



Le langage Java

Classe et Instance

Objectifs Pédagogiques

À l'issue de cette formation, vous serez en mesure de :

- ✓ Comprendre la structure d'un objet
- ✓ Savoir instancier un objet

Programme détaillé ou sommaire

Notion de bloc et portée des variables

Notion de variable globale

Les limites de l'approche impérative

L'approche objet

Notion de classe

Création de classe

Notion de package

Déclaration de package

Utilisation d'une classe

Notion d'instance

Notion de bloc



Notion de bloc

Définition: un bloc est délimité par une accolade ouvrante et une accolade fermante

```
{
```

```
}
```

En Java, le bloc structure le code.

Notion de bloc et sous-blocs

Règle: un bloc peut contenir des sous-blocs

```
{  
  {  
  }  
  {  
  }  
}
```

Blocs et instructions

Un bloc peut contenir des instructions et des déclarations de variables.

Notion de portée de la variable.

```
{  
  {  
    int a = 15;  
    System.out.println(a);  
  }  
  {  
    // La variable a n'est pas utilisable ici  
  }  
}
```

Blocs et portée des variables

Une variable n'est visible que dans son bloc et ses sous-blocs

```
{  
    int a = 15;  
    System.out.println(a); // La variable a est utilisable ici  
    {  
        System.out.println(a); // La variable a est utilisable ici  
    }  
    System.out.println(a); // La variable a est utilisable ici  
}
```

 System.out.println(a); // La variable a N'EST PAS utilisable ici

Conséquences de la portée des variables

- Une classe est un bloc qui contient des variables et des méthodes.
 - Une **variable déclarée dans le bloc de la classe** est utilisable dans toutes les méthodes de cette classe.
- Une méthode est un bloc
 - une **variable déclarée dans une méthode** n'est pas utilisable à l'extérieur de la méthode

```
public class Personne {  
    String nom;  
  
    void direBonjour() {  
        int age = 10;  
        System.out.println("Je m'appelle " + nom + " et j'ai " + age + " ans");  
    }  
}
```

Diagram illustrating variable scope in Java:

- Bloc de la classe:** Indicated by a dashed line from the `public class` line to the closing brace of the class.
- Bloc de la méthode:** Indicated by a dashed line from the `void direBonjour()` line to the closing brace of the method.

Annotations in the code:

- The variable `nom` is circled in orange, indicating it is declared in the class scope.
- The variable `age` is circled in green, indicating it is declared in the method scope.

Bloc et structure if

L'exécution d'un bloc peut être soumise à une condition avec l'instruction if.

```
if (a > 5) {  
    a++;  
    System.out.println(a);  
}
```

Bloc et structure de contrôle for

L'exécution d'un bloc peut être répétée plusieurs fois avec l'instruction for.

```
int a = 15;  
for (int i = 0; i < 10; i++) {  
    a++;  
    System.out.println(a);  
}
```

Les limites de l'approche impérative



Les limites de l'approche impérative

Dans la vie courante nous manipulons des concepts et non des variables isolées.

Quelques exemples de concepts:

- Un montant est le regroupement de 2 informations : une valeur + une devise
- Un poids est le regroupement de 2 informations : une valeur + une unité
- Une adresse est le regroupement de plusieurs informations: numéro de rue, libellé de rue, code postal ville.

Votre ancienne adresse

Numéro/Etage : _____

Rue : _____

Code postal : _____

Ville : _____

Votre nouvelle adresse

Numéro/Etage : _____

Rue : _____

Code postal : _____

Ville : _____

Les limites de l'approche impérative

Pour représenter une adresse postale, on peut utiliser 4 variables:

```
int numeroRue = 5;  
String libelleRue = "rue des Tulipes";  
int codePostal = 14500;  
String ville = "Vire";
```

Qu'on peut propager dans les appels de méthodes :

```
afficherAdresse(numeroRue, libelleRue, codePostal, ville);  
imprimerAdresse(numeroRue, libelleRue, codePostal, ville);  
...  
envoyerCourrier(numeroRue, libelleRue, codePostal, ville);
```

Les limites de l'approche impérative

Mais, que se passe t'il si on me demande d'ajouter des informations ?

Exemple:

Le pays !

Les limites de l'approche impérative

```
int numeroRue = 5;  
String libelleRue = "rue des Tulipes";  
int codePostal = 14500;  
String ville = "Vire";  
String pays = "France";
```

Tous les appels de méthode doivent être modifiés:

```
afficherAdresse(numeroRue, libelleRue, codePostal, ville, pays);  
imprimerAdresse(numeroRue, libelleRue, codePostal, ville, pays);  
...  
envoyerCourrier(numeroRue, libelleRue, codePostal, ville, pays);
```

En approche impérative, les modifications de structure d'un concept peuvent être très coûteuses !

L'approche objet



L'approche objet

Et si je pouvais déclarer une variable de type AdressePostale :

```
AdressePostale adresse1 = ... ;
```

Dans les appels de méthodes j'aurais quelque chose de ce type :

```
afficherAdresse(adresse1);  
imprimerAdresse(adresse1);  
...  
envoyerCourrier(adresse1);
```


L'idée serait de créer notre propre type AdressePostale, ce qui rendrait la manipulation des données beaucoup plus simple !

Comment créer ses propres types ?

On souhaite pouvoir **créer ses propres types**, par exemple pour représenter des concepts de la vie courante: une **Adresse postale**.

L'idée va être de passer de l'écriture de gauche (ci-dessous), à l'écriture de droite.

```
int numeroRue = 5;  
String libelleRue = "rue des Tulipes";  
int codePostal = 14500;  
String ville = "Vire";
```



```
AdressePostale adresse1 = ?;
```

Le développeur doit pouvoir

Définir un type complexe : AdressePostale

Déclarer des nouvelles variables sur la base de ce type complexe

Création d'une classe



Une classe est constituée a minima :

- d'une signature: mot clé **class** + **nom de la classe**
- d'un bloc avec les différentes variables composant une adresse postale (+ des méthodes comme on le verra par la suite).

Les méthodes

Une **méthode** est un **bloc** précédé de ce qu'on appelle la **signature de la méthode**

Une méthode est **TOUJOURS** dans une classe !

```
package org.diginamic;
```

```
public class Bonjour {
```

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

```
    }
```

```
}
```

Signature

A yellow box containing the word "Signature" with an arrow pointing to the signature of the main method in the code block above.

Notion de variable « globale »

Règle: Les variables globales (qu'on appelle en réalité variables d'instance), sont utilisables dans toutes les méthodes (En effet, rappelez vous qu'une variable est visible dans son bloc et ses sous-blocs)

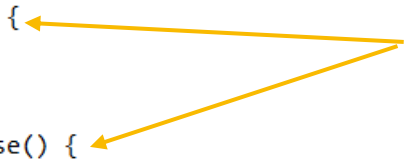
```
package org.diginamic;

public class Compteur {

    // valeur est visible dans toutes les méthodes de la classe: c'est une variable globale
    int valeur;

    void incrementer() {
        valeur++;
    }

    void faireAutreChose() {
        valeur--;
    }
}
```



méthodes

Création d'instances



Déclaration d'une classe

Déclaration de la classe AdressePostale

```
class AdressePostale {  
    int numeroRue;  
    String libelleRue;  
    int codePostal;  
    String ville;  
}
```

Comment créer une "variable"
de type AdressePostale ?

```
class EssaiAdresse {  
    public static void main(String[] args) {  
        AdressePostale a = ?;  
    }  
}
```


Instance de classe

Utilisation de l'opérateur **new**

```
class EssaiAdresse {  
    public static void main(String[] args) {  
        AdressePostale adr = new AdressePostale();  
    }  
}
```

- Création d'une variable **adr** de type AdressePostale
- On dit que **adr** est une instance de la classe AdressePostale

Que fait l'opérateur new ?

Utilisation de l'opérateur new

```
class EssaiAdresse {  
    public static void main(String[] args) {  
        AdressePostale adr = new AdressePostale();  
    }  
}
```

- Réservation d'un espace mémoire : **adr** est une référence vers cette adresse mémoire
- Initialisation des variables internes
- invocation de AdressePostale() appelée également constructeur (cf. chapitre consacré)

MÉMOIRE (Heap)

Adresse 0001 :

```
numeroRue = 0  
libelleRue = null;  
codePostal = 0;  
Ville = null;
```

Créations de plusieurs instances

Un objet est créé à partir d'une classe => cet objet s'appelle une **instance** de la classe

```
class EssaiAdresse {  
    public static void main(String[] args){  
        AdressePostale adr1 = new AdressePostale();  
        AdressePostale adr2 = new AdressePostale();  
    }  
}
```

2 **instances** de la classe AdressePostale
appelées **adr1** et **adr2**

A ce stade nous n'avons pas encore donné de valeurs au numéro de rue, libellé de rue, code postal et ville de chacune de nos variables. Nous n'avons fait que créer les espaces mémoires avec des variables initialisées par défaut (0, null, false, etc.).

Créations de plusieurs instances

```
AdressePostale adr1 = new AdressePostale();
```

Pour l'instant les variables stockées dans `adr1` ont des valeurs par défaut

`AdressePostale adr1`

```
int numeroRue;    // vaut 0
String libelleRue; // vaut null
int codePostal;   // vaut 0
String ville;     // vaut null
```

Comment donner des valeurs aux variables globales (d'instance) ?

AdressePostale contient des variables.

Comment leur donner une valeur ?

```
class EssaiAdresse {  
    public static void main(String[] args){  
  
        AdressePostale adr1 = new AdressePostale();  
        adr1.numeroRue = 5;  
        adr1.libelleRue = "des Maréchaux";  
        adr1.codePostal = 44100;  
        adr1.ville = "Nantes";  
    }  
}
```

```
class AdressePostale {  
  
    int numeroRue;  
    String libelleRue;  
    int codePostal;  
    String ville;  
}
```

On verra ultérieurement
l'utilisation des
constructeurs et de
méthodes get/set.

Les packages



Package

Regroupement fonctionnel de classes

- classes utilitaires
- classes applicatives

Même nom de classe possible si dans des packages différents

- `java.sql.Date`
- `java.util.Date`

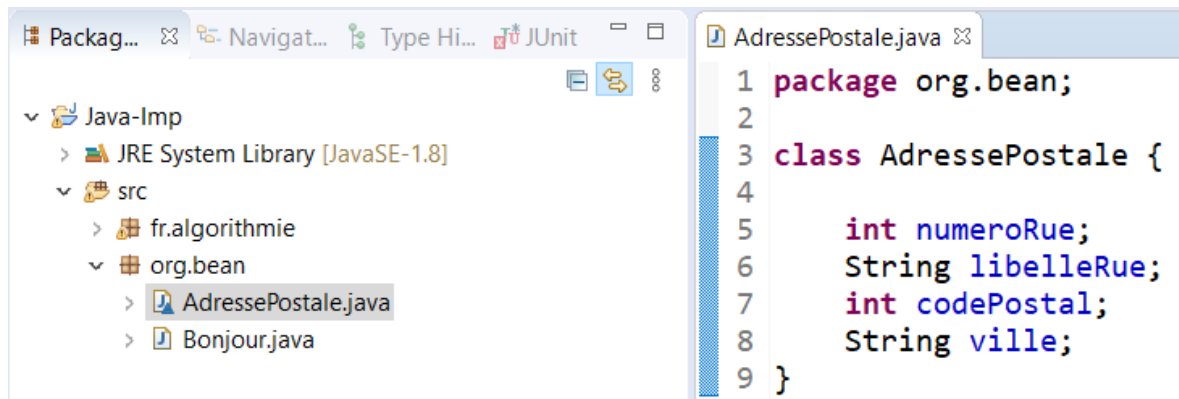
Déclaration de package

Pour indiquer à quel package une classe appartient

➤ `package nomDuPackage;`

Règle

➤ Obligatoirement la première instruction dans le fichier **.java**



Package et impact

Pour utiliser une classe située dans un autre package il faut l'importer.

- Utilisation de la clause **import**.
- Exemple: **import** java.util.List
- Seules les classes du package **java.lang** sont toutes importées par défaut
 - String*
 - System*

Syntaxe

import de toutes les classes d'un package
import nomdupackage.*;

import d'une seule classe
import nomdupackage.NomClasse;

Atelier (TP)

OBJECTIFS : Créer des classes et les utiliser

DESCRIPTION :

- Dans le TP qui suit vous allez devoir créer des classes et apprendre à les utiliser.