Bases du langage
JAVA

Mama AMAR

Agenda

Les objectifs de cette présentation sont les suivants:

- Structure d’une application JAVA
- Types de données
- Variables
- Constantes
- Opérateurs arithmétiques
- Conversion de type
- Opérateurs d’incrémentation et de décrémentation

Application Java

Le code source d'un programme Java est contenu dans **un** ou **plusieurs** fichiers d'extension .java

- Une seule classe **publique** par fichier;
- Le nom du fichier doit être le même que celui de la classe;

Structure d'une classe Java

```
/**
 */
public class ExempleClasse {
   public static void main(String [] args) {
      //instructions
       ..............
      //instructions
   }
}
```

Nommage

- Notation similaire à **camelCase** pour les noms de variables (première lettre en minuscule et les mots suivants avec première lettre en majuscule)
  ex: **m**on**T**aux**I**nteret
- Notation majuscule séparée par le symbole _ pour les constantes
- Ex: TAUX_INTERET
- **Texte en commentaire** : on mettra au début du texte les symboles **/*** et le texte sera fermé par les symboles ***/**

Classe

- Le code source d'une classe commence par le mot-clé class suivi de son contenu :

class *<nom de la classe>* {

<contenu de la classe>

}

- Une classe peut contenir une ou plusieurs **méthodes**

Nom de la classe

- Le fichier source contient la classe et le nom du fichier doit correspondre exactement avec le nom de la classe
  - **Par convention, le nom d'une classe commence toujours par une majuscule.**

Méthode

- Par convention, le nom d'une **méthode** commence toujours par une **minuscule**.
  - Méthode: une suite d'instructions Java qui réalisent une fonctionnalité précise
- Une méthode peut avoir des **paramètres** et ceux-ci obéissent aux mêmes règles de nommage

```
public static void main(String [] args){
        System.out.println(‘’Ceci est un bloc”);
}
```

Instructions

- Le code Java est constitué d'une suite d'instructions
- Chaque instruction est terminée par le symbole ;
- Une instruction peut être sur une ou plusieurs lignes

Bloc d'Instructions

- Un bloc d'instructions sera délimité par les symboles **{** et **}**.
- Un bloc peut être imbriqué dans une méthode
- Un bloc peut être imbriqué dans un autre bloc

```
public static void main(String [] args){
        {
//ceci est un bloc independent
                System.out.println(‘’Ceci est un bloc”);
        }
}
```

Commentaires

- Les commentaires seront inclus en utilisant deux approches
- **Ligne en commentaire** : on mettra au début de la ligne deux barres obliques **//**
- **Texte en commentaire** : on mettra au début du texte les symboles **/*** et le texte sera fermé par les symboles ***/**

Commentaires

```
/**
 * @(#)ExempleClasse.java
 *
 *
 *
 */
public class ExempleClasse {
   public static void main(String [] args) {
      //instructions
       ..............
      //instructions
   }
}
```

Commentaires sur plusieurs lignes

Commentaire sur une ligne

Types primitifs de données

Même types qu'en C

- int
- float
- double
- short
- long
- char
- void

Deux nouveaux types pour Java

- boolean
- byte

Plus la classe :

- String

Vérification typage

- Compilateur vérifie systématiquement l'usage des variables et constantes par rapport à la déclaration

Déclaration de variables

- **Une seule variable**
  Type nomvariable ;
- **Plusieurs variables**
  Type nomvariable1, nomvariable2 ;
- Ex:
  int nombreAnnee;
  double tauxImpositionCumule;

Déclaration de constante

- **Cas d'une seule constante**

final Type NOM_CONSTANTE = valeur1 ;

- **Cas de plusieurs constantes**

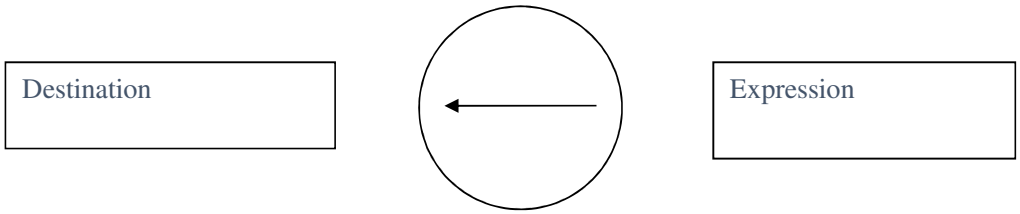
final Type NOM_CONST = val1 ;

final NOM_CONST_2 = val2

final NOM_CONST_N = valn ;

- Ex:
  final double TAUX_INTERET = 0.12;

Expressions Java et opérations arithmétiques



- Opérateur d'affectation en Java est le symbole **=**
- Destination: variable ou une constante
- Expression: peut être une variable, une constante ou une composition de variables et constantes reliés par des opérateurs.

Opérateurs arithmétiques

Opérateur	Opération	Ordre d'évaluation
( )	Permet des regroupements	Gauche - droite
*, / , %	Multiplication, division, modulo	Gauche - droite
+, -	Addition, soustraction	Gauche - droite
=	affectation	Droite - gauche

Précédence

- Ordre de précédence des différents opérateurs disponibles en partant de la première ligne
- Dans le cas d'ambiguïté dans l'ordre d'évaluation, il sera préférable de recourir aux parenthèses et de regrouper ensemble les opérations nécessaires.

Division entière

- La division d'un entier par un autre nombre entier donne aussi un entier.

```
public class ExempleClasse {
    public static void main(String [] args) {
        //déclaration de variables
        int var1=13 , var2=2 ;
        int resultat ;
        //calcul arbitraire
                resultat = var1 / var2;
                System.out.println("Le resultat est:"+resultat);
    }
}
```

En sortie on aura :

```
Le resultat est:6
```

Modulo

- Modulo **%** : donne le nombre ***entier*** correspondant au reste de la division.

```
public class ExempleClasse {
    public static void main(String [] args) {
        //déclaration de variables
        int var1=13 , var2=2 ;
        float resultat ;
        int reste;
        //calcul arbitraire
                resultat = var1 / var2;
                reste =  var1 % var2;
                System.out.println("Le resultat est: "+resultat);
                System.out.println("Le reste est: "+reste);
    }
}
```

et en sortie, on aura :

```
Le resultat est: 6.0
Le reste est: 1
```

Division par zéro

- Les cas possibles seront :
- Division d'entiers dont le dénominateur est zéro.

Compilation : OK

Exécution : message d'erreur indiquant une division par zéro.

- Division de réels dont le dénominateur est zéro.

Compilation : OK

Exécution : OK et le résultat de la division est affiché comme : Infinity.

- Division de réels avec les deux opérandes égaux à zéro.

Compilation : OK

Exécution : OK et le résultat de la division est affiché comme : NaN

Exemple: Division par zéro

```
public class ExempleClasse {
    public static void main(String [] args) {
        //déclaration de variables
        float var3=0, var4=0;
        float var5;
        //calcul arbitraire
                var5 =  var3 / var4;
                var3 = var5+3;
                System.out.println("Le resultat est: "+var5);
                System.out.println("Le resultat addition est: "+var3);
    }
}
```

et le résultat est en sortie:

```
Le resultat est: NaN
Le resultat addition est: NaN
```

Conversion de type lors des opérations

- il arrive souvent que les variables et constantes utilisées dans le calcul ne soient pas toutes du même type.
- En Java, l'approche utilisée est basée sur la conversion (**casting**) de type automatique lors des *calculs*
- les opérandes au niveau de chaque opérateur seront *implicitement* (automatiquement) convertis au type de l'opérande ayant la plus haute précision.

Exemple

```
public class ExempleClasse {
    public static void main(String [] args) {
        //déclaration de variables
        double var1;
        double var2 = 4.3;
        int var3 = 5;
        //calcul arbitraire
                var1 =  var2 + var3;
                System.out.println("Le resultat est: "+var1);
    }
}
```

Règles de conversion

Type opérande 1	Type opérande 2	Conversion	Type résultat opération
double	char, byte, short, int, long, float	double	double
float	char, byte, short, int, long	float	float
long	char, byte, short, int	long	long
int	char, byte, short	int	int
short	char, byte	int	int
byte	char	int	int

Conversion explicite

- La syntaxe Java pour la conversion explicite sera alors :

  (**Type**) (*expression*)

Exemple: Conversion explicite

```
public class ExempleClasse {
    public static void main(String [] args) {
       //déclaration de variables
       double var1;
       double var2 = 4.3;
       int var3 = 5;
       //calcul arbitraire
              var1 =  var2 + (double)var3;
              System.out.println("Le resultat est: "+var1);
    }
}
```

Opérateurs d'incrémentation et décrémentation

- **Opérateur d'incrémentation**: **++**
  ajoute la valeur 1 à l'opérande

- **Opérateur de décrémentation**: **– –**
  retranche la valeur 1

Opérateurs d'incrémentation et décrémentation

- **Version préfixe** : L'opérateur est appliqué **avant** l'utilisation de l'opérande dans l'expression.

```
public class ExempleClasse {
    public static void main(String [] args) {
        //déclaration de variables
        int var1 = 5;
        int var2 = 10;
        int inc, dec;
        //calcul arbitraire
                inc =  ++var1 +3 ;
                dec = --var2 +4;
                System.out.println("var1 est: "+var1);
                System.out.println("Increment est: "+inc);
                System.out.println("var2 est: "+var2);
                System.out.println("Decrement est: "+dec);
    }
}
```

```
var1 est: 6
Increment est: 9
var2 est: 9
Decrement est: 13
```

Opérateurs d'incrémentation et décrémentation

- **Version postfixe** : L'opérateur est appliqué après l'utilisation de l'opérande dans l'expression.

```
public class ExempleClasse {
    public static void main(String [] args) {
        //déclaration de variables
        int var1 = 5;
        int var2 = 10;
        int inc, dec;
        //calcul arbitraire
                inc =  var1++ +3 ;
                dec = var2-- +4;
                System.out.println("var1 est: "+var1);
                System.out.println("Increment est: "+inc);
                System.out.println("var2 est: "+var2);
                System.out.println("Decrement est: "+dec);
    }
}
```

```
var1 est: 6
Increment est: 8
var2 est: 9
Decrement est: 14
```

Autres notations utiles en Java

- Dans le cas d'opérations sur une seule valeur comme dans l'exemple suivant :
  Var1 = var1 + 13 ;

- On peut recourir à une notation beaucoup plus courte mais qui est équivalente soit :
  Var1 += 13 ;

Conclusion

Dans cette présentation, on a vu les points suivants:

- Structure d'une application JAVA
- Types de données
- Variables
- Constantes
- Opérateurs arithmétiques
- Conversion de type
- Opérateurs d'incrémentation et de décrémentation