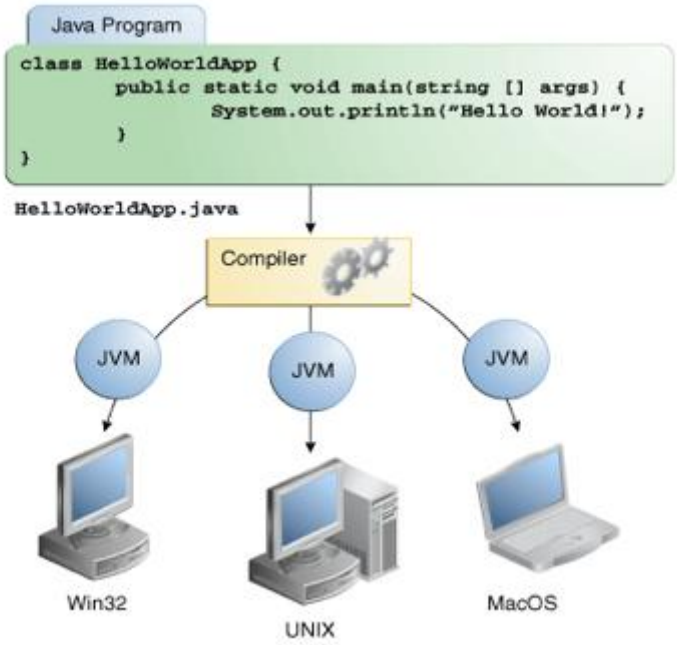
Cours de Java : Les bases Chapitre 1

Ce chapitre présente les connaissances indispensables à l'acquisition des techniques de la programmation en Java : les éléments fondamentaux de la syntaxe du langage.

Java est un langage de programmation et un environnement d'exécution. Les applications peuvent s'exécuter sur toutes les plates formes (Windows, Linux, Unix, Mac) qui possèdent un environnement d'exécution **JRE** (Java Runtime Environment) adapté. Le site d'Oracle propose les JRE pour les différents systèmes d'exploitation actuels.

Le **JRE** inclut une **JVM**, machine virtuelle Java (Java Virtual Machine), ainsi que les librairies et les composants nécessaires à l'exécution des applications java.

Une application java peut être développée sur une machine Windows, et par exemple s'exécuter sur une machine Linux: on dit qu'une application Java est **portable**.



Le langage Java a été développé par la société Sun. La société Sun n'existe plus car elle a été rachetée par Oracle.
Les exemples suivants seront développés et exécutés sur une machine Windows.
Il existe des produits de développements performants comme **Eclipse** ou **JBuilder** ou **NetBean** que nous n'utiliserons pas dans ce chapitre. Ces produits ne sont pas indispensables, et pour l'apprentissage du langage Java, mieux vaut ne pas les utiliser tout de suite.

1. Outils de base nécessaires au développement d'applications Java.

- Installer un **JDK**, Java Development Kit, par exemple le jdk 1.8 qui s'installe par défaut dans le dossier **C:\Program Files\Java\jdk1.8.0_05.**
- Dans le **panneau de configuration** de Windows, sélectionner **Système et Sécurité/Système/Paramètres Système avancés/Variables système** **ajouter** à la **variable système PATH**, le chemin suivant:

Ici pour le PATH

C:\Program Files\Java\jdk1.8.0_05\bin.

C'est dans ce répertoire que l'on trouve les commandes nécessaires à la compilation et à l'exécution des applications Java, et notamment:

javac qui est la commande de compilation.

java qui est la commande d'exécution.

- Utiliser un éditeur de texte ASCII comme **Notepad++** pour écrire les programmes sources.

Pour développer une application Java, on passe par les étapes suivantes:

- Ecriture du fichier source, de nom d'extension **.java**, avec un éditeur de texte, enregistrement du fichier.
- Compilation avec le compilateur **javac** dans une fenêtre console (invite de commandes). On utilise pour cela la commande **javac**.
- Exécution avec l'interpréteur **java** dans une fenêtre console. On utilise pour cela la commande **java**.

Q0 Tester l'installation

Ouvrir une fenêtre console (invite de commandes ou Démarrer/Exécuter/cmd puis OK). Taper **javac**, une liste de messages propres à Java doit s'afficher.
Si un message du genre « javac n'est pas reconnu comme commande interne ou externe… » est affiché, cela veut dire que soit le jdk n'est pas installé, soit le PATH n'est pas correctement configuré, voir ci-dessus.

2. Premier exemple.

Le langage Java est 100% objet, et toute application est contenue dans une ou plusieurs classes.

Q1 Créer un dossier pour les exercices du TP1, par exemple courstp1.

- Ecrire le fichier source ci-dessous, avec Notepad++.

```
public class Bonjour1
{
   public static  void main(String[] args)
   {
       System.out.println("Bonjour !! ");
   }
}
```

- Enregistrer le fichier sous le nom exact: **Bonjour1.java**
  Vous n'avez pas d'autre choix, car java impose l'extension .java, et veut que le nom du fichier soit le même que le nom de la classe, aux majuscules minuscules près.
  bonjour1.java Bonjour1.txt essai.java sont 3 erreurs.

💣 *Il faut maintenant ouvrir une fenêtre console (invite de commandes ou terminal ou exécution de **cmd** sous Windows) et se déplacer avec la commande **cd** dans le dossier*

*contenant le fichier Bonjour1.java. La commande **dir** (**ls** sous Linux) dans ce dossier doit afficher le nom du fichier Bonjour1.java.*

- Compiler le fichier Bonjour1.java avec le compilateur javac, en écrivant la commande suivante :

```
javac Bonjour1.java
```

S'il y a des erreurs de syntaxe, elles sont indiquées par le compilateur, et il faut revenir dans l'éditeur pour les corriger.
Si la compilation réussit, elle crée un fichier **Bonjour1.class**.

Le fichier Bonjour1.class est un fichier de **bytes codes**, ou suite d'octets, qui ne sont pas directement compréhensibles par la machine réelle : c'est **la machine virtuelle java** (**JVM**) qui contrôle l'exécution des bytes codes, comme le fait un microprocesseur qui exécute une suite d'instructions.

- Exécuter l'application avec l'interpréteur java, en écrivant la commande suivante :

```
java Bonjour1
```

L'exécution affiche **Bonjour !!** sur l'écran en mode texte.

Ajout de commentaires :

Un programme Java, avant de pouvoir être exécuté, doit être compilé en bytes codes. La possibilité de forcer le compilateur à ignorer certaines instructions existe ! C'est ce qu'on appelle les **commentaires**, et deux syntaxes sont disponibles pour commenter un programme :
1. Les commentaires unilignes : introduits par les symboles « // », ils mettent tout ce qui les suit en commentaire, du moment que le texte se trouve sur la même ligne ;
2. Les commentaires multilignes : ils sont introduits par les symboles « /* » et se terminent par les symboles « */ ».

Explication du premier exemple.

Pour cela, des commentaires ont été ajoutés :

```
public class Bonjour1  //définition de la classe Bonjour1
{ // début de la définition de la classe Bonjour1

  /* On place dans la définition de la classe les membres de la classe :
   Ici, la classe ne contient qu'un seul membre, c'est la méthode (ou fonction membre)
   nommée main()*/

   public static  void main(String[] args) // entête de la méthode main()
   {    // début du corps de la méthode main()
        System.out.println("Bonjour !! ");
   }    // fin du corps de la méthode main()

} // fin de la définition de la classe Bonjour1
```

- On doit écrire une **classe**. On a choisi le nom **Bonjour1** comme nom de la classe. La définition de la classe doit être comprise dans un bloc délimité par une accolade ouvrante et une accolade fermante : « { .... } »

- Une **méthode (ou fonction membre) est une suite d'instructions à exécuter**. C'est un morceau de logique de notre programme. Une méthode contient :
  - un en-tête : celui-ci va être en quelque sorte la carte d'identité de la méthode ;
  - un corps : le contenu de la méthode. Il est placé dans un bloc délimité par des accolades : « { .... } »

- Dans cette classe, il n'existe qu'une seule **méthode (ou fonction membre)** : **la méthode main().** Cette méthode main() est **le point d'entrée d'exécution du programme.** Tout programme exécutable java doit contenir une classe avec une méthode **main()**.

- **public static void main(String[] args)** // entête de la méthode main()
  La méthode main() peut recevoir une liste d'arguments (ou paramètres) de type **String (**chaîne de caractères**)**. Elle est **public comme la classe car ainsi elle peut être appelée/utilisée par un objet externe à la classe Bonjour1**. Elle est **static car elle doit être appelée par la JVM alors que l'objet instance de la classe Bonjour1 n'est pas créé.** Elle ne retourne jamais de valeur et donc le type de sa valeur de retour est **void**.
  La JVM exécute **Bonjour1.main()** pour exécuter l'application.

- Dans la méthode main(), c'est à dire dans le corps de cette méthode, il n'y a qu'une seule instruction. Celle-ci est **terminée par un point virgule** (comme toutes les instructions du langage Java) et réalise l'affichage simple d'un texte :

  System.out.println("Bonjour !! ");

  La classe **System** est une classe du **package java.lang**.
  out permet d'accéder à l'écran qui est la sortie d'affichage standard. **out** est un **objet** (flux instance de la classe PrintStream) qui identifie la sortie standard, cet objet est contenu dans la classe System, il est systématiquement et automatiquement créé pour chaque exécution d'un programme Java.
  println() est une méthode (fonction membre) de la classe **PrintStream** qui écrit dans un flux de sortie, le texte passé en paramètre (entre les parenthèses) avec retour à la ligne à la fin.
  Remarque : l'opérateur « **.** » permet d'accéder à un membre d'une classe ou d'un objet.

  Une application java commence le plus souvent par des **importations** de classes contenues dans des packages, en écrivant des lignes de code comme
  import java.lang.*;

  La ligne de code précédente permet d'importer toutes les classes (grâce à l'utilisation du symbole *****) contenues dans le package java.lang
  java.lang est le seul package dont toutes les classes sont importées automatiquement. Donc, la ligne de code précédente est inutile.
  Un package est une entité java qui regroupe plusieurs classes déjà écrites par les développeurs du langage. On peut voir un package comme une **librairie de classes**. De nombreux packages existent : pour le réseau, pour faire des ihm (fenêtres), pour accéder aux bases de données...

Q2 Importer toutes les classes du package java.lang dans le premier exemple, compiler,

exécuter, qu'est-ce que cela change ?

Pour écrire des applications Java, il faut acquérir de l'expérience sur les packages et les classes, et utiliser la documentation Java Doc accessible (ou téléchargeable) sur le site d'Oracle à l'adresse : **https://docs.oracle.com/javase/8/docs/index.html**

3. Exemple n°2

La méthode print() est une méthode de la classe **PrintStream** qui écrit dans un flux de sortie, le texte passé en paramètre (entre les parenthèses) **sans** retour à la ligne.

Q3 Compiler et exécuter le programme suivant :

```
public class Bonjour2  //définition de la classe Bonjour2
{ // début de la définition de la classe Bonjour2

   /* On place dans la définition de la classe les membres de la classe :
    Ici, la classe ne contient qu'un seul membre, c'est la méthode (ou fonction membre)
    nommée main()*/

    public static  void main(String[] args) // entête de la méthode main()
    {    // début du corps de la méthode main()
         System.out.print("Bonjour !! ");
         System.out.print("Bienvenue ");
         System.out.print("Nous sommes le lundi 13 avril.");
    }    // fin du corps de la méthode main()

} // fin de la définition de la classe Bonjour2
```

Q4 Modifier le programme pour que les 3 messages s'affichent sur des lignes différentes.

Pour insérer un saut de ligne, on peut aussi utiliser le caractère d'échappement \n. De même, le caractère d'échappement \t permet d'insérer une tabulation.

Q5 Compiler et exécuter le programme suivant :

```
public class Bonjour3
{
    public static  void main(String[] args)
    {
         System.out.print("Bon\tjour");
         System.out.println("\nBien\nvenue ");
         System.out.print("Nous som\tmes le lundi 13 avril.");
    }
}
```

La chaîne de caractères que l'on affiche est entourée par des « " ». En Java, les guillemets (doubles) sont des délimiteurs de chaînes de caractères ! Si vous voulez afficher un guillemet double sur la sortie standard, vous devez « l'échapper » avec un « \ », ce qui donne :
"Test \"ok\" !"

4. Les variables

Elles permettent de **stocker** des nombres, des caractères, des booléens… dont **la valeur peut être modifiée** lors de l'exécution du programme.

Une variable a un nom (identifiant) et un type.

La **déclaration** d'une variable est constituée d'un **type suivi d'un nom** et est terminée par un point virgule :

<type de la variable> <nom de la variable> ;

exemple :

int i ;	déclaration d'une variable i de type int (entier)
int j, k ;	déclaration de 2 variables j et k de type int (entier)
char c ;	déclaration d'une variable c de type char (caractère)
double x , y ;	déclaration de 2 variables x et y de type double (réel)

Remarque : On choisit le type d'une variable en fonction de ce que l'on souhaite stocker (ou mettre) dans la variable.

En Java, il y a deux types de variables :

1. **des variables de type simple ou « primitif » ;**
2. **des variables de type complexe ou des « objets ».**

4.1 Les variables de type simple ou « primitif »

Pour les types simples ou "primitifs", la déclaration d'une variable **réserve l'espace mémoire utilisé par cette variable.**

4.1.1 Les différents types numériques entiers

Type	Espace mémoire occupé	Valeurs possibles
byte	1 octet	$-128 (= -2^7)$ à $+127 (=2^7 -1)$
short	2 octets	$-32768 (= -2^{15})$ à $+32767 (= 2^{15} -1)$
int	4 octets	$- 2\,147\,483\,648 (= -2^{31})$ à $2\,147\,483\,647 (= 2^{31} -1)$
long	8 octets	$- 9.10^{18} (= -2^{63})$ à $+9.10^{18} (= 2^{63} -1)$

Exemples de déclaration et d'initialisation de variables en utilisant les types ci-dessus :

```
public class Test1
{
   public static  void main(String[] args)
   {
        byte temp; // déclaration d'une variable nommée temp de type byte
        temp = 64;      /* on affecte 64 dans la variable temp
                           Comme c'est la 1ère fois qu'on affecte une valeur dans la variable temp,
                           il s'agit d'une initialisation */

        short vitesseMax;      // déclaration d'une variable nommée vitesseMax de type short
        vitesseMax = 32000;  // initialisation de la variable vitesseMax

        // on peut initialiser une variable en même temps qu'on la déclare
```

```
        int tempSoleil = 15600000;
        System.out.println(tempSoleil);    //affichage du contenu de la variable  tempSoleil

        long anneeLumiere;
        anneeLumiere = 9460700000000000L; //ne compile pas sans le L
    }
}
```

L'opérateur « = » est l'opérateur d'affectation. Affecter signifie ranger, mettre ou stocker une valeur dans une variable.

Par défaut (sans suffixe), les constantes entières sont placées dans des int. Donc, sans le L à la fin de la constante entière **9460700000000000**, le compilateur essaie de placer cette valeur dans un int (4 octets), ce qui est impossible. Cela engendre une erreur de compilation.

Remarque : préfixes utilisées en java pour écrire des constantes entières :

- sans préfixe, le nombre est écrit en décimal;
- le préfixe 0x indique que le nombre est écrit en hexadécimal (base 16);
- le préfixe 0b permet d'écrire un nombre en binaire (base 2).

Exemple : 92 = 0x5C = 0b1011100

4.1.2 Les différents types numériques réels

Type	Espace mémoire occupé	Valeurs possibles
float	4 octets	$-3{,}4\times10^{38}$ à $+3{,}4\times10^{38}$ (environ 7 chiffres significatifs)
double	8 octets	$-1{,}79\times10^{308}$ à $+1{,}79\times10^{308}$ (environ 17 chiffres significatifs)

Exemples de déclaration et d'initialisation de variables en utilisant les types ci-dessus :

```
public class Test2
{
    public static  void main(String[] args)
    {
        float pi;
        pi = 3.141592653f; //ne compile pas sans le f

        float nombre = 2;  //ok
        nombre = 2.0f;      //ne compile pas sans le suffixe f

        double division;
        division = 0.33333333333333333333333333334d; // le suffixe d est optionnel
        System.out.println(nombre);    //affichage du contenu de la variable nombre
    }
}
```

Q6 Tester le programme ci-dessus.

Par défaut (sans le suffixe f), les constantes réelles sont placées dans des doubles. Quand une telle constante est affectée dans une variable de type float, il faut donc la convertir en float.

Cette conversion en float est dégradante car elle entraîne une perte de précision et donc cela provoque une erreur de compilation.
Conclusion : pour affecter une constante réelle dans une variable de type float, il faut utiliser le suffixe f (pour que la constante soit placée dans un float !).

4.1.3 Le type char

Une variable de type char permet de stocker un caractère. Les caractères sont codés en unicode sur 16 bits : de 0x0000 à 0xFFFF.
En simplifiant pour les caractères ASCII : Octet de poids fort à 0
octet de poids faible: code ASCII

Le code ASCII
L'ASCII (American Standard Code for Information Interchange) utilise **7 bits** pour représenter un caractère : ils sont donc numérotés en décimal de **0 à 127**, codés en binaire de 0b0000000 à 0b1111111 et en hexadécimal de 0x00 à 0x7F.
Donc, seuls **128** caractères sont représentés.

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	DEL

Les caractères de numéro 0 à 31 et le 127 ne sont pas affichables : ils correspondent à des commandes de contrôle de terminal informatique. Le caractère numéro 32 est l'espace. Les autres caractères sont les chiffres arabes, les lettres latines majuscules et minuscules sans accent et quelques symboles de ponctuation.

Exemples :

```
char a;              //déclaration d'une variable nommée a de type char
a = 'B';             //on affecte le code ASCII de la lettre B dans la variable a
System.out.println(a); //affichage du contenu de la variable a
a = 67;              //on affecte le code ASCII de la lettre C dans la variable a
System.out.println(a); //affichage du contenu de la variable a
a = 0x65;            //on affecte le code ASCII de la lettre e dans la variable a
System.out.println(a); //affichage du contenu de la variable a
```

Remarques : - **les expressions entre quotes (ou apostrophes) sont de type char.**

Q7 Tester le programme ci-dessus.

4.1.4 Le type boolean

Une variable de type boolean, ne peut prendre (ou contenir) que deux valeurs : **true** (vrai) ou **false** (faux), sans guillemets (ces valeurs sont natives dans le langage, il les comprend directement et sait les interpréter).

```
boolean a;              //déclaration d'une variable nommée a de type boolean
a = true;               //on affecte true dans la variable a
System.out.println(a); //affichage du contenu de la variable a
a = false;              //on affecte false dans la variable a
System.out.println(a); //affichage du contenu de la variable a
```

Q8 Tester le programme ci-dessus.

4.1.3 Pour déclarer plusieurs variables du même type :

```
int a = 5, b = 3, c = 85;
```

L'instruction précédente déclare 3 variables de type int. Ces variables sont nommées a, b et c et sont respectivement initialisées à 5, 3 et 85.

4.2 Les objets de type String

Le **type String est une classe** qui fait partie du paquetage java.lang.
Une **variable** de type String, comme toute variable de type classe, est un **objet**.
Les objets String sont des objets constants (immutables).

Exemples :

```
String phrase;                  //déclaration d'un objet nommé phrase de type String
phrase = "toto à la piscine";   //on affecte une chaine de caractères constante dans phrase
System.out.println(phrase);     //affichage du contenu de l'objet phrase
String ph2 = "titi";            //on peut initialiser l'objet ph2 en même temps qu'on le déclare
System.out.println(ph2);        //affichage du contenu de l'objet ph2
```

Remarque : les **chaînes de caractères constante se notent entre guillemets** (doubles) : « " ».

5. Convention de nommage

Pour faciliter la lecture des programmes, les conventions suivantes sont communément adoptées :
- tous les noms de classes doivent commencer par une majuscule ;
- tous les noms de variables doivent commencer par une minuscule ;
- si le nom d'une variable est composé de plusieurs mots, le premier commence par une minuscule, le ou les autres par une majuscule, et ce, sans séparation (exemple : int nombrePlusGrand;)

Remarques :
- tous ces noms doivent être sans accents.
- les noms des types primitifs commencent par une minuscule, ce qui permet de les différencier des types classes.

6. Les opérateurs

6.1 Les opérateurs arithmétiques

Notation en langage java	Signification
+	Addition
-	Soustraction
*	Multiplication
/(avec des variables de type numériques réels)	Division (réelle)
/ (avec des variables de type numériques entiers)	Division entière
% (avec des variables de type numériques entiers)	Reste de la division entière

Ces opérateurs sont des **opérateurs binaires** : ils doivent être appliqués à 2 opérandes (2 variables, 2 constantes ou une variable et une constante).

Exemples :

```
int n1, n2, n3, n4;          //déclaration de variables
n1 = 3 + 1;                  // n1 vaut 4
n2 = 2*6;                    // n2 vaut 12
n3 = n2 / n1;                // n3 vaut 3
n1 = n1 + 1;                 // la JVM calcule la valeur de n1 + 1 et affecte le résultat (5) dans n1
n2 = n2 + 2;                 // n2 vaut 14
n4 = n2 / n1;                // n4 vaut  2 car 14 = 2*5 + 4
n1 = n2 % n1;                // n1 vaut 4
```

6.2 Les opérateurs d'incrémentation et de décrémentation

Incrémenter une variable signifie "Ajouter 1 à la variable".
Décrémenter une variable signifie "Retirer 1 à la variable".

Notation en langage java	Signification
++	Incrémentation
--	Décrémentation

Ces opérateurs sont des opérateurs unaires : ils doivent être appliqués à 1 seul opérande : une variable.

Exemples :

```
int n1 = 0;          //déclaration d'une variable
n1++;                // n1 vaut 1
n1++;                // n1 vaut 2
n1 = n1 + 1;         // n1 vaut 3 (idem n1++;)
n1--;                // n1 vaut 2
n1 = n1 - 1;         // n1 vaut 1 (idem n1- -;)
```

6.3 Les opérateurs d'affectation étendue (+= ; -=; *=; /=; %=)

Exemples :

```
int n1 = 9, n2 = 5;  //déclaration de variables
n1 += 5;             // équivalent à n1 = n1 + 5 => n1 vaut 14
n1 /= n2;            // équivalent à n1 = n1 / n2 => n1 vaut 2
n1 += 7;             // n1 vaut 9
n1 %= 2;             // équivalent à n1 = n1 % 2 => n1 vaut 1
```

7. Les saisies (au clavier)

Pour saisir des variables, on peut utiliser la classe Scanner qui fait partie du package java.util. Comme cette classe Scanner ne fait pas partie du paquetage java.lang dont toutes les classes sont importées par défaut, il faut importer cette classe explicitement en début de programme :

```
import java.util.Scanner;
```

Ensuite dans la définition de la fonction main(), il créer un objet de la classe Scanner de la façon suivante :

```
Scanner sc = new Scanner(System.in);
```

On peut diviser l'instruction précédente en 2 :

```
Scanner sc;                    //déclaration de l'objet nommé sc          (*1)
sc = new Scanner(System.in);   //création de l'objet de la classe Scanner (*2)
```

(*1) : la déclaration de l'objet correspond plus exactement à la **déclaration d'une variable référence** nommée sc destinée à contenir l'adresse mémoire (ou la référence) d'un objet de type classe Scanner.
Vous avez le droit de choisir un autre nom pour cette variable référence.

(*2) : l'opérateur **new permet de créer un objet**, **ce qui implique la réservation de l'espace mémoire occupé par cet objet.** Il faut lui indiquer le type de l'objet à créer (ici, le type classe Scanner). De plus, la création de cet objet (de type Scanner) nécessite la transmission d'un paramètre : l'objet nommé in de type InputStream, qui fait partie de la classe System (l'objet in est associé à l'entrée standard, par défaut le clavier). Après avoir créer l'objet de la classe Scanner, **l'opérateur new renvoie l'adresse mémoire de l'objet créé et cette dernière est affectée dans sc.**

7.1 Saisie d'une chaîne de caractère

Il faut utiliser la méthode (ou fonction membre) nextLine() de la classe Scanner. Cette méthode renvoie un String (*3), qu'il suffit d'affecter dans un objet de type String (afin de pouvoir utiliser la chaîne saisie dans la suite du programme).

```
String str;                          //déclaration de l'objet nommé str                    (*4)
System.out.print("Saisir une chaine de caractères : ");
str = sc.nextLine();                 //saisie d'un String et affectation de la chaine saisie dans str   (*5)
System.out.print("Merci pour : ");
System.out.println(str);             // affichage de la chaine contenue dans str           (*6)
```

(*4) : plus exactement, déclaration d'une variable référence destinée à contenir l'adresse mémoire (ou la référence) d'un objet de type String.

(*5) : L'opérateur « **.** » indique que l'on appelle la méthode nextLine() pour l'objet sc. Suite à l'appel de cette fonction, le programme attend que l'utilisateur saisisse une chaîne de caractères et appuie sur la touche "Entrée". Après l'appui sur "Entrée", la fonction fabrique ou crée un objet de type String et y stocke la chaîne saisie. Quand l'exécution de la fonction nextLine() se termine, elle renvoie l'adresse mémoire du String qu'elle a créée et cette dernière est affectée dans str.

(*3) : la fonction nextLine() renvoie la référence ou l'adresse mémoire du String.

(*6) : affichage de la chaîne de caractère contenu dans le String référencé par str.

Q9 Tester le programme ci-dessus.

7.2 Saisie d'une variable de type primitif

La classe Scanner possède une méthode par type primitif : les méthodes nextByte(), nextShort(), nextInt(), nextLong(), nextFloat(), nextDouble() et nextBoolean() permettent la saisie de variables des types correspondants.

Exemples :

```
long nb4;                            //déclaration d'une variable de type int nommée nb
boolean bo;
System.out.print("Saisir un entier : ");
nb4 = sc.nextLong();                 //saisie d'un long et affectation de l'entier saisi dans nb4
System.out.print("Merci pour : ");
System.out.println(nb4);                        // affichage de la valeur contenu dans nb4
System.out.print("Saisir un boolean : ");
bo = sc.nextBoolean();               //saisie d'un boolean et affectation du boolean saisi dans bo
System.out.print("Merci pour : ");
System.out.println(bo);                         // affichage de la valeur contenu dans bo
```

Q10 Tester le programme ci-dessus.

Remarques :

- si l'utilisateur saisit n'importe quoi (une chaîne de caractères à la place d'un nombre par exemple), le programme plante. On verra comment gérer ce problème par la suite car d'une façon générale, on ne peut pas faire confiance aux données saisies par l'utilisateur.
- dans la classe Scanner, il n'existe pas de fonction (nommée nextChar()) permettant la saisie d'un simple caractère. On ne peut que saisir une chaîne de caractère (objet de type String) avec la fonction nextLine() puis accéder aux divers caractères saisis en utilisant la fonction membre charAt() de la classe String (cf. exemple suivant)

```
String str;                          //déclaration de l'objet nommé str

System.out.print("Saisir une chaine de caractères : ");
str = sc.nextLine();                 //saisie d'un String et affectation de la chaine saisie dans str
```

```
System.out.print("Merci pour : ");
System.out.println(str);                    // affichage de la chaine contenue dans str

char car1, car2, car3;              //déclaration de 3 variables de type char
car1 = str.charAt(0);               // (*7)
car2 = str.charAt(1);
car3 = str.charAt(2);
System.out.println("Les 3 premiers caractères :");
System.out.println(car1);
System.out.println(car2);
System.out.println(car3);
```

(*7) : appel de la fonction charAt() pour l'objet str. Cette fonction prend un paramètre qui représente l'indice ou la position du caractère qu'elle renvoie. **Le 1er caractère est situé à l'indice 0 (**cf. cours sur les tableaux**)**, le second à l'indice 1, le 3ème à l'indice 2, etc...

Q11 Tester le programme ci-dessus.

7.3 Exercice

Q12 Réaliser un programme qui :

- déclare 2 variables longueur et largeur de type entier, saisit les 2 entiers correspondants,
- calcule et affiche la surface du rectangle,
- calcule et affiche le périmètre du rectangle.

Q13 Réalisez un programme qui demande le prix TTC d'un objet et qui en affiche le prix HT ainsi que la TVA (20% en France).
Modifier ce programme en ajoutant la demande du nombre d'objet puis en calculant et affichant le prix final TTC, total HT et la TVA totale.

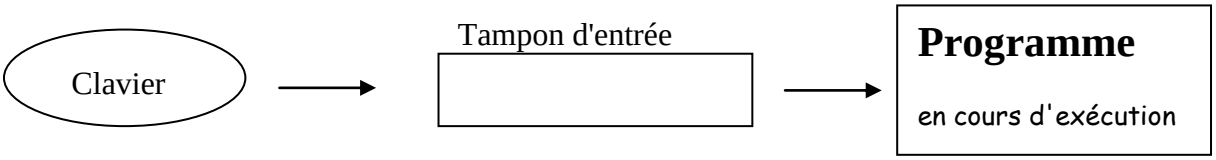
Q14 Ecrire un programme qui échange la valeur de deux variables.
Exemple, si a = 2 et b = 5, le programme donnera a = 5 et b = 2.

Q15 Réaliser un programme qui :

- demande la saisie d'un entier, puis affiche l'entier saisi,
- demande la saisie d'une chaîne de caractère, puis affiche la chaîne saisie.

Si vous respectez ce qui a été vu précédemment, ça ne fonctionne pas : l'utilisateur n'a pas la main pour saisir la chaîne de caractères.

Explication :
Toutes les saisies se font par l'intermédiaire d'un **tampon** (ou buffer) **d'entrée** :



Le contenu du tampon est "analysé" lorsque l'utilisateur "tape" sur la touche Entrée : le caractère Line Feed (LF, '\n' en langage java) est alors inséré dans le tampon.

Soit le code suivant :

```
String str;                    //déclaration de l'objet nommé str
int nb;

System.out.print("1. Saisir un entier : ");
nb = sc.nextInt();             //saisie d'un int et affectation de l'entier saisi dans nb
System.out.print("1. Merci pour : ");
System.out.println(nb);        // affichage de la valeur contenue dans nb

System.out.print("2. Saisir une chaine de caractères : ");
str = sc.nextLine();           //saisie d'un String et affectation de la chaine saisie dans str
System.out.print("2. Merci pour : ");
System.out.println(str);             // affichage de la chaine contenue dans str
```

Lors de l'exécution, si l'utilisateur **saisit "123456\n"** à la suite de l'invitation "1. Saisir un entier : ", ces 7 caractères sont transmis dans le tampon d'entrée et analysés au moment de l'appui sur la touche Entrée. Les 6 chiffres sont alors extraits du tampon et utilisés pour former un nombre qui est affecté dans nb. **Le caractère '\n' est lui laissé dans le tampon.**
Du coup lors de la saisie de la chaîne de caractères qui suit, comme le tampon contient déjà un '\n', l'appel de la fonction nextLine() n'est pas bloquant. Cette fonction renvoie une chaîne de caractères vide et extrait le caractère '\n' (restant) du tampon. Donc après l'exécution de l'instruction « str = sc.nextLine(); », le tampon est vide !

Remarque : le comportement de la méthode nextLine() est différent de celui des autres méthodes de la classe Scanner (comme nextInt(), nextDouble(), nextBoolean(), ...) :

Comportement de la méthode nextLine() :
Lors de l'appel de cette méthode, elle attend qu'il y ait un caractère '\n' dans le tampon. A partir du moment où un '\n' est présent dans le tampon d'entrée, les caractères qui le précèdent sont utilisés pour former le String renvoyé par la méthode, **et le caractère '\n' est supprimé du tampon**. Donc, **après l'appel de nextLine(), le tampon est forcément vide**. De même, si il y avait un '\n' dans le tampon avant l'appel de nextLine(), l'utilisateur n'a pas la main pour saisir quelque chose.

Comportement des autres méthodes (comme nextInt(), nextDouble(), nextBoolean(), ...) :
Lors de l'appel d'une de ces méthodes, si il reste un '\n' dans le tampon , celui est extrait du tampon et la **méthode correspondante attend la saisie** d'un entier, d'un réel ou d'un booléen terminée par un appui sur la touche Entrée, ce qui provoque l'insertion d'un '\n' dans le tampon. Donc, l'utilisateur a forcément la main et s'il saisit **"158.6\n"** dans le cas d'un appel de nextDouble() par exemple, les 5 premiers caractères sont extraits pour former un réel qui est renvoyé en valeur de retour de la méthode. **Le caractère '\n' (**qui suit le '6'**) est lui laissé dans le tampon.**

Conclusion : on peut enchaîner les saisies en appelant les méthodes comme nextInt(), nextDouble(), nextBoolean(), ... sans se poser de question. Mais quand, après l'appel d'une de ces méthodes, on veut saisir un String, il faut vider le tampon (en appelant "à blanc" la méthode nextLine()) pour pouvoir effectuer la saisie de ce String.

Solution de l'exercice :

```
String str;                    //déclaration de l'objet nommé str
int nb;
```

```
System.out.print("1. Saisir un entier : ");
nb = sc.nextInt();                  //saisie d'un int et affectation de l'entier saisi dans nb
System.out.print("1. Merci pour : ");
System.out.println(nb);             // affichage de la valeur contenue dans nb

System.out.print("2. Saisir une chaine de caractères : ");
sc.nextLine();                      // pour supprimer le '\n' qui reste dans le tampon (*1)
str = sc.nextLine();                //saisie d'un String et affectation de la chaine saisie dans str
System.out.print("2. Merci pour : ");
System.out.println(str);                    // affichage de la chaine contenue dans str
```

(*1) : le String renvoyé par nextLine() contenant une chaîne vide est perdu !

Q14 Tester le programme ci-dessus.

8. Les conditions

Elles permettent d'**exécuter des instructions différentes suivant différents cas de figures**.

8.1 Structure algorithmique conditionnelle

```
int nb;

System.out.print("Saisir un entier : ");
nb = sc.nextInt();                  //saisie d'un int et affectation de l'entier saisi dans nb

if (nb < 0)         // if (condition logique)
{
        System.out.println("nb est négatif."); // exécuté que si nb <0
}
System.out.println("Fin du programme.");     // exécuté dans tous les cas.
```

Le mot « if » est un mot clé du langage java. Il doit être suivi d'un **condition logique** placée entre parenthèses. Cette condition logique est un **booléen qui ne peut donc prendre que 2 valeurs : VRAI ou FAUX.** Elle est évaluée (ou calculée) au moment de l'exécution du programme. Le code placé dans le bloc (délimité par les accolades) qui suit n'est exécuté que si la condition logique est évaluée à VRAI.

Q15 Tester le programme ci-dessus dans les 2 cas de figure.

8.2 Structure algorithmique alternative

```
int nb;

System.out.print("Saisir un entier : ");
nb = sc.nextInt();                  //saisie d'un int et affectation de l'entier saisi dans nb

if (nb < 0)         // if (condition logique)
{
        // exécuté si la condition logique évaluée est vraie
        System.out.print("nb");
```

```
        System.out.println(" est négatif.");
}
else
{
        System.out.println("nb est positif ou nul.");    // exécuté si la condition logique est fausse
}
System.out.println("Fin du programme.");     // exécuté dans tous les cas.
```

Q16 Tester le programme ci-dessus dans les 2 cas de figure.

Le mot « else » est un mot clé du langage java. **Il ne peut être utilisé qu'à la suite d'un bloc précédé d'un « if »**. En effet, les instructions contenues dans le bloc à la suite du « else » ne sont exécutées que si la condition logique évaluée dans le « if » est fausse.

Remarque : l'utilisation d'une clause « else » à la suite d'un « if » est **facultative.**

8.3 Structures algorithmiques alternatives imbriquées

```
int nb;
System.out.print("Saisir un entier : ");
nb = sc.nextInt();                  //saisie d'un int et affectation de l'entier saisi dans nb

if (nb < 0)        // if (condition logique)
{
        // exécuté si la condition logique évaluée est vraie
        System.out.print("nb");
        System.out.println(" est négatif.");
}
else
{
        if (nb >0)
        {
                System.out.print("nb");
                System.out.println(" est positif .");
        }
        else
        {
                System.out.println("nb est nul.");
        }
}
System.out.println("Fin du programme.");     // exécuté dans tous les cas.
```

Q17 Tester le programme ci-dessus dans les 3 cas de figure.

Remarque : à chaque entrée dans un nouveau bloc, j'ai décalé d'une tabulation vers la gauche les instructions contenues dans ce bloc. Ces divers décalages constitue **l'indentation** du programme et ont pour but de faciliter sa lecture. Conseil : pour indenter un programme, ne jamais utiliser d'espace, **toujours utiliser des tabulations**.

8.4 Remarque

En réalité, l'utilisation d'un **bloc** (délimité par les accolades) à la suite d'une clause « if » n'est **obligatoire que si on souhaite exécuter plus d'une instruction** quand la condition logique testée est vraie. En effet, sans les accolades à la suite d'un if, seule l'instruction (terminée par

un point virgule) qui suit le if est exécutée lorsque la condition logique testée est vraie.
Idem pour la clause else.

Q18 Reprendre les exemples des paragraphes 8.1, 8.2 et 8.3 en supprimant les accolades.

8.5 Les opérateurs relationnels

Pour fabriquer une condition logique ou un booléen, vous disposez des opérateurs relationnels suivants :

Notation en java	Signification
= =	Egal à
!=	Différent de
<	Inférieur à
>	Supérieur à
<=	Inférieur ou Egal à
>=	Supérieur ou Egal à

Ces opérateurs sont des **opérateurs binaires** : ils doivent **être appliqués à 2 opérandes** et **fournissent en résultat une expression logique ou un booléen.**

Exemple : reprise de l'exemple 8.1

```
int nb;
boolean bo;

System.out.print("Saisir un entier : ");
nb = sc.nextInt();                    //saisie d'un int et affectation de l'entier saisi dans nb

bo = nb < 0;
if (bo)   // idem  if (bo == true)
{
        System.out.println("nb est négatif."); // exécuté que si nb <0
}
System.out.println("Fin du programme.");     // exécuté dans tous les cas.
```

Q19 Tester le programme ci-dessus

8.6 Les opérateurs logiques

Notation en java	Signification
&&	Fonction logique ET
||	Fonction logique OU
!	Fonction logique NON

Ces opérateurs sont des **opérateurs binaires** : ils doivent être appliqués à **2 opérandes qui sont interprétées comme des booléens**. Ces opérateurs **fournissent en résultat une expression logique ou un booléen.**

Tables de vérité des différents opérateurs logiques :
Soit A et B des expressions logiques :

A	B	A || B
false	false	false
false	true	true
true	false	true
true	true	true

A	B	A && B
false	false	false
false	true	false
true	false	false
true	true	true

A	!A
false	true
true	false

Exemple à tester dans la fonction main():
Avec :

```
int largeur = 4, longueur = 10;

// ! (longueur == 10)   vaut !(true) ce qui vaut false
System.out.println(longueur == 10);
System.out.println(!(longueur == 10));
System.out.println(longueur > 10));

// (largeur > 5) || (longueur <= 10) vaut false || true ce qui vaut true
// (!(largeur > 6)) && (longueur < 11) vaut (! false) && true ce qui vaut true
System.out.println((largeur > 5) || (longueur <= 10));
System.out.println((largeur > 5) && (longueur <= 10));
```

8.7 Structure algorithmique à choix multiples : SWITCH CASE

Cette structure algorithmique permet d'effectuer des actions différentes **en fonction des valeurs discrètes que peut prendre une même variable**.

```
int nb;

System.out.print("Saisir un entier : ");
nb = sc.nextInt();                  //saisie d'un int et affectation de l'entier saisi dans nb

switch (nb)      //
{
        case 1 :  // exécuté si nb vaut 1
                  System.out.println("nb vaut 1.");
                  break;          //signifie : sortir du block
        case 2 :  // exécuté si nb vaut 2
                  System.out.println("nb vaut 2.");
                  break;
        case 3 :  // exécuté si nb vaut 3
                  System.out.print("nb");
                  System.out.println(" vaut 3.");
                  break;

        default : // exécuté si nb est différent 1, 2 et 3
                  System.out.println("nb est différent des valeurs testées dans les case.");
}
System.out.println("Fin du programme.");     // exécuté dans tous les cas.
```

Les mots « switch», « case», « break» et « default» sont des mots clés du langage java.

Q20 Tester le programme ci-dessus dans tous les cas de figure.

Q21 Réaliser un programme qui donne les mêmes résultats en utilisant les mots clés « if » et « else » à la place des mots clés « switch », « case », « break » et « default ».

Q22 Retour sur le programme de la question 18. Supprimer toutes les instructions « break; » et tester le programme dans tous les cas de figure. Conclusion?

9. Exercices

9.0 Les règles d'écriture d'un programme :

- Mettre des commentaires
- Respecter l'indentation
- Choisir des noms de variables explicites, commencent par une lettre minuscule
- Choisir des noms de classes explicites (et donc des fichiers en Java) avec la 1ère lettre en majuscule)

9.1 Exercice 1 : réaliser un programme qui calcule l'aire d'un rectangle dont la longueur et la largeur seront saisies par l'utilisateur, puis affiche le résultat.

9.2 Exercice 2 : réaliser un programme qui :

- demande la saisie de l'heure (sans les minutes);
- affiche "Bonjour" si l'heure est inférieure ou égale à 16 et "Bonsoir" dans le cas contraire.

9.3 Exercice 3 :

Un magasin consent une réduction dans les conditions suivantes :

- *Si le montant de l'achat est strictement inférieur à 350€, il n'y a pas de réduction*
- *Si le montant de l'achat est compris entre 350€ et 600€, 3% de réduction est accordé*
- *Si le montant de l'achat est supérieur ou égal à 600€, 5% de réduction est accordé*

Réaliser un programme qui :

- demande la saisie du montant de l'achat;
- calcule le prix net à payer et l'affiche.

9.4 Exercice 4 : Calcul de l'IMC (indice de masse corporelle)
Rappel : l'IMC est calculé est divisant le poids par la taille au carré.
L'interprétation de l'IMC se fait selon les critères définis par l'organisation mondiale de la santé.

IMC ($Kg.m^{-2}$)	*Interprétation(d'après l'OMS)*
Moins de 16.5	Dénutrition
16.5 à 18.5	Maigreur
18.5 à 25	Corpulence normale
25 à 30	Surpoids
Plus de 30	Obésité

Remarque : cette classification est statistique, elle ne s'applique pas forcément à tous les adultes, notamment les sportifs ou les seniors.

Réaliser un programme qui :

- calcule et affiche l'IMC d'une personne (le poids et la taille seront saisis par l'utilisateur);
- interprète l'IMC calculé en se servant du tableau ci-dessus.

10. Les structures algorithmiques itératives (ou boucles)

Elles permettent de **répéter plusieurs fois le même traitement.**

10.1 La boucle while

Elle permet de **répéter un ensemble d'instructions tant qu'une condition logique est vraie.**

Exemple 1 :

```
int nb = 0;

while( nb != 1) {
        // les instructions contenues dans ce bloc sont répétées tant que la condition
        // logique (nb != 1) est vraie
        System.out.print("Saisir un entier, 1 pour sortir : ");
        nb = sc.nextInt();
}
System.out.println("Fin du programme.");     // exécuté quand on sort de la boucle
```

Le mot « while » est un mot clé du langage java. Il doit être suivi d'un **condition logique** placée entre parenthèses. Cette condition logique est un **booléen qui ne peut donc prendre que 2 valeurs : VRAI ou FAUX.** Elle est évaluée (ou calculée) **avant chaque éventuelle exécution des instructions placées dans le bloc qui suit**. Le code placé dans le bloc (délimité par les accolades) est exécuté tant que la condition logique est évaluée à VRAI.

Q23 Tester le programme ci-dessus.

Q24 Modifier le programme en initialisant la variable nb à la valeur 1. Tester le programme. Conclusion ?

Exemple 2 :

```
int nb = 0;

while( nb < 5) {
        // les instructions contenues dans ce bloc sont répétées tant que la condition
        // logique (nb < 5) est vraie
        nb++;
        System.out.print("nb = ");
        System.out.println(nb);
}
System.out.println("\nFin du programme.");  // exécuté quand on sort de la boucle
```

Q25 Tester le programme ci-dessus. Justifier le résultat obtenu.

Q26 Commenter l'instruction qui incrémente nb. Tester le programme. Pour interrompre son exécution, appuyer sur la combinaison de touches : CTRL C. Conclusion?

10.2 La boucle do while

Dans cette structure, le traitement est **exécuté une fois puis sa répétition se poursuit tant que la condition logique est vérifiée.**

Exemple :

```
int nb = 1;
do {
        // on est certain que les instructions contenues dans ce bloc seront exécutées
        // au moins une fois
        System.out.print("Saisir un entier, 1 pour sortir : ");
        nb = sc.nextInt();
} while( nb != 1); // ne pas oublier le point virgule
System.out.println("\nFin du programme.");  // exécuté quand on sort de la boucle
```

Les mots « do » et « while » sont des mots clés du langage java. **Après chaque exécution** des instructions contenues dans le bloc, la condition logique qui suit le « while » est évaluée. Si elle est évaluée à VRAI, on recommence une nouvelle exécution des instructions contenues dans le bloc, sinon on sort de la boucle.

Q27 Tester le programme ci-dessus.

10.3 La boucle for

Une boucle « for » a la structure suivante :

```
for (expression1 ; expression2 ; expression3) {
        traitement;      //à répéter
}
```

et est équivalente à :

```
expression1;
while (expression2) {
        traitement;      //à répéter
        expression3;
}
```

Le mot « for » est un mot clé du langage java. Il doit être suivi de 3 expressions placées entre parenthèses et séparées par des points virgule. L'expression1 sert généralement à **initialiser une variable utilisée par la boucle**. L'expression2 est **interprétée comme une condition logique**. Quant à l'expression 3, elle sert généralement à **modifier la variable utilisée par la boucle.**

Exemple :

```
int i;

for (i = 0 ; i < 3 ;  i++) {
        //traitement à répéter
        System.out.print("Valeur de i : ");
        System.out.println(i);
}
System.out.print("Quand on sort de la boucle, i vaut ");
System.out.println(i);
```

Dans l'exemple précédent, le traitement est exécuté pour i = 0, puis pour i = 1 et enfin pour i = 2. Quand on sort de la boucle i vaut 3.

Q28 Tester le programme.

Q29 Ecrire un programme qui produit le même résultat en utilisant une boucle while.

Conclusion : la boucle « for » est utilisée quand le nombre de répétitions est connu à l'avance. Elle permet d'écrire ce type de boucle de façon plus concise qu'une boucle « while ».

11. Exercices

11.1 Exercice 1 : réaliser un programme qui affiche 5 fois "Bonjour" en utilisant une boucle.

11.2 Exercice 2 : réaliser un programme qui affiche plusieurs fois "Bonjour" : le nombre d'affichage doit être saisi par l'utilisateur.

11.3 Exercice 3 :
Réaliser un programme qui demande à l'utilisateur un nombre compris entre 0 et 3 (inclus) jusqu'à ce que la réponse convienne. Dans le cas où la réponse ne convient pas, il faut le signaler à l'utilisateur.

11.4 Exercice 4 :
Réaliser un programme qui demande un nombre compris entre 10 et 20 (inclus), jusqu'à ce que la réponse convienne. En cas de réponse supérieure à 20, il faut faire apparaître un message : « Plus petit ! », et inversement, « Plus grand ! » si le nombre est inférieur à 10.

11.5 Exercice 5 :
Réaliser un programme qui demande un nombre de départ, et qui ensuite affiche les dix nombres suivants. Par exemple, si l'utilisateur entre le nombre 17, le programme doit afficher les nombres de 18 à 27.

11.6 Exercice 6 :
Réaliser un programme qui demande un nombre de départ, et qui ensuite écrit la table de multiplication de ce nombre, présentée comme suit (cas où l'utilisateur entre le nombre 7) :

Table de 7 :
7 x 1 = 7
7 x 2 = 14
7 x 3 = 21
...
7 x 10 = 70

11.7 Exercice 7 :
Réaliser un programme qui demande un nombre de départ, et qui calcule la somme des entiers jusqu'à ce nombre. Par exemple, si l'on entre 5, le programme doit calculer :

1 + 2 + 3 + 4 + 5 = 15

NB : on souhaite afficher uniquement le résultat, pas la décomposition du calcul.

11.8 Exercice 8 :
Réaliser un programme qui :
- demande à l'utilisateur de saisir 10 nombres,
- détermine et affiche le plus grand nombre parmi les 10 nombres saisis :

Entrez le nombre numéro 1 : 12
Entrez le nombre numéro 2 : 14
etc.
Entrez le nombre numéro 10 : 6
Le plus grand de ces nombres est : 14

Modifier ensuite l'algorithme pour que le programme affiche en plus la position à laquelle le plus grand nombre est saisi :

Le plus grand nombre a été saisi à la position 2

11.9 Exercice 9 :
Même problème que dans l'exercice 8, mais cette fois-ci le nombre de saisies de l'utilisateur n'est pas fixé à l'avance : la saisie des nombres s'arrête lorsque l'utilisateur entre un zéro.

11.10 Exercice 10 :
Ecrire un programme qui affiche à l'écran le motif suivant :

```
*
**
***
****
*****
******
```

11.11 Exercice 11 :

Ecrire un programme qui affiche à l'écran le motif suivant :

```
     *
    ***
   *****
  *******
**********
```

11.12 Exercice 12 :

Modifier le programme 11.11 afin de saisir la base et la hauteur du sapin.