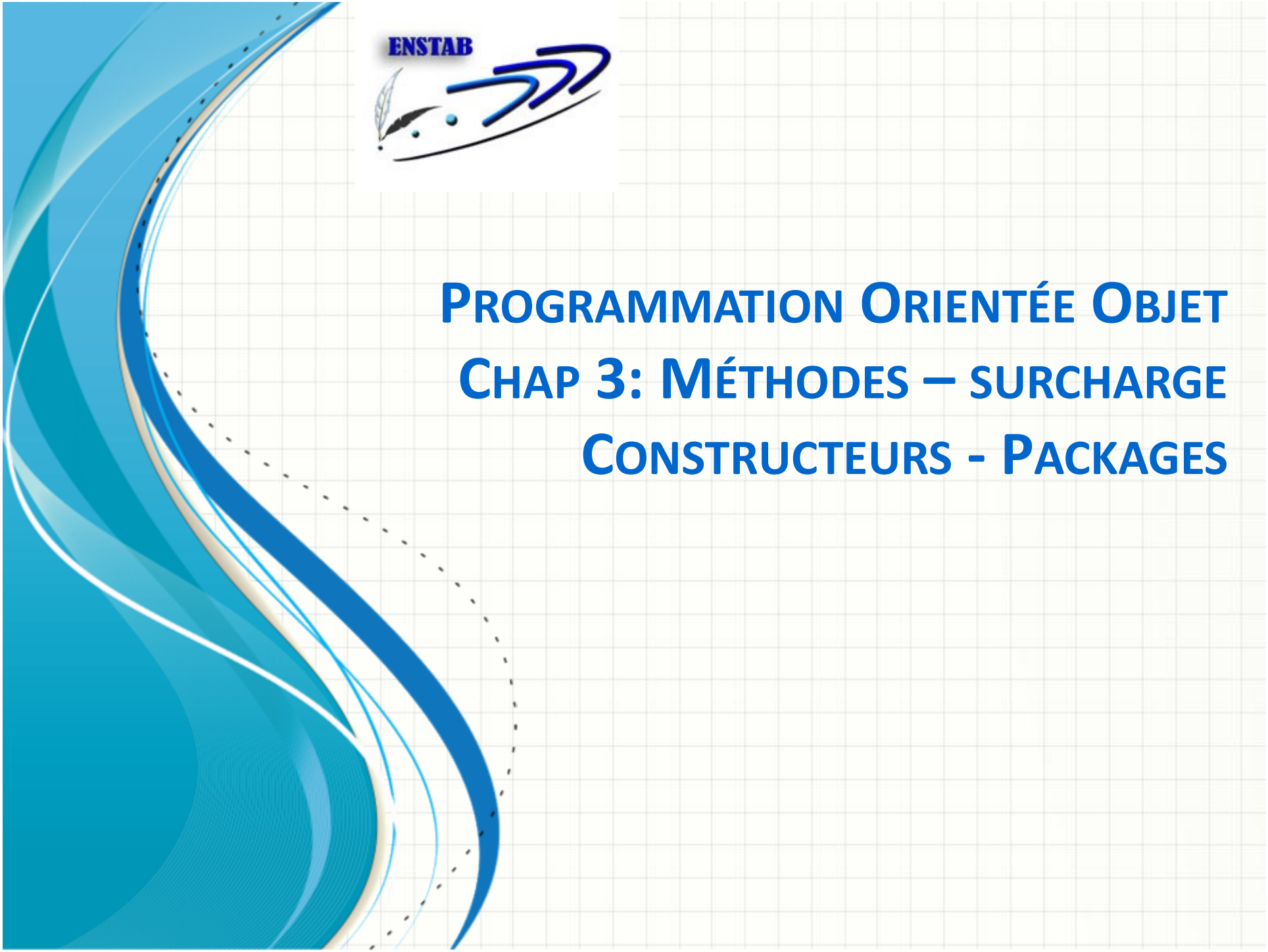


PROGRAMMATION ORIENTÉE OBJET
CHAP 3: MÉTHODES – SURCHARGE CONSTRUCTEURS - PACKAGES

Méthodes

```
public void rouler(double distance){
    kilometrage += distance;
}
```

- Dans l'entête de cette méthode nous trouvons:
  - public: il correspond au modificateur d'accès de la méthode rouler(). Ce modificateur peut être private ou protected.
  - void: type de retour de la méthode
  - rouler: nom de la méthode
  - double distance: un paramètre formel de la méthode précédé par son type.

Méthodes: surcharge

- En POO, il est possible d'appeler plusieurs fonctions par le même nom, du moment que celles-ci ont **des arguments différents : en type et/ou en nombre**.
- Ce principe est appelé **surcharge de fonctions**. Il permet de donner le même nom à des fonctions comportant des paramètres différents et simplifie donc l'écriture de fonctions ayant des paramètres différents mais qui sont **sémantiquement similaires**.
- Une méthode est déterminée par ce que l'on appelle sa **signature**, c'est-à-dire:
  - **Son nom**
  - **Ses paramètres**
- Il est ainsi possible de définir une fonction réalisant la même opération sur des variables différentes en nombre et/ou en type.

Méthodes: surcharge

- Exemple:

Voiture.java

```
public double prixTtc(){
    return prix * 1.19;
}

public double prixTtc(double taxe){
    return prix * (1+taxe/100);
}
public double prixTtc(int remise, double taxe){
    return prix * (1+taxe/100) * (100-remise)/100;
}
```

- Dans cet exemple, trois méthodes différentes s'appellent *prixTtc.*

Méthodes: surcharge

MainClass.java

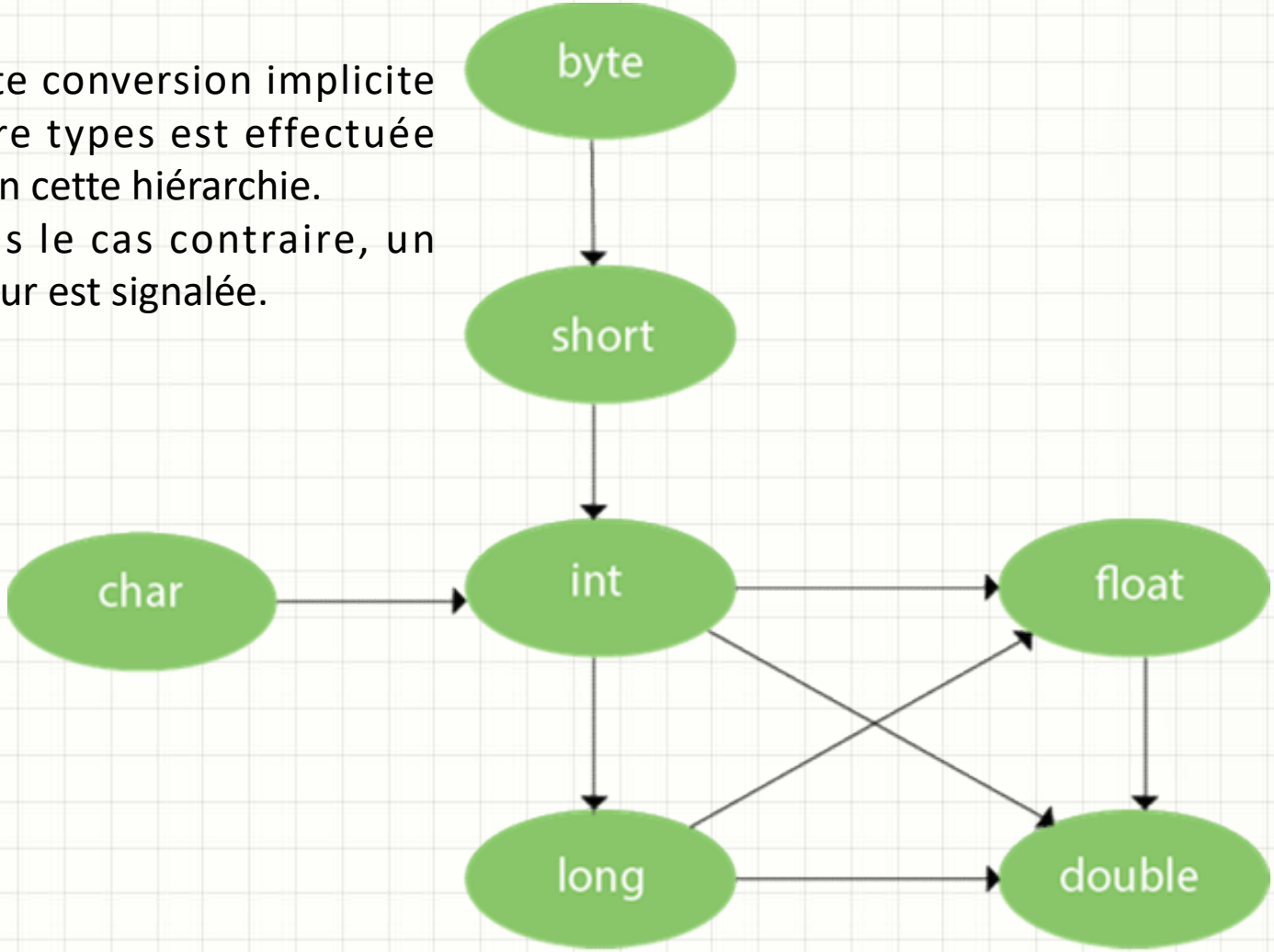
```
public class MainClass {
    public static void main(String[] args){
        Voiture v2 = new Voiture();
        double pTtc1, pTtc2, pTtc3;
        v2.setPrix(30);
        pTtc1 = v2.prixTtc();
        pTtc2 = v2.prixTtc(5.5);
        pTtc3 = v2.prixTtc(20, 19 );
        System.out.println("Le prix de v2 en HT: "+v2.getPrix()+
         "\nVariante 1 TTC: "+pTtc1+
         "\nVariante 2 TTC: "+pTtc2+
         "\nVariante 3 TTC: "+pTtc3);
   }
}
```

Le prix de v2:
HT: 30.0
Variante 1 TTC: 35.699999999999996
Variante 2 TTC: 31.65
Variante 3 TTC: 28.559999999999995

Nous remarquons que la 3ème variante de la méthode *prixTtc* est appelée pour le calcul de *pTtc3* même si le 2ème argument est de type int car il y a une conversion implicite qui est effectuée par le compilateur du type int au type double.

Méthodes: surcharge

Cette conversion implicite entre types est effectuée selon cette hiérarchie.
Dans le cas contraire, un erreur est signalée.



Méthodes: Surcharge

MainClass.java

```
public class MainClass {
    public static void main(String[] args){
        Voiture v2 = new Voiture();
        double pTtc1, pTtc2, pTtc3, pTtc4;
        v2.setPrix(30);
        pTtc1 = v2.prixTtc();
        pTtc2 = v2.prixTtc(5.5);
        pTtc3 = v2.prixTtc(20, 19 );
        pTtc4 = v2.prixTtc(20.5, 3.5);
        System.out.println("Le prix de v2:\nHT: "+v2.getPrix()+
                "\nVariante 1 TTC: "+pTtc1+
                "\nVariante 2 TTC: "+pTtc2+
                "\nVariante 3 TTC: "+pTtc3+
                "\nVariante 4 TTC: "+pTtc4);
    }
}
```

Error:(9, 28) java: incompatible types: possible lossy conversion from double to int

Constructeurs

- Dans la classe Voiture, nous avons fait appel à la fonction configurer pour initialiser les attributs d'un objet de type Voiture.
- Mais il faut l'appeler au moment opportun pour éviter d'utiliser un objet Voiture dont les attributs ne sont pas initialisés.

```
public void configurer(String m, int a, double km, double px){
        marque = m;
        annee = a;
        kilometrage = km;
        prix = px;
    }
```

- Pour automatiser cette démarche, on a recours à la notion de **« constructeur »**.

Constructeurs

- Un constructeur est une méthode sans valeur de retour.
- Il porte le même nom que la classe.
- Son rôle est d'initialiser les attributs d'un objet.
- Il peut disposer d'un nombre quelconque d'arguments (éventuellement aucun).
- En Java, il existe 2 types de constructeurs:
  - Un constructeur paramétrique.
  - Un constructeur par défaut.
- Comme pour le cas de méthodes, une même classe peut avoir plusieurs constructeurs (surcharge).

Constructeurs

- Considérons ce constructeur paramétrique :

Voiture.java

```
public class Voiture {
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;

    public Voiture(String m, int a,
double k, double p) {
        marque = m;
        annee = a;
        kilometrage = k;
        prix = p;
    }
// …
}
```

- Quand on déclare un constructeur paramétrique, on doit l'utiliser pour créer les objets.

MainClass.java

```
public class MainClass {
    public static void main(String[]
args){

        Voiture v2 = new Voiture();

        Voiture v3 = new
Voiture("Toyota", 2010, 85000, 26);
 }
}
```

Constructeurs

- Par contre, on peut rajouter, grâce au mécanisme de surcharge, un 2ème constructeur qui soit paramétrique ou par défaut:

Voiture.java

```
public class Voiture {

// ... Déclaration des attributs
   public Voiture(String m, int a, double k, double p)
{
                marque = m;
                annee = a;
                kilometrage = k;
                prix = p;
   }
   public Voiture(double p) {
                prix = p;
                kilometrage = 0;
                annee = 2019;
                marque = "Volvo";
   }

   public  Voiture(){
                marque = "Ford";
                annee = 2017;
                kilometrage = 100000;
                prix = 30;
   }
// ...
}
```

Constructeurs paramétriques

Constructeur par défaut

Constructeurs

MainClass.java

- Dans ce cas, on peut créer des objets de types Voiture de plusieurs manières différentes:

```
public class MainClass {
    public static void main(String[] args){
        Voiture v1 = new Voiture();
        Voiture v2 = new Voiture(40);
        Voiture v3 = new Voiture("Toyota", 2010, 85000, 26);
        v1.afficher();
        v2.afficher();
        v3.afficher();
```

Marque = Ford
Annee = 2017
Kilometrage = 100000.0 km
Prix = 30.0 Kdnt
Marque = Volvo
Annee = 2019
Kilometrage = 0.0 km
Prix = 40.0 Kdnt
Marque = Toyota
Annee = 2010
Kilometrage = 85000.0 km
Prix = 26.0 Kdnt

Champs et méthodes de classe

- En Java, on peut définir des champs qui, au lieu d'exister dans chacune des instances de la classe, n'existent qu'en un seul exemplaire pour toutes les instances d'une même classe. Exemple: le champ *nombre*.

Voiture.java

```
public class Voiture {
    private static int nombre = 0;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;

    public Voiture(String marque, int annee, double kilometrage, double prix)
{
        nombre++;
        System.out.println("Nombre de Voitures = "+ nombre);
        this.marque = marque;
        this.annee = annee;
        this.kilometrage = kilometrage;
        this.prix = prix;
    }
//...
}
```

Champs et méthodes de classe

- Il s'agit en quelque sorte de données globales partagées par toutes les instances d'une même classe. On parle alors de **champs de classe** ou de **champs statiques**.

MainClass.java

```
public class MainClass {
    public static void main(String[] args){
        Voiture v1 = new Voiture("Ford", 2015, 32000, 35);
        Voiture v2 = new Voiture("Peugeot", 2018, 7200, 42);
        Voiture v3 = new Voiture("Toyota", 2010, 85000, 26);
    }
}
```

Nombre de Voitures = 1
Nombre de Voitures = 2
Nombre de Voitures = 3

Champs et méthodes de classe

- De manière analogue, certaines méthodes d'une classe ont un rôle indépendant d'un quelconque objet.
- C'est notamment le cas d'une méthode se contentant d'agir sur des champs de classe ou de les utiliser.
- Une **méthode de classe**, ou **méthode statique** ne peut en aucun cas agir sur des champs usuels (non statiques) puisque, par nature, elle n'est liée à aucun objet en particulier.

Champs et méthodes de classe

- Pour appeler une telle méthode de classe, il est conseillé de la précéder par le nom de la classe.

Voiture.java

```
public class Voiture {
      private static int nombre = 0;
      //…
      public static int getNombre(){
            return nombre;
      }
}
```

MainClass.java

```
public class MainClass {
    public static void main(String[] args){
        Voiture v1 = new Voiture("Ford", 2015, 32000, 35);
        Voiture v2 = new Voiture("Peugeot", 2018, 7200, 42);
        Voiture v3 = new Voiture("Toyota", 2010, 85000, 26);

        System.out.println("Le nombre de voiture est: "+Voiture.getNombre());
    }
}
```

Exercice:

Donner le résultat d'exécution de ce programme.

```
public class Etudiant{
  private int numInscr;
  private String nom;
  private static String ecole =“ENSTAB";

  public Etudiant(int i, String n){
                numInscr = i;
                nom= n;
  }
  public void afficher (){ System.out.println(numInscr+" "+nom+" "+ecole); }
}
```

Etudiant.java

```
public class TestStaticVariable1{
   public static void main(String args[]){
                Etudiant e1 = new Etudiant(111,“Wael");
                Etudiant e2 = new Etudiant(222,“Wafa");
                e1.afficher();
                e2.afficher();
} }
```

TestStaticVariable1.java

Les constantes

- Le mot clé ***final*** est utilisé en Java pour déclarer une constante.
- Les constantes sont en général écrite en majuscules:

```
final int NOMBRE = 20 ;
final int LIMITE = 2 * NOMBRE + 3 ;
```

- Cependant, il n'est pas obligatoire d'initialiser une variable déclarée final lors de sa déclaration.

- Java demande simplement qu'une variable déclarée ***final*** ne reçoive qu'une seule fois une valeur.
- Considérons ces exemples:

```
final int n ;                 // OK, même si n n'a pas (encore) reçu de valeur
   .....
n = Clavier.lireInt() ;       // première affectation de n : OK
   .....
n++ ;                         // nouvelle affectation de n : erreur de compilation


final int n ;
   .....
if (...) n = 10 ;                      // OK
    else n = 20
```

- Il est toutefois recommandé d'initialiser une variable le plus près possible de sa déclaration.

Les constantes

- Considérons cet exemple:

```
public class Voiture {
    final int vitesseLimite = 160;
    private static int nombre = 0;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;
// ...
}
```

- Aucune modification de la valeur de vitesseLimite n'est autorisée.

- Il est d'usage d'associer ***static*** à ***final*** pour permettre l'utilisation des valeurs constantes par les classes et les objets:

```
public class Voiture {
    static final int VITESSE_LIMITE = 160;
    private static int nombre = 0;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;
// ...
}
```

- Ce choix est plus économique en terme de mémoire

Les variables locales

- Il n'y a pas de variables globales en Java.
- Une variable déclarée **dans le corps d'une méthode et les paramètres formels** sont des *variables locales*.
- Leur portée se limite au bloc de code dans lequel elles ont été déclarées. Elles ne sont pas conservées d'un appel à l'autre de la méthode.
- Les variables définies dans la méthode main sont aussi des variables locales.
- Les variables locales ne sont pas initialisées de façon implicite. **Elles doivent donc être initialisées avant d'être utilisées**.

Les variables locales

Voiture.java

```
public class Voiture {
    private static int nombre = 0;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;

     public double prixTtc(){
         double pTtc;
         System.out.println(pTtc);
         pTtc = prix * 1.19;
         return pTtc;
     }
}
```

Error:(86, 28) java: variable pTtc might not have been initialized

- Il faut initialiser la variable locale pTtc avant toute utilisation.

```
public class Voiture {
    private static int nombre = 0;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;

   public double prixTtc(){
         double pTtc = prix * 1.19;
         System.out.println(pTtc);
         return pTtc;
    }

}
```

Voiture.java

Les variables d'instance

- Les variables d'instance sont déclarées **dans le corps de la classe en dehors de toute méthode**.
- Elles permettent de stocker les informations relatives à un objet.
- Un objet est une *instance* d'une classe. **Chaque objet** est créé à partir d'une classe et **possède sa propre copie de toutes les variables d'instance**.
- Contrairement aux variables locales, les variables d'instance sont initialisées par défaut.

Types	Valeur
byte, short, int, long	0
float, double	0.0
char	'\0'
boolean	false

```
public class Voiture {
    private static int nombre = 0;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;
    public double prixTtc(){
        double pTtc;
        pTtc = prix * 1.19;
        return pTtc;
    }
}
```

- marque, annee, kilometrage et prix sont des variables d'instance.

Pour résumer

```
public class Voiture {
    private static int nombre = 0;
    final int vitesseLimite = 160;
    private String marque;
    private int annee;
    private double kilometrage;
    private double prix;

     public double prixTtc(){
         double pTtc;
         System.out.println(pTtc);
         pTtc = prix * 1.19;
         return pTtc;
    }
    public static int getNombre(){
         return nombre;
     }
}
```

- Il existe donc 3 types de variables :
  - Des variables/champs de classe (static) : *nombre*
  - Des variables locales : *pTtc*
  - Des variables d'instance (attributs): *marque, annee, kilometrage, prix.*
- Et des constantes: vitesseLimite

Le mot clef « this »

- Dans toutes les classes, on dispose d'une variable de référence ayant pour nom ***this*** qui réfère à l'objet courant.
- Ainsi lorsque l'on désire accéder à une donnée membre d'un objet ou appeler une méthode, il suffit de faire précéder le nom de la variable d'instance/méthode par ***this.***

- ***this*** est obligatoire dans certains cas pour éviter l'ambiguité entre variables locales (paramètres formels) et variables d'instance ayant des noms identiques.

```
public Voiture(String marque, int
annee, double kilometrage, double
prix) {
    this.marque = marque;
    this.annee = annee;
    this.kilometrage = kilometrage;
    this.prix = prix;
}
```