IIT
INSTITUT
INTERNATIONAL
TECHNOLOGIE
Université Nord Américaine privée

# Programmation Orientée Objet


Saïd Taktak

said.taktak@iit.ens.tn
said.taktak@gmail.com

# Les langages de programmation

- Il existe de nombreux langages de programmation :
  - Langages impératifs et procéduraux (C, Pascal, Fortran...)
  - logiques (Prolog...)
  - LangagLangageses fonctionnels (OCaml, Haskell, Erlang...)
  - Langages à pile (PostScript...)
  - **Langages à objets (Java, C++, C#...)**

- L'exécution se fait soit par :
  - Interprétation (Python, PHP, JavaScript, Ruby...)
  - Compilation et exécution (C++, C, Pascal...)
  - **Semi-compilation et exécution sur machine virtuelle (Java...)**

# La programmation orientée objet (POO)

- Les objectifs :
  - Faciliter le développement et l'évolution des applications ;
  - Permettre le travail en équipe ;
  - Augmenter la qualité des logiciels (moins de bugs).
- Solutions proposées :
  - Découpler (séparer) les parties des projets ;
  - Limiter (et localiser) les modifications lors des évolutions ;
  - Réutiliser facilement du code.

# La programmation orientée objet (POO)

- **Java**
- **C++ : très utilisé**
- **C# : langage de Microsoft (appartient à .NET)**
- **Objective C : langage utilisé par Apple**
- **PHP : langage très utilisé sur le Web**
- **Python**
- **Ruby**
- Eiffel
- Ada
- Smalltalk
- ...

# Java c’est quoi ?

- **Un langage** : Orienté objet fortement typé avec classes
- **Un environnement d’exécution (JRE)** : Une machine virtuelle et un ensemble de bibliothèques
- **Un environnement de développement (JDK)** : Un compilateur et un ensemble d’outils

# Java pourquoi ?

Java est devenu aujourd'hui l'un des langages de programmation les plus utilisés.

Il est incontournable dans plusieurs domaines :

- **Systèmes dynamiques :** Chargement dynamique de classes
- **Internet :** Les *Applets java*
- **Systèmes communicants :** *RMI, Corba, EJB, etc.*

# Java *versus C++*

- Filiation historique :
  - **1983 (AT&T Bell) : C++**
  - **1991 (Sun Microsystems) : Java**
- Java est très proche du langage C++ (et donc du langage C).
- Toutefois Java est plus simple que le langage C++, car les points "critiques" du langage C++ (ceux qui sont à l'origine des principales erreurs) ont été supprimés.
- Cela comprend :
  - Les pointeurs
  - La surcharge d'opérateurs
  - L'héritage multiple

# Java *versus C++*

- De plus,
  - Tout est dynamique : les instances d'une classe sont instanciées dynamiquement.
  - La libération de mémoire est transparente pour l'utilisateur. Il n'est pas nécessaire de spécifier de mécanisme de destruction.
- La libération de l'espace mémoire est prise en charge un gestionnaire appelé

  garbage collector chargé de détecter les objets à détruire.

**Notes**

- gain de fiabilité (pas de désallocation erronée).
- a un coût (perte en rapidité par rapport au C++).

# Java *versus C++*

- Une fois achevée la production du logiciel, un choix doit être fait entre fournir le source ou le binaire pour la machine du client.

- Généralement, une entreprise souhaite protéger le code source et distribuer le code binaire.

- Le code binaire doit donc être portable sur des architectures différentes (processeur, système d'exploitation, etc.).

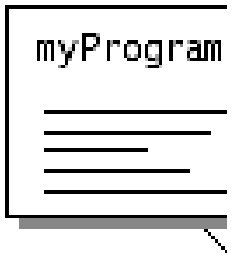
# Java *versus C++*

- À l'instar du compilateur C, le compilateur C++ produit du code natif, *i.e., qu'il produit un exécutable propre à l'environnement de* travail ou le code source est compilé.

- On doit donc créer les exécutables pour chaque type d'architecture potentielle des clients.

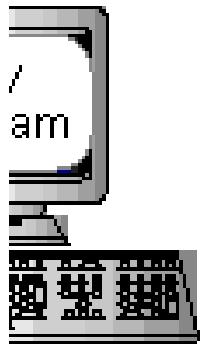
# Java *versus C++*

- En Java, le code source n'est pas traduit directement dans le langage de l'ordinateur.
- Il est d'abord traduit dans un langage appelé "bytecode", langage d'une machine virtuelle (JVM – Java Virtual Machine) définie par Sun.




**Portabilité**

Le *bytecode* généré par le compilateur ne dépend pas de l'architecture de la machine où a été compilé le code source, c'est-à-dire que les bytecodes produits sur une machine pourront s'exécuter (au travers d'une machine virtuelle) sur des architectures différentes.



# Java *versus C++*

- Le langage Java est :
  - « C-like » : Syntaxe familière aux programmeurs de C
  - Orienté objet : Tout est objet, sauf les types primitifs (entiers, flottants, booléens, ...)
  - Robuste : Typage fort, pas de pointeurs, etc.
  - Code intermédiaire : Le compilateur ne produit que du bytecode indépendant de l'architecture de la machine où a été compilé le code source

**Note**

Java perd (un peu) en efficacité par rapport à C++// mais gagne (beaucoup) en portabilité.

# L’environnement actuel Java 8 Standard Edition J2SE

- L’outil de base : le JDK (Java Development Kit) de SUN :
  - http://java.sun.com.
  - gratuit.
  - comprend de nombreux outils :
    - le compilateur.
    - le compilateur à la volé "JIT".
    - le débogueur.
    - le générateur de documentation
- Des environnements de développements gratuits
  - NetBeans : http://www.netbeans.org/
  - Eclipse : http://www.eclipse.org/

# Mon premier programme en Java

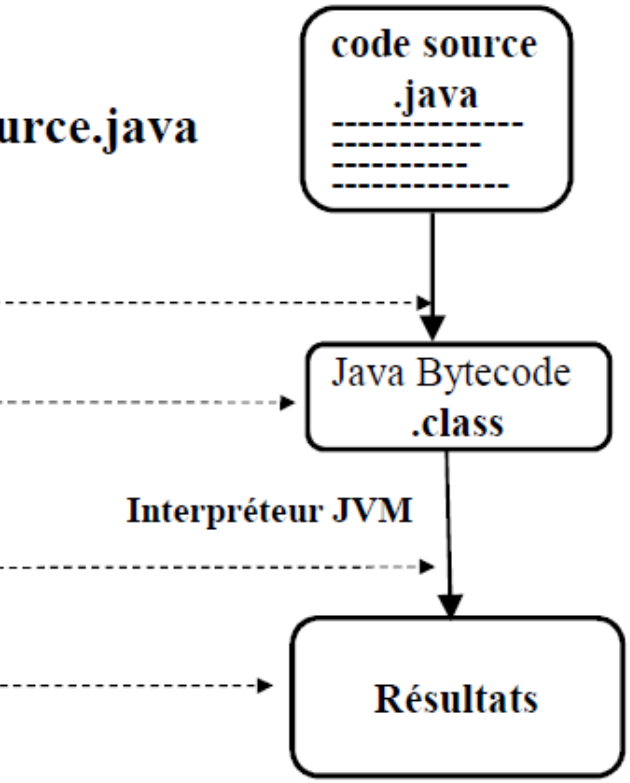
❑ **Les étapes d'exécution d'un programme java**

1. **Écriture de programme** : **code-source.java**
2. ***Compilation***

   *javac code-source.java*

   Ceci génère ***code-source.class***
3. ***Exécution***

   *java code-source*
4. **Résultat de l'exécution**



# Mon premier programme en Java

- Le programme HelloWorld.java

```
class HelloWorld {
  public static void main(String arg[]) {
    System.out.println("Hello world !");
  }
}
```

- Compilation et exécution

  >javac HelloWorld.java

  >java HelloWorld

  >HelloWorld !

# Commentaires

Les commentaires permettent de rajouter du texte dans votre programme sans que le compilateur ne le considère comme du code :

```
class HelloWorld {
  public static void main(String arg[]) {
    /* Commentaire
       sur plusieurs lignes.
    */
    // sur une seule ligne.
    System.out.println("Hello world!");
  }
}
```

# Instructions, Variables

Les données sont manipulées à l'aide de variables et d'expressions :

```
class Exemple {
  public static void main(String arg[]) {
    int a = 5;    // Déclaration d'une variable
    int b = 9;    // Déclaration d'une deuxième variable
    b = b + a;    // L'expression "b+a" est évaluée
                  // et le résultat est affecté à "b"
    a = 2*b - 3; // 25 est affecté à "a"
    System.out.println(a); // Affichage du contenu de "a"
    System.out.println(b); // Affichage du contenu de "b"
  }
}
```

Notez que les instructions sont séparées par des points-virgules.

# Les Variables

Une variable possède un nom, un type et une valeur.

```
int a = 3;   // Déclaration d'une variable
             // de nom "a", de type "int" (entier)
             // avec une valeur initiale égale à 3.
```

Il est ensuite possible de changer la valeur de la variable :

```
int a = 3;
a = 12;      // Affectation d'une nouvelle valeur
```

En Java, le symbole = désigne l'affectation et n'a pas la même signification qu'en mathématiques.

Lors de la compilation, Java vérifie que chaque expression située à droite de l'affectation est compatible avec le type de la variable.

# Les Types Primitifs

| | | | | |
|---|---|---|---|---|
| byte | entier | 8 bits | -128 à 127 | 0 |
| short | entier | 16 bits | -32768 à 32767 | 0 |
| int | entier | 32 bits | $-2^{31}$ à $2^{31}-1$ | 0 |
| long | entier | 64 bits | $-2^{63}$ à $2^{63}-1$ | 0 |
| float | flotant | 32 bits | | 0.0 |
| double | flotant | 64 bits | | 0.0 |
| char | caractère | 16 bits | caractères Unicode | \u0000 |
| boolean | boolean | 1 bit | false ou true | false |

# Les Opérateurs

| | | |
|---|---|---|
| Négation | `-a` | Opposé de a |
| Addition | `a + b` | Somme de a et b |
| Soustraction | `a - b` | Différence de a et b |
| Multiplication | `a * b` | Produit de a et b |
| Division | `a / b` | Quotient de a et b |
| Modulo | `a % b` | Reste de a divisé par b |

| | | |
|---|---|---|
| non | `!a` | `true` si a est égal à `false` |
| et | `a && b` | `true` si a et b sont égaux `true` |
| ou | `a \|\| b` | `true` si a ou b est égal `true` |

| | | |
|---|---|---|
| égal | `a == b` | `true` si a est égal à b |
| différent | `a != b` | `true` si a est différent de b |
| plus petit | `a < b` | `true` si a est strictement plus petit que b |
| plus grand | `a > b` | `true` si a est strictement plus grand que b |
| inférieur ou égal | `a <= b` | `true` si a est plus petit ou égal à b |
| supérieur ou égal | `a >= b` | `true` si a est plus grand ou égal à b |

# Incrémentation et affectation

| | | |
|---|---|---|
| Pre-incrémente | `++a` | Incrémente a de 1, puis retourne a |
| Post-incrémente | `a++` | Retourne a, puis incrémente a de 1 |
| Pré-décrémente | `--a` | Décrémente a de 1, puis retourne a |
| Post-décrémente | `a--` | Retourne a, puis décrémente a de 1 |

| | | |
|---|---|---|
| Affectation | `a=b` | Affecte b à a |
| Addition | `a+=b` | Ajoute b à a |
| Soustraction | `a-=b` | Soustrait b à a |
| Multiplication | `a*=b` | Multiplie a par b |
| Division | `a/=b` | Divise a par b |

# Les instructions conditionnelles if

La syntaxe :

```
if (condition) instruction;
/* ou */
if (condition) instruction; else instruction;
```

Exemple :

```
boolean a = true;
boolean b = false;
int c = 10, d = 12, e;
if (a || b) e = c + d; else e = c - d;
if (a && b) e++; else e--;
```

# Les instructions conditionnelles switch

La syntaxe :

```
switch (expression) {
  case valeur1 : instruction; instruction; /*...*/ break;
  case valeur2 : instruction; instruction; /*...*/ break;
  /* ... */
  default : instruction; instruction; /*...*/ break;
}
```

Exemple :

```
int a = 2;
switch (a) {
  case 1 : System.out.println("un"); break;
  case 2 : System.out.println("deux"); break;
  case 3 : System.out.println("trois"); break;
  default : System.out.println(a); break;
}
```

# Les boucles while et do/while

La syntaxe :

```
while (condition) instruction;
do instruction; while (condition);
```

Exemple :

```
int a = 3;
while (a <= 10) a++;
do a--; while (a>=3);
```

# Les boucles for

La syntaxe :

```
for (initialisation; condition; incrementation)
    instruction;
```

Exemple :

```
for (int i = 0; i < 3; i++) System.out.println(i);
for (int i = 10; i >= 10; i--) System.out.println(i);
for (int i = 0; i <= 10; i+=2) System.out.println(i);
```

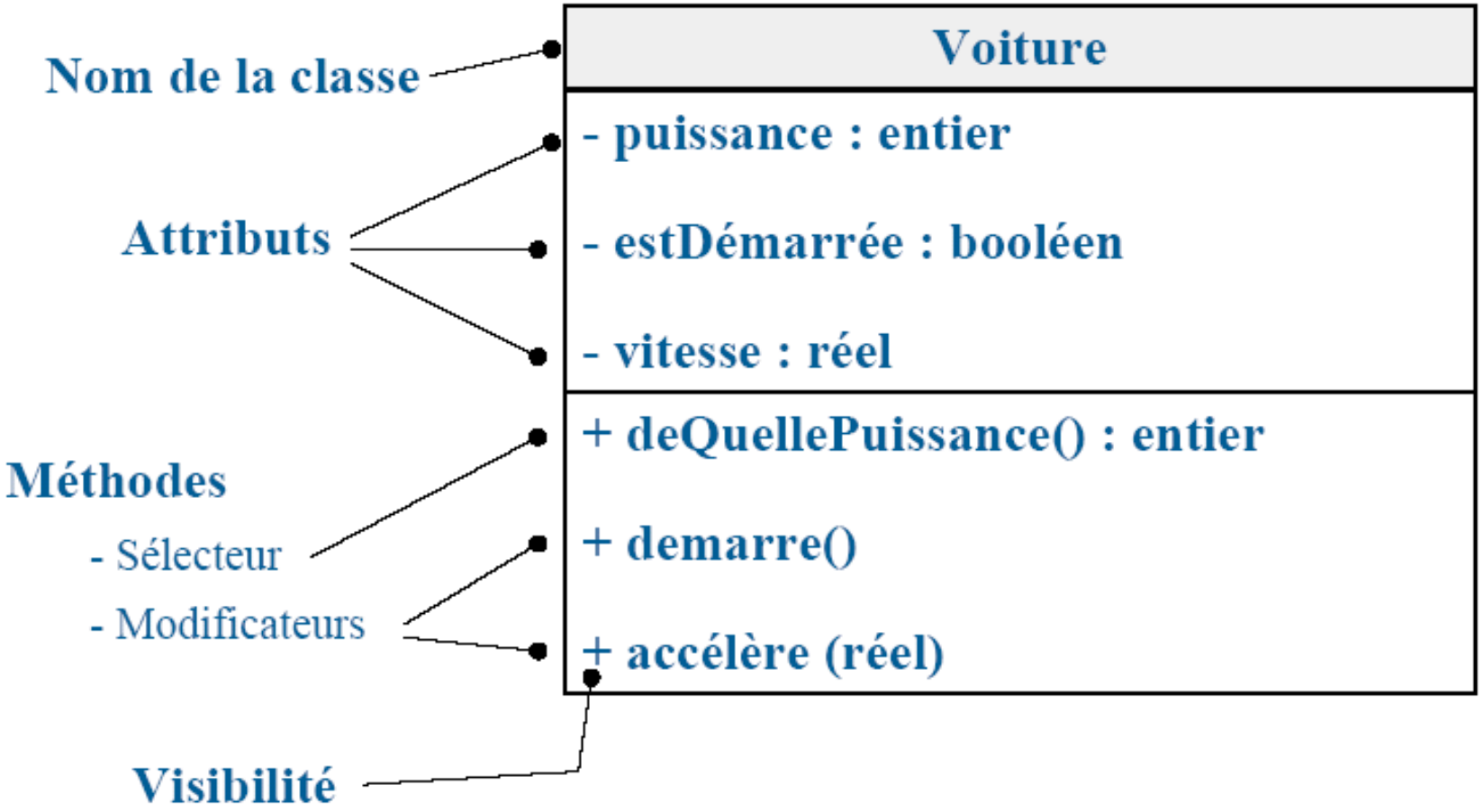
Équivalence avec le `while` :

```
initialisation;
while (condition) { instruction; incrementation; }
```

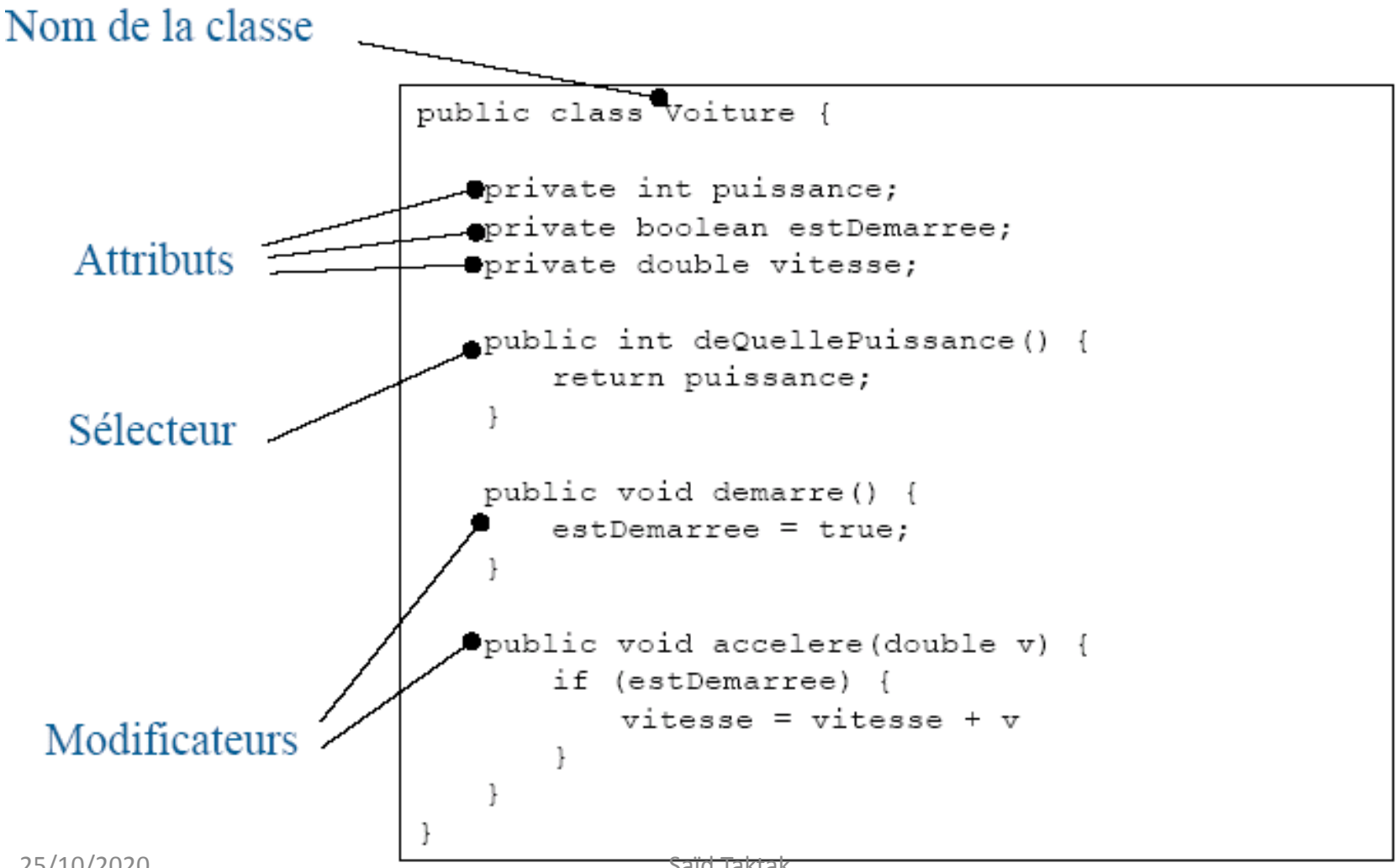
# Classe : Définition

- Une classe est constituée :
  - 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 (**instanciation**), du modèle défini pour eux au travers de la classe
  - Une classe permet d'**instancier** (créer) plusieurs objets
  - Chaque objet est instance d'une classe et une seule

# Classe : Notations



# Codage de la classe « voiture »



# Classe : Attributs

- Caractéristique d'un attribut :
  - Variables « globales » de la classe
  - Accessibles dans toutes les méthodes de la classe

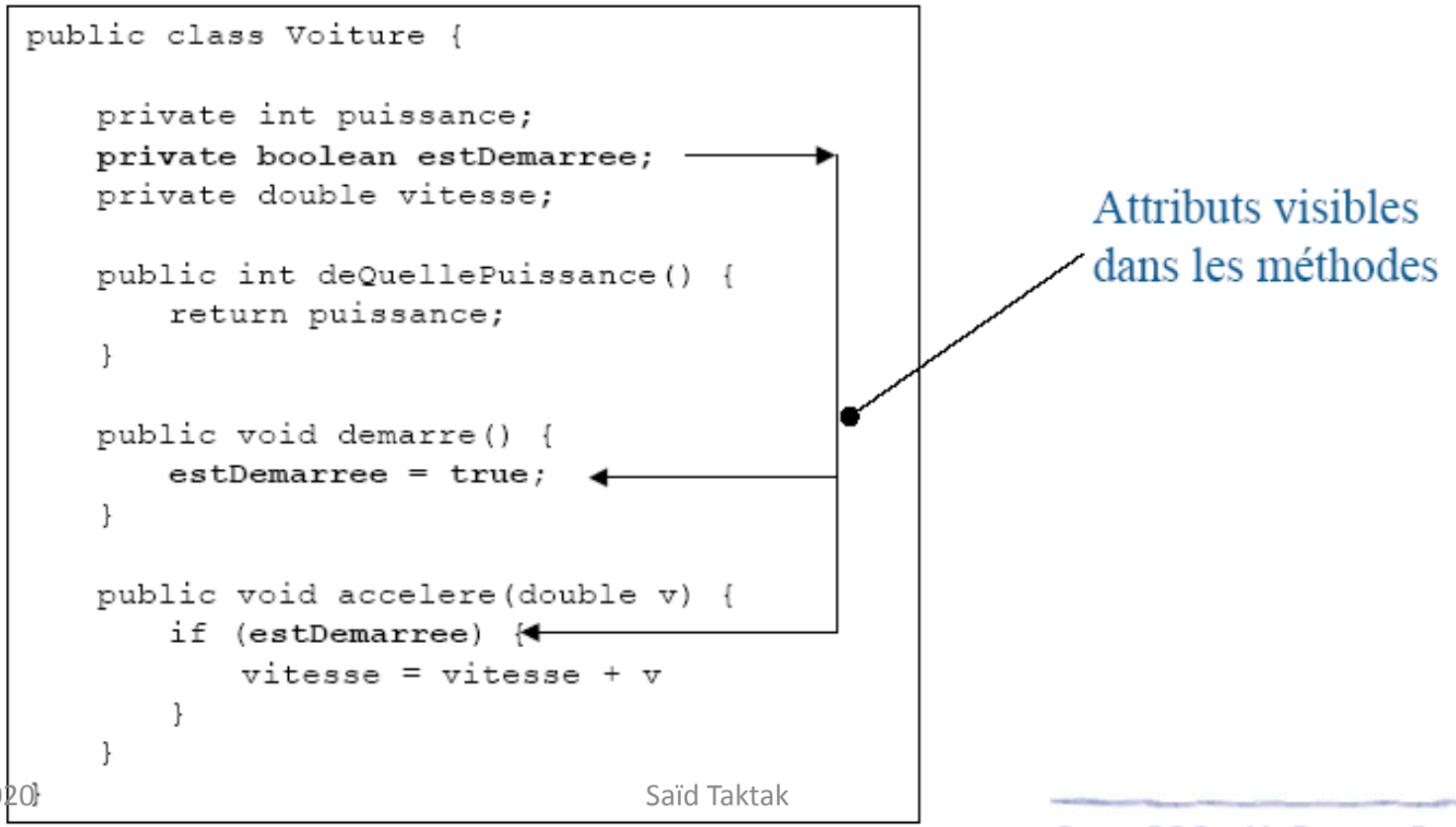
```
public class Voiture {

    private int puissance;
    private boolean estDemarree;
    private double vitesse;

    public int deQuellePuissance() {
        return puissance;
    }

    public void demarre() {
        estDemarree = true;
    }

    public void accelere(double v) {
        if (estDemarree) {
            vitesse = vitesse + v
        }
    }
}
```



# Classe : Attributs et variables

- Caractéristique d'une variable :
  - Visible à l'intérieur du bloc qui le définit

```
public class Voiture {

    private int puissance;
    private boolean estDemarree;
    private double vitesse;

    public int deQuellePuissance() {
        return puissance;
    }

    public void demarre() {
        estDemarree = true;
    }

    public void accelere(double v) {
        if (estDemarree) {
            double avecTolerance;
            avecTolerance = v + 25;
            vitesse = vitesse + avecTolerance
        }
    }
```

Variable visible uniquement dans cette méthode

Variable peut être définie n'importe où dans un bloc

# Conventions en Java

- Conventions de noms
  - CeciEstUneClasse
  - celaEstUneMethode(…)
  - jeSuisUneVariable
  - 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 fichi
    (cas des *Inner classes*)

**Respecter les minuscules et les majuscules des noms**

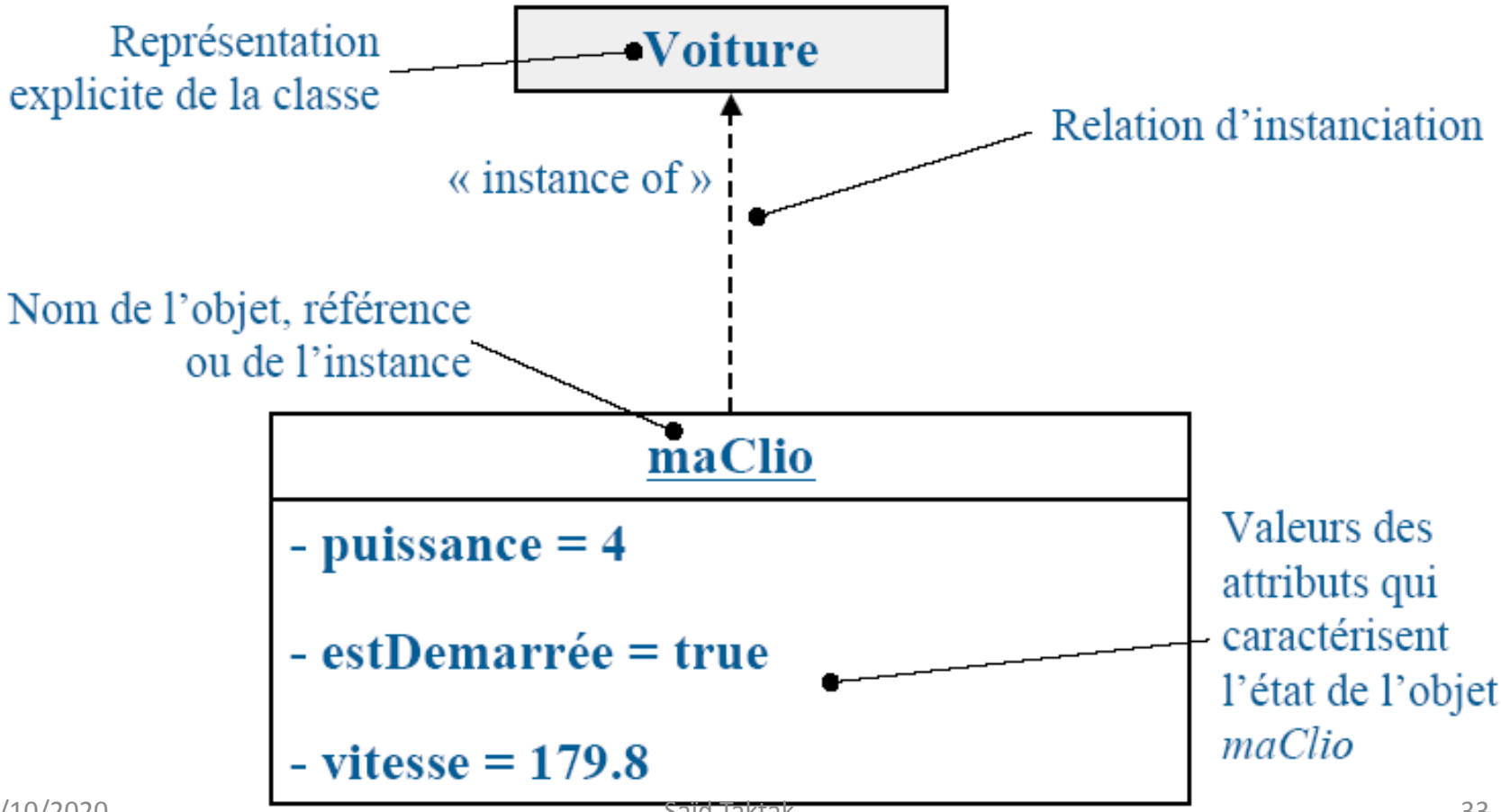
# Objet : Définition

- Un objet est **instance** d'une seule classe :
  - Se conforme à la description que celle-ci fournit
  - Admet une valeur propre à l'objet 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
  - Utilisation de pointeur caché (plus accessible que le C++)
  - On parle indifféremment d'**instance**, de **référence** ou d'**objet**

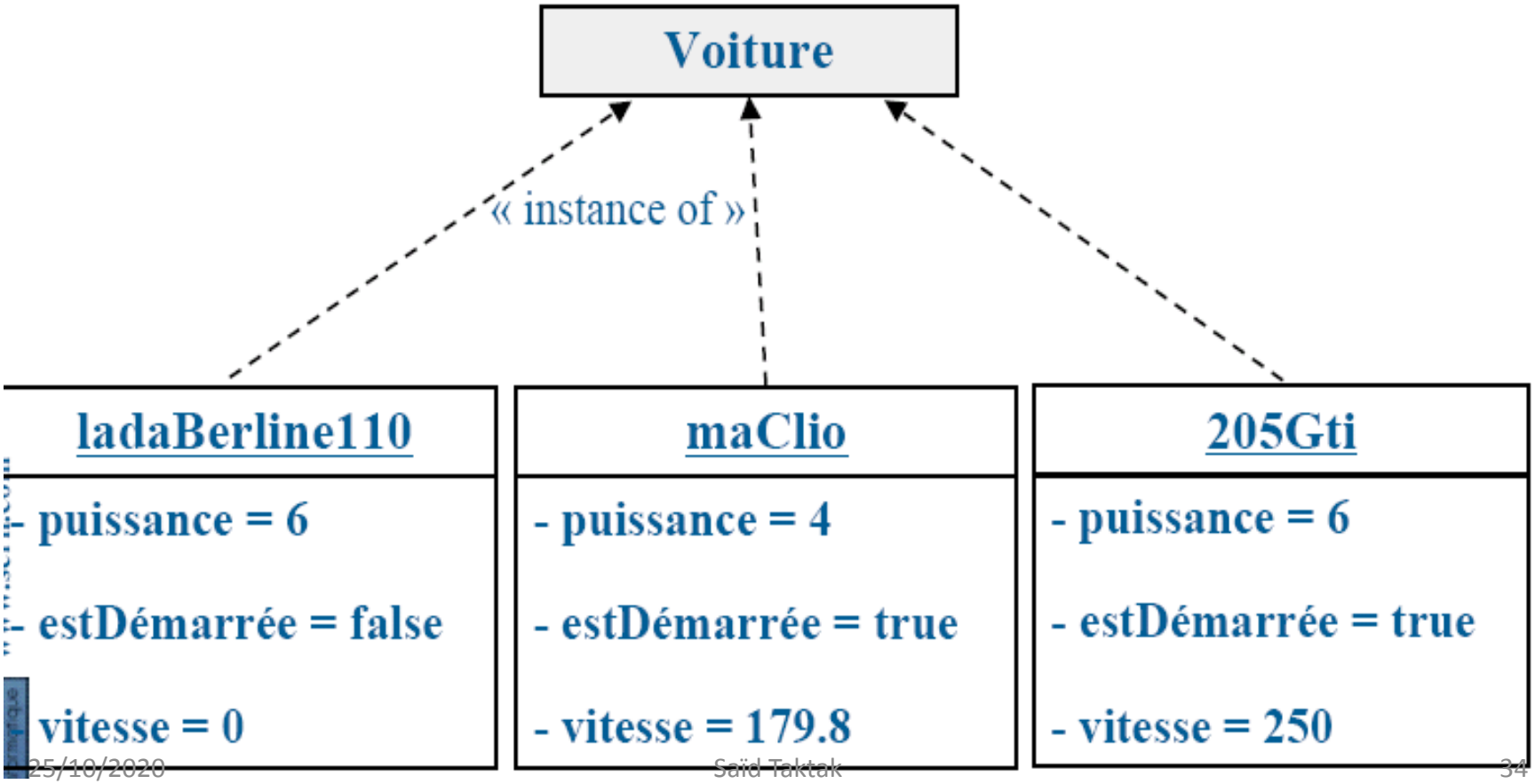
# Objet : Notations

- maClio est une instance de la classe *Voiture*



# Etats des objets

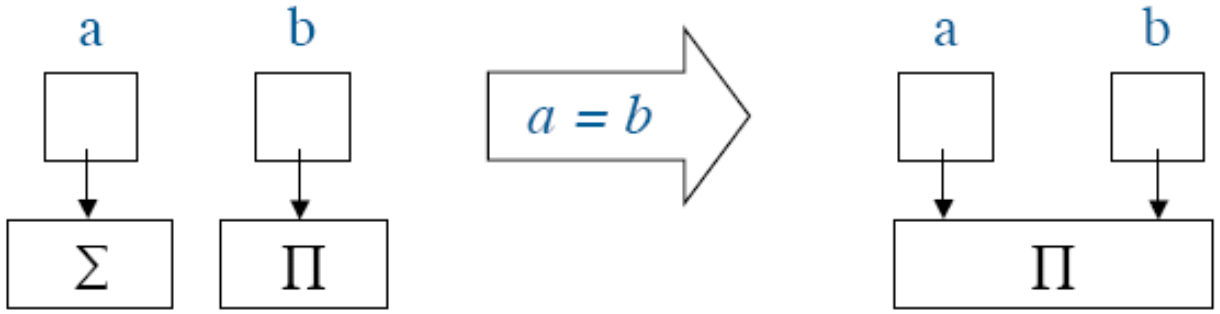
- Chaque objet qui est une instance de la classe *Voiture* possède ses propres valeurs d'attributs



# Affectation et comparaison

- Affecter un objet
  - « a = b » signifie a devient identique à b
  - Les deux objets a et b sont identiques et toute modification de a entraîne celle de b

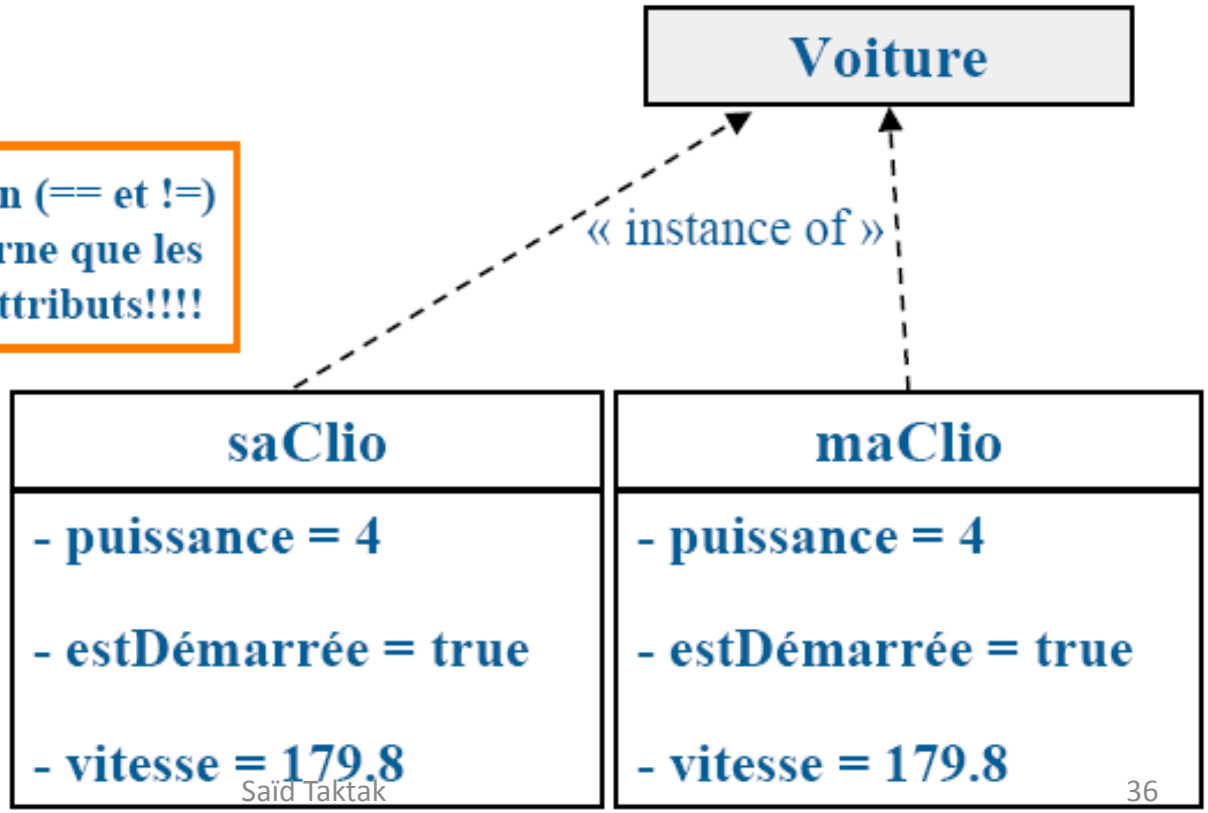
- Comparer deux objets
  - « a == b » retourne « true » si les deux objets sont identiques
  - C'est-à-dire si les références sont les mêmes, cela ne compare pas les attributs



# Affectation et comparaison

- L'objet **maClio** et **saClio** ont les mêmes attributs (états identiques) mais ont des références différentes
  - **maClio != saClio**

**Le test de comparaison (== et !=) entre objets ne concerne que les référence et non les attributs!!!!**



# Cycle de vie d'un objet

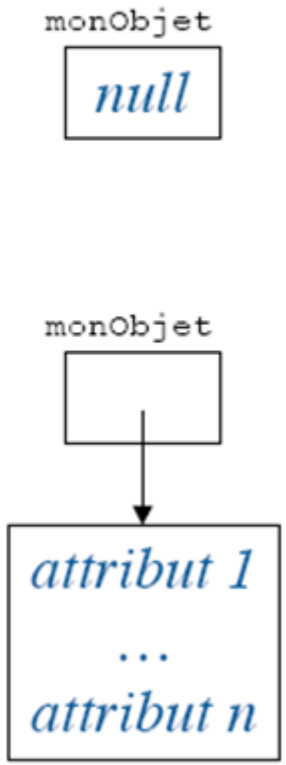
- Création
  - Usage d'un Constructeur
  - L'objet est créé en mémoire et les attributs de l'objet sont initialisés
- Utilisation
  - Usage des Méthodes et des Attributs (non recommandé)
  - Les attributs de l'objet peuvent être modifiés
  - Les attributs (ou leurs dérivés) peuvent être consultés

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

- Destruction et libération de la mémoire lorsque :
  - Usage (éventuel) d'un *Pseudo-Destructeur*
  - L'objet n'est plus référencé, la place mémoire qu'il occupait est récupérée

# Création d'objet

- La création d'un objet à partir d'une classe est appelée une **instanciation**.
- L'objet créé est une **instance** de la classe
- Déclaration
  - Définit le nom et le type de l 'objet
  - Un objet seulement déclaré vaut « **null** » (mot réservé du langage)
- Création et allocation de la mémoire
  - Appelle de méthodes particulières : les constructeurs
  - La création réservent la mémoire et initialise les attributs



# Création d'objet

- La création d'un objet est réalisée par **new** Constructeur(paramètres)
  - Il existe un constructeur par défaut qui ne possèdent pas de paramètre (si aucun autre constructeur avec paramètre n'existe)

**Les constructeurs portent le même nom que la classe**

Déclaration

Création et allocation mémoire

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

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

        // Déclaration et création en une seule ligne
        Voiture maSecondeVoiture = new Voiture();
    }
}
```

# Création d'objet

- Exemple : construction d'objets

```
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

# Constructeur de « Voiture »

- Actuellement
  - On a utilisé le constructeur par défaut sans paramètre
  - On ne sait pas comment se construit la « Voiture »
  - Les valeurs des attributs au départ sont indéfinies et identiques pour chaque objet (puissance, etc.)

**Les constructeurs portent le même nom que la classe et n'ont pas de valeur de retour**

- Rôle du constructeur en Java
  - Effectuer certaines initialisations nécessaire 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

# Constructeur de « Voiture »

- Le constructeur de « Voiture »
  - Initialise « vitesse » à zéro
  - Initialise « estDémaree » à faux
  - Initialise la « puissance » à la valeur passée en paramètre du constructeur

Constructeur avec un paramètre

```
public class Voiture {

    private int puissance;

    private boolean estDemarree;

    private double vitesse;

    public Voiture(int p) {
        puissance = p;
        estDemaree = false;
        vitesse = 0;
    }
    ...
}
```

# Construire une voiture de 7CV

- Création de la « Voiture » :
  - Déclaration de la variable « maVoiture »
  - Création de l'objet avec la valeur 7 en paramètre du constructeur

Déclaration

Création et allocation mémoire avec Voiture(int)

```
public class TestMaVoiture {

    public static void main(String[] argv)   {

        // Déclaration puis création
        Voiture maVoiture;

        maVoiture = new Voiture(7);

        Voiture maSecVoiture;
        // Sous entendu qu'il existe
        // explicitement un constructeur : Voiture(int)

        maSecVoiture = new Voiture(); // Erreur
    }
}
```

# Constructeur sans argument

- Utilité :
  - Lorsque l'on doit créer un objet sans pouvoir décider des valeurs de ses attributs au moment de la création
  - Il remplace le constructeur par défaut qui est devenu inutilisable dès qu'un constructeur (avec paramètres) a été défini dans la classe

```
public class Voiture {

    private int puissance;
    private boolean estDemarree;
    private double vitesse;

    public Voiture() {
        puissance = 4;
        estDemaree = false;
        vitesse = 0;
    }

    public Voiture(int p) {
        puissance = p;
        estDemaree = false;
        vitesse = 0;
```

```
public class TestMaVoiture {

    public static void main (String[] argv) {

        // Déclaration puis création
        Voiture maVoiture;
        maVoiture = new Voiture(7);
        Voiture maSecVoiture;
        maSecVoiture = new Voiture(); // OK
    }
}
```

# Accès au 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**

# 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 »)
```

- Envoi de message similaire à un appel de fonction
  - Le code défini dans la méthode est exécuté
  - Le contrôle est retourné au programme appelant

# Appel de méthodes

**Ne pas oublier les parenthèses pour les appels aux méthodes**

| Voiture |
| --- |
| - … |
| + deQuellePuissance() : entier<br>+ demarre()<br>+ accélère (réel)<br>+ … |

```
public class TestMaVoiture {

    public static void main (String[] argv)  {

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

        // La voiture démarre
        maVoiture.demarre();

        if (maVoiture.deQuellePuissance() == 4)  {
            System.out.println("Pas très Rapide…");
        }

        // La voiture accélère
        maVoiture.accélère(123.5);
    }
```

Envoi d'un message à l'objet *maVoiture*

Appel d'un modificateur

Envoi d'un message à l'objet *maVoiture*

Appel d'un sélecteur