

Java Syntaxe

Syntaxe

- Syntaxe issue du langage C
- Les instructions sont terminées par le point-virgule ;
- Les instructions sont contenues dans des blocs de code {}
- Le langage est case sensitive
- L'opérateur point . permet de manipuler les membres d'une classe ou d'une instance

Syntaxe

- Java est un langage fortement typé
 - les variables et propriétés doivent être typées lors de la déclaration
 - le type de retour des méthodes est précisé
- Les variables doivent être déclarées avant de pouvoir être utilisées
 - les variables se trouvent dans les méthodes
 - leur portée est celle du bloc de déclaration
 - la déclaration peut être effectuée juste avant l'utilisation

Syntaxe

- Les commentaires
 - `//` commentaire jusqu'à la fin de la ligne
 - `/* ... */` bloc de commentaire
 - `/** ... */` bloc de documentation
 - cf. outil javadoc

Les types en Java

- Tout est objet en Java
 - toutes les classes sont construites à partir d'une classe `Object`
- Les types primitifs existent quand même
 - ils sont manipulés directement
 - passage par valeur
 - ils possèdent un équivalent objet
 - avec autoboxing depuis le JDK 1.5

Types primitifs

Type	Description	Valeur par défaut	Wrapper
boolean	booléen	false	Boolean
char	caractère unicode (2 octets)	\u0000	Character
byte	entier signé sur 1 octet	0	Byte
short	entier signé sur 2 octets	0	Short
int	entier signé sur 4 octets	0	Integer
long	entier signé sur 8 octets	0	Long
float	réel sur 4 octets	0.0	Float
double	réel sur 8 octets	0.0	Double

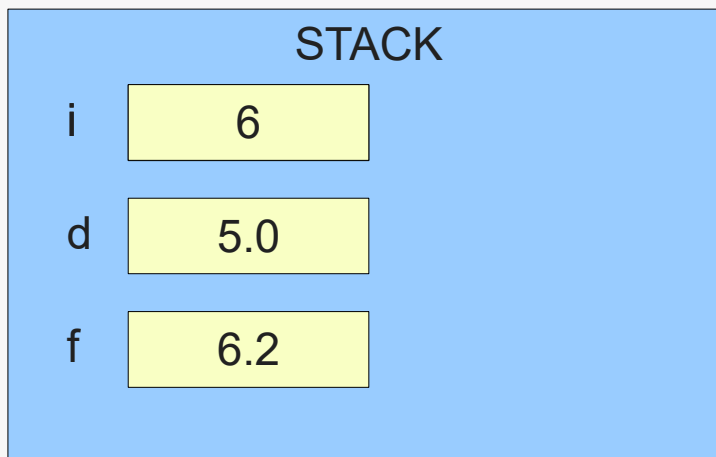
Syntaxe

- Déclaration de variables

```
int i = 3;  
double d = 5.0;  
float f = 6.2F;  
i = (int) d;
```

pour forcer le type du littéral

cast pour forcer le passage de double vers int



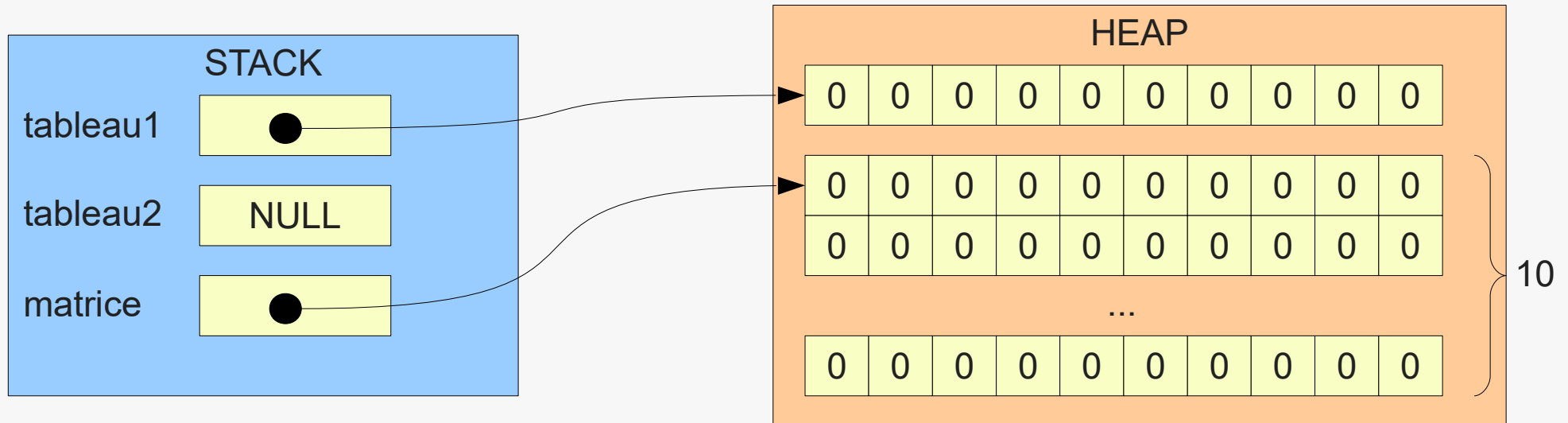
Syntaxe

- Déclaration de tableaux

```
int[] tableau1;  
int tableau2[];  
  
tableau1 = new int[20];  
  
int[][] matrice = new int[10][10];
```

pour créer le tableau

un tableau à toujours une taille fixe qui doit être précisée avant l'affectation des valeurs



Syntaxe

- Utilisation des tableaux
 - les éléments d'un tableau sont numérotés à partir de 0
 - l'indice entre crochets peut varier de 0 à taille-1

```
tableau1[0] = 3;  
tableau1[30] = 10;  
int indice = 3;  
tableau1[indice] = 25;  
int r = tableau1[9];  
  
matrice[2][5] = 23;
```

le compilateur ne vérifie pas les bornes d'un tableau
cela générera une erreur à l'exécution

Exception in thread "main" [java.lang.ArrayIndexOutOfBoundsException: 30](#)
at org.antislashn.Hello.main([Hello.java:18](#))

Syntaxe

- Les chaînes de caractères
 - utilise une classe `java.util.String`
 - la valeur d'une chaîne ne peut pas être modifiée
 - l'opérateur `+` est utilisé pour la concaténation
 - `String` possède un ensemble de méthodes permettant d'effectuer de nombreuses opérations : recherche, comparaison, tests, ...
 - cf. la documentation

Syntaxe

- Les chaînes de caractères

```
String s1 = "Hello,";  
String s2 = "world";  
String s3 = s1 + " " + s2;  
String s4 = new String("Bonjour");
```

crée une chaîne à partir d'une chaîne littérales

concaténation de deux chaînes

crée une chaîne sur le tas

Syntaxe

- Opérateurs mathématiques

Opérateur	Description	Exemple
=	affectation	<code>int i = 3;</code>
*	multiplication	<code>int x = i*3;</code>
/	division (attention aux divisions entre entier)	<code>int x = i/3;</code>
%	modulo (reste d'une division entière)	<code>int x = 10 % 3;</code>
+	addition	<code>int x = 20 + i;</code>
-	soustraction	<code>int x = 10 - 6;</code>

Syntaxe

- Opérateurs de comparaison

Opérateur	Description
==	égalité
!=	inégalité
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

- Opérateurs logiques

Opérateur	Description	Exemple
!	NON	<code>if (!vide) ...</code>
&&	ET	<code>if (i==3 && b<10) ...</code>
 	OU	<code>if (i==3 b==10) ...</code>

Syntaxe

- Opérateurs binaires

Opérateur	Description	Exemple
&	ET	<code>i & 0x00FF;</code>
	OU	<code>i 0x0001;</code>
^	OU EXCLUSIF (XOR)	<code>i ^ 0x00FF;</code>
~	complément	<code>~i;</code>
>>	décalage à droite	<code>i>>2;</code>
>>>	décalage à droite avec remplissage 0	<code>-1>>>2;</code>
<<	décalage à gauche	<code>i<<2;</code>

Syntaxe

- Autres opérateurs

Opérateur	Description	Exemples
.	point, opérateur d'accès aux membres d'une instance	<code>voiture.rouler();</code>
()	parenthèses, opérateur de regroupement d'expression	<code>(2+3)*5;</code>
[]	crochet, opérateur de gestion des tableaux	<code>tab[2] = 3;</code>
instanceof	vérifie si l'instance est d'un type particulier	<code>if(d instanceof De)</code>
new	instancie une classe	<code>De d = new De();</code>
-	opérateur unaire, signe opposé	<code>i = -i;</code>
? :	opérateur ternaire	<code>int i = j>0 ? j : -j;</code>

Opérateurs ++ et --

- Opérateur d'incrémentation ++
 - augmente de 1 la valeur d'un variable
 - ++i : pré-incrémentation
 - l'incrémentation est effectuée avant l'évaluation

```
i = 3;  
int k = ++i;
```

k et i valent 4

- i++ : post-incrémentation
 - l'incrémentation est effectuée après l'évaluation

```
i = 3;  
int k = i++;
```

k vaut 3 et i vaut 4

Opérateurs ++ et --

- Opérateur d'incrémentation --
 - diminue de 1 la valeur d'un variable
 - --i : pré-décrémentement
 - la décrémentement est effectuée avant l'évaluation

```
i = 3;  
int k = --i;
```

k et i valent 2

- i-- : post-décrémentement
 - la décrémentement est effectuée après l'évaluation

```
i = 3;  
int k = i--;
```

k vaut 3 et i vaut 2

Affectation raccourcie

- Il existe un raccourci pour les expressions du type

```
variable = variable OP expression;
```

- ou OP est un opérateur (+,-,/,%,>>,...)
- expression renvoie un résultat

- Raccourci

```
variable OP= expression;
```

- Exemples

```
i += 3;  
k %= 2;  
j >>= 4;
```

Organisation du code

- Un programme Java est constitué d'un ensemble de classes
 - une classe représente les éléments manipulés et les opérations qui y sont associées
 - propriétés et méthodes
- Une classe modélise un objet qui est manipulé par l'application
 - Compte, Banque, Client, Fournisseur, ...

Organisation du code

- Une classe publique par fichier
 - le nom de la classe correspond au nom du fichier
- Les packages permettent d'organiser les classes
 - arborescence de répertoires
 - la directive `package` indique le package auquel appartient la classe
 - doit être la première instruction
 - l'import des packages se fait par la directive `import`
 - pas nécessaire pour `java.lang`

Méthode `main()`

- Point d'entrée d'une application
 - méthode invoquée par la JVM
 - une application peut posséder plusieurs `main()` dans des classes différentes
 - signature :

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

- le tableau `args` correspond aux éventuels paramètres passés à la ligne de commande

Instructions conditionnelles

- Bloc `if`
 - syntaxe

```
if(<condition>) <bloc_condition_vraie> [else <bloc_condition_fausse>]
```

- un bloc de code est soit
 - une instruction (terminée par `;`)
 - plusieurs instructions entre accolades `{ }`
- la condition doit renvoyer une valeur booléenne

```
if(a==b)
    System.out.println("a == b");
else
    System.out.println("a != b");
```

Instructions conditionnelles

- Bloc `if`

```
if( a!=b && d>10){  
    System.out.println("a != b");  
    System.out.println("d > 10");  
}
```

```
if(cde.equals("START"))  
    System.out.println("Commande envoyée : START");
```

Instructions conditionnelles

- Opérateur ?:
 - syntaxe

```
<condition> ? <instruction_si_vraie> : <instruction_si_fausse>]
```

```
int max = (a>b) ? a : b;
```


Instructions itératives

- `while`
 - tant que la condition est vraie, le bloc de code est exécuté
 - bloc de code = corps de la boucle
- syntaxe

```
while(<condition>) <bloc_de_code>
```

```
int i=0;
while(i<10){
    System.out.println("i == "+i);
    ++i;
}
```

la variable i est incrémentée

Instructions itératives

- `do ... while`
 - l'évaluation de la condition est effectuée après l'exécution du corps de boucle
 - celui-ci est donc exécuté au moins une fois
 - syntaxe `do <bloc_de_code> while(<condition>)`

```
do{  
    System.out.println("i == "+i);  
    ++i;  
} while(i<10);
```

Instructions itératives

- `for`
 - boucle constituée de 3 clauses
 - clause d'initialisation
 - exécutée une seule fois, avant l'évaluation de la condition
 - permet d'initialiser une variable
 - clause de condition
 - évaluée avant chaque exécution du corps de boucle
 - si l'évaluation est fausse
 - le corps de boucle n'est pas exécuté
 - on sort de la boucle `for`
 - clause de changement
 - exécutée après chaque exécution du corps de boucle

Instructions itératives

- `for`

- **syntaxe** `for(<initialisation>;<condition>;<changement>) <bloc>`

```
for(int j=0 ; j<tableau1.length ; ++j){  
    System.out.println("j == "+j);  
    System.out.println("valeur de l'élément du tableau "+tableau1[j]);  
}
```

Instructions itératives

- La boucle `for` possède une version permettant de parcourir les éléments d'une collection
 - depuis JDK 1.5
 - tableaux et objets de type `Collection`
 - syntaxe `for(<type> <élément> : <collection>) <bloc>`

```
for(int elt : tableau1)
    System.out.println("élément == "+elt);
```

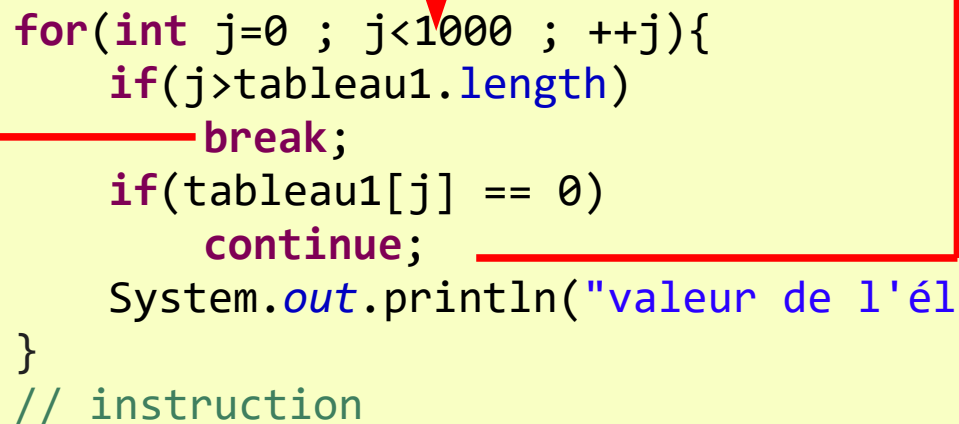
Choix

- `switch`
 - le bloc a exécuté dépend de la valeur de la variable
 - de type entière (`String` permis depuis JDK 1.7)
 - l'exécution d'un bloc ne s'arrête pas automatiquement
 - `break` permet de finir l'exécution
 - un bloc `default` peut être ajouté si aucun bloc `case` n'a été pris en compte
- syntaxe

```
switch(<variable>){  
    case <valeur1> : <bloc1>  
    case <valeur2> : <bloc2>  
    ...  
    default : <bloc_default>  
}
```

Gestion des boucles

- `continue`
 - l'exécution du corps de boucle est arrêtée, mais pas celle de la boucle
- `break`
 - l'exécution de la boucle est arrêtée



```
for(int j=0 ; j<1000 ; ++j){  
    if(j>tableau1.length)  
        break;  
    if(tableau1[j] == 0)  
        continue;  
    System.out.println("valeur de l'élément du tableau "+tableau1[j]);  
}  
// instruction
```

Les annotations

- Appelées également métadonnées
 - préfixées par @
- Les annotations marquent certains éléments du langage
 - classe, propriété, arguments et/ou méthode
- Les annotations peuvent être utilisées par :
 - le compilateur (SOURCE)
 - ou autre outil utilisant le source
 - la machine virtuelle, par introspection (RUNTIME)
 - par un outil externe (CLASS)
 - annotation présente dans le binaire, mais pas utilisable par la JVM

Les annotations

- Annotations standards

- `@Deprecated`

- signale au compilateur que l'élément marqué ne devrait plus être utilisé

- `@Override`

- précise au compilateur que la méthode est redéfinie
 - sur JDK 1.5 ne marque pas les méthodes héritées d'interface

- `@SuppressWarnings`

- indique au compilateur de ne pas afficher certains warnings
 - prend en attribut le nom du warning

Les génériques

- Ajout du JDK 1.5
- Classes qui sont paramétrées par un type
 - se rapproche des templates C++
 - mais produit un code unique
- Améliore la sécurité à l'écriture et à l'exécution
 - le compilateur vérifie que le type passé est compatible avec celui prévu en paramètre
 - pas de cast nécessaire

Les génériques

- Utilisation des symboles < et > pour préciser le type
- L'ensemble des collections Java utilise la généricité
- Seuls les objets sont utilisables avec les types génériques
 - utilisation de l'auto-boxing si nécessaire

Les génériques

- L'utilisation conjointe des types génériques et de la boucle `for` rend le code plus lisible

```
public class IterationNew
{
    public static void main(String[] args)
    {
        ArrayList<Integer> liste = new ArrayList<Integer>();
        for(int i=0 ; i<10 ; i++)
        {
            liste.add(i);
        }

        for(int i : liste)
        {
            System.out.println(i);
        }
    }
}
```

Les génériques

- Vous pouvez utiliser les génériques
 - dans les classes
 - dans les interfaces
 - dans les méthodes
- Pour définir une classe utilisant les génériques il faut les déclarer dans la signature de la classe
 - entre < et >
 - si la classe utilise plusieurs génériques il faut les séparer par des virgules

Les génériques

- Exemple de création d'une classe avec types paramétrés

```
public class Generic_01<T,U>
{
    private T paramT;
    private U paramU;

    public Generic_01()
    {
    }

    public Generic_01(T paramT, U paramU)
    {
        super();
        this.paramT = paramT;
        this.paramU = paramU;
    }

    public T getParamT()
    {
        return paramT;
    }
    ...
    public void setParamU(U paramU)
    {
        this.paramU = paramU;
    }
}
```

Les génériques

- Utilisation de la classe

```
public class Test_01
{
    public static void main(String[] args)
    {
        Generic_01<String, Integer> g = new Generic_01<String, Integer>("Bonjour",3);
        System.out.printf("%s %d\n",g.getParamT(),g.getParamU());
    }
}
```

Les génériques

- Le type des tableaux est réifié
 - vous ne pouvez pas créer de tableaux sur des types paramétrés
 - car le type T n'est pas réifié

```
public class Generic_03<T>
{
    private T[] tableau;

    public Generic_03()
    {
        tableau = new T[10];
    }
    public static void main(String[] args)
    {
        Generic_03<String> t = new Generic_03<String>();
    }
}
```

erreur de compilation