

L3 Informatique - 2020-2021

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

La POO: une autre manière de structurer les programmes

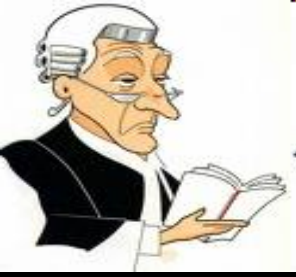
- Démarche centrée sur la **recherche des « Objets »** (données) à travers leurs propriétés caractéristiques et leur comportement
- Un programme est **constitué d'objets** collaborant entre eux par envois de messages

Programmation structurée classique

- Ensembles de **fonctions** qui s'appellent entre elles

Programmation orienté Objets

- Ensemble de **classes** permettant de définir des objets



Principes de base de la POO

Abstraction



*Objets et Classes,
Mécanisme d'instanciation*

Encapsulation



Attributs, méthodes, visibilité

Hiérarchie d'héritage



Polymorphisme

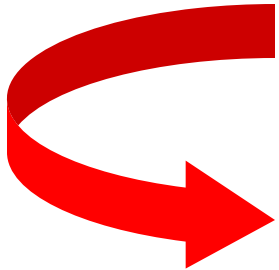


C'est l'objectif de ce cours!!

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



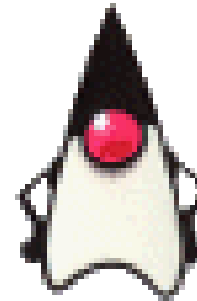
Le langage Java

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



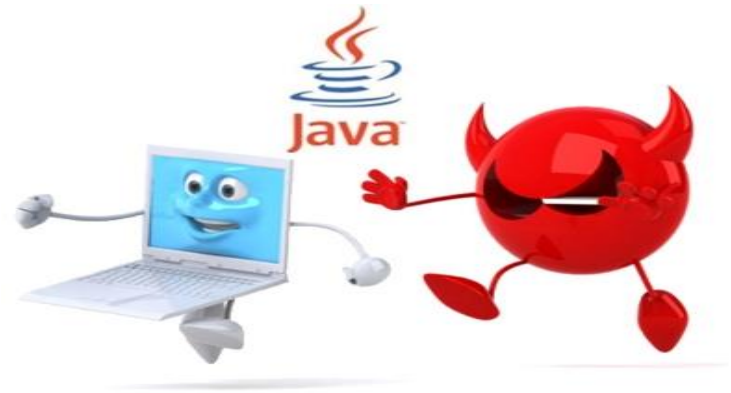
ORACLE®

Rachat en
2009

Présentation de JAVA

JAVA est une "plateforme"

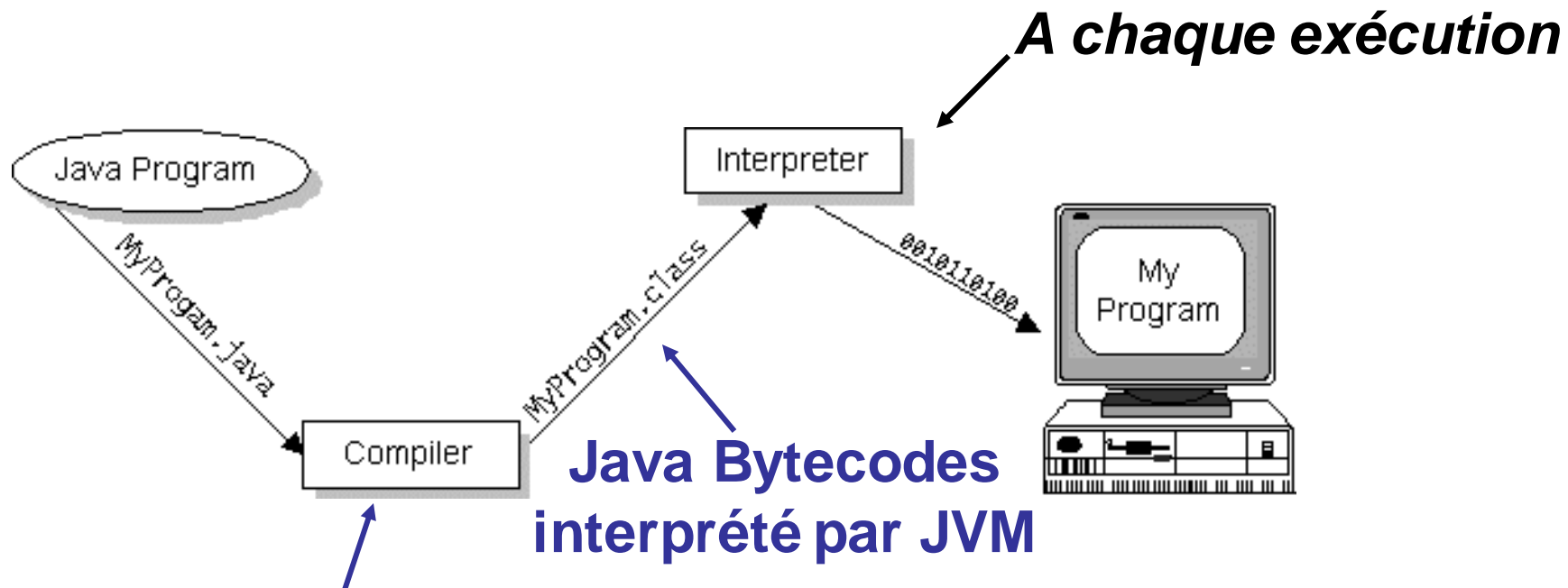
- Un langage de programmation orienté objets
 - Un ensemble de classes standards réparties dans différents **packages** (**API**)
 - Un ensemble d'outils (le **JDK**,...)
- Un environnement d'exécution :
la **machine virtuelle** (JVM)





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.



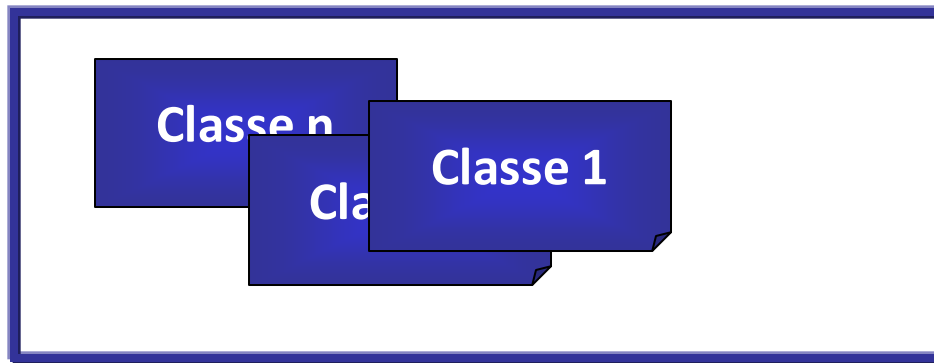
Présentation de JAVA

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

Types de programmes Java

- **Les Applications indépendantes**
 - Programmes autonomes (stand-alone)



- **Les Applets**
 - Programmes exécutées dans l'environnement d'un navigateur Web et chargés au travers de pages HTML

Seuls diffèrent les contextes d'invocation et d'exécution

- Les droits des applets et des applications ne sont pas les mêmes

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[ ])  
{  
.../...  
}
```


Création d'une application

- Prenez votre éditeur de texte préféré...

```
/**
 * La classe HelloWorldApp implémente une
 * application qui
 * affiche simplement "Hello World!" sur la sortie
 * standard.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        // Affichage du message
        System.out.println("Hello World!");
    }
}
```

Même les
MAJUSCULES
sont
importantes :
A ≠ a

- Sauvegardez le fichier sous le nom
HelloWorldApp.java

TP Cours

- Récupérer le sujet de TP Cours
- Téléchargez et installez l'environnement de développement Eclipse en suivant la démarche indiquée dans le sujet :
 - <https://www.eclipse.org/downloads/>
- Créez votre premier programme « hello world »



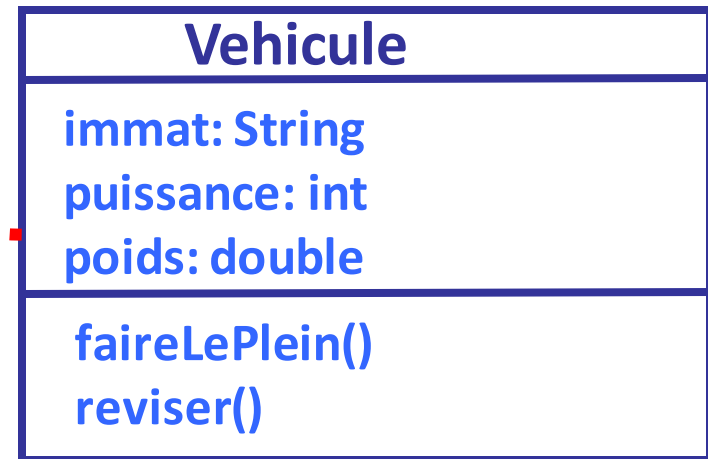
Définition de classes

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

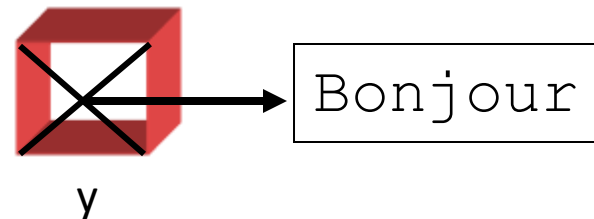
`int x=4`



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

```
    }
```

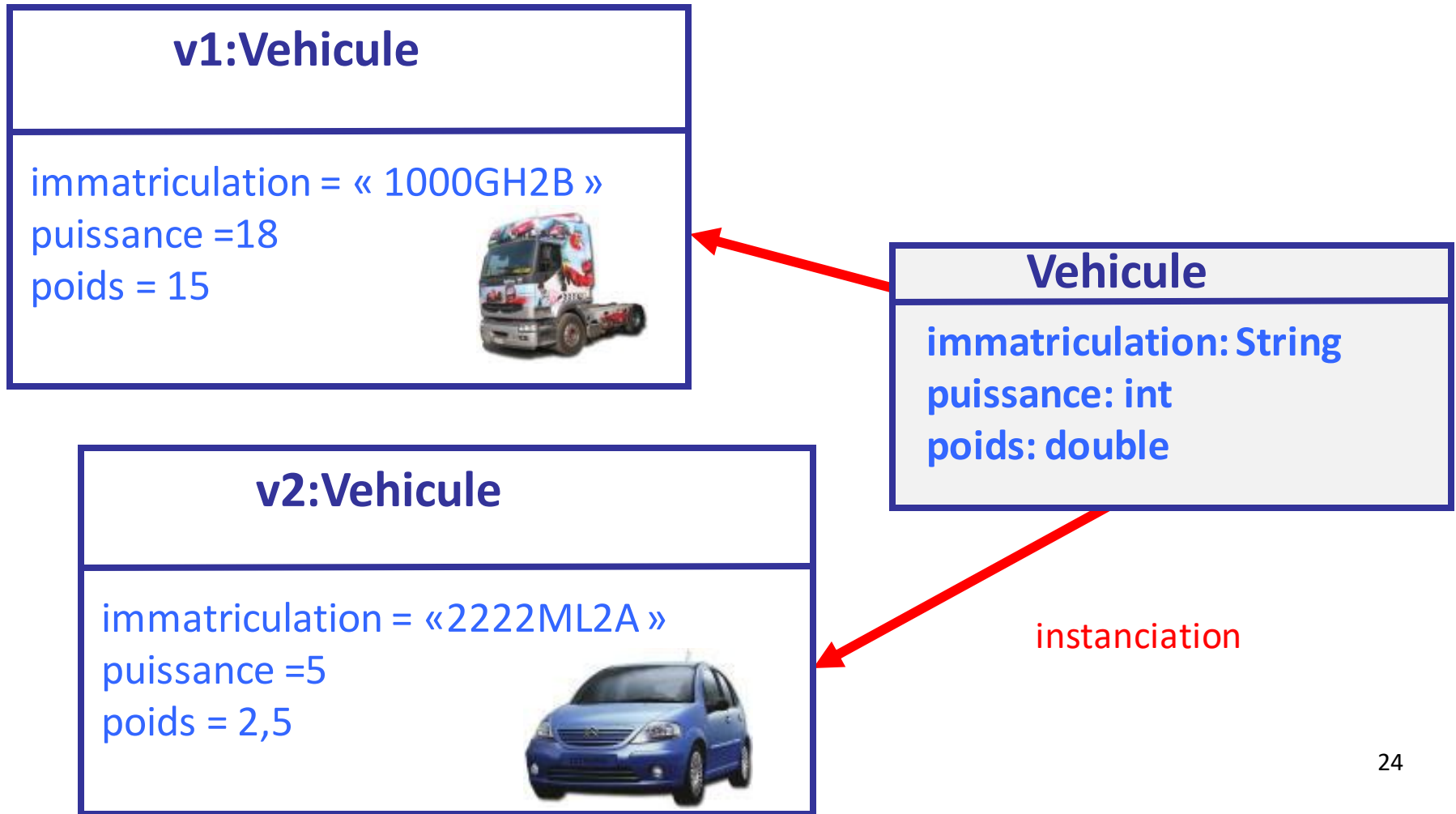




Création d'instances (ou objets)

Notion d'instance

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



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

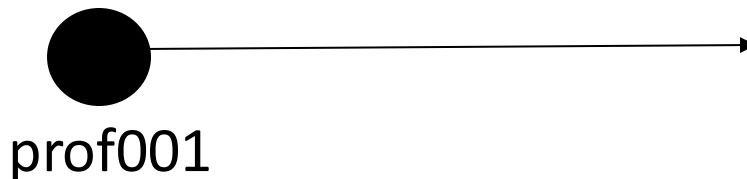


Valeurs propres à chaque instance



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)

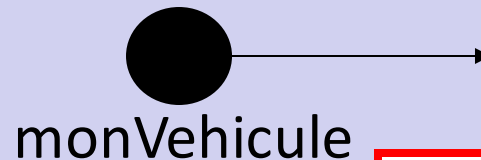


Vehicule monVehicule;

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

Avant

Après



??

monVehicule = *null*

Création d'objets en JAVA

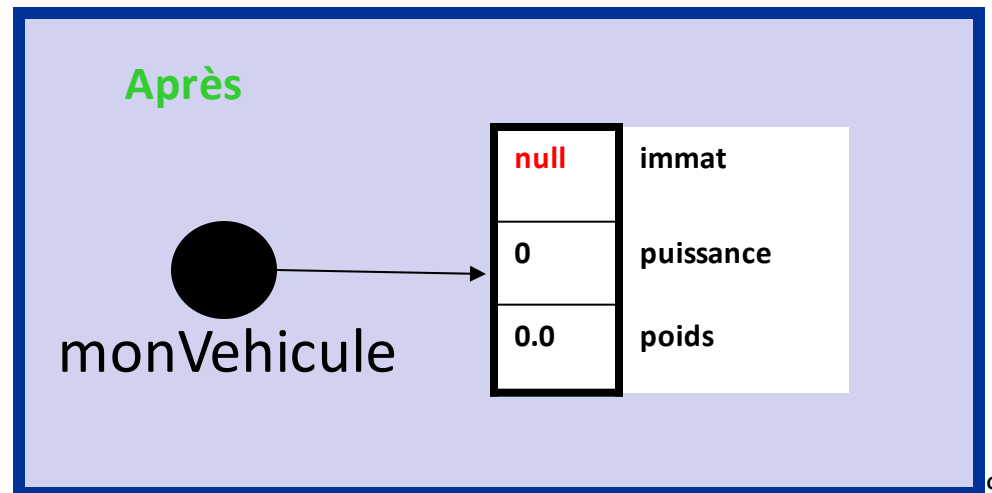
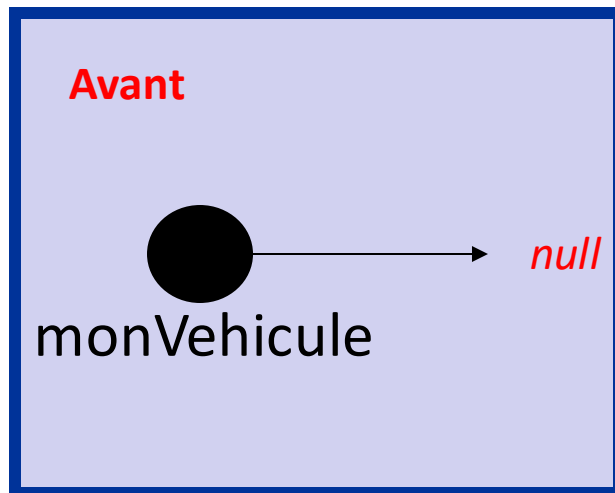


Création de l'objet (instanciation)

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

⇒ réserve la mémoire pour stocker l'objet

⇒ associe l'objet à la référence



Création d'objets en JAVA

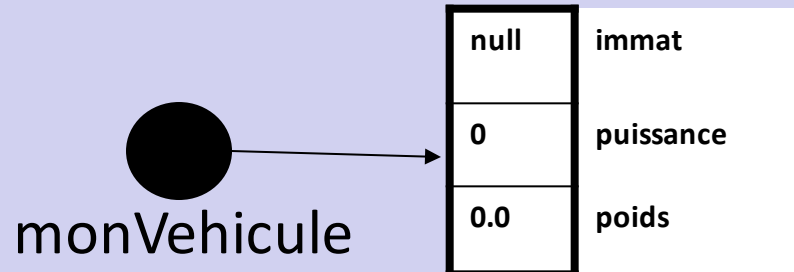


Déclaration et instantiation

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

Avant

Après



Accès aux valeurs des attributs



Une classe

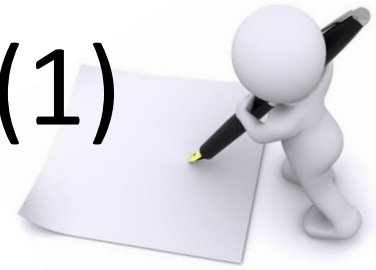
```
class Vehicule{  
    /** l'immatriculation de ce véhicule */  
    String immat;  
    /** La puissance */  
    int puissance;  
    /** Le poids de ce véhicule. */  
    double poids;  
    ... }  
}
```

Une classe de test

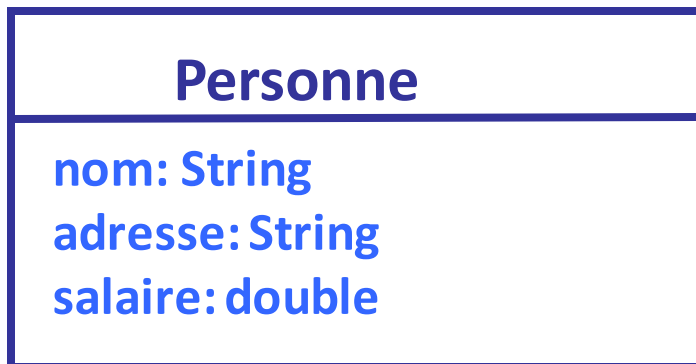
```
class TestVehicule {  
    public static void main(String[] args){  
        Vehicule monVehicule=new Vehicule();  
        monVehicule.immat="1000GH2B";  
        monVehicule.puissance=18;  
        monVehicule.poids=15.5;  
        ....  
    }  
}
```



TP Cours - Exercice (1)



- Définir un package `exercice` dans votre projet TP Cours.
- 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.

v3:Vehicule

immatriculation = «1111ML2A»
poids = 15
age = **2**
jauge = 10



v4:Vehicule

immatriculation = «4444MS2R»
poids = 9
age = **7**
jauge = 15



Vehicule

immatriculation: String
age: int
jauge: double
poids: double

afficherJauge
augmenterAge
identiteVehicule: String
remplirJauge(q: double)
jaugeEstVide(): boolean

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

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 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();  
    }  
}
```



TP Cours - Exercice (2)



- Compléter en Java la classe `Personne` de l'exercice (1) par la définition des méthodes conformément à la représentation UML suivante:
- 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 1000 euros.

Personne

`nom: String`
`adresse: String`
`salaire: double`

`afficher ()`
`changerAdresse(nouvelle:String)`
`salaireAnnuel(): double`
`salaireEstSup1000():boolean`

Affichage Ecran

Titi habite Corte
Titi habite Ajaccio
Salaire annuel de Titi: 24000 euros
Titi a un salaire supérieur à 1000



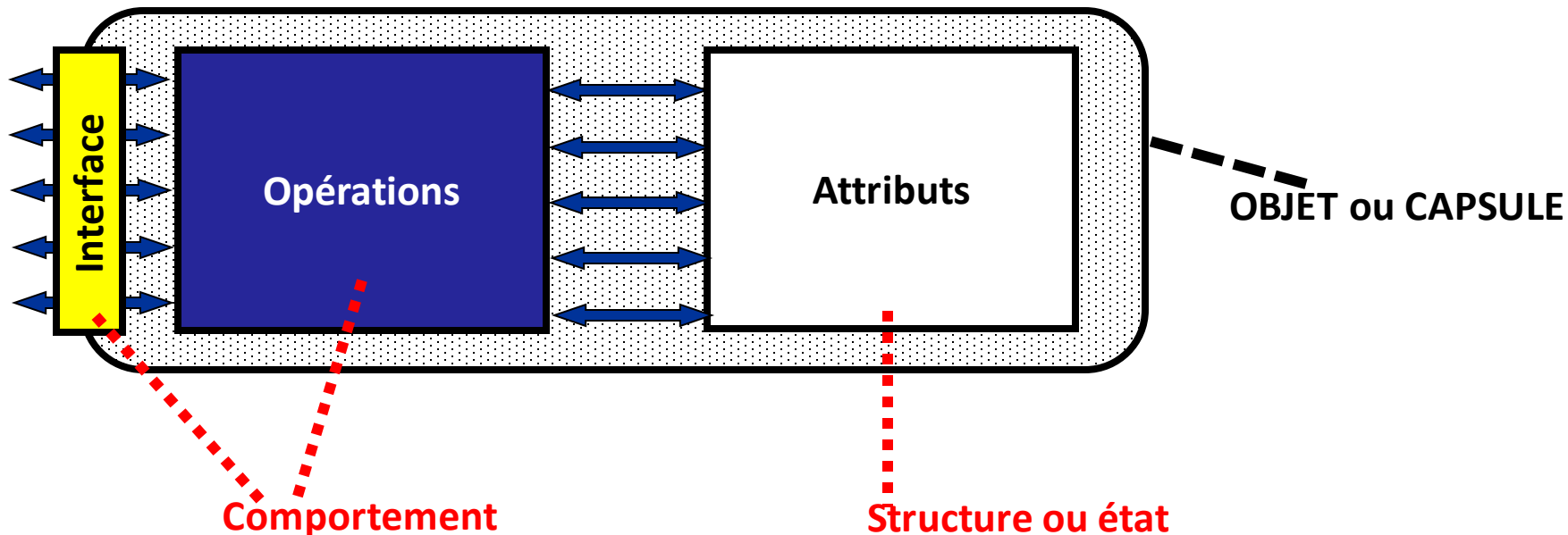
Principe d'encapsulation et visibilités

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

Encapsulation

Principe = séparation spécification/réalisation

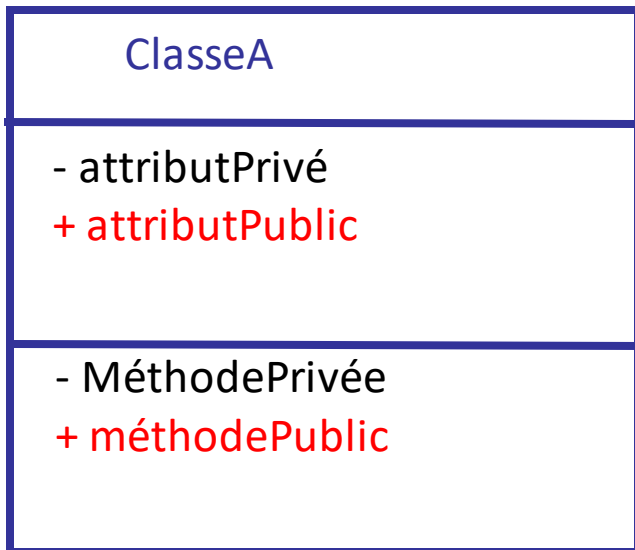
- Les objets ne sont manipulés qu'à travers leur interface
- Les détails de l'implémentation sont occultés



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?

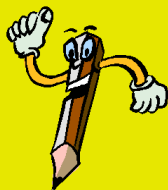


- Pour sécuriser le code!
- Certaines classes sont développées par d'autres programmeurs (vos fournisseurs)
 - Ils vous offrent des services (méthodes)
 - Vous êtes de simples utilisateurs
 - vous n'avez pas à connaître la structure de leurs classes (attributs) et les algorithmes de leurs méthodes
 -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

Accesseurs et modificateurs



- **Méthodes type `getNom Champ()`**
 - Retournent la valeur du champ
- **Méthodes void `setNomChamp(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 `TestPersonne`.

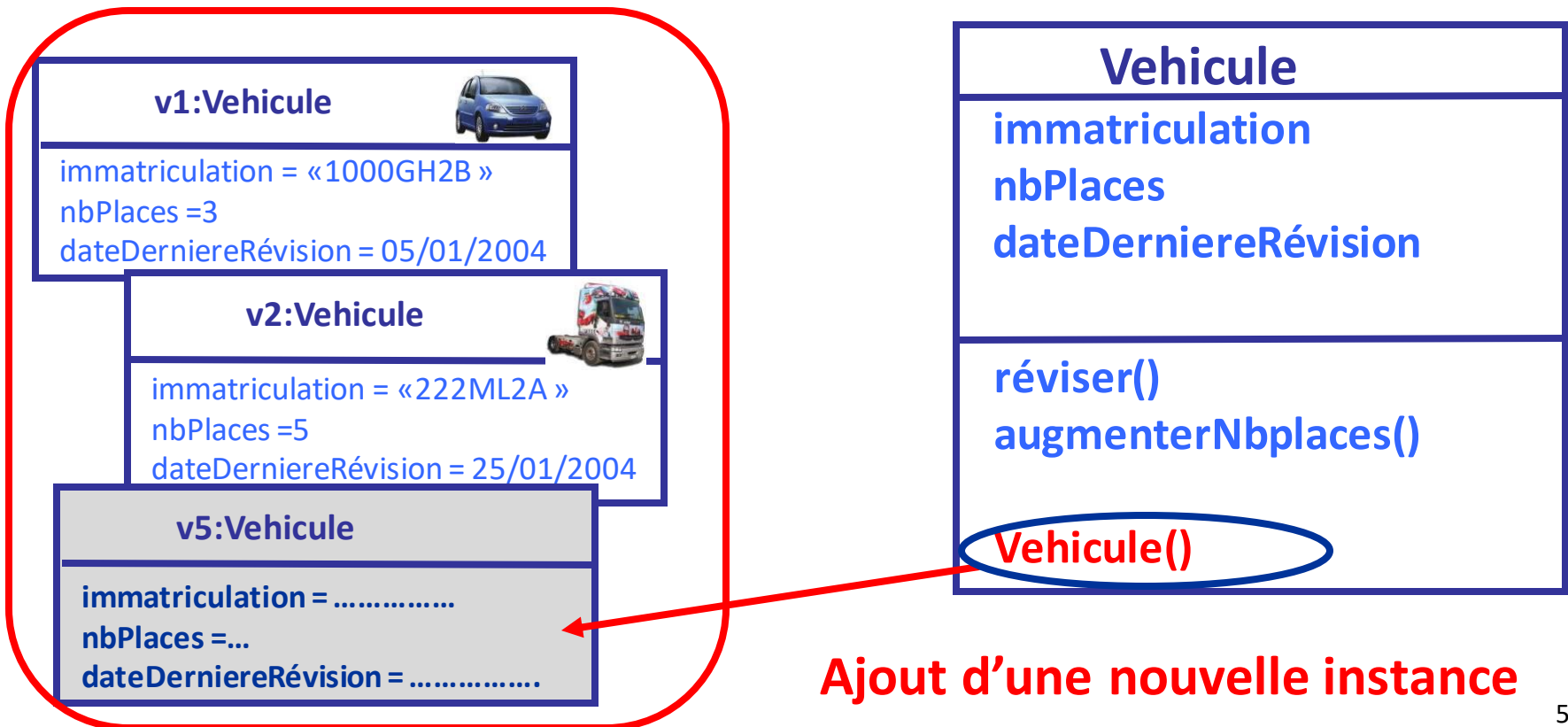


Définition de constructeur

Instanciación et Constructeurs

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

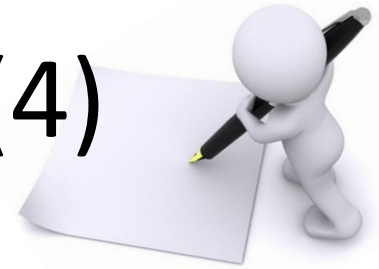
- Référence à l'instance (l'objet) courante
- **this** permet de lever une ambiguïté de nommage

```
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



TP Cours - 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?
- 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.



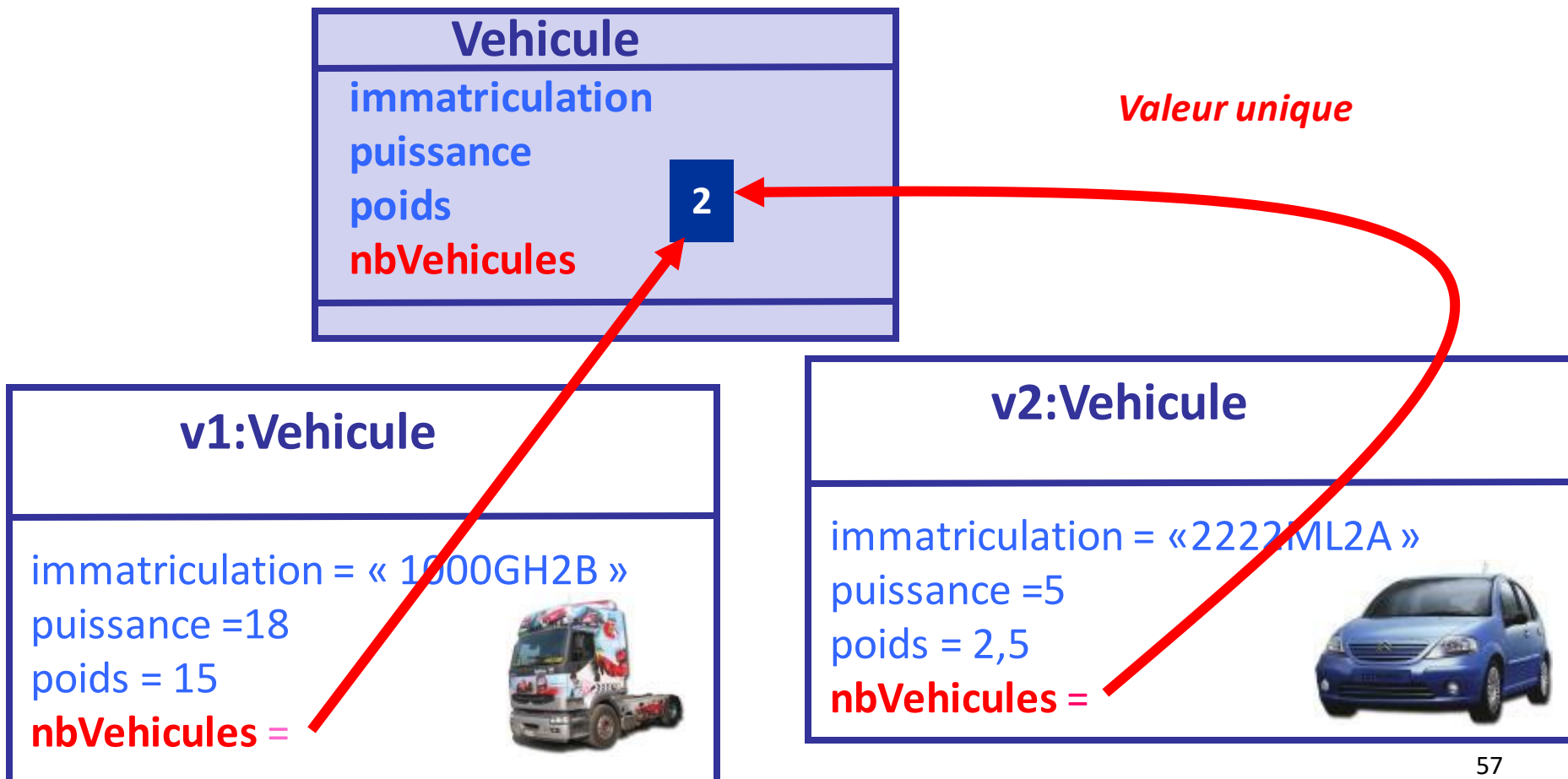
Attributs et méthodes « de classe »

- 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 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;  
    ...  
}
```

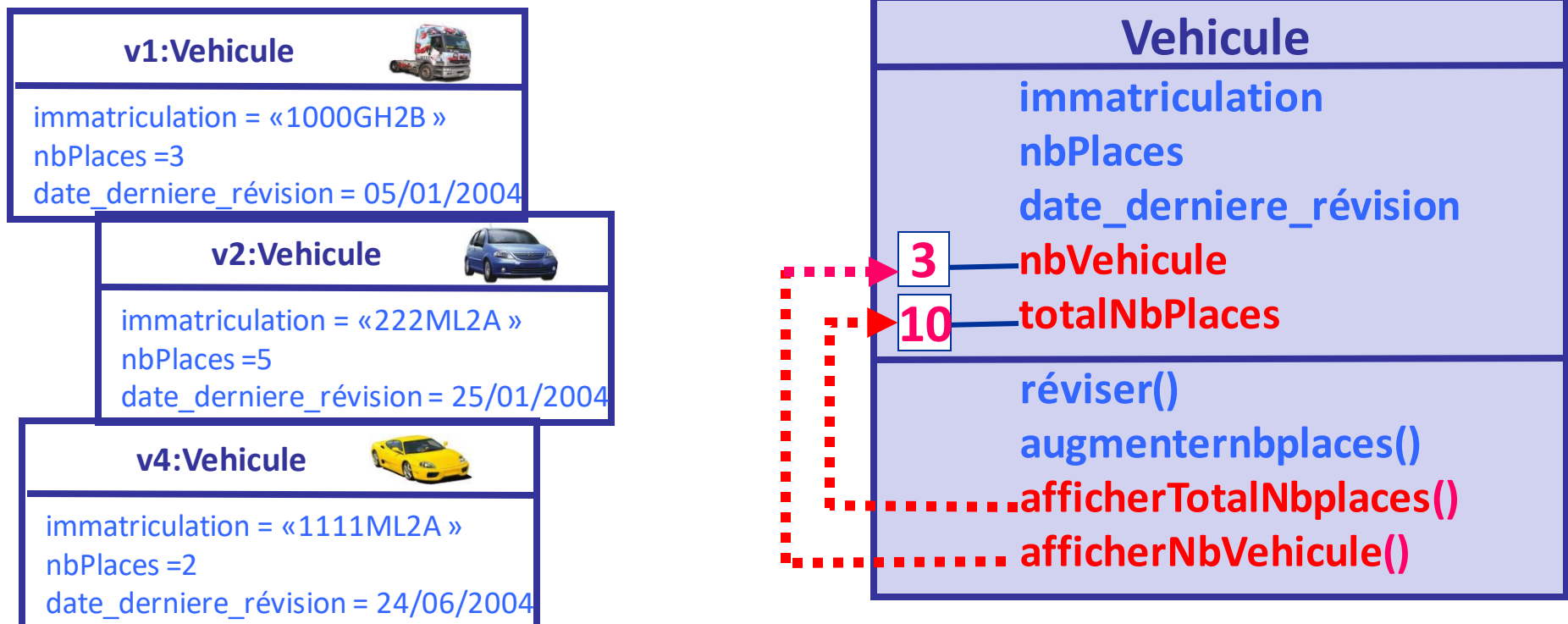


Variable ou champ statique:
attribut de classe en Java

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;  
        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

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());  
    }  
}
```

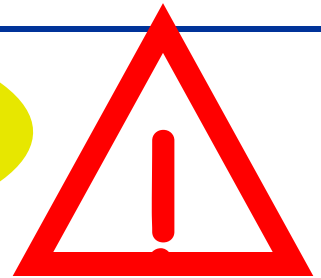


Opération de classe en Java

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

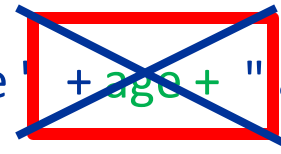
```
public class Vehicule{  
    Private int age;  
    private static int nbVehicule=0;  
  
    public void augmenterAge(){ age++;}
```

Ce code
comporte-t-il une
erreur?



INTERDIT car age est une variable d'instance

```
    public static void afficherNbVehicule(){  
        System.out.println("Nombre de Vehicules "+ nbVehicule + " de " + age + " ans");  
    }
```





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...

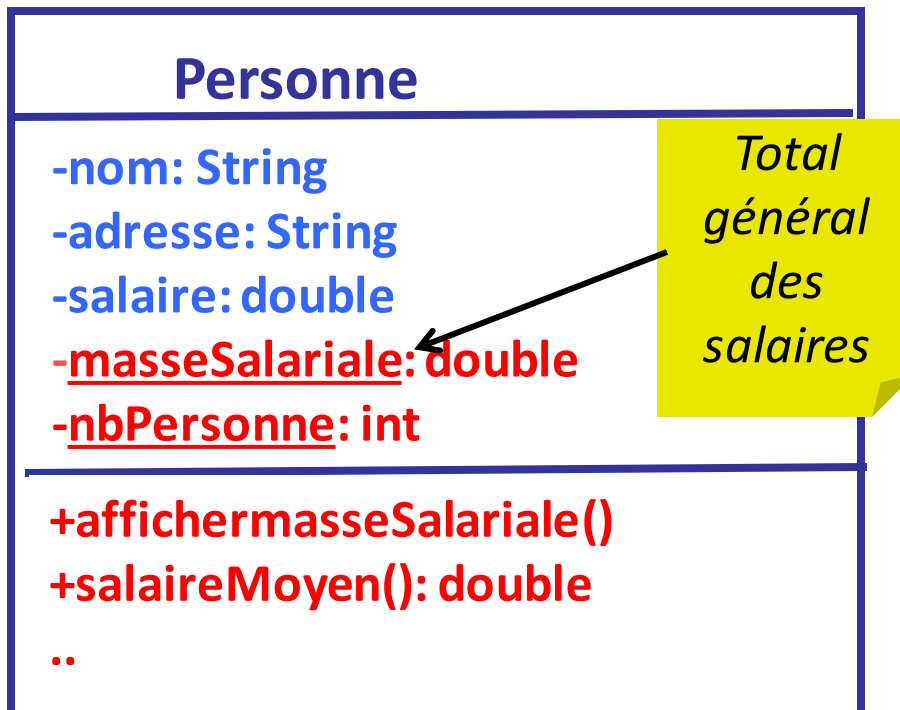
```
void methodeTest(final Personne p) {...}  
// p est inchangé, mais son nom peut l'être
```



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
 - Différent 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 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);  
    }  
}
```

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
```

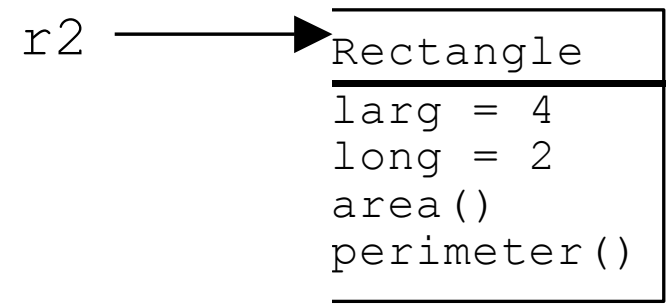
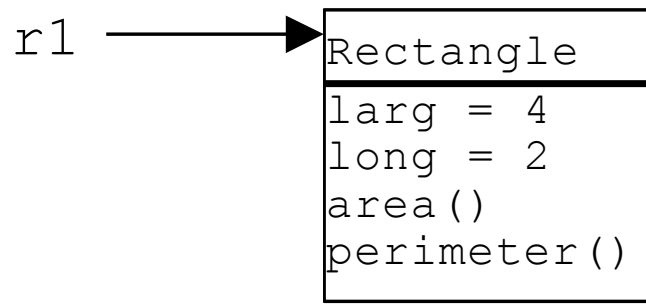



Comparaison d'objets

- Que compare-t'on ?

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

Le test
rend
FAUX !!



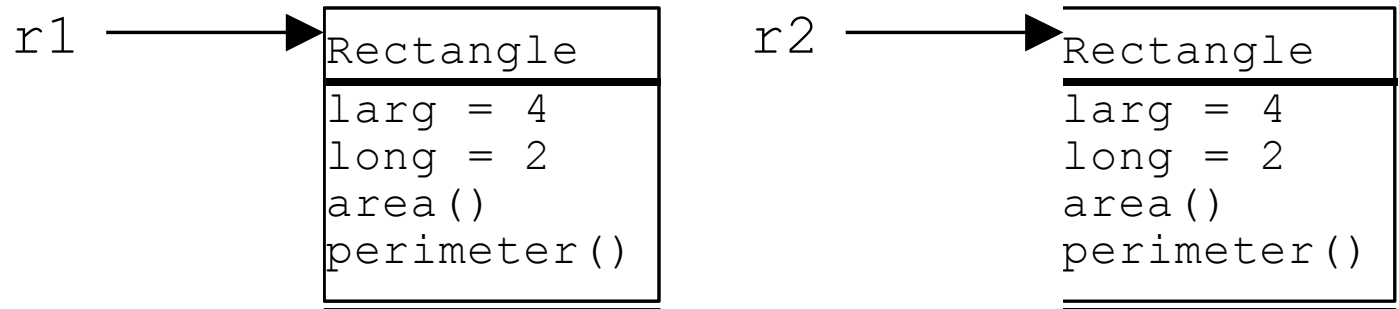


Comparaison d'objets

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

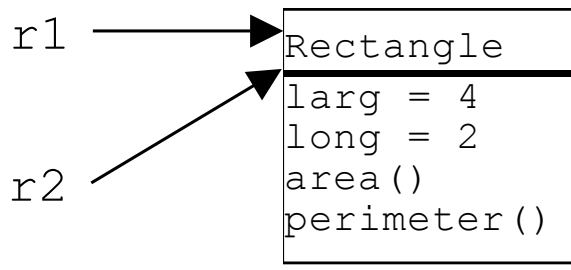
Il faut que la méthode equals ait été explicitement
redéfinie dans la classe Rectangle

Nous y reviendrons plus tard !
Cf. Chapitre 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

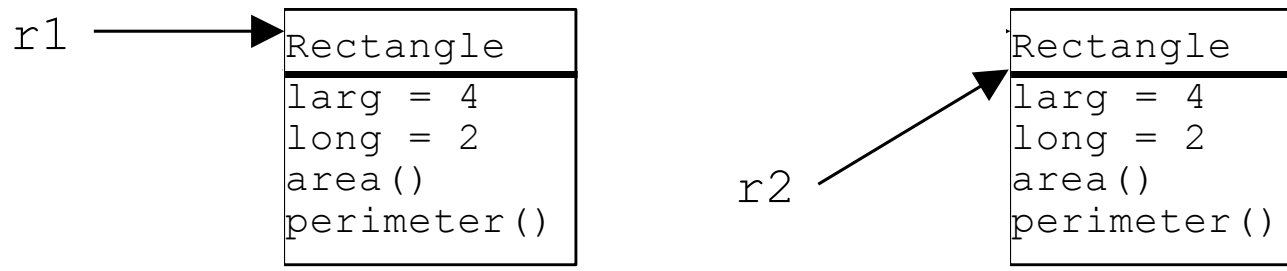
Une véritable copie est un « clonage » de l'objet



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

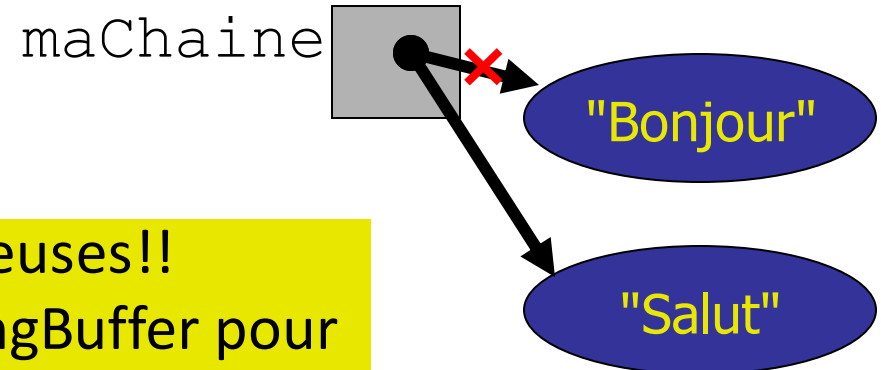
POURQUOI?

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";
```



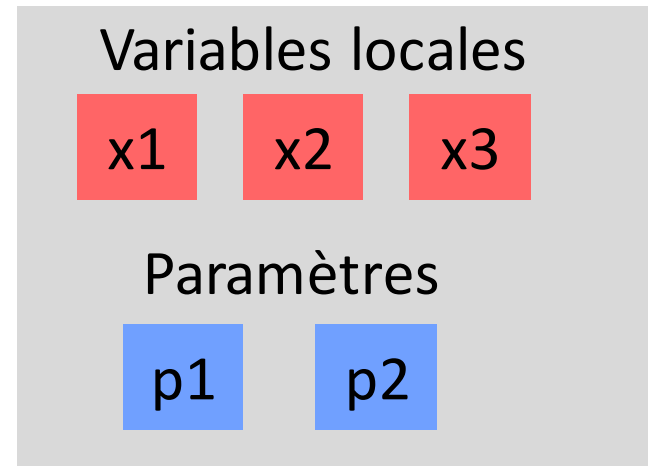
Les concaténations sont couteuses!!
Utiliser de préférence la classe StringBuffer pour
créer des strings mutables

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

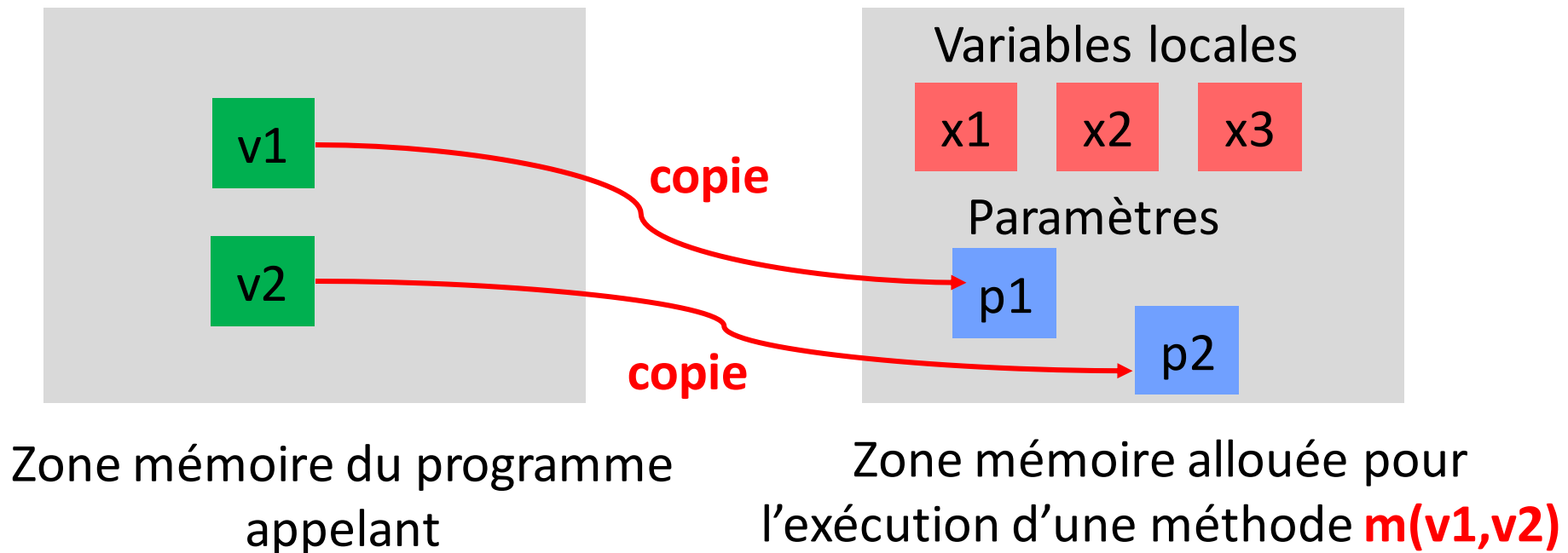


Zone mémoire allouée pour l'exécution d'une méthode $m(v1, v2)$

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



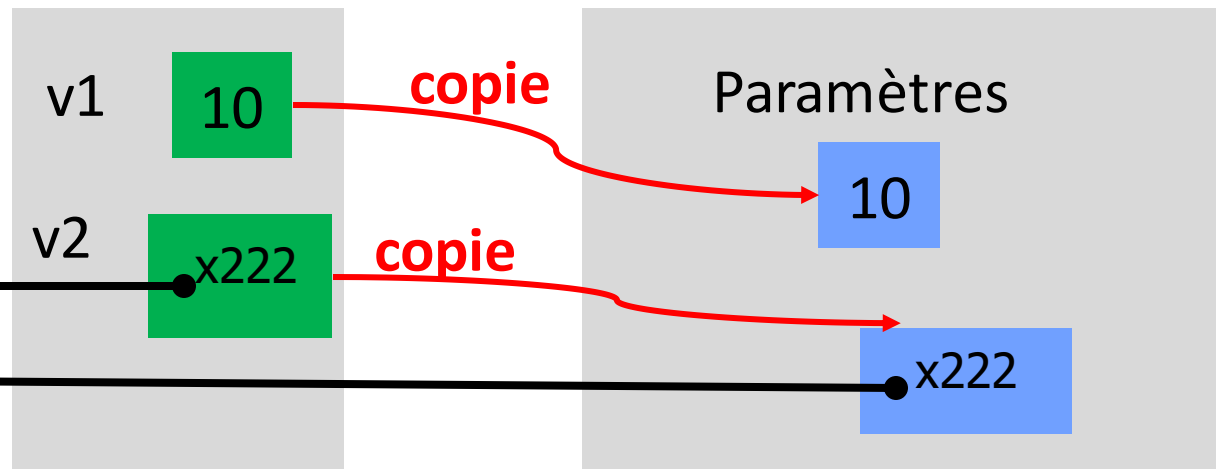
Principes de transmission des paramètres à une méthode



- 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

methode2()

i

0

s

x129

"abc"

methode1(j,st)

j

0

st

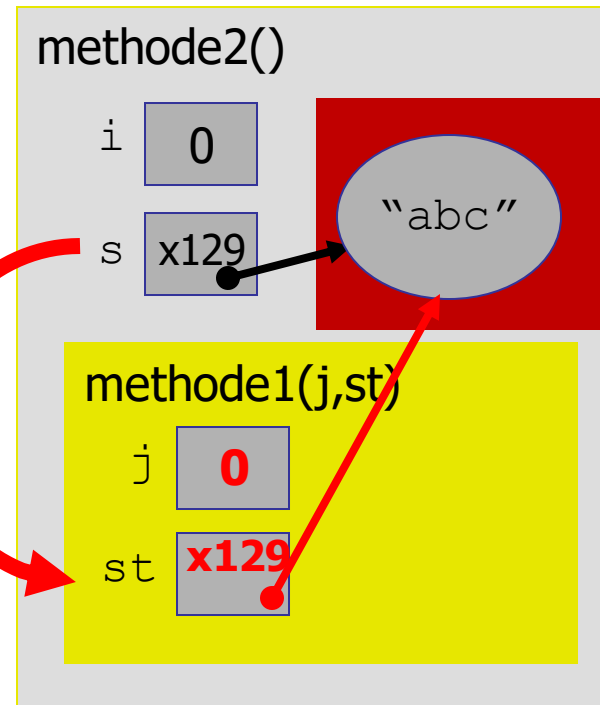


Méthodes en Java

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

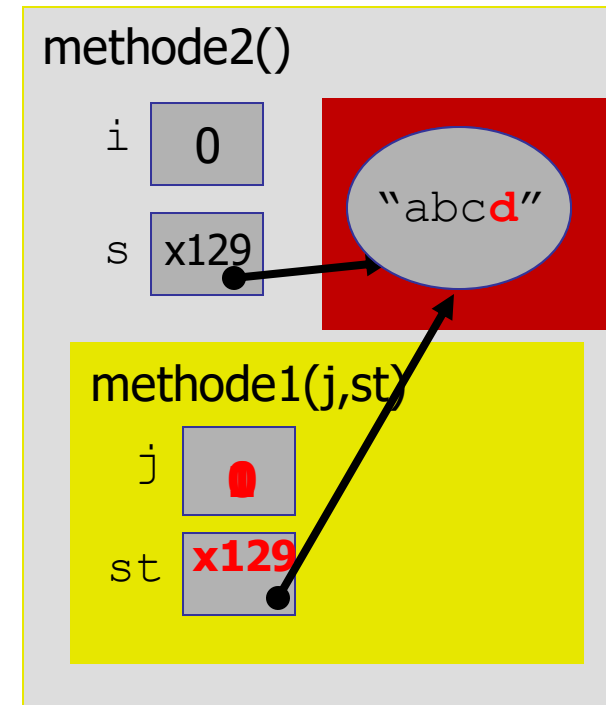




Méthodes en Java

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



Tableaux statiques en java

Déclaration et Création d'un tableau

tableau

```
int [] unTableau = new int [nbElt] ;
```

Type des
Composants

Type primitif ou Classe

nom de la
variable

Allocation de Mémoire pour:
les **valeurs** de type primitif.
ou les **références** de type Classe.

- Taille du tableau mémorisée dans :
 - `unTableau.length`
 - Nombre d'éléments
 - Dernier élément : `unTableau[unTableau.length-1]`
 - Non modifiable
`unTableau.length = 5 ;`



Tableau d'éléments de type primitif

```
int [] unTableau = new int [nbElt] ;
```

- `unTableau[indice]`

```
unTableau[i] = 10;
```

Éléments numérotés à partir de 0

// i compris entre 0 et nbElt-1

```
int x;
```

```
x = unTableau[i];
```

- Erreur classique : être en dehors des limites du tableau !

- Les indices débutent en 0

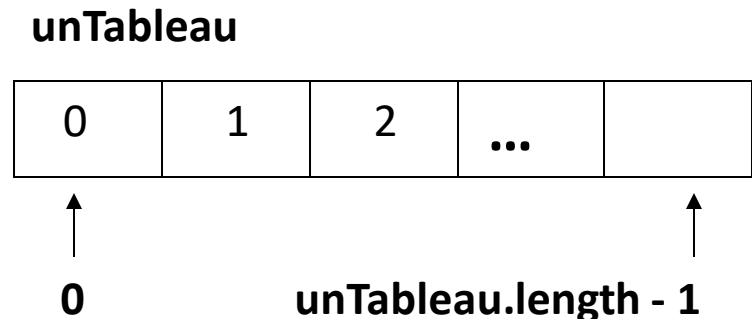


Tableau d'éléments de type primitif

```
int [] monTableau = { 1, 2, 4, 8, 16 };
```



1	2	4	8	16
---	---	---	---	----

Indice

0

4

Taille du tableau : 5
indice de 0 à 4

- Création du tableau et Initialisation des éléments avec les valeurs spécifiées

Manipulation de tableau

exemple Tableau d'entiers

```
int nbElt=10; //nb d'éléments
//Déclaration du tableau
int [] unTableau = new int [nbElt]
int indice, somme = 0 ;

//Saisie clavier des éléments du tableau
for ( indice = 0; indice < nbElt; indice++ ) {
    monTableau[indice] = Clavier.lireInt(("Entrez
    la valeur de l'element " +indice);}

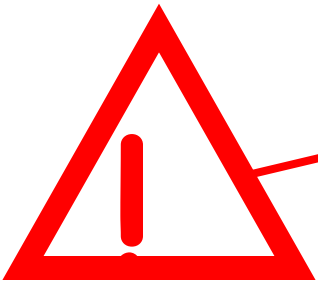
//Calcul de la somme des éléments du tableau
for ( indice = 0; indice < nbElt; indice++ ) {
    somme = somme + monTableau[indice];}

System.out.println("la somme vaut : " + somme );
```

Tableaux d'objets en JAVA

```
class Client {  
    String nom;  
    int age;  
    public Client(.....) ...  
    void vieillir() { ... }  
    ....  
}
```

```
class Agence {  
    String directeur;  
    String adresse;  
    Client[] lesClients;  
    public Agence( String directeur, String adresse)  
    {  
        ...  
        lesClients = new Client[500];  
        ...  
    }  
    void changerDirecteur(...) { ... }  
    void ajouterUnClient(...) { ... }  
}
```



Création du tableau pas des objets!!

Tableaux d'objets en JAVA

```
class Agence {  
    String directeur;  
    String adresse;  
    int nbClients = 0 ;  
    Client[] lesClients;  
    public Agence( String directeur, String adresse) {  
        this.directeur= directeur;  
        this.adresse=adresse;  
        lesClients = new Client[500]; ...}  
  
    void ajouterUnClient(Client unClient) {  
        lesClients[nbClients] = un Client;  
        nbClients ++ ;  
    }  
}
```

*Nombre d'éléments
du tableau*

*Instanciation du
du tableau*

*Méthode d'ajout
d'ajout de références*

Manipulation de tableau

Méthodes utilitaires

- La classe **Arrays** (du package `java.util`) définit une série de méthodes statiques utilitaires permettant de manipuler des tableaux :
 - Assignment d'une valeur à certains éléments d'un tableau .. `fill()`
 - Test d'égalité du contenu de deux tableaux..... `equals()`
 - Tri du contenu d'un tableau `sort()`
 - Conversion d'un tableau en liste (`List`) `asList()`
 - Recherche binaire dans un tableau trié `binarySearch()`
- La classe **System** (du package `java.lang`) contient une méthode statique `arraycopy()` permettant de copier les éléments spécifiés d'un tableau dans un autre tableau, à une position donnée (*Shallow copy*).
Le second tableau doit être du même type que le premier. Il peut même s'agir du même tableau.

TPCours – Exercice (7)

- Dans votre projet TPCours, Ajoutez un package tableaux.
- Définissez les classe Client, Agence et TestAgence présentes dans les diapos précédentes.
- Définissez une méthode **afficherClients** dans la classe Agence pour afficher la liste des noms des clients de l'agence (clients présents dans le tableau lesClients).
- Et testez son fonctionnement dans votre classe TestAgence



Tableaux dynamiques en java

- Collection ArrayList

Tableaux dynamiques en Java

Un tableau dynamique est un **objet** de la classe ArrayList
(**java.util.ArrayList**)

- ArrayList est une des nombreuses classes de l'API Java permettant de définir des **collections** ou containers d'objets
- Notion de liste de références d'objets
- Taille extensible en fonction des besoins
- Avant Java 5, ArrayList d'objets non différenciés (cf. compléments)

ArrayList : *la liste peut contenir des objets de différentes classes*

Tableaux dynamiques en Java

(à partir de Java 5)

A partir de la version 5, grâce à la généricité, Java permet de spécifier le type d'objets contenus dans un ArrayList lors de sa déclaration:

`ArrayList <String>` : la liste ne peut contenir que des chaînes

`ArrayList <Client>` : la liste ne peut contenir que des Clients



Paramètre de généricité

ArrayList

Déclaration et Instanciation d'un ArrayList

```
ArrayList<Element> a= new ArrayList<Element> ()
```

*a est un objet de type ArrayList contenant
des objets de type Element*

Exemples:

Création d'une liste vide de capacité initiale 10

```
ArrayList<String> phrase= new ArrayList<String> ()
```

Création d'une liste vide de capacité initiale 50

```
ArrayList<String> phrase= new ArrayList<String> (50)
```

Méthodes de manipulation d'un ArrayList

■ Ajout :

■ à la fin :

```
boolean add( Element obj )
```

■ à la position index :

```
void add (int index, Element obj,)
```

index est compris entre 0 et nombre d'éléments de la liste -1

■ Remplacement :

■ à la position index: renvoie l'élément précédemment situé à cette position

```
Element set(int index, Element obj,)
```

```
phrase.add ("Merci" );  
phrase.add ( " Monsieur" );  
phrase.add(1," beaucoup" );  
phrase.set( 2," Madame" );
```

Méthodes de manipulation d'un ArrayList

- Déterminer le nombre d'éléments (taille « utile »)

`int size ()` : renvoie toujours 0 après
la création de la liste

- Tester si la liste est vide
`boolean isEmpty ()`

- Obtenir un élément

`Element get(int index)`

- Rechercher
 - `boolean contains (Element obj)`
 - `int indexOf (Element obj)` : retourne -1 si la recherche n'aboutit pas
 - `int lastIndexOf (Element obj)` : index de la dernière occurrence

Méthodes de manipulation d'un ArrayList

- Suppression de l'élément situé à la position index

Element **remove** (int index)

Renvoie l'élément supprimé

- Vidage de l'arrayList

void **removeAll** ()

- Suppression de la première occurrence d'un objet

boolean **remove** (**Element** obj)

Le résultat est false si l'objet n'est pas trouvé

```
phrase.remove ( "beaucoup" );
```

Étapes de définition d'un ArrayList

1 - Déclaration

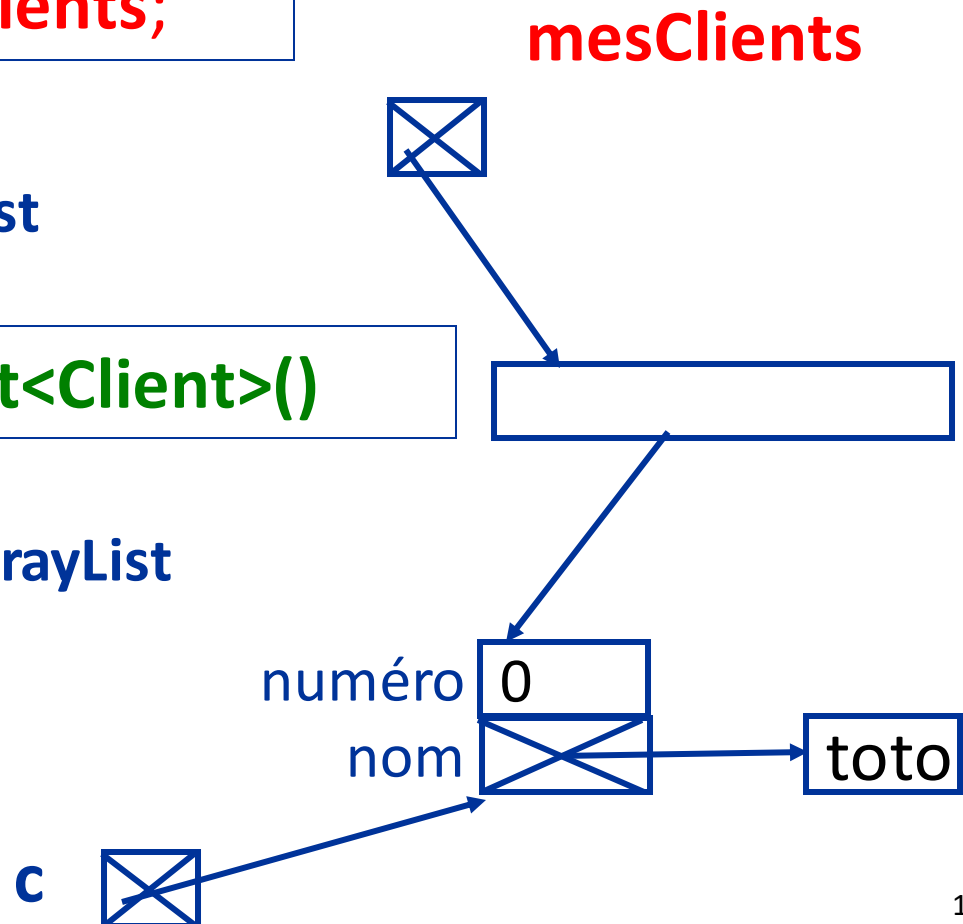
```
ArrayList<Client> mesClients;
```

2 – Création de l'objet ArrayList

```
mesClients = new ArrayList<Client>();
```

3 – Ajout d'un objet dans l'arrayList

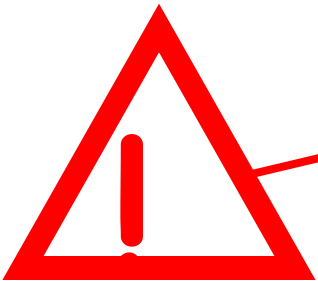
```
mesClients.add(c);
```



Liste d'objets en JAVA

```
class Client {  
    String nom;  
    int age;  
    public Client(.....  
    void vieillir() { ...  
    ....  
}
```

```
class      {  
    String directeur;  
    String adresse;  
    ArrayList<Client>    lesClients;  
    public Agence( String directeur, String adresse)  
    {  
        lesClients = new ArrayList<Client>();  
        ...}  
    void changerDirecteur(...) { ... }  
    void ajouterUnClient(...) { ... }  
}
```



Création de l'objet ArrayList!!

Liste d'objets en JAVA

```
class Agence {  
    String directeur;  
    String adresse;  
    int nbClients = 0 ;  
    ArrayList<Client> lesClients;  
    public Agence( String directeur, String adresse) {  
        this.directeur= directeur;  
        this.adresse=adresse;  
        lesClients = new ArrayList<Client>(); ...}  
  
    void ajouterUnClient(Client unClient) {  
        lesClients.add(unClient);  
        nbClients= lesClients.size();  
    }  
}
```

*Nombre d'éléments
du tableau*

*Instanciation du
du ArrayList*

*Méthode
d'ajout de références
dans la liste*

Liste d'objets en JAVA

```
class      {  
.....  
public void afficheListeNomsClients(){  
    System.out.println ("Liste des noms de Clients");  
    for (int i=0; i < lesClients.size() ; i++ )  
        System.out.println( (lesClients.get(i)).getNom());  
}
```

Accès au client d'indice i dans l'arrayList

Parcours d'unArrayList (à partir de Java 5)

```
ArrayList<Element> listeElement;  
for (Element var:listeElement){  
    //var prend successivement la valeur de chacun  
    des éléments de listeElement  
}
```

```
public void afficheListeNomsClients(){  
    System.out.println ("Liste des noms de Clients");  
    for (Client c:lesClients )  
        System.out.println(c.getNom());  
}
```



*Variable de parcours
de lesClients*

TPCours – Exercice (8)

- Définissez un package arraylist dans votre projet TPCours.
- Faites un copier/coller de vos classes du package tableai et modifiez les afin d'utiliser un ArrayList et non un tableau statique.
- On considère la méthode **moyenneAgeClients()** de la classe Agence.
- Cette méthode calcule la moyenne des âges des clients de l'agence et renvoie un résultat de type double.
- On suppose que la classe Client dispose d'une méthode **getAge()** renvoyant l'âge d'un client.

Définissez deux versions de la méthode **moyenneAgeClient**:

- une version utilisant une boucle for classique
- une version utilisant une boucle du type
for (Element var:listeElement)