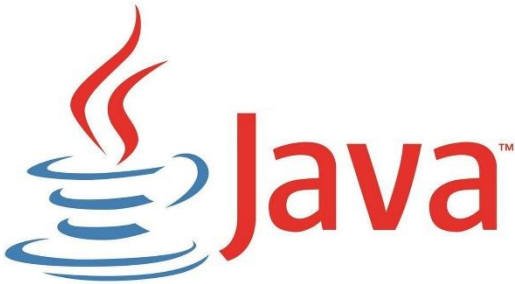
Java

- Chapitre : Les Modificateurs des Identificateurs-

Non-Access Modifiers

Ichrak MEHREZ



Les Classes

- *[Access][modifiers]* class *ClassName [extends SuperClassName] [implements InterfaceName]*

- *Access* représente un des mots clés :

  - *public* signifie que la classe peut être utilisée par tous et qu'elle est donc visible de l'extérieur

  - *package* permet à une classe d'être visible à l'intérieur de son package uniquement (par défaut)

Les Classes

- *[modifiers]* class *ClassName [extends SuperClassName] [implements InterfaceName]*

- *modifiers* représente un des mots clés :

  - *abstract* signifie que la classe ne peut pas être instanciée: classe *template*. Il est nécessaire de créer une classe concrète qui l'étende, et d'instancier cette classe concrète (Héritage).

  - *final* signifie qu'on ne peut pas hériter de cette classe. Les comportement de cette classe ne peuvent pas être modifié

Les variables

- *[Access][modifiers] type nomVar*
- *Access* représente un des mots clés :
- *public* permet à une variable d'être accessible dans tout le code sans aucune restriction, du moment où la classe qui la déclare est visible elle aussi.
- *private* permet à une variable d'être accessible uniquement dans le code de la classe qui la déclare.

Les variables

- *[Access][modifiers] type nomVar*
- *Access* représente un des mots clés :
- *protected* permet à une variable d'être visible à l'intérieur du package de la classe qui la déclare. Il permet aussi à la variable d'être visible dans le code des sous-classes de la classe qui la déclare, même s'elles sont déclarées dans d'autres packages.
- *package* permet à une variable d'être visible uniquement à l'intérieur du package de la classe qui la déclare.

Les variables

- *[Access][modifiers] type nomVar*
- *Modifier* représente un des mots clés :
- ***static*** implique que la variable aura une seule copie (classe copie) qui sera partagée par toutes les instances de la classe.

  Une variable statique peut être utilisée sans créer une instance de la classe; une variable statique est accessible après le chargement de la classe en mémoire par la JVM.
- ***final*** déclare une variable comme constante

Les variables

- *[Access][modifiers] type nomVar*
- *Modifier* représente un des mots clés :
- ***volatile*** permet aux threads de voir la dernière valeur modifiée de la variable (une sorte de synchronisation sans la charge supplémentaire du verrouillage).
- ***transient*** indique que la variable ne doit pas être prise en compte lors de la sérialisation de l'objet qui définit cette variable.

Les variables

- *[Access][modifiers] type nomVar*
- *Modifier* représente un des mots clés :
- ***volatile*** permet aux threads de voir la dernière valeur modifiée de la variable (une sorte de synchronisation sans la charge supplémentaire du verrouillage).
- ***transient*** indique que la variable ne doit pas être prise en compte lors de la **sérialisation** de l'objet qui définit cette variable.

On appelle sérialisation la technique utilisée pour écrire ou lire des objets dans un flux binaire. La sérialisation peut servir à archiver des objets dans un fichier et permet de transmettre des objets à travers un réseau ; elle intervient aussi pour permettre à des applications de communiquer par des appels à des méthodes distantes,

Les méthodes

- *[Access][modifiers] type_retour nomFonction (Liste_param)*
- *Access* représente un des mots clés :
- ***public*** permet à une méthode d'être accessible dans tout le code sans aucune restriction (à condition que la classe qui la déclare est visible elle aussi).
- ***private*** permet à une méthode d'être accessible uniquement dans le code de la classe qui la déclare.

Les méthodes

- *[Access][modifiers] type_retour nomFonction (Liste_param)*
- *Access* représente un des mots clés :
- ***protected*** permet à une méthode d'être visible à l'intérieur du package de la classe qui la déclare. Il permet aussi à la méthode d'être visible dans le code des sous-classes de la classe qui la déclare, même s'elles sont dans d'autres packages.
- ***package*** permet à une méthode d'être visible uniquement à l'intérieur du package de la classe qui la déclare (par défaut).

Les méthodes

- *[Access][modifiers] type_retour nomFonction (Liste_param)*
- *modifiers* représente un des mots clés :
- ***static :*** Une méthode statique peut être invoquée sans créer une instance de la classe; une méthode statique est accessible au chargement de la classe en mémoire par la JVM.
- ***abstract*** : Une méthode abstraite est une méthode *template* qui n'implémente pas un code. Une méthode abstraite:
  - peut être définit uniquement dans une classe abstraite.
  - peut avoir trois niveaux d'accès : *public, protected, et package*.
  - ne peut pas utiliser les modificateurs *final, private, et static*.

Les méthodes

- *[Access][modifiers] type_retour nomFonction (Liste_param)*
- *modifiers* représente un des mots clés :
- ***final*** empêche une méthode d'être redéfinit ; ce qui veut dire que son comportement ne peut être modifié (héritage).
- ***synchronized*** permet à une méthode d'être exécuter par un seul thread à la fois.
- ***strictfp*** (strict floating-point) permet à une méthode d'être conforme avec le standard concernant la gestion des nombres réels.
- ***native*** indique que la méthode est implémenter dans un autre langage (exemple C++).

Les Blocs d'initialisation

- Les blocs initialisation permettent d'initialiser les valeurs lors du ***chargement*** de la classe ou lors de l'***instanciation***

- Chargement de la classe :

```
Client client;
```

- Création ou l'instanciation de la classe :

```
client = new Client();
```

Les Blocs d'initialisation Static

- Ces blocs s'exécutent une seule fois lors du chargement de la classe et ne s'exécutent pas lors des instanciation de la classe ,ils sont lancés avant l'appel des constructeurs.

- On peut créer des blocs ***static*** comme ceci : *static { ... }*

```
public class Client {

    private int id;
    Client(){  System.out.println("constructeur");}
    static{  System.out.println("Bloc static d'initialisation 1");}
    static{  System.out.println("Bloc static d'initialisation 2");}

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

```
Bloc static d'initialisation 1
Bloc static d'initialisation 2
------------------------------------------------
BUILD SUCCESS
------------------------------------------------
```

Les Blocs d'initialisation Static

- Ces blocs s'exécutent une seule fois lors du chargement de la classe et ne s'exécutent pas lors des instanciation de la classe ,ils sont lancés avant l'appel des constructeurs.

- On peut créer des blocs ***static*** comme ceci : *static { ... }*

```
public class Client {

    private int id;
    Client(){  System.out.println("constructeur");}
    static{  System.out.println("Bloc static d'initialisation 1");}
    static{  System.out.println("Bloc static d'initialisation 2");}

    public static void main(String[] args) {
        Client c = new Client();
    }
}
```

```
Bloc static d'initialisation 1
Bloc static d'initialisation 2
constructeur
------------------------------------------------------------------------
BUILD SUCCESS
------------------------------------------------------------------------
```

Les Blocs d'initialisation d'instance

- Ils se déclarent entre deux crochets {} sans le mot clé static et s'exécutent lors de l'instanciation de la classe, ils s'exécutent un par un suivant l'ordre définie dans la classe.

```
public class Client {

    private int id;
    Client(){  System.out.println("constructeur");}
    {
        System.out.println("Bloc instance 1");
    }
    static{  System.out.println("Bloc static d'initialisation 1");}
    {
        System.out.println("Bloc instance  2");
    }
    static{  System.out.println("Bloc static d'initialisation 2");}

    public static void main(String[] args) {
        Client c = new Client();
    }
}
```

```
Bloc static d'initialisation 1
Bloc static d'initialisation 2
Bloc instance 1
Bloc instance  2
constructeur
------------------------------------
BUILD SUCCESS
```