JAVA – LA SYNTAXE

Wijin

v1.0 2024

Plan

- Types de base et encapsulés
- Opérateurs
- Structures conditionnelles
- Boucles

Java - Types de base et encapsulés

Java – Types de base et encapsulés

- Les types de base ne sont pas des objets
  - Aucune méthode
- Pour chaque type de base, il existe une classe correspondante
  - Riches en fonctionnalités

Java – Types de base et encapsulés

Type	Signification	Taille (en octets)	Plage de valeurs acceptées
char	Caractère Unicode	2	'\u0000' ? '\uffff' (0 à 65535)
byte	Entier très court	1	-128 ? +127
short	Entier court	2	-32 768 ? +32 767
int	Entier	4	-2^{31}?$-2{,}147\times10^{9}$? $+2^{31}-1$?$2{,}147\times10^{9}$
long	Entier long	8	-2^{63}?$-9{,}223\times10^{18}$? $+2^{63}-1$?$9{,}223\times10^{18}$
float	Nombre réel simple	4	$\pm2^{-149}$?1.4×10^{-45} ? $\pm2^{128}-2^{104}$? 3.4×10^{38}
double	Nombre réel double	8	$\pm2^{-1074}$?$4{,}9\times10^{-324}$? $\pm2^{1024}-2^{971}$? $1{,}8\times10^{308}$
boolean	Valeur logique (booléen)	1	true (vrai), ou false (faux)

Java – Types de base et encapsulés

Type de base	**Classe associée**
int	Integer
char	Character
float	Float
double	Double
long	Long
boolean	Boolean
byte	Byte
short	Short

Java – Types de base et encapsulés

- Transformations

```
Double monDouble = 3.14159;
String pi = Double.toString(monDouble);
System.out.println("La valeur de pi = " + pi);

String valeurNumerique = "100";
Integer valeur = Integer.parseInt(valeurNumerique);
System.out.println("La valeur est : " + valeur);
```

Java – Types de base et encapsulés

- « Boxing » et « Unboxing »
  - **Boxing** : conversion automatique du type primitif en son équivalent objet

```java
Integer valeur = 100;
System.out.println(valeur);

int valeurPrimitive = 2;
Integer valeurObjetBoxee = valeurPrimitive;
System.out.println("La valeur boxée en Integer est : " + valeurObjetBoxee);
```

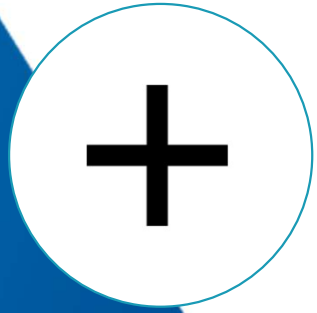
Java – Types de base et encapsulés

- « Boxing » et « Unboxing »
  - **Unboxing** : conversion automatique d'un objet en son équivalent primitif

```
String valeurNumerique = "100";
Integer valeur = Integer.parseInt(valeurNumerique);
System.out.println("La valeur est : " + valeur);

int valeurPrimitiveUnboxing = valeur;
System.out.println("La valeur unboxée en int est : " + valeurPrimitiveUnboxing);
```

Java - Opérateurs



Java - Opérateurs

- Opérateur d'**affectation**
  - Syntaxe : X = Y
  - **Les 2 opérandes doivent être de même type !**

```
Integer valeur1 = 100;
Integer valeur2 = valeur1;
```

Java - Opérateurs

- Opérateurs **arithmétiques**
  - +, -, *, /, %
  - **Opérations arithmétiques de base**

```
// Opérateurs
Integer nombreArticles = 5;
Double prixUnitaire = 12.45;
Double total = nombreArticles * prixUnitaire;
Double rabais = total / 10;
Double aPayer = total - rabais;
System.out.println("Total à payer = " + aPayer);
```

Java - Opérateurs

- Opérateurs **unitaires**
  - ++ et --
  - **Addition ou soustraction de 1**

```
Integer compteur = 10;
compteur = compteur + 1;
System.out.println("Valeur du compteur = " + compteur);
```

  - **Equivalent**

```
Integer compteur = 10;
compteur++;
System.out.println("Valeur du compteur = " + compteur);
```

Java - Opérateurs

- Opérateurs **binaires**
  - ==, !=, >, >=, <, <=
  - **Utilisés pour les tests sur des valeurs numériques**

```
int entier1 = 10;
int entier2 = 10;
System.out.println(entier1 == entier2);

String chaine1 = "Formation Java";
String chaine2 = new String("Formation Java");
System.out.println("Résultat comparaison référence = " + (chaine1 == chaine2));
System.out.println("Résultat comparaison objet = " + chaine1.equals(chaine2));
```

Java - Opérateurs

- Opérateurs **de comparaison**
  - **&&** et **||**
  - **ET logique** et **OU logique**

```
int entier1 = 10;
int entier2 = 20;

String chaine1 = "Formation Java";
String chaine2 = "Formation Java";

System.out.println(entier1==entier2 && chaine1.equals(chaine2));
System.out.println(entier1==entier2 || chaine1.equals(chaine2));
```

Java - Opérateurs

- Opérateur **de négation**
  - **!**
  - **Négation du résultat d'une évaluation**

```
int entier1 = 10;
int entier2 = 20;

String chaine1 = "Formation Java";
String chaine2 = "Formation Java";

System.out.println(!(entier1==entier2) && chaine1.equals(chaine2));
System.out.println(!(entier1==entier2 || chaine1.equals(chaine2)));
```

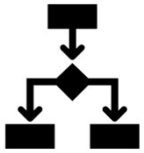
Java - Opérateurs

- Opérateur **ternaire**
  - … **?** … **:** …
  - Permet de remplacer une instruction conditionnelle **if/then/else**

```
int resultat = (entier1==entier2)?10:20;
System.out.println("Résultat = " + resultat);
```

Si a = b, alors on retourne 10, sinon on retourne 20

JAVA - STRUCTURES CONDITIONNELLES



Java – Structures conditionnelles

- Structure **if**
- Structure **switch**

Java – Structures conditionnelles

- Structure **if**
  - Structure de condition **simple** : exécute un bloc de code si une condition est vérifiée
  - Le « else » associé n'est pas obligatoire, et sera présent si l'on doit exécuter un bloc de code dans le cas où le bloc « if » associé n'est pas vérifié
  - Toute évaluation retournant un **booléen** (*true* ou *false*) peut être utilisée dans le bloc « if »

Java – Structures conditionnelles

- Structure **if**

```java
int temperature = 110;

if (temperature < 0) {
    System.out.println("Il gèle !");
} else {
    if (temperature == 0) {
        System.out.println("Point de transformation de l'eau en glace!");
    } else {
        if (temperature > 99) {
            System.out.println("L'eau boue");
        } else {
            System.out.println("L'eau est liquide");
        }
    }
}
```

Java – Structures conditionnelles

- Structure **switch**
  - Le switch apporte de la clarté au code par rapport au « if/else », dans le cas où un traitement différent doit être appliqué selon les différentes valeurs d'une variable
  - Plutôt que de coder plusieurs if ou if/else, on préférera utiliser le switch
  - Le switch peut également être utilisé pour tester des chaines de caractères

Java – Structures conditionnelles

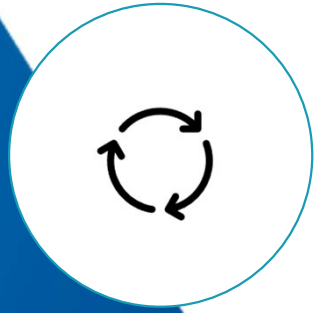
- Structure **switch**
  - **Important** : stipuler la sortie du bloc switch dès lors que l'on a exécuté le traitement correspondant à une valeur
    - instruction **break**
  - Si aucun cas ne correspond à la valeur testée, alors l'entrée nommée default du switch est exécutée

Java – Structures conditionnelles

- Structure **switch**

```
char note = 'D';
switch (note) {
    case 'A':
        System.out.println("Excellent");
        break;
    case 'B':
    case 'C':
        System.out.println("Bien");
        break;
    case 'D':
        System.out.println("Passable");
    case 'F':
        System.out.println("Recommencez!");
        break;
    default:
        System.out.println("Note inconnue");
}
System.out.println("Votre note est " + note);
```

Java - Boucles



Java – Boucles

- 3 types de **boucle**
  - **for**
  - **do … while**
  - while

Java – Boucles

- Boucle **for**

  for (initialisation; test de sortie; incrémentation) {
  }

  - Structure existant dans beaucoup de langages, elle est ainsi constituée :

    - 1 – L'initialisation est exécutée, une fois
    - 2 – Le test est évalué, s'il est false, il y a sortie de la boucle
    - 3 – Le bloc de commandes du for est exécuté
    - 4 – L'incrémentation est exécutée
    - 5 – Le test est évalué : s'il est vrai alors on reprend le processus en 3, sinon on sort de la boucle

Java – Boucles

- Boucle **for**
  - 1er exemple

```
for (int compteurBoucle = 0; compteurBoucle < 5; compteurBoucle++) {
    System.out.println("Valeur courante de compteur = " + compteurBoucle);
}
```

Java – Boucles

- Boucle **for**
  - 2ème exemple

```
String[] tableau = {"ain", "aisne", "allier"};
System.out.println("Taille du tableau = " + tableau.length);
for (int indice = 0; indice < tableau.length; indice++) {
    System.out.println(tableau[indice]);
}
```

Java – Boucles

- Boucle **do … while**
  - Particularité : on passe **au moins une fois** dans la structure
  - Le **while**, qui évalue un booléen en fin de structure, permet de décider de sortir de la boucle ou non
  - C’est au développeur de gérer la sortie de boucle

Java – Boucles

- Boucle **do … while**
  - Exemple

```
String[] villes = {"Paris", "Lyon", "Marseille"};
Integer indice = 0;
do {
    System.out.println(villes[indice]);
    indice++;
} while (indice < villes.length);
```

Java – Boucles

- Boucle **while**
  - Boucle standard, avec dans le **while**, décision d'exécuter ou non le bloc qui suit
  - C'est au développeur de gérer la sortie de boucle

Java – Boucles

- Boucle **while**
  - Exemple :

```
System.out.println("test while :");
Integer increment = 0;
while (increment < villes.length) {
    System.out.println(villes[increment]);
    increment++;
}
```

Java – Boucles

- Instructions **break** et **continue**
  - Forcer la sortie d'une boucle : **break**
  - Forcer le passage à l'itération suivante : **continue**

Java – Boucles

- Instructions **break** et **continue**
  - Exemple :

```
Integer index = 0;
String[] pays = {"France", "Italie", "Monaco", "Suisse", "Belgique", "USA", "Australie"};
while (index < pays.length) {
    if (pays[index].contentEquals("Suisse")) {
        index++;
        continue;
    }
    if (pays[index].contentEquals("USA")) {
        break;
    }
    System.out.println(pays[index]);
    index++;
}
```

Java – Boucles

- Boucle intelligente : **foreach**
  - Pas besoin de gérer la sortie de boucle !
  - La boucle s'appuie sur une **collection** ou un **tableau**

```
for (Type variable : tableau | collection){
      //instructions
}
```

Java – Boucles

- Boucle intelligente : **foreach**
  - A chaque itération, l'élément de la collection ou du tableau est **chargée** dans la variable
  - La progression est **automatique** jusque la fin du tableau
  - **continue** et **break** restent utilisables

Java – Boucles

- Boucle intelligente : **foreach**
  - Exemple :

```
String[] pays = {"France", "Italie", "Monaco", "Suisse", "Belgique", "USA", "Australie"};

for(String paysCourant : pays) {
    if (paysCourant.contentEquals("Suisse")) {
        continue;
    }
    if (paysCourant.contentEquals("USA")) {
        break;
    }
    System.out.println(paysCourant);
}
```

Java - Quiz

Java – Quiz

- Les types de base Java possèdent des méthodes utilitaires
  - **Vrai**
  - **Faux**

Java – Quiz

- String est un type de base de Java
  - **Vrai**
  - **Faux**

Java – Quiz

- Pour réaliser une boucle, Java propose les syntaxes suivantes :
  - **do { … } while (…)**
  - **loop(…) { … }**
  - **for (initialisation;condition;modification) { … }**
  - **while (…) { … }**

Java – Quiz

- Pour réaliser des tests, Java propose les instructions suivantes :
  - L'opérateur ternaire ?:
  - if (…) { … } else { … }
  - switch(…) {case: … ; case: … }
  - choose(…) { then: … ; then … }

Java – Quiz

- Pour réaliser une boucle du type for (Type objet : XXX), il est nécessaire que XXX soit :
  - Un tableau
  - N’importe quel type de base
  - Une collection
  - N’importe quelle classe

MERCI POUR VOTRE ATTENTION

Faites moi part de vos remarques concernant le cours afin qu'il soit amélioré pour les prochaines sessions : nicolas.sanou@wijin.tech