

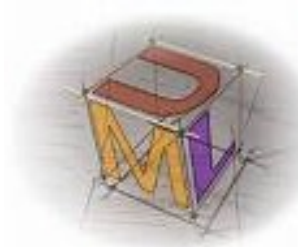
UNIVERSITE DE CORSE

2023-2024

Licence SPI 2ème année
parcours Informatique

UE Programmation Orientée Objet

CH 2 – Objets et Classes

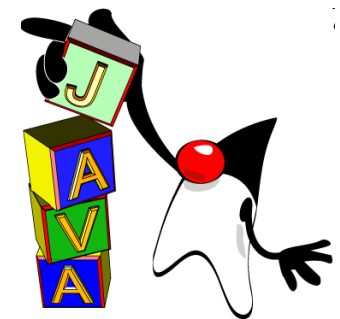


Evelyne VITTORI

vittori_e@univ-corse.fr

Paul PINA-GHERARDI

PINA-GHERARDI_p@univ-corse.fr



Programmation Orientée objet

Plan du Cours

CH1 – PARADIGME ORIENTE OBJET

 CH2 – OBJETS et CLASSES

CH3 – COMMUNICATION ENTRE OBJETS

CH4 – HERITAGE et POLYMORPHISME

CH2 – Classes et Objets



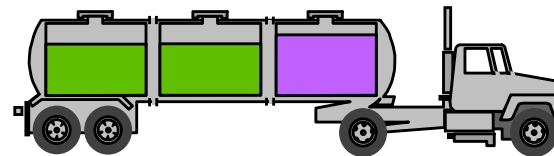
- ▶ Notion intuitive d'Objet
- ▶ Définitions de classe
- ▶ Création d'instances (ou objets)
- ▶ Déclaration et invocation de méthodes
- ▶ Principe d'encapsulation et visibilité
- ▶ Définition de constructeur
- ▶ Attributs et Méthodes de classe
- ▶ Surcharge de constructeurs et de méthodes
- ▶ Spécificités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



Notion intuitive d'objet

Qu'est-ce qu'un objet?

Entité physique



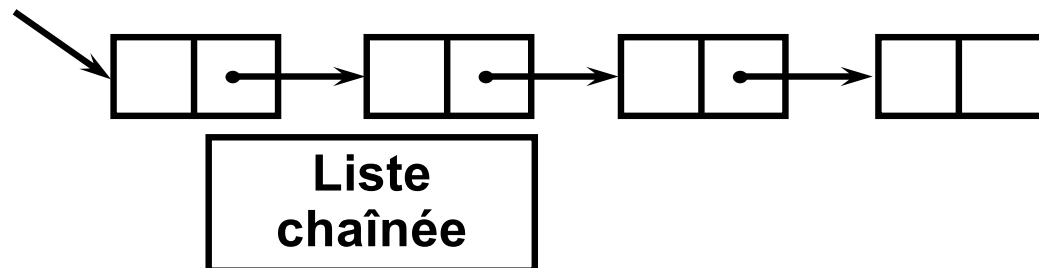
Camion

Entité conceptuelle



Processus
chimique

Entité logicielle

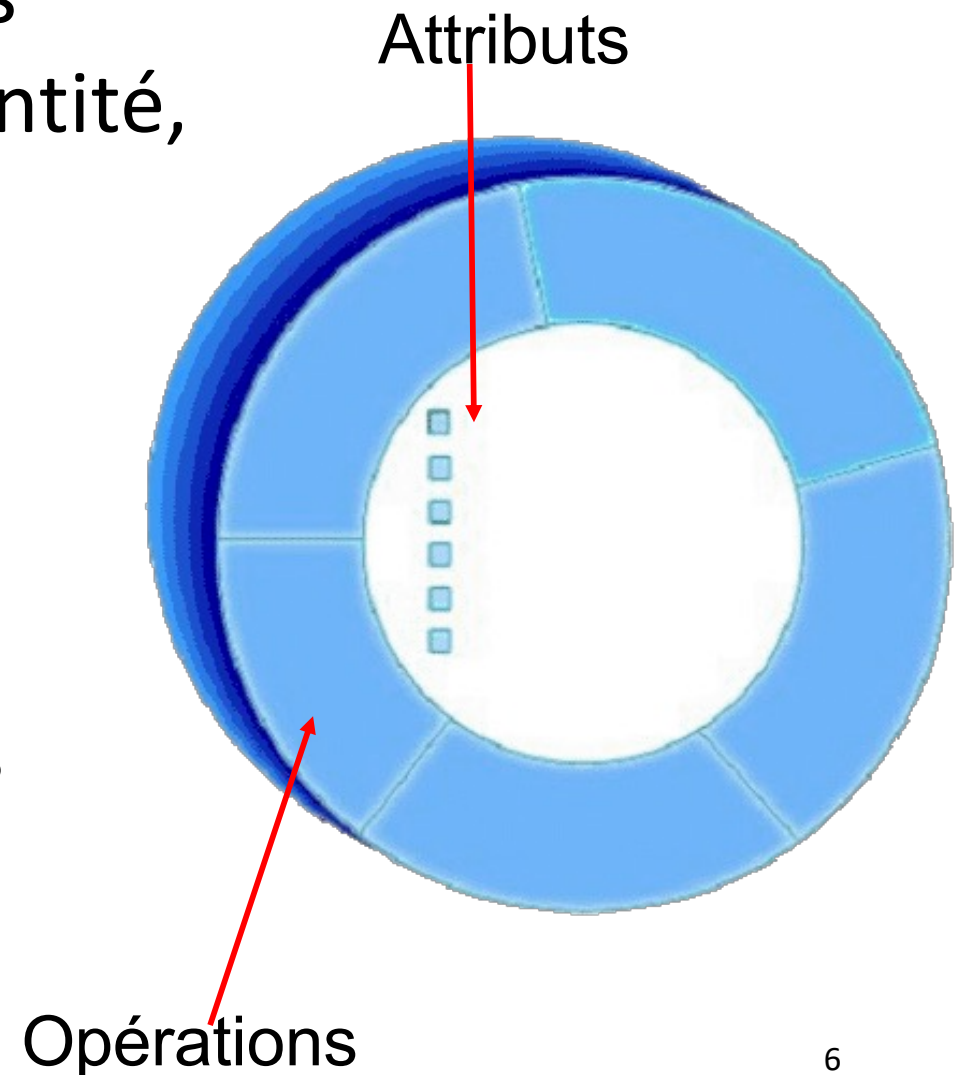


Qu'est-ce qu'un objet?

Objet = entité aux frontières précises qui possède une identité, un état et un comportement

Etat = attributs + *liens*

Comportement = opérations



Etat d'un objet

L'état d'un objet change normalement avec le temps.



Nom: Nimbus
Age: 30
Discipline: Physique
Grade: MCF Ech.6
Statut: Actif
NbCoursMax:4



Professeur Nimbus

Comportement d'un objet

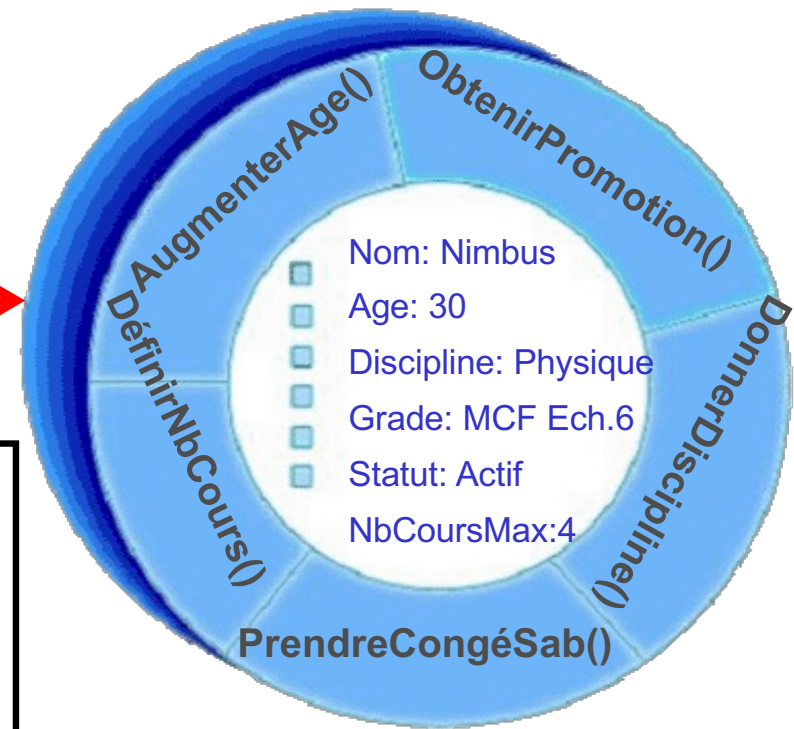
Détermine comment un objet agit et réagit.

- Ensemble des messages auxquels l'objet peut répondre (opérations que l'objet peut réaliser).



Comportement du professeur Nimbus

- augmenterAge
- définirNbCours
- prendreCongéSab
- donnerDiscipline
- obtenirPromotion

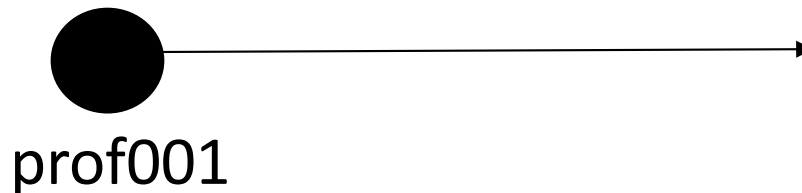


Professeur Nimbus



Notion d'objet en JAVA

Variable définie sur une classe



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**

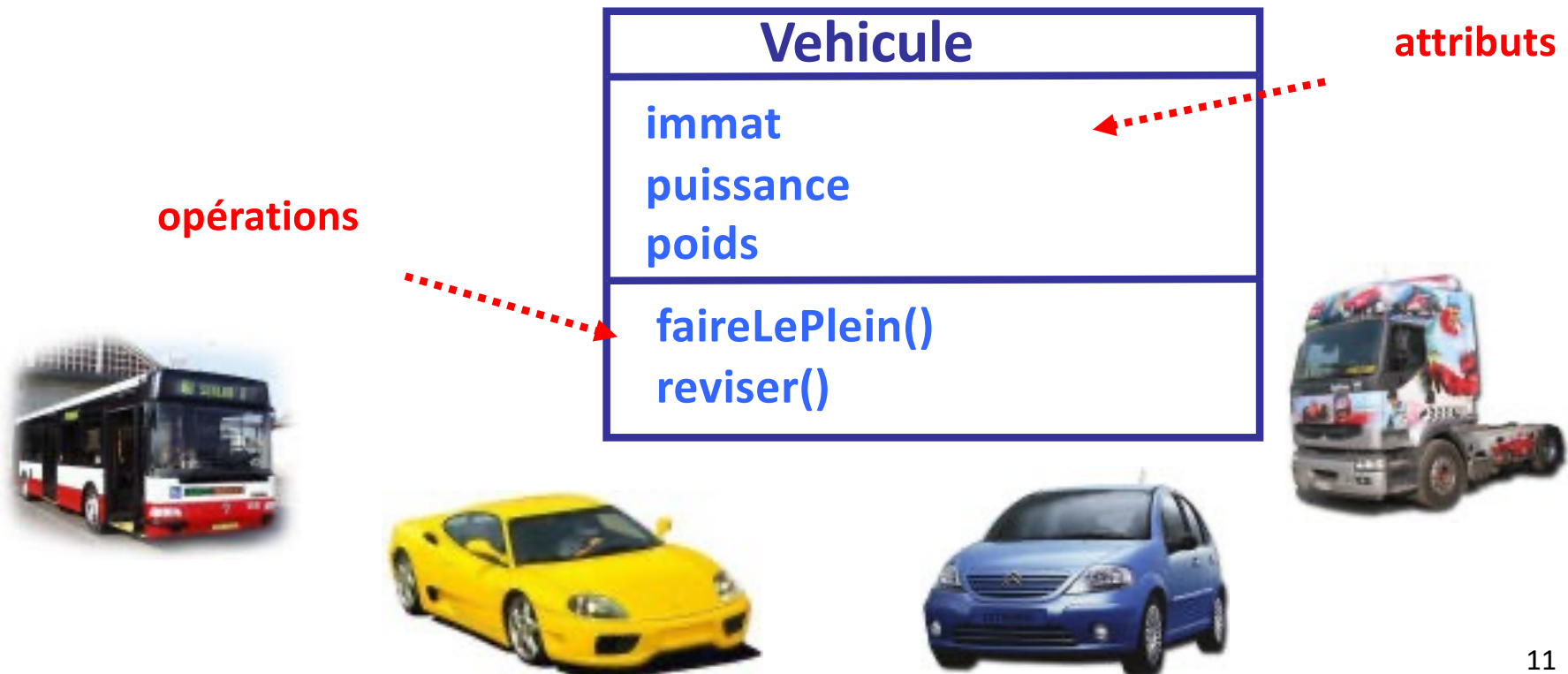


Définition de classes

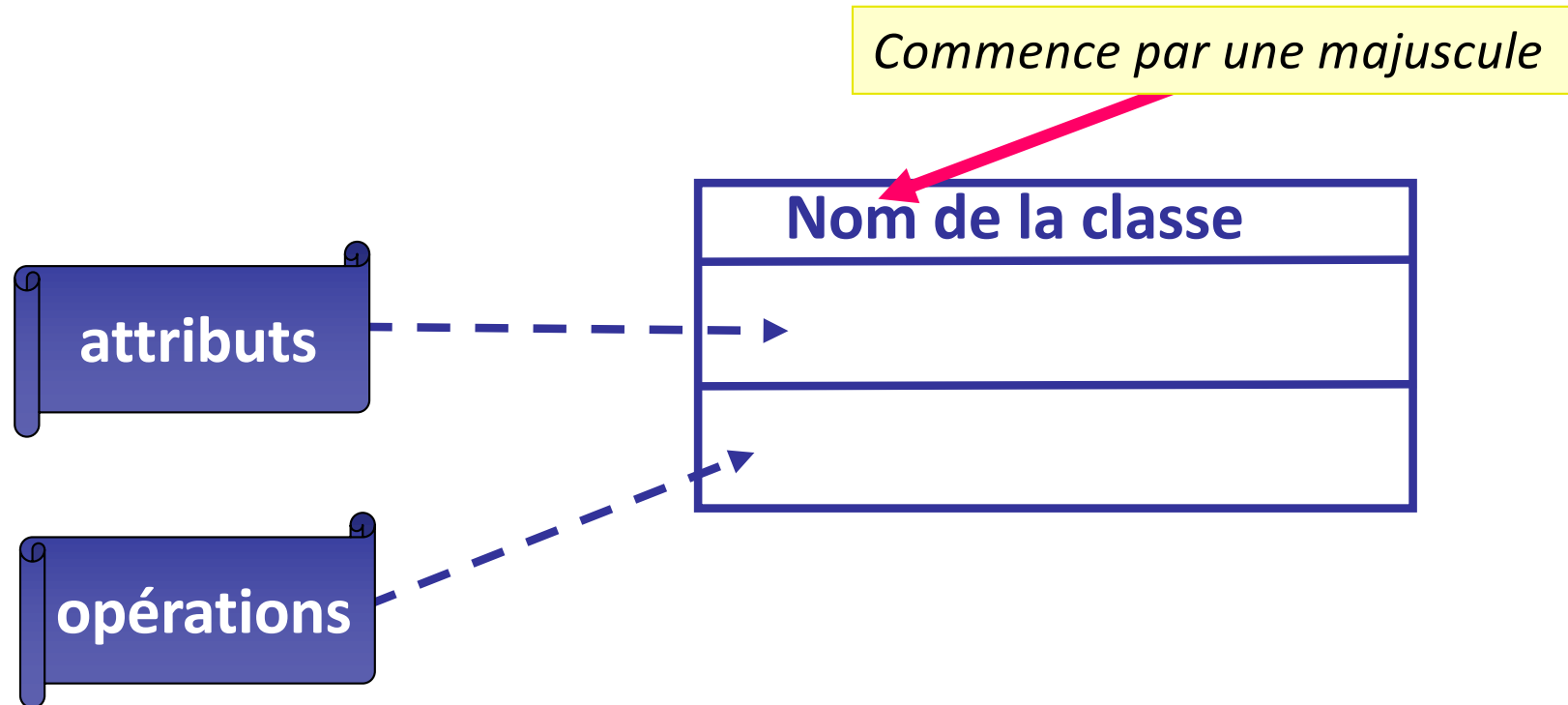
Qu'est-ce qu'une classe?

Une classe est une **abstraction** d'un ensemble d'objets

- Mise en valeur des caractéristiques les plus importantes.
- Suppression des autres caractéristiques.



Représentation des classes en UML



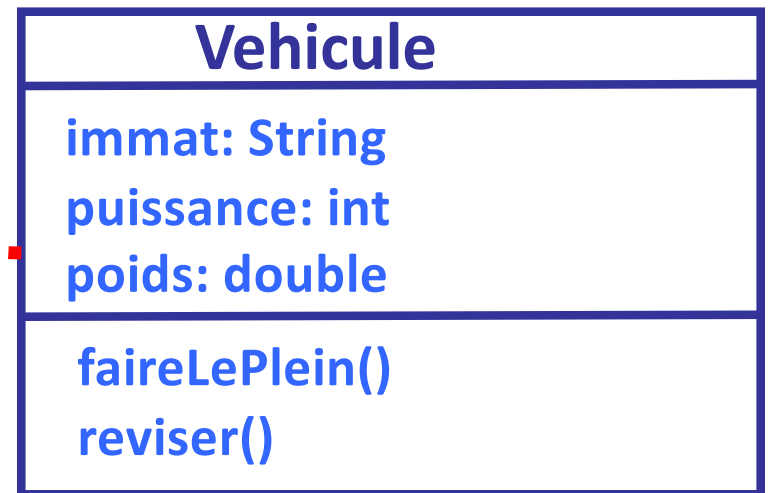
Les **caractéristiques** d'une classe (attributs et opérations) peuvent être définis de manière plus ou moins détaillée.

Classes 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



Syntaxe générale

```
[modificateurs] class NomClasse  
    [extends ClasseMère][implements Interface]  
{  
    [déclaration des attributs]  
    [déclaration de méthodes]  
}
```

abstract	classe abstraite, non instanciable
final	classe non dérivable
public	visible et accessible par tous
default (ou rien)	visible et accessible aux classes du package et aux classes filles

Déclaration de classe en JAVA

Déclaration des attributs

```
[modificateur] type nomVariable ;
```

```
class Vehicule{
```

```
/** l'immatriculation de ce véhicule */
```

```
String immat;
```

```
/** la puissance */
```

```
int puissance;
```

Variable d'instance ou champ
attribut en Java



...

Variable déclarée en dehors de toute méthode

Déclaration de classe en JAVA

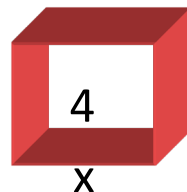
Type des Variables



Types primitifs

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

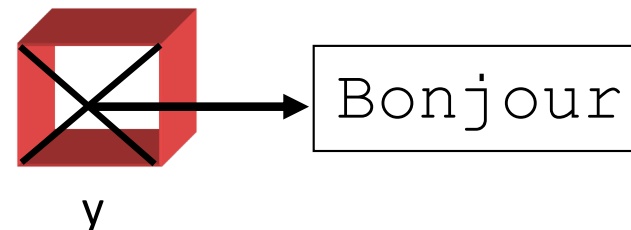
`int x=4`



Types objets (Classes)

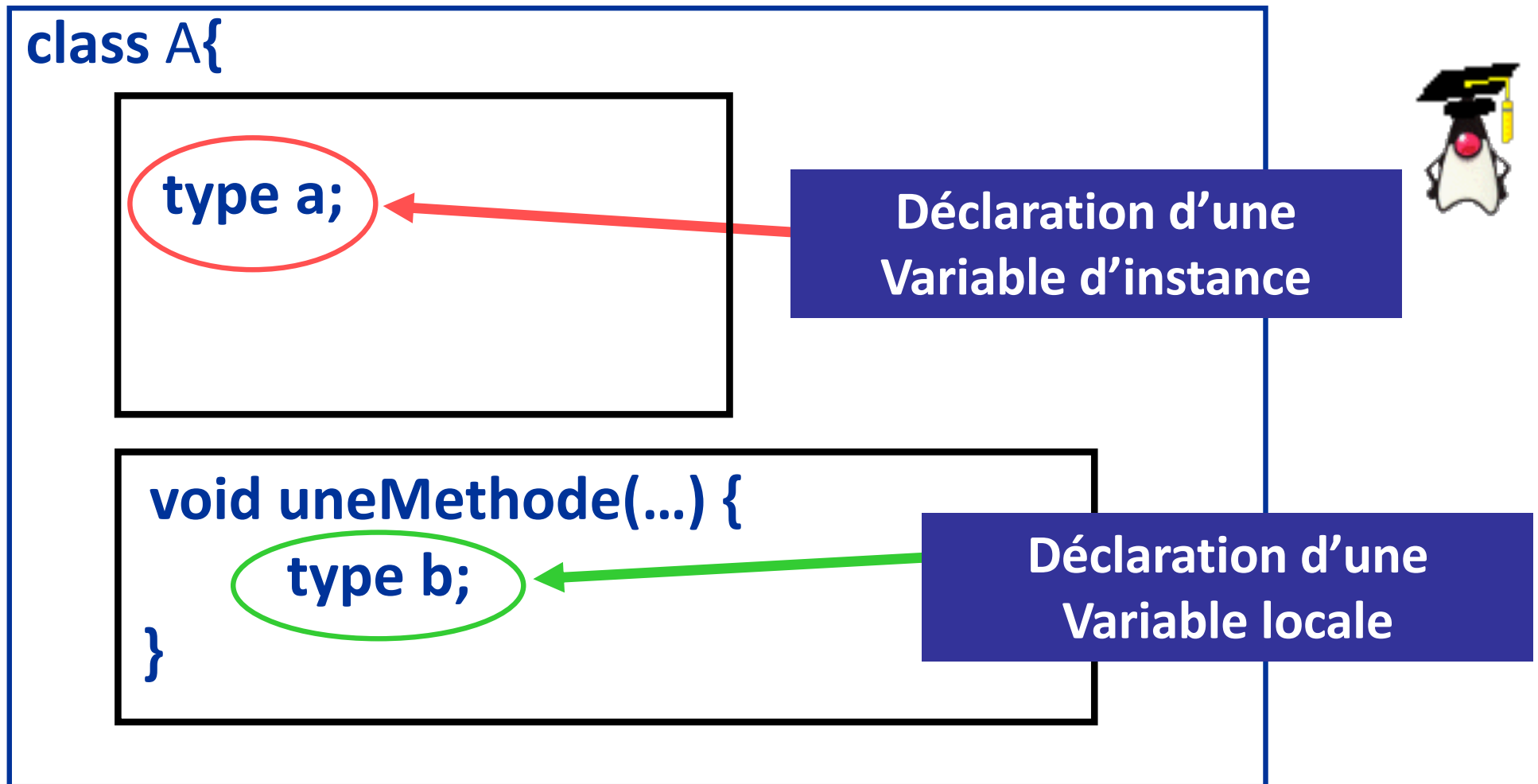
- Classes de l'API java: String, ...
- Classes de l'application (nos propres classes)

`String y="Bonjour"`



Déclaration de classe en JAVA

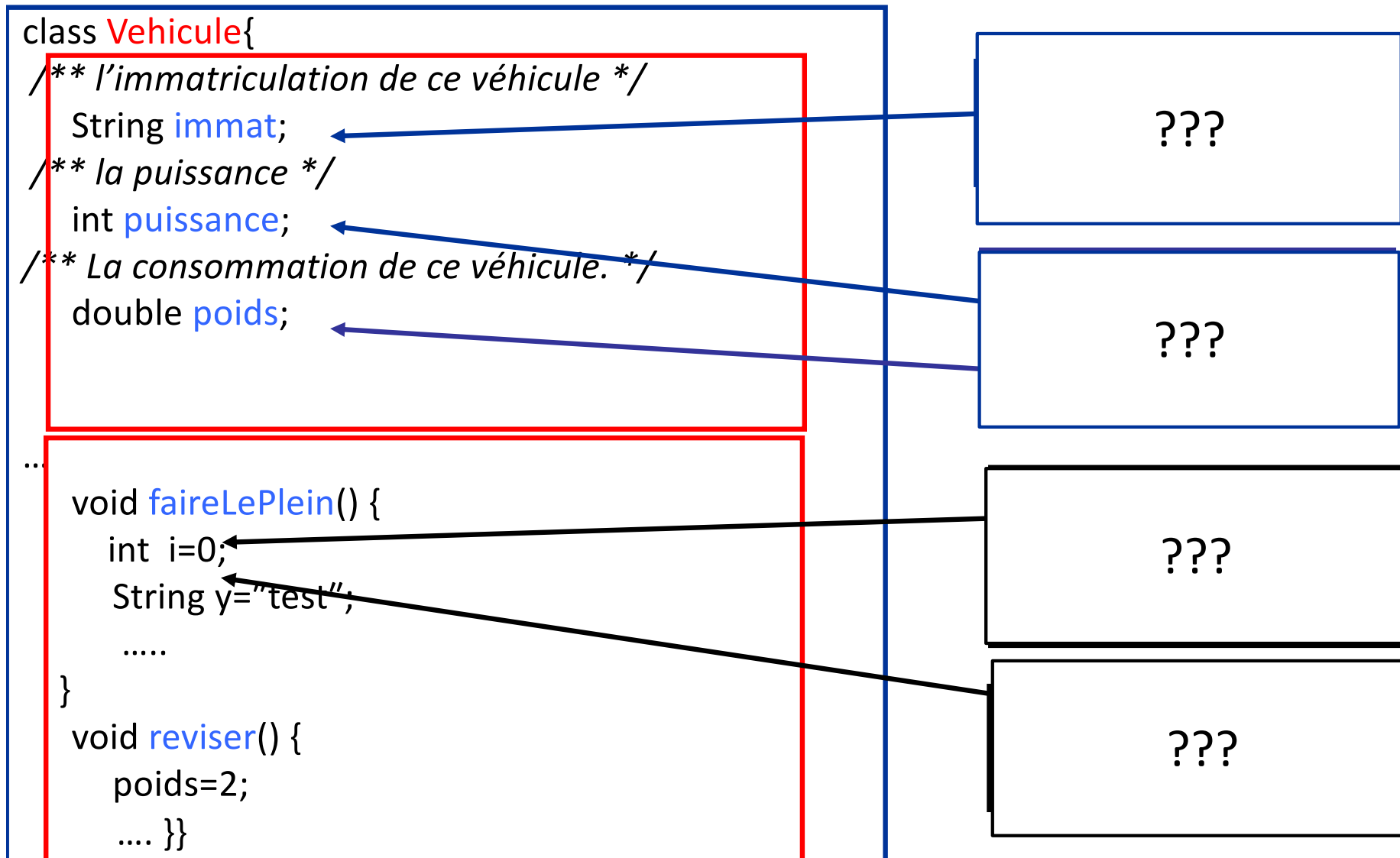
Variables d'instances et Variables locales



Variables et classes en Java

Variable d'instance ou variable locale?

Définie sur un type de base ou sur une classe?




Déclaration de classe en JAVA

Variables d'instances	Variables locales (à une méthode)
Définissent un attribut d'une classe	Variables « utilitaires » classiques
Déclaration en dehors des méthodes	Déclaration dans une méthode
Initialisation automatique aux valeurs par défaut	Initialisation obligatoire avant leur utilisation
Accessibles dans toutes les méthodes de la classe	Accessible uniquement dans la méthode où elle est déclarée

Classes en JAVA et en UML

Terminologie

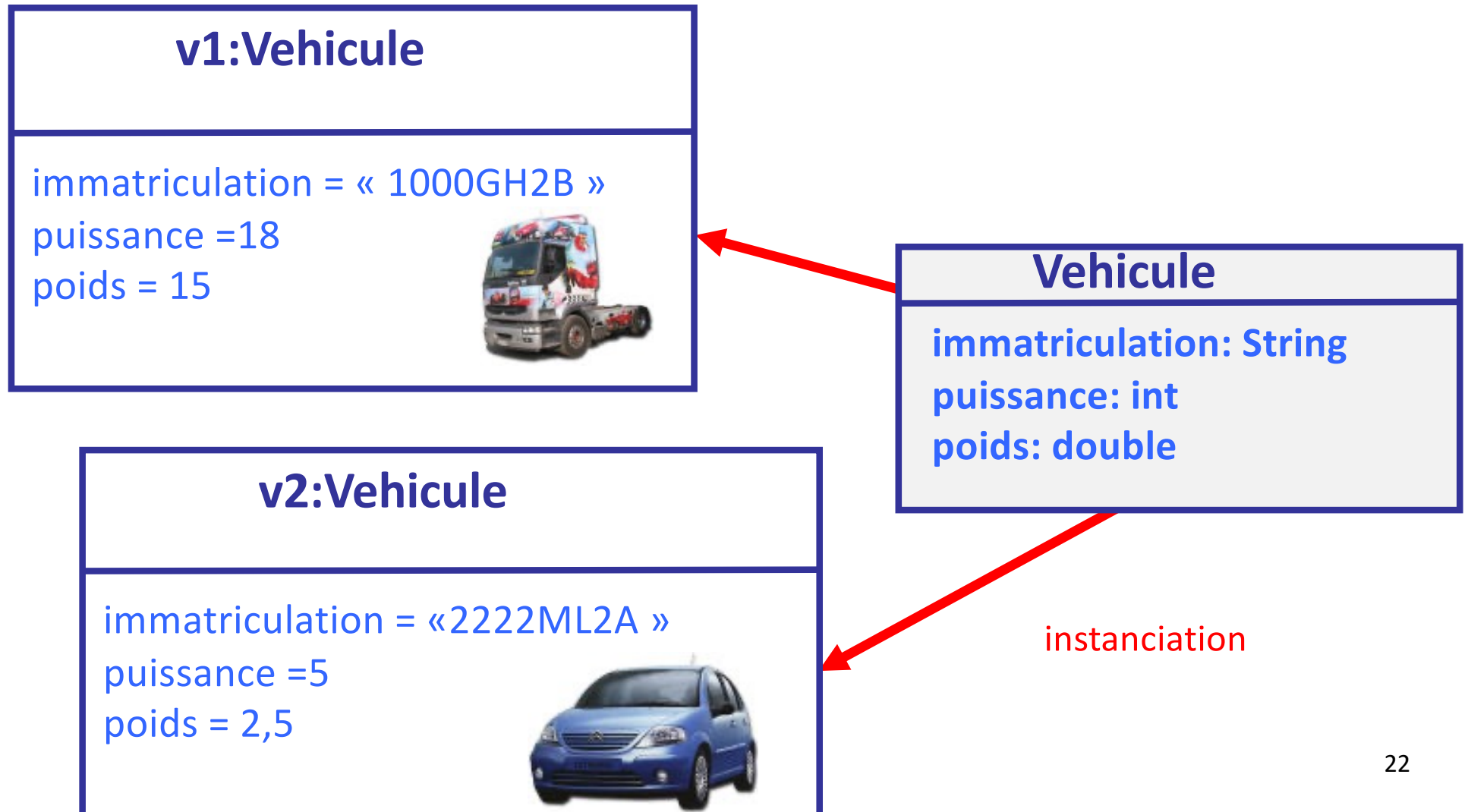
UML	JAVA
<div><div>Opération Attribut (ou propriété)</div><div>Caractéristiques</div></div>	<div><div>Méthode Variable d'instance (ou champ)</div><div>Membres</div></div> 



Création d'instances (ou objets)

Notion d'instance

Un **objet** est une **instance** de classe

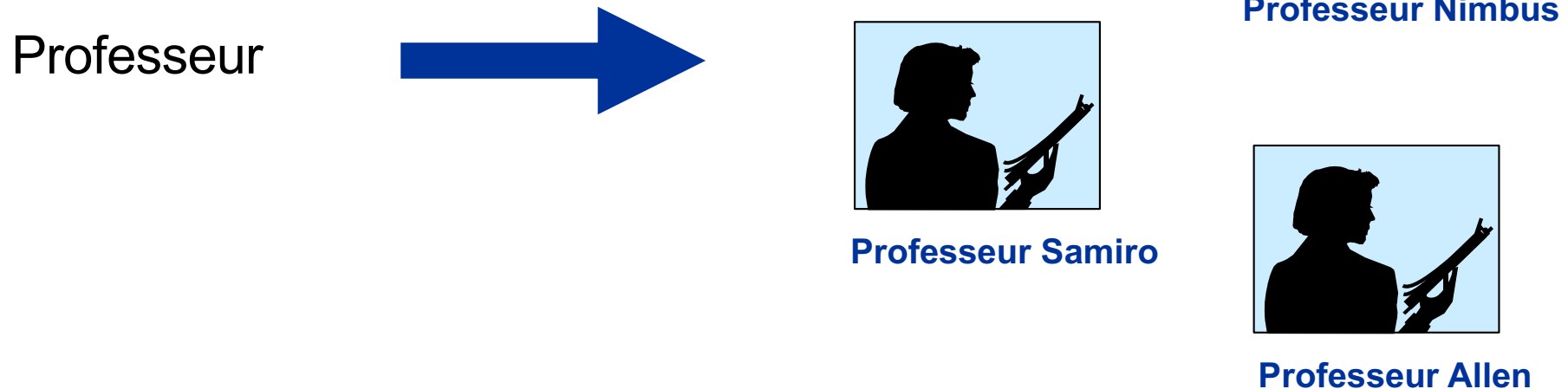


Instances et Classes

Une classe est un **modèle** pour la création de ses instances (objets).

Elle définit leurs caractéristiques communes:

- Structure : **attributs**
- Comportement: **opérations**




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

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 */  
    }
```

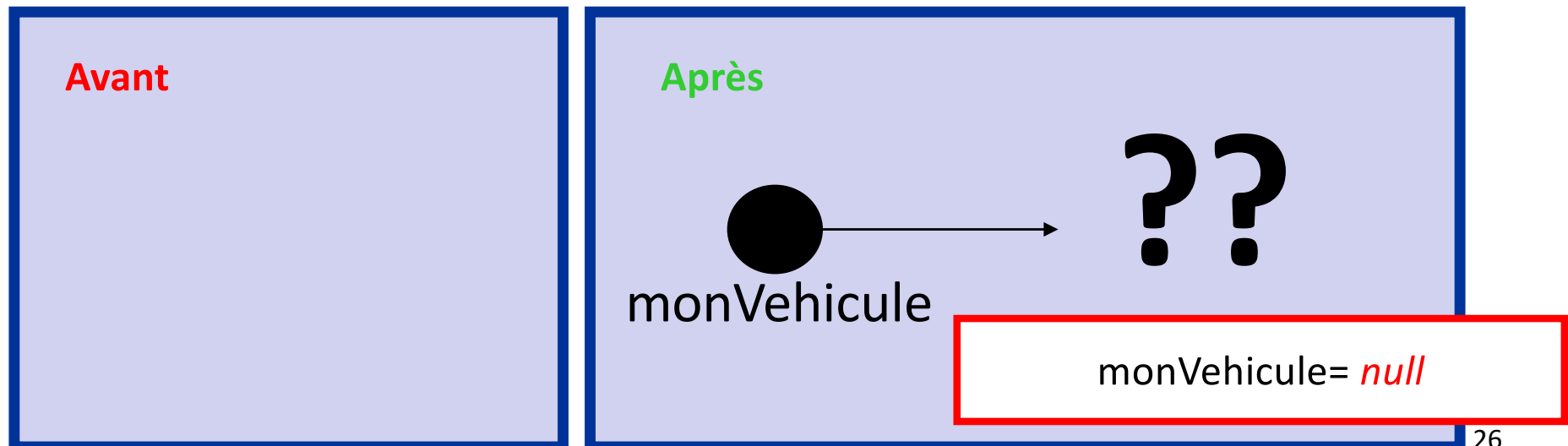
Création d'objets en JAVA

Déclaration d'une variable (Référence)



Vehicule monVehicule;

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



Création d'objets en JAVA

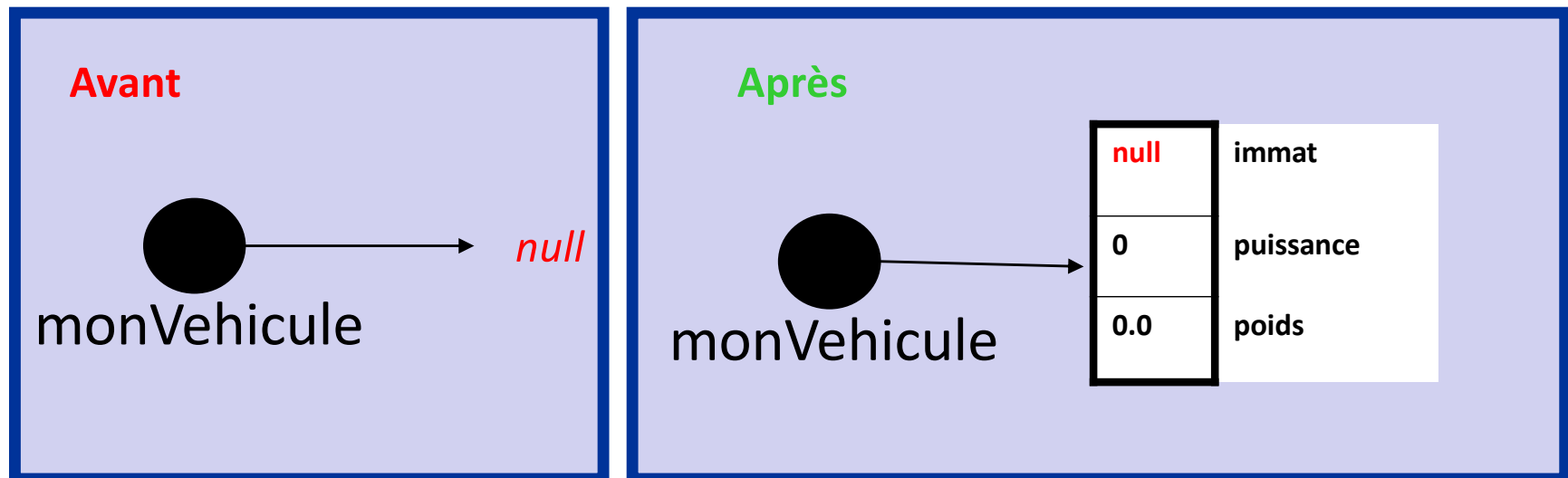


Création de l'objet (instanciation)

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

⇒ réserve la mémoire pour stocker l'objet

⇒ associe l'objet à la référence

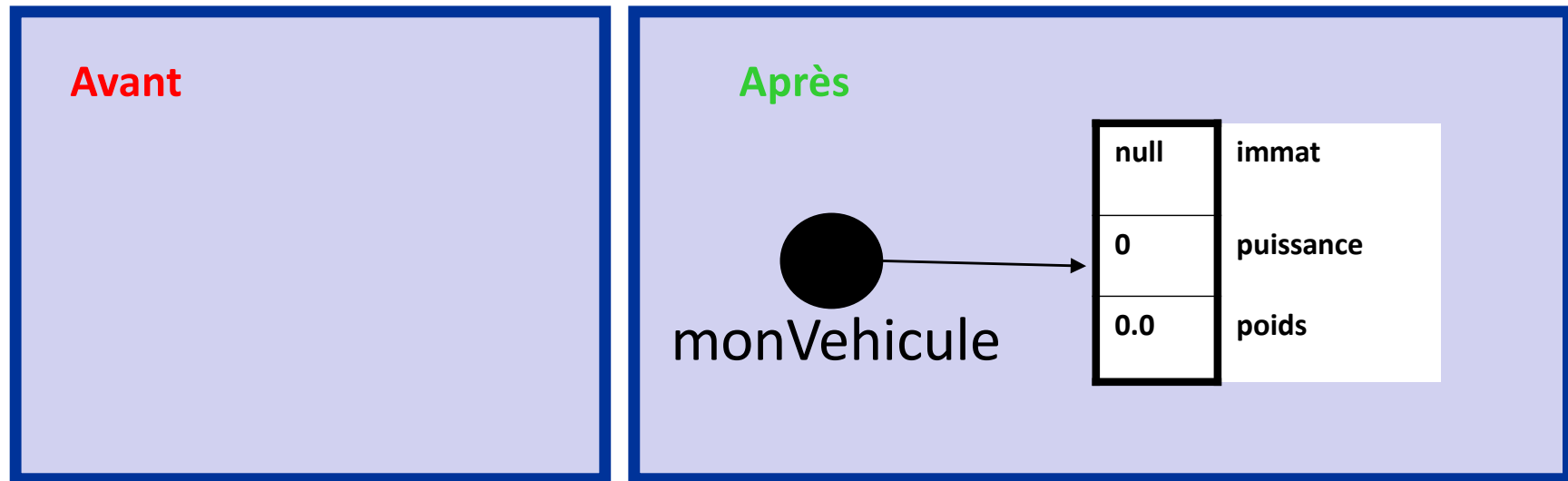


Création d'objets en JAVA



Déclaration et instanciation

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



Accès aux valeurs des attributs



Une classe

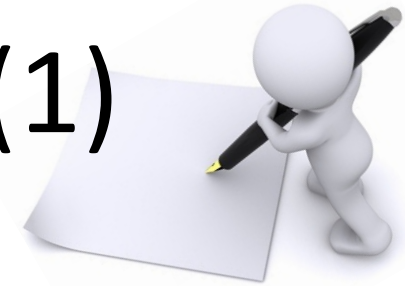
```
class Vehicule{  
    /** l'immatriculation de ce véhicule */  
    String immat;  
    /** La puissance */  
    int puissance;  
    /** Le poids de ce véhicule. */  
    double poids;  
    ... }  
}
```

Une classe de test

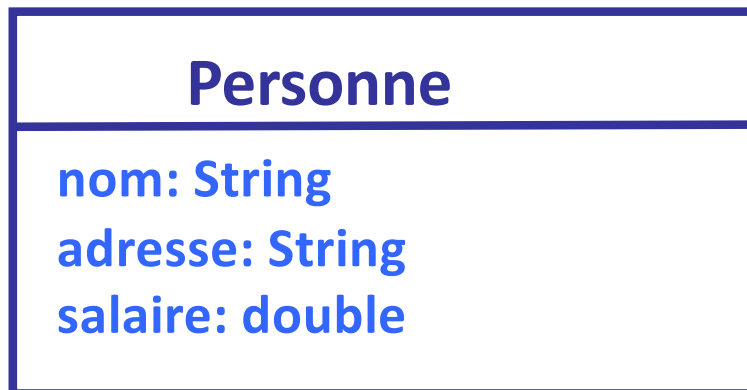
```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule monVehicule=new Vehicule();  
        monVehicule.immat="1000GH2B" ;  
        monVehicule.puissance=18;  
        monVehicule.poids=15.5;  
        System.out.println( "Immatriculation : " +  
            monVehicule.immat);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



TPCours - Exercice (1)



- Définir un package `exCH2_1` dans votre projet TPCours.
- Définir en Java une classe **Personne** correspondant à la représentation UML :
- Définir en Java une classe **TestPersonne** contenant une méthode `main()` qui définit une personne ayant pour nom *Titi*, habitant *Corté* et ayant un salaire de *2000* euros, et affiche son nom et son adresse sous la forme suivante:



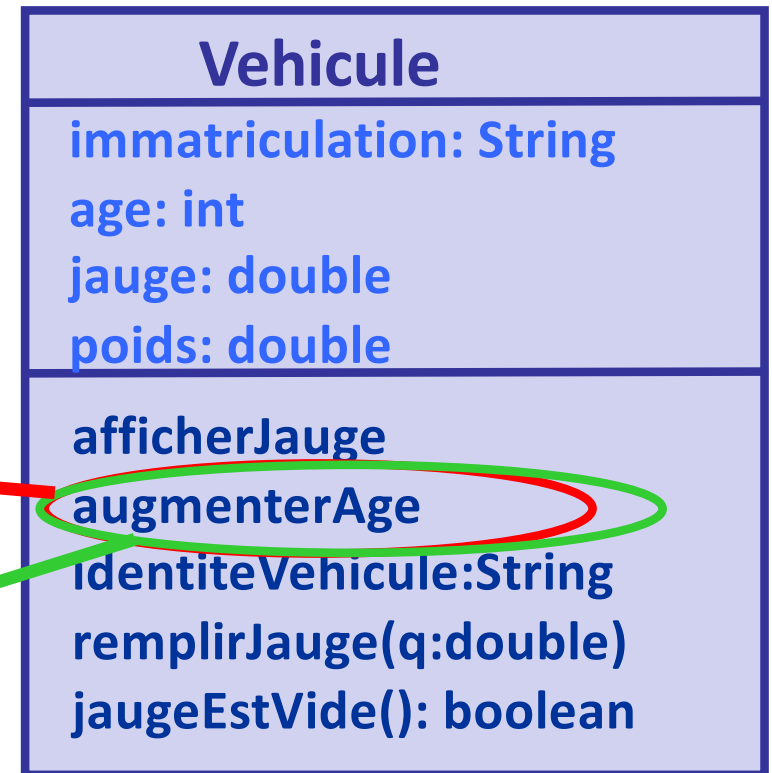
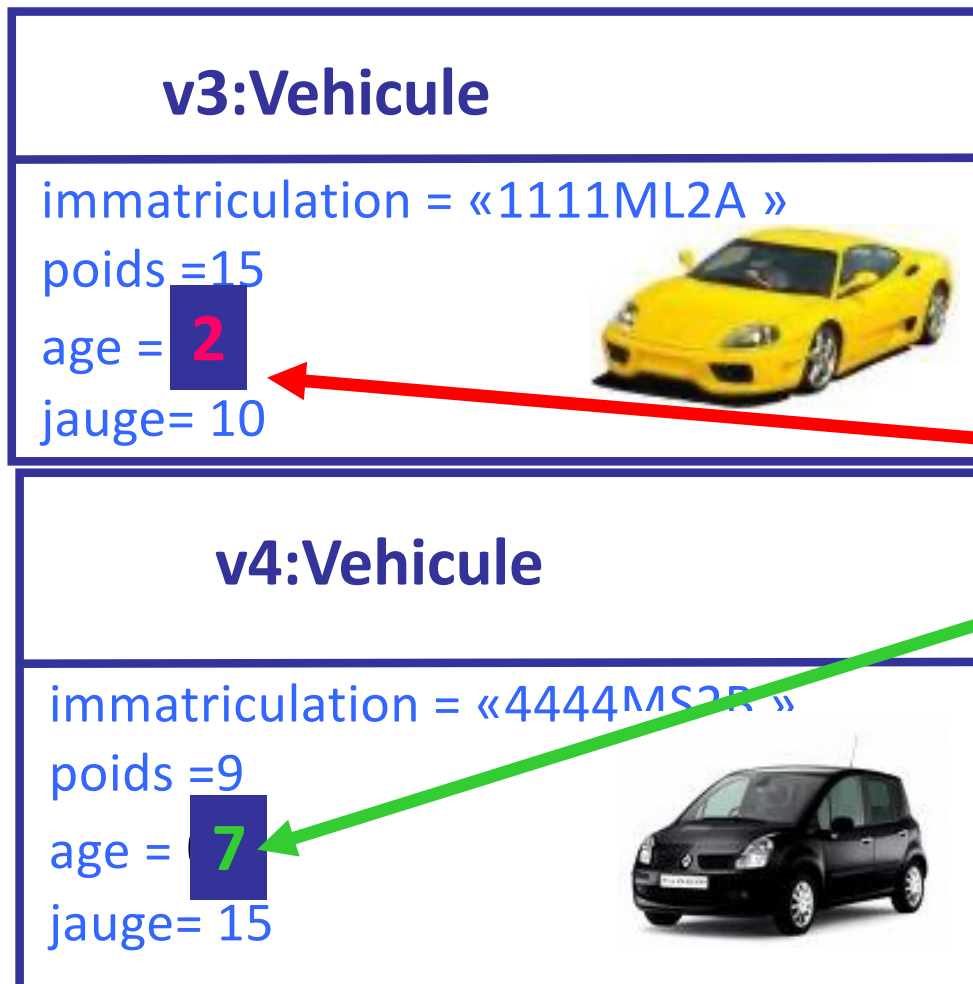
Nom= Titi
Adresse = Corté



Déclaration et invocation de méthodes

Manipulation d'objets: méthodes

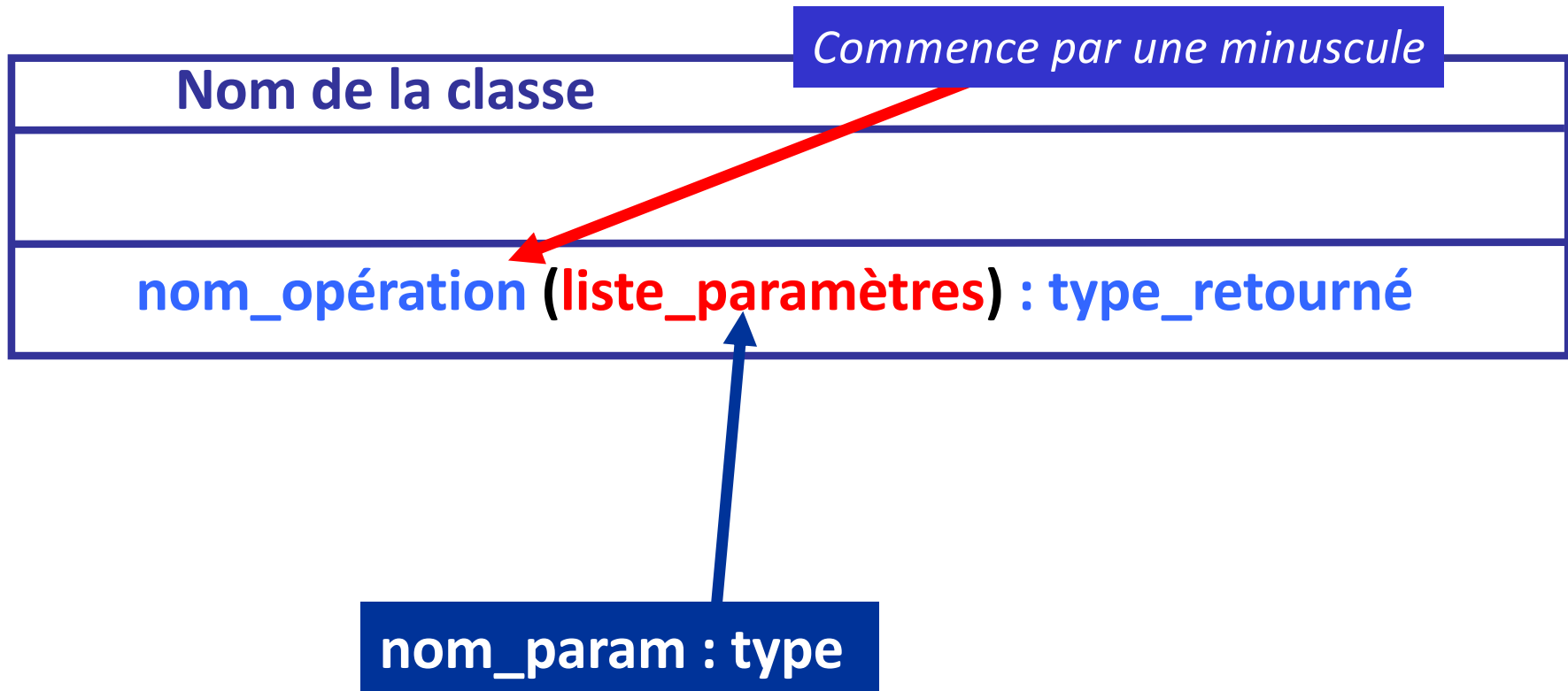
- Une **opération (méthode)** est attachée à une classe.
- L'exécution (invocation) d'une opération porte sur **une instance** particulière.



- Déclenchement par envoi d'un message à une instance particulière

Manipulation d'Objets

Opérations en UML



Déclaration de méthodes en Java



- Comprendre et manipuler l'état d'un objet
- Contrôler les accès aux champs des objets
- Forme générale

Signature

```
[modificateur] type nomMéthode(typeArg arg,... {  
    // variables locales et instructions  
}
```

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 sans résultat (void)



`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();  
    }  
}
```

Vérification de compatibilité entre invocation et déclaration

- Nombre de paramètres effectifs identique au nombre de paramètres formels
- Types deux à deux compatibles:
 - chaque paramètre effectif doit avoir un type compatible avec le type du paramètre formel qui lui correspond

Déclaration

```
Class A {  
    void nomMethode (type1 f1, type2 f2, type3 f3)  
    ...  
}
```

Invocations

```
// méthode appelante (main par ex)  
type1 p1, x ;  
type2 p2, y ;  
type3 p3, z ;  
//...  
A uneVarA=new A()  
uneVarA.nomMethode ( p1, p2, p3)  
...  
uneVarA.nomMethode ( x, y, z)
```

Un paramètre effectif peut-être une variable, une constante ou une expression

Déclarations et invocations de méthodes renvoyant un résultat



```
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; ....  
        String res= " description: " + v1.identiteVehicule();  
        System.out.println(v1.identiteVehicule());  
    }  
}
```

Résultat d'une méthode

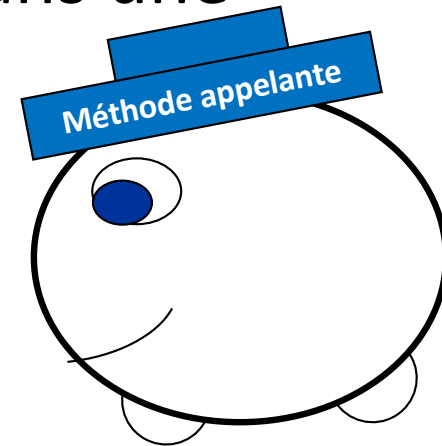
- Comme une fonction classique, une méthode peut renvoyer un **résultat**
- Le résultat de la méthode peut être d'un type quelconque: int, double, boolean, String, Voiture...
- Le corps d'une méthode renvoyant un résultat comporte au moins une instruction **return** suivi d'une expression conforme au type du résultat de la méthode

```
typeResultat nomMethode (...) {  
    instructions  
    return expression  
}
```

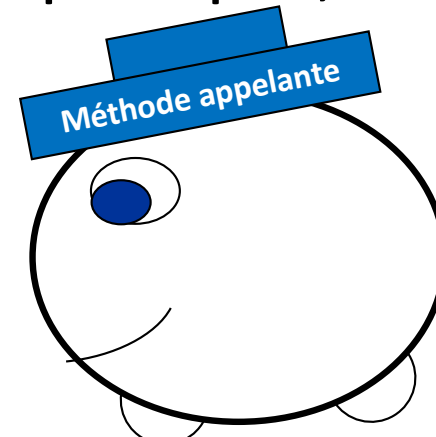
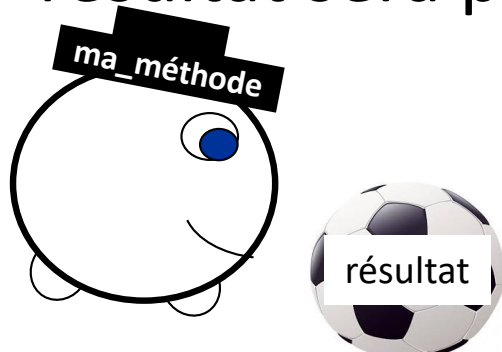
Expression. de type
typeResultat

Invocation d'une méthode renvoyant un résultat

- Si une méthode renvoie un résultat, la méthode qui l'appelle **doit récupérer** ce résultat dans une expression



- Si la méthode appelante ne le récupère pas, le résultat sera perdu!!



Invocation d'une méthode renvoyant un résultat

- L'invocation de la méthode doit apparaître dans une **expression** conformément au type de son résultat.

nomMethode, méthodes une classe A
uneVarA: variable de type A

- Exemples:

Partie droite d'une affectation

```
x = uneVarA.nomMethode()
```

#x de même type que le résultat de la méthode

Affichage

```
print(uneVarA.nomMethode())
```

Condition

```
if (uneVarA.nomMethode() == x)
```

Déclarations et invocations de méthodes



```
class Vehicule {
```

```
....
```

```
boolean jaugeEstVide ()
```

```
{
```

```
    if (jauge==0)
```

```
        return true;;
```

```
}
```

```
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();
```

```
    }
```



TPCours - Exercice (2)



- Compléter en Java la classe `Personne` de l'exercice (1) par la définition des méthodes conformément à la représentation UML suivante:

Personne
<code>nom: String</code> <code>adresse: String</code> <code>salaire: double</code>
<code>afficher ()</code> <code>changerAdresse(nouvelle:String)</code> <code>salaireAnnuel(): double</code> <code>salaireEstSup2000():boolean</code>

- Compléter la méthode `main` de la classe `TestPersonne` afin d'afficher le nom et adresse de Titi (*invocation de la méthode `afficher`*), d'enregistrer son déménagement vers Ajaccio (*invocation de la méthode `changerAdresse`*), d'afficher son salaire annuel ainsi qu'un message indiquant si son salaire mensuel est supérieur à 2000 euros.

Affichage Ecran

Titi habite Corte
Titi habite Ajaccio
Salaire annuel de Titi: 24000 euros
Titi a un salaire supérieur à 2000



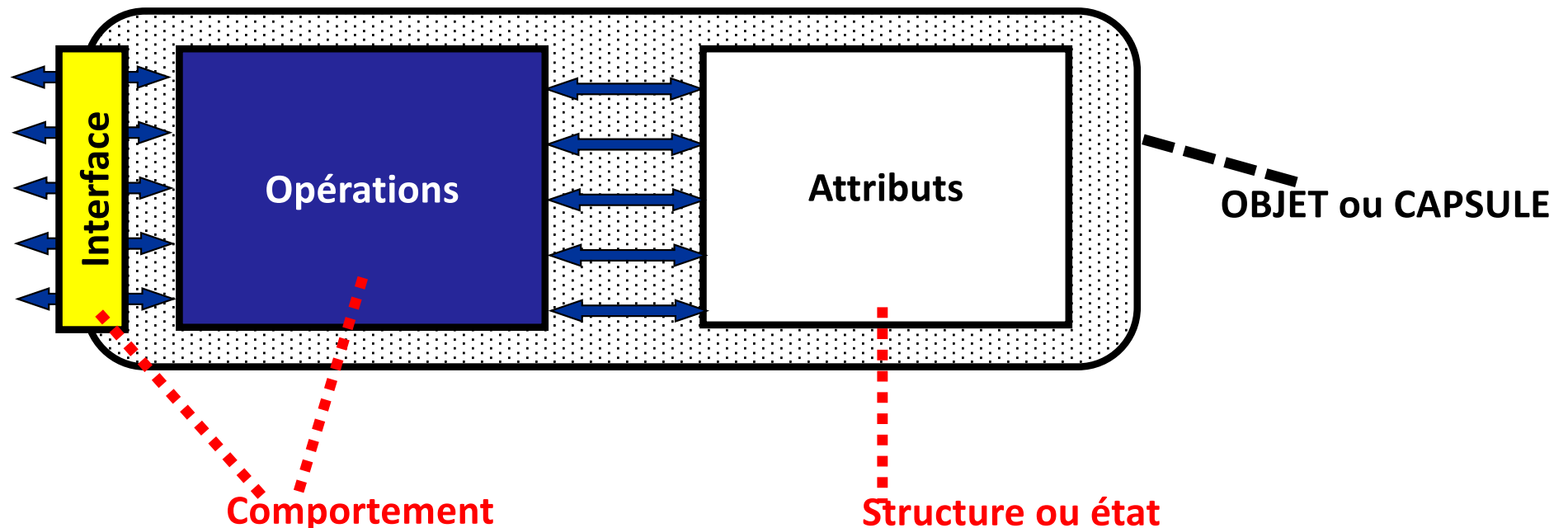
Principe d'encapsulation et visibilités

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

Encapsulation

Principe = séparation spécification/réalisation

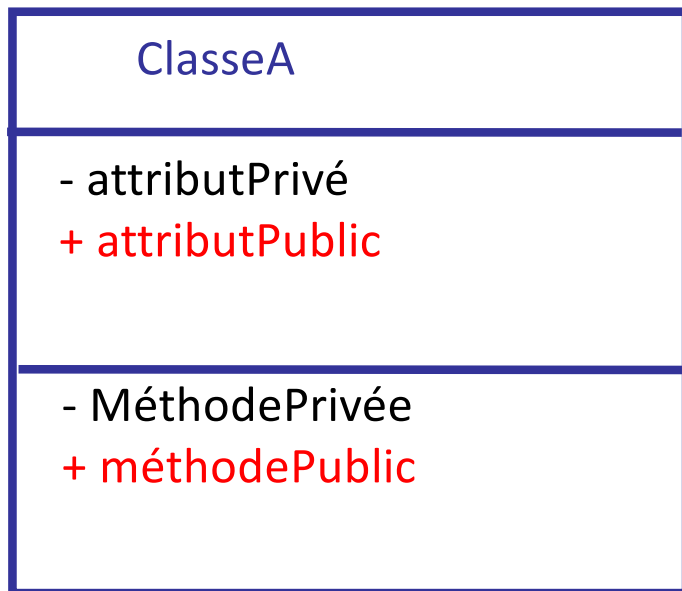
- Les objets ne sont manipulés qu'à travers leur interface
- Les détails de l'implémentation sont occultés



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 »



Mise en oeuvre du principe d'encapsulation

Getter et Setter

```
public class Vehicule {  
    private String immat;  
    private short puissance;  
    private double jauge;  
  
    public void setImmat(String i){  
        immat=i;  
    }  
    public String getImmat(){  
        return immat;  
    }  
}
```

```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule v1=new Vehicule();  
        //v1.immat="1000GH2B" ;  
        v1.setImmat("1000GH2B" );  
        //System.out.println("Imatt = " + v1.immat);  
        System.out.println("Imatt = " +  
        v1.getImmat());  
    }  
}
```

Pourquoi l'encapsulation?

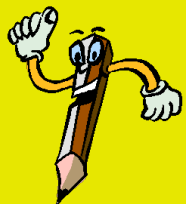


- Pour sécuriser le code!
- Certaines classes sont développées par d'autres programmeurs (vos fournisseurs)
 - Ils vous offrent des services (méthodes)
 - Vous êtes de simples utilisateurs
 - vous n'avez pas à connaître la structure de leurs classes (attributs) et les algorithmes de leurs méthodes
 - 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)

Accesseurs et modificateurs



- Méthodes type **get**Nom Champ()
 - Retournent la valeur du champ
- Méthodes void **set**NomChamp(type val)
 - Permettent de modifier la valeur du champ
 - Permettent de paramétrer et de contrôler la modification



Ces méthodes ne sont définies
que si elles sont utiles



TPCours - Exercice (3)



- Modifiez la classe Personne afin de la rendre conforme au principe d'encapsulation des attributs.
- Votre classe TestPersonne est-elle encore correcte? Pourquoi?
- Définissez les accesseurs et modificateurs nécessaires dans la classe Personne.
- Le modificateur de l'attribut salaire ne doit pas autoriser la modification du salaire si le nouveau salaire est inférieur au salaire net minimum (SMIC=1219 euros)
- Modifiez en conséquence votre classe TestPersonne₅₂

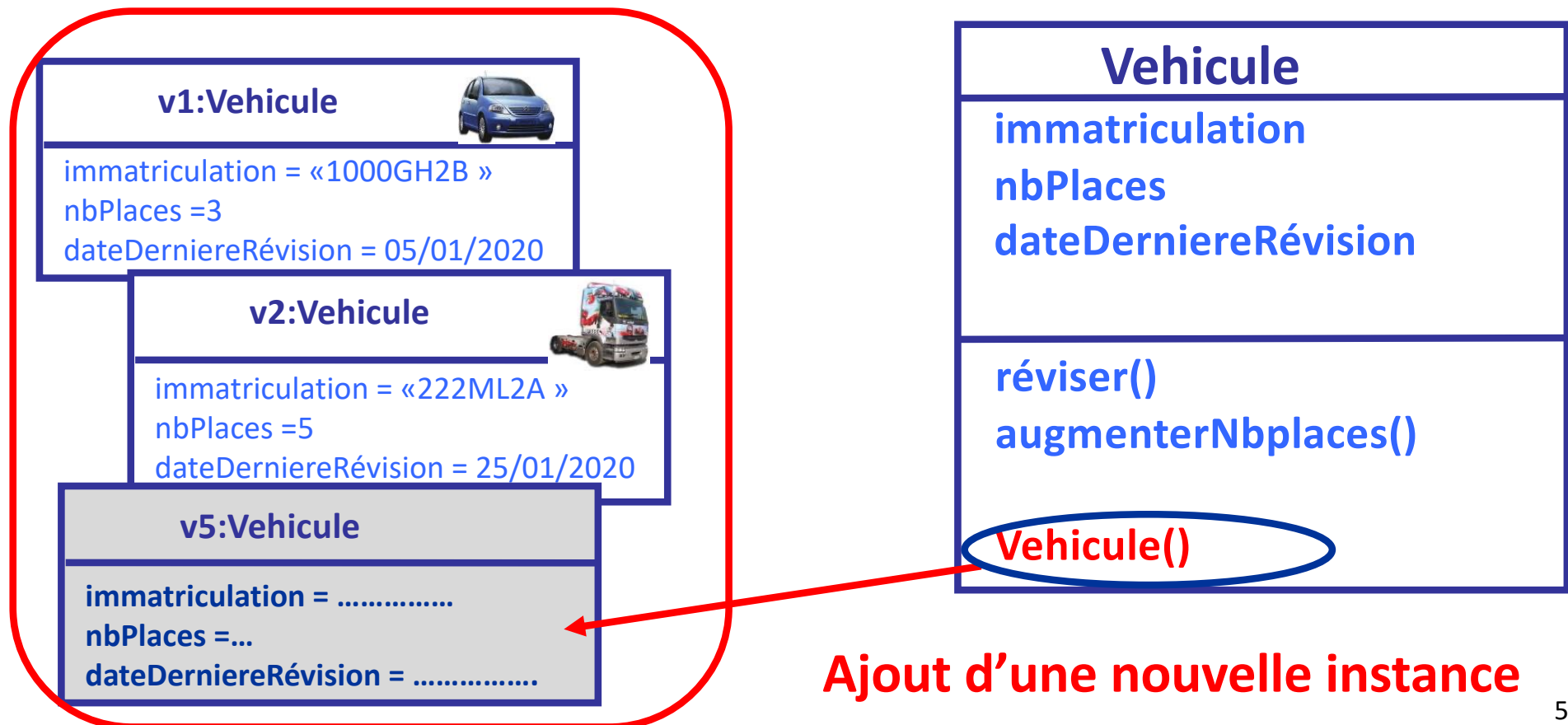


Définition de constructeur

Instanciación y Constructores

Mécanisme d'instanciation =

- Activation de l'opération de création d'instance de la classe: **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



Notion de Constructeur

- Méthode de création d'un objet
- Rôle :
 - **Allouer** les ressources mémoire
 - **Initialiser** les variables d'instances
 - Renvoyer une occurrence de l'objet
- Porte le **même nom** que la classe
- N'a **pas de valeur de retour** (sinon méthode)



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



TPCours - Exercice (4)



- Complétez en java la classe `Personne` de l'exercice (3) par la définition d'un constructeur d'initialisation ayant la signature suivante (en UML):

`Personne(nom:String, adresse: String, salaire:double)`

- Votre classe `TestPersonne` est-elle encore correcte? Pourquoi?

Le constructeur par défaut disparaît lorsque l'on définit un constructeur explicite

- Dans la classe `TestPersonne` (main), remplacez les lignes de création de l'objet `Personne` ayant pour nom *Titi*, habitant *Corté* et ayant un salaire de *2000* euros par une seule ligne d'invocation du constructeur ci-dessus.



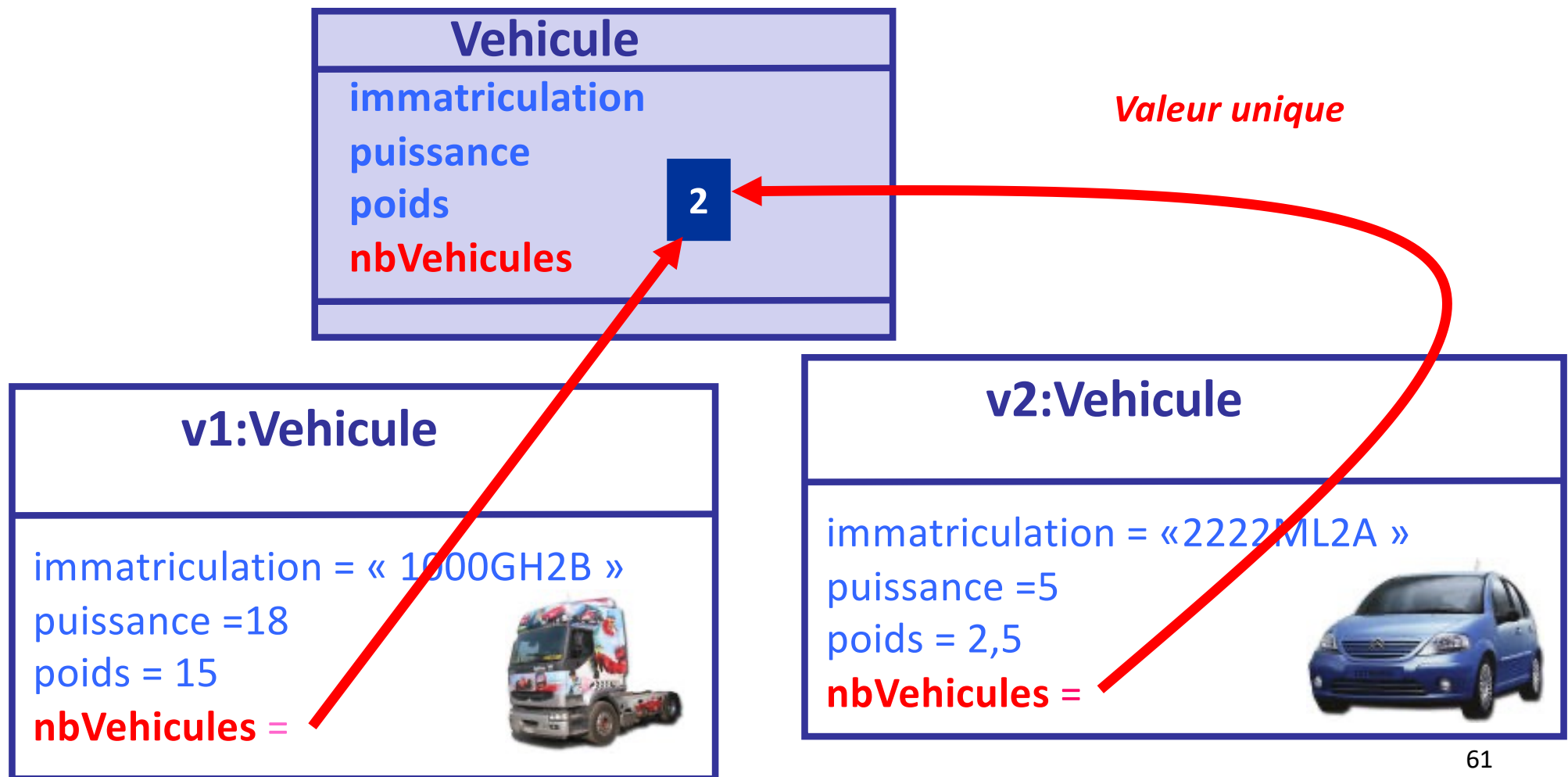
Attributs et méthodes « de classe »

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

Attributs « de classe »

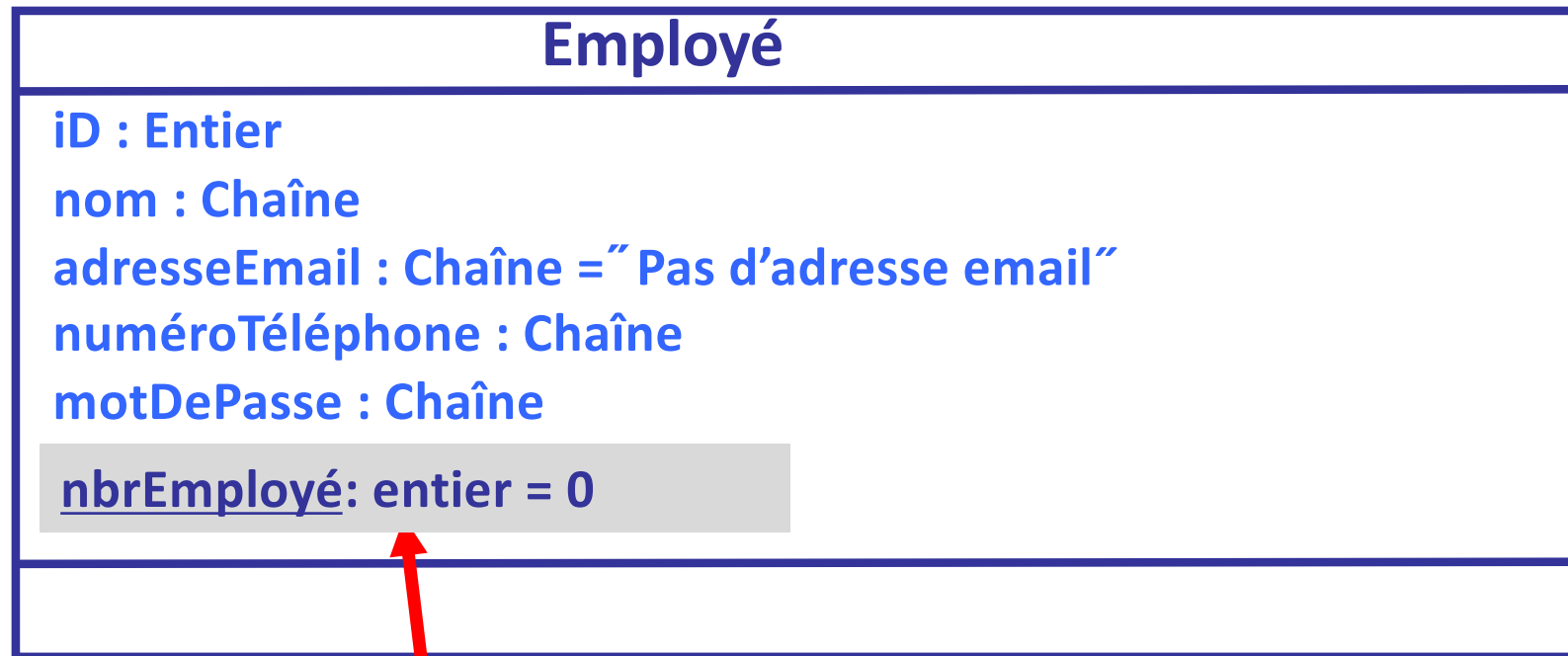
Attribut de classe =

Valeur partagée par toutes les instances de la classe



Attributs de classe en UML

Exemple



Attribut de classe (nom souligné):
valeur partagée par tous les objets d'une 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;  
    ...  
}
```

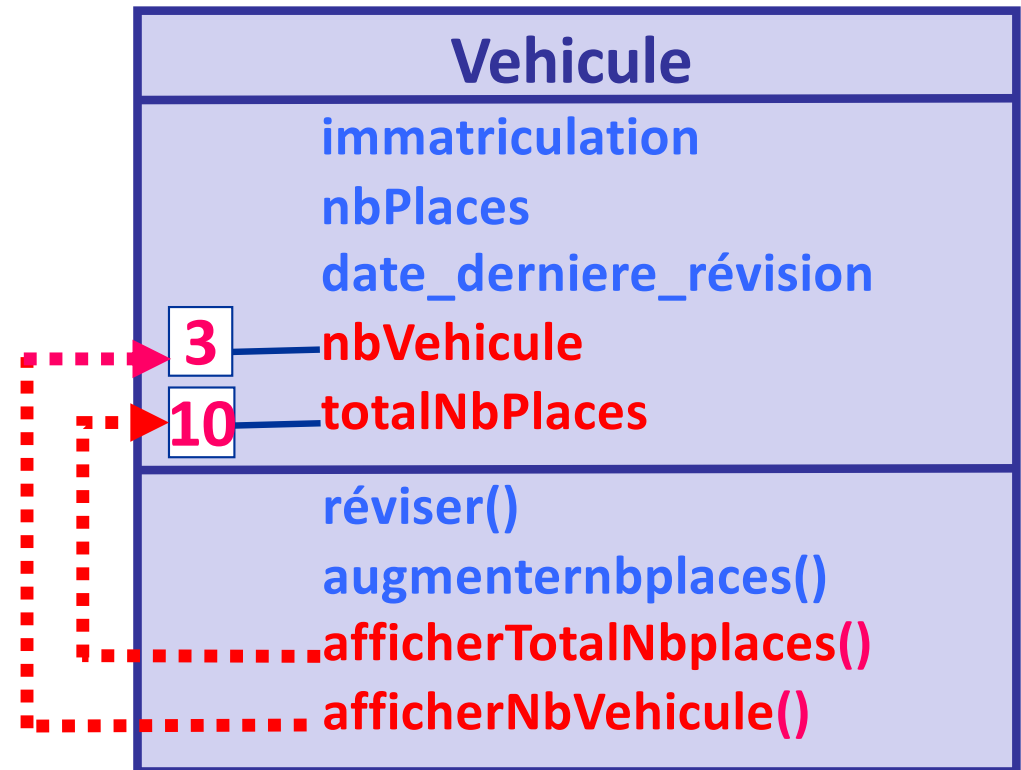
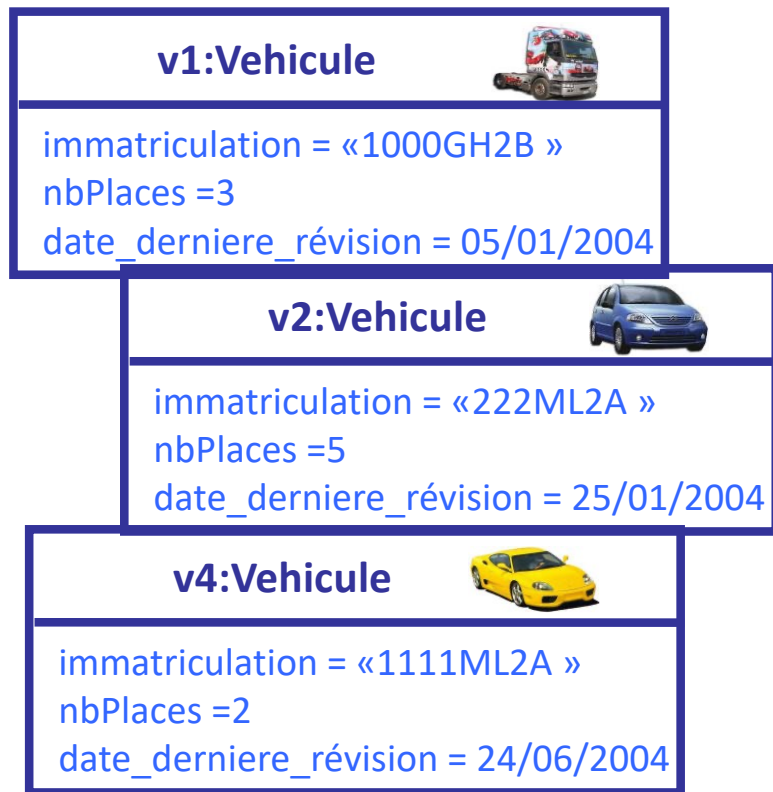


Variable ou champ statique:
attribut de classe en Java

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

- Une opération de classe est appelée **méthode statique** en Java.
- Une méthode statique n'est pas liée à une instance mais à une 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;  
        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

Invocation des attributs et méthodes de classe



■ Déclaration

```
static type nomVariable;  
static typeRetour nomMéthode(type arg,...) {...}
```

■ Invocation

```
NomClasse.nomVariable;  
NomClasse.nomMéthode(arg,...) ;
```



Utilisation du nom de la 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());  
    }  
}
```

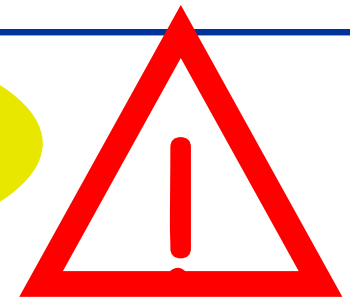


Opération de classe en Java

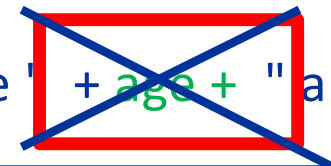
- Une méthode statique ne peut manipuler que les variables statiques de sa classe.
- Une méthode statique ne **peut pas manipuler des variables d'instances.**

```
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

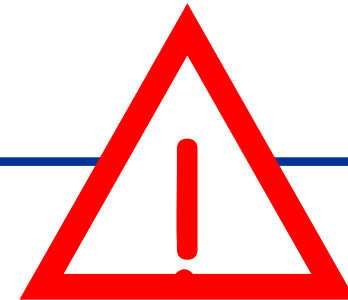




Opération de classe en Java

- **main** est une méthode de classe
- Pourquoi le programme suivant provoque-t-il une erreur de compilation?

```
public class UneClasse
{
    public int unAttribut;
    public static void main(String args[])
    {
        unAttribut = 5;
    }
}
```



INTERDIT car *unAttribut*
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.



Le modificateur final

- Si la variable est d'un type primitif sa valeur ne peut être changée
- Si la variable est une référence à un objet, on ne peut pas modifier la référence, mais par contre on peut faire évoluer l'objet...

```
final Personne p = new Personne("Machin", 25);  
...  
p.nom = "Truc"; //Autorisé  
p.setAge(6); //Autorisé  
p = autrePersonne; //Erreur ! interdit
```


Constantes en java



Une variable déclarée **final et static** doit être initialisée à la déclaration et ne peut plus être modifiée ensuite

```
static final double PI = 3.1416;
```



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
```

Les membres statiques



```
public class Circle{
    public static int count = 0;
    public static final double PI = 3.14;
    private double x,y,r;
    public Circle(double r){
        this.r = r; count++;
    }
    public Circle bigger(Circle c){
        if (c.r>this.r) return c;
        else return this;
    }
    public static Circle bigger(Circle c1, Circle c2){
        if (c1.r>c2.r) return c1;
        else return c2;
    }
}
```

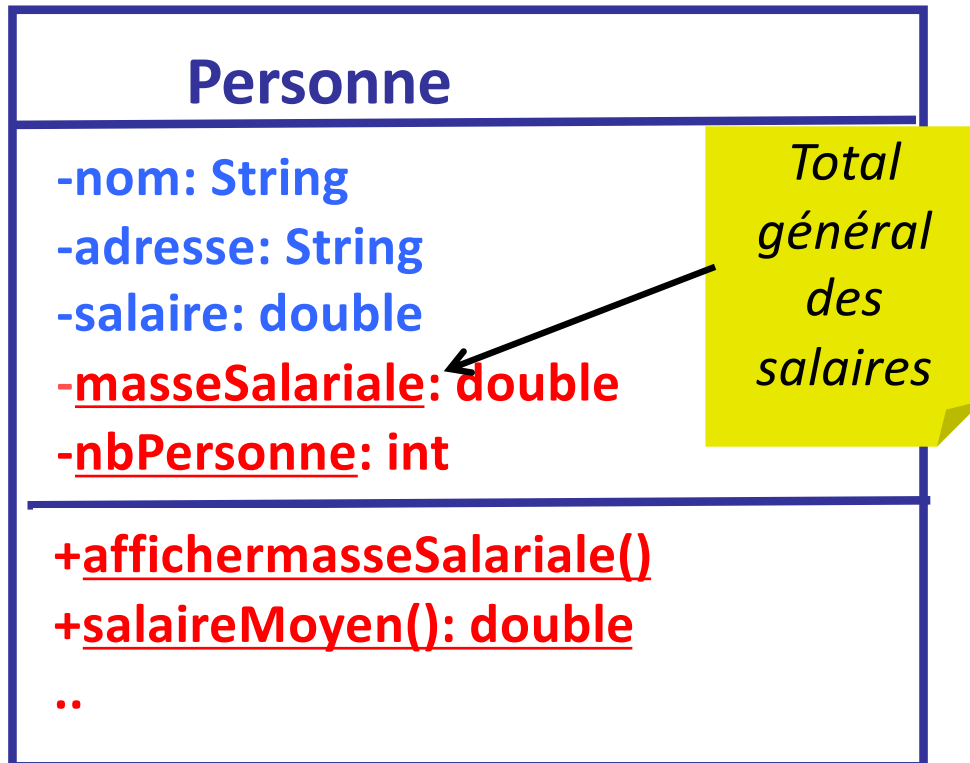
```
Circle c1 = new Circle(10);
Circle c2 = new Circle(20);
int n = Circle.count; // n = 2
Circle c3 = c1.bigger(c2); // c3 = c2
Circle c4 = Circle.bigger(c1, c2); // c4 = c2
```



TPCours - Exercice (5)



- Complétez la classe Personne par la définition des attributs et méthodes statiques conformément à la représentation UML ci-dessous (**attention! d'autres méthodes devront être modifiées**):



- Complétez la méthode main de votre classe TestPersonne afin de faire afficher la masse salariale et le salaire moyen sous la forme suivante:

Affichage Ecran

Masse Salariale totale: ... euros
Salaire Moyen: euros



Surcharge

- Surcharge de constructeurs
- Surcharge de méthodes

Surcharge de Constructeurs



- Constructeur par défaut, implicite
 - Constructeur vide : `NomClasse()`
 - Ne fait rien, peut être redéfini
- Plusieurs constructeurs, surcharge
 - Diffèrent par le nombre et/ou le type des paramètres, c-a-d par leur signature



Si un constructeur est défini, le constructeur vide implicite disparaît



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 immat, short puissance) {
        //constructeur 1
        this.immat = immat;    this.puissance = puissance;
    }
    public Vehicule(String immat) {
        this(immat,0); //appel du constructeur 1
    }
    public Vehicule(Vehicule v) {
        this(v.immat, v.puissance); //appel du constructeur 1
    }
    public Vehicule() {
        this("", 0); //appel du constructeur 1
    }
}
```


Le mot clé this

- Passer une référence à l'instance courante dans un appel de méthode

```
public void trace() {System.out.println(this); }
```



TPCours - Exercice (6)



- Complétez la classe `Personne` par la définition d'un deuxième constructeur ne comportant que deux paramètres `nom` et `salaire` (l'adresse sera initialisée à vide)
- Complétez la méthode `main` de la classe `TestPersonne`
 - par la définition d'une deuxième personne ayant pour nom « `Machin` » et pour salaire 2000 euros. Utilisez pour cela une invocation du constructeur à 2 paramètres défini ci-dessus.
 - Par l'affichage de cette personne.



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
```

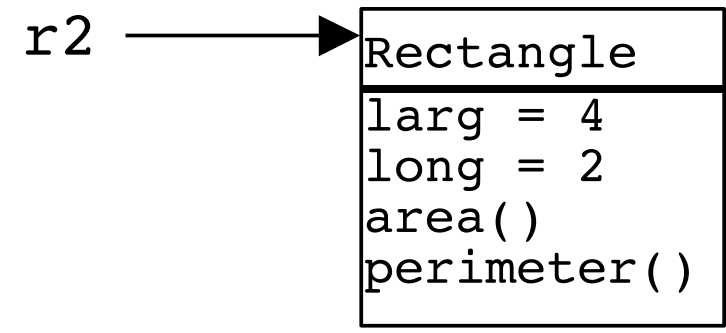
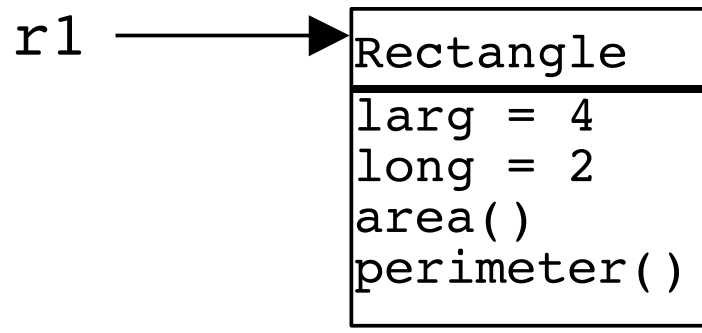


Comparaison d'objets

- Que compare-t'on ?

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

Le test
rend
FAUX !!



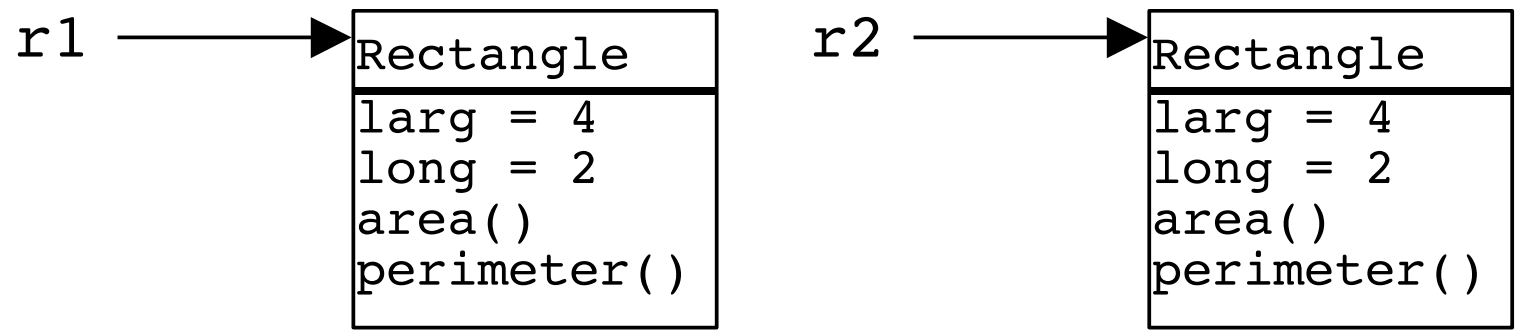


Comparaison d'objets

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

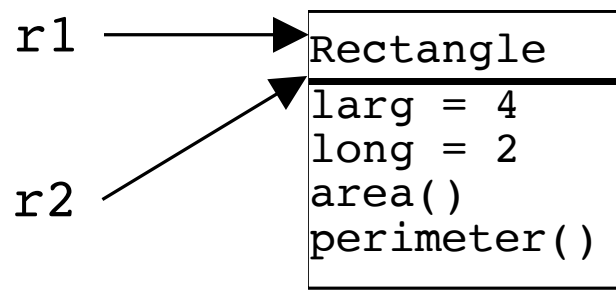
Il faut que la méthode `equals` ait été explicitement **redéfinie** dans la classe `Rectangle`

Nous y reviendrons plus tard !
Cf. Chapitre 4-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

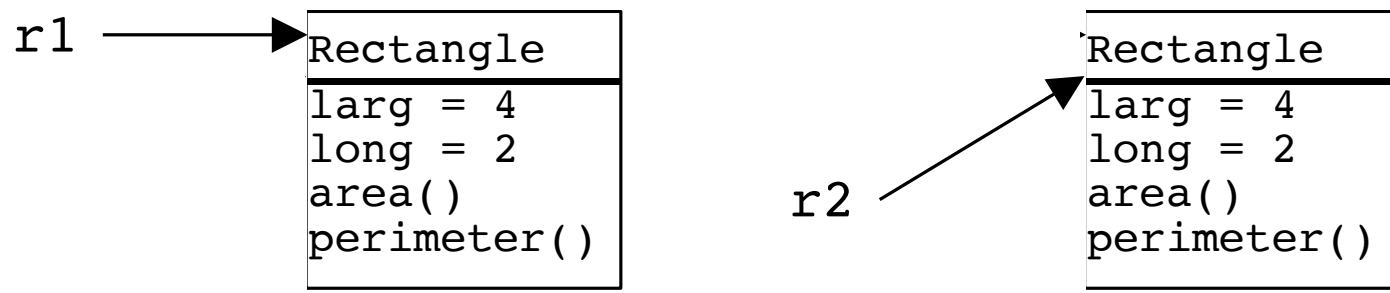
Une véritable copie est un « clonage » de l'objet



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 4-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?

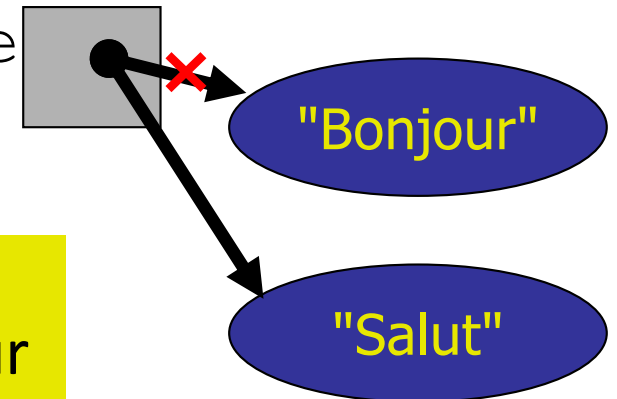
Les Strings sont des
objets immutables

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";
```

maChaine



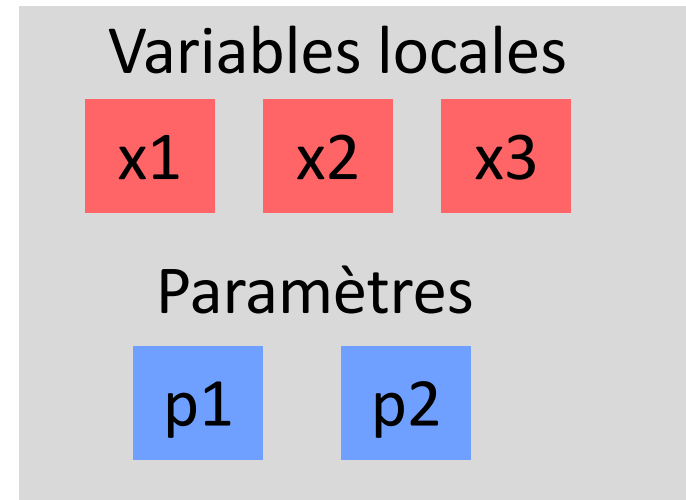
Les concaténations sont couteuses!!
Utiliser de préférence la classe StringBuffer pour
créer des strings mutables

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

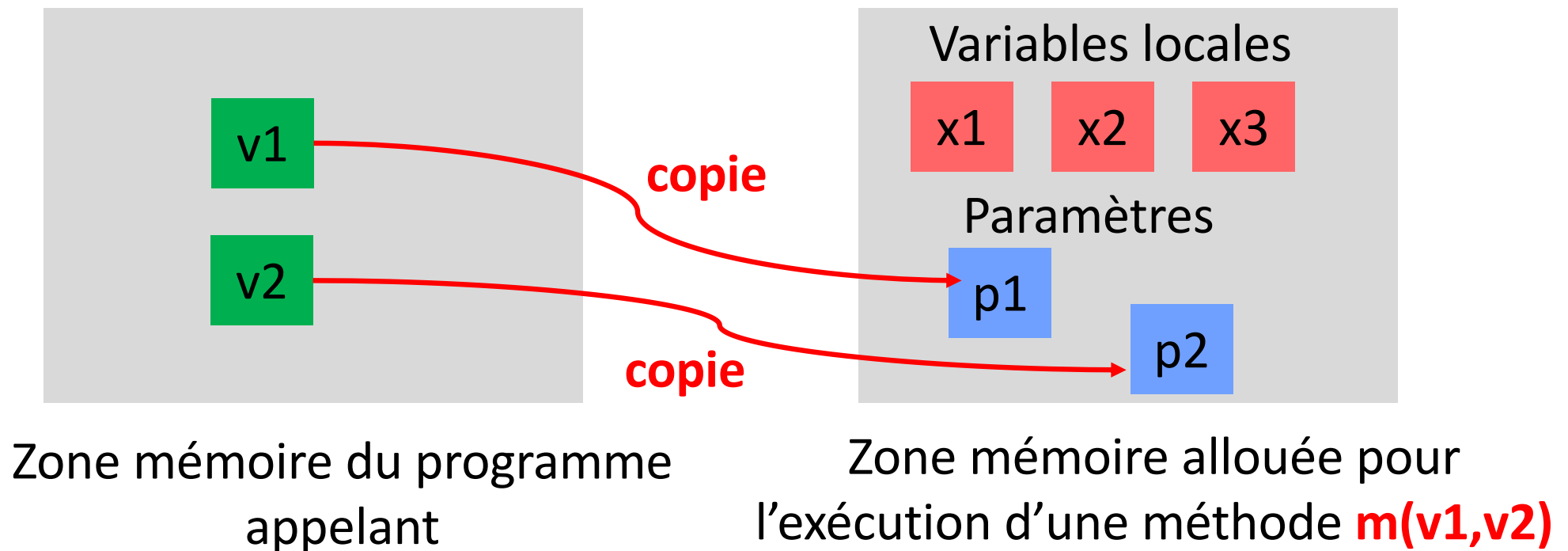


Zone mémoire allouée pour l'exécution d'une méthode $m(v1, v2)$

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



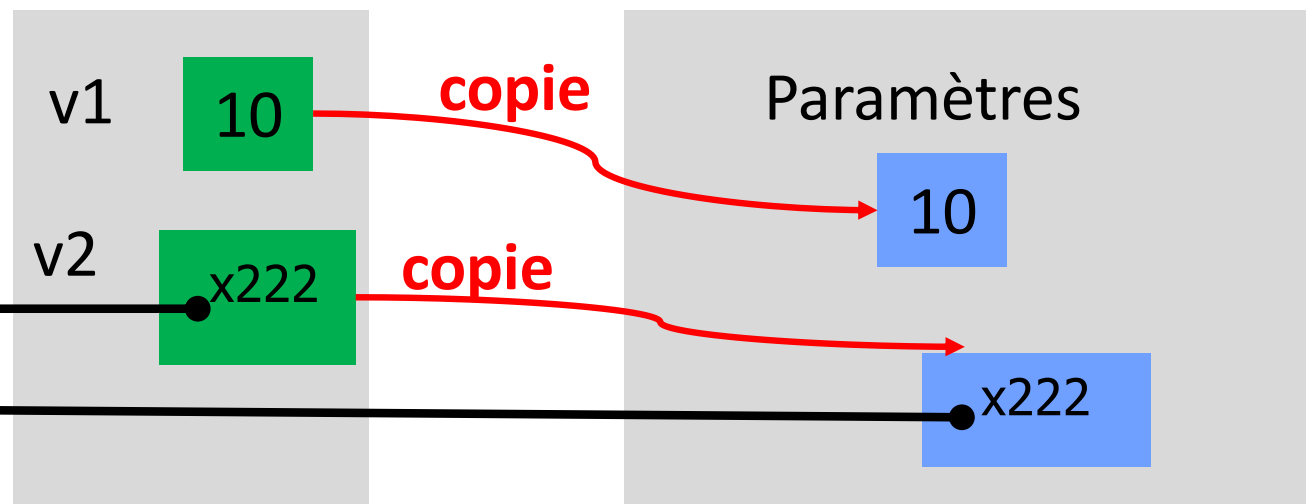
Principes de transmission des paramètres à une méthode



- 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()
```



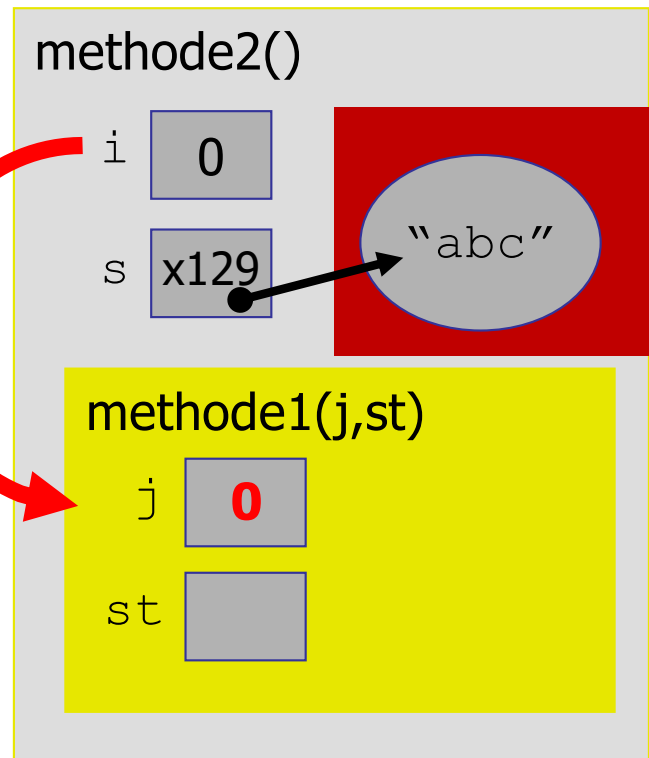


Méthodes en Java

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



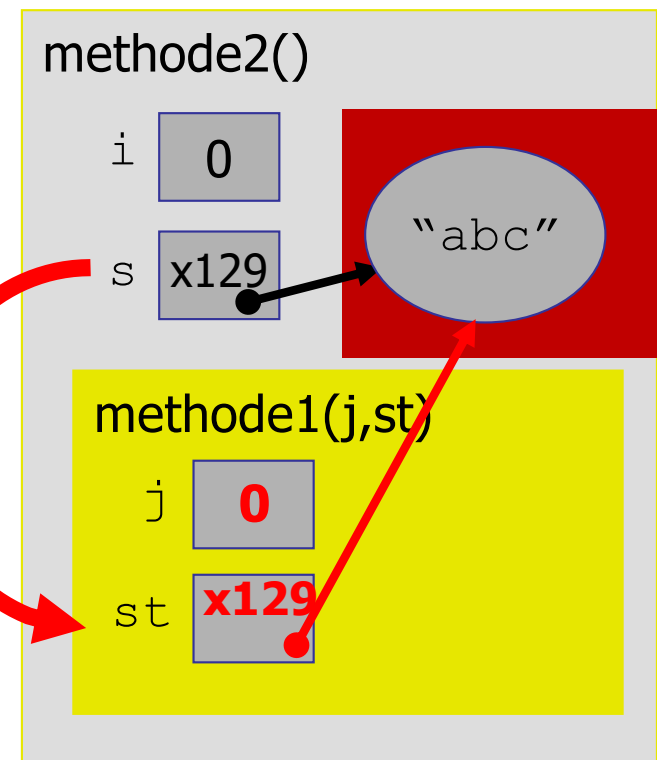


Méthodes en Java

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

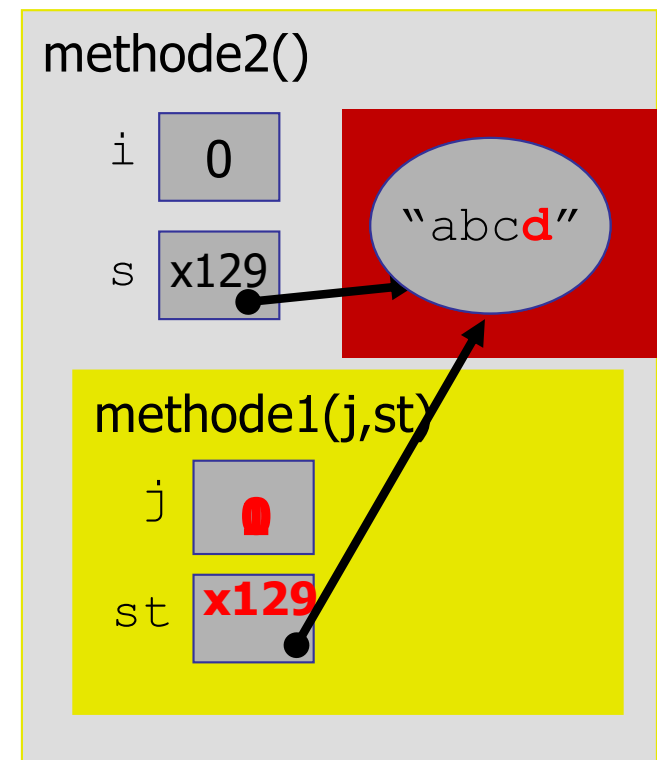




Méthodes en Java

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);  
    }  
}
```

methode2()

i 0

s x129

"abcd"

methode1(j,st)

j 1

st null

Affichage de i=0 ,s=abcd



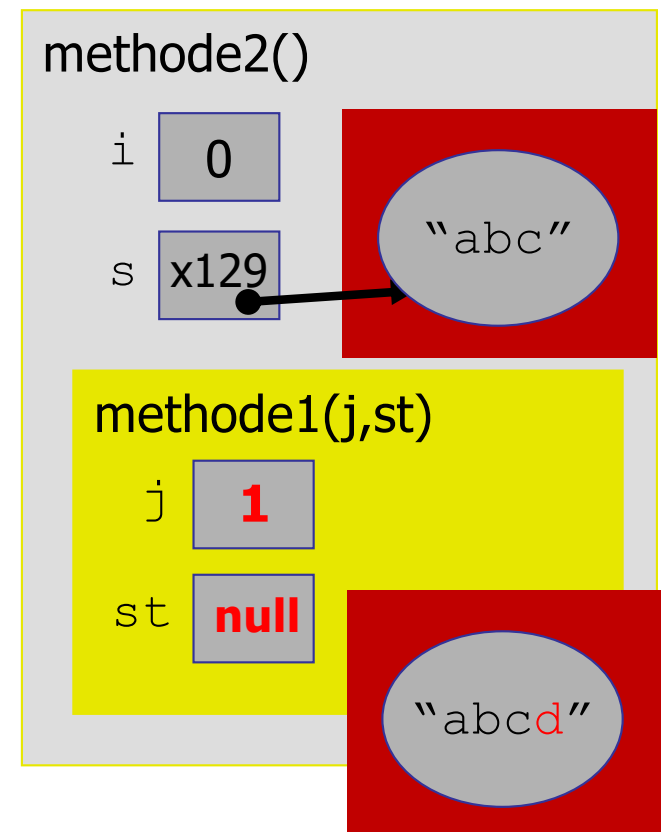
Méthodes en Java

Passage de paramètres

Avec des String, quels changements?

```
public class Essai2 {  
    void method1(int j, String st) {  
        j++;  
        st=st+"d";  
        st = null;  
    }  
    void method2() {  
        int i = 0;  
        String s = "abc";  
        method1(i,s);  
        System.out.println ("i="+i+",s="+s);  
    }  
}
```

Affichage de i=0 ,s=abc



Destruction d'objet en JAVA



- Automatique, ou presque...
 - Prise en charge par le *garbage-collector*
- Dès que la référence est vide ou hors de portée
 - Destruction de l'objet
 - Libération de la mémoire
- Méthode `finalize()`...