Amphi 3

Classes et Objets ensemblistes

Classes & Objets

```
class X {
```

attributs (variables d'instance)

constructeurs

méthodes d'instance

```
}
```

- La manière que les objets doivent être conçus
- Un objet est vu comme une entité qui encapsule un état interne (attributs) et qui fournit des services (méthodes d'instance)

Classes & Objets

- Le paradigme Orienté-Objet a permis:

  d'inclure dans les objets des méthodes d'instance
  "responsables" d'agir sur leurs attributs

- Les objets sont maintenant à la fois

  des structures de données (attributs valués) et des
  fournisseurs de fonctionnalités (méthodes d'instance)

- Au final, les objets apparaissent donc comme

  de véritables entités dotées d'une certaine autonomie.

Ensembles d'objets

Les problèmes réels manipulent souvent de nombreuses données d'un même type:

Jeu de dominos

Bibliothèque

- Il devient donc nécessaire de disposer de classes permettant de traiter des ensembles d'objets d'un même type: classes ensemblistes
- Deux manières de traiter les ensembles d'objets:

à l'aide de Tableaux

à l'aide de Classes génériques à spécialiser

Direction 1: Tableaux

Comment avons-nous traiter les ensembles d'objets jusque là?

```
Livre l1 = new Livre("Fourmis sans ombre - Le Livre du haïku",
      "Maurice COYAUD", "Phébus", 320, new ISBN(2, 85940, "017", "6"));

Livre l2 = new Livre("Conscience et environnement : La symphonie de la vie",
      "Pierre Rabhi", "Le Relié", 233, new ISBN(2, 354, "90023", "6"));

Livre[] livres = {l1, l2};

System.out.println(setLivresToString(livres));
```

Direction 1 : Tableaux

Comment avons-nous traiter les ensembles d'objets jusque là?

```
Livre l1 = new Livre("Fourmis sans ombre - Le Livre du haïku",
      "Maurice COYAUD", "Phébus", 320, new ISBN(2, 85940, "017", "6"));

Livre l2 = new Livre("Conscience et environnement : La symphonie de la vie",
      "Pierre Rabhi", "Le Relié", 233, new ISBN(2, 354, "90023", "6"));
```

```
Livre[] livres = {l1, l2};

System.out.println(setLivresToString(livres));
```

méthode d'instance impossible

Direction 1: Tableaux

Comment avons-nous traiter les ensembles d'objets jusque là?

```
Livre l1 = new Livre("Fourmis sans ombre - Le Livre du haïku",
      "Maurice COYAUD", "Phébus", 320, new ISBN(2, 85940, "017", "6"));

Livre l2 = new Livre("Conscience et environnement : La symphonie de la vie",
      "Pierre Rabhi", "Le Relié", 233, new ISBN(2, 354, "90023", "6"));
```

```
Livre[] livres = {l1, l2};

System.out.println(setLivresToString(livres));
```

méthode d'instance impossible

- Insatisfaisant en Orienté-Objet: on ne peut leurs associer aucune méthode d'instance (sauf méthode length()).
- En absence de classe pour recevoir le tableau d'objets, le paradigme de l'O-O ne s'applique pas au tableau

Classe encapsulant un tableau d'objets

Créer une classe spécifique pour contenir un ensemble des livres

```
class Bibliotheque{

  // attribut
  Livre [] biblio;

  //constructeur
  Bibliotheque(...){...}

  // methodes d'instance
  public String bibliothequeToString(){
      String s = "";
      for (Livre l: this.biblio){
         s = s + l.livreToString();
      }
      return s;
  }

}
```

Classe encapsulant un tableau d'objets

Créer une classe spécifique pour contenir un ensemble des livres

Encapsuler le tableau de livres dans la classe comme un varible d'instance

```
class Bibliotheque{

  // attribut
  Livre [] biblio;

  //constructeur
  Bibliotheque(...){...}

  // methodes d'instance
  public String bibliothequeToString(){
      String s = "";
      for (Livre l: this.biblio){
         s = s + l.livreToString();
      }
      return s;
  }

}
```

Classe encapsulant un tableau d'objets

Créer une classe spécifique pour contenir un ensemble des livres

Encapsuler le tableau de livres dans la classe comme un varible d'instance

Doter la classe des méthodes d'instance utiles

```
class Bibliotheque{

  // attribut
  Livre [] biblio;

  //constructeur
  Bibliotheque(...){...}

  // methodes d'instance
  public String bibliothequeToString(){
      String s = "";
      for (Livre l: this.biblio){
         s = s + l.livreToString();
      }
      return s;
  }

}
```

Classe encapsulant un tableau d'objets

Créer une classe spécifique pour contenir un ensemble des livres

Encapsuler le tableau de livres dans la classe comme un varible d'instance

Doter la classe des méthodes d'instance utiles

```
class Bibliotheque{

  // attribut
  Livre [] biblio;

  //constructeur
  Bibliotheque(...){...}

  // methodes d'instance
  public String bibliothequeToString(){
      String s = "";
      for (Livre l: this.biblio){
         s = s + l.livreToString();
      }
      return s;
  }

}
```

Il reste à prévoir comment la bibliothèque pourra être construit.

Classe encapsulant un tableau d'objets

```
class Bibliotheque{

  // attribut
  Livre [] biblio;

  //constructeur
  Bibliotheque(int taille){
      this.biblio = new Livre [taille];
  }

  // methodes d'instance
  public ajouter(Livre l){
  }
  public supprimer(Livre l){
  }
  public String bibliothequeToString(){
  }
}
```

Construire une bibliothèque avec taille donnée

Ajouter des **méthodes d'instance** pour manipuler

Classe encapsulant un tableau d'objets

```
class Bibliotheque{
  // attribut
  Livre [] biblio;
  //constructeur
  Bibliotheque(int taille){
      this.biblio
          = new Livre [taille];
  }
  // methodes d'instance
  public int size(){
      return this.biblio.length();
  }
  public String
      bibliothequeToString(){...
  }
}
```

```
class MainBibilio{
public static void main(String
[] args){
      Bibliotheque b = new
            Bibliotheque (10);
      ...
      System.out.println
      ("Bibliotheque contient "
         + b.size() + "livres
           suivants: "
         +
        b.bibliothequeToString
              ());
  }
}
```

Classe encapsulant un tableau d'objets

```
class Bibliotheque{
  // attribut
  Livre [] biblio;
  //constructeur
  Bibliotheque(int taille){
      this.biblio
          = new Livre [taille];
  }
  // methodes d'instance
  public int size(){
      return this.biblio.length();
  }
  public String
      bibliothequeToString(){...
  }
}
```

```
class MainBibilio{
public static void main(String
[] args){
      Bibliotheque b = new
            Bibliotheque (10);
      ...
      System.out.println
      ("Bibliotheque contient "
         + b.size() + "livres
           suivants: "
         +
        b.bibliothequeToString
              ());
   }
}
```

Bonne pratique dans le paradigme Orienté-Objet

Objectifs du paradigme O-O

Ce que nous venons de mettre en oeuvre est en parfait accord avec l'objectif du paradigme Orienté-Objet

Un problème étant posé:

- le décomposer en plusieurs petites entités plus simples (classes), faire apparaître des classes d'objets bien ciblées
  - leur donner le maximum d'autonomie

    (en évitant qu'elles délèguent à d'autres ce qu'elles peuvent faire)
  - leur permettre d'être manipulées extérieurement le plus facilement possible

    (en les dotant de méthodes d'instance utiles à la résolution globale)
- résoudre le problème global posé en faisant interagir ces entités entre elle. Ici, il ne s'agit plus que de manipuler les entités depuis l'extérieur.

Limite des classes encapsulant un tableau d'objets

```
class Bibliotheque{

  // attribut
  Livre [] biblio;

  //constructeur
  Bibliotheque(int taille){
      this.biblio
        = new Livre [taille];
  }

  // methodes d'instance

}
```

- Lorsque la taille des ensembles d'objets est variable, cela induit un aspect dynamique
- Cet aspect dynamique complique la gestion des tableaux car ceux-ci sont des structures de données statiques
- Besoin: les classes encapsulant des ensembles d'objets de taille dynamique

Direction 2: Classes génériques à spécialiser

- But: manipuler dynamiquement des ensembles d'éléments

- En Java, on dispose notamment de classes génériques prêtes à être spécialisées.

Classe génériques à spécialiser

La classe générique ArrayList<E>

ArrayList<E> doit être spécialisée à l'aide d'un nom de classe d'objets

- Example de spécialisation de classe générique

  `ArrayList <Domino>` représente la classe des listes de dominos

  `ArrayList <Livre>` représente la classe des listes de livres

  `ArrayList <int>` interdit

- Instanciation (utilisation d'un constructeur par défaut)

```
ArrayList <Domino> listDominos = new ArrayList<Domino>();

ArrayList <Livre>  listLivres  = new ArrayList<Livre>();
```

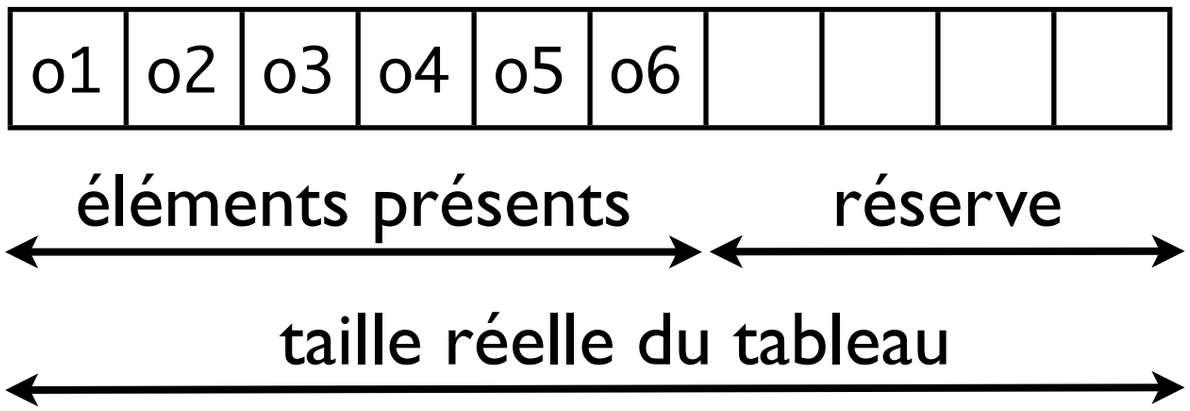
Classe génériques à spécialiser

Une fois la classe génériques spécialisée, on dispose d'une véritable classe qui:

- encapsule un ensemble d'objets
  - la classe ArrayList<E> encapsule un tableau d'objets et en gère les aspect dynamique
- dispose de nombreuses méthodes d'instance publiques
  - par rapport à l'ensemble `size(), isEmpty()`
  - par rapport à la lecture `get(), contains()`
  - par rapport à l'écriture `add(), remove()`
- considère un ordre naturel sur l'ensemble
- les méthodes d'instance sont optimisées

ArrayList<E> & tableaux d'objets

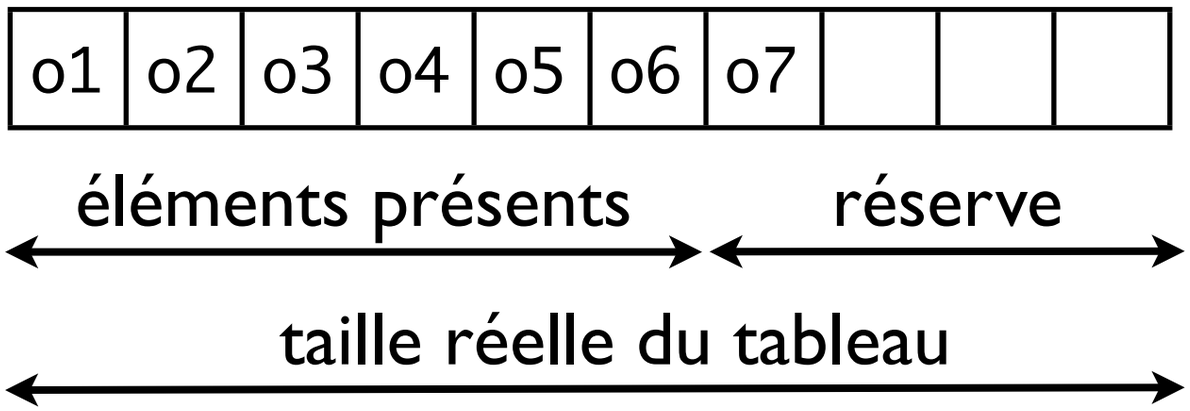
Comment ArrayList<E> gère-t-elle son tableau d'objet?



- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ArrayList<E> & tableaux d'objets

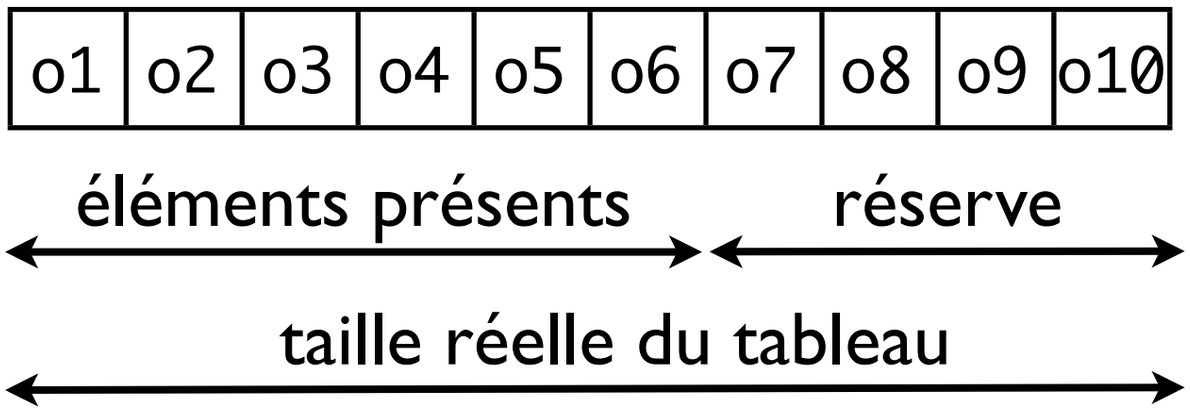
Comment ArrayList<E> gère-t-elle son tableau d'objet?



- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ArrayList<E> & tableaux d'objets

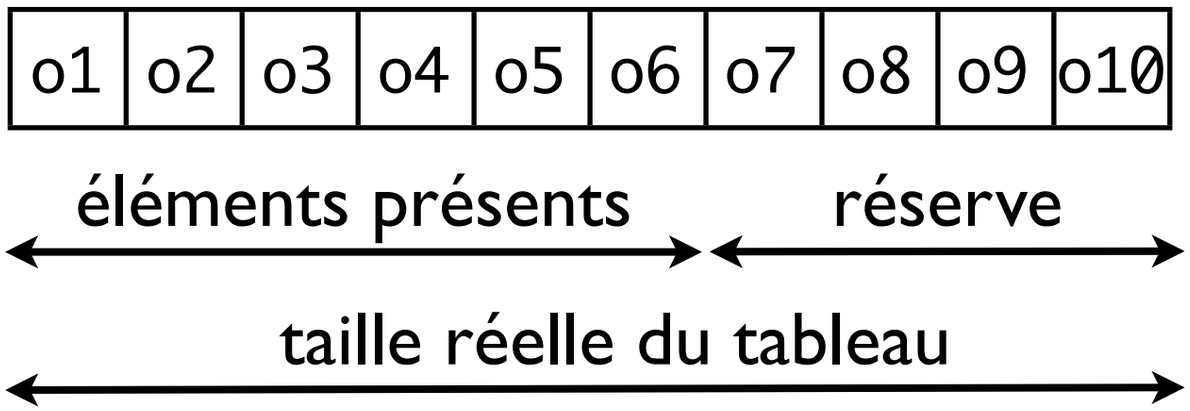
Comment ArrayList<E> gère-t-elle son tableau d'objet?

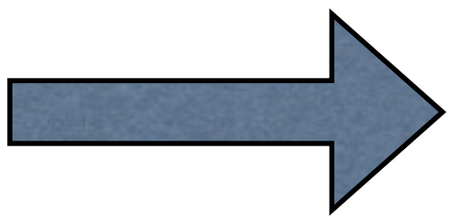


- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ArrayList<E> & tableaux d'objets

Comment ArrayList<E> gère-t-elle son tableau d'objet?



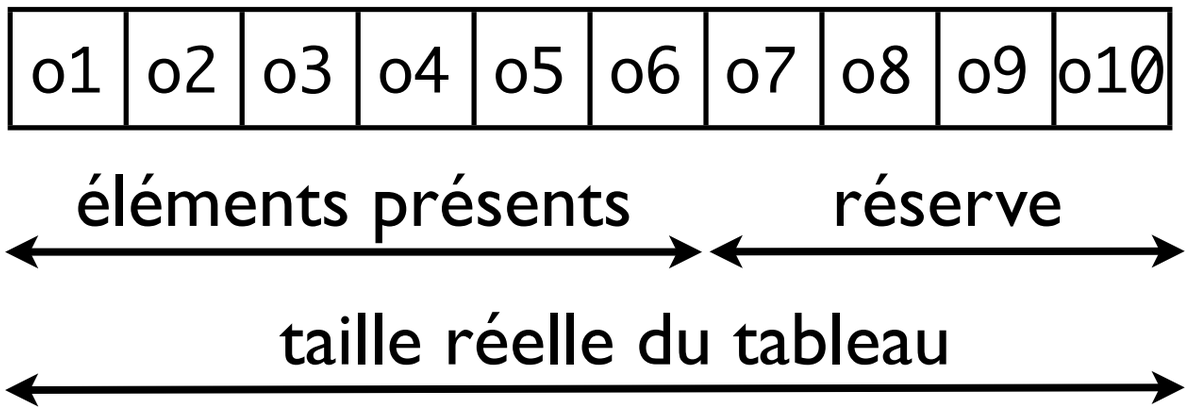


ajouter d'un nouveau élément

- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

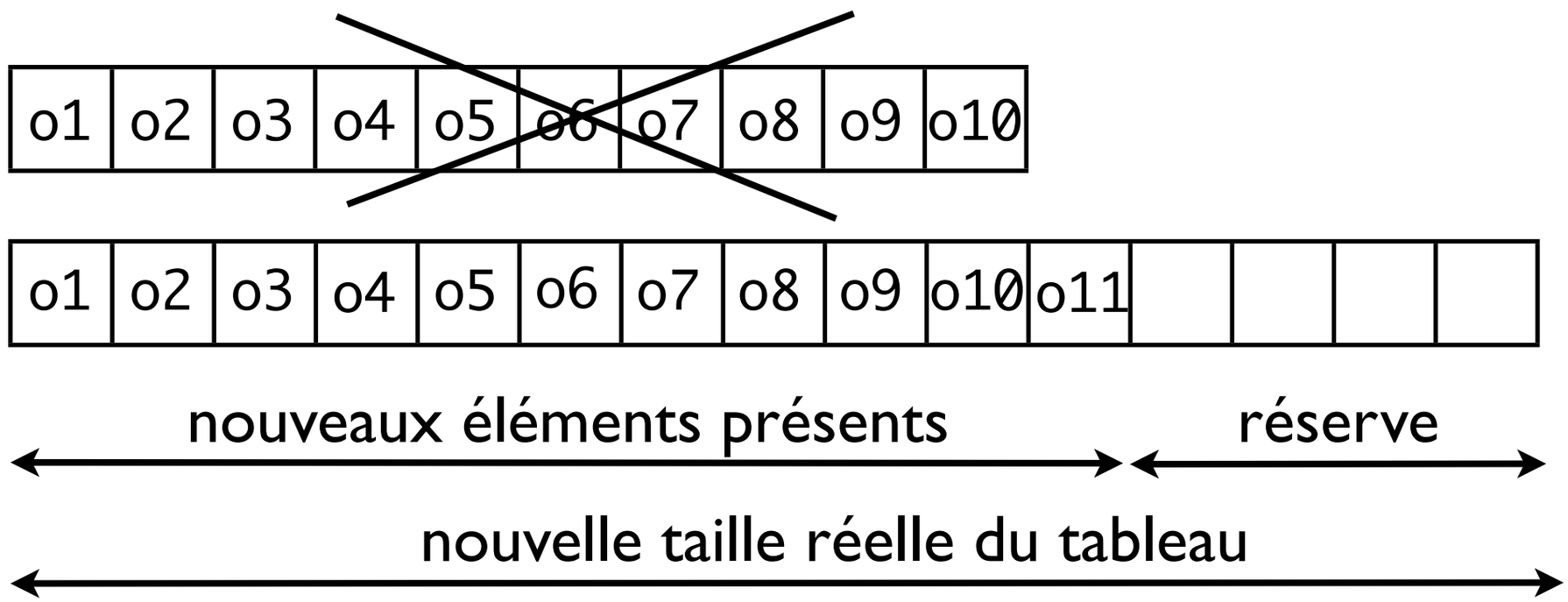
ArrayList<E> & tableaux d'objets

Comment ArrayList<E> gère-t-elle son tableau d'objet?



- elle gère le nombre d'éléments valides ainsi qu'une réserve d'emplacements libres
- un tableau rarement complètement rempli

ajouter d'un nouveau élément



ArrayList<E> & méthodes utiles

- par rapport à l'ensemble:

"E" : Livre, Domino, etc

```
int size()              retourne la taille de l'ensemble
boolean isEmpty()       teste si l'ensemble est vide
Object clone()          retourne une copie de l'objet ArrayList<E>
```

- par rapport à la lecture dans l'ensemble:

```
E get(int index)        retourne l'élément d'indice index
boolean contains (E o)          teste si l'ensemble contient o
int indexOf (E o)               retourne l'indice de l'élément o ou -1 s'il n'existe pas
```

- par rapport à l'écriture dans l'ensemble

```
boolean add(E e)        ajouter un élément e à la fin du tableau (retourner true)
void add(int index, E e)    insérer l'élément e à l'indice index
E set(int index, E e)   remplace l'élément d'indice index par e et retourner l'ancien
E remove(int index)     supprimer l'élément d'indice index et le retourne
E remove(E e)           supprimer e et retourn true s'il existait, false sinon
```

ArrayList<E> & Utilisation

Comment utiliser la classe ArrayList<E>?

- Directement

  On instancie directement un objet ArrayList<E>

```
ArrayList<Domino> jeu = new Array<Domino>();
```

  Puis, utiliser à travers ses méthodes d'instance

```
jeu.add(new Domino(2,3));
```

- Indirectement

  Créer une classe encapsulant un objet ArrayList<E>

  Notion "héritage"

  ne considérons pas dans ce cours

Static

- Certaines méthodes sont avec static, certaines ne le sont pas

```
static void f(){}        int g(){}
```

- static s'applique aux variables et aux méthodes
- les méthodes/variables static représentent:
  - les méthodes/variables sont définies pour la classe
  - ils ne sont pas unique pour chaque instance ("à utiliser en commun")
  - peuvent-être utilisées sans instancier un objet de classe

Static

```
class Bebe {

  String name;
  double poids;
  boolean estGarcon;
  static int numBebes = 0;

  public Bebe (String s, double p,
               boolean b){
      this.name = s;
      this.poids = p;
      this.estGarcon = b;
  }
}
```

```
public class MainBebes{

  public static void main (String[] args){
      Bebe.numBebes = 100;
      Bebe b1 = new Bebe ("A", 4.0, true);
      Bebe b2 = new Bebe ("B", 3.9, true);
      Bebe.numBebes = 2;

      System.out.println(b1.numBebes);
      System.out.println(b2.numBebes);
  }
}
```

Sortie du programme?

Static

```
class Bebe {

  String name;
  double poids;
  boolean estGarcon;
  static int numBebes = 0;

  public Bebe (String s, double p,
               boolean b){
      this.name = s;
      this.poids = p;
      this.estGarcon = b;
  }
}
```

```
public class MainBebes{

  public static void main (String[] args){
      Bebe.numBebes = 100;
      Bebe b1 = new Bebe ("A", 4.0, true);
      Bebe b2 = new Bebe ("B", 3.9, true);
      Bebe.numBebes = 2;

      System.out.println(b1.numBebes);
      System.out.println(b2.numBebes);
  }
}
```

Sortie du programme?

```
2
2
```

Variables static

Compter le nombre de bébé

```
class Bebe {

  String name;
  double poids;
  boolean estGarcon;
  int numBebes = 0;

  public Bebe (String s, double p,
               boolean b){
      this.name = s;
      this.poids = p;
      this.estGarcon = b;
      numBebes += 1;
  }
}
```

```
public class MainBebes{

  public static void main (String[] args){

      Bebe b1 = new Bebe ("A", 4.0, true);
      Bebe b2 = new Bebe ("B", 3.9, true);
      Bebe b3 = new Bebe ("C", 3.9, false);

      System.out.println(b1.numBebes);
      System.out.println(b2.numBebes);
  }
}
```

Sortie du programme?

Variables static

Compter le nombre de bébé

```
class Bebe {

  String name;
  double poids;
  boolean estGarcon;
  int numBebes = 0;

  public Bebe (String s, double p,
               boolean b){
      this.name = s;
      this.poids = p;
      this.estGarcon = b;
      numBebes += 1;
  }
}
```

```
public class MainBebes{

  public static void main (String[] args){

      Bebe b1 = new Bebe ("A", 4.0, true);
      Bebe b2 = new Bebe ("B", 3.9, true);
      Bebe b3 = new Bebe ("C", 3.9, false);

      System.out.println(b1.numBebes);
      System.out.println(b2.numBebes);
  }
}
```

Sortie du programme?

```
1
1
```

Variables static

Compter le nombre de bébé

```
class Bebe {

  String name;
  double poids;
  boolean estGarcon;
  static int numBebes = 0;

  public Bebe (String s, double p,
               boolean b){
      this.name = s;
      this.poids = p;
      this.estGarcon = b;
      numBebes += 1;
  }
}
```

```
public class MainBebes{

  public static void main (String[] args){

      Bebe b1 = new Bebe ("A", 4.0, true);
      Bebe b2 = new Bebe ("B", 3.9, true);
      Bebe b3 = new Bebe ("C", 3.9, false);

      System.out.println(b1.numBebes);
      System.out.println(b2.numBebes);
  }
}
```

Sortie du programme?

```
1
1
```

Variables static

Compter le nombre de bébé

```
class Bebe {

  String name;
  double poids;
  boolean estGarcon;
  static int numBebes = 0;

  public Bebe (String s, double p,
               boolean b){
      this.name = s;
      this.poids = p;
      this.estGarcon = b;
      numBebes += 1;
  }
}
```

```
public class MainBebes{

  public static void main (String[] args){

      Bebe b1 = new Bebe ("A", 4.0, true);
      Bebe b2 = new Bebe ("B", 3.9, true);
      Bebe b3 = new Bebe ("C", 3.9, false);

      System.out.println(b1.numBebes);
      System.out.println(b2.numBebes);
  }
}
```

Sortie du programme?

```
1   3
1   3
```

Méthodes static

```
class Bebe {

  String nom;

  static void ditBonjour(Bebe b){
      System.out.println(b.nom + “Bonjour”);
  }
}
```

```
class Bebe {

  String nom;

  void ditBonjour(){
      System.out.println(nom + “Bonjour”);
  }
}
```

Pourquoi toujours static dans main ?

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