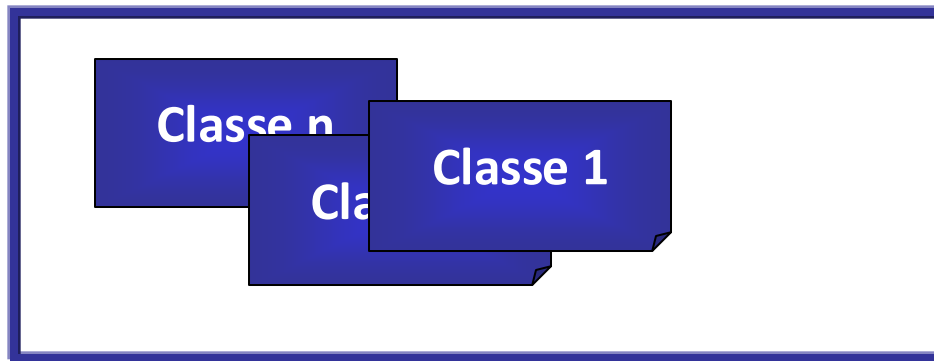
Présentation de JAVA

Documentation des classes

- Documentation standardisée au format HTML :
 - classes de l'API
 - possibilité de génération automatique avec l'outil **Javadoc**.
 - intérêt de l'hypertexte pour naviguer dans la documentation
- Accessible en ligne : <http://docs.oracle.com/javase/8/docs/api/index.html>
ou téléchargeable gratuitement

Présentation de JAVA

- **Programme Java** = ensemble de classes



En général, on a une classe par fichier. Ce n'est pas une obligation mais c'est préférable pour des raisons de clarté

Présentation de JAVA

- **Application Java indépendante**= une classe doit contenir la méthode « **main** »

Classe n

Classe ..

Classe

Classe Depart

```
public static void  
main(String args[ ])  
{  
.../...  
}
```



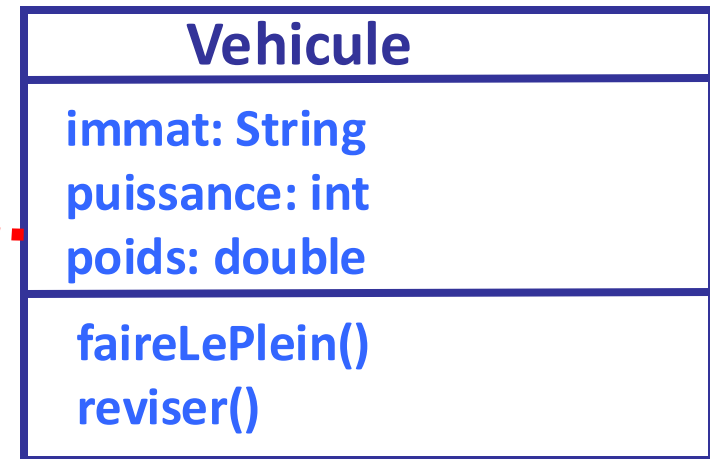
Définition de classes

Déclaration d'une classe en JAVA et en UML

Une classe en JAVA

```
class Vehicule{  
    /** l'immatriculation de ce véhicule */  
    String immat;  
    /** la puissance */  
    int puissance;  
    /** La consommation de ce véhicule. */  
    double poids;  
  
    ...  
    void faireLePlein() {  
        .....  
    }  
    void reviser() {  
        .....  
    }  
}
```

Une classe en UML



Déclaration de classe en JAVA

Type des Variables



Types primitifs

- byte, short, int, long
- float, double
- boolean
- char

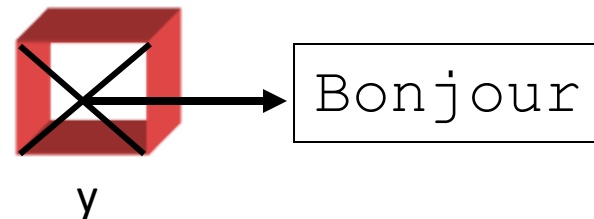
`int x=4`



Types objets (Classes)

- Classes de l'API java: String, ...
- Classes de l'application (nos propres classes)

`String y="Bonjour"`



Déclaration de classe en JAVA

Variables d'instances et Variables locales

```
class A{
```

```
    type a;
```

Déclaration d'une
Variable d'instance

```
    void uneMethode(...) {
```

```
        type b;
```

Déclaration d'une
Variable locale

```
    }
```





Création d'instances (ou objets)

Notion d'instance

Un objet (instance) est caractérisé par les **valeurs** de ses attributs.

v1:Vehicule

- immatriculation = « 1000GH2B »
- puissance = 18
- poids = 15



v2:Vehicule

- immatriculation = « 2222ML2A »
- puissance = 5
- poids = 2,5

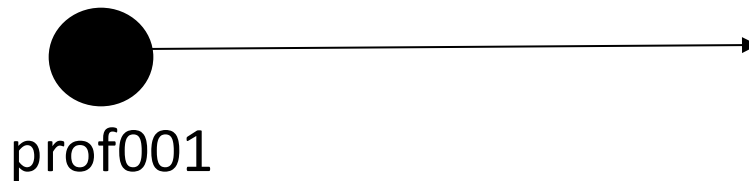


Valeurs propres à chaque instance



Notion d'objet en JAVA

Variable définie sur une classe



nom

age

discipline

grade

Statut

nbCoursMax

Nimbus

30

physique

MCF Ech.6

actif

4

nom	Nimbus
age	30
discipline	physique
grade	MCF Ech.6
Statut	actif
nbCoursMax	4

- Une **adresse (ou référence)** en mémoire qui permet d'identifier l'objet
- Un **état** qui est représenté par un ensemble de valeurs attribuées à ses **variables d'instances**
- Un **comportement** défini par des fonctions ou sous-programmes appelés **méthodes**

Création d'objets en Java



Etapes de création d'un objet

1. Déclaration d'une variable (référence)
2. Création de l'objet associé (instanciation)
3. Accès aux attributs et méthodes de l'objet

```
class Vehicule {  
    String immat;  
    int puissance;  
    ...  
}
```

```
class TestVehicule {  
    public static void main(String[ ] args) {  
        /* Création et manipulation  
        d'objets de la classe Véhicule */  
    }  
}
```

Création d'objets en JAVA

Déclaration d'une variable (Référence)




Vehicule monVehicule;

- monVehicule peut référencer un objet Vehicule
- l'objet de monVehicule n'existe pas encore !!!

Avant

Après

 →
monVehicule

??

monVehicule= *null*

Création d'objets en JAVA

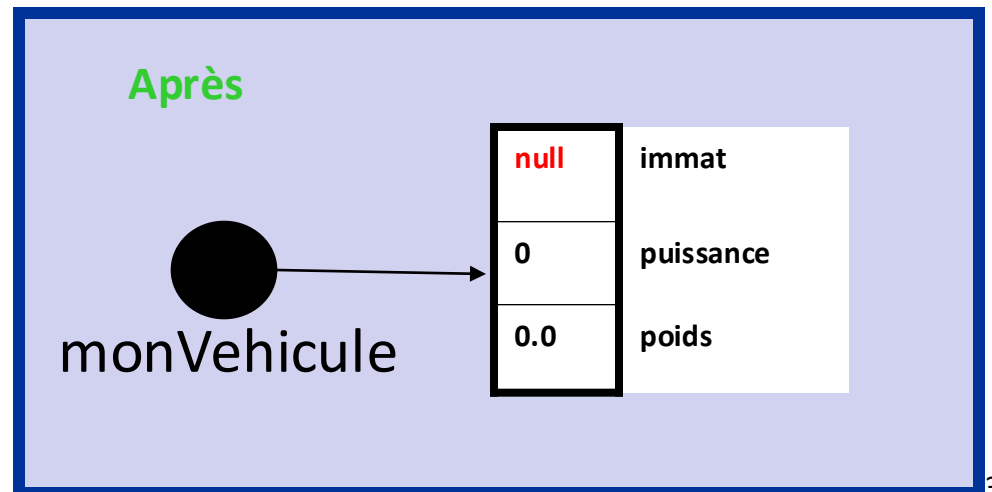
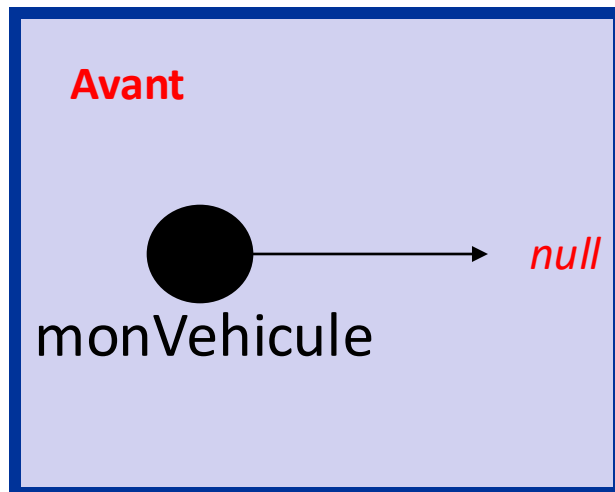


Création de l'objet (instanciation)

```
monVehicule = new Vehicule();
```

⇒ réserve la mémoire pour stocker l'objet

⇒ associe l'objet à la référence



Création d'objets en JAVA

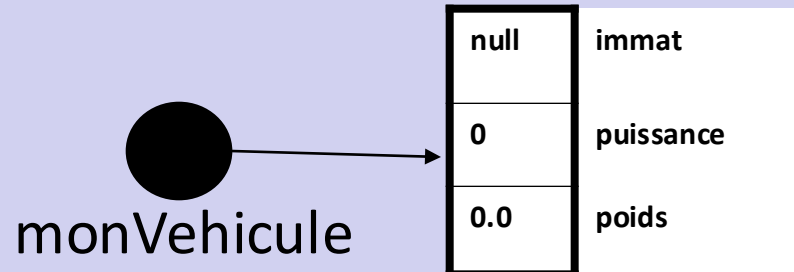


Déclaration et instanciation

Vehicule monVehicule = **new** Vehicule();

Avant

Après



Accès aux valeurs des attributs



Une classe

```
class Vehicule{  
    /** l'immatriculation de ce véhicule */  
    String immat;  
    /** La puissance */  
    int puissance;  
    /** Le poids de ce véhicule. */  
    double poids;  
    ... }  
}
```

Une classe de test

```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule monVehicule=new Vehicule();  
        monVehicule.immat="1000GH2B";  
        monVehicule.puissance=18;  
        monVehicule.poids=15.5;  
        ....  
    }  
}
```



Déclaration et invocation de méthodes

Déclarations de méthodes en Java

```
class Vehicule{  
    String immat;  
    double poids;  
    double jauge;  
    int age;  
  
    void remplirJauge (double quantite)  
    {  
        jauge + = quantite;  
    }  
  
    void afficherJauge ()  
    {  
        System.out.println("Le niveau de la jauge est : "+ jauge) ;  
    }  
}
```





Invocations de méthodes en Java

`monObjet.méthodeInvoquée(para1, para2,..., paran)`

```
class Vehicule {  
    String immat;  
    double poids;  
    double jauge;  
    int age;  
    void remplirJauge(double quantite)  
    {  
        ...  
    }  
    void afficherJauge ()  
    {  
        ...  
    }  
}
```

```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule v1=new Vehicule();  
        v1.immat="1000GH2B" ;  
        v1.poids=3.5;  
        v1.jauge=15;  
        v1.remplirJauge(100);  
        v1.afficherJauge();  
    }  
}
```

Déclarations et invocations de méthodes



```
class Vehicule {  
    String immat;  
    double poids;  
    double jauge;  
    int age;
```

```
    String identiteVehicule() {  
        String description= "Le véhicule "+  
            immat + " est âgé de " + age + " an(s)";  
        return description;  
    }
```

```
}  
  
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule v1=new Vehicule();  
        v1.immat="1000GH2B" ; v1.age=1; ....  
        System.out.println(v1.identiteVehicule());  
    }
```

Déclarations et invocations de méthodes



```
class Vehicule {
```

```
....
```

```
    boolean jaugeEstVide ()  
    {  
        return (jauge==0);  
    }
```

```
    void augmenterAge()  
    {  
        age++;  
    }
```

```
}
```

```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule v1=new Vehicule();  
        v1.augmenterAge();  
        if ( v1.jaugeEstVide() )  
            System.out.println("La jauge est vide");  
        else v1.afficherJauge();  
    }
```



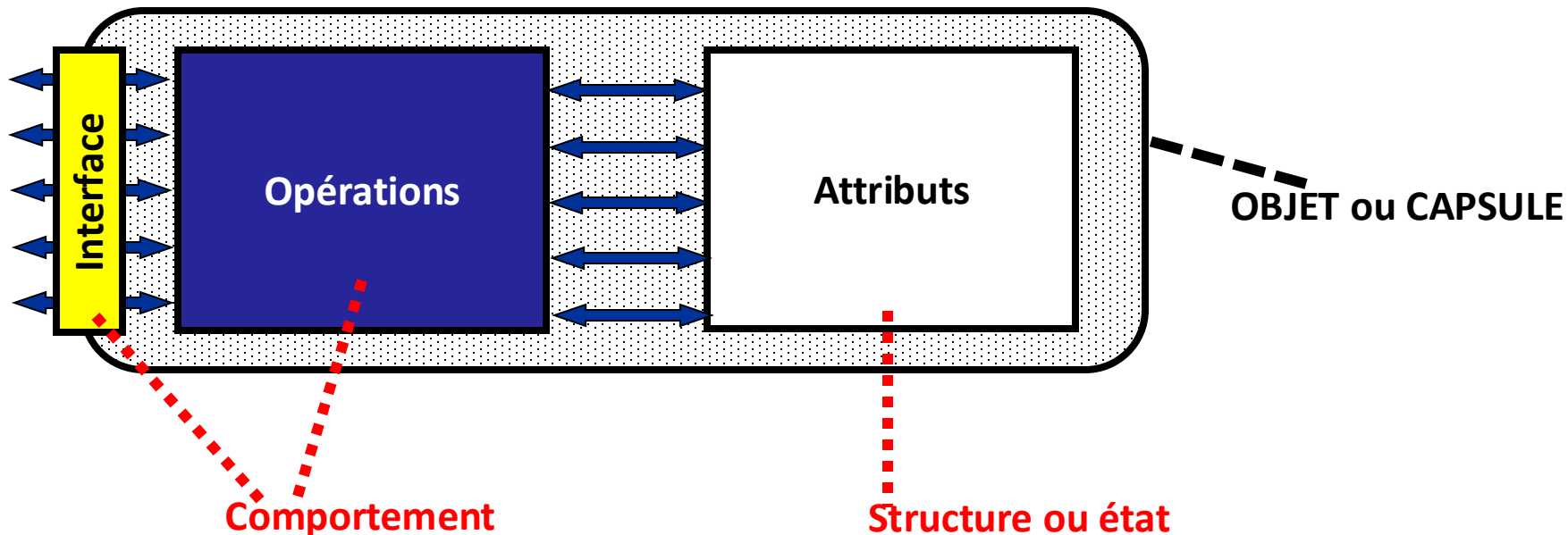
Principe d'encapsulation et visibilités

- Notion de visibilité
- Accesseurs et
modificateurs :
méthodes get et set

Encapsulation

Principe = séparation spécification/réalisation

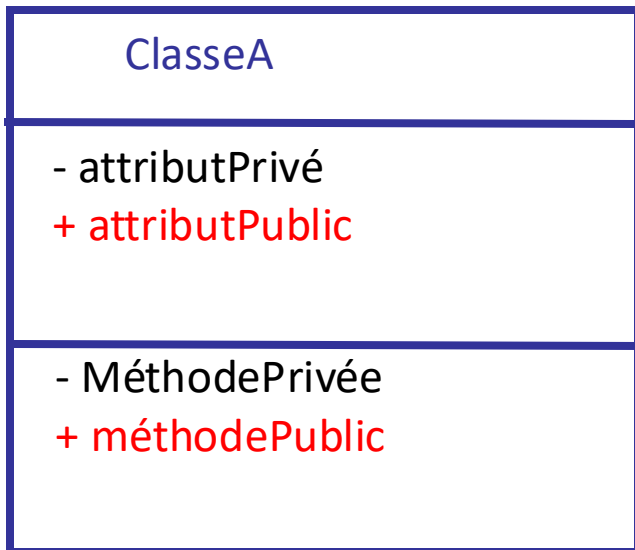
- Les objets ne sont manipulés qu'à travers leur interface
- Les détails de l'implémentation sont occultés



Encapsulation

Les niveaux de visibilité sont les outils de mise en œuvre de l'encapsulation

Niveaux de visibilité
en UML



Modificateurs en Java



private : accès réduit, seulement depuis la classe

public : accès libre depuis partout

package (ou rien) : accès depuis la classe et les classes du package



Mise en oeuvre du principe d'encapsulation

Visibilité des attributs et méthodes

```
public class Vehicule {  
    private String immat;  
    private short puissance;  
    private double jauge;  
}
```

```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule v1=new Vehicule();  
        v1.immat="1000GH2B" ;  
        v1.puissance=18;  
        v1.jauge=15;  
    }  
}
```


Les attributs doivent être invisibles à l'extérieur de la classe : ils sont déclarés **private**

Accès interdits


Mise en oeuvre du principe d'encapsulation



```
public class Vehicule {  
    private String immat;  
    private short puissance;  
    private double jauge;
```



```
    public void setImmat(String x)  
    {  
        immat = x;  
    }
```



```
    public String getImmat()  
    {  
        return immat;  
    }  
}
```

Déclaration de
méthodes d'accès et de
modification

Les « Getter/setter »

Pourquoi l'encapsulation?



- Pour sécuriser le code!
- Certaines classes sont développées par d'autres programmeurs (vos fournisseurs)
 - Ils vous offrent des services (méthodes)
 - Vous êtes de simples utilisateurs
 - vous n'avez pas à connaître la structure de leurs classes (attributs) et les algorithmes de leurs méthodes
 - Votre fournisseur doit pouvoir changer ses algorithmes sans que vous ayez à modifier vos programmes
- Vos classes pourront servir à d'autres programmeurs (vos clients)



Définition de constructeur



Instanciación et Constructeurs

```
public class TestVehicule
{
    public static void main(String args[])
    {
        Vehicule maVoiture = new Vehicule()
        maVoiture.setImmat("2222 AJ 2A");
    }
}
```

Création d'une
instance

Invocation d'un Constructeur

- Invocation implicite du **constructeur par défaut** si aucun constructeur n'est défini dans la classe
- Invocation d'un **constructeur défini** dans la classe



Constructeurs en Java

- Un **constructeur** est une méthode d'instanciation et d'initialisation

```
public class TestVehicule {  
    public static void main(String[] args) {  
        Vehicule maVoiture = new Vehicule ("2222 AJ 2A" , 6);  
        System.out.print(" Immatriculation " + maVoiture.getImma());  
    }  
....
```

```
public class Vehicule{  
    private String immat;  
    private int puissance;  
    public Vehicule(String i, int p) {  
        immat = i;  
        puissance = p;  
    }  
}
```



Le mot clé this

- Référence à l'instance (l'objet) courante
- **this** permet de lever une ambiguïté de nommage

```
public Vehicule(String immat, int puissance) {  
    this.immat = immat;  
    this.puissance = puissance;  
}
```

this.x fait référence au champs **x** de l'objet alors que **x** fait référence au premier argument du constructeur



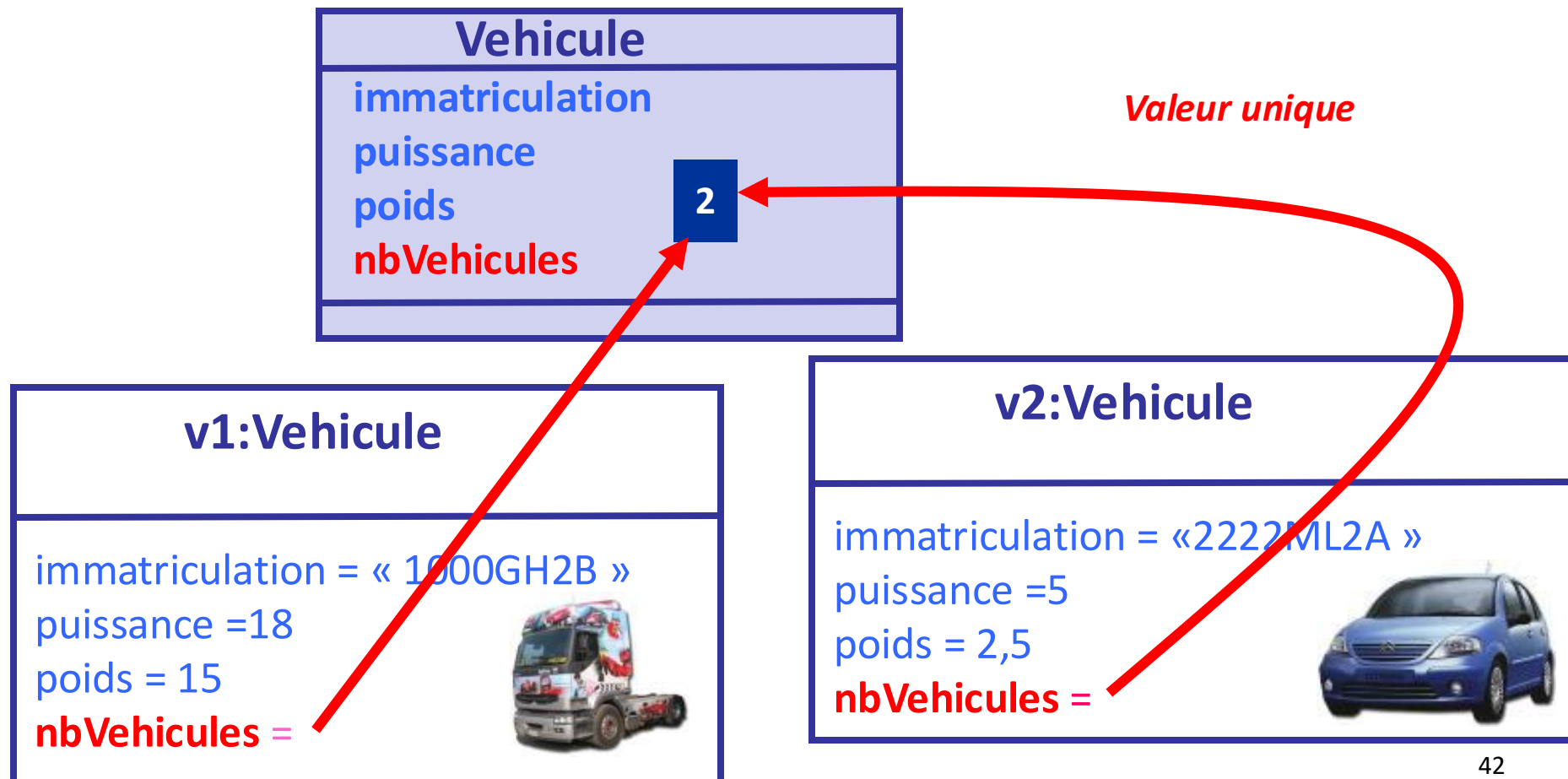
Attributs et méthodes « de classe »

- Attributs et méthodes static en Java
- Mot clé final
- Déclaration de constantes

Attributs « de classe »

Attribut de classe =

Valeur partagée par toutes les instances de la classe



Attributs de classe en Java

```
public class Vehicule{  
    /** l'immatriculation de ce véhicule */  
    private String immat;  
    /** la puissance */  
    private short puissance;  
  
    /** Nombre total de véhicules */  
    public static int nbVehicules = 0;  
    ...  
}
```

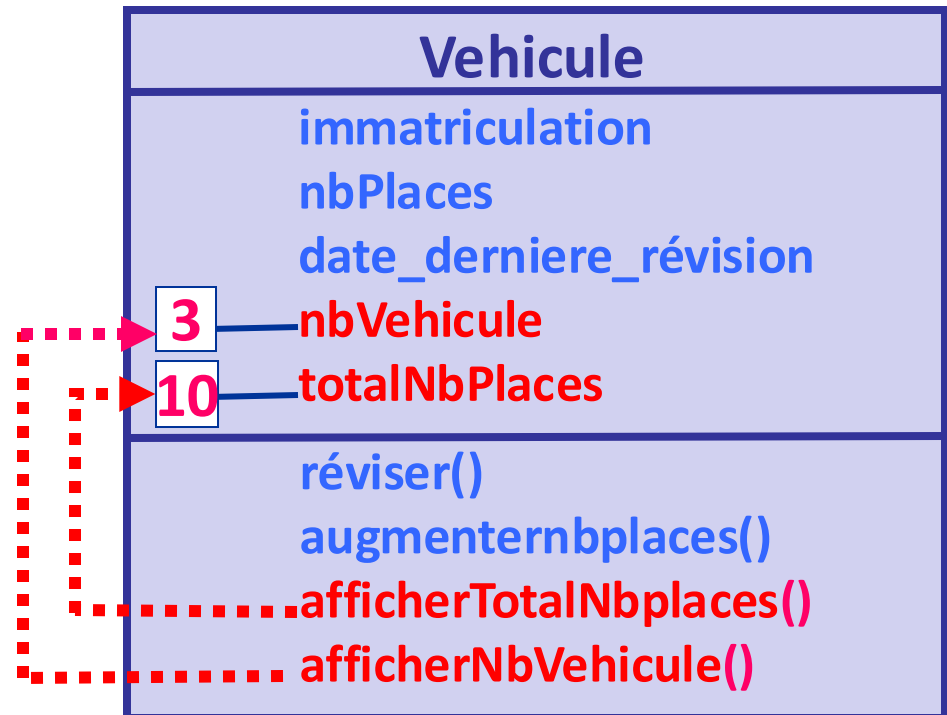
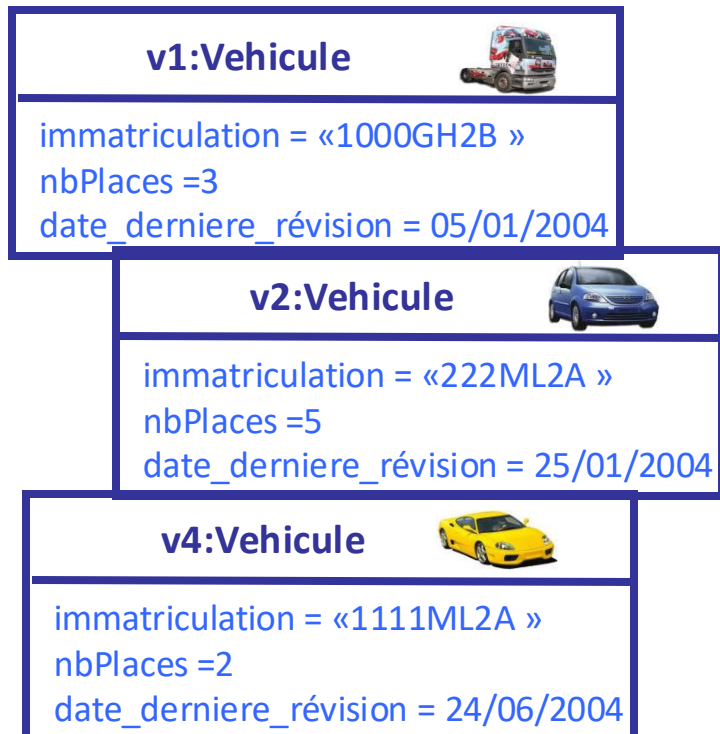


Variable ou champ statique:
attribut de classe en Java

Opérations de classe

Opérations de classe =

- Exécution déclenchée par un message **envoyé à la classe**.
- Une opération de classe ne peut manipuler que des attributs de classe



Opération de classe en Java



```
public class Vehicule{  
    private String immatriculation;  
    private int nbPlaces;  
    Private int age;  
    private static int nbVehicule=0;  
    private static int totalNbPlaces=0;  
    Public Vehicule(String immatriculation,  
                    int nbPlaces){  
        this.immatriculation=immatriculation;  
        this.nbPlaces=nbPlaces;  
        this.age=0;  
        nbVehicule++;  
        totalNbPlaces=totalNbPlaces+nbPlaces;  
    }  
    public void augmenterAge(){ age++;}  
    public static void afficherNbVehicule()  
        System.out.println("Nombre de Vehicules "+ nbVehicule);  
}
```

instance

```
class TestVehicule{  
    public static void main(String[] args){  
        Vehicule v1=new  
        Vehicule("2222 AJ 2A" , 6);  
        v1.augmenterAge();  
        Vehicule.afficherNbVehicule();  
    }  
}
```

classe

Opération de classe en Java



```
public class Vehicule{  
    private String immat;  
    private short puissance;  
    /** Nombre total de véhicules */  
    private static int nbVehicules = 0;  
    ....  
    public static int getNbVehicules(){  
        return nbVehicules;  
    }  
}
```

méthode statique:
méthode de classe en Java

v1 ou Vehicule ??
Vehicule: réponse correcte
v1: accepté mais déconseillé

```
class TestVehicule{  
    public static void main(String[] args){  
        Vehicule v1=new Vehicule();  
        System.out.println("Nombre de véhicules" +  
        ???.getNbVehicule());  
    }  
}
```

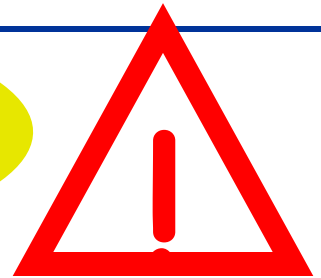


Opération de classe en Java

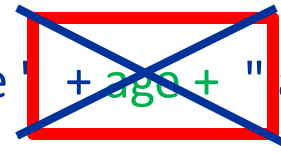
- Une méthode statique ne peut manipuler que les variables statiques de sa classe.
- Une méthode statique ne **peut pas manipuler des variables d'instances.**

```
public class Vehicule{  
    Private int age;  
    private static int nbVehicule=0;  
  
    public void augmenterAge(){ age++;}  
  
    public static void afficherNbVehicule(){  
        System.out.println("Nombre de Vehicules "+ nbVehicule + " de " + age + " ans");  
    }  
}
```

Ce code
comporte-t-il une
erreur?



INTERDIT car age est une variable d'instance





Le modificateur final

- Indique que la valeur d'une variable (d'instance, de classe, ou locale) ne peut être modifiée
- Elle ne peut recevoir de valeur **qu'une seule fois** : à la déclaration ou plus tard.
- Une variable d'instance déclarée **final** est constante pour chaque instance, mais peut avoir des valeurs différentes pour deux instances.



Méthodes en Java

Le modificateur final

- Assure à l'utilisateur que la valeur du paramètre passé n'est pas modifiée à l'intérieur de la méthode
 - Si c'est un type primitif la valeur reste inchangée

```
int methodeTest(final int i) {...} // i est inchangé
```

- Si c'est une référence à un objet, la référence sera inchangée, mais le contenu de l'objet lui peut être changé...

```
void methodeTest(final Personne p) {...}  
// p est inchangé, mais son nom peut l'être
```



Surcharge

- Surcharge de constructeurs
- Surcharge de méthodes



Surcharge de Constructeurs

- La classe offre **plusieurs possibilités** pour définir ses instances.

```
public class Vehicule{  
    private String immat;   private short puissance;  
  
    public Vehicule(String i, short p) {  
        immat = i;    puissance = p;  
    }  
    public Vehicule(String i) {  
        immat = i;    puissance = 0;  
    }  
    public Vehicule(Vehicule v) { //constructeur de copie  
        this.immat=v.immat; this.puissance=v.puissance;  
    }  
    public Vehicule() {  
        this.immat="" ;this.puissance=0;  
    }  
}
```



Surcharge de Constructeurs

- Le mot clé **this** permet d'invoquer un autre constructeur de la classe dans la définition

```
public class Vehicule{  
    private String immat;  private short puissance;  
    public Vehicule(String i, short p) {  
        immat = i;    puissance = p;  
    }  
    public Vehicule(String i) {  
        this(i, 0);  
    }  
    public Vehicule(Vehicule v) {  
        this(v.immat, v.puissance);  
    }  
    public Vehicule() {  
        this("", 0);  
    }  
}
```



Méthodes en Java

Surcharge ou Surdéfinition:

- Deux méthodes ont le **même nom** et le **même type de retour** mais des **signatures différentes**

Exemple : les constructeurs

- Le choix de la méthode appelée dépend des paramètres d'appel (déterminé à la compilation)

≠ Redéfinition (*cf. Héritage et Polymorphisme*)

- Des **méthodes différentes** ont le **même nom** et la **même signature**
- Le choix de la méthode appelée dépend du type réel de l'objet (déterminé à l'exécution)



La surcharge de méthodes

```
public double distance(Point p1, Point p2) { // }  
public double distance(Point p) { // ... }  
public int distance (Point p) { // ... }
```

Erreur de compilation



```
EquationCons(4, 9.81);  
// appel à EquationCons(int a, double b)  
EquationCons(9.81, 7);  
// fait appel à EquationCons(double a, int b)
```



Spécificité des objets

- Affectation
- Comparaison
- Copie
- Transmission de paramètres

Un objet est une référence:

Conséquences

- Affectations d'objets
 - Que copie-t-on?
- Comparaison d'objets
 - Que compare-t-on?
- Transmission d'objets en paramètres de méthodes
 - Que transmet-on?

Des références et non
des valeurs!

Objets, valeurs et affectations: un petit exemple

```
public class Point {  
    char nom;    // nom du point  
    double abs; // abscisse
```

```
public class TestObjet {  
    public static void main(String[] args) {  
        int x=10;  
        int y=x;  
        y++;  
        System.out.println("x="+x+" y="+y);  
        Point p1=new Point('A',10);  
        Point p2=p1;  
        p2.setAbs(12);  
        System.out.println("p1.abs="+p1.getAbs()+" p2.abs="+p2.getAbs());  
    }  
}
```

Qu'affiche le
programme suivant?

```
x=10 y=11  
p1.abs=12.0 p2.abs=12.0
```

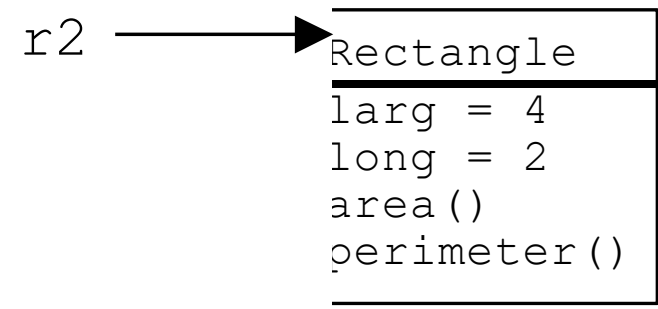
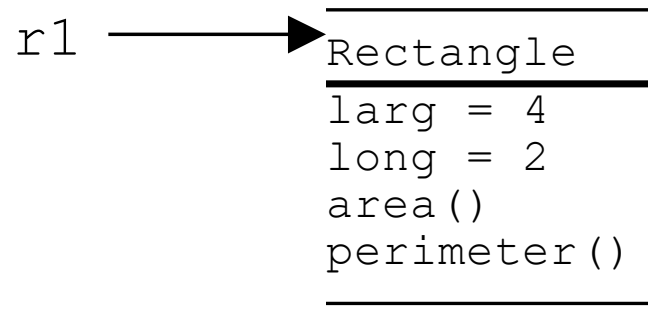


Comparaison d'objets

- Que compare-t'on ?

```
Rectangle r1 = new Rectangle(2,4);  
Rectangle r2 = new Rectangle(2,4);  
if (r1 == r2) then ...
```

Le test
rend
FAUX !!



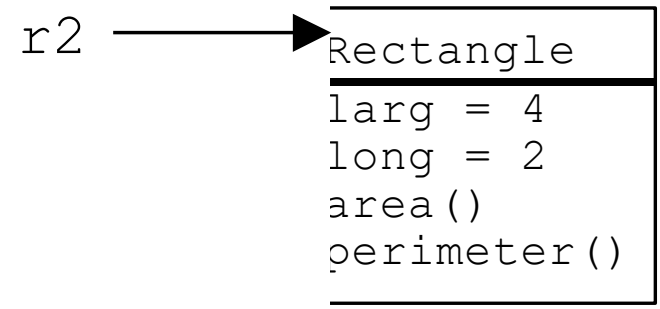
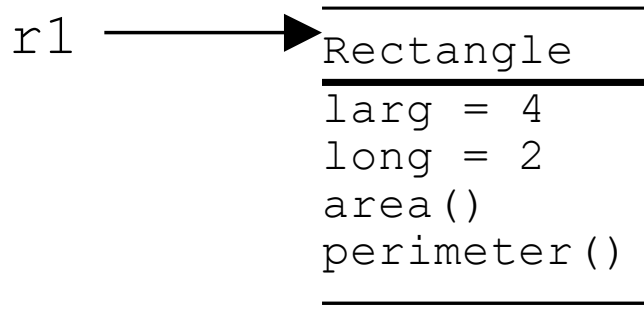


Comparaison d'objets

Méthode equals: pour comparer le contenu des objets et pas seulement les références

```
Rectangle r1 = new Rectangle(2,4);  
Rectangle r2 = new Rectangle(2,4);  
if (r1.equals(r2)) then ...
```

Le test
rend
VRAI !!



Utile pour la comparaison de Strings

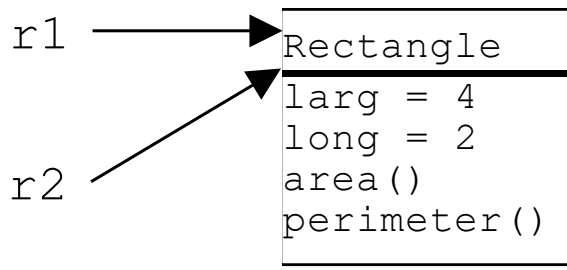
Il faut que la méthode equals ait été explicitement
redéfinie dans la classe Rectangle

Nous y reviendrons plus tard !
Cf. Chapitre 2-Héritage



Affectation d'objets

- Que copie-t-on?...



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

- Il n'y a pas copie, duplication, il n'y a toujours qu'un seul objet

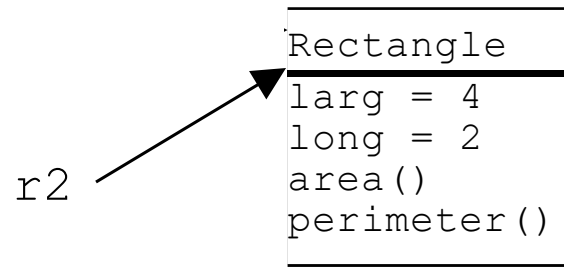
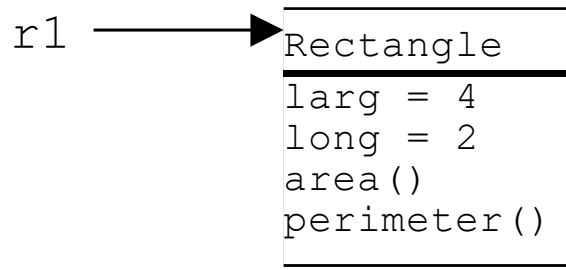
Une véritable copie est un « clonage » de l'objet



Clonage d'objets

- **Méthode clone**: Permet de faire une véritable copie, duplication d'un objet

```
Rectangle r1 = new Rectangle(2,4);  
Rectangle r2 = (Rectangle) r1.clone();  
if (r1==r2) then ...  
If (r1.equals(r2)) then ...
```



Il faut que la méthode clone ait été explicitement
redéfinie dans la classe Rectangle

Nous y reviendrons plus tard !
Cf. Chapitre 2-Héritage

Objets, valeurs et affectations: un autre petit exemple

```
public class TestObjet {  
    public static void main(String[] args) {  
        String s1="Bonjour";  
        String s2=s1;  
        s2+=" Monsieur";  
        System.out.println("s1="+s1+" s2="+s2);  
    }  
}
```

Qu'affiche le
programme suivant?

~~s1=Bonjour Monsieur s2=Bonjour Monsieur~~

OU

s1=Bonjour s2=Bonjour Monsieur

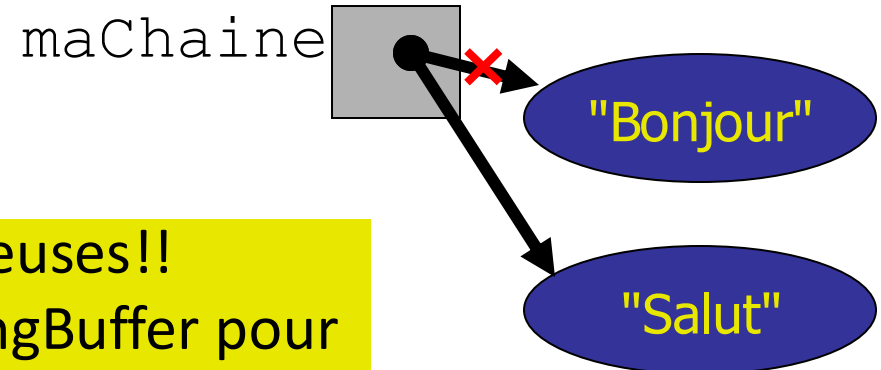
POURQUOI?

Les Strings sont des
objets immutables

Notion d'objets immutables

- Les objets de certaines classes ne peuvent pas être modifiés, ils sont dits « **immutables** »
- Si l'on tente de les modifier une **nouvelle instance est créée.**
- Un exemple: les objets de la classe String en java sont immutables

```
String maChaine = "Bonjour";  
maChaine = "Salut";
```



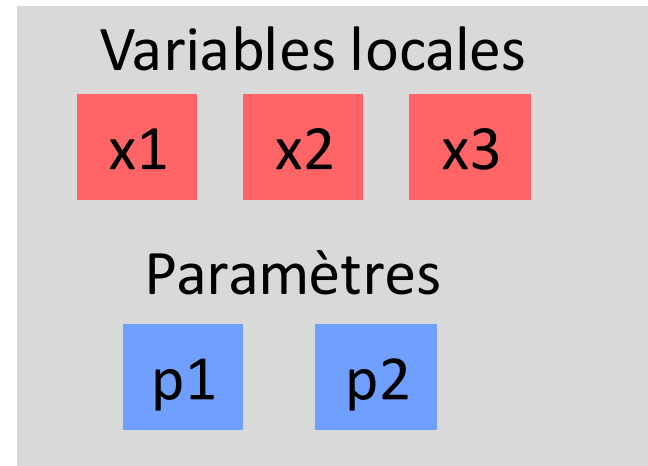
Les concaténations sont couteuses!!
Utiliser de préférence la classe StringBuffer pour
créer des strings mutables

Principes de transmission des paramètres à une méthode



Lorsque une méthode est invoquée:


1. Une zone mémoire est allouée (empilée) pour
 - Ses variables locales
 - Ses paramètres
2. Ses paramètres sont initialisés en fonction des paramètres effectifs utilisés dans l'appel
3. La méthode s'exécute

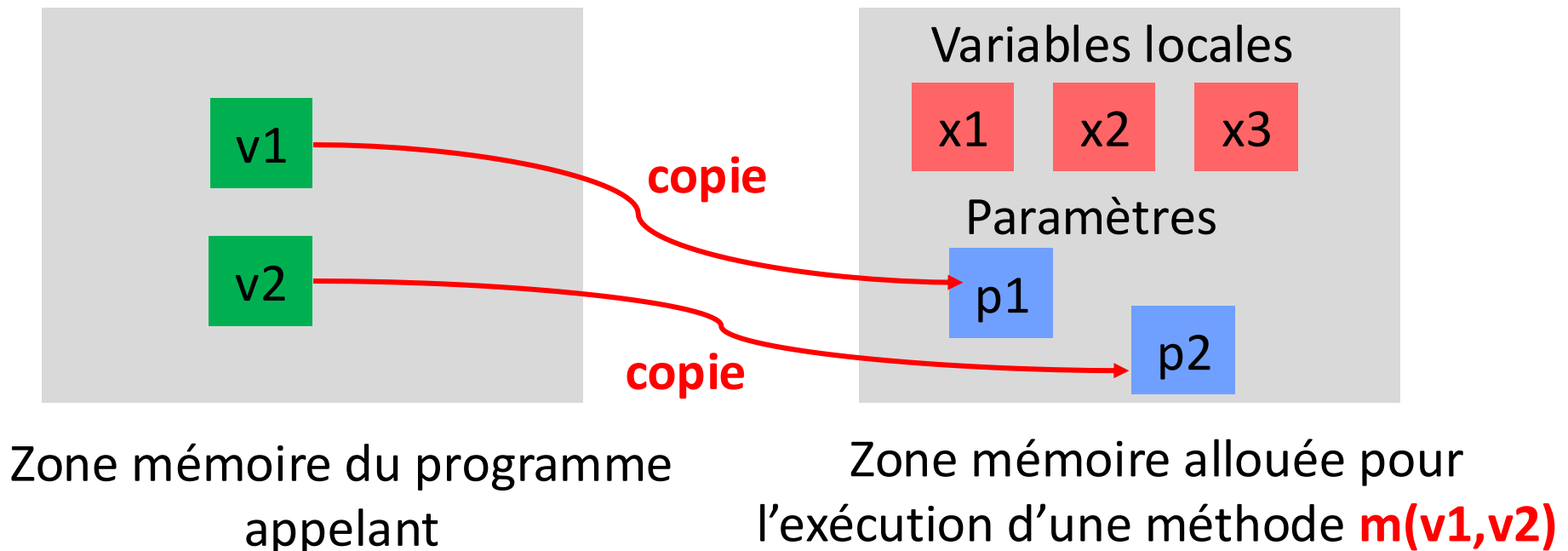


Zone mémoire allouée pour l'exécution d'une méthode $m(v1, v2)$

Mise en place de la Transmission des paramètres

Principes de transmission des paramètres à une méthode

- En java, la transmission se fait « **par valeur** » 
- Les paramètres effectifs (utilisés dans l'appel) sont copiés dans les paramètres de la zone mémoire de la méthode



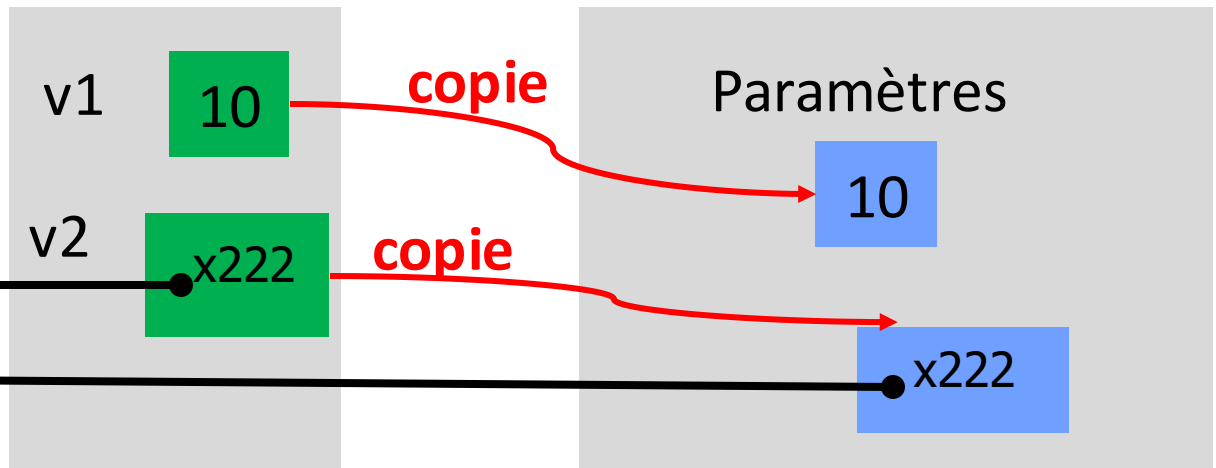
Principes de transmission des paramètres à une méthode



- Si le paramètre est d'un type primitif
 - C'est la valeur qui est copiée
- Si le paramètre est une référence à un objet
 - C'est la référence qui est copiée
 - Attention ! ce n'est pas une copie de l'objet !

Tas: zone mémoire
de stockage des
objets

```
Rectangle  
larg = 4  
long = 2  
area()  
perimeter()
```



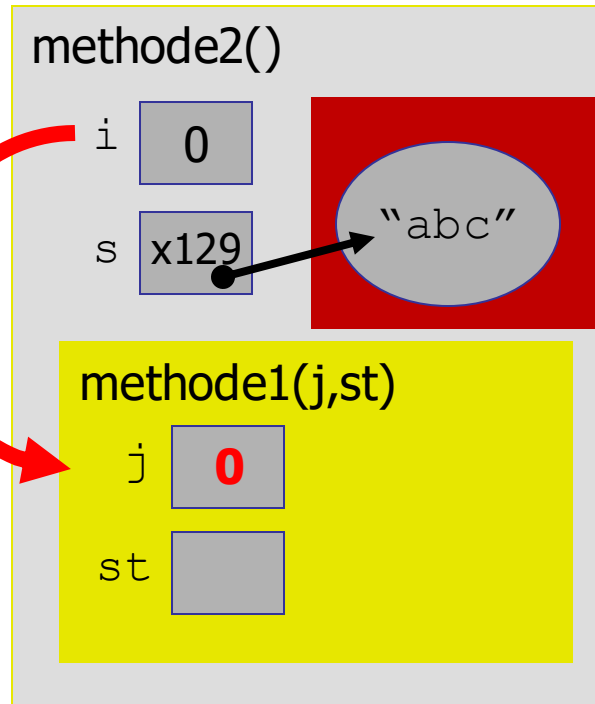


Méthodes en Java

Passage de paramètres

```
public class Essai {  
    void methode1(int j, StringBuffer st) {  
        j++;  
        st.append("d");  
        st = null;  
    }  
    void methode2() {  
        int i = 0;  
        StringBuffer s = new StringBuffer("abc")  
        → methode1(i,s);  
        Sytem.out.println ("i="+i+",s="+s);  
    }  
}
```

Copie



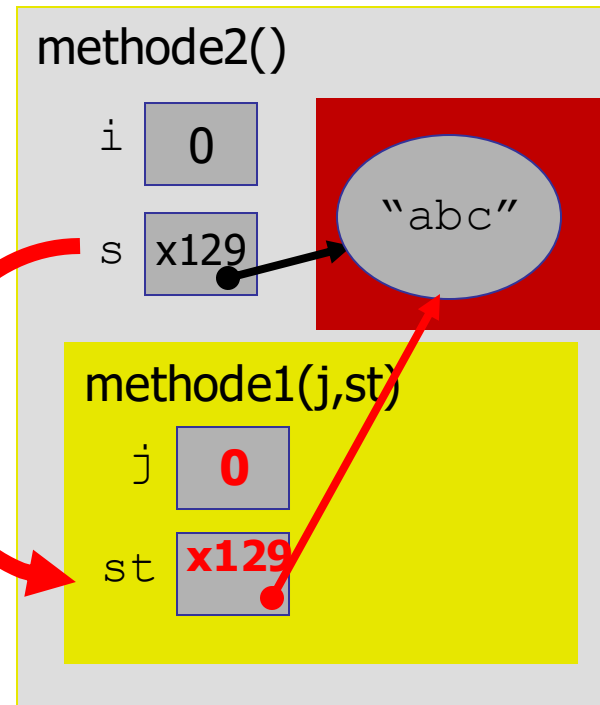


Méthodes en Java

Passage de paramètres

```
public class Essai {  
    void methode1(int j, StringBuffer st) {  
        j++;  
        st.append("d");  
        st = null;  
    }  
    void methode2() {  
        int i = 0;  
        StringBuffer s = new StringBuffer("abc")  
        → methode1(i,s);  
        Sytem.out.println ("i="+i+",s="+s);  
    }  
}
```

Copie

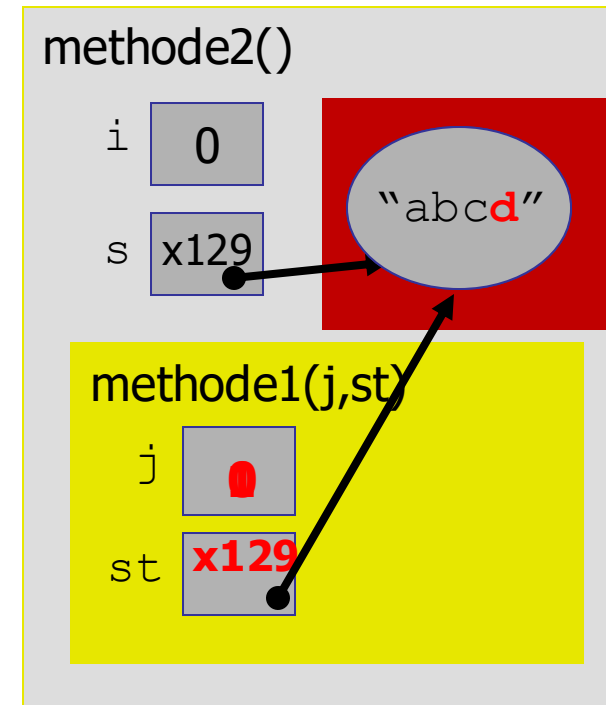




Méthodes en Java

Passage de paramètres

```
public class Essai {  
    void methode1(int j, StringBuffer st) {  
        → j++;  
        → st.append("d");  
        → st = null;  
    }  
    void methode2() {  
        int i = 0;  
        StringBuffer s = new StringBuffer("abc");  
        → methode1(i,s);  
        Sytem.out.println ("i="+i+",s="+s);  
    }  
}
```





Méthodes en Java

Passage de paramètres

```
public class Essai {  
    void methode1(int j, StringBuffer st) {  
        j++;  
        st.append("d");  
        → st = null;  
    }  
    void methode2() {  
        int i = 0;  
        StringBuffer s = new StringBuffer("abc");  
        → methode1(i,s);  
        → System.out.println ("i="+i+",s="+s);  
    }  
}
```

Affichage de i=0 ,s=abcd

methode2()

i

0

s

x129

"abcd"

methode1(j,st)

j

1

st

null