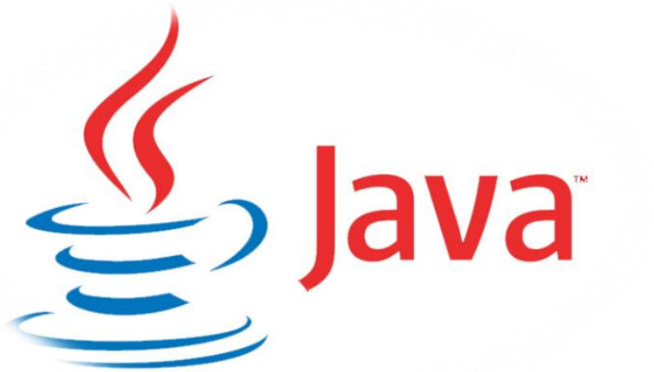
Classe & Objet



Prof: Mme Sara SAIB

Les principes de la POO: Programmation par objet

• Un objet est un ensemble de propriétés ayant des valeurs et des actions (opérations ou méthodes) agissant sur les valeurs de ces propriétés.

• Les objets communiquent entre eux par des messages. Un objet peut recevoir un message qui déclenche :

- une méthode qui modifie son état

et/ou

- une méthode qui envoie un message à un autre objet

Définition de la classe

Une classe est constituée de:

- Données ce qu'on appelle des **attributs**
- Procédures et/ou des fonctions ce qu'on appelle des **méthodes**

Une classe est un modèle de définition pour des objets

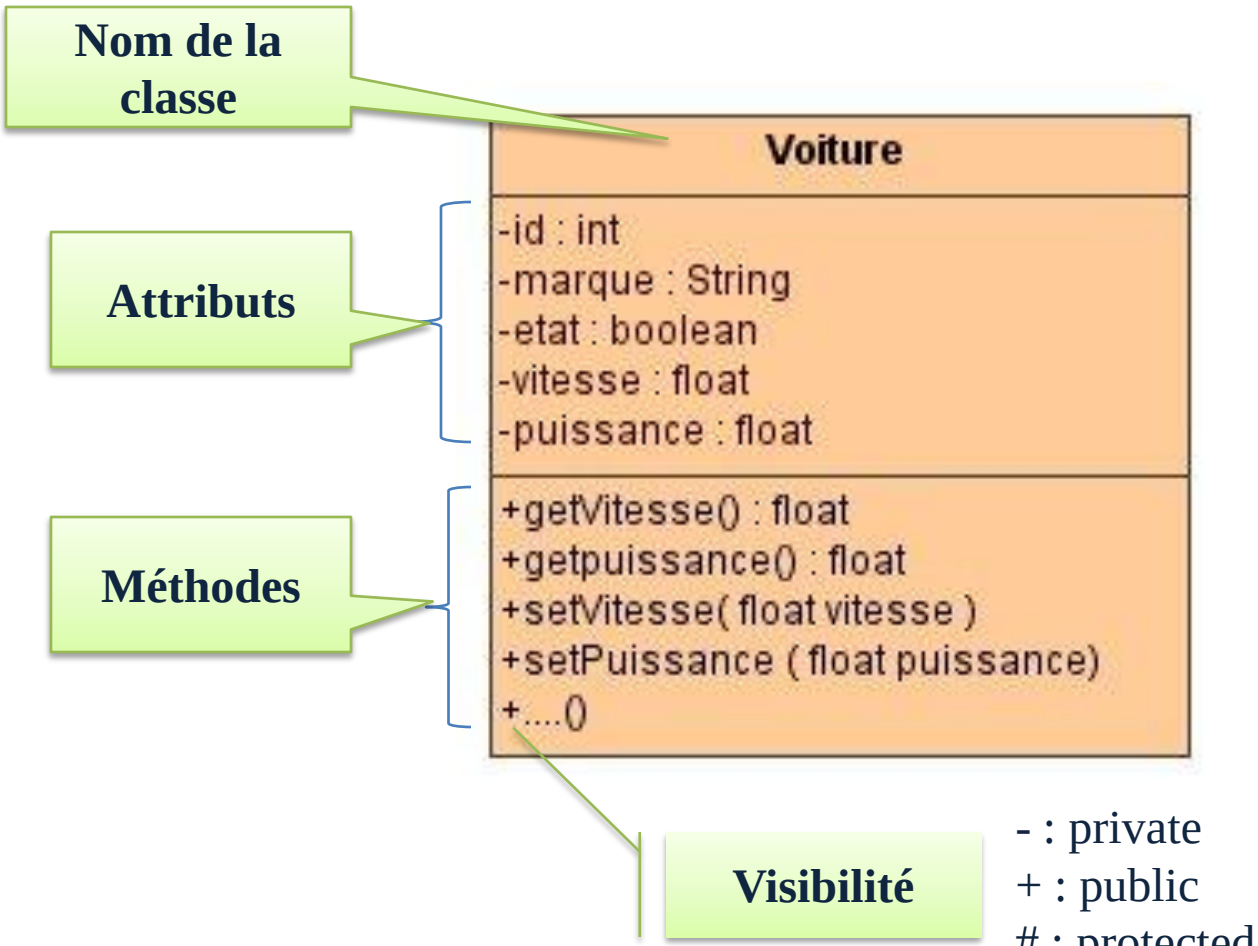
- Ayant même structure (même ensemble d'attributs)
- Ayant même comportement (même méthodes)
- Ayant une sémantique commune

Les **objets sont des représentations dynamiques du modèle** défini pour eux au travers de la classe (**instanciation)**

Une classe permet d'**instancier (créer) plusieurs objets**

Chaque objet est une instance d'une classe et une seule

Classe et notation UML



Codage de la classe Voiture

```
public class Voiture {
        private int id;
        private String marque;
        private float vitesse;
        private float puissance;
        private boolean etat;

        public Voiture(int id, String marque, float vitesse,
        float puissance, boolean etat){
        this.id=id;
        this.marque=marque;
        this.vitesse=vitesse;
        this.puissance=puissance
        this.etat=etat;
        }
        public int getId() {
                return id;
        }
        public String getMarque() {
                return marque;
        }
        public void setMarque(String marque) {
                this.marque = marque;
        }
```

Nom de la classe

Attributs

```
public float getVitesse() {
                return vitesse;
}
public void setVitesse(float vitesse) {
                this.vitesse = vitesse;
}
public float getPuissance() {
                return puissance;
}
public void setPuissance(float puissance) {
                this.puissance = puissance;
}
 public boolean isEtat() {
                return etat;
}
public void setEtat(boolean etat) {
                this.etat = etat;
}
```

Méthodes

```
}
```

Visibilité

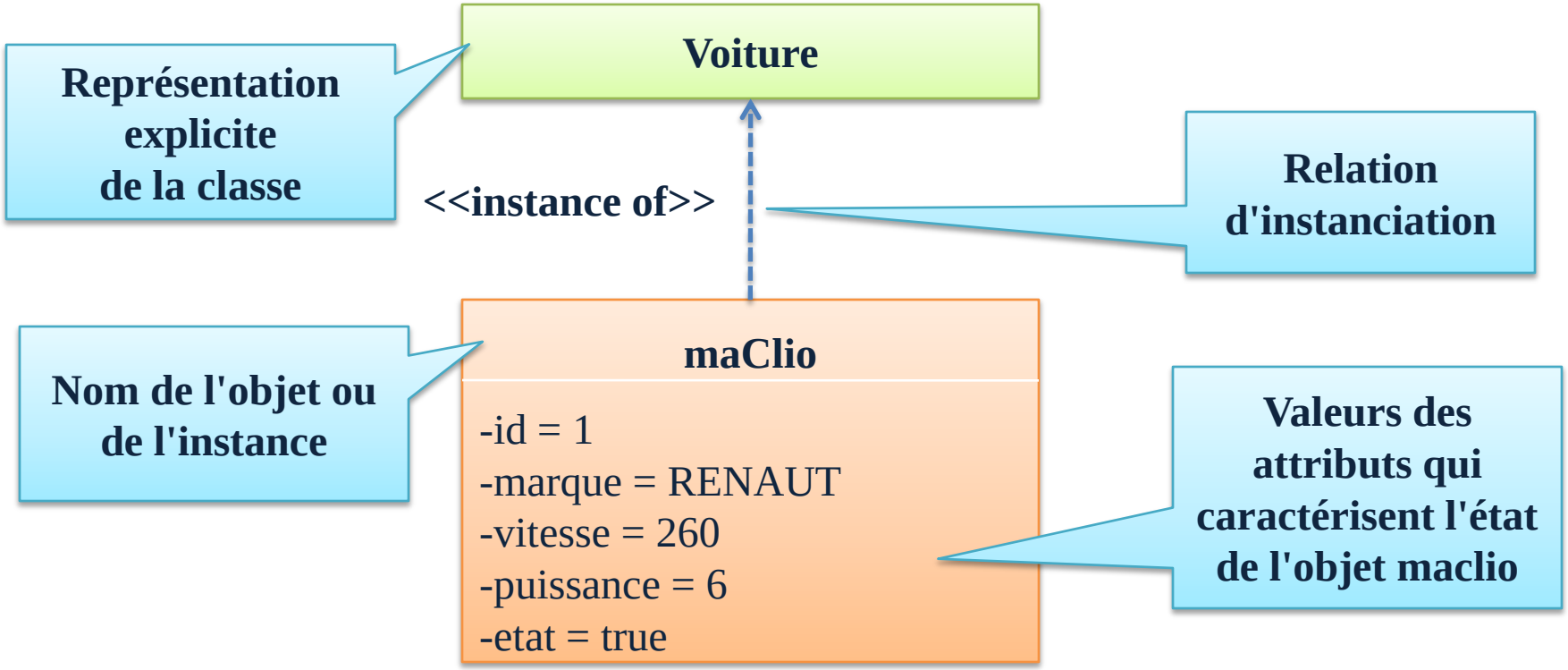
Définition d’un objet

Un objet est une instance d’une seule classe

- Il est conforme à la description que la classe fournit
- Il admet une valeur propre pour chaque attribut déclaré dans la classe
- Les valeurs des attributs caractérisent l’état de l’objet
- Possibilité de lui appliquer toute opération (méthode) définie dans la classe.
- Tout objet est manipulé et identifié par sa référence

Objet et notation UML

maClio est une instance de la classe Voiture



Déclaration des classes

Syntaxe générale

[modificateurs] **class** *nomClasse* ***[extends*** *nomSurClasse**]*** ***[implements*** *interface**]*** ***{***

[déclaration des attributs]

[déclaration de méthodes]

}

- Le mot clé **extends** permet de spécifier une superclasse éventuelle : ce mot clé permet de préciser la classe mère dans une relation d'héritage.

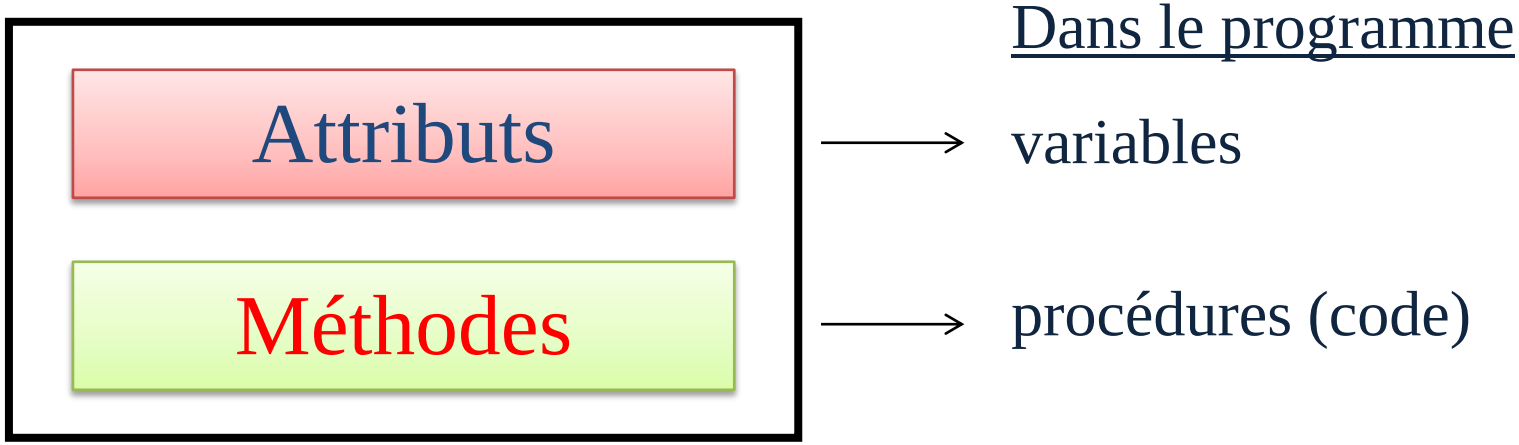
- Le mot clé **implements** permet de spécifier une ou des interfaces que la classe implémente.

Déclaration des classes: Les modificateurs

	Rôle
abstract	la classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée .
final	Les classes déclarées final ne peuvent pas avoir de classes filles.
private	la classe n'est accessible qu'à partir du fichier où elle est définie
	Elle est accessible uniquement à l'intérieur du même package
public	La classe est accessible partout

L’encapsulation

- Un attribut ou une méthode sont dits **privés** si leur utilisation est interdite en dehors de la classe.
- Un attribut ou une méthode sont dits **publics** si leur utilisation est autorisée en dehors de la classe



ENCAPSULATION

Le principe d'encapsulation est de déclarer les **attributs de façon privée** et les **méthodes de façon publique**

Encapsulation des données

Exemple de Classe Personne

```
public class Personne {
  private String nom, prenom, adresse ;
  private String  Tel ;
  private int age;

// méthodes d'accès aux données privées
  public String getNom() {
     return nom;
  }
  public String getPrenom() {
     return prenom;
  }
  // méthodes d'affectation aux données privées
  public void setNom(String lenom) {
     nom=lenom.toUpperCase();
  }
  ..........
} // fin classe Personne
```

toUpper Case(): une méthode qui retourne une chaîne égale à la chaîne convertie en majuscules.

Construction des objets

Les objets sont **des instances de classe**. (i.e. des exemplaires de classe).

En Java, **il ne suffit pas de nommer les variables objets**, il faut **les construire explicitement** et les initialiser.

Syntaxe

1- déclarer le nom d'un objet de la classe sans définir l'objet lui-même

nomClasse nomObjet;

2- construire l'objet

nomObjet = new constructeurClasse([liste de paramètres]) ;

1 et 2- il est possible de fusionner les 2 étapes

nomClasse nomObjet = new constructeurClasse([liste de paramètres]) ;

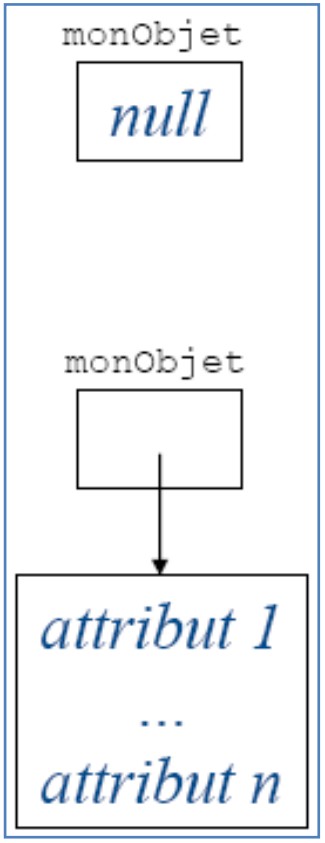
Construction des objets

Déclaration: Définition du nom de l'objet

- Un objet seulement déclaré vaut « **null** »
  (mot réservé du langage)

Création et allocation de la mémoire: création réelle de l'objet à l'aide du constructeur

- Appelle de méthodes particulières : les constructeurs
- La création réserve la mémoire et initialise les attributs



Construction des objets

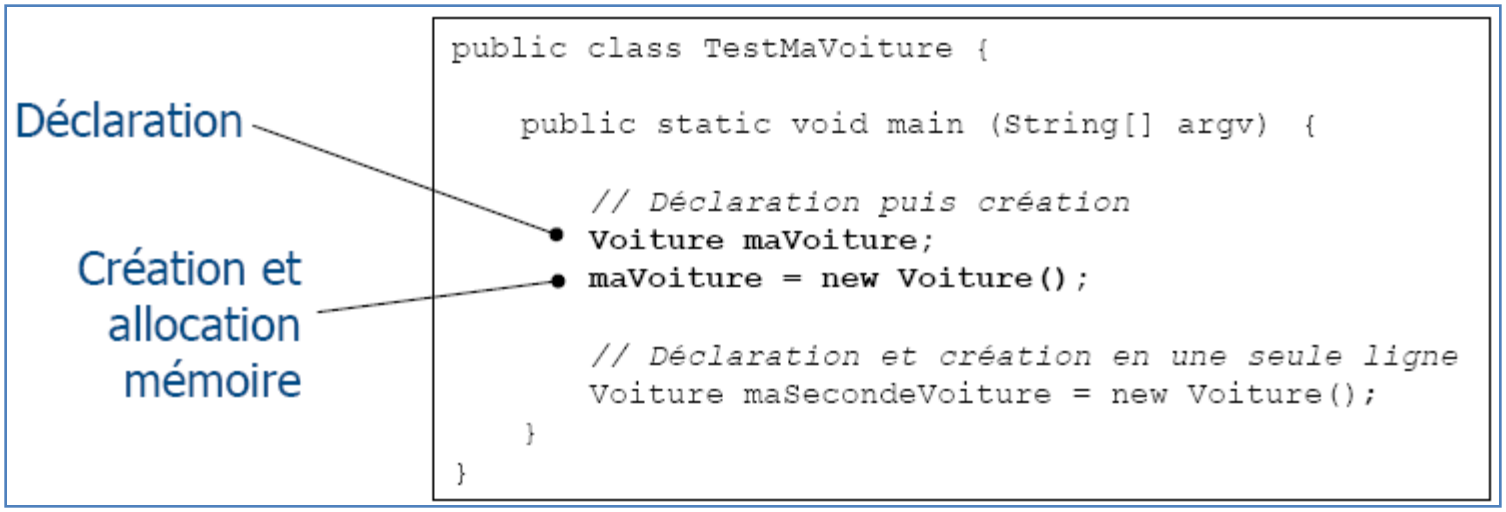
Exemple

Voiture maVoiture ;

maVoiture = new Voiture() ;

Ou bien

Voiture maVoiture = new Voiture() ;



Les constructeurs d’instances

Le **constructeur** d'un objet est une **méthode** spéciale **indispensable qui est appelée** automatiquement pour créer une instance.

Le constructeur doit respecter les règles suivantes :

- Il possède le **même nom** que la classe
- Il **n'est pas typé**
- Il ne possède pas de valeur de retour (même pas void).

Les constructeurs d’instances

- On peut définir plusieurs constructeurs, chargés d'initialiser différemment les attributs de l’objet.

- Deux cas peuvent se présenter :

1. Le programmeur n'a défini explicitement aucun constructeur. Alors, le système utilise le **constructeur par défaut** qui s'écrit avec le nom de la classe et des parentheses vides.

2. Le programmeur a défini un ou plusieurs constructeurs grâce à la possibilité de **surcharge** **(c'est-à-dire : ils diffèrent par le nombre ou le type de leurs arguments).** Dans ce cas le constructeur par défaut n'est plus utilisable. Le compilateur utilisera le constructeur adéquat en fonction des arguments.

Le constructeur par défaut

Si on utilise un constructeur par défaut (sans paramètre)

- On ne sait pas comment se construit l'objet
- Les valeurs des attributs au départ sont indéfinies et identiques pour chaque objet.

Rôle du constructeur en Java

- Effectuer certaines initialisations nécessaires pour le nouvel objet créé

Toute classe Java possède au moins un constructeur

- Si une classe ne définit pas explicitement de constructeur, un constructeur par défaut sans arguments et qui n'effectue aucune initialisation particulière est invoquée

Définitions de plusieurs constructeurs

```
// définition de différents constructeurs pour la classe Cercle
public class Cercle {
 // déclaration de 3 attributs de type double
 double x, y, r ;

 public double circonference() {
 return 2*Math.PI*r ;
 }
 public double surface {
 return Math.PI *r*r ;
 }
 public Cercle(double r) {
  this.r = r;
 }
 public Cercle(double xo, double yo , double rayon) {
  this.x = xo ; this.y = yo ; this.r = rayon ;
 }
 public Cercle() {
  this.r = 1 ;
 }
}
```

1. Initialise r (le rayon) à la valeur passée en paramètre du constructeur
2. Initialise x, y et r à la valeur passée en paramètre du constructeur
3. Initialise r à 1

Utilisation des objets: Accès aux attributs

Pour accéder aux données d'un objet on utilise une notation pointée

identificationObjet.nomAttribut

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture v1 = new Voiture();
        Voiture v2 = new Voiture();

        // Accès aux attributs en écriture
        v1.puissance = 110;

        // Accès aux attributs en lecture
        System.out.println("Puissance de v1 = "  + v1.puissance);
    }
}
```

Il n'est pas recommandé d'accéder directement aux attributs d'un objet

Envoi de messages: appel de méthodes

Pour « demander » à un objet d'effectuer un traitement il faut lui **envoyer un message**

Un message est composé de trois parties:

- Une référence permettant de désigner l'objet à qui le message est envoyé
- Le nom de la méthode ou de l'attribut à exécuter
- Les éventuels paramètres de la méthode

identificationObjet.nomDeMethode(« Paramètres éventuels »)

L'envoi de message est similaire à un appel de fonction

- Le code défini dans la méthode est exécuté
- Le contrôle est retourné au programme appelant

Envoi de messages: appel de méthodes

Ne pas oublier les parenthèses pour les appels des méthodes

Envoi d'un message à l'objet maVoiture : Appel d'un modificateur

Envoi d'un message à l'objet maVoiture : Appel d'un sélecteur

```
public class TestMaVoiture {

    /**
     * @param args
     */
    public static void main(String[] args) {

        //Décalaration puis Création
        Voiture maVoiture = new Voiture();
        //La voiture démarre
        maVoiture.setEtat(true);

        if(maVoiture.getVitesse() == 4)
        {
            System.out.println("Pas très rapide");
        }
        //La voiture accélère
        maVoiture.setVitesse(12.5f);

    }

}
```

Envoi de messages: passage de paramètres

Un paramètre d'une méthode peut être

- Une variable de type simple
- Une référence d'un objet typée par n'importe quelle classe

En Java tout est passé par valeur

- Les paramètres d'une méthode sont effectifs
- La valeur de retour d'une méthode est effective aussi (si différente de void)

L'objet courant

- L'objet « courant » est désigné par le mot clé **this**
- Permet de désigner l'objet dans lequel on se trouve

Ne pas tenter d'affecter une nouvelle valeur à this !!!!

```
this = ... ; // Erreur
```

Utilité de l'objet « courant »

- Rendre explicite l'accès aux propres attributs et méthodes définies dans la classe
- Passer en paramètre d'une méthode la référence de l'objet courant

L’objet courant: attributs et méthodes

L’objet courant désigne des variables ou méthodes définies dans une classe

```
public Voiture(float puissance, boolean etat) {
    this.puissance = puissance;
    this.etat = etat;
}
```

Désigne l'attribut etat

Désigne la variable etat

this n’est pas nécessaire lorsque les identificateurs de variables ne présentent aucun équivoque

La durée de vie d’un objet

Les objets ne sont pas des éléments statiques et leur durée de vie ne correspond pas forcément à la durée d'exécution du programme. La durée de vie d'un objet passe par trois étapes:

- **La déclaration de l'objet et l'instanciation grâce à l'opérateur new**
- **L'utilisation de l'objet en appelant ses méthodes**
- **La suppression de l'objet : elle est automatique en Java grâce à la machine virtuelle. La restitution de la mémoire inutilisée est prise en charge par le récupérateur de mémoire (garbage collector).**

L’utilisation d’un objet non construit provoque une exception de type NullPointerException

Cycle de vie d'un objet

1- Création de l'objet

- Utilisation d'un Constructeur
- L'objet est créé en mémoire et les attributs de l'objet sont initialisés

2- Utilisation de l'objet

- Utilisation des Méthodes et des Attributs (non recommandé)
- Les attributs de l'objet peuvent être modifiés
- Les attributs (ou leurs dérivés) peuvent être consultés

L'utilisation d'un objet non construit provoque une exception de type NullPointerException

3- Destruction et libération de la mémoire lorsque

- Utilisation(éventuel) d'un Pseudo-Destructeur
- L'objet n'est plus référencé, la place mémoire occupée est récupérée

Affectation d'objet: création d'objet identique

Affecter un objet

- « a = b » signifie a devient identique à b et toute modification de a entraîne celle de b
- a et b contiennent la même référence et pointent donc tous les deux sur le même objet : les modifications faites à partir d'une des variables modifient l'objet.

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture();

        // Déclaration d'une deuxième voiture
        Voiture maVoitureCopie;
        // Attention!! pour l'instant maVoitureCopie vaut null


        // Test sur les références.
        if (maVoitureCopie == null) {

            // Création par affectation d'une autre référence
            maVoitureCopie = maVoiture;
            // maVoitureCopie possède la même référence que maVoiture
        }
        ...
    }}
```

Déclaration

Affectation par référence

Les références et la comparaison d'objets

Comparer deux objets

- « a == b » retourne « true » si les deux objets sont identiques
- C'est-à-dire si les références sont les mêmes, **cela ne compare pas les attributs**

La méthode toString

La méthode **toString** est définie dans la classe Object, toutes les classes Java en hérite.

La **classe Object** est la **super classe de toutes les classes Java** : toutes ses méthodes (toString, equals …) sont donc héritées par toutes les classes.

La méthode toString renvoie le nom de la classe de l'objet concerné suivi de l'adresse de cet objet.

Lorsqu'on définit une classe, il peut être très utile de redéfinir la méthode toString afin de donner une description satisfaisante des objets de cette classe.

Renvoyer une chaîne de caractères servant à décrire l'objet concerné.

```
public String toString() {
      return " Voiture : puissance" + this.puissance + "   Démarrée :  " +
this.estdamarre;
}
```

La surcharge

La surcharge

La surcharge (**overloading**) appliquée à un constructeur ou à une méthode est **le processus permettant dans une même classe** de donner **le même nom à un constructeur ou une méthode**.

Le compilateur saura identifier exactement **la variante appelée, en fonction de sa signature**.

En effet, les méthodes surchargées doivent se distinguer par des signatures clairement différentes :

- des nombre d'arguments différents
- et/ou des types d'arguments différents

Des méthodes surchargées peuvent avoir des types de retour différents à condition qu'elles aient des arguments différents

La surcharge

Ces 2 constructeurs de la classe Personne sont ils compatibles ?

- public Personne (String leNom, String lePrenom, int lAge) { ..... }
- public Personne (String lePrenom, String leSexe, int lAge) { .... }

/* Modification de l'ordre des paramètres ci-dessous pour éviter une signature dupliquée, et donc un refus de compilation. */

public Personne (String lePrenom, int lAge, String leSexe)
{ ... // ce constructeur est alors accepté }

La surcharge

Exemple : une voiture surchargée

Des méthodes surchargées peuvent avoir des types de retour différents à condition qu'elles aient des arguments différents

```
public class Voiture {
    private int id;
    private String marque;
    private float vitesse;
    private float puissance;
    private boolean etat;

    public void accelre (float v){
        if(etat)
        {
            this.vitesse = this.vitesse + v;
        }
    }

    public void accelre (int v){
        if(etat)
        {
            this.vitesse = this.vitesse + v;
        }
    }
```

```
public class TestMaVoiture {

    /**
     * @param args
     */
    public static void main(String[] args) {

        //Décalaration puis Création
        Voiture maVoiture = new Voiture();

        maVoiture.accelre(12.56f);
        maVoiture.accelre(12);
        maVoiture.acc

    }

}
```

accelre(float v) void - Voiture
accelre(int v) void - Voiture

Attribut et méthode statiques

Attributs statiques (variables de classe)

- Il est parfois utile de spécifier qu'un **attribut doit posséder la même valeur pour toutes les instances**, autrement dit **avoir une seule valeur partagée par toutes les instances de la classe**. →**Attribut statique**

- Si on change la valeur de cet attribut, ce changement sera partagé et connu dans toutes les instances.

- Ce sont d'une certaine manière des **variables globales** au niveau d'une classe.

Attributs statiques (variables de classe)

Déclaration d’attributs statiques

Elles sont déclarées avec le modificateur **static**. C'est pourquoi une telle variable est appelée **variable de classe** ou **variable statique**.

Une variable de classe est une variable partagée par tous les objets de la classe (exemple le nombre de voiture)

Accès aux attributs statiques

Logiquement, ces **variables de classe sont préfixées par le nom de la classe**, puisqu'elles ne dépendent pas de l'instance.

Méthodes de classe (Méthode statique)

Utilisation

- Ce sont des méthodes qui ne s'intéressent pas à un objet particulier
- Elles sont utiles pour des calculs intermédiaires internes à une classe
- Elles sont utiles également pour retourner la valeur d'une variable de classe en visibilité private (privée)

Elles sont définies comme les méthodes d'instances, mais avec le mot clé **static**

```
public static double vitesseMaxToleree() {
    return vitesseMaxAutorisee*1.10;
}
```

Pour y accéder, il faut utiliser non pas un identificateur d'objet mais le nom de la classe (idem variables de classe)

Voiture.vitesseMaxToleree()

Méthodes de classe (Méthode statique)

Exemple : méthode de classe

```
public class Voiture {

    private static int nbVoitureCreees;
    ...

    public static int getNbVoitureCreees(){
        return Voiture.nbVoitureCreees;
    }
}
```

Déclaration d'une variable de classe privée. Respect des principes d'encapsulation.

Déclaration d'une méthode de classe pour accéder à la valeur de la variable de classe.

```
public class TestMaVoiture {

    public static void main (String[] argv)  {
        // Déclaration puis création de maVoiture
        Voiture maVoiture = new Voiture(2500);
        ...

        System.out.println("Nbre Instance :" +
                    Voiture.getNbVoitureCreees());
    }
}
```

Méthodes de classe (Erreur classique)

Exemple (suite) : méthode de classe

```
public class Voiture {

    private Galerie laGalerie;
    ...

    public Voiture(Galerie g) {
        laGalerie = g;
        ...
    }

    public static boolean isGalerieInstall() {
        return (laGalerie != null)
    }
}
```

Déclaration d'un objet Galerie non statique

Erreur : Utilisation d'un attribut non statique dans une zone statique

On ne peut pas utiliser dans une zone statique des variables d'instance

Notion de Package

- En Java, il existe un moyen de regrouper des classes voisines ou qui couvrent un même domaine : **ce sont les packages**.

- Syntaxe: *package nom-du-package*;

- D'une façon générale, l'instruction package associe toutes les classes qui sont définies dans un fichier source à un même package.

- Le mot clé package doit être **la première instruction** dans un fichier source et il ne doit être présent **qu'une seule fois dans le fichier source** (une classe ne peut pas appartenir à plusieurs packages).

Les conventions Java

- **Nommage**

Les noms utilisés doivent être explicites, c'est-à-dire le nom doit expliquer le contenu de l'objet, le rôle de la méthode…

- **Package/Paquetage**

Les noms des paquetage doivent être en minuscules.

//Correct **package com.monprojet.monpaquet;**	**//Incorrect** **package Com.MonProjet.MonPaquet;**

Les conventions de codage

Les noms des fonctions et des paramètres, ne doivent pas avoir de préfixe, commencent par une minuscule et chaque partie de mot est en majuscule.

```
public class MaClass
{
        private String maChaine;
        public void maMethode (String monParametre)
        {

        }
}
```

Les conventions de codage

- **Attributs, variables et paramètres**

  Les attributs des classes, les variables ainsi que les paramètres des méthodes doivent être en minuscules hormis les initiales des mots qui les composent (sauf le premier). Les variables de boucle doivent porter une seule lettre: i, j, k … Les signes dollars ($) et soulignement (_) sont interdits.

```
//Correct
Voiture prochaineVoiture = voitures.get (this.id + 1)
float  laTaille = 145.5;

//Incorrect
Voiture a = voitures.get (this.id + 1)
float  la_Taille = 145.5;
```

Les conventions de codage

Constantes

Les noms des constantes doivent être écrits entièrement en majuscules. Le séparateur de mot est le caractère de soulignement (underscore).

//Correct **final int LOG_CONSOLE = 1;**	**//Incorrect** **final int LOGCONSOLE = 1;** **final int console_Log = 1;** **final int Console_LOG = 1;**

Les conventions de codage

Déclaration

Les variables doivent être déclarées ligne par ligne.

L'initialisation doit se faire lors de la déclaration lorsque cela est possible.

Les noms des méthodes sont accolés à la parenthèse ouvrante listant leurs paramètres. Aucun espace ne doit y être inséré.

//Correct **int niveau = 10;** **void maMethode( ) {**	**//Incorrect** **int niveau;** **niveau = 10;** **void maMethode ( ) {**

Les conventions de codage

- **Instructions**

Une ligne de code ne peut contenir qu'une seule instruction.

//Correct **count++;** **i--;** **println (''Bonjour'');**	**//Incorrect** **count++; i--; println(''Bonjour '');**

Les conventions de codage

Conventions de noms

- CeciEstUneClasse
- celaEstUneMethode(…)
- jeSuisUneVariable
- jeSuisUnAttribut
- jeSuisUnParametre
- JE_SUIS_UNE_CONSTANTE

Un fichier par classe, une classe par fichier

- Classe « Voiture » décrite dans le fichier Voiture.java
- Il peut exceptionnellement y avoir plusieurs classes par fichier (cas des Inner classes)

Respecter les minuscules et les majuscules des noms