



Introduction à la programmation et au développement d'applications en Java

Constructeurs et initialisation de classes

Introduction

- ❑ État initial
- ❑ Initialisation des champs
- ❑ Constructeurs
- ❑ Chaînage de constructeurs et visibilité
- ❑ Blocs d'initialisation
- ❑ Ordre de construction et d'initialisation

Initialisation d'objets

- ❑ Quand un objet est créé, il doit avoir un état initial cohérent
- ❑ Java donne automatiquement un état par défaut, mais ce n'est pas toujours suffisant
- ❑ L'objet à créer pourrait avoir besoin d'une initialisation explicite des valeurs ou même d'exécuter du code avant que sa référence puisse finalement être délivrée au client

Mécanismes d'initialisation d'objets

Dans une classe, Java offre trois mécanismes pour initialiser un objet :

1. Initialisation de champs
2. Constructeurs
3. Blocs d'initialisation

État initial des champs

- ❑ Rappel : une variable doit toujours être explicitement initialisée avant d'être utilisée (`int a; int b = a + 1; // NON`)
- ❑ En revanche, les champs (attributs) d'une classe reçoivent une valeur par défaut au moment de la création de l'objet, valeur automatiquement donnée par Java
- ❑ Valeur par défaut : valeur « zéro », c'est-à-dire :

types entiers : 0	types réels : 0.0
booléen : false	char : '\u0000'
types références (classes) : null	
- ❑ Mais vous pouvez initialiser les champs si ces valeurs ne conviennent pas

Initialisation des champs

Permet de spécifier une valeur initiale du champs **en même temps** que la déclaration

```
class Terre {  
    long circonferEnMiles = 24901;  
  
}
```

❑ affectation simple

Initialisation des champs

Permet de spécifier une valeur initiale du champs **en même temps que la déclaration**

- ❑ affectation simple
- ❑ expression calculée

```
class Terre {  
  
    long circonferMiles = 24901;  
    long circonferKm = (long) (24901 * 1.6d);  
  
}
```

Initialisation des champs

Permet de spécifier une valeur initiale du champs **en même temps que la déclaration**

- ❑ affectation simple
- ❑ expression calculée
- ❑ peut référencer d'autres champs

```
class Terre {  
  
    long circonferEnMiles = 24901;  
    long circonferEnKm =  
        (long) ( circonferEnMiles * 1.6d );  
  
}
```


Initialisation des champs

Permet de spécifier une valeur initiale du champs **en même temps que la déclaration**

- ❑ affectation simple
- ❑ expression calculée
- ❑ peut référencer d'autres champs
- ❑ peut utiliser un appel de méthode

```
class Terre {  
  
    long circonferMiles = 24901;  
    long circonferKm =  
        Math.round(circonferMiles * 1.6d);  
  
}
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Ne sont pas des méthodes au sens du langage :
 - ❑ pas de type de retour
 - ❑ même nom que la classe
 - ❑ chaque classe en a au moins un

```
public class Vol {  
  
    int nbPassagers;  
    int nbPlaces;  
  
    public Vol() {  
        nbPassagers = 150;  
        nbPlaces = 0;  
    }  
  
}
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Ne sont pas des méthodes au sens du langage :
 - ❑ pas de type de retour
 - ❑ même nom que la classe
 - ❑ chaque classe en a au moins un

```
public class Vol {  
  
    int nbPassagers;  
    int nbPlaces = 150;  
  
    public Vol() {}  
  
}
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Ne sont pas des méthodes au sens du langage :
 - ❑ pas de type de retour
 - ❑ même nom que la classe
 - ❑ chaque classe en a au moins un

```
public class Passager {  
  
    private int bagagesEnregistres;  
    private int bagagesLibres;  
    private double coutParBagage;  
  
}
```

```
Passager tom = new Passager();
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Si pas de constructeur, le compilateur Java en génère un en interne automatiquement

```
public class Passager {  
  
    private int bagagesEnregistres;  
    private int bagagesLibres;  
    private double coutParBagage;  
  
    public Passager() { }  
  
}
```

```
Passager tom = new Passager();
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Si pas de constructeur, le compilateur Java en génère un interne automatiquement
- ❑ Plusieurs constructeurs possibles (différentes listes de paramètres)

```
Passager tom = new Passager();
```

```
public class Passager {  
  
    private int bagagesEnregistres;  
    private int bagagesLibres;  
    private double coutParBagage;  
  
    public Passager() { }  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
}
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Si pas de constructeur, le compilateur Java en génère un en interne automatiquement
- ❑ Plusieurs constructeurs possibles (différentes listes de paramètres)

```
Passager tom = new Passager();  
Passager lea = new Passager(2);
```

```
public class Passager {  
  
    private int bagagesEnregistres;  
    private int bagagesLibres;  
    private double coutParBagage;  
  
    public Passager() { }  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
}
```

Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Si pas de constructeur, le compilateur Java en génère un interne automatiquement
- ❑ Plusieurs constructeurs possibles (différentes listes de paramètres)

```
Passager tom = new Passager();  
Passager lea = new Passager(2);
```

```
public class Passager {  
  
    private int bagagesEnregistres;  
    private int bagagesLibres;  
    private double coutParBagage;  
  
public Passager() {}  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
}
```


Constructeurs

- ❑ Constructeur : code exécuté **au moment de la création** de l'objet pour établir l'état initial
- ❑ Si pas de constructeur, le compilateur Java en génère un interne automatiquement
- ❑ Plusieurs constructeurs possibles (différentes listes de paramètres)

```
Passager tom = new Passager();  
Passager lea = new Passager(2);
```

```
public class Passager {  
  
    private int bagagesEnregistres;  
    private int bagagesLibres;  
    private double coutParBagage;  
  
    public Passager() {  
    }  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
}
```

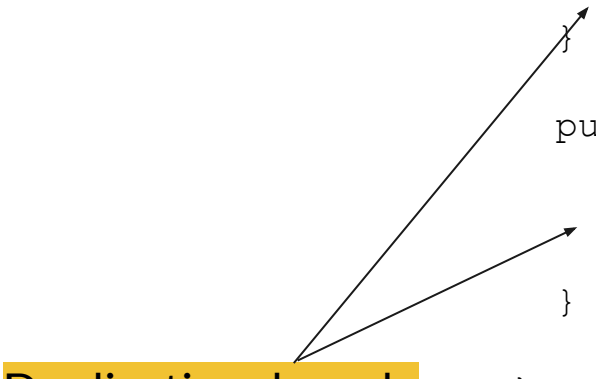
Chaînage de constructeurs

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this.bagagesLibres = bagagesLibres;  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
}
```

```
Passager tom = new Passager();  
Passager lea = new Passager(2);  
Passager leo = new Passager(2, 3);
```

Chaînage de constructeurs

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this.bagagesLibres = bagagesLibres;  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
}
```



Duplication de code

```
Passager tom = new Passager();  
Passager lea = new Passager(2);  
Passager leo = new Passager(2, 3);
```

Chaînage de constructeurs

Un constructeur peut appeler un autre constructeur :

- ❑ **this(...)** avec la liste des paramètres
- ❑ doit être la **première ligne** du code du constructeur

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres) ;  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
}
```

```
Passager tom = new Passager();  
Passager lea = new Passager(2);  
Passager leo = new Passager(2, 3);
```

Visibilité des constructeurs

- ❑ Contrôler la visibilité, surtout s'ils sont plusieurs
- ❑ On utilise les modificateurs d'accès
- ❑ On contrôle quel code peut exécuter des créations spécifiques

```
public class Passenger {  
  
    public Passenger() {  
  
    }  
  
    public Passenger(int bagagesLibres) {  
  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passenger(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres);  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
}
```

Visibilité des constructeurs

- ❑ Contrôler la visibilité, surtout s'ils sont plusieurs
- ❑ On utilise les modificateurs d'accès
- ❑ On contrôle quel code peut exécuter des créations spécifiques

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres);  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
    public Passager(double coutParBagage) {  
        this.coutParBagage = coutParBagage;  
    }  
  
}
```

Visibilité des constructeurs

- ❑ Contrôler la visibilité, surtout s'ils sont plusieurs
- ❑ On utilise les modificateurs d'accès
- ❑ On contrôle quel code peut exécuter des créations spécifiques

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres);  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
    public Passager(double coutParBagage) {  
        this.coutParBagage = coutParBagage;  
    }  
  
}  
  
Passager radin = new Passager(0.01d);
```

Visibilité des constructeurs

- ❑ Contrôler la visibilité, surtout s'ils sont plusieurs
- ❑ On utilise les modificateurs d'accès
- ❑ On contrôle quel code peut exécuter des créations spécifiques

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres);  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
    private Passager(double coutParBagage) {  
        this.coutParBagage = coutParBagage;  
    }  
  
}
```

```
Passager radin = new Passager(0.01d);
```


Visibilité des constructeurs

- ❑ Contrôler la visibilité, surtout s'ils sont plusieurs
- ❑ On utilise les modificateurs d'accès
- ❑ On contrôle quel code peut exécuter des créations spécifiques

```
public class Passenger {  
  
    public Passenger() {  
  
    }  
  
    public Passenger(int bagagesLibres) {  
        this(bagagesLibres > 1 ? 25.0d : 40.0d);  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passenger(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres);  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
    private Passenger(double coutParBagage) {  
        this.coutParBagage = coutParBagage;  
    }  
  
}
```

Visibilité des constructeurs

- ❑ Contrôler la visibilité, surtout s'ils sont plusieurs
- ❑ On utilise les modificateurs d'accès
- ❑ On contrôle quel code peut exécuter des créations spécifiques

```
Passager tom = new Passager(2);  
Passager lea = new Passager(2, 3);
```

```
public class Passager {  
  
    public Passager() {  
  
    }  
  
    public Passager(int bagagesLibres) {  
        this(bagagesLibres > 1 ? 25.0d : 40.0d);  
        this.bagagesLibres = bagagesLibres;  
    }  
  
    public Passager(int bagagesLibres,  
                    int bagagesEnregistres) {  
        this(bagagesLibres);  
        this.bagagesEnregistres = bagagesEnregistres;  
    }  
  
    private Passager(double coutParBagage) {  
        this.coutParBagage = coutParBagage;  
    }  
  
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces;  
    private char categorie;
```

```
    public Vol() {  
        nbPlaces = 150;  
        nbPassagers = 0;  
    }
```

```
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces;  
    private char categorie;
```

```
    public Vol() {  
        nbPlaces = 150;  
        nbPassagers = 0;  
    }  
}
```

```
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;
```

```
    public Vol() {  
        nbPlaces = 150;  
    }  
}
```

```
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;  
  
    public Vol() {  
  
    }  
  
    public Vol(int numero) {  
  
        this.numero = numero;  
    }  
  
    public Vol(char categorie) {  
  
        this.categorie = categorie;  
    }  
  
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;  
    private boolean[] placeEstDispo;  
  
    public Vol() {  
        placeEstDispo = new boolean[nbPlaces];  
        for (int i = 0; i < nbPlaces; i++) {  
            placeEstDispo[i] = true;  
        }  
    }  
  
    public Vol(int numero) {  
  
        this.numero = numero;  
    }  
  
    public Vol(char categorie) {  
  
        this.categorie = categorie;  
    }  
  
}
```


Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {
    private int nbPassagers, numero, nbPlaces=150;
    private char categorie;
    private boolean[] placeEstDispo;

    public Vol() {
        placeEstDispo = new boolean[nbPlaces];
        for (int i = 0; i < nbPlaces; i++) {
            placeEstDispo[i] = true;
        }
    }

    public Vol(int numero) {
        this();
        this.numero = numero;
    }

    public Vol(char categorie) {
        this();
        this.categorie = categorie;
    }
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;  
    private boolean[] placeEstDispo;  
  
    public Vol() {  
        placeEstDispo = new boolean[nbPlaces];  
        for (int i = 0; i < nbPlaces; i++) {  
            placeEstDispo[i] = true;  
        }  
    }  
  
    public Vol(int numero) {  
        this();  
        this.numero = numero;  
    }  
  
    public Vol(char categorie) {  
        this();  
        this.categorie = categorie;  
    }  
}
```

partagé par tous les constructeurs

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur
- ❑ Inclure le code entre accolades en dehors de toute méthode ou de tout constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;  
    private boolean[] placeEstDispo;  
  
    public Vol() {  
        placeEstDispo = new boolean[nbPlaces];  
        for (int i = 0; i < nbPlaces; i++) {  
            placeEstDispo[i] = true;  
        }  
    }  
  
    public Vol(int numero) {  
        this();  
        this.numero = numero;  
    }  
  
    public Vol(char categorie) {  
        this();  
        this.categorie = categorie;  
    }  
}
```

partagé par tous les constructeurs

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur
- ❑ Inclure le code entre accolades en dehors de toute méthode ou de tout constructeur

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;  
    private boolean[] placeEstDispo;  
  
    {  
        placeEstDispo = new boolean[nbPlaces];  
        for (int i = 0; i < nbPlaces; i++) {  
            placeEstDispo[i] = true;  
        }  
    }  
  
    public Vol() { }  
  
    public Vol(int numero) {  
        this();  
        this.numero = numero;  
    }  
  
    public Vol(char categorie) {  
        this();  
        this.categorie = categorie;  
    }  
}
```

Blocs d'initialisation

- ❑ Les blocs d'initialisation sont **partagés** entre constructeurs
- ❑ exécutés comme si le code était placé au début de chaque constructeur
- ❑ Inclure le code entre accolades en dehors de toute méthode ou de tout constructeur
- ❑ Utilisé notamment pour les *classes anonymes*

```
public class Vol {  
    private int nbPassagers, numero, nbPlaces=150;  
    private char categorie;  
    private boolean[] placeEstDispo;  
  
    {  
        placeEstDispo = new boolean[nbPlaces];  
        for (int i = 0; i < nbPlaces; i++) {  
            placeEstDispo[i] = true;  
        }  
    }  
  
    public Vol() { }  
  
    public Vol(int numero) {  
        this.numero = numero;  
    }  
  
    public Vol(char categorie) {  
        this.categorie = categorie;  
    }  
}
```

Ordre d'initialisation / construction

1. Initialisation des champs

```
class TropDInitTueLInit {  
  
    private int leChamp = 1;  
  
    public int getLeChamp() { return leChamp; }  
  
}
```

```
TropDInitTueLInit c =  
    new TropDInitTueLInit();
```

```
System.out.println(c.getLeChamp()); // 1
```

Ordre d'initialisation / construction

1. Initialisation des champs

2. Blocs d'initialisation

```
class TropDInitTueLInit {  
    private int leChamp = 1;  
  
    public int getLeChamp() { return leChamp; }  
  
    { leChamp = 2; }
```

```
}
```

```
TropDInitTueLInit c =  
    new TropDInitTueLInit();
```

```
System.out.println(c.getLeChamp()); // 2
```

Ordre d'initialisation / construction

1. Initialisation des champs

2. Blocs d'initialisation

3. **Constructeurs**

```
class TropDInitTueLInit {  
    private int leChamp = 1;  
  
    public int getLeChamp() { return leChamp; }  
  
    { leChamp = 2; }  
  
    public TropDInitTueLInit() {  
        leChamp = 3;  
    }  
}
```

```
TropDInitTueLInit c =  
    new TropDInitTueLInit();
```

```
System.out.println(c.getLeChamp()); // 3
```


Ordre d'initialisation / construction

1. Initialisation des champs

2. Blocs d'initialisation

3. Constructeurs

```
class TropDInitTueLInit {  
    private int leChamp = 1;  
  
    public int getLeChamp() { return leChamp; }  
  
    { leChamp = 2; }  
  
    public TropDInitTueLInit() {  
        leChamp = 3;  
    }  
}
```

```
TropDInitTueLInit c =  
    new TropDInitTueLInit();
```

```
System.out.println(c.getLeChamp()); // 3
```

Erreur la plus fréquente : code dans le bloc d'initialisation invalidé dans un constructeur

Résumé

- ❑ Les objets devraient être créés avec un état cohérent (pas forcément utile, mais au moins qui ne provoque pas d'erreurs)
- ❑ Les initialisations de champs fournissent des valeurs lors de la déclaration des champs
- ❑ Chaque classe a au moins un constructeur (écrit ou généré automatiquement) et peut en avoir plusieurs qui diffèrent par leur signatures
- ❑ Un constructeur peut en appeler un autre (chaînage, première ligne)
- ❑ Les blocs d'initialisation (entre accolades) permettent de partager du code entre constructeurs
- ❑ Gardez en tête l'ordre d'initialisation / construction