Chapitre 2

La syntaxe et les éléments de bases de java

1. Règles de base

- Les blocs de code sont encadrés par des accolades
- Chaque instruction se termine par un ";"
- Une instruction peut tenir sur plusieurs lignes.
- L'indentation (la tabulation) est ignorée du compilateur mais elle permet une meilleure compréhension du code par le programmeur.

2. Les identificateurs

Chaque objet, classe, programme ou variable est associé à un nom : l'identificateur qui peut se composer de tous les caractères alphanumériques et des caractères "_" et "$". Le premier caractère doit être une lettre, le caractère de soulignement ou le signe "$".

3. Les commentaires

Ils ne sont pas pris en compte par le compilateur donc ils ne sont pas inclus dans le pseudocode.
Ils ne se terminent pas par un ";".
Il existe trois types de commentaires en Java :

Type de commentaires	Exemple
Commentaire abrégé	`// commentaire sur une seule ligne` `int N=1; // déclaration du compteur`
Commentaire multiligne	`/* commentaires ligne 1` `commentaires ligne 2 */`
Commentaire de documentation automatique	`/** commentaire */`

4. La déclaration et l'utilisation de variables

4.1. La déclaration de variables

Une variable possède un nom, un type et une valeur. La déclaration d'une variable doit donc contenir deux choses : un nom et le type de données qu'elle puisse contenir. Une variable est utilisable dans le bloc où elle est définie.

La déclaration d'une variable permet de réserver la mémoire pour en stocker la valeur.

Le type d'une variable peut être un type élémentaire ou un objet :

type_élémentaire variable ;

class variable ;

Exemple *:*

```
long nombre;
```

Rappel : les noms de variables en Java peuvent commencer par une lettre, par le caractère "_" ou par le signe "$". Le reste du nom peut comporter des lettres ou des nombres mais jamais d'espaces. Il est possible de définir plusieurs variables de même type en séparant chacune d'elles par une virgule.

Exemple:

```
int jour, mois, annee ;
```

Java est un langage à typage rigoureux qui ne possède pas de transtypage automatique lorsque ce transtypage risque de conduire à une perte d'informations.

Pour les objets, il est nécessaire en plus de la déclaration de la variable de créer un objet avant de pouvoir l'utiliser. Il faut réserver de la mémoire pour la création d'un objet (**Remarque :** un tableau est un objet en java) avec l'instruction **new**. La libération de la mémoire se fait automatiquement grâce au garbage collector.

Exemple *:*

```
MaClasse instance; // déclaration de l'objet
instance = new maClasse(); // création de l'objet
OU MaClasse instance = new MaClasse(); // déclaration et création de
// l'objet
```

Exemple **:**

```
int[] nombre = new int[10];
```

Il est possible en une seule instruction de faire la déclaration et l'affectation d'une valeur à une variable ou plusieurs variables.

Exemple **:**

```
int i=3 , j=4 ;
```

4.2. Les types élémentaires

Les types élémentaires ont une taille identique quelque soit la plate-forme d'exécution : c'est un des éléments qui permettent à java d'être indépendant de la plate-forme sur lequel le code s'exécute.

Type	Désignation	Longueur	Valeurs	Commentaires
boolean	valeur logique : true ou false	8 bits	True ou false	pas de conversion possible vers un autre type
byte	Octet signé	8 bits	-128 à 127	
short	Entier court signé	16 bits	-32768 à 32767	
char	caractère Unicode	16 bits	\u0000 à \uFFFF	entouré de cotes simples dans un programme Java
int	Entier signé	32 bits	-2×10^{9} à 2×10^{9}	
Float	virgule flottante simple précision (IEEE754)	32 bits	1.401e-045 à 3.40282e+038	
Double	virgule flottante double précision (IEEE754)	64 bits	2.22507e-308 à 1.79769e+308	
Long	Entier long	64 bits	-9×10^{18} à 9×10^{18}	

Remarque : Les types élémentaires commencent tous par une minuscule.

4.3. Le format des types élémentaires

- **Le format des nombres entiers**

Les types **byte**, **short**, **int** et **long** peuvent être codés en décimal, hexadécimal ou octal. Pour un nombre hexadécimal, il suffit de préfixer sa valeur par 0x. Pour un nombre octal, le nombre doit commencer par un zéro. Le suffixe l ou L permet de spécifier que c'est un entier long.

- **Le format des nombres flottants**

Les types **float** et **double** stockent des nombres flottants : pour être reconnus comme tel ils doivent posséder soit un point, un exposant ou l'un des suffixes f, F, d, D. Il est possible de préciser des nombres qui n'ont pas la partie entière ou décimale.

Exemple :

```
float pi = 3.141f;
double v = 3d
float f = +.1f , d = 1e10f;
```

Par défaut, un littéral est de type double : pour définir un float il faut le suffixer par la lettre f ou F.

Exemple **:**

```
double w = 1.1;
Attention : float pi = 3.141; // erreur à la compilation
```

- **Le format des caractères**

Un caractère est codé sur 16 bis car il est conforme à la norme Unicode. Il doit être entouré par des apostrophes. Une valeur de type char peut être considérée comme un entier non négatif de 0 à 65535. Cependant la conversion implicite par affectation n'est pas possible.

Exemple **:**

```
/* test sur les caractères */
class test1 {
public static void main (String args[]) {
char code = 'D';
int index = code - 'A';
System.out.println("index = " + index);
}
}
```

4.4. L'initialisation des variables

Exemple **:**

```
int nombre; // déclaration
nombre = 100; //initialisation
OU int nombre = 100; //déclaration et initialisation
```

En java, toute variable appartenant à un objet (définie comme étant un attribut de l'objet) est initialisée avec une valeur par défaut en accord avec son type au moment de la création. Cette initialisation ne s'applique pas aux variables locales des méthodes de la classe.
Les valeurs par défaut lors de l'initialisation automatique des variables d'instances sont :

Type	Valeur par défaut
Boolean	False
byte, short, int, long	0
float, double	0.0
Char	\u000

Remarque : Dans une applet, il est préférable de faire les déclarations et les initialisations dans la méthode init().

4.5. L'affectation

Le signe "=" est l'opérateur d'affectation, il est utilisé avec une expression de la forme :

variable = expression.

L'opération d'affectation est associatif de droite à gauche : il renvoie la valeur affectée ce qui permet d'écrire : **x = y = z = 0;**

Il existe des opérateurs qui permettent de simplifier l'écriture d'une opération d'affectation associée à un opérateur mathématique :

Opérateur	Exemple	Signification
=	a=10	équivalent à : a = 10
+=	a+=10	équivalent à : a = a + 10
-=	a-=	équivalent à : a = a - 10
*=	a*=	équivalent à : a = a * 10
/=	a/=10	équivalent à : a = a / 10
%=	a%=10	reste de la division
^=	a^=10	équivalent à : a = a ^ 10
<<=	a<<=10	équivalent à : a = a << 10 a est complété par des zéros à droite
>>=	a>>=10	équivalent à : a = a >> 10 a est complété par des zéros à gauche
>>>=	a>>>=10	équivalent à : a = a >>> 10 décalage à gauche non signé

L'opérateur "<<" signifie un décalage gauche signé. Il décale à gauche les bits de l'opérande gauche, du nombre de chiffres spécifié dans l'opérande droit, complète la droite par des 0. Les bits de poids fort sont perdus.

L'opérateur ">>" signifie un décalage droit signé. Il décale à droite les bits de l'opérande gauche, du nombre de chiffres spécifié dans l'opérande droit. Si l'opérande gauche est négatif, la partie gauche est complétée par des 0 ; s'il est positif, elle est complétée par des 1. Cela préserve le signe initial.

Remarque : Dans un nombre entier signé, le bit le plus à gauche indique le signe positif ou négatif du nombre entier : le bit a la valeur 1 si l'entier est négatif, 0 s'il est positif. Dans Java, les nombres entiers sont toujours signés, alors que dans C/C++, ils sont signés par défaut.

Cependant, dans Java, les opérateurs de décalage conservent le bit du signe, de sorte que le bit du signe est dupliqué, puis décalé. Par exemple, décaler à droite 10010011 de 1 donne 11001001.

Attention : Lors d'une opération sur des opérandes de types différents, le compilateur détermine le type du résultat en prenant le type le plus précis des opérandes. Par exemple, une multiplication d'une variable de type float avec une variable de type double donne un résultat de type double. Lors d'une opération entre un opérande entier et un flottant, le résultat est du type de l'opérande flottant.

4.6. Les comparaisons

Java propose des opérateurs pour toutes les comparaisons :

Opérateur	Exemple	Signification
>	a > 10	strictement supérieur
<	a < 10	strictement inférieur
>=	a >= 10	supérieur ou égal
<=	a <= 10	inférieur ou égal
==	a == 10	Egalité
!=	a != 10	différent de
&	a & b	ET binaire
^	a ^ b	OU exclusif binaire
|	a | b	OU binaire
&&	a && b	ET logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient fausse
||	a || b	OU logique (pour expressions booléennes) : l'évaluation de l'expression cesse dès qu'elle devient vraie
? :	a ? b : c	opérateur conditionnel : renvoie la valeur b ou c selon l'évaluation de l'expression a (si a alors b sinon c) : b et c doivent retourner le même type

Les opérateurs sont exécutés dans l'ordre suivant à l'intérieur d'une expression qui est analysée de gauche à droite:

- incréments et décréments
- multiplication, division et reste de division (modulo)
- addition et soustraction
- comparaison

- le signe = d'affectation d'une valeur à une variable

L'usage des parenthèses permet de modifier cet ordre de priorité.

5. Les opérations arithmétiques

Les opérateurs arithmétiques se notent **+** (addition), **-** (soustraction), ***** (multiplication), / (division) et % (reste de la division). Ils peuvent se combiner à l'opérateur d'affectation.

Exemple **:**

```
nombre += 10;
```

5.1. L'arithmétique entière

Pour les types numériques entiers, Java met en oeuvre une sorte de mécanisme de conversion implicite vers le type int appelée promotion entière. Ce mécanisme fait partie des règles mise en place pour renforcer la sécurité du code.

Exemple :

```
short x = 5 , y = 15;
x = x + y ; //erreur à la compilation
ncompatible type for =. Explicit cast needed to convert int to short.
x = x + y ; //erreur à la compilation
^
1 error
```

Les opérandes et le résultat de l'opération sont convertis en type int. Le résultat est affecté dans un type short : il y a donc risque de perte d'informations et donc erreur à la compilation est émise. Cette promotion évite un débordement de capacité sans que le programmeur soit pleinement conscient du risque : il est nécessaire, pour régler le problème, d'utiliser une **conversion explicite** ou **cast**.

Exemple :

```
x = (short) ( x + y );
```

Il est nécessaire de mettre l'opération entre parenthèse pour que ce soit son résultat qui soit converti car le cast a une priorité plus forte que les opérateurs arithmétiques. La division par zéro pour les types entiers lève l'exception ArithmeticException

Exemple :

```
/* test sur la division par zéro de nombres entiers */
```

```
class test3 {
public static void main (String args[]) {
int valeur=10;
double résultat = valeur / 0;
System.out.println("index = " + résultat);
}
}
```

5.2. L'arithmétique en virgule flottante

Avec des valeurs float ou double, la division par zéro ne produit pas d'exception mais le résultat est indiqué par une valeur spéciale qui peut prendre trois états :

- indéfini : Float.NaN ou Double.NaN (not a number)
- indéfini positif : Float.POSITIVE_INFINITY ou Double.POSITIVE_INFINITY, + ∞
- indéfini négatif : Float.NEGATIVE_INFINITY ou Double.NEGATIVE_INFINITY, + ∞

5.3. L'incrémentation et la décrémentation

Les opérateurs d'incrémentation et de décrémentation sont : n++, ++n, n--, --n

Si l'opérateur est placé avant la variable (préfixé), la modification de la valeur est immédiate sinon la modification n'a lieu qu'à l'issu de l'exécution de la ligne d'instruction (postfixé). L'opérateur ++ renvoie la valeur avant incrémentation s'il est postfixé, après incrémentation s'il est préfixé.

Exemple :

```
System.out.println(x++); // est équivalent à
// System.out.println(x); x = x + 1;
System.out.println(++x); // est équivalent à
// x = x + 1; System.out.println(x);
```

Exemple :

```
/* test sur les incrementations prefixees et postfixees */
class test4 {
public static void main (String args[]) {
int n1=0;
int n2=0;
System.out.println("n1 = " + n1 + " n2 = " + n2);
n1=n2++;
```

```
System.out.println("n1 = " + n1 + " n2 = " + n2);
n1=++n2;
System.out.println("n1 = " + n1 + " n2 = " + n2);
n1=n1++; //attention
System.out.println("n1 = " + n1 + " n2 = " + n2);
}
}
```

Résultat :

```
int n1=0;
int n2=0;
// n1=0 n2=0n1=n2++;
// n1=0 n2=1n1=++n2;
 // n1=2 n2=2n1=n1++;
// attention : n1 ne change pas de valeur
```

6. La priorité des opérateurs

Java définit les priorités dans les opérateurs comme suit (du plus prioritaire au moins prioritaire).

les parenthèses	( )
les opérateurs d'incrémentation	++ et --
les opérateurs de multiplication, division, et modulo	* / et %
les opérateurs d'addition et soustraction	+ et -
les opérateurs de décalage	<< et >>
les opérateurs de comparaison	< > <= et >=
les opérateurs d'égalité	== et !=
l'opérateur OU exclusif	^
l'opérateur ET	&
l'opérateur OU	|
l'opérateur ET logique	&&
l'opérateur OU logique	||
les opérateurs d'assignement	= += et -=

Les parenthèses ayant une forte priorité, l'ordre d'interprétation des opérateurs peut être modifié par des parenthèses.

7. Les structures de contrôles

7.1. Les boucles

➢ La boucle while

```
while ( boolean )
{
... // code à exécuter dans la boucle
}
```

Le code est exécuté tant que le booléen est vrai. Si avant l'instruction while, le booléen est faux, alors le code de la boucle ne sera jamais exécuté .

Ne pas mettre de ";" après la condition sinon le corps de la boucle ne sera jamais exécuté

➢ La boucle do .. while

```
do {
...
} while ( boolean )
```

Cette boucle est au moins exécuté une fois quelque soit la valeur du booléen.

➢ La boucle for

```
for ( initialisation; condition; modification) {
...
}
```

Exemple :

```
for (i = 0 ; i < 10; i++ ) { ....}
for (int i = 0 ; i < 10; i++ ) { ....}
for ( ; ; ) { ... } // boucle infinie
```

L'initialisation, la condition et la modification de l'index sont optionnelles.

Dans l'initialisation, on peut déclarer une variable qui servira d'index et qui sera dans ce cas locale à la boucle.

Il est possible d'inclure plusieurs traitements dans l'initialisation et la modification de la boucle : chacun des traitements doit être séparé par une virgule.

Exemple :

```
for (i = 0 , j = 0 ; i * j < 1000;i++ , j+= 2) { ....}
```

La condition peut ne pas porter sur l'index de la boucle :

Exemple :

```
boolean trouve = false;
```

```
for (int i = 0 ; !trouve ; i++ ) {
if ( tableau[i] == 1 )
trouve = true;
... //gestion de la fin du parcours du tableau
}
```

Il est possible de nommer une boucle pour permettre de l'interrompre même si cela est peu recommandé :

Exemple :

```
int compteur = 0;
boucle:
while (compteur < 100) {
for(int compte = 0 ; compte < 10 ; compte ++) {
compteur += compte;
System.out.println("compteur = "+compteur);
if (compteur> 40) break boucle;
}
}
```

7.2. Les branchements conditionnels

```
if (boolean) {
...
} else if (boolean) {
...
} else {
...
}
swith (expression) {
case constante1 :
instr11;
instr12;
break;
case constante2 :
...
default :
...
}
```

On ne peut utiliser switch qu'avec des types primitifs d'une taille maximum de 32 bits (byte, short, int, char).

Si une instruction case ne contient pas de break alors les traitements associés au case suivant sont exécutés.

Il est possible d'imbriquer des switch

- **L'opérateur ternaire :** (condition) ? valeur-si-vrai : valeur-si-faux

Exemple :

```
if (niveau == 5) // equivalent à total = (niveau ==5) ? 10 : 5;
total = 10;
else total = 5 ;
System.out.println((sexe == « H ») ? « Mr » : « Mme »);
```

7.3. Les débranchements

break : permet de quitter immédiatement une boucle ou un branchement. Utilisable dans tous les contrôles de flot

continue : s'utilise dans une boucle pour passer directement à l'itération suivante

break et **continue** peuvent s'exécuter avec des blocs nommés. Il est possible de préciser une étiquette pour indiquer le point de retour lors de la fin du traitement déclenché par le break.

Une étiquette est un nom suivi de " ;" qui définit le début d'une instruction.

8. Les tableaux

Ils sont dérivés de la classe **Object** : il faut utiliser des méthodes pour y accéder dont font parti des messages de **Object** tel que **equals()** ou **getClass()**. Le premier élément possède l'indice 0.

8.1. La déclaration des tableaux

Java permet de placer les crochets après ou avant le nom du tableau dans la déclaration.

Exemple :

```
int tableau[] = new int[50]; // déclaration et allocation
OU int[] tableau = new int[50];
OU int tab[]; // déclaration
tab = new int[50]; //allocation
```

Java ne supporte pas directement les tableaux à plusieurs dimensions : il faut déclarer un tableau de tableau.

Exemple :

```
float tableau[][] = new float[10][10];
```

La taille du tableau de la seconde dimension peut ne pas être identique pour chaque occurrence.

Exemple:

```
int dim1[][] = new int[3][];
dim1[0] = new int[4];
dim1[1] = new int[9];
dim1[2] = new int[2];
```

Chaque élément du tableau est initialisé selon son type par l'instruction new : **0** pour les numériques, **\0** pour les caractères, **false** pour les booléens et **nil** pour les chaînes de caractères et les autres objets.

8.2. L'initialisation explicite d'un tableau

Exemple :

```
int tableau[5] = {10,20,30,40,50};
int tableau[3][2] = {{5,1},{6,2},{7,3}};
```

La taille du tableau n'est pas obligatoire si le tableau est initialisé à sa création.

Exemple :

```
int tableau[] = {10,20,30,40,50};
```

Le nombre d'éléments de chaque ligne peut ne pas être identique.

Exemple :

```
int[][] tabEntiers = {{1,2,3,4,5,6},
{1,2,3,4},
{1,2,3,4,5,6,7,8,9}};
```

8.3. Le parcours d'un tableau

Exemple :

```
for (int i = 0; i < tableau.length ; i ++) { ... }
```

La variable **length** retourne le nombre d'éléments du tableau.

Pour passer un tableau à une méthode, il suffit de déclarer les paramètres dans l'en-tête de la méthode.

Exemple :

```
public void printArray(String texte[]){ ...
}
```

Les tableaux sont toujours transmis par référence puisque se sont des objets.

Un accès à un élément d'un tableau qui dépasse sa capacité, lève une exception du type **java.lang.arrayIndexOutOfBoundsException**.

9. Les conversions de types

Lors de la déclaration, il est possible d'utiliser un cast :

Exemple :

```
int entier = 5;
float flottant = (float) entier;
```

La conversion peut entraîner une perte d'informations.

Il n'existe pas en java de fonctions pour convertir : les conversions de type se font par des méthodes. La bibliothèque de classes API fournit une série de classes qui contiennent des méthodes de manipulation et de conversion de types élémentaires.

Classe	Rôle
String	pour les chaînes de caractères Unicode
Integer	pour les valeurs entières (integer)
Long	pour les entiers long signés (long)
Float	pour les nombres à virgules flottante (float)
Double	pour les nombres à virgule flottante en double précision (double)

Les classes portent le même nom que le type élémentaire sur lequel elles reposent avec la première lettre en majuscule.

Ces classes contiennent généralement plusieurs constructeurs. Pour y accéder, il faut les instancier puisque de sont des objets.

Exemple :

```
String montexte;
montexte = new String("test");
```

L'objet montexte permet d'accéder aux méthodes de la classe java.lang.String

9.1. La conversion d'un entier int en chaîne de caractère String

Exemple :

```
int i = 10;
```

```
String montexte = new String();
montexte =montexte.valueOf(i);
```

valueOf est également définie pour des arguments de type boolean, long, float, double et char.

9.2. La conversion d'une chaîne de caractères String en entier int

Exemple :

```
String montexte = new String("10");
Integer nomnombre=new Integer(montexte);
int i = monnombre.intValue(); //conversion d'Integer en int
```

9.3. La conversion d'un entier int en entier long

Exemple :

```
int i=10;
Integer monnombre=new Integer(i);
long j=monnombre.longValue();
```

10. La manipulation des chaînes de caractères

10.1. La définition d'un caractère

Exemple :

```
char touche = '%';
```

10.2. La définition d'une chaîne

Exemple :

```
String texte = "bonjour";
```

Les variables de type String sont des objets. Partout ou des constantes chaînes figurent entre guillemets, le compilateur Java génère un objet de type String avec le contenu spécifié. Il est donc possible d'écrire : **String texte = "Java Java Java".replace('a','o');**

Les chaînes ne sont pas des tableaux : il faut utiliser les méthodes de la classes String d'un objet instancié pour effectuer des manipulations.

Il est impossible de modifier le contenu d'un objet String construit à partir d'une constante. Cependant, il est possible d'utiliser les méthodes qui renvoient une chaîne pour modifier le contenu de la chaîne

Exemple :

```
String texte = "Java Java Java";
```

```
texte = texte.replace('a','o');
```

Java ne fonctionne pas avec le jeu de caractères ASCII ou ANSI, mais avec Unicode (Universal Code). Ceci concerne les types char et les chaînes de caractères. Le jeu de caractères Unicode code un caractère sur 2 octets. Les caractères 0 à 255 correspondent exactement au jeu de caractères ASCII.

10.3. Les caractères spéciaux dans les chaînes

Caractères spéciaux	Affichage
\'	Apostrophe
\"	Guillemet
\\	antislash
\t	Tabulation
\b	retour arrière (backspace)
\r	retour chariot
\f	saut de page (form feed)
\n	saut de ligne (newline)
\0ddd	caractère ASCII ddd (octal)
\xdd	caractère ASCII dd (hexadécimal)
\udddd	caractère Unicode dddd (hexadécimal)

10.4. L'addition de chaînes

Java admet l'opérateur + comme opérateur de concaténation de chaînes de caractères.

L'opérateur + permet de concaténer plusieurs chaînes. Il est possible d'utiliser l'opérateur +=

Exemple :

```
String texte = "";
texte += "Hello";
texte += " World3";
```

Cet opérateur sert aussi à concaténer des chaînes avec tous les types de bases. La variable ou constante est alors convertie en chaîne et ajoutée à la précédente. La condition préalable est d'avoir au moins une chaîne dans l'expression sinon le signe "+" est évalué comme opérateur mathématique.

Exemple :

```
System.out.println("La valeur de Pi est : " + Math.PI);
int duree = 121;
System.out.println("durée = " + duree);
```

10.5. La comparaison de deux chaînes

Il faut utiliser la méthode **equals()**

Exemple :

```
String texte1 = "texte 1";
String texte2 = "texte 2";
if ( texte1.equals(texte2) )...
```

10.6. La détermination de la longueur d'une chaîne

La méthode **length()** permet de déterminer la longueur d'une chaîne.

Exemple :

```
String texte = "texte";
int longueur = texte.length();
```

10.6. La modification de la casse d'une chaîne

Les méthodes Java **toUpperCase()** et **toLowerCase()** permettent respectivement d'obtenir une chaîne tout en majuscule ou tout en minuscule.

Exemple :

```
String texte = "texte";
String textemaj = texte.toUpperCase();
```

11. Les exceptions en Java

La cause de mauvais fonctionnement d'un programme peut être de deux types d'erreurs : une erreur de programmation ou une erreur pendant l'exécution du programme. Si le programmeur commet une erreur de programmation, le programme ne se compilera généralement pas, mais il existe des erreurs de programmation qui surviennent après la compilation, à l'appel de la machine virtuelle par **java.exe**(pour le jdk1.2 de sun). Toutes ces erreurs peuvent et doivent être corrigées par le programmeur. Dans les deux cas précédents, le programme ne démarre pas. Le second type d'erreur survient pendant l'exécution du programme. Ce sont en quelques sortes des "bugs". On distingue deux catégories : les erreurs de machine virtuelle (on ne peut rien y faire), et les **exceptions qu'on peut gérer**. Ces exceptions peuvent se produire dans plusieurs cas, elles ont cependant plus de chance de survenir lors d'un transfert de données. Par exemple, si un

programme essaie d'écrire sur un disque plein ou protégé en écriture, une exception de type IOException (input/output ou entrée/sortie de données) sera générée. La gestion de cette exception permettra au programme de ne pas "se planter". On peut également spécifier ce que le programme doit faire en cas d'exceptions.

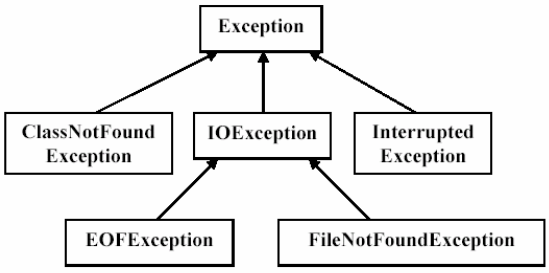
try…catch

try…catch peut être expliqué comme : « essaye ce bout de code (try), si une exception survient, attrape-la (catch) et exécute le code de remplacement (s'il y en a un) »

```
try {
// Code susceptible de produire des exceptions
}
catch (Exception e){
// Code de remplacement, se limite souvent à
System.out.println("Une
// erreur : " + e) ;
}
```

Exception : est la superclasse de toutes les exceptions, elle capte toutes les exceptions. Elle fonctionne dans tous les cas, cependant, *il est préférable de bien canaliser l'exception en spécifiant le type de l'exception le plus précis possible*. On doit spécifier un nom à cette exception créée, ici c'est e, on choisit généralement e ou ex. on peut en raison de la portée de variables spécifier le même nom pour toutes les exceptions (ce nom n'est reconnu qu'à l'intérieur de catch).

Quelques exceptions contrôlées du JDK



Finally : La clause *finally* permet d'exécuter le code qu'elle renferme quoiqu'il arrive. Qu'il y ait une exception ou pas, les instructions de la clause finally sont exécutées. Elle fonctionne également avec try.

```
try {
// instructions à essayer à réaliser
return ;
}
finally{
// instructions à réaliser absolument
}
return ;
```

Déclarer des méthodes susceptibles de générer des exceptions

On utilise la clause **throws** dans la déclaration de méthode. Exemples :

```
Public boolean MaMethode (xxx) throws UneException {…}
Public boolean MaMethode (xxx) throws UneException, UneAutreException, AutreException {…}
Public void MaMethode () throws IOException {…}
```

Générer des exceptions

Ceci sert à faire croire au programme qu'une exception d'un certain type est apparue.

```
NotInServiceException() nis=new NotInServiceException("Exception : DataBase out of use") ;
throw nis ;
```

Notons qu'il s'agit de throw et non pas de throws.

Créer des exceptions

Dans des programmes complexes, il se peut que les exceptions standard de java ne soient pas suffisantes et qu'il y aura par conséquent besoin de créer nos propres définitions. Dans ce cas, les exceptions devront hériter d'une exception plus haute dans la hiérarchie.

```
public class SunSpotException extends Exception {
public SunSpotException () {}
```

```
public SunSpotException(string msg) {
super(msg) ;
}
}
```

Résumons :

- Les exceptions sont des instances de classes dérivant de **java.lang.Exception**
- La levée d'une exception provoque une remontée dans l'appel des méthodes jusqu'à ce qu'un **bloc catch** acceptant cette exception soit trouvé. Si aucun bloc catch n'est trouvé, l'exception est capturée par l'interpréteur et le programme s'arrête.
- L'appel à une méthode pouvant lever une exception doit :
  - soit être contenu dans un bloc **try/catch**
  - soit être situé dans une méthode propageant (throws) cette classe d'exception
- Un bloc (optionnel) **finally** peut-être posé à la suite des **catch**. Son contenu est exécuté après un **catch** ou après un **break**, un **continue** ou un **return** dans le bloc **try**.

12. Les unités de compilation

- Il est préférable (et parfois obligatoire (dans le cas des classes publiques)) de mettre les définitions des classes dans des fichiers séparés ;
- Le code source d'une classe est appelé *unité de compilation* ;
- Il est recommandé (mais pas imposé) de ne mettre qu'une classe par unité de compilation ;
- L'unité de compilation (le fichier) doit avoir le même nom que la classe qu'elle contienne.

13. Les packages

13.1. La définition d'un package

En java, il existe un moyen de regrouper des classe voisines ou qui couvrent un même domaine : ce sont les packages. Pour réaliser un package, on écrit un nombre quelconque de classes dans plusieurs fichiers d'un même répertoire et au début de chaque fichier on met la directive ci-dessous où nomPackage doit être identique au nom du répertoire :

package nomPackage ;

La hiérarchie d'un package se retrouve dans l'arborescence du disque dur puisque chaque package est dans un répertoire nommé du nom du package.

Remarque : Il est préférable de laisser les fichiers source .java avec les fichiers compilés .class

D'une façon générale, l'instruction package associe toutes les classes qui sont définies dans un fichier source à un même package.

Le mot clé package doit être la première instruction dans un fichier source et il ne doit être présent qu'une seule fois dans le fichier source (une classe ne peut pas appartenir à plusieurs packages).

13.2. L'utilisation d'un package

Pour utiliser ensuite le package ainsi créé, on l'importe dans le fichier :

import nomPackage.*;

Pour importer un package, il y a trois méthodes si le chemin de recherche est correctement renseigné :

Exemple	Rôle
`import nomPackage;`	les classes ne peuvent pas être simplement désignées par leur nom et il faut aussi préciser le nom du package
`import nomPackage.*;`	toutes les classes du package sont importées
`import nomPackage.nomClasse;`	appel à une seule classe : l'avantage de cette notation est de réduire le temps de compilation

Attention : l'astérisque n'importe pas les sous paquetages. Par exemple, il n'est pas possible d'écrire import java.*.

Il est possible d'appeler une méthode d'un package sans inclure ce dernier dans l'application en précisant son nom complet :

nomPackage.nomClasse.nomméthode(arg1, arg2 ...)

Il existe plusieurs types de packages : le package par défaut (identifié par le point qui représente le répertoire courant et permet de localiser les classes qui ne sont pas associées à un package particulier), les packages standards qui sont empaquetés dans le fichier classes.zip et les packages personnels.

Le compilateur implémente automatiquement une commande import lors de la compilation d'un programme Java même si elle ne figure pas explicitement au début du programme :

import java.lang.*; Ce package contient entre autre les classes de base de tous les objets java dont la classe Object.

Un package par défaut est systématiquement attribué par le compilateur aux classes qui sont définies sans déclarer explicitement une appartenance à un package. Ce package par défaut correspond au répertoire courant qui est le répertoire de travail.

13.3. Quelques packages prédéfinis

- Le package **math** permet d'utiliser les fonctions mathématiques ;
- Dans **util** se trouve du vrac : Dictionnary, Hashtable, Properties pour classer des objets, la Date, Random, les structures de gestion d'objets : Vector, Stack, Enumeration ;
- Dans **io** tout ce qu'il faut pour manipuler les entrées / sorties, les cascader, ...
- **sql** contient les classes et méthodes pour interfacer vos applications avec des bases de données (interface JDBC).

13.4. La collision de classes.

Deux classes entrent en collision lorsqu'elles portent le même nom mais qu'elles sont définies dans des packages différents. Dans ce cas, il faut qualifier explicitement le nom de la classe avec le nom complet du package.

13.5. Les packages et l'environnement système

Les classes Java sont importées par le compilateur (au moment de la compilation) et par la machine virtuelle (au moment de l'exécution). Les techniques de chargement des classes varient en fonction de l'implémentation de la machine virtuelle. Dans la plupart des cas, une variable d'environnement CLASSPATH référence tous les répertoires qui hébergent des packages susceptibles d'être importés.

Exemple sous Windows :

CLASSPATH = .;C:\Java\JDK\Lib\classes.zip; C:\rea_java\package

L'importation des packages ne fonctionne que si le chemin de recherche spécifié dans une variable particulière pointe sur les packages, sinon le nom du package devra refléter la structure du répertoire où il se trouve. Pour déterminer l'endroit où se trouvent les fichiers .class à importer, le compilateur utilise une variable d'environnement dénommée CLASSPATH. Le compilateur peut lire les fichiers .class comme des fichiers indépendants ou comme des fichiers ZIP dans lesquels les classes sont réunies et compressées.

13.6. Exemple

```
// réptravail/classes/graph/2D/Circle.java
package graph.2D;
```

```
public class Circle
{ ... }
// réptravail/classes/graph/3D/Sphere.java
package graph.3D;
public class Sphere
{ ... }
// réptravail/classes/paintShop/MainClass.java
package paintShop;
import graph.2D.*;
public class MainClass
{
public static void main(String[] args) {
graph.2D.Circle c1 = new graph.2D.Circle(50)
Circle c2 = new Circle(70);
graph.3D.Sphere s1 = new graph.3D.Sphere(100);
Sphere s2 = new Sphere(40); // error: class
paintShop.Sphere not found
}
```