Cours

Cours essentiel de Java

Rédigé par

David ROUMANET
Professeur BTS SIO

Changement

Date	Révision

Sommaire

Nomenclature :

- **Assimiler** : cours pur. Explication théorique et détaillée (globalement supérieur à 4 pages).

- **Décoder** : fiche de cours, généralement inférieure à 5 pages.
- **Découvrir** : Travaux dirigés. Faisable sans matériel.
- **Explorer** : Travaux pratiques. Nécessite du matériel ou des logiciels.
- **Mission** : Projet encadré ou partie d'un projet.
- **Voyager** : Projet en autonomie totale. Environnement ouvert : Vous êtes le capitaine !

A Syntaxe Java

A.1 Vue générale "hello world"

Le premier code de base en Java se présente comme suit :

```
/* voici un exemple de code */
public class HelloWorld {
    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }
}
```

Pour les exemples qui suivront, il faudra placer les instructions à la place de la 4e et 5e ligne (lignes orange)

A.2 Syntaxe d'écriture

Java utilise une syntaxe simple mais très codifié :

- Les instructions se terminent par un point-virgule (;)
- Les noms de classes commencent par une majuscule
- Les méthodes et variables ne commencent pas par une majuscule
- Java utilise la notation "camel" à l'exception des règles sur la première lettre. Ainsi, une méthode (fonction) pour colorier un rectangle s'écrira "colorierRectangle". *Seules les classes s'écrivent avec la première lettre en majuscule*
- Les conditions sont encadrées par des parenthèses ().
- Les blocs d'instructions sont encadrés par des crochets {}
- Un commentaire commence par deux 'slash' //
- Un bloc de commentaires (plusieurs lignes) s'écrit entre /* et */
- Un bloc de commentaires pour Javadoc s'écrit entre /** et */
- Les crochets carrés sont utilisés pour les tableaux [ ]
- Le point (.) est généralement utilisé pour lier les méthodes (fonctions) aux variables

B Variables et types

En Java, les variables doivent être typées : cela signifie qu'elles ne pourront contenir que des informations du type qu'elles ont.

B.1 Les types primitifs

Seules les données numériques sont des types primitifs, avec deux exceptions : le caractère et le booléen.

Les types primitifs sont déclarés avec un mot-clé en minuscule.

Type	Représente	Plage	Valeur par défaut
bool	bololéen	True ou False	False
byte	8 bits ~~non~~ signés	-128 à +127	0
char	16 bits Unicode	U +0000 à U +FFFF	'\0'
~~decimal~~	~~128 bits~~	~~-7.9x10^{28} à +7.9x10^{28}~~	~~0.0M~~
double	64 bits	+/- 5x10^{-324} à +1.7*10^{308}	0.0D
float	32 bits	-3.4 x 10^{38} à +3.4 x 10^{38}	0.0F
int	32 bits signés	-2147483648 à +2147483647	0
long	64 bits signés	+/- 9223372036854775808 environ	0L
sbyte	8 bits signés	-128 à +127 (0x7F)	0
short	16 bits signés	-32768 à +32767 (0x7FFF)	0
~~uint~~	~~32 bits non signés~~	~~0 à 4294967295 (0xFFFFFFFF)~~	~~0~~
~~ulong~~	~~64 bits non signés~~	~~0 à 18446744073709551615~~	~~0~~
~~ushort~~	~~16 bits non singés~~	~~0 à 65535~~	~~0~~

Les types qui sont rayés n'existent pas en Java mais sont présents en C#.

B.1.1 Déclaration

```
int unEntier;
char uneLettre;
```

B.1.2 affectation

```
unEntier = 25;
uneLettre = 'c';
```

Il est possible de faire la déclaration et l'affectation sur la même ligne :

```
double unDecimal = 2.5;
```

B.2 Les classes conteneurs

Pour manipuler plus facilement les types primitifs ou permettre l'utilisation de chaînes de caractères, Java propose les classes conteneurs (ou classes références, en anglais "wrapper classes"). On les reconnaît par la présence d'une majuscule dans le type : cela indique que le type est un objet.

Les classes conteneurs usuelles sont :

- Byte, Short, Integer, Long
- Float, Double
- Character, Boolean

B.2.1 Déclaration

```
Integer unEntier;
String unMot;
Boolean monDrapeau;
```

B.2.2 affectation

```
unEntier = 25;
unMot = "Bonjour le monde";
monDrapeau = true;   // ne peut prendre que true ou false
```

En effet, certaines méthodes ne permettent que l'utilisation d'objet (comme les méthodes de la classe ArrayList). Il était nécessaire d'encapsuler les types primitifs dans un objet. Par exemple :

```
int myInt = 5 ;
list.add(new Integer(myInt)) ;   // ce mécanisme de boxing est relativement lourd. Les classes conteneurs facilitent tout.
```

B.3 Les constantes

Pour déclarer n'importe quelle variable (on dit aussi attribut), il suffit d'ajouter le mot-clé final d'avant la déclaration. Cette variable n'est alors plus modifiable ailleurs.

```
final double pi = 3.14159;
```

B.4 Portée de variables

La portée est une notion qui signifie simplement que plus un attribut est déclaré dans un bloc supérieur, plus il est visible. Ce qui entraîne qu'un attribut déclaré dans une boucle, n'est visible que dans cette boucle. Pour rappel, un bloc est généralement représenté entre { }.

```
for (int x=0; x < 10; x++) {
    for (int y=0; y < 10; y++) {
        System.out.println(x+"-"+y);
        int z = x*y;
    }
    System.out.println("Cette ligne entraine une erreur. Z = "+z);
}
```

B.5 Les conversions

L'interpréteur Java refuse les conversions qui risquent de faire perdre de l'information.

B.5.1 Conversions implicites

Ainsi, l'opération suivante ne pose aucun problème, car un entier est inclus dans les décimaux :

```
double monNombre = 0;
int monAutreNombre = 5;
monNombre = monAutreNombre;
```

B.5.2 Conversions explicites

En revanche, l'inverse nécessite que le programmeur 'confirme' de manière explicite le risque de perte d'information, par l'ajout du type souhaité entre parenthèse avant le nom de variable :

```
double monNombre = 7.35;
int monAutreNombre = 5;
monAutreNombre = (int)monNombre;   // on appelle cela un cast
```

*Pour synthétiser, un **byte** est inclus dans un **short** qui est inclus dans un **int** qui est inclus dans un **long** qui est inclus dans un **float** qui est inclus dans un **double**.un **char** sera inclus uniquement à partir d'un **int** car il n'a pas de bit de signe.*

B.5.3 Conversions de chaîne vers nombre

La conversion d'une chaîne de caractères vers un nombre est possible par l'utilisation de méthodes existant dans les classes correspondantes. Il faut donc utiliser les classes conteneurs vues précédemment.

B.5.3.a La méthode ‘parse()’

```
String maChaine = "5.68";
Double d;
d = Double.parseDouble(maChaine);
```

B.5.3.b La méthode ‘Convert.’

```
String maChaine="5.68";
Double d;
d = Convert.ToDouble(maChaine);
```

B.5.4 Conversions de nombre vers chaîne

L’opération permettant de convertir un nombre dans une chaîne existe aussi, en utilisant là encore des méthodes dans les classes conteneurs.

B.5.4.a La méthode ‘valueOf’

```
String maChaine = "";
Double d = 5.68;
maChaine = maChaine.valueOf(d);
```

B.5.4.b La méthode ‘toString’

```
String maChaine = "";
Double d = 5.68;
maChaine = d.toString();
```

B.5.4.c La méthode bizarre

```
String maChaine = "";
Double d = 5.68;
maChaine = "" + d;
```

C Les opérations

C.1 Les opérations arithmétiques

C.1.1 Opérations simples

Les opérations d'addition, soustraction, multiplication, soustraction et modulo sont supportées :

```
Double d = 5.68;
Integer e = 120;
d = d * 1.20 / 5 + 3.14 - 6.222;
e = e % 100;         // retourne 20
```

Les priorités sont respectées (multiplication et division passent avant l'addition et la soustraction)

C.1.2 Opérations complexes

Les opérations trigonométriques, la génération de nombres aléatoires, l'utilisation de racines carrés ou puissances nécessitent les méthodes contenues dans la classe Math.

```
Double a, b, c, d;
a = Math.sin(3.14/2);
b = Math.pow(5,2);           // 5² ou 5 à la puissance 2
c = Math.random() * 49 + 1; // random() choisit un nombre entre 0 et
0.9999...
d = Math.sqrt(c);            // √c (racine, en anglais, square)
```

C.1.3 Opérations particulières

Quelques opérations sont pratiques et doivent êtres connues.

- Incrémentation : d**++** (équivalent à d = d + 1)
- Décrémentation : d**--** (équivalent à d = d - 1)
- Cumul : total**+=** d (équivalent à total = total + d)

C.2 Les opérations binaires

Il s'agit d'opérations sur les bits des variables : ET, OU, NON, << et >>, etc.

```
int a, b, c, d, e, f;
a = 0b1100 & 0b1010;   // ET : renvoie 1000
b = 0b1100 | 0b1010;   // OU : renvoie 1110
c = 0b1100 ^ 0b1010;   // OU exclusif : renvoie 0110
```

```
d = !0xF0;              // NON : renvoie 0F
e = 0x0F << 8;          // Décalage par 8 à gauche : renvoie F0
```

Attention à ne pas confondre opérations binaires et comparateurs (&&, ||, !=, ==, ≥=, <=...)

C.3 Les opérations de chaînes

Ce sont les opérations disponibles pour le traitement de chaînes de caractères ou de caractères : il faut cependant se souvenir que Java travaille en Unicode ! La taille d'un caractère n'est donc pas un octet comme en ASCII.

- Méthodes toLowerCase() et toUpperCase() changent la casse de la chaîne
- Méthode length() renvoie la taille de la chaîne (en nombre de caractère)
- Méthodes charAt(position) et substring(pDepart, pFin) affiche un ou plusieurs caractères
- Méthode replace(chOrigine,chFinale) remplace une chaîne par une autre
- Méthode de recherche contains(strRecherchee) pour trouver une chaîne dans une autre
- Méthode split(strSepare) pour séparer une chaîne en plusieurs chaînes

Quelques exemples d'utilisations sont données ci-dessous :

```
String maChaine = "Bonjour";
String monPrenom = "Coco";
char maLettre = ' ';

System.out.println(maChaine.toLowerCase());     // affiche 'bonjour'
System.out.println(maChaine.toUpperCase());     // affiche 'BONJOUR'
System.out.println(maChaine.length());          // affiche 7
System.out.println(maChaine.charAt(4));         // affiche ''
String messageCool = maChaine+maLettre+monPrenom; contient 'Bonjour Coco'
String message2 = maChaine.concat(monPrenom);   // contient BonjourCoco
```

Voir https://docs.oracle.com/javase/6/docs/api/java/lang/String.html

C.3.1 Formatage des affichages

Si vous devez écrire dans la console, des nombres avec un formatage particulier, la méthode printf() est très puissante. L'écriture propose d'indiquer avec le préfixe `%` et le suffixe '`lf`' de placer une valeur

numérique. Entre les deux, on décrit combien de chiffres avant et après la virgule. Enfin, les variables sont ajoutées, mais en dehors de la chaîne. Un exemple :

```
double a = 5.6d;
double b = 2d;
double mult = a * b;
String nom = "Chuck NORRIS";
int naissance = 1940;
System.out.println("%s est né en %d", nom, naissance);
System.out.printf("%3.2lf fois %3.2lf égal %3.2lf\n", a, b , mult);
```

affichera : 005.60 fois 002.00 égal 011.20 (soit 3 chiffres avant le point et deux chiffres après).

D Les tableaux

Un tableau est un ensemble de variables de même type dans une quantité définie.

Chaque case du tableau est numérotée, en partant de 0 : un tableau contenant 10 valeurs aura les numéros de cases de 0 à 9 (on parle de l'indice de cellule).

D.1.1 déclaration

```
int monTableau[] = {1995, 1996, 1997, 1998, 2000, 2001};
```

0	1	2	3	4	5
1995	1996	1997	1998	2000	2001

Une autre déclaration (pour un tableau de chaîne vide par exemple) est la suivante :

```
String monTableau[] = new String[9];
String[] autreTableau = new String[9];
```

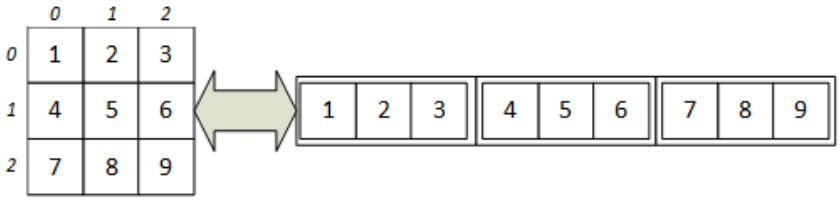
Ces deux tableaux contiendront donc 10 chaînes de caractères (l'index commence à 0 !).

```
int tailleTableau = autreTableau.length();
```

Enfin, il est possible d'utiliser des tableaux à plusieurs dimensions (comme pour la bataille navale) :

```
char[][] monTableau2D = new char[9][9];
int[][] autreTableau2D = {{1,2,3},{4,5,6},{7,8,9}};
```

Le premier tableau est une grille de 10x10 éléments tandis que le deuxième tableau contient 3x3 éléments. Il faut donc voir les tableaux à dimensions multiples comme des tableaux de tableau.



D.1.2 affectation

Dans l'exemple ci-dessus, la case autreTableau2D[0] contient le tableau {1, 2, 3}

Il existe une classe spéciale (Arrays, dans java.util.*) qui offre des méthodes de gestion des tableaux, notamment le tri (sort), le remplissage (fill), l'égalité (equals) et la taille (length).

```
AutreTableau2D[0][1] = 666;
System.out.println("la case coordonnées 0,1 a pour valeur
"+autreTableau[0][1]);
```

E Les entrées-sorties standards

En Java, les entrées et sorties concernent autant le mode console, que les fichiers et connexions. Seule la gestion des interfaces graphiques est différente (utilise les notions d'événements). Les notions utiles sont :

- Stream : un flux qui ne s'arrête pas, comme une émission en direct à la télévision
- Buffer : une zone mémoire pour ne pas perdre d'information (notamment si le flux n'est pas lu suffisamment souvent).

E.1 Mode console

Pour afficher et récupérer du texte, les méthodes de la classe System et java.util sont nécessaires.

E.1.1 Affichage

Voici comment comprendre le code d'affichage suivant :

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

- **System** est la classe utilitaire (aussi appelé classe technique, car commune à l'ensemble des applications par défaut).
- **out** est l'objet dans cette classe, qui gère les sorties : ici, l'objet **out** est l'écran principal de la console. Il existe un objet **err** qui permet de noter les erreurs. L'objet **in** est utilisé pour récupérer les flux du clavier.
- **println()** est la méthode qui s'applique sur l'objet out : cette méthode envoie à cet objet les données comprises entre ses parenthèses. Le 'ln' de println signifie qu'il y aura un saut de ligne. D'autres méthodes existent, comme print(), printf(), etc.

Exemple :

```
System.out.print("Bonjour ");
System.out.println("Coco !");      // affichera 'Bonjour Coco !' sur la
même ligne
System.err.println("Un problème"); // affichera 'Un problème' en rouge
```

E.1.2 Saisie

La console est ouverte par défaut, ce n'est pas le cas du flux en provenance du clavier. Il faut initialiser le flux (une seule fois) puis utiliser les méthodes de lecture nextline(), nextint(), etc.

```
Import java.util.*
Scanner clavier = new Scanner(System.in);     // à faire une seule fois
String monNom = "";
```

```
System.out.println("Entrez votre nom");
monNom = clavier.nextline();
```

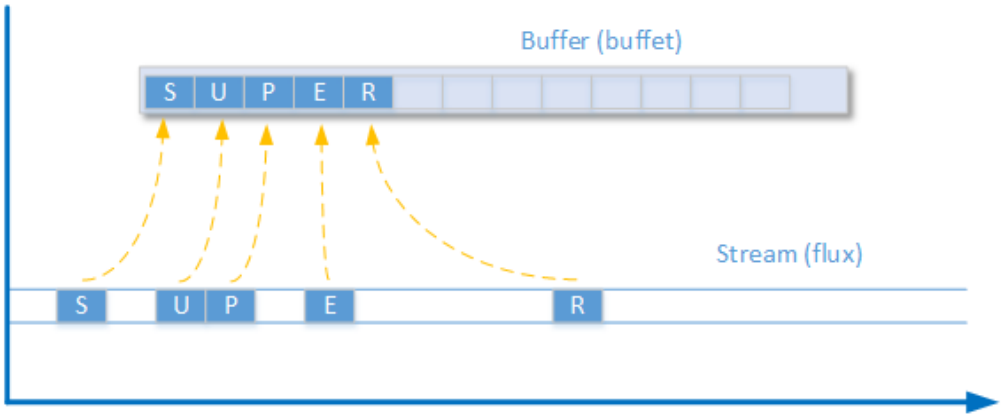
Il n'existe pas de lecture de clavier caractère par caractère qui ne serait pas bloquant, à part d'utiliser l'astuce suivante (basé sur la méthode available() de la classe technique java.io) :

```
import java.io.*;
InputStreamReader fluxClavier = new InputStreamReader(System.in);
 BufferedReader memoireClavier = new BufferedReader(fluxClavier);
for (;;) {
    if (memoireClavier.available() > 0) {
        // faire quelque chose car il y a des caractères dans la mémoire
    }
}
```

E.2 Fichiers

L'astuce ci-dessus nous amène à nous intéresser à la lecture et l'écriture de fichier.

Comme évoqué précédemment, pour Java, la plupart des périphériques sont considérés comme des flux entrants et sortants. Cela signifie que si le flux n'est pas mémorisé, les données transmises sont perdues (un peu comme une chaîne en direct : si vous êtes absent de l'écran, les informations sont perdues).



Comme la plupart des périphériques acceptent les entrées et les sorties, cela multiplie par deux les flux et buffets à déclarer.

Attention, la lecture d'un flux vide entraîne la génération d'une exception qui bloque l'exécution du programme.

Il existe plusieurs classes techniques pour gérer les flux, un schéma est présent en annexe.

Voici donc un exemple de code pour lire un fichier texte, à copier tel quel pour le moment :

```
import java.io.*;

 FileReader fluxFichier = new FileReader(new File("test.txt"));
 BufferedReader tamponFichier = new BufferedReader(fluxFichier);

 String ligne = null;
 while ((ligne = tamponFichier.readLine()) != null) {
      System.out.println(ligne);   // ou remplacer par une action
 }
```

L'écriture d'un fichier n'est pas plus difficile :

```
import java.io.* ;

BufferedWriter bw = new BufferedWriter(new FileWriter(new
File("C:/new.txt"), true));
bw.write("Bonjour Coco !");
bw.newLine();
bw.close();
```

F Boucles et conditions

Comme dans d'autres langages, Java propose un ensemble d'instructions pour créer des boucles et permettre des choix à l'aide de conditions.

F.1 Boucles

Il existe 3 boucles connues (répétition, test avant de commencer la boucle et test en fin de boucle).

F.1.1 Boucle for

La boucle for permet un comptage ou une énumération : elle s'écrit avec 3 paramètres séparés par un point-virgule.

- Déclaration et initialisation de la variable de comptage
- Condition de sortie de boucle (tant que la condition n'est pas remplie, on reste)
- Choix de l'incrément (on peut aller de 2 en 2 ou bien partir dans le sens inverse en décrémentant la variable)

```
for (int t=0 ; t < 7 ; t++) {
    System.out.println("boucle N°"+t) ;
}
```

On peut aussi créer une boucle infinie comme ceci (*sortie possible avec une instruction 'break'*) :

```
for (;;) {
    System.out.println("ne s'arrête jamais... ahahah !") ;
}
```

Enfin, lorsqu'on a un tableau ou une liste d'objet, on peut la parcourir automatiquement

```
for (int compteur : monTableau) {
    System.out.println("index "+compteur+" du tableau est
"+monTableau[compteur]);
}
```

F.1.2 Boucle while

Cette boucle classique propose **un test en début de boucle**. La boucle s'exécute tant que la condition reste vraie. Si la condition est fausse dès le début, les instructions du bloc ne sont pas exécutées.

```
int t=0;
while (t < 10) {
   // action à faire...
   t++;
}
```

F.1.3 Boucle Do… while

La condition se trouve à la fin de la boucle. Les instructions seront exécutées au moins une fois. Cependant, contrairement à une boucle repeat… until(), la sortie de la boucle se fait lorsque la condition est fausse (comme dans la boucle while classique).

```
do {
    nbreHasard = (int) (Math.random() * 49 + 1);
} while (grilleLoto.contains(nbreHasard));
```

Il est possible de sortir d'une boucle sans que la condition ne soit modifiée, par l'utilisation de l'instruction break !

```
int t=0;
while (t < 10) {
   if (t = 5) {
       break;
   }
   t++;
}
```

F.2 Conditions

Il y a deux types de conditions :

- les IF, ELSE IF, ELSE
- les SWITCH, CASE

F.2.1 Conditions IF - ELSE

Le contenu du bloc est exécuté si l'ensemble de la condition est vrai : s'il y a plusieurs conditions (dans la même parenthèse), elles doivent toutes être vraies. C'est le modèle le plus connu.

```
int testscore = 76;
char grade;
boolean work = true;

if (testscore >= 90 && work) {
  grade = 'A';
} else if (testscore >= 80) {
    grade = 'B';
} else if (testscore >= 70) {
  grade = 'C';
} else if (testscore >= 60) {
   grade = 'D';
} else {
grade = 'F';
}
System.out.println("Grade = " + grade);
```

Il existe quelques cas dans lesquels on peut créer un test, sur une seule ligne : c'est le cas lorsqu'il n'y a qu'une instruction. Dans ce cas, on peut omettre les accolades.

```
if (ajustement == 99)  maReponse = 'Décalage -1%';
if (ajustement == 100) maReponse = 'Aucun décalage';
if (ajustement == 101) maReponse = 'Décalage +1%';
```

F.2.2 Conditions SWITCH - CASE

Un peu moins visible, ce système de conditions est très structuré :

```
a = 1;
switch(a) {
   case 0 :
       System.out.println("nul"); break;
   case 1 :
       System.out.println("un"); break;
   case 3 :
       System.out.println("trois"); break;
}
```

Il y a cependant un impératif : ajouter une instruction 'break' avant chaque nouveau bloc 'case', sinon dès qu'une condition est réalisée, tous les blocs suivants sont exécutés.

Essayez le programme précédent en enlevant les mots-clés 'break'. Le résultat est-il cohérent ?

D'autre part, en Java, la gestion des plages (intervalles) n'est pas intuitive ! Le code le plus proche est le suivant :

```
switch (num) {
  case 1: case 2: case 3: case 4: case 5:
     System.Out.Println("testing case 1 to 5");
     break;
  case 6: case 7: case 8: case 9: case 10:
     System.Out.Println("testing case 6 to 10");
     break;
  default:
     //
}
```

Les utilisateurs de Pascal et Purebasic seront surpris du manque de souplesse mais PHP et C# fonctionnent de la même manière que Java. Enfin, il n'y a pas de switch/case en Python.

G Les exceptions

JAVA fournit un moyen de contrôle des erreurs lors du déroulement des applications. Cette solution se faisant au détriment de la performance et nécessitant du code supplémentaire, il est fréquent que seules les parties sensibles du code soient protégées.

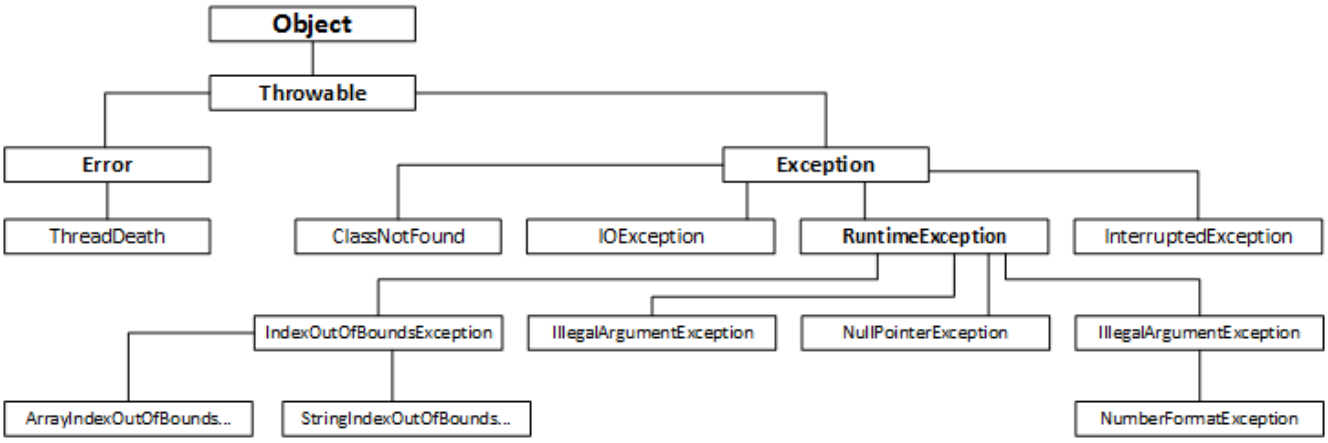
G.1 Arborescence des exceptions

Il existe plusieurs exceptions qui peuvent survenir par différentes situations :

- Lecture de données dans un fichier vide
- Lecture d'un fichier sur un média inexistant (retrait de clé USB par exemple)
- Division par zéro dans un calcul
- etc.

Une grande partie de ces erreurs sont habituelles et ne doivent pas bloquer le programme ou le planter. Java peut donc décider d'avertir le programme du problème, et si le programmeur n'a rien prévu, arrêter le programme.

Voici les classes des objets liés aux erreurs fréquentes (par exemple, la lecture d'un index de tableau au-delà de la taille maximale du tableau générera une exception "ArrayIndexOutOfBounds").



G.1.1 Exemple de code avec erreur

L'exemple le plus simple est la gestion d'une division par zéro.

```
public static void main(String[] args) {
   int a=20;
   int b=0;
   System.out.println(a+" divisé par "+b+" égal "+a/b);
}
```

Ce code entraînera l'erreur ArithmeticException :

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at net.roumanet.Main.main(Main.java:19)

Process finished with exit code 1
```

G.2 Gestion des exceptions rencontrées

La méthode est relativement simple et repose sur quelques instructions :

- try { … } : le bloc contenant le code contenant un risque d'exceptions
- catch { … } : un des blocs dédié au traitement des erreurs. Il peut y avoir plusieurs catch {}
- finally { … } : le bloc qui traite la sortie du code normal (try) ou des codes de traitements des erreurs (pour fermer un fichier par exemple).

G.2.1 Exemple de code avec gestion d'erreur

Dans le code précédent, il suffit d'ajouter try/catch sur l'erreur ArithmeticException pour pouvoir afficher un message d'erreur plutôt qu'arrêter l'application.

```
public static void main(String[] args) {
   // Préparation des attributs
   int a=20;
   int b=0;
   try {
      System.out.println(a + " divisé par " + b + " égal " + a / b);
   }
   catch (ArithmeticException e) {
      System.out.println("diviseur nul : division par zéro impossible");
   }
}
```

Ici, le programme ne traitera que cette erreur, mais il est possible de traiter chaque erreur (ou exception) différemment : c'est l'ordre des catch {…} qui déterminera le premier traitement.

H Les objets

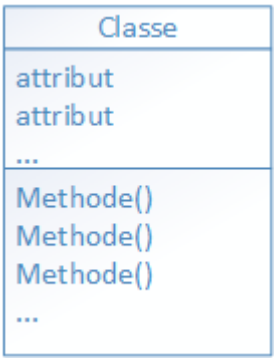
En Java, les objets s'utilisent un peu comme des "super" variables.

Un objet en Java, correspond à un ensemble de variables et des fonctions intégrées. Grâce à une notion importante appelé encapsulation, il est possible de n'accéder à ces variables qu'au travers de ces fonctions.

Ainsi, une classe est donc un bloc autonome, et les objets créés à l'aide de cette classe ne peuvent pas être mis en "pièces détachées".

H.1 Vocabulaire

Le langage orienté objet propose un vocabulaire à connaître impérativement. Ci-contre, la représentation symbolique d'une classe.



Une **classe** est constituée de **membres** :

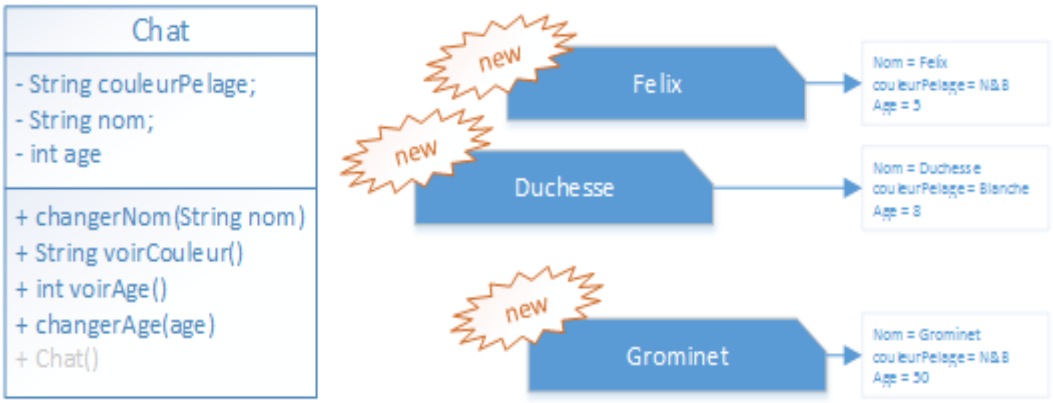
- les **attributs** : ce sont les variables de la classe. Sauf s'ils sont publics, les attributs ne sont pas visibles depuis une autre classe (étrangère).
- Les **méthodes** : ce sont les fonctions ou procédures de la classe. Elles aussi peuvent être publiques ou privées.

Une classe n'est qu'un patron, un plan d'objet : il faut construire les objets à partir de ce plan, en objet le terme utilisé : **instancier**.

Pour instancier une classe, on utilise une méthode de la classe particulière, qui porte le nom de la classe, et qu'on appelle un **constructeur**.

H.2 Représentation

Voici un exemple graphique :



La classe Chat contient les attributs privés couleurPelage, nom et age (un signe – les précède).

Elle contient les méthodes publiques (signe +) permettant de modifier ou lire les attributs : cela signifie qu'il n'est possible de modifier ces attributs que par ces méthodes, ce qui est très sécurisant.

H.2.1 instanciation

Pour créer le chat Félix, il suffit d'instancier la classe Chat et de stocker le résultat dans une "variable" de type Chat (ce n'est pas un entier, une chaîne… c'est un chat).

La syntaxe d'instanciation est la suivante :

```
Chat felix = new Chat();

felix.changerNom("Felix");
felix.changerAge(5);
```

le mot-clé **new** indique à Java de construire un objet, mais comme pour une maison, l'objet est initialement vide.

Notez qu'il n'est pas nécessaire de nommer l'objet avec le véritable nom du chat.

```
Chat chatRoumanet = new Chat() ;
chatRoumanet.changerNom("Pirouette") ;
…
```

A noter : si le constructeur de la classe n'est pas déclaré explicitement, Java va le créer automatiquement, avec tous les attributs par défaut.

Astuce : il est possible d'avoir plusieurs constructeurs, à la condition que le nombre d'arguments soit différents pour chaque constructeur et qu'il soit public (donc visible par tous).

Le mot-clé **this** permet de préciser à Java qu'il s'agit de l'attribut de l'objet en cours de création, permettant de ne pas mélanger avec le nom de l'argument transmis :

```
public Chat() {}

public Chat(String nom) {
    this.nom = nom;      //this.nom correspond à l'objet, nom à
l'argument méthode
}

public Chat(int age, String nomChoisi) {
    this.age = age;
    this.nom = nomChoisi ;
}
```

H.2.2 Accesseurs et mutateurs

Les méthodes qui permettent de lire et modifier les attributs privés dans la classe, sont appelées des **accesseurs** et **mutateurs**. En anglais, les termes pour…

- les accesseurs qui permettent de lire : les **getters**
- les mutateurs qui permettent de modifier : les **setters**

Dans la classe Chat, Les méthodes voirAge() est un accesseur et changerAge(int age) est un mutateur.

H.2.3 Portées et modificateurs

Plusieurs fois présenté pour certaines, les portées servent à protéger les attributs et les méthodes. Les 3 modificateurs importants pour le moment sont private, public et static.

Dans le détail,

H.2.3.a pour les classes

static	La classe ne peut pas être instanciée : elle est unique et définie une seule fois en mémoire.
abstract	La classe contient une ou des méthodes abstraites, qui n'ont pas de définition explicite. Une classe déclarée abstract ne peut pas être instanciée : il faut définir une classe qui hérite de cette classe et qui implémente les méthodes nécessaires.
final (sécurité)	La classe ne peut pas être modifiée, sa redéfinition grâce à l'héritage est interdite. Les classes déclarées final ne peuvent donc pas avoir de classes filles.

private (accessibilité)	La classe n'est accessible qu'à partir du fichier où elle est définie.
public (accessibilité)	La classe est accessible partout. (C'est la valeur par défaut, mais il faut la préciser pour que la classe soit accessible en dehors de son package.)

H.2.3.b Pour les méthodes

static	La méthode static déclarée statique ne peut agir que sur les attributs de la classe et pas sur les attributs des instances. La méthode main() est une méthode statique car il ne peut y avoir qu'un point de démarrage du programme.
Abstract	La méthode n'a pas de code. En cas d'héritage par une classe fille, celle-ci devra définir sa propre méthode.
private (accessibilité)	La méthode n'est accessible qu'à l'intérieur de la classe où elle est définie.
public (accessibilité)	La méthode est accessible partout.

H.2.3.c Pour les attributs

static	L'attribut est unique pour l'ensemble des instances. Il est généralement utilisé pour un comptage
final	L'attribut est une constante : il ne peut être modifié après sa première initialisation
private (accessibilité)	L'attribut n'est accessible qu'à l'intérieur de la classe où elle est définie.
Protected (accessibilité)	L'attribut est accessible par sa classe mais aussi les classes filles (héritage)
public (accessibilité)	L'attribut est accessible partout. Exemple : maVoiture.puissance = 215;

H.2.4 Analyse d'un programme

Le premier programme saisie (HelloWorld) peut donc maintenant être totalement analysé :

```
/* voici un exemple de code */
public class HelloWorld {
    public static void main(String[] args) {
        // Prints "Hello, World" to the terminal window.
        System.out.println("Hello, World");
    }
}
```

La classe publique HelloWorld n'a pas d'attribut.

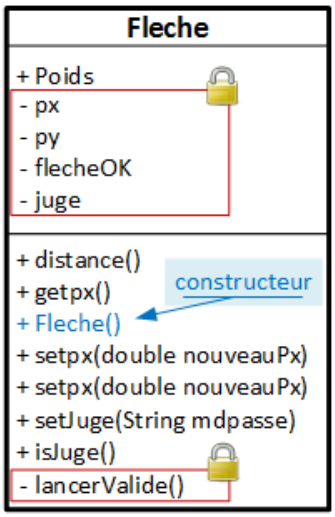
Elle ne contient qu'une méthode publique dont voici les arguments :

public	Il s'agit d'une méthode publique, utilisable depuis une autre classe
static	Cette méthode n'est pas instanciable : c'est toujours le cas pour main() qui est la méthode de lancement des applications (il ne peut en exister qu'une seule officielle par programme)
void	La méthode ne renvoie aucun argument (le type void signifie 'vide')
main	Le nom de la méthode
String[] args	La méthode accepte des arguments en entrée : un tableau de chaîne de caractères.

H.3 Résumé

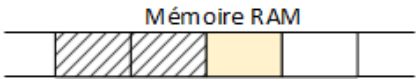
Création d'une classe

```
Public class Fleche {

   public int poids;
   private double px;
   –


   public distance() {
   }
   public Fleche() {
   }

   private lancerValide() {
   }
}
```

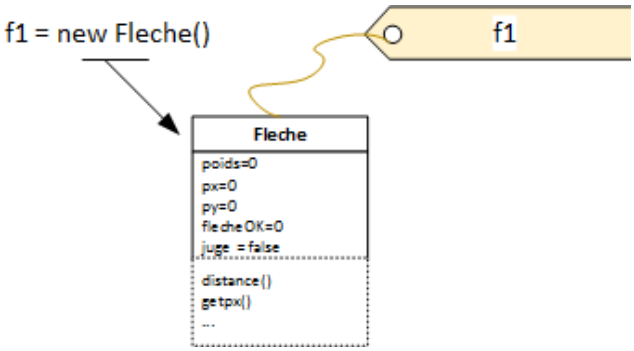


Déclaration d'un objet de la classe

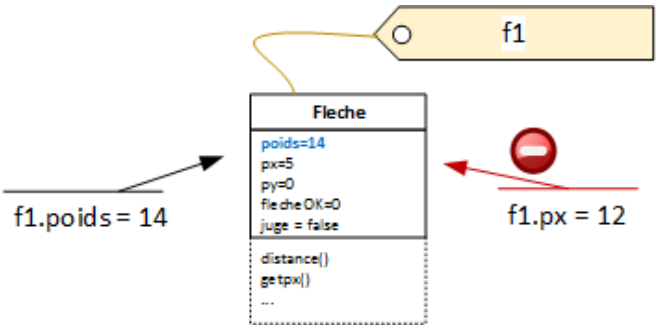
(il s'agit d'une réservation d'espace)



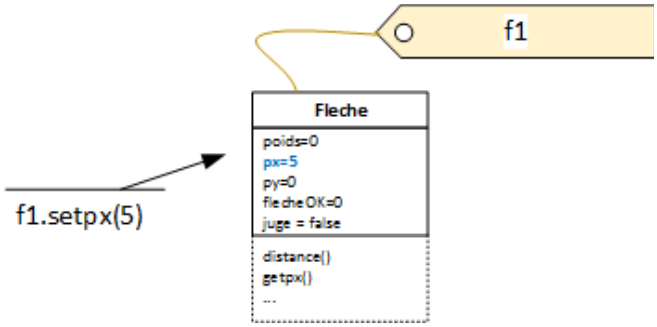
Instanciation d'un objet



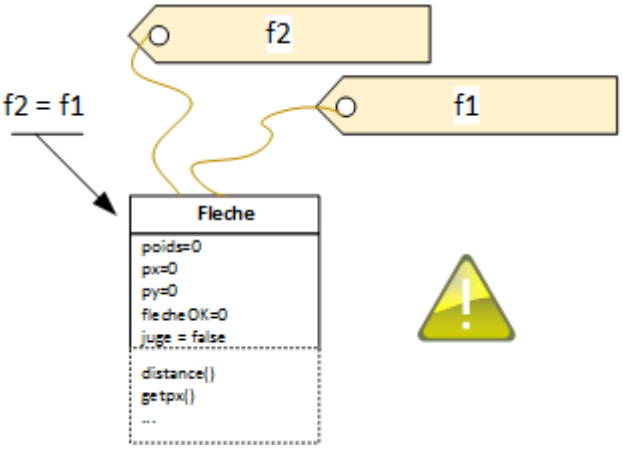
Modification d'un attribut public/privé



Modification d'un attribut privé (mutateur)



Copie de la référence

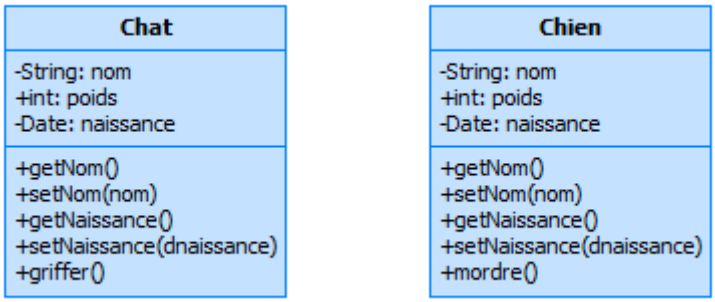


H.4 Héritage

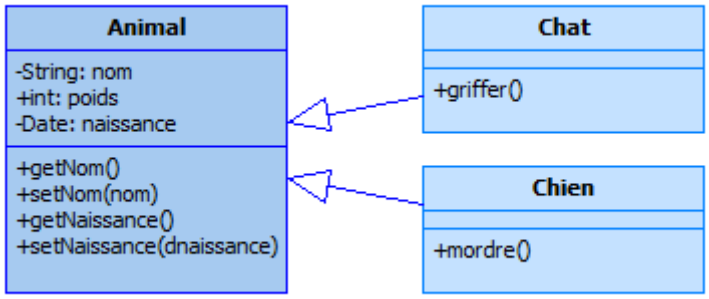
L'intérêt majeur de la programmation objet est probablement l'héritage : cette notion permet en effet de profiter des propriétés d'une classe existante et de pouvoir étendre ses capacités.

H.4.1 Exemple

Une clinique vétérinaire qui doit enregistrer des chats et des chiens pourrait avoir deux classes, comme dans l'image suivante :



Cependant, on constate que de nombreux éléments sont communs, hormis la particularité des chats pour griffer et celle des chiens pour mordre.
L'héritage permet de simplifier le développement en ne créant qu'une seule fois les membres (attributs et méthodes) communs et en créant les classes propres aux deux espèces.



H.4.2 Déclaration

La déclaration d'un héritage de classe se fait par l'utilisation du mot 'extends' :

```
public class Chat extends Animal {
    public void griffer() {
        System.out.println("Griffé !") ;
    }
}
public class Chien extends Animal {
    public void mordre() {
        System.out.println("Mordu !") ;
    }
}
```

Contrairement au C++, En Java (et même en C#) une classe ne peut hériter que d'une seule classe.

H.4.3 Utilisation

Une fois déclarée, la classe s'utilise normalement.

```
Chat monChat = new Chat();
Chien monChien = new Chien();
Animal maTortue = new Animal();
```

H.4.4 Exemple de codage (TD à reproduire)

Dans la pratique, il n'est pas possible de placer les classes filles dans le même fichier que la classe mère. Il faut donc créer un fichier par classe.

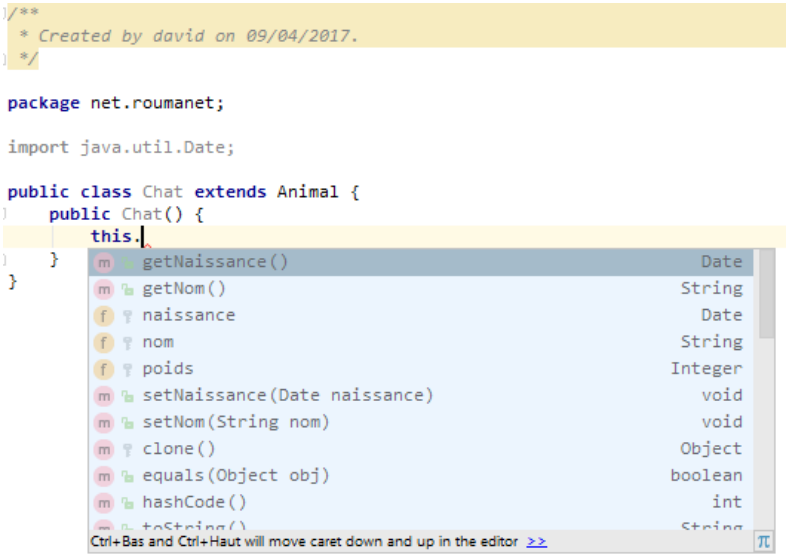
Voici la classe mère Animal.java (dans un projet existant, clic droit, New ► Java class) :

```
/**
 * Created by david on 09/04/2017.
 */

package net.roumanet;

import java.util.Calendar;
import java.util.Date;

public class Animal {

    protected String nom;
    protected Integer poids;
    protected Date naissance;

    // constructeur explicite pour créer l'objet et affecter les attributs en même
temps
    Animal(String nom, Integer poids, Date dnaissance) {
        this.nom = nom;
        this.poids = poids;
        this.naissance = dnaissance;
    }
    Animal() {
        this.nom = "(inconnu)";
        this.poids = 0;
        this.naissance = Calendar.getInstance().getTime();
    }
```

```
    // Astuce IntelliJ : placer le curseur sur un attribut, puis Alt+Ins pour générer
les getters/setters automatiquement
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
        this.nom = nom;
    }

    public Date getNaissance() {
        return naissance;
    }
    public void setNaissance(Date naissance) {
        this.naissance = naissance;
    }
}
```

Toujours dans le même paquetage (package), clic droit, New ► Java Class (classe Chat)

Il suffit d'écrire les quelques lignes ci-contre et de valider que l'IDE présente les attributs de la classe mère lorsque vous créez le constructeur de la classe.



Ici, naissance, nom et poids (ordre alphabétique) appartiennent bien à la classe Animal dans le fichier Animal.java.

```
/**
 * Created by david on 09/04/2017.
 */

package net.roumanet;

import java.util.Date;

public class Chat extends Animal {
    public Chat() {
        this.nom = "Minou inconnu";
```

```
    }
}
```

Enfin, éditer la méthode main() de votre programme comme suit :

```
/**
 * Created by david on 09/04/2017.
 */

package net.roumanet;

    public static void main(String[] args) throws IOException {
        // Préparation des attributs
        Chat leMinou = new Chat();
        System.out.println("Le chat ayant pour étiquette 'leMinou'
s'appelle "+leMinou.nom);
    }
}
```

Le résultat sera :

```
"C:\Program Files (x86)\Java\jdk1.8.0_121\bin\java" ...
Le chat ayant pour étiquette 'leMinou' s'appelle Minou inconnu

Process finished with exit code 0
```

H.4.5 Exercice

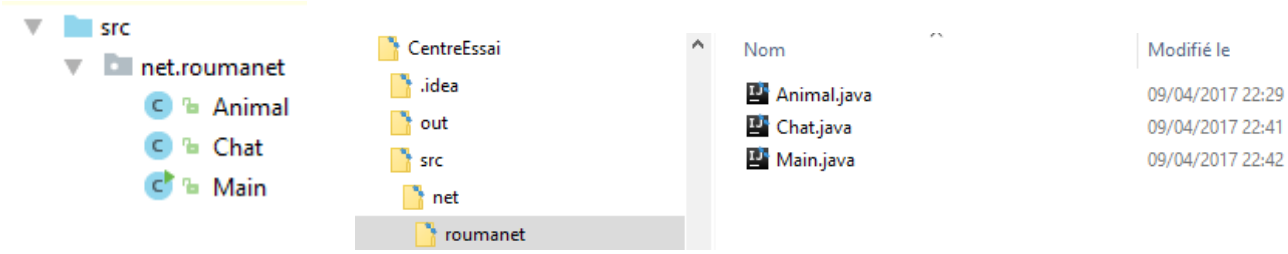
Finissez la classe Chat en ajoutant la méthode griffer(). Cette méthode contient un tirage au hasard entre 0 et 0.999 et affichera

"*le-nom-du-chat* a essayé de vous griffer" si le tirage est supérieur à 0.8, sinon

"*le-nom-du-chat* vous aime bien et ne vous griffe pas".

Créez également la classe Chien en y intégrant la méthode mordre() ayant un comportement similaire que la méthode griffer().

Pour validation auprès du professeur, envoyez seulement le contenu du répertoire 'src' dans un fichier ZIP.



↔

I Les classes abstraites et interfaces

J JavaFX

La création d'interfaces graphiques sous Java faisait appel aux Classes AWT et Swing.

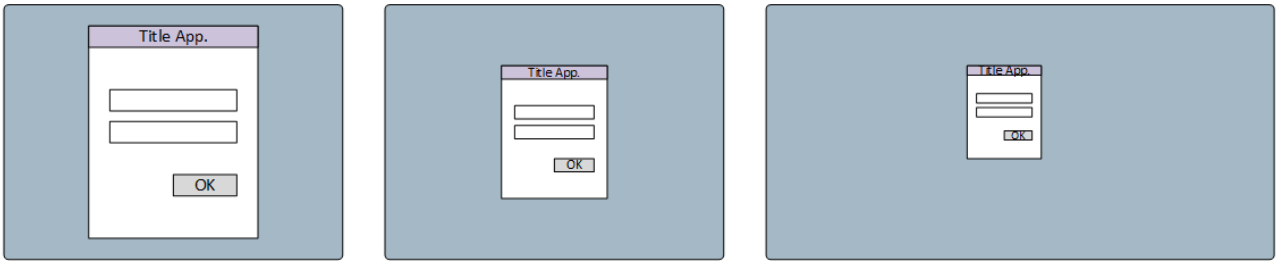
AWT proposait une apparence très typée qui ne s'intégrait pas forcément à l'apparence des applications sur l'OS.

Swing tentait de remédier à un certain nombre de difficultés, dont notamment les dispositions d'éléments dans la fenêtre.

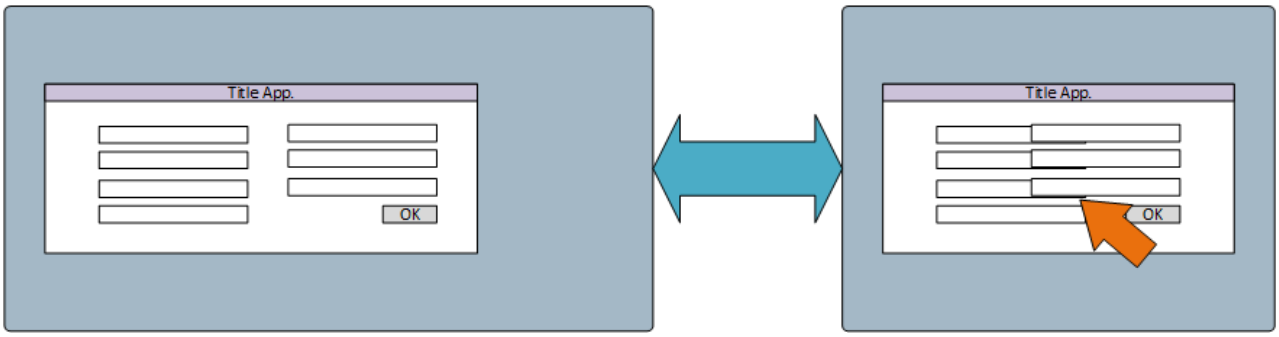
JavaFX apporte désormais une description standard pour les éléments à afficher, avec les avantages de CSS en termes de mise en forme.

J.1 Interface graphique

Dès que l'utilisateur est un humain, il faut alors proposer un mode de dialogue compréhensible et adapté à son environnement : certains utilisent un écran HD (1920 × 1080), d'autre se servent d'un écran de contrôle (par exemple 800 × 600). L'application peut fonctionner en plein écran ou en mode fenêtré… son aspect va donc changer.



Le risque principal est le redimensionnement de fenêtre : en pleine largeur sur un écran 16/10^{e} une application risque de cacher certains éléments en 4/3 ou les replacer avec un recouvrement.



J.2 AWT et Swing

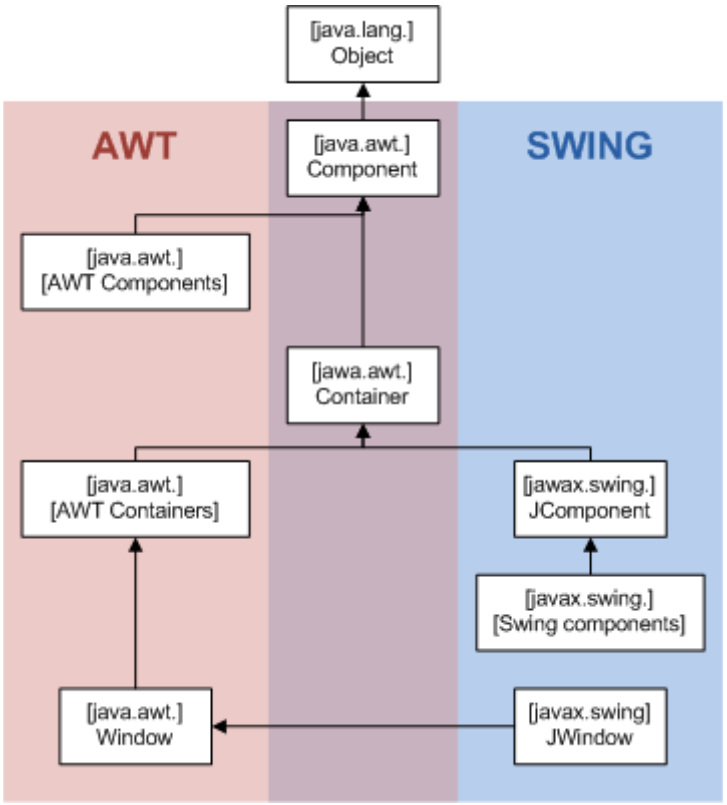
AWT (Abstract Window Toolkit) est la première couche graphique pour Java. Les interfaces sont typiques.

Au-delà de l'aspect graphique, un certain nombre d'objets graphiques n'existent pas.

Créée en 1995, AWT propose plusieurs bibliothèques pour la gestion des fenêtres (window), des événements (event) et des dispositions (layout).

Swing apporte cependant une plus grande souplesse, en ajoutant de nombreux éléments complémentaires. L'adoption de Swing a cependant été freinée par une incompatibilité de cadre (mauvaise gestion du "Z-Order" entre AWT et Swing). En effet, la représentation des éléments à afficher fait penser à des **calques**, sur lesquels l'utilisateur ne peut agir que sur le plus proche.

Depuis Java 6, update 12, les problèmes sont résolus et AWT et Swing cohabitent pour proposer des interfaces riches et souples à la fois.



Tout comme dans Purebasic, la création de fenêtre et le placement des formes sur celles-ci se fait par programmation.

Exemple de code (Wikipedia) :

```
import java.awt.FlowLayout;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.WindowConstants;
import javax.swing.SwingUtilities;

public class SwingExample implements Runnable {

    @Override
    public void run() {
        // Create the window
        JFrame f = new JFrame("Hello, !");
        // Sets the behavior for when the window is closed
        f.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
```

```
        // Add a layout manager so that the button is not placed on top
of the label
        f.setLayout(new FlowLayout());
        // Add a label and a button
        f.add(new JLabel("Hello, world!"));
        f.add(new JButton("Press me!"));
        // Arrange the components inside the window
        f.pack();
        // By default, the window is not visible. Make it visible.
        f.setVisible(true);
    }

    public static void main(String[] args) {
        SwingExample se = new SwingExample();
        // Schedules the application to be run at the correct time in the
event queue.
        SwingUtilities.invokeLater(se);
    }
}
```

J.3 JavaFX

Version la plus récente de bibliothèques graphique pour Java, JavaFX est une technologie créée en 2009 par Sun Microsystems mais rendue standard depuis mars 2014 par Oracle, avec Java 8.

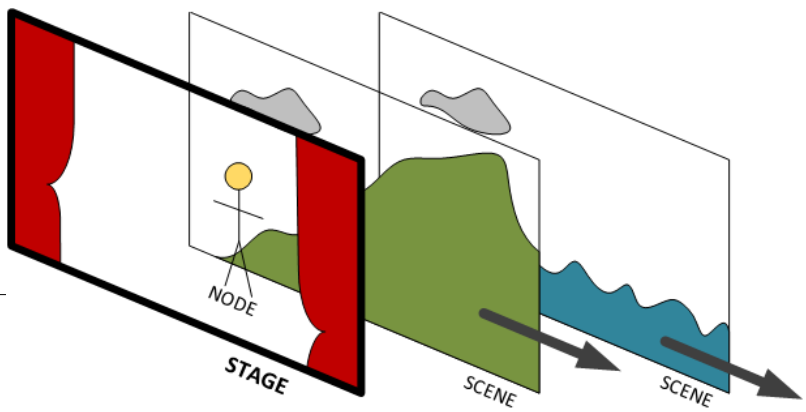
JavaFX est une API complète, capable de gérer entre autres, les médias audio et vidéo, le graphisme 2D et 3D. Les avantages de JavaFX sur Swing sont :

- Le formatage des composants est compatible avec les styles CSS (skins possibles)
- La gestion des événements est plus fine (événements plus nombreux)
- La possibilité d'utiliser des effets spéciaux (par exemple un flou) et des animations
- Le support des équipements tactiles

J.3.1 Représentation et analogie

JavaFX utilise un vocabulaire emprunté au spectacle pour permettre de créer une interface graphique réussie. Voyons à quoi cela correspond :

- **Stage** : c'est le cadre général, rien ne peut en sortir. Au théatre, cela correspond à l'espace ou les acteurs jouent.



- **Scène** : c'est le décor devant lequel les acteurs évoluent. Si la scène est plus grande que le stage, elle sera tronquée.
- **Nœud** (Node) : ce sont les acteurs, ceux qui interagissent. Ils peuvent être regroupés par groupes !

Si une scène ne contient pas trop d'éléments (et surtout pas d'encombrantes images), il est possible de les charger simultanément. Toutefois, il peut être préférable de charger ou décharger les scènes pour des raisons de performances.

J.3.2 Fonctionnement

Il y a deux possibilités de création d'interfaces JavaFX :

- **Fichier FXML** : c'est un fichier de **déclaration**, il ne contient pas de code Java, mais des directives pour dessiner une scène. Il faut donc un fichier FXML par scène.
- **Instruction Java** : il est possible de créer les éléments par programmation **procédurale**, ce qui reste le plus pratique pour ajouter des listes d'objets.

```
<BorderPane maxWidth="-Infinity"
            minHeight="-Infinity"
            minWidth="-Infinity"
            PrefHeight="400.0"
            PrefWidth="750.0"
 xmlns="http://javafx.com/javafx/8.0.112"
xmlns:fx="http://javafx.com/fxml/1"
 fx:controller="BoxVelbus.Controller">
 BorderPane.alignment="CENTER">
         <children>
            <TextField fx:id="sIPAddress"
             promptText="Enter IP"
             text="192.168.1.1">
```

```
public class Scene1 extends Application {
    Label lExplication;
    Button btn1, btn2;

    @Override
    Public void start(Stage premierStage)
{
        btn1 = new Button("Hello");
        btn2 = new Button("GoodBye");
    ...
```

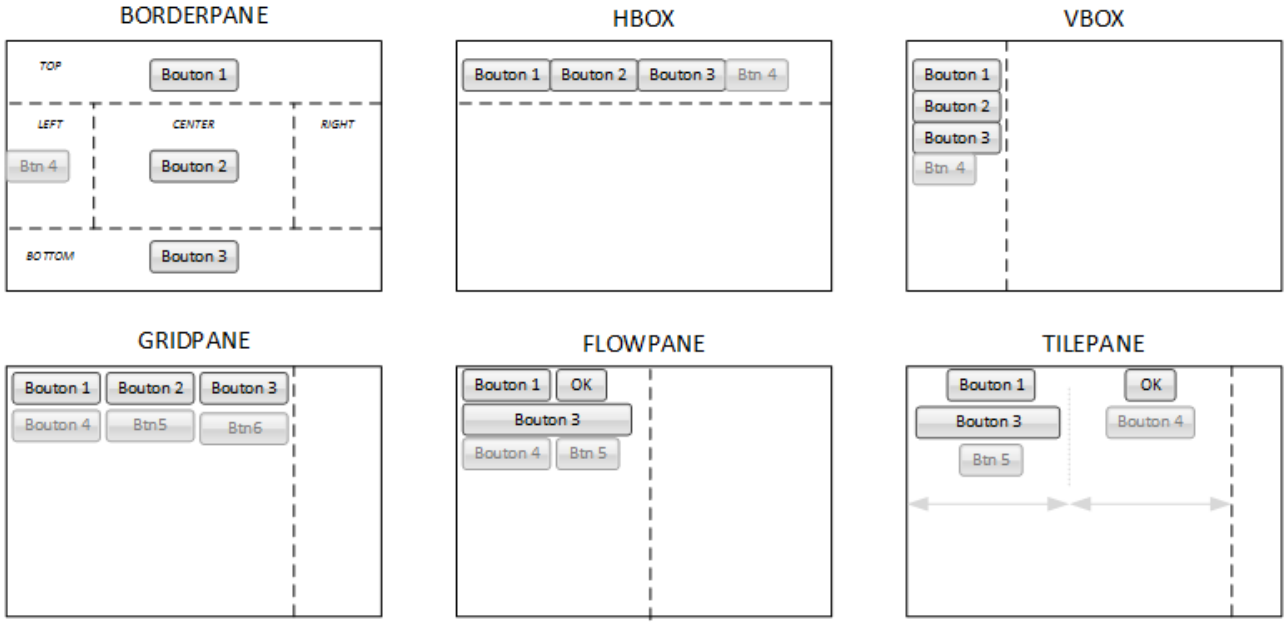
J.4 Dispositions (layouts)

Java ayant pour vocation de fonctionner sur un maximum d'équipements différents, allant du téléphone à l'écran interactif, les interfaces graphiques intègrent la notion de layout : il s'agit de la capacité de placement dynamique des éléments de la fenêtre. Les layouts sont disponibles avec AWT, Swing et JavaFX bien sûr. La disposition permet d'adapter automatiquement la disposition des composants d'une interface, en fonction de la résolution ou de l'orientation.

J.4.1 Les modèles de disposition

Il y a quelques modèles de base comme "Border Pane", "VBox", "HBox", "GridPane", "FlowPane", "AnchorPane", "StackPane", "TilePane"…

Le plus intéressant est qu'il est possible de placer chacun de ces layouts à l'intérieur d'un autre layout, ce qui permet de créer des interfaces complexes.



Il est également possible de placer les composants directement avec les coordonnées : cela présentera cependant un inconvénient majeur en cas de redimensionnement de la fenêtre ou du cadre (frame).

Il est bien sûr possible de créer plusieurs scènes, en choisissant celle qui doit être visible :

```
scene1 = new Scene(pane1, 200, 100);

scene2 = new Scene(pane2, 200, 100);

primaryStage.setTitle("Hello World!");
primaryStage.setScene(scene1);
primaryStage.show();
```

J.5 TD : Création d'une interface graphique JavaFX

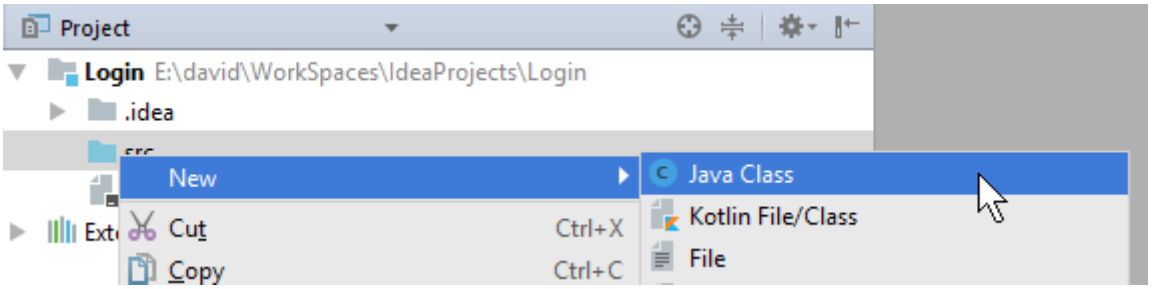
Ce TD a pour objectif de créer une fenêtre d'authentification avec 2 champs et un bouton. La création est faite en mode procédural pour permettre la compréhension des instructions.

Le TD est basé sur un exemple du site d'Oracle : http://docs.oracle.com/javafx/2/get_started/form.htm

J.5.1 Création

Il faut créer un nouveau projet dans l'IDE qui se nommera Login.

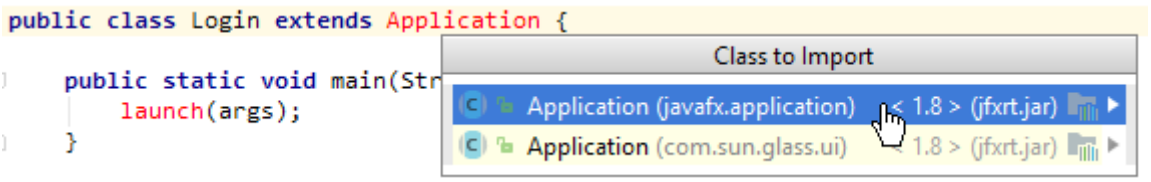
Dans ce projet, il faut créer une nouvelle classe Java qui s'appellera Login.java



Le code suivant est à modifier en ajoutant extends Application (implémente la classe Application) :

```
public class Login extends Application {
}
```

L'IDE détecte une erreur, l'utilisation de [Alt]+[Entrée] permet à l'IDE de proposer une action (Intention action). Dans les choix possible, prendre ceux qui concerne JavaFX.



Le code devient le suivant :

```
import javafx.application.Application;
public class Login extends Application {
}
```

Il faut ensuite écrire la fonction principale dans la classe :

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

L'écriture launch() permet de lancer une application et ne passer à l'instruction suivante que lorsque l'application sera terminée. Ainsi, main() envoie les arguments à la méthode launch() de la classe Application et c'est cette classe qui gère le déroulement de l'application.

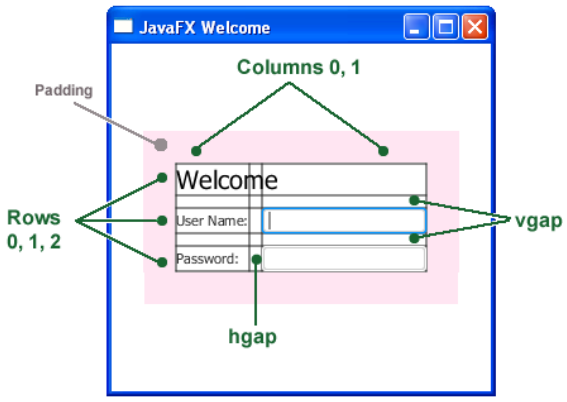
La deuxième méthode qui doit-être écrite (ou plutôt écrasée) est la méthode start()

```
    @Override
    public void start(Stage primaryStage) {
        // code de l'interface graphique à placer ici
    }
```

L'instruction @override permet d'indiquer à Java que la méthode qui suit est une ré-écriture d'une méthode existante dans la classe mère. En effet, start() est la méthode appelée par la classe Application. Désormais, le code contenu par notre méthode start() sera exécuté à la place de la méthode start() de la classe parent. L'intérêt est que le compilateur continue à vérifier que la méthode est utilisée comme la méthode de la classe mère (mêmes arguements, même type de retour…)

La méthode peut aussi être une interface (depuis Java 1.6)

Une fois l'objet stage reçu, il est possible de créer les éléments graphiques :



L'idée est de créer une grille de 3 lignes et 2 colonnes :

- Chaque cellule de la grille est espacée de 10 pixels (vgap et hgap).
- La grille se situe à 25 pixels des bords de la fenêtre (stage) : c'est le padding.
- La grille sera au centre de la fenêtre.

Voici les lignes de code à insérer à l'intérieur de la méthode start() :

```
        primaryStage.setTitle("JavaFX Welcome");
        GridPane grid = new GridPane();
        grid.setAlignment(Pos.CENTER);
        grid.setHgap(10);
        grid.setVgap(10);
        grid.setPadding(new Insets(25, 25, 25, 25));
```

Il faut maintenant remplir la grille avec des éléments : 2 champs de texte pour les saisies, des labels pour les descriptions et un bouton. Un autre élément de type texte permet de créer un titre à l'intérieur de la grille (différent du titre de la fenêtre) :

```
Text scenetitle = new Text("Welcome");
scenetitle.setFont(Font.font("Tahoma", FontWeight.NORMAL, 20));
grid.add(scenetitle, 0, 0, 2, 1);

Label userName = new Label("User Name:");
grid.add(userName, 0, 1);

TextField userTextField = new TextField();
grid.add(userTextField, 1, 1);

Label pw = new Label("Password:");
grid.add(pw, 0, 2);

PasswordField pwBox = new PasswordField();
grid.add(pwBox, 1, 2);

Button btn = new Button("Sign in");
HBox hbBtn = new HBox(10);
hbBtn.setAlignment(Pos.BOTTOM_RIGHT);
hbBtn.getChildren().add(btn);
grid.add(hbBtn, 1, 4);

Scene scene = new Scene(grid, 300, 275);
primaryStage.setScene(scene);
primaryStage.show();
```

La démarche est toujours la même : l'objet est instancié, puis placé à l'intérieur de la grille (d'abord la colonne puis la ligne) .

A chaque fois que l'éditeur affiche une instruction en rouge, il faut normalement appuyer sur [Alt]+[Entrée] pour que celui-ci propose l'importation des classes manquantes. Cependant on peut alléger le code généré en remplaçant le dernier item par étoile (mais ça ne marche pas toujours).

Voici un exemple de ce qu'il est possible de faire (code en début du programme, premières lignes) :

```
import javafx.application.Application;
import javafx.geometry.*;
import javafx.scene.*;
```

```
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.scene.text.*;
import javafx.stage.Stage;
```

Apprenez à découvrir les classes nécessaires à certaines méthodes, cela vous donnera plus d'autonomie et de compréhension à la création ou correction de codes.

En face de la classe Login et de la méthode main() il doit maintenant y avoir un symbole vert : 

Cela signifie qu'elles sont exécutables. Testez donc votre code (et éliminez éventuellement les erreurs) : n'oubliez pas de fermer votre application avant de continuer à éditer le code.

J.5.2 Gestion d'action

Nous allons maintenant associer une action au bouton : lorsqu'il sera cliqué, un texte s'affichera.

Ajoutez le code suivant, juste avant la ligne "Scene scene = new Scene(...)"

```
final Text actiontarget = new Text();
grid.add(actiontarget, 1, 6);

btn.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        actiontarget.setFill(Color.FIREBRICK);
        actiontarget.setText("button pressed, username =
"+userTextField.getText());
    }
});
```

et le code suivant avec les autres importations :

```
import javafx.event.*;
import javafx.scene.paint.Color;
```

Le mot-clé 'final' n'est pas nécessaire ici, mais il vaut mieux le laisser pour permettre l'accès à ce t objet depuis ailleurs.

Notez que tout ce qui est contenu dans la parenthèse est considéré comme une expression lambda.

La méthode setOnAction() enregistre une action à exécuter lorsque un événement est détecté sur l'objet (ici btn).

Pour être précis, un objet de type listener ("qui écoute") sur une liste (ActionEvent) exécute les actions placées dans la méthode handle(). Il pourrait y avoir plusieurs événements pour certains objets (clic droit, clic gauche, etc.)

Testez à nouveau votre code :

Essayez avec des noms très longs, des caractères accentués ou bizarres, etc.

Pour aller plus loin, voici comment réaliser le même travail en mode déclaratif FXML) :

http://docs.oracle.com/javafx/2/get_started/fxml_tutorial.htm

K Annexes

K.1 Correction Exercice écriture fichier

```
/**
 * @author David ROUMANET
 * @description programme permettant la saisie du nom, prénom et age et leur écriture dans un
fichier.
 *              Exercice permettant la mise en oeuvre de fichier, de boucle, de conversion et de
méthode.
 */
package net.roumanet;
import java.io.*;
import java.util.Scanner;
public class Main {
   // méthode principale appelée par Java pour exécuter le programme
   public static void main(String[] args) throws IOException {
      // Préparation des attributs
      Scanner clavier = new Scanner(System.in);  // scanner qui permet de récupérer les entrées
clavier
      String monNom="", monPrenom="";
      Integer monAge=0;
      do {
         System.out.println("Entrez votre nom :");
         monNom = clavier.nextLine();
         System.out.println("Entrez votre prénom :");
         monPrenom = clavier.nextLine();
         System.out.println("Entrez votre âge :");
         monAge = clavier.nextInt();
         clavier.nextLine();
         if (monAge<1 || monAge > 130) {
            System.out.println("L'age saisie est impossible. Veuillez reprendre la saisie !");
         }
      } while (monAge <1 || monAge > 130);
      // Enregistrement des informations en mode texte dans un fichier
      if (enregistreFichier(monNom, monPrenom, monAge)) {
         System.out.println("Enregistrement réussi.");
      }
   }
   // Ce n'est parce que c'est du Java, qu'on oublie de programmer avec des procédures (méthodes) !
   public static boolean enregistreFichier(String nom, String prenom, Integer age) throws
IOException {
      // throws IOException est nécessaire pour gérer le risque d'échec de création de fichier
      BufferedWriter bw = new BufferedWriter(new FileWriter(new File("E:/Temp/contact.txt"),true));
      // on applique ici quelques trucs vus dans le cours : toUpperCase(), toString(), etc.
      bw.write("NOM: "+nom.toUpperCase()+",PRENOM: "+prenom+", AGE: "+age.toString());
      bw.newLine();
      bw.close();
      return true;
   }
}
```