UNIVERSITE VIRTUELLE DU SENEGAL

foo nekk foofu la

Langage JAVA

Mama AMAR

Séquence 2 : Les bases du Langage Java

Chapitre 2 : Les bases du Langage Java

Introduction

Cette section décrit les caractéristiques traditionnelles de la langue, y compris les commentaires, les variables, les types de données, les opérateurs et le flux de contrôle.

Les Commentaires

Java offres deux types de commentaires : Les commentaires du codes qui sont délimités par /*...*/, et // ainsi que les commentaires java doc (connus sous le nom de "commentaires de doc") sont délimités par /**...*/. Les commentaires de Doc peuvent être extraits dans des fichiers HTML à l'aide de l'outil javadoc.

I. Les commentaires du code

- Les commentaires tiennent sur une ligne :
  //tout le reste de la ligne est un commentaire
- Les commentaires sont multi-lignes :
  / ceci est un commentaire*

  ** tenant sur deux lignes*

  **/*

II. Les commentaires sont destinés au système javadoc

Les commentaires JavaDoc décrivent des classes Java, des interfaces, des constructeurs, des méthodes et des attributs. Chaque commentaire JavaDoc est défini dans les délimiteurs de commentaire /**...*/, avec un commentaire par classe, interface ou membre. Ce commentaire devrait apparaître juste avant la déclaration:

code :

*/** ceci est un commentaire spécial destiné au système JavaDoc */*

Note : Les commentaires ne sont pas compilés avec le code.

Les Types de données

Java est un langage fortement typé. Cela signifie que le type de chaque variable doit être déclaré. Le type de données d'une variable détermine les valeurs qu'il peut contenir, ainsi que les opérations qui peuvent y être effectuées. Le langage de programmation Java prend en charge huit types de données primitifs. Un type primitif est prédéfini par le langage et est nommé par un mot-clé réservé. Les valeurs primitives ne partagent pas d'état avec d'autres valeurs primitives. Les huit types de données primitifs pris en charge par le langage de programmation Java sont:

I. Les Entiers

Les types entiers représentent les nombres sans partie décimale. Les valeurs négatives sont autorisées. Java offre quatre types de données entiers présentées dans le tableau.

Type	**Occupation en mémoire**	**Intervalles**	
byte	8 bits	-128	+127
short	16 bits	-32 768	+32 767
int	32 bits	-2147483648	+2147483647
long	64 bits	-9223372036854775808	+9223372036854775808

Tableau 1 : Les types entiers en java

Le choix d'un type doit tenir en compte les valeurs minimales et maximales que vous envisagez de stocker dans de stocker dans la variable afin d'optimiser la mémoire utilisée par la variable.

Le type *int* se révèle le plus pratique dans la majorité des cas. Les types byte et short sont essentiellement destinés à des applications spécialisées, telles que la gestion bas niveau des fichiers ou la manipulation de tableaux volumineux, lorsque l'occupation mémoire doit être réduite au minimum.

En Java, la plage valide des types de nombres entiers ne dépend pas de la machine sur laquelle s'exécute le code. Cela épargne bien des efforts au programmeur souhaitant porter un logiciel d'une plateforme vers une autre, ou même entre différents systèmes d'exploitation sur une même plateforme. Comme les programmes Java doivent s'exécuter de la même manière sur toutes les machines, les plages de valeur des différents types sont fixes.

II. Les décimaux

Les types décimaux expriment les nombres réels disposant d'une partie décimale. Il existe deux types décimaux, présentés dans le Tableau.

Type	**Occupation en mémoire**	**Intervalles**	
float	4 Octets	1.4E-45	3.4028235E38
double	8 Octets	4.9E-324	1.7976931348623157E308

Tableau 2 : Les types décimaux en java

Le terme *double* indique que les nombres de ce type ont une précision deux fois supérieure à ceux du type *float* (on les appelle parfois nombres à double précision). On choisira de préférence le type *double* dans la plupart des applications. La précision limitée de *float* se révèle insuffisante dans de nombreuses situations. On ne l'emploiera que dans les rares cas où la vitesse de calcul (plus élevée pour les nombres à précision simple) est importante pour l'exécution, ou lorsqu'une grande quantité de nombres doit être stockée (afin d'économiser la mémoire).

Les nombres de type *float* ont pour suffixe F, par exemple 3.402F. Les nombres à décimales exprimés sans ce suffixe F, par exemple 3.402, sont toujours considérés comme étant du type *double*. Pour ces derniers, il est également possible de spécifier un suffixe D, par exemple 3.402D.

Tous les types décimaux sont signes et peuvent donc contenir des valeurs positives ou négatives.

IV. Les caractères (char)

Le type *char* est utilisé pour stocker un caractère unique. Une variable de type *char* utilise deux octets (16 bits) pour stocker le code Unicode du caractère. Il a une valeur minimale de '\u0000' (ou 0) et une valeur maximale de '\uffff' (ou 65 535 inclus).

Séquence d'échappement	**Nom**	**Valeur Unicode**
\b	Retour arrière	\u0008
\t	Tabulation	\u0009
\n	Séparation de ligne	\u000a
\r	Retour chariot	\u000d
\"	Guillemet	\u0022
\'	Apostrophe	\u0027
\\	Barre oblique inverse	\u005c

Tableau 3 : Les types char en java

V. Les Booléen

Le type de données booléen n'a que deux valeurs possibles: ***true*** et ***false***. Utilisez ce type de données pour les indicateurs simples qui suivent les conditions vraies / fausses. Ce type de données représente un peu d'information, mais sa «taille» en mémoire n'est pas est précisément définie.

VI. Les valeurs par défaut

Il n'est pas toujours nécessaire d'attribuer une valeur lorsqu'un champ est déclaré. Les champs qui sont déclarés mais non-initialisés seront initialisées par le compilateur avec des valeurs par défaut.

Le tableau suivant résume les valeurs par défaut des types de données primitifs en Java.

Type de données	**Valeur par défaut**
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'

String (ou tout objet)	null
boolean	false

Tableau 4 : Les valeurs par défaut des types en java

Les variables locales sont légèrement différentes; Le compilateur n'attribue jamais une valeur par défaut à une variable locale non initialisée. Si vous ne pouvez pas initialiser votre variable locale à l'endroit où elle est déclarée, assurez-vous de lui attribuer une valeur avant de tenter de l'utiliser. L'accès à une variable locale non initialisée entraînera une erreur de compilation.

VII. Conversions de type

Les conversions de types consistent à transformer une variable d'un type à un autre type.

Le langage java inclus une conversion (casting) de type automatiquement lors des calculs. Ainsi, les opérandes dans une opération sont (*implicitement*) convertis au type de l'opérande ayant la plus haute précision.

Le tableau suivant résume les règles de conversions lors de l'évaluation d'expressions.

Type opérande 1	**Type opérande 2**	**Conversion**	**Type résultat**
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

Tableau 5 : La conversion de types en java

Dans le cas où l'on veut faire une conversion manuellement au moment de l'opération, on utilisera la conversion *explicite*.

La syntaxe pour la conversion explicite en Java est :

```
(Type) (expression)
```

où expression peut être une variable, une constante, un nombre ou toute autre expression complexe.

Exemple

```
public class ConversionTypeExemple {
        public static void main(String [] args) {
                //declaration de variables
                double varDouble1 ;
                double varDouble2 = 4.3 ;
                int varInt = 5 ;
                //calcul
                varDouble1 = varDouble2 + (double) varInt ;
                System.out.println("Le resultat est : " + varDouble1)
        }
}
Le resultat est : 9.5
```

Les Variables et les constantes

Une variable en java est une zone mémoire qui permet de stocker des données qui sont utilisées lors de l'exécution d'un programme. Une variable doit être déclarée avant son utilisation et est caractérisée par :

Son identificateur : Il permet de nommer une variable, la variable sera reconnue par ce nom.

Son type : définit le type (genre) de données que la variable devra contenir

Sa valeur: contenu de la zone mémoire représentant la variable.

Le langage de programmation Java définit les types de variables suivants:

- **Variable d'instance :** Elle est déclarée a l'intérieure d'une classe et n'existe que si une instance de la classe est créée. La valeur d'une variable d'instance est propre à chaque instance d'une classe (à chaque objet, en d'autres termes);
- **Variable de classe :** Elle correspond à tout champ déclaré avec le modificateur ***static***; Cela indique au compilateur qu'il existe exactement une seule copie de cette variable pour chaque instance de la classe. Elle est accessible directement avec le nom de la classe.
- **Variable locale :** Elle est déclarée à l'intérieure d'une fonction (méthode). Les variables locales n'existe que pendant l'exécution de la méthode et ne sont visibles que par les méthodes dans lesquelles elles sont déclarées; Ils ne sont pas accessibles du reste de la classe.
- **Paramètres de fonctions :** Ils peuvent être considères comme des variables locales. L'initiation de ces variables se fait lors de l'appel de la fonction.

VIII. Nom des variables

Les règles et conventions pour nommer vos variables peuvent être résumées comme suit:

- Les noms de variable sont sensibles à la casse. La convention, est de toujours commencer les noms de variable avec une lettre, et non "$" ou "_".
- Lorsque vous choisissez un nom pour vos variables, utilisez des mots complets plutôt que des abréviations cryptiques. Cela facilitera la lecture et la compréhension du code. Dans certain il permet une auto-documentation du code;
- Le nom choisit pour une variable ne doit pas être un mot clé ou un mot réservé.
- Si le nom d'une variable se compose d'un seul mot, toutes les lettres doivent être en minuscules. Si elle se compose de plus d'un mot, mettez en majuscule la première lettre de chaque mot.
- Si la variable est une constante, telle que ***static final int NUM_SUIVI = 6***, la convention est de mettre tous les lettres en majuscules en séparant les mots avec le caractère de soulignement. Par convention, le caractère de soulignement n'est utilisé que pour les constantes.

IX. Déclaration des variables

En java, la déclaration d'une variable permet de la créer. Elle se fait en spécifiant le type et le nom de la variable. La déclaration d'une variable étant une instruction doit se terminer par un point-virgule.

Syntaxe

```
//Cas d'une seule variable
Type nomVaribale;

//Cas de plusieurs varibales
Type nomVariable1, nomVaribale2;
```

Exemple

```
int nombreDemployes;
double salaireDeBase;

Classe nomObjet;
Interface nomObjet;
```

La déclaration peut apparaitre n' importe où dans le code. Il faut simplement que la déclaration précède l'utilisation de la variable. Cependant, il est recommandé de déclarer les variables aussi près que possible du point de leur première utilisation.

Il est possible également de déclarer plusieurs variables de même type sur une seule ligne, comme suit :

```
int I, j; // ce sont deux entiers en Java
```

Ce style de déclaration n'est pas recommandé. Si vous définissez chaque variable séparément, vos programmes seront plus faciles à lire.

X. Initialisation des variables

Après avoir déclaré une variable, vous devez explicitement l'initialiser à l'aide d'une instruction d'affectation. Vous ne devez pas utiliser une variable non-initialisée. Le compilateur Java signale une erreur quand vous tentez d'utiliser une variable non-initialisée.

L'affectation d'une variable déclarée se fait à l'aide du symbole d'égalité (=), précédé à gauche du nom de la variable et suivi à droite par une expression Java représentant la valeur appropriée :

```
int nombreDemployes;
nombreDemployes = 10;
double salaireDeBase;
salaireBase = 300000.00;
```

Java offre également la possibilité de définir des modificateurs d'accès (qui seront présentées dans les chapitres suivants) et une valeur initiale lors de la déclaration des variables.

```
private int nombreDemployes = 10;
protected double salaireDeBase = 30000.00;
```

XI. Portées des variables

La portée d'une variable définit le bloc de code à partir duquel la variable peut être utilisée. Elle est fonction de l'emplacement ou est déclarée la variable dans le code. Cette déclaration peut être faite dans un bloc de code d'une classe, le bloc de code d'une méthode, ou le bloc de code

à l'intérieure d'une méthode. Si le même bloc de code est exécuté plusieurs fois dans un programme (cas des boucles), la variable sera créée à chaque passage de l'instruction d'affectation dans la boucle. L'initialisation de la variable est obligatoire dans ce cas. Il est possible de déclarer une variable interne à une méthode, ou un paramètre d'une méthode avec le même nom qu'une variable déclarée au niveau de la classe. La variable déclarée au niveau de la classe est alors masquée par la variable de la méthode.

XII. Durée de vie des variables

La durée de vie d'une variable permet de spécifier pendant combien de temps durant l'exécution du programme le contenu de cette variable sera disponible.

Pour une variable déclarée dans une méthode, la durée de vie correspond à la durée d'exécution de la méthode. La variable est automatiquement éliminée de la mémoire à la fin de l'exécution du programme et elle est recréée lors du prochain appel de la méthode.

Une variable déclarée à l'intérieure d'une classe sera utilisable tant qu'une instance de la classe existe. Les variables déclarées avec le mot clé ***static*** sont accessibles pendant toute la durée de fonctionnement du programme.

XIII. Les Constantes

En java, on peut définir des valeurs numériques ou chaines de caractères qui ne seront pas modifiées lors de l'exécution d'un programme. Ces valeurs sont alors définit comme des constantes.

La déclaration d'une constante se fait en ajoutant le mot clé *final* avant devant le type de la variable lors de sa déclaration. Le mot clé *final* signifie que vous affectez une valeur à la variable une seule fois, et une fois pour toutes. Par convention, les noms des constantes sont entièrement en majuscules.

```
Cas d'une seule constante
final Type NOM_CONSTANTE = valeur;

Cas de plusieurs constantes
final Type NOM_CONST1 = val1,  NOM_CONST2 = val2;
```

Il est préférable de séparer la déclaration de plusieurs constantes sur différentes lignes afin d'améliorer la lisibilité du code. Ainsi la déclaration de plusieurs constantes peut s'écrire :

```
final Type NOM_CONST1 = val1,
final Type NOM_CONST2 = val2;
final Type NOM_CONST3 = val2;
```

Exemple

```
final double TAUX_TVA = 0.18 ;
```

La valeur d'une constante peut être calculée à partir d'une autre constante.

```
int tauxBase = 0.1;
final double TAUX_TVA = tauxBase*0.18 ;
```

Les Opérateurs

Les opérateurs sont des mots-clés du langage qui permettent d'effectuer des opérations spécifiques sur un, deux ou trois opérandes, puis renvoient un résultat.

XIV. Opérateurs arithmétiques

Le langage de programmation Java fournit des opérateurs qui effectuent l'addition, la soustraction, la multiplication et la division. En plus de ces opérateurs, java offre l'opérateur modulo "%", qui retourne le reste de la division entière.

Opérateur	**Description**
+	Opérateur d'addition (aussi utiliser pour la concaténation des chaines de caractères String)
-	Opérateur de Soustraction
*	Opérateur de Multiplication
/	Opérateur de Division
%	Opérateur Modulo (Reste de la division entière)

Tableau 6 : Les opérateurs arithmétiques en java

Opérateurs unaires

Les opérateurs unaires n'exigent qu'un opérande; Ils effectuent diverses opérations telles que l'incrémentation / décrémentation d'une valeur par un, la négation d'une expression, ou l'inversion de la valeur d'un booléen.

Opérateur	**Description**
+	Opérateur unaire plus; Indique une valeur positive (les nombres sont positifs sans cet opérateur)
-	Opérateur unaire moins; indique une valeur négative
++	Incrémentation
--	Décrémentation
!	Négation
~	Complément a un

Tableau 7 : Les opérateurs unaires en java

XV. Opérateurs d'affectation

L'opérateur d'affectation en Java est l'opérateur « = » qui permet d'affecter une valeur a une variable.

```
int nbreDemploye = 22 ;
double salaireBase = 120000 ;
```

Cet opérateur peut aussi être utilisé en combinaison avec un opérateur arithmétique, logique et binaire.

```
x += 4;
```

Équivaut à l'instruction

```
x = x + 4;
```

En règle générale, placez l'opérateur arithmétique à gauche du signe =, par exemple *= ou %=.

XVI. Opérateurs d'égalité, relationnels et conditionnels

- **Les opérateurs d'égalité et relationnels**

Les opérateurs d'égalité et relationnels déterminent si un opérande est supérieur, inférieur, égal ou non égal à un autre opérande. Ils sont utilisés dans les structures de contrôles.

Opérateur	Opération réalisée	Exemple	Résultat
==	Egale a	2==4	Faux
!=	Diffèrent de	1 !=3	Vrai
>	Supérieur a	2>5	Faux
>=	Supérieur ou égal a	3>=2	Vrai
<	Inferieur a	1<5	Vrai
<=	Inférieur ou égal a	5<=3	Faux

Tableau 8 : Les opérateurs d'égalité et relationnels

- **Les opérateurs conditionnels**

Les opérateurs logiques permettent de combiner les expressions dans des structures de conditionnelles ou des structures de boucles.

Opérateur	Opération réalisée	Exemple	Résultat
&	Et logique	if((test1)&(test2))	Vrai si test1 et test2 sont vrai
|	Ou logique	if((test1)|(test2))	Vrai si test1 ou test2 est vrai
^	Ou exclusif	if((test1)^(test2))	Vrai si test1 ou test2 est vrai mais pas les deux simultanément
!	Négation	if(!test1)	Inverse le résultat du test
&&	Et logique	if((test1)&&(test2))	Vrai si test1 et test2 sont vrai
||	Ou logique	if((test1)||(test2))	Vrai si test1 ou test2 est vrai

Tableau 9 : Les opérateurs conditionnels

Les opérateurs && et || sont évalués de manière optimisée (en court-circuit). Le deuxième argument n'est pas évalué si le premier détermine déjà la valeur. Si vous combinez deux expressions avec l'opérateur &&,

expression1 && expression2

La valeur de la deuxième expression n'est pas calculée si la première a pu être évaluée à false (puisque le résultat final serait false de toute façon).

Java gère l'opérateur ternaire ? qui se révèle utile à l'occasion. L'expression

```
condition ? expression1 : expression2
```

est évaluée à expression1 si la condition est ***true***, à expression2 sinon. Par exemple,

```
x < y ? x : y
```

donne le plus petit entre x et y.

- L'instance de l'opérateur de comparaison de types

L'opérateur ***instanceof*** compare un objet à un type spécifié. Il est utilisé pour tester si un objet est une instance d'une classe, une instance d'une sous-classe ou une instance d'une classe qui implémente une interface particulière.

Opérateur	Opération réalisée	Exemple	Résultat

Instanceof	Comparaison du type de la variable avec le type indiqué	Objet Instanceof Personne	True si la variable objet référence un objet crée a partir de la classe Personne ou d'une de ses sous-classe

Tableau 10 : L'opérateur de comparaison de types ***instanceof***

XVII. Ordre d'évaluation des Opérateurs

Lorsque plusieurs opérateurs sont combinés dans une expression, ils sont évalués dans un ordre bien précis.

Le tableau suivant présente la liste des opérateurs en fonction de l'ordre de priorité. Plus un opérateur est proche du haut de la table, plus sa priorité est élevée. Les opérateurs ayant une priorité plus élevée sont évalués avant les opérateurs ayant une priorité relativement plus faible. Les opérateurs sur la même ligne ont la même priorité. Si des opérateurs avec le même ordre d'évaluation sont dans une expression, les opérateurs binaires, à l'exception des opérateurs d'affectation, sont évalués de gauche à droite; Les opérateurs d'affectation sont évalués de droite à gauche.

Opérateurs	**Ordre d'évaluation**
postfix	expr++ expr--
unaire	++expr --expr +expr -expr ~ !
multiplicatif	* / %
additif	+ -
relationnel	< > <= >= instanceof
égalité	== !=
bit a bit ET	&
bit a bit exclusif OU	^
bit a bit inclusif OU	|
et logique	&&
ou logique	||
ternaire	? :
affectation	= += -= *= /= %= &= ^= |=

Tableau 11 : ordre d'évaluation des opérateurs

Les Tableaux

Les tableaux (arrays) en anglais sont utilisés en java pour organiser une liste de données de même type par le même nom et d'utiliser un indice pour les différencier. Le premier élément d'un tableau se trouve à l'indice zéro (0). Le nombre d'éléments d'un tableau est spécifié à la création. Les tableaux peuvent contenir des donnés de types primitifs, ainsi que des objets. Par exemple, les tableaux peuvent être utilisés dans un programme pour stocker les noms et prénoms des étudiants d'une école, les noms et prénoms des employés d'une Entreprise, etc.

La figure suivante est une représentation d'un tableau avec les indices et les valeurs correspondants à chaque indice.

Indice	0	1	2	3	4	5	6	7	8	9
Valeur	12	15	16	11	11	17	19	13	10	2

Figure 1. Représentation d'un tableau

XVIII. Déclarer et utiliser les tableaux

Création d'un tableau

La déclaration d'un tableau se fait comme la déclaration d'une variable sauf qu'il faut ajouter les caractères [] après le type de données. L'exemple suivant permet de déclarer ***notesEtudiants*** comme un tableau d'éléments de type entier.

```
int [] notesEtudiants;
```

Apres déclaration, le tableau doit être instancié avec l'opérateur ***new***. Celle-ci alloue l'espace mémoire nécessaire pour stocker les éléments du tableau.

```
notesEtudiants = new int[20];
```

La déclaration et l'instanciation peuvent se faire avec une seule instruction. L'exemple suivant est une déclaration et l'instanciation d'un tableau de vingt entiers pour stocker les notes des étudiants.

Syntaxe :

```
int [] notesEtudiants= new int[20] ;
```

Lors de la création du tableau, il est possible aussi d'initialiser le contenu du tableau. Dans ce cas, il n'est pas nécessaire de spécifier la taille du tableau. La syntaxe est la suivante :

```
int [] notesEtudiants = {15, 12, 11, 11, 8, 17, 16, 18, 10, 12};
```

Utilisation d'un tableau

Les éléments d'un tableau sont accessibles comme les variables à l'exception qu'il faut ajouter l'indice de l'élément qu'on veut accéder. Par exemple, vous pouvez afficher les éléments avec leurs index comme suit:

```
System.out.println("élement 1 à l'index 0: " + notesEtudiants[0]);
System.out.println("Élement 2 à l'index 1: " + notesEtudiants[1]);
System.out.println("Élement 3 à l'index 2: " + notesEtudiants[2]);
```

Les tableaux ont une limitation importante : la taille d'un tableau est fixe. Si vous commencez avec un ensemble de 10 éléments et vous décidez plus tard d'ajouter des éléments supplémentaires, alors vous devez créer un nouveau tableau et copier tous les éléments du tableau existant dans nouveau tableau.

XIX. Parcours de tableau

Pour parcourir complètement ou partiellement un tableau, il suffit simplement d'utiliser une boucle for. Voici un exemple qui parcoure complètement un tableau pour l'afficher à la console :

```
double[] tab = {1.4, 9.2, 10.4};

for (int i = 0; i < tab.length; i++)
{
   System.out.println (tab[i]);
}
```

Java 5.0 offre une nouvelle boucle for qui permet de facilement parcourir tous les objets qui représentent des collections d'objets. Le listing suivant donne la syntaxe de cette boucle for.

Syntaxe :

```
for (type_de_donnée nom_variable : collection_à_parcourir)
{
   // Corps de la boucle
}
```

On précise donc un nom pour une variable locale qui représentera l'élément en cours et on précise la collection qu'il faut parcourir. Voici l'exemple ci-dessous avec la boucle for améliorée.

```
double[] tab = {1.4, 9.2, 10.4};
 for (double nb : tab)
{
   System.out.println (nb);
}
```

Il ne faut même plus se soucier de la taille du tableau, cette boucle for s'occupe de parcourir tous les éléments du tableau, les plaçant tour à tour dans la variable locale spécifiée.

XX. Passer des tableaux en paramètre

Un tableau peut être passé comme paramètre à une méthode. Comme les tableaux sont des objets en java, une copie de la référence du tableau original est passée en paramètre.

Exemple

```
public class TestTableaux {
	public static void main(String[] args) {
		// Déclaration et initialisation du tableau avec les valeurs
		int[] notesEtudiants = { 15, 12, 11, 11, 8, 17, 16, 18, 10, 12 };

		// Appel de la méthode afficherTableau avec passage du tableau
		// par rérérence
		afficherTableau(notesEtudiants);
	}

	public static void afficherTableau(int tableau[]) {

		for (int i = 0; i < tableau.length; i++) {
			System.out.println("élement à l'index " + i + " : " + tableau[i]);
		}
	}
}
```

Une méthode qui reçoit un tableau en paramètre peut modifier un élément du tableau car elle fait référence à la valeur de l'élément original du tableau. Par contre elle ne peut modifier la référence originale du tableau car la méthode n'a reçue qu'une copie de la référence du tableau. Dans l'exemple suivant, on modifie la valeur à l'indice 0 directement au niveau de la méthode ***modifierElementTableau()***;

Exemple :

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

	//Déclaration et initialisation du tableau avec les valeurs
	int[] notesEtudiants = { 15, 12, 11, 11, 8, 17, 16, 18, 10, 12 };

	//Appel de la méthode modifierElementTableau
	modifierElementTableau(notesEtudiants);

	//Appel de la méthode afficherTableau avec passage du tableau
	//par rérérence
	afficherTableau(notesEtudiants);
}

public static void modifierElementTableau(int tableau[]) {
	tableau[0] += 5;
```

```
}

public static void afficherTableau(int tableau[]) {

        for (int i = 0; i < tableau.length; i++) {
                System.out.println("élement à l'index " + i + " : " + tableau[i]);
        }

}
```

Ces règles sont compatibles avec les règles qui régissent tout type d'objet. Et dans ce cas on parle de passage par référence.

Si on veut faire passer un élément du tableau en paramètre on peut le faire comme avec les variables. Ainsi, si le type d'élément est un type primitif, une copie de la valeur est passée. Par contre, si cet élément est une référence à un objet, une copie de la référence d'objet est passée. Dans l'exemple suivant, on fait passer l'élément sur l'indice 3 comme paramètre à la méthode ***modifierElementTableau()***;

Exemple ;

```
public static void main(String[] args) {
        // Déclaration et initialisation du tableau avec les valeurs
        int[] notesEtudiants = { 15, 12, 11, 11, 8, 17, 16, 18, 10, 12 };

        // Appel de la méthode modifierElementTableau
        modifierElementTableau(notesEtudiants[3]);

        // Appel de la méthode afficherTableau avec passage du tableau
        // par rérérence
        afficherTableau(notesEtudiants);
}

public static void modifierElementTableau(int var) {
        var = 13;
}

public static void afficherTableau(int tableau[]) {
        for (int i = 0; i < tableau.length; i++) {
                System.out.println("élement à l'index " + i + " : " + tableau[i]);
        }

}
```

XXI. Tableau à plusieurs dimensions

Les tableaux à plusieurs dimensions sont des tableaux contenant d'autres tableaux. La syntaxe de déclaration est semblable à celle d'un tableau simple à l'exception qu'il faut spécifier autant de paires de crochets que de dimensions.

Syntaxe :

```
int [][] matrice ;
```

De même, l'instanciation et l'initialisation d'un tableau de plusieurs dimensions sont semblables à celles d'un tableau simple hormis qu'il faut spécifier la taille de chaque dimension.

Syntaxe :

```
matrice = new int [2][3] ;
int grille = {{11, 12, 13}, {21, 22, 23}. {31, 32, 33}} ;
```

Exemple :

```
class TableauMultiDimDemo {
   public static void main(String[] args) {
      String[][] names = {
         {"Mr. ", "Mme. ", "M. "},
         {"Fall", "Diop"}
      };
      // Mr. Fall
      System.out.println(names[0][0] + names[1][0]);
      // Ms. Diop
      System.out.println(names[0][2] + names[1][1]);
   }
}
```

Les Chaînes de Caractères (String)

Les chaines de caractères sont représentées en Java par le type ***String***. Le type String permet ainsi de manipuler les chaines de caractères par une application Java.

XXII. Création d'une chaine de caractères

La création d'une variable de type chaine de caractères est semblable à celle d'une variable de type primitif. Il suffit de précédé le type ***String*** du nom de la variable et de lui affecter une valeur initiale spécifiée entre les caractères " et ". Il est aussi possible d'utiliser l'opérateur ***new*** pour l'affectation d'une valeur car le type ***String*** est un type objet et non un type primitif. La valeur par défaut d'une variable de type ***String*** est ***null***.

Syntaxe :

```
String chaine1 = "Ceci est une chaine";
String chaine2 = new String("Ceci est une chaine");
```

Apres sa création, une chaine de caractère ne peut plus être modifiée. L'affectation d'une nouvelle valeur à la variable provoque la création d'une nouvelle instance de la classe ***String***. La classe String contient de nombreuses méthodes permettant de la modification de chaines de caractères. Ces méthodes retournent une nouvelle instance contenant le résultat de la chaîne modifiée au lieu de modifier le contenu de la chaîne initiale.

Lors de L'affectation, si la valeur content le caractère « " », il faut le protéger par une séquence d'échappement comme dans l'exemple ci-dessous.

```
String chaine = "Il a dit : \" Ça suffit ! \"";
System.out.println(chaine);
```

XXIII. Extraction d'un caractère

La fonction ***charAt*** permet d'obtenir le caractère située à une position donnée d'une chaîne de caractères. Cette méthode reçoit comme argument l'index du caractère à extraire. Le premier caractère se trouve à l'index zéro comme pour un tableau. La fonction retourne un caractère.

```
String chaine = "L'été est pluvieux à Dakar";
System.out.println("Le troisième caractère de la chaîne est : " + chaine.charAt(2));
```

XXIV. Découpage de chaîne

La fonction ***substring*** de la classe String retourne une portion de chaîne en fonction de la position de départ et de la position de fin qui lui sont passées en paramètres. La chaîne obtenue commence par le caractère situé à la position de départ et se termine au caractère précèdent la position de fin.

```
String chaine = "L'été est pluvieux à dakar";
System.out.println("Ceci est une partie de la chaîne : " + chaine.substring(2, 9));
Ceci est une partie de la chaîne : été est
```

XXV. Comparaison de chaînes

Le type chaîne de caractères étant un type objet, il existe des méthodes de la classe String permettant de comparer des chaînes de caractères. La méthode ***equals*** effectue une comparaison de la chaîne avec celle qui est passée en paramètre. Elle retourne un ***boolean*** (***true*** si les chaînes sont identiques ***false*** sinon). Cette fonction fait une distinction entre majuscules et minuscules lors de la comparaison. La méthode ***equalsIgnoreCase*** ne tient pas compte de cette distinction.

```
String chaine1 = "Programmation en Java";
String chaine2 = "Programmation en C";

if(chaine1.equals(chaine2)) {
        System.out.println("Les deux chaînes sont identiques");
} else {
        System.out.println("Les deux chaînes sont différentes");
}
```

Pour réaliser une comparaison avec classement, il faut utiliser la méthode ***compareTo*** ou ***compareToIgnoreCase*** de la classe String. Ces deux méthodes prennent comme paramètres la chaîne de caractères à comparer. Le résultat de la comparaison est retourné sous forme d'un entier inférieur à zéro si la chaîne est inférieure à celle reçue en paramètre, égal à zéro si les deux chaine sont identiques, et supérieure à zéro si la chaîne est supérieure à celle reçue en paramètre.

```
String chaine1 = "Programmation en Java";
String chaine2 = "Programmation en C";

if(chaine1.compareTo(chaine2) > 0) {
        System.out.println("Chaîne1 est supérieure à Chaîne2");
}else if(chaine1.compareTo(chaine2) < 0) {
        System.out.println("Chaîne1 est inférieure à Chaîne2");
} else {
        System.out.println("Les deux chaînes sont différentes");
}
```

XXVI. Remplacement dans une chaîne

Java offre aussi la possibilité de remplacer une sous-chaîne dans une chaîne. La méthode replace permet de spécifier une chaîne de substitution à la chaine recherchée. Elle prend deux paramètres :

- la chaîne recherchée
- la chaîne de remplacement

```
String chaine = "On veut remplacer test dans la chaîne";
chaine.replace("test","valeur");
System.out.println(chaine);
```

XXVII. Formatage d'une chaîne

La méthode format de la classe String permet d'éviter des opérations de conversations et de concaténations. Le premier paramètre attendu par cette fonction est une chaîne de caractères spécifiant sous quelle forme on souhaite afficher le résultat. Cette chaîne peut contenir un ou plusieurs motifs de formatage représentés par le caractère % suivi d'une lettre indiquant sous quelle forme sera présentée l'information. La chaîne retourné est construite en remplaçant chaque motif par le paramètre correspondant Le tableau suivant présente les principaux motifs disponibles.

Motif	Description
%b	Insertion d'un booléen
%s	Insertion d'une chaîne de caractères
%d	Insertion d'un nombre entier
%o	Insertion d'un nombre entier en octal
%x	Insertion d'un nombre entier en hexadécimal
%f	Insertion d'un nombre décimal
%e	Insertion d'un nombre décimal au format scientifique
%n	Insertion d'un saut de ligne

```
System.out.println(String.format("boolean : %b %n"
                                 + "chaîne de caractères : %s %n"
                                 + "entier : %d %n"
                                 + "entier en hexadécimal : %x %n"
                                 + "entieren octal : %o %n"
                                 + "décimal : %f %n"
                                 + "décimal au format scientifique : %e %n",
                                 b, s, i, i, i, d, d));
```

Plus d'information sur le type String :

https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

https://docs.oracle.com/javase/8/docs/api/java/lang/StringBuilder.html

Les Structures de contrôles

Les instructions à l'intérieur du code d'un programme Java sont généralement exécutées de haut en bas, dans l'ordre dans lequel elles apparaissent. Cependant, les instructions de contrôle de flux permettent de modifier l'ordre d'exécution en employant la prise de décision, le bouclage et le branchement, permettant au programme d'exécuter conditionnellement des blocs de code particuliers. Cette section décrit les instructions de décision (if-then, if-then-else), les instructions de bouclage (for, while, do-while) et les instructions de branchement (break, continue, return) offertes en java.

XXVIII. Les instructions de décision : if-then, if-then-else

L'instruction de décision ***if-then*** est la plus simple des instructions de décision. Elle permet d'exécuter une instruction si une condition est respectée. La condition doit être une expression, qui une fois évaluée, doit fournir un boolean (true ou false). Avec cette syntaxe, toutes les instructions situées entre les braquets du *if* seront exécutées.

Syntaxe

```
if (condition) {
      // Instruction(s);
 }
```

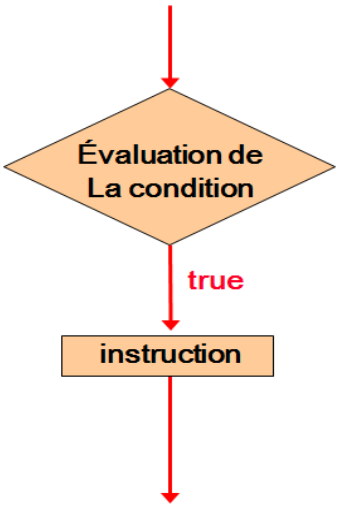


Figure 1 : Logique de l'instruction –if-then

Exemple

```
int varX = -1 ;
ff(varX > 0) {
        System.Out.println("a est positif") ;
}
```

Dans le cas de *if-then-else*, Il s'agit d'exécuter une ou plusieurs instructions si une condition est vraie et d'exécuter une ou plusieurs autres instructions si la condition est fausse. Une seule des instructions est exécutée mais pas les deux.

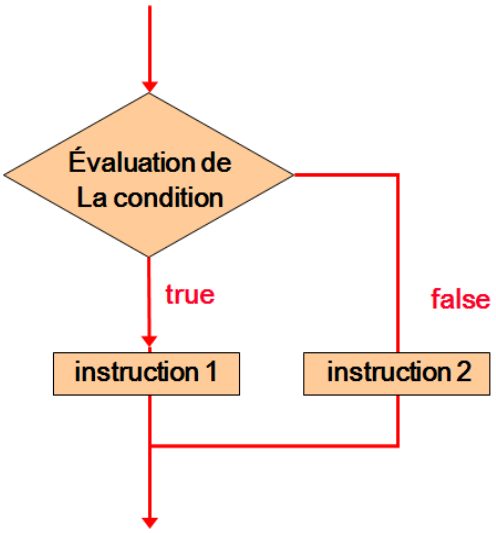


Figure 2 : Logique de l'instruction –if-then-else

Syntaxe

```
if (condition) {
      //instruction(s);
} else {
      //instruction(s);
}
```

Exemple

```
int varX = -1 ;
int varY = 1 ;

if(varX > varY) {
     System.Out.println("varX est superieur à b") ;
else{
     System.Out.println("varX est inferieur à b") ;
}
```

XXIX. Les instructions de décision : switch

La structure ***switch*** permet un fonctionnement équivalent à la structure if mais offre une meilleure lisibilité du code. Contrairement aux instructions if-then et if-then-else, l'instruction switch peut avoir un nombre de chemins d'exécution possibles. La structure switch fonctionne avec les types de données primitives (byte, short, char et int). Il fonctionne également avec des types énumérés, la classe String et quelques classes spéciales qui enveloppent certains types primitifs: Character, Byte, Short et Integer.

Syntaxe

```
switch (expression)
{
case valeur1:
instruction(s); break;
case valeur2:
instruction(s); break;
...
default:
instruction(s) ;
}
```

Le corps d'une instruction ***switch*** est connu sous le nom de **bloc switch**. Une instruction dans le bloc switch peut être étiquetée avec un ou plusieurs cas ou par un cas par défaut. L'instruction ***switch*** évalue son expression, puis exécute toutes les instructions qui suivent le cas correspondant.

Exemple

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

      int month = 8;
      String monthString;
      switch (month) {
         case 1:  monthString = "Janvier";
                  break;
         case 2:  monthString = "Février";
                  break;
         case 3:  monthString = "Mars";
                  break;
         case 4:  monthString = "Avril";
                  break;
         case 5:  monthString = "Mai";
                  break;
         case 6:  monthString = "Juin";
                  break;
         case 7:  monthString = "Juillet";
                  break;
         case 8:  monthString = "Août";
                  break;
         case 9:  monthString = "Septembre";
                  break;
         case 10: monthString = "Octobre";
                  break;
         case 11: monthString = "Novembre";
                  break;
         case 12: monthString = "Décembre";
                  break;
         default: monthString = "Mois non valide";
                  break;
      }
      System.out.println(monthString);
   }
}

// Août
```

XXX. L'opérateur conditionnel

Java dispose d'un opérateur conditionnel qui utilise une condition booléenne pour choisir laquelle de deux expressions est évaluée

Syntaxe

```
condition ? expression1 : expression2
```

Si la condition est true, expression1 est evaluée; Si false, expression2 est evaluée La valeur retournée est la valeur de l'expression sélectionnée. L'opérateur conditionnel est similaire à la structure if-else sauf qu'il retourne une valeur

Exemple

```
maVar = ((num1 > num2) ? num1 : num2);
```

Si num1 est supérieur à num2, alors num1 est affecté à maVar; sinon, num2 est affecté à maVar

Remarque: *operateur conditionnel est ternaire car il exige 3 opérandes.*

XXXII. Les instructions de bouclage : while, do-while

While

L'instruction *while* exécute continuellement un bloc d'instructions alors qu'une condition particulière est vraie. Sa syntaxe peut être exprimée comme suit:

Syntaxe

```
while (condition)
{
    instruction(s)
}
```

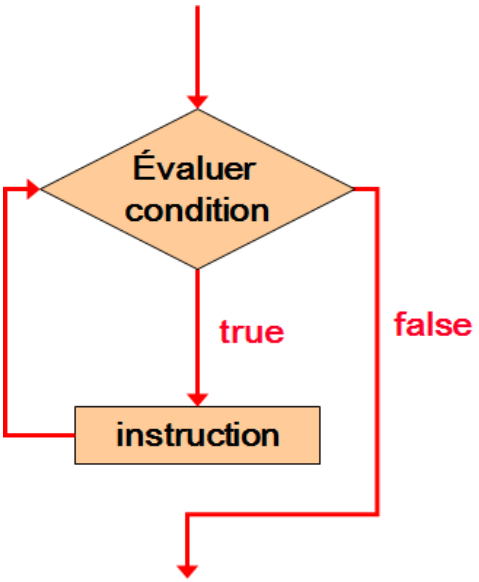


Figure 3 : Logique de l'instruction de bouclage –while-

L'instruction while évalue l'expression, qui doit retourner une valeur booléenne. Si l'expression est évaluée à true, l'instruction while exécute l'instruction (s) dans le bloc while. L'instruction while continue à tester l'expression et à exécuter son bloc jusqu'à ce que l'expression s'évalue à false.

Exemple

```
class WhileTest {
   public static void main(String[] args){
      int count = 1;
      while (count < 11) {
         System.out.println("Le nombre est : " + count);
         count++;
      }
   }
}
```

Do-While

La différence entre do-while et while est que do-while évalue son expression au bas de la boucle au lieu du top. Par conséquent, les instructions dans le bloc do sont toujours exécutées au moins une fois.

Syntaxe

```
do
{
    instruction(s)
} While (condition);
```

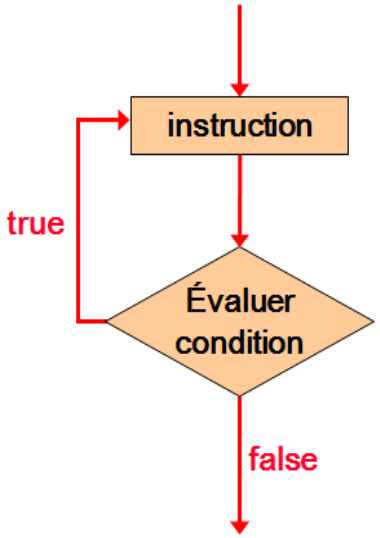


Figure 4 : Logique de l'instruction de bouclage –do-while-

Exemple

```
class DoWhileTest {
   public static void main(String[] args){
      int count = 1;
      do {
         System.out.println("Count is: " + count);
         count++;
      } while (count < 11);
   }
}
```

XXXIV. Les instructions de bouclage : for

L'instruction for fournit un moyen compact d'itérer sur une plage de valeurs. Les programmeurs se réfèrent souvent à elle comme la «boucle for» en raison de la façon dont il boucle plusieurs fois jusqu'à ce qu'une condition particulière est satisfaite.

La boucle for:

- L'expression d'initialisation initialise la boucle; Il est exécuté une fois, lorsque la boucle commence.
- Lorsque l'expression de terminaison est évaluée à false, la boucle se termine.
- L'expression d'incrémentation est appelée après chaque itération à travers la boucle; Il est parfaitement acceptable pour cette expression d'incrémenter ou de décrémenter une valeur.

Syntaxe

```
for (initialisation; condition; incrémentation) {
        instruction(s) ;
}

for (type item : Objet) {
        instruction(s) ;
}
```

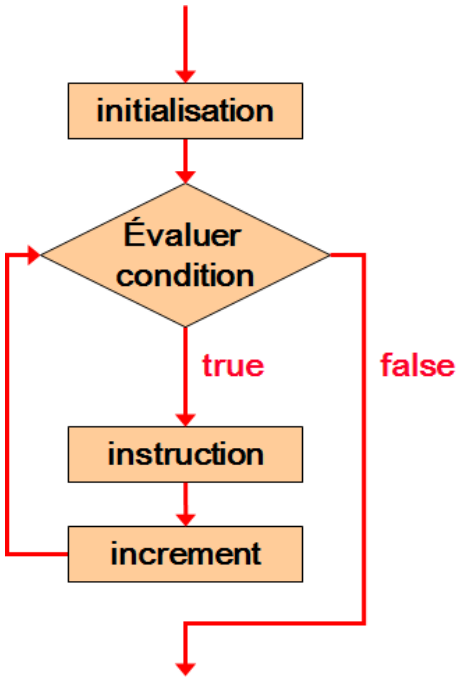


Figure 5 : Logique de l'instruction de bouclage –for–

Exemple

```
class ForTest {
   public static void main(String[] args){
       for(int i=1; i<11; i++){
           System.out.println("Les nombre est : " + i);
       }
   }
}

//Affichage
La sortie de ce programme est:
```

```
Les nombre est : 1
Les nombre est : 2
Les nombre est : 3
Les nombre est : 4
Les nombre est : 5
Les nombre est : 6
Les nombre est : 7
Les nombre est : 8
Les nombre est : 9
Les nombre est: 10

Class ForTest2 {
   public static void main(String[] args){
        int[] numbers =  {1,2,3,4,5,6,7,8,9,10};
        for (int item : numbers) {
            System.out.println("Count is: " + item);
        }
   }
}
```

XXXV. Les instructions de branchement

Trois instructions permettent de modifier le fonctionnement normal des structures de boucles : Break, Continue, Return.

Break

Si cette instruction est placée à l'intérieur du bloc de code d'une structure de boucle elle provoque la sortie immédiate de ce bloc de code. L'exécution se poursuit par l'instruction placée après le bloc de la structure de boucle. Cette instruction doit en général être exécutée de manière conditionnelle, sinon les instructions situées après à l'intérieur de la boucle ne seront jamais exécutées.

Dans le cas de boucles imbriquées, il est possible d'utiliser l'instruction Break associée avec une étiquette.

Exemple

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

      int[] tableauDentiers = { 32, 87, 3, 589, 12, 1076, 2000, 8, 622, 127 };
      int valeurRecherche = 12;

      int i;
      boolean trouve = false;

      for (i = 0; i < tableauDentiers.length; i++) {
         if (tableauDentiers [i] == valeurRecherche) {
            trouve = true;
            break;
         }
      }

      if (trouve) {
         System.out.println(valeurRecherche + "Trouvé  à l'index " + i);
      } else {
         System.out.println(valeurRecherche + " n'es pas dans le tableau ");
      }
   }
}
```

Continue

Cette instruction permet d'interrompre l'exécution de l'itération courante d'une boucle et de continuer l'exécution à l'itération suivante après vérification de la condition de sortie de boucle. Comme pour l'instruction Break, elle doit être exécutée de manière conditionnelle et accepte également l'utilisation d'une étiquette.

Exemple

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

      String phraseText = "Pascal porte une parapluie parce qu'il pleut.";
      int longueur = phraseText.length();
      int nombreP = 0;

      for (int i = 0; i < longueur; i++) {
         // chercher si la phrase contient un p
         if (phraseText.charAt(i) != 'p')
            continue; //continuer l'exécution à l'itération suivante (i+1)
```

```
            //incrémenter le nombre de p
            nombreP++;
        }
        System.out.println("Nombre de p trouvés dans la phrase : " + nombreP + ".");
    }
}
```

Return

La dernière des instructions de branchement est l'instruction return. L'instruction return sort de la méthode courante et le flux de contrôle retourne à l'endroit où la méthode a été invoquée. L'instruction return a deux formes: une qui renvoie une valeur, et une qui ne renvoie rien. Pour retourner une valeur, placez simplement la valeur (ou une expression qui calcule la valeur) après le mot-clé return.

```
return ++count;
```

Le type de données de la valeur renvoyée doit correspondre au type de la valeur de retour déclarée de la méthode. Lorsqu'une méthode est déclarée nulle, utilisez la forme de retour qui ne renvoie pas de valeur.

```
return;
```

Exemple

```
// Une méthode pour calculer la surface d'un rectangle
public int getSurface() {
        return longueur * largeur;
}
```

Résumé

Les huit types de données primitifs sont: *byte*, *short*, *int*, *long*, *float*, *double*, *booleen* et *char*. La classe *java.lang.String* représente les chaînes de caractères. Le compilateur attribuera une valeur par défaut raisonnable pour les champs des types ci-dessus; Pour les variables locales, une valeur par défaut n'est jamais affectée. Un littéral est la représentation du code source d'une valeur fixe.

Le langage de programmation Java utilise à la fois des "champs" et des "variables" dans sa terminologie. Les variables d'instance (champs non statiques) sont uniques à chaque instance d'une classe. Les variables de classe (champs statiques) sont des champs déclarés avec le modificateur statique; Il y a exactement une copie d'une variable de classe, peu importe combien de fois la classe a été instanciée. Les variables locales stockent l'état temporaire dans une méthode. Les paramètres sont des variables qui fournissent des informations supplémentaires à une méthode; Les variables locales et les paramètres sont toujours classés comme des «variables» (et non des «champs»). Lorsque vous nommez vos champs ou variables, il existe des règles et des conventions que vous devez (ou devez) suivre.

L'instruction ***if-then*** est la plus élémentaire de toutes les instructions de contrôle de flux. Il indique à votre programme d'exécuter une certaine section de code uniquement si un test particulier est évalué à vrai. L'instruction ***if-then-else*** fournit un chemin d'exécution secondaire lorsqu'une clause ***"if"*** est évaluée à false. Contrairement à ***if-then*** et ***if-then-else***, l'instruction ***switch*** permet un nombre quelconque de chemins d'exécution possibles. Les instructions ***while*** et ***do-while*** exécutent continuellement un bloc d'instructions tant qu'une condition particulière est vraie. La différence entre ***do-while*** et ***while*** est que ***do-while*** évalue son expression en bas de la boucle au lieu du top. Par conséquent, les instructions dans le bloc ***do*** sont toujours exécutées au moins une fois. L'instruction ***for*** fournit une méthode compacte pour itérer sur une plage de valeurs. Il a deux formes, dont l'une a été conçue pour boucler à travers des collections et des tableaux.