Module de mise à niveau en Java
Cours 2

Laure KAHLEM
laure.kahlem@univ-orleans.fr

Références

- Kathy Sierra et Bert Bates. Java Tête la première. O'Reilly, 2007.
- Mickaël Kerboeuf. Fondements de la programmation orientée objet avec Java8. Ellipses, 2016
- David Flanagan. Java in a Nutshell. Manuel de référence. O'Reilly, 2002.
- Cyrille Herby. Apprenez à programmer en JAVA. Broché, 2012.
- José Paumard. « Java en ligne » [en ligne]. http://blog.paumard.org/cours/java/ (Consulté le 15/07/2015).

Les objets et les classes

- **Principes de la programmation orientée objet:**
  - Un programme est constitué **d'objets qui interagissent** pour fournir les fonctionnalités attendues en échangeant **des messages**.
  - Chaque objet conserve des données.
  - Chaque **objet peut être considéré comme un fournisseur de services** utilisés par d'autres objets. C'est **l'interface de l'objet**.
  - Un objet connaît un certain nombre d'objets à qui il peut envoyer des messages pour appeler ses services.

Les objets et les classes

- **Caractéristiques d'un objet**
  - **Une identité**
  - **Un état** : les données que l'objet porte en lui
  - **Un comportement** : déterminé par les messages qu'il reçoit.

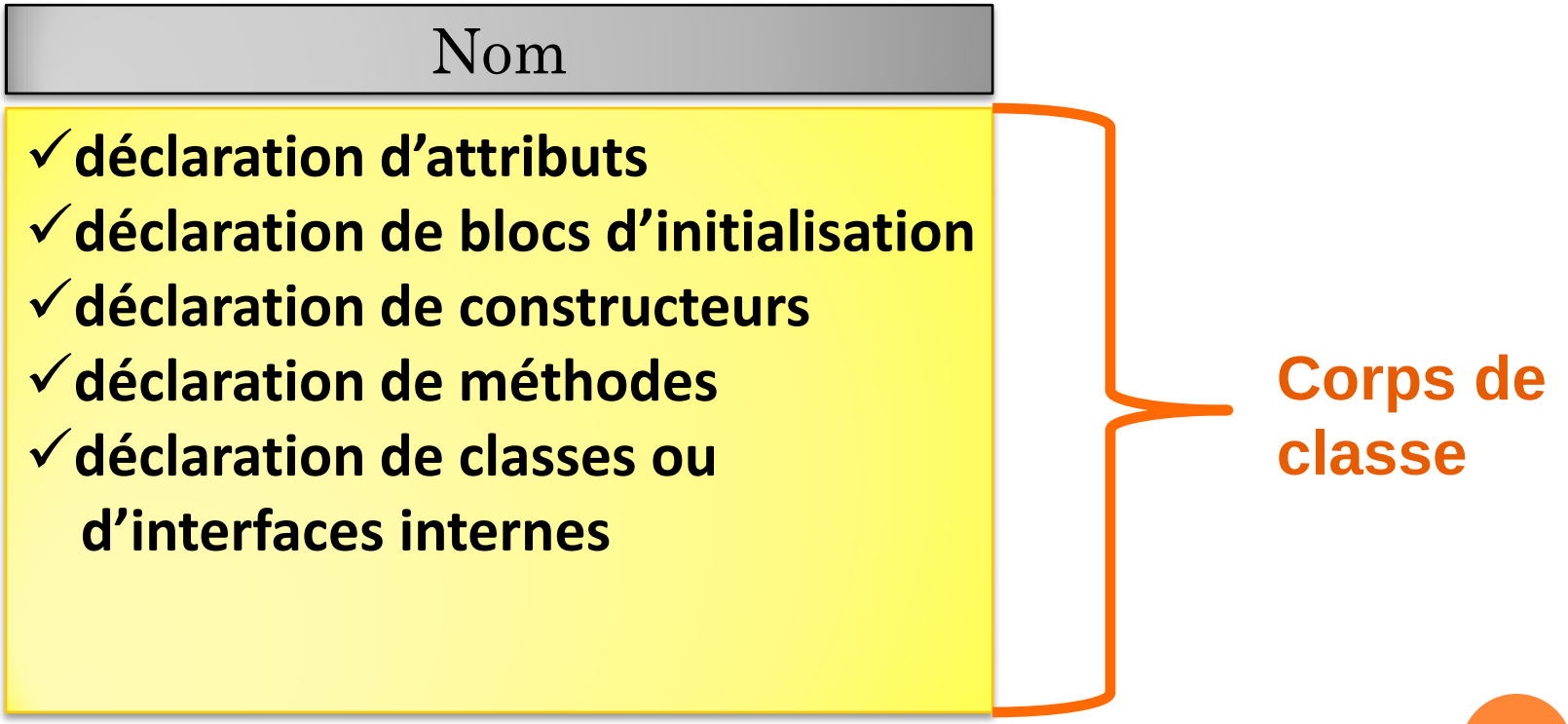
Principe d'encapsulation des données : Seul l'objet peut accéder à ses données. Pour qu'un objet extérieur y accède, il faut que l'objet ait été conçu pour proposer ce service.

Les objets et les classes

- **Concept de classe**
  - **modèle** décrivant les caractéristiques communes à plusieurs objets.
  - Une classe définit
    - les **attributs** ou **champs**
    - les **méthodes**.
  - Une classe sera **instanciée** pour créer un objet conforme au modèle.

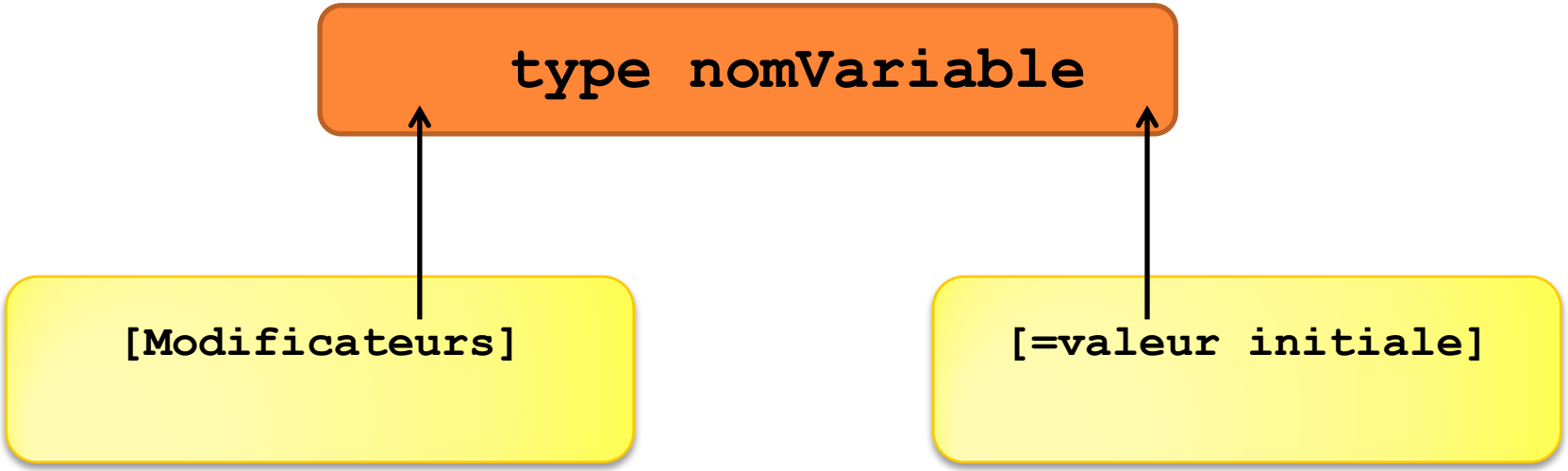
Les objets et les classes

- **Déclaration de classe**



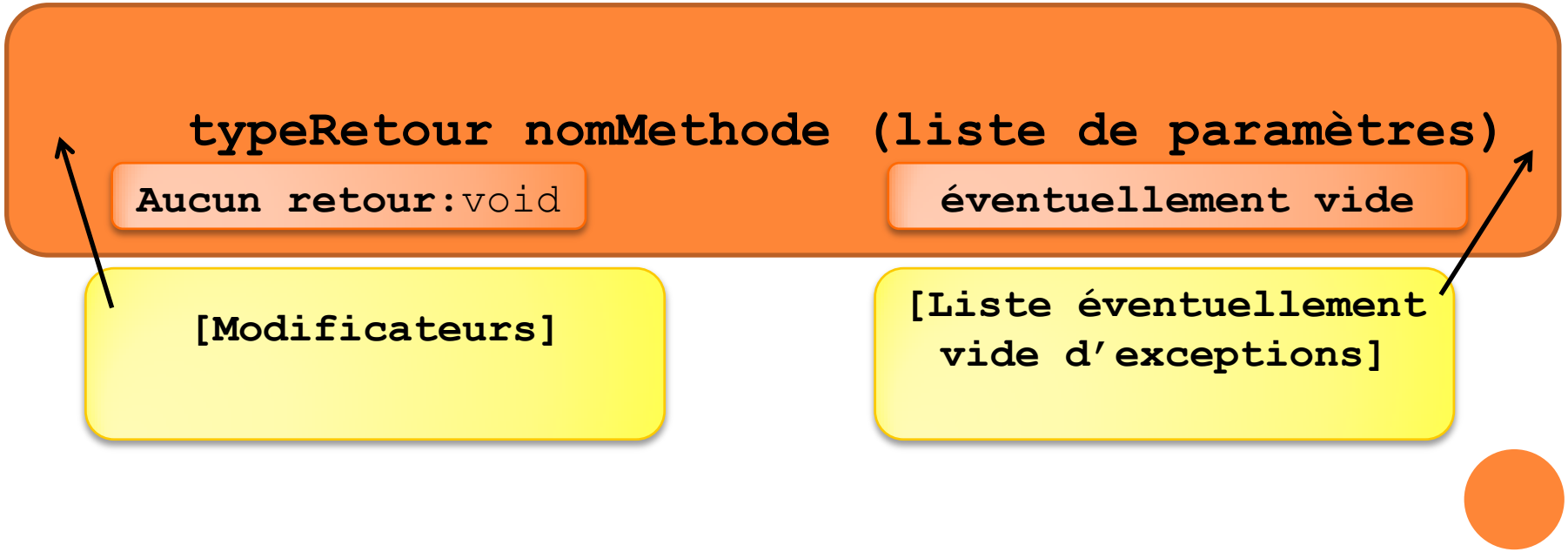
Les objets et les classes

- **Déclaration d'attribut**
  - **Attribut = variable déclarée dans le corps de la classe**



Les objets et les classes

- **Déclaration de méthode**
  - **Méthode = fonction déclarée dans le corps de la classe**



Les objets et les classes

- **Paramètres de méthode**
  - **Les paramètres sont toujours passés par valeur** : La méthode reçoit une copie de toutes les valeurs de paramètre.
  - Une méthode ne peut pas modifier un paramètre de type primitif.
  - Une méthode peut modifier l'état d'un paramètre objet.

Les objets et les classes

- **Accès aux attributs et invocations de méthodes** : opérateur (.)
  - **Exemple :**

```
class A {
 int compteur = 0;
 void incremente() {
      compteur ++;
 }
}

class B {
 A a = new A();
 int c = a.compteur;
 a.incremente();
}
```

Les objets et les classes

- **Surcharge de méthodes**
  - Possibilité dans le corps d'une classe de définir **plusieurs méthodes ayant le même nom**.
    - Elles doivent **différer en nombre et/ou en type de paramètres.**
    - On ne peut définir deux méthodes ayant même nom et ayant comme seule différence le type de retour.

Les objets et les classes

- **Nombre variable de paramètres dans une méthode**

```
typeRetour nomMethode (type param1, type param2,
type ... param3)
```

• Représente **une séquence éventuellement vide de valeurs du type déclaré**
• Il **y en a au plus une** dans une méthode
• Elle apparaît **en dernier** dans la liste des paramètres de la méthode

Les objets et les classes

- **Nombre variable de paramètres dans une méthode**
  - **Exemple**

```
class A {
    String m (int i, String... t){
          if (i < t.length && i>=0)
                return t[i];
          else return null;
    }
}
class B {
     A a = new A();
     String[] tab= {"a","b","c"};
     String s2 = a.m(2, tab);
     String s1 = a.m(0, "a","b","c");
    }
```

t correspond à un tableau

Deux appels possibles

Les objets et les classes

- **Modificateurs**
  - **final**

Attribut	Méthode	Classe
Attribut constant **Exemple :** `final int MIN = 3;`	Méthode non redéfinissable	Ne peut être décomposée en sous-classe

Les objets et les classes

- **Modificateurs**
  - **static**

Attribut statique (de classe)	Méthode statique
Commun à toutes les instances	**Fonction utilitaire de la classe**
Accès direct en utilisant le nom de la classe	Ne s'applique pas à un receveur. Accès direct en utilisant le nom de la classe
Stockage en mémoire dans une zone dédiée à la classe elle-même à laquelle toutes les instances sont liées.	Ne peut accéder aux attributs d'instance, seulement aux attributs statiques.

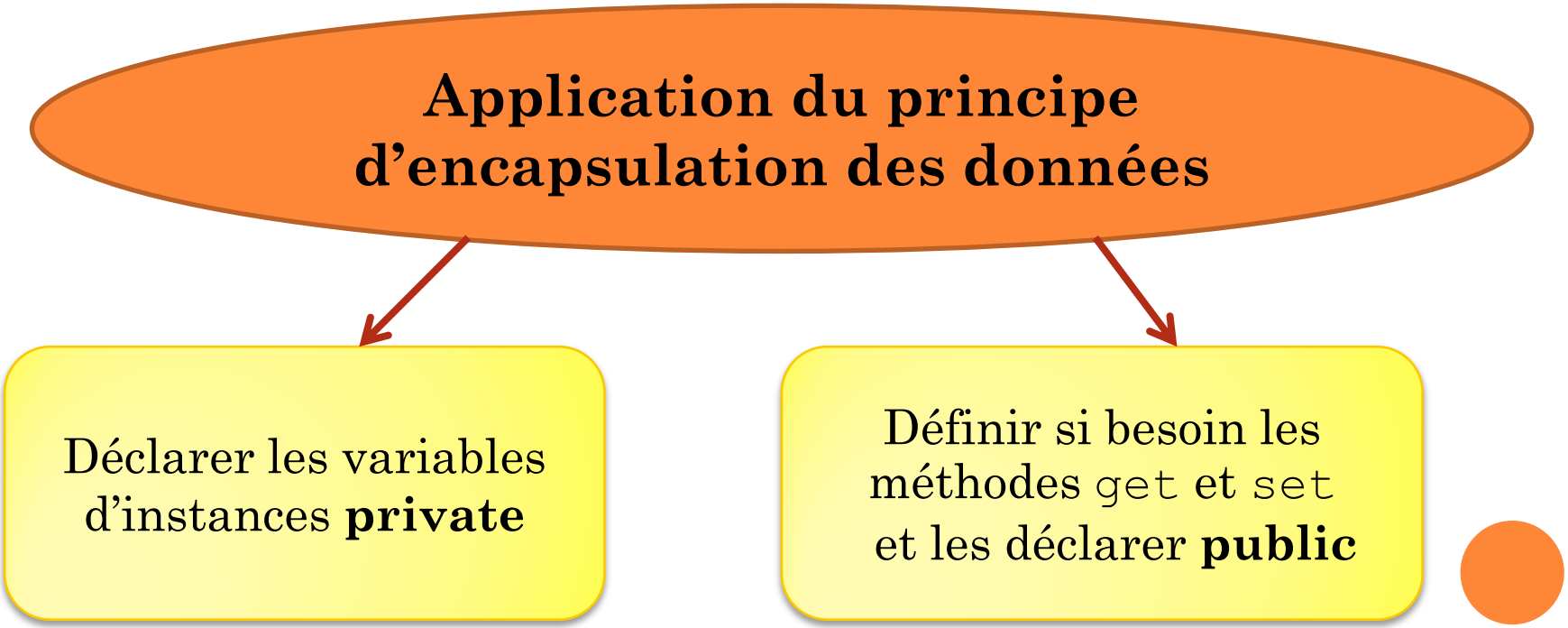
Les objets et les classes

- **Modificateurs définissant le niveau de visibilité d'un attribut ou d'une méthode ou d'une classe**

Déclaration	Classe	Package	Sous-classe	autre
`public m`	ok	ok	ok	ok
`protected m`	ok	ok	ok	
`m`	ok	ok		
`private m`	ok			

Les objets et les classes

- **Modificateurs définissant le niveau de visibilité d'un attribut ou d'une méthode ou d'une classe**



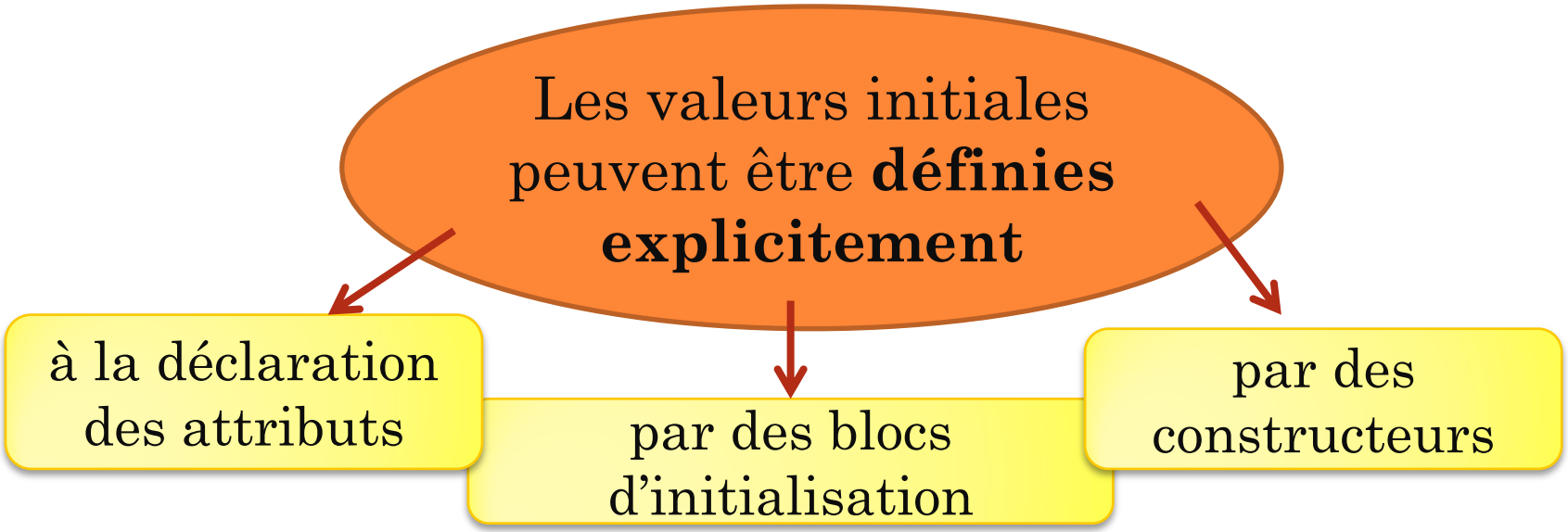
Les objets et les classes

- **Initialisation des objets**

À la création d'un objet, les attributs doivent recevoir des valeurs initiales.

Les objets et les classes

- **Initialisation des objets**



Sans initialisation explicite d'un attribut, le compilateur affecte une valeur par défaut

Les objets et les classes

- **Blocs d'initialisation**
  - **Blocs d'instructions encadrées par des accolades définis au niveau de la classe**

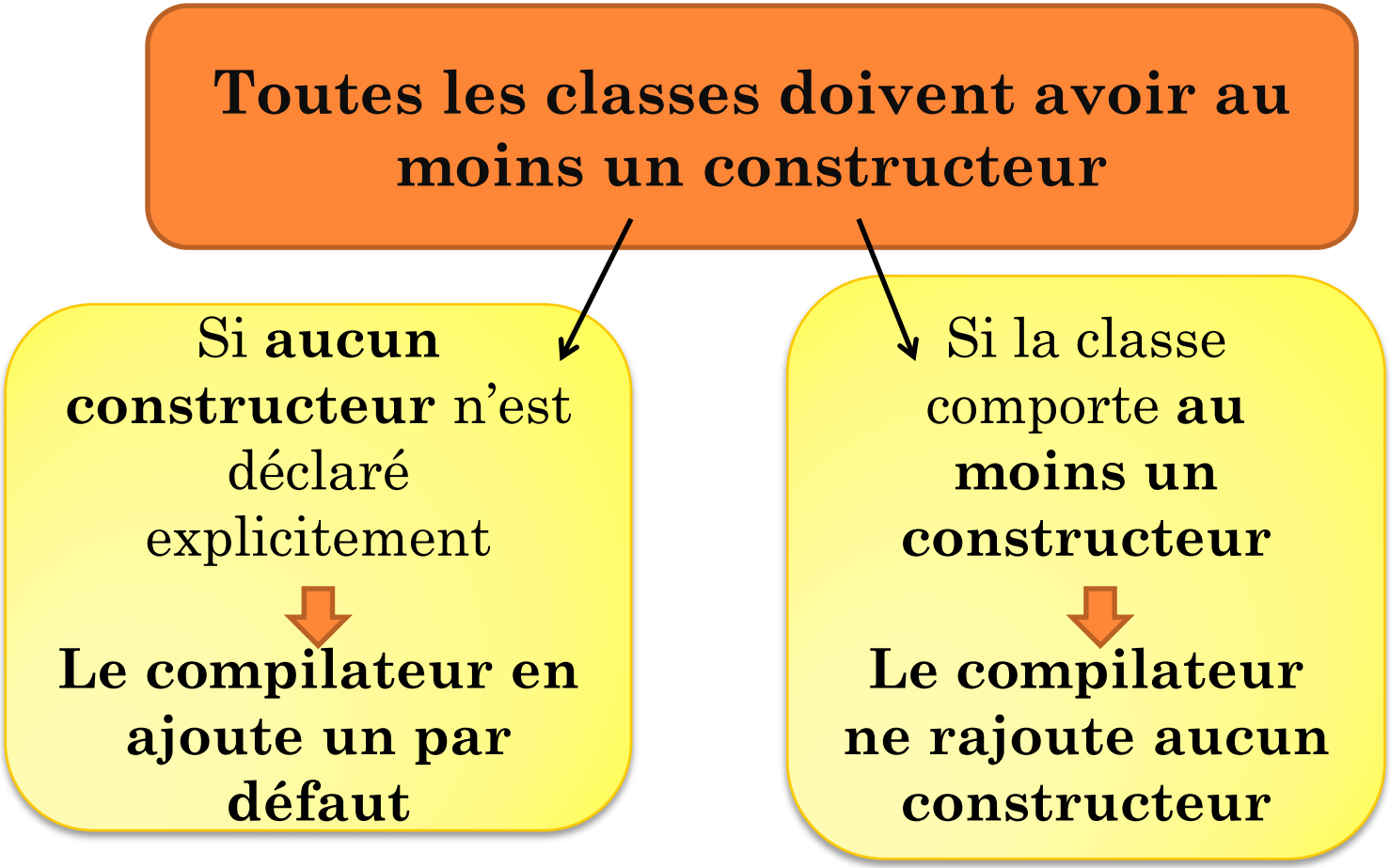
Bloc d'initialisation d'instance	Bloc d'initialisation statique
• Regroupe les instructions qui doivent **être exécutées à chaque instanciation de la classe** • Code commun à tous les constructeurs	Instructions exécutées **une seule fois lors du chargement dynamique de la classe** (lors de la première utilisation de la classe)

Les objets et les classes

- **Constructeurs**
  - Un constructeur est **une méthode particulière** qui sert à **initialiser les attributs de l'objet**.
  - **Il a les caractéristiques suivantes :**
    - Porte le **nom de la classe**
    - Ne comporte **pas de type de retour**
    - Est invoqué par l'opérateur **`new`**
  - **Surcharge de constructeurs**
    - On peut invoquer un constructeur dans un autre en le désignant par **`this`**. L'appel doit se faire en **première ligne** du constructeur.

Les objets et les classes

- **Constructeurs**



Les objets et les classes

- **Exemple 1 :**
  - Création de la classe CompteBancaire:
    - **Variables d'instance (état) :**
      - Numéro du compte
      - Nom
      - Adresse
      - Solde
    - **Méthodes (comportement)**
      - Créer un compte en passant le numéro, le nom et l'adresse en paramètres, solde=0.
      - Créer un compte en passant le numéro, le nom, l'adresse en paramètres et le solde
      - créditer
      - débiter

```
public class CompteBancaire {
    int numeroCompte ;
    String nom ;
    String adresse ;
    double solde ;
    CompteBancaire(int numeroCompte, String nom, String
   adresse, double solde){
                this.numeroCompte = numeroCompte;
                this.nom= nom;
                this.adresse=adresse;
                this.solde = solde;
    }
    CompteBancaire(int numeroCompte, String nom, String
   adresse){
                this (numeroCompte, nom, adresse, 0.0);

    }
     void crediter(double montant){
                solde = solde + montant;
     }
     void debiter(double montant){
                solde = solde - montant;
     }
}
```

Invocation du constructeur précédent

Application du principe d'encapsulation

```
public class CompteBancaire {
   private int numeroCompte ;
   private String nom ;
   private String adresse ;
   private double solde;

   public CompteBancaire(int numeroCompte, String nom, String
   adresse, double solde){
                this.numeroCompte = numeroCompte;
                this.nom= nom;
                this.adresse=adresse;
                this.solde = solde;
  }
   public CompteBancaire(int numeroCompte, String nom, String
   adresse){
                this (numeroCompte, nom, adresse, 0.0);
   }
   public String getNom(){
        return nom;
   }
   public void setNom(String nom){
        this.nom = nom;
   }
```

```
    public String getAdresse(){
          return adresse;
    }
    public void setAdresse(String adresse){
          this.adresse = adresse;
    }
    public int getNumeroCompte(){
          return numeroCompte;
    }
    //pas de méthode setNumeroCompte (int n)

    public int getSolde(){
          return solde;
    }
    //pas de méthode setSolde (double s)
    //le solde ne peut être modifié qu'en créditant ou débitant le compte

    public void crediter(double montant){
          solde = solde + montant;
    }
    public void debiter(double montant){
          solde = solde - montant;
    }
}
```

Les objets et les classes

- **On souhaite que le numéro du compte s'incrémente automatiquement à chaque création d'un nouveau compte.**
  Le numéro du compte ne sera plus passé en paramètre du constructeur.

```
public class CompteBancaire {
   private int numeroCompte ;
   private String nom ;
   private String adresse ;
   private double solde;
   //variable commune à toutes les instances
   private static int numeroSuivant =10;

   public CompteBancaire(String nom, String adresse, double
   solde){
                this.numeroCompte=numeroSuivant;
                numeroSuivant += 10;
                this.nom= nom;
                this.adresse=adresse;
                this.solde = solde;
                        }
   public CompteBancaire(String nom, String adresse){
                this (nom, adresse, 0.0);
   }
   …
}
```

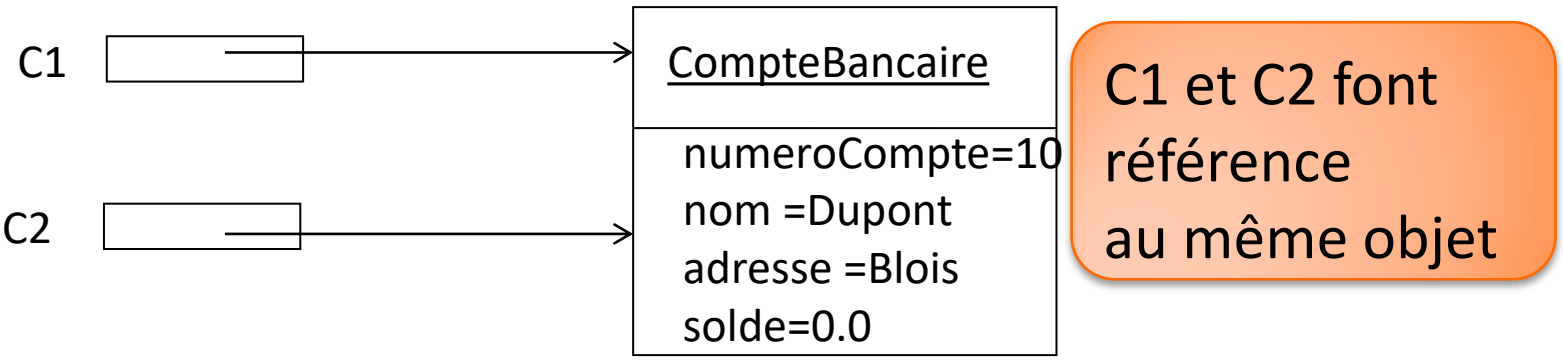
```
public class CompteBancaire {
   private int numeroCompte ;
   private String nom ;
   private String adresse ;
   private double solde;
   private static int numeroSuivant;
//bloc exécuté une seule fois au chargement dynamique de la classe
   static { numeroSuivant = 10; }
//bloc exécuté à chaque création d'un nouveau compte bancaire
   { numeroCompte = numeroSuivant; numeroSuivant += 10;}
   public CompteBancaire(String nom, String adresse, double
   solde){
                this.nom= nom;
                this.adresse=adresse;
                this.solde = solde;
                        }
   public CompteBancaire(String nom, String adresse){
               this (nom, adresse, 0.0);
   }
   …
}
```

Autre manière d'initialiser : utilisation de blocs d'initialisation

Les objets et les classes

- **Instanciation de la classe `CompteBancaire`**

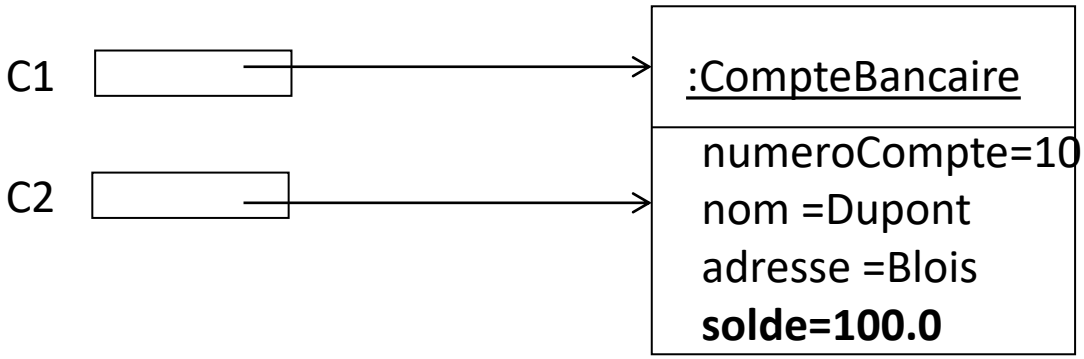
```
CompteBancaire c1 = new CompteBancaire ("Dupont",
    "Blois");
CompteBancaire c2 = c1;
```



Les objets et les classes

- **Appel de la méthode crediter**

```
c1.crediter (100);
```



Les objets et les classes

- **Packages**
  - Un package est une **structure logique** permettant de regrouper des classes (et des interfaces).
  - **Pas de notion de sous-package.**
  - Pour définir une classe (ou une interface) à l'intérieur d'un package **p**, il faut la définir dans un fichier Java dont la première ligne est : **`package p;`**

// Fichier **fichier1.java**

```
package p;
class A {/*…*/}
class B {/*…*/}
```

// Fichier **fichier2.java**

```
package p;
class E {/*…*/}
interface I {/*…*/}
```

Les objets et les classes

- **Packages**
  - **Conventions de nommage**
    - Toujours en minuscule
    - Caractères alphanumériques et des points « . »
    - Tout package devrait commencer par un nom de domaine à l'envers.
      Exemple : com.domaine.monpackage
    - Chaque nom de package correspond à une arborescence de dossier
      com/domaine/monpackage

Les objets et les classes

- **Packages et visibilité**
  - **Visibilité par défaut**

Par défaut, les classes et interfaces ne sont visibles et accessibles que par les membres du même package.

  - **Extension de visibilité avec le modificateur de classe `public`**

Une classe **publique** est **accessible à l'extérieur de son package.**

Les objets et les classes

- **Package et visibilité**

Si le fichier contient une classe publique, elle donne son nom au fichier.

```
// Fichier A.java
package p;
public class A {/*…*/}
class B {/*…*/}
```

Une seule classe ou interface publique par fichier

```
// Fichier C.java
package q;
import p.A;
public class C {
   A a;
      …
}
```

Classes particulières

- **La classe `String`**
  - Le **chaînes de caractères** en Java sont des **objets** instances de la classe **`String`**.
  - On ne peut pas ajouter, supprimer ou mettre à jour un caractère de la chaîne.

Les chaînes de caractères sont constantes

Leurs références peuvent être partagées sans risque

Classes particulières

- **La classe `String`**
  - **Différents constructeurs**
    - **Création d'une chaîne à partir d'un caractère**

```
String s1 = new String ('a'); // valeur "a"
```

    - **Création d'une chaîne à partir d'un tableau de caractères**

```
char[] charTab = {'b', 'o','n','j','o','u','r'};
String s2 = new String (charTab);
// valeur "bonjour"
```

    - **Création d'une chaîne à partir d'un littéral de type String**

```
String s3 = "bonjour" ;
```

Classes particulières

- **La classe `String`**
  - **Opérateur de concaténation : +**
  - **Fonctions de manipulation**
    - **Nombre de caractères de la chaîne**

      `int length()`
    - **Caractère à l'indice `index`**

      `char charAt (int index)`
    - **Tableau de caractères correspondant à la chaîne**

      `char[] toCharArray ()`

Classes particulières

- **La classe `String`**
  - **Fonctions de comparaison**
    - **Test d'égalité avec ou sans prise en compte de la casse**

      ```
      boolean equals(Object o)
      boolean equalsIgnoreCase(String str)
      ```

    - **Comparaison lexicographique avec ou sans prise en compte de la casse**

      ```
      int compareTo (String str)
      int compareToIgnoreCase(String str)
      ```

Classes particulières

- **La classe `String`**
  - **Fonctions de transformation**
    - **Création d'une chaîne par changement de casse**

      ```
      String toLowerCase()
      String toUpperCase()
      ```

    - **Création d'une chaîne en remplaçant un caractère**

      ```
      String replace (char oldChar, char newChar)
      ```

    - **Création d'une chaîne par suppression des espaces du début et de la fin**

      ```
      String trim ()
      ```


Classes particulières

- **La classe `String`**
  - **Fonctions d'extraction**
    - **Extraction d'une sous-chaîne de caractères**

      `String subString (int beginIndex, int endIndex)`
    - **Extraction de sous-chaînes de caractères séparées par les motifs correspondant à une expression régulière**

      `String[] split (String regex)`

Classes particulières

- **La classe `String`**
  - **Retourne la chaîne de caractère correspondant au type primitif**

    static `String valueOf (int i)`

    static `String valueOf (char c)`

    static `String valueOf (boolean b)`

Classes particulières

- **Les classes `StringBuffer` et `StringBuilder`**
  - Représentent des chaînes de caractères **modifiables**
  - Exemple des méthodes d'ajout , de suppression et de modification de caractères de la chaîne
    - `StringBuffer insert (int offset , String str)`
    - `StringBuffer deleteCharAt(int index)`
    - `StringBuffer replace(int start, int end , String str)`
  - Méthode permettant de concaténer une chaîne avec une autre : `append`

Classes particulières

- **Les classes enveloppes**
  - **Chaque type primitif a une classe correspondante, nommée classe enveloppe.**

Type primitif	`boolean`	`byte`	`char`	`short`
Classe enveloppe	`Boolean`	`Byte`	`Character`	`Short`
Type primitif	`int`	`long`	`float`	`double`
Classe enveloppe	`Integer`	`Long`	`Float`	`Double`

Classes particulières

- **Les classes enveloppes**
  - Les classes enveloppes fournissent les méthodes de base permettant de convertir des chaînes de caractères en nombre
    - Exemple :

```
int x = Integer.parseInt (s);
```

Classes particulières

- **La classe `java.util.Scanner`**
  - **Depuis le JDK 5.0,** on a la possibilité de balayer un texte et d'en extraire les éléments de type primitif ou les chaînes de caractères.
  - Un objet de type **`Scanner`** va décomposer son entrée en mots en utilisant un séparateur, qui par défaut est l'espace vide.

Classes particulières

- **La classe `java.util.Scanner`**
  - **Méthodes de la classe Scanner pour lire les entrées**
    - `public String nextLine()`
    - `public String next ()`
    - `public int nextInt()`
    - `public double nextDouble()`
  - **Méthodes de la classe Scanner pour tester l'existence d'un élément à lire**
    - `public boolean hasNextLine()`
    - `public boolean hasNext()`
    - `public boolean hasNextInt()`
    - `public boolean hasNextDouble()`

Classes particulières

- **La classe `java.util.Scanner` permet de lire les entrées**
  - **depuis un objet de type String (ex : `String leTexte`)**

```
Scanner sc = new Scanner (leTexte);
int i = sc.nextInt();
```

  - **depuis la fenêtre de la console**

On construit un objet Scanner attaché à l'unité « d'entrée standard » System.in :

```
Scanner sc = new Scanner(System.in);
int i = sc.nextInt();
```

  - **depuis un fichier contenant des entiers de type long**

```
Scanner sc = new Scanner(new File("DesNombres"));
while (sc.hasNextLong())
        { long aLong = sc.nextLong(); }
```

Classes particulières

- **La classe Arrays**
  - **Fournit de nombreuses méthodes statiques de manipulation des tableaux**
    - **Création d'une copie d'un tableau avec plus ou moins de cases**
      - **Méthode pour un tableau d'entiers:**

```
public static int[] copyOf (int[] original, int newLength)
```

    - **Trier un tableau**
      - **Méthode pour un tableau d'entiers:**

```
public static void sort(int[] a)
```

Classes particulières

- **La classe Arrays**
  - **Fournit de nombreuses méthodes statiques de manipulation des tableaux**
    - **Test d'égalité de deux tableaux de tout type (même taille et même valeurs dans le même ordre)**
      - **Méthode pour deux tableaux d'entiers:**

        ```
        public static boolean equals(int[] a,
        int[] a2)
        ```

    - **Test d'égalité de deux tableaux d'objets en profondeur** (utile pour les tableaux à n dimensions)

```
public static boolean deepEquals(Object []
    a1, Object[] a2)
```

Classes particulières

- **La classe Arrays**
  - **Production d'une chaîne de caractères représentant les éléments du tableau**
    - **Méthode pour un tableau d'entiers:**

      `public static String` **`toString`**`(int[] a)`
    - **Méthode pour un tableau à n dimensions:**

      `public static String` **`deepToString`**`(Object[] a)`