



Le langage Java

Modificateurs d'accès, attributs et méthodes

Objectifs Pédagogiques

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

- ✓ Comprendre la structure d'une méthode
- ✓ Savoir appeler une méthode
- ✓ Savoir utiliser des variables et méthodes static

Programme détaillé ou sommaire

Les modificateurs d'accès

Déclaration des attributs dans une classe

Valeurs par défaut des attributs

Les variables locales

Déclaration des méthodes et appels

Utilisation des attributs dans une classe

Passage par valeur / passage par référence

Zoom sur les variables et méthodes static

Les modificateurs d'accès

Le modificateur public



```
package fr.diginamic.beans;
```

```
public class Voiture {
```

```
    String marque;
```

```
    String modele;
```

```
    double prix;
```

```
}
```

```
package fr.diginamic.app;
```

```
import fr.diginamic.beans.Voiture;
```

```
public class MonApplication {
```

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

```
        Voiture v = new Voiture();
```

```
    }
```

```
}
```



Si une classe a le modificateur **public** elle peut être importée par une classe située dans un **autre package**.

Les modificateurs d'accès

Absence de modificateur

```
package fr.diginamic.beans;
```

```
class Voiture {  
  
    String marque;  
    String modele;  
    double prix;  
  
}
```

La classe **n'est pas visible en dehors de son package.**

Une classe située dans un autre package **ne peut pas l'importer.**

```
src  
├── fr.diginamic  
│   ├── app  
│   │   └── MonApplication.java  
│   └── beans  
│       └── Voiture.java
```

```
package fr.diginamic.app;  
  
import fr.diginamic.beans.Voiture;  
  
public class MonApplication {  
  
    public static void main(String[] args) {  
  
        Voiture v = new Voiture();  
  
    }  
}
```




Declaration des attributs dans une classe

Déclarés de préférence au début

Attributs d'instance

valeurs différentes pour chaque instance.

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



Valeurs par défaut des attributs d'instance

Les attributs peuvent être initialisés lors de leur déclaration:

```
int a = 10;
```

Valeurs par défaut pour les attributs (d'instance ou static) non initialisés :

- 0 pour les nombres,
- false pour les booléens,
- '\u0000' pour un char (caractère "nul")
- **null** pour les références d'objet

Les variables locales

➤ Les variables locales

- Déclarées dans un bloc
 - *De méthode*
 - *Dans une structure conditionnelle (if)*
 - *Dans une boucle (for, while, do/while)*
- Visibles uniquement dans le bloc de déclaration, sinon **erreur de compilation**.

```
package com.applications;

public class EssaiBloc {

    public static void main(String[] args) {

        float montantCompte = -2000;

        if (montantCompte < 0){
            int tauxDecouvert = 5;
            montantCompte = montantCompte-montantCompte*tauxDecouvert/100;
        }

        System.out.println(tauxDecouvert);
    }

}
```


Valeurs par défaut des variables locales

- Une variable locale non initialisée n'a pas de valeur

- Exemple avec erreur de compilation



The local variable val may not have been initialized

```
public class TestVarNonInit {  
  
    public static void main(String[] args) {  
  
        // La variable locale val n'a pas de valeur  
        int val;  
  
        // Erreur de compilation  
        val++;  
  
    }  
}
```

Déclaration des méthodes

Une méthode est constituée

- d'une **signature**
- d'un **corps** délimité par un bloc {...} qui suit la signature

Accessibilité (ex: public, private)

Type de retour

Nom de la méthode

paramètres

```
[modificateur] typeRetour nomMethode (String param1, int param2, ...) {  
    ...  
}
```

Déclaration des méthodes – Exemple (1)

Déclaration d'une méthode **public sans paramètre** qui ne retourne pas de valeur:
void

```
public void nomMethode () {  
  
}
```

Déclaration des méthodes – Exemple (2)

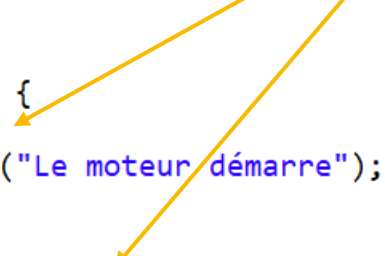
Déclaration d'une méthode public qui retourne une valeur de type **int**:

```
public int addition(int a, int b){  
    return a + b;  
}
```

Comment invoquer une méthode ? (1/2)

Si elle est dans la même classe, il suffit de **l'invoquer** par son nom avec les paramètres attendus.

```
class Voiture {  
    public void demarrer() {  
        mettreLeContact();  
        System.out.println("Le moteur démarre");  
    }  
    public void mettreLeContact() {  
        System.out.println("Contact enclenché");  
    }  
}
```



Comment invoquer une méthode ? (2/2)

Si la méthode appartient à une autre classe, il faut utiliser une **instance** de la classe pour invoquer la méthode.

```
Operation ope = new Operation();  
int resultat = ope.addition(8, 7);
```

```
public class Operation {  
  
    int addition(int a, int b) {  
        int c = a + b;  
        return c;  
    }  
}
```

Il existe un autre type de méthode (méthode static) qui s'invoque différemment.

Atelier (TP)

OBJECTIFS : TP permettant de mettre en œuvre les méthodes d'instance.

DESCRIPTION :

- Dans les TP n°3 vous allez implémenter quelques méthodes d'instance et/ou apprendre à en utiliser.

Variables et méthodes static



Déclaration des attributs dans une classe

Déclarés de préférence au début

```
public class AdressePostale {
```

Attribut de classe

Valeur de **classe**, commune à toutes les instances et modifiable.

```
→ static int nbDepartements = 101;
```

```
→ static final int NB_REGIONS = 18;  
}
```

Constantes

Valeur de classe, commune à toutes les instances et **non modifiable**.

Utilisation des attributs de classe

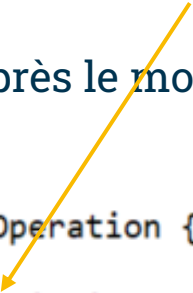
```
public class TestAdressePostale {  
  
    public static void main(String[] args) {  
  
        // Affichage de la variable static nbDepartements  
        System.out.println(AdressePostale.nbDepartements);  
  
        // Modification de la variable static nbDepartements  
        AdressePostale.nbDepartements = 102;  
  
        // La variable NB_REGIONS est static et constante  
        System.out.println(AdressePostale.NB_REGIONS);  
  
    }  
}
```

```
public class AdressePostale {  
  
    static int nbDepartements = 101;  
  
    static final int NB_REGIONS = 18;  
}
```

Déclaration d'une méthode static

Méthode de classe, avec le mot clé **static**

Ce mot clé se positionne après le modificateur de visibilité.



```
public class Operation {  
    public static int addition(int a, int b) {  
        int c = a + b;  
        return c;  
    }  
}
```


Invoker une méthode static

Une méthode **static** s'invoque directement sur la **classe** elle-même et non une instance de la classe.

On l'appelle une **méthode de classe**.

```
public class Operation {  
    static int addition(int a, int b) {  
        int c = a + b;  
        return c;  
    }  
}
```

```
public class TestOperation {  
    public static void main(String[] args) {  
        int resultat = Operation.addition(5, 3);  
        System.out.println(resultat);  
    }  
}
```

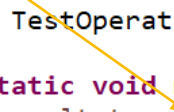


Différence entre méthode static et d'instance

static = appel sur la classe elle-même

```
public class Operation {  
    static int addition(int a, int b) {  
        int c = a + b;  
        return c;  
    }  
}
```


```
public class TestOperation {  
    public static void main(String[] args) {  
        int resultat = Operation.addition(5, 3);  
        System.out.println(resultat);  
    }  
}
```



Non static = appel sur une instance

```
public class Operation {  
    int addition(int a, int b) {  
        int c = a + b;  
        return c;  
    }  
}
```

```
public class TestOperation {  
    public static void main(String[] args) {  
        Operation op = new Operation();  
        int resultat = op.addition(5, 3);  
        System.out.println(resultat);  
    }  
}
```



Compléments sur les méthodes

Passage par valeur vs passage par référence (voir diapos suivantes)

- passés par **valeur** pour les types primitifs
- passés par **référence** pour les objets

Pas de retour attendu

- type de retour **void**
- l'expression **return**; peut être utilisée pour une sortie explicite avant la fin de la méthode (par exemple sur un test conditionnel)

Retour d'un objet à la fin d'une méthode

- mot-clé **return obligatoire** suivi de l'objet
- éventuellement **return null**

Le passage par valeur

Uniquement pour les types primitifs: la valeur de la variable est copiée

```
public class TestReference {  
  
    public static void main(String[] args) {  
        int a = 1;  
  
        // Java va créer une variable b qui est une copie  
        // de la variable a : transfert de valeur  
        int b = a;  
  
        // Affiche : 1 1  
        System.out.println(a + " " + b);  
  
        // j'augmente a de 1  
        a++;  
  
        // Affiche : 2 1  
        System.out.println(a + " " + b);  
    }  
}
```

Le passage par référence

Uniquement pour les objets: la valeur de l'adresse mémoire est copiée

```
public static void main(String[] args) {  
  
    // Création de c1  
    Compteur c1 = new Compteur();  
  
    // Affiche 0  
    c1.afficher();  
  
    // c2 référence le même adresse mémoire que c1  
    Compteur c2 = c1;  
  
    // Incrémente val de 1  
    c2.increment();  
  
    // Affiche 1 pour c1 et c2  
    c1.afficher();  
    c2.afficher();  
  
}
```

```
public class Compteur {  
  
    private int val;  
  
    public void increment(){  
        val++;  
    }  
  
    public void afficher(){  
        System.out.println(val);  
    }  
  
}
```


Le passage par référence

Uniquement pour les objets: : la valeur de l'adresse mémoire est copiée

Référence	Mémoire
Compteur c1	= new Compteur(); // Création d'un objet adresse : 0001
Compteur c2	= new Compteur(); // Création d'un objet adresse : 0002
Compteur c3 = c2;	// c3 n'est pas un nouvel objet (new pas utilisé) // c3 référence la même adresse mémoire que c2 : 0002

Dans cet exemple j'ai 3 références mais seulement 2 objets en mémoire.
Si je modifie c3 alors je modifie également c2.

Exécution d'une application

Une classe peut être exécutée
si elle comporte la Méthode static void ***main***

```
public static void main(String[] args){  
    // code à exécuter  
}
```

Atelier (TP)

OBJECTIFS : TP permettant de mettre en œuvre les méthodes static.

DESCRIPTION :

- Dans les TP n°4 vous allez implémenter quelques méthodes static et/ou apprendre à en utiliser.