Programmation Orientée Objet
Programmation JAVA

FISE 3A ICy / FISA 3A Informatique

CM2 : Les bases de langage Java

Mohamed Amine BOUDIA
UPHF, CNRS, UMR 8201 - LAMIH, F-59313 Valenciennes, France
Email : mohamedamine.boudia@uphf.fr

LAMIH UMR CNRS 8201 | Université Polytechnique HAUTS-DE-FRANCE | cnrs

Objectif Zéro Papier

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES HAUTS-DE-FRANCE

Notre écosystème

- Java 8 ou + .
- Eclipse : véritable machine à gaz .
- Netbeans : plus souple que Eclipse .
- Autre IDE (Visual Studio Code, IntelliJ IDEA, ..) .

JAVA

- Développée par SUN Microsystems en 1995, cette dernière fut acheté par Oracle en 2009 .

  – JAVA est un langage de programmation .

  – environnement logiciel ou une plate-forme dans lequel les programmes s'exécutent .

  - 2006 : passage de JAVA sous licence GPL .

  - 2007 : Passage en Open Access .

- Présente dans de très nombreux domaines d'applications : serveurs, smartphone..

  - estimation d'Oracle : 800 million de PC, 2.1milliards de téléphones portables, 3.5 milliard de carte puce; décodeurs, imprimantes, automobiles, borne de paiement …

JAVA

- « *Java est un langage simple, orienté objet, distribué, robuste, sûr, indépendant des architectures matérielles, portable, de haute performance, multithread et dynamiquePrésente dans de très nombreux domaines d'applications : serveurs, smartphone..* » [définition Sun]

Caractéristique de Java

- S'inspire de **C++ et Smalltalk** avec moins de complexités : pointeurs, allocation mémoire …
  - Gestion automatique de la mémoire .
- Orienté objet,
  - Objet : unité ayant un ensemble de variables et de fonctions (ou "méthodes") qui lui sont propres .
  - les objets sont regroupés en classe, qui est l'unité de base en Java
  - une classe organise un ensemble de variables et de fonctions qui seront disponibles dans les objets créés à partir de cette classe.
  - de nombreuses classes sont disponibles
  - un objet créé à partir d'une classe est une "instance" de cette classe

Caractéristique de Java

- Typage statique fort .
  - Tout objet a un type non modifiable
  - Le type d'un objet détermine à la fois son état et son comportement
- Bibliothèque de classes et de packages très riche .
- Polymorphisme et introspection .

- Robuste : typage des données très strict, pas de pointeurs (un "garbage collector")
- Sûr et portable : compilation en "byte code" pour une machine virtuelle.
- Haute performance :très discutable car Java est un langage pseudo interprété.

Caractéristique de Java

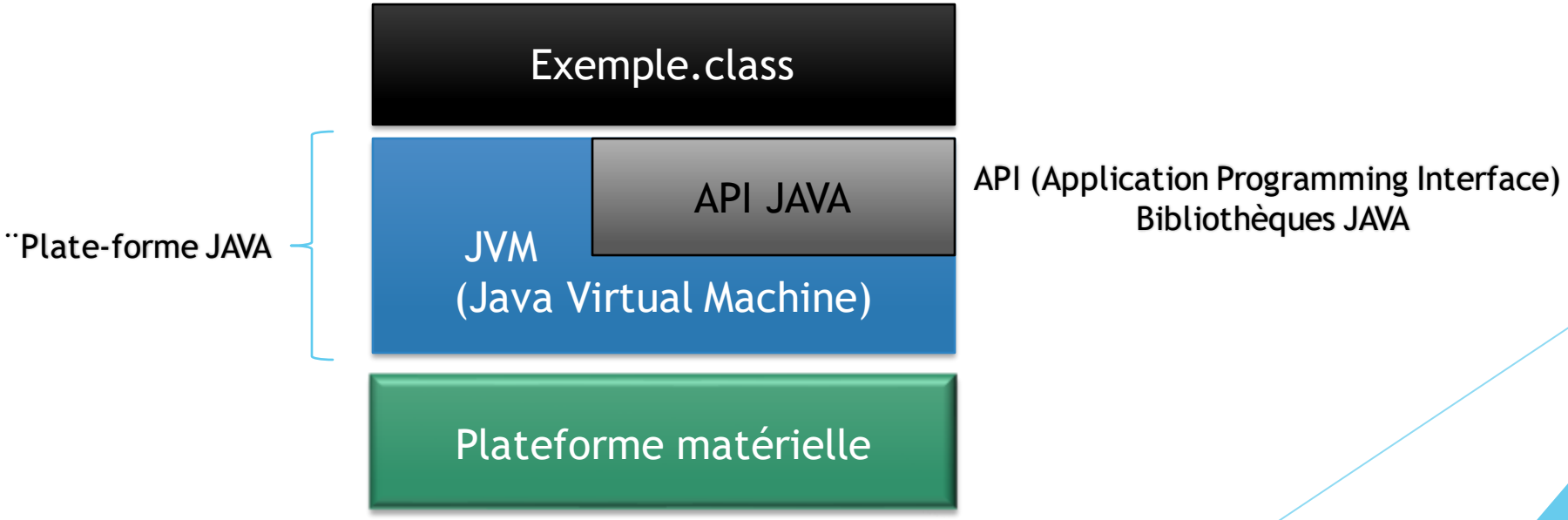
- Distribué : les fonctions d'accès au réseau et les protocoles Internet courants sont intégrés
- Multi thread : on peut découper une application en unités d'exécution séparées et apparemment simultanées.
- Dynamique : les classes peuvent être modifiées sans avoir à modifier les programmes qui les utilisent.
- Portabilité : compilation vers le byte-code qui s'exécute dans une machine virtuelle .

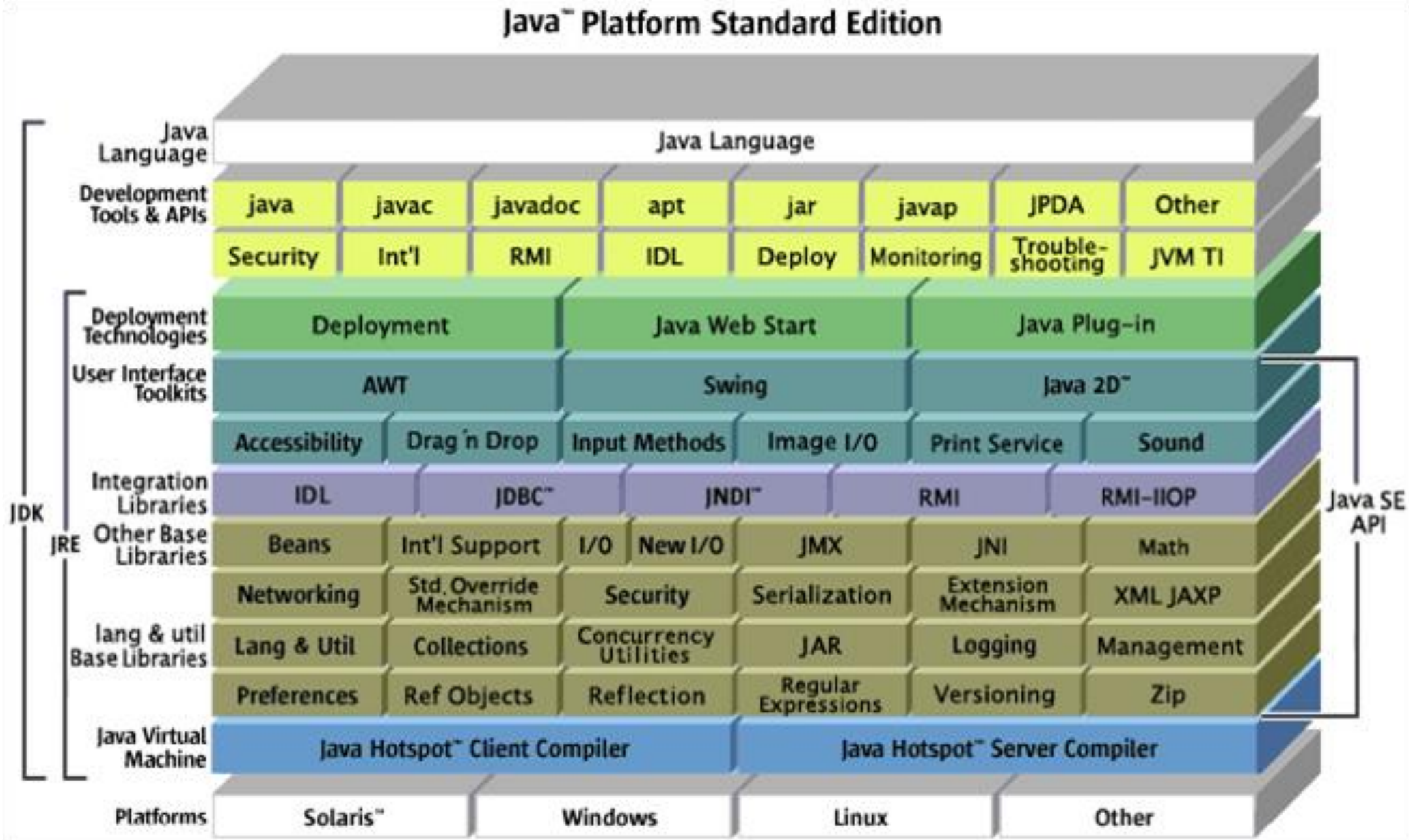
Différentes éditions de JAVA

- JSE : Java Standard Edition : pour les ordinateurs de bureau, celle que nous utiliserons,
  - 2 principaux composants :
    - ✓ JRE (Java Runtime Environment) : C'est l'environnement nécessaire à l'exécution d'appels et d'applications crées à l'aide de Java =>implémentation JVM
    - ✓ JDK (Java Development Kit) : Qui contient le compilateur Java, Javadoc, le débogueur…
- JEE : Java Entreprise Edition : Version pour les application « d'entreprise » qui permet la création d'application distribuées et accès SGBD
- J2ME : Java 2 Micro Edition : version dédiée aux dispositifs mobiles

Plate-forme

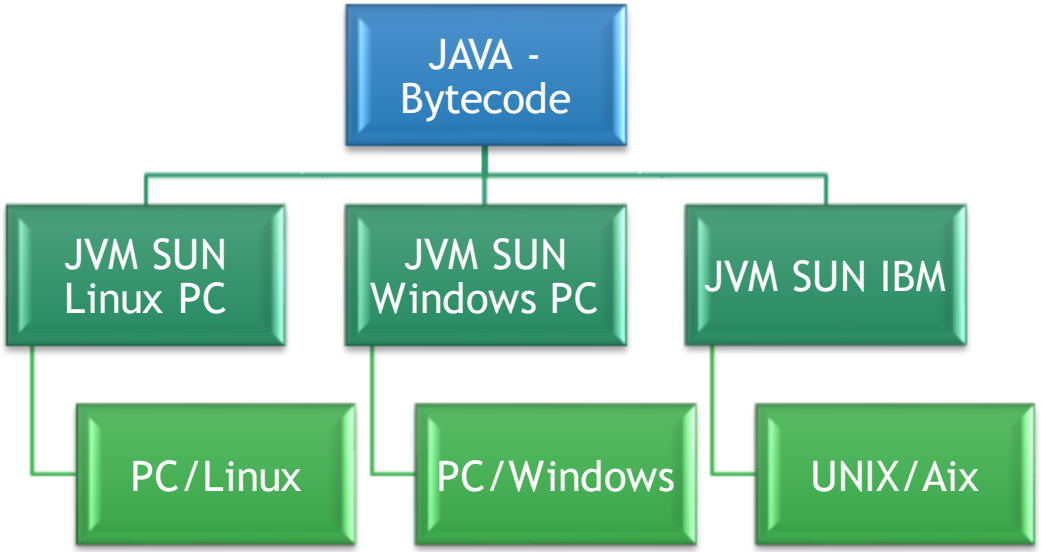
- Généralement on entend par plate-forme : la combinaison d'un système d'exploitation et du matériel .
  - Exemple : Win + Intet, Linux + Intel, MacOS + PowerPC…
- La plateforme JAVA est couche logicielle (JVM) qui s'implémente au dessus du matérielles et permet d'exécuter correctement des programme JAVA.




Java™ Platform Standard Edition
Java Language
Java Language
Development Tools & APIs
java
javac
javadoc
apt
jar
javap
JPDA
Other
Security
Int'l
RMI
IDL
Deploy
Monitoring
Trouble-shooting
JVM TI
Deployment Technologies
Deployment
Java Web Start
Java Plug-in
User Interface Toolkits
AWT
Swing
Java 2D™
Accessibility
Drag 'n Drop
Input Methods
Image I/O
Print Service
Sound
Integration Libraries
IDL
JDBC™
JNDI™
RMI
RMI-IIOP
Other Base Libraries
Beans
Int'l Support
I/O
New I/O
JMX
JNI
Math
Networking
Std. Override Mechanism
Security
Serialization
Extension Mechanism
XML JAXP
lang & util Base Libraries
Lang & Util
Collections
Concurrency Utilities
JAR
Logging
Management
Preferences
Ref Objects
Reflection
Regular Expressions
Versioning
Zip
Java Virtual Machine
Java Hotspot™ Client Compiler
Java Hotspot™ Server Compiler
Platforms
Solaris™
Windows
Linux
Other
JDK
JRE
Java SE API

La machine virtuelle JAVA

- La portabilité des programmes JAVA est assurée par le byte-code :
  - Langage pour une machine virtuelle .
  - À l'exécution un interpréteur simule cette machine virtuelle appelée JVM .



API Java

- Notion d’API : modèle définissant une classe, les attributs accessibles, le mode d'appel des méthodes et leur retour, ainsi que les erreurs qu'il est possible d'intercepter.
- Les API JAVA : sont une grande nombre de fonctions/méthodes/services appelés collection de composant logiciel organisée en bibliothèques appelées packages .
- Un programmeur JAVA doit connaître au moins une partie des API ou au moins son existence ou savoir retrouver les services nécessaires .

Principaux paquetage:

- java.applet : création d'applets.
- java.awt : création d'interfaces graphiques portables avec AWT.
- java.bean : création de Java Bean.
- java.io : accès et gestion des flux en entrées/sorties.
- java.lang : classes et interfaces fondamentales.
- java.math : opérations mathématiques.
- java.net : accès aux réseaux, gestion des communications à travers le réseau

D’autre paquetage plus compliqué

- javax.crypto : cryptographie.
- Javax.3D
- javax.imageio
- javax.jws : Java Web Services.
- javax.lang.model :
- javax.management : API JMX.
- javax.naming : API JNDI (accès aux annuaires)

Heap(Tas) et Stack (Pile) en Java

- Stack est la portion de mémoire qui a est assignée à chaque programme. Elle est fixée.
- D'autre part, la mémoire Heap est la portion qui n'a pas été allouée au programme java mais qui sera disponible pour être utilisée par le programme java quand il en aura besoin, principalement pendant l'exécution du programme.
- Java utilise cette mémoire comme:
  - Lorsque nous écrivons un programme Java, toutes les variables, méthodes, etc. sont stockées dans la mémoire de pile(Stack).
  - Et lorsque nous créons un objet dans le programme Java, cet objet est créé dans la mémoire du tas(Heap). Et il est référencé à partir de la mémoire de la pile(Stack).

En java : pas de pointeurs

- Les pointeurs sont assez compliqués et dangereux à utiliser pour les programmeurs débutants.
- Java se concentre sur la simplicité du code, et l'utilisation de pointeurs peut le rendre difficile.
- L'utilisation de pointeurs peut également provoquer des erreurs potentielles.
- De plus, la sécurité est également compromise si des pointeurs sont utilisés, car les utilisateurs peuvent accéder directement à la mémoire à l'aide des pointeurs.
- Ainsi, un certain niveau d'abstraction est fourni en n'incluant pas les pointeurs dans Java.

Mon premier programme

```
Public class Helloword {

Public static void main (String [] args)

        {

        System.out.println  (''Hello World !'');

        }

}
```

Tout code java doit être défini à l'intérieur d'une classe : public class HelloWorld

- La description de la classe est effectuée à l'intérieur d'un bloc { } ;
- Le code de la classe doit être enregistré dans un fichier de même nom que la classe (casse comprise) HelloWorld.java ;
- Le point d'entrée comme en C est la méthode main (nous reverrons le concept de méthodes ultérieurement)

```
Public class Helloword {
Public static void main (String [] args)
      {
      System.out.println (”Hello World !”);
      }
}
```

HelloWorld.java

Javac.exe

1000 1100 1010 0000 1001 1110 1000
0110 0010 0000 0111 1010 1111 0000

HelloWorld.class

java.exe

Smartphone

PC

MAC

Interface console
interface graphique (GUI)

- En Java, pour que le fichier soit valide, il doit porter exactement le même nom que la classe publique qu'il contient.

- Programmes à interface console (en ligne de commande)
  - MS-DOS sous Windows
  - Fenêtre de commande sous Unix / Lunix
    - javac Helloword.java
    - java Helloword

- Interface graphique GUI

1er Programme Java

```
import java.io.*;
class Test
{
Int i;
void method  () { }
int function  () { return 0;}

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

Bibliothèques (packages en anglais)

Le mot clé class suivi du nom du programme

Variable(s)

Méthode (s) / Procédure

Méthode (s) / Fonction

le point d'entrée de tout programme Java

Contenu du programme

Remarque

- Application est définie par un ensemble de classes dont une jouera le rôle de classe principale .
- La compilation de la classe principale entraîne la compilation de toutes les classes utilisées sauf celles qui dont fournies et font partie de la hiérarchie java .
- pour exécuter l'application, on indique à l'interpréteur java le nom de la classe principale .
- java charge les classes nécessaires au fur et à mesure de l'exécution .
- Comme en C, les instructions Java sont séparées par des points virgules (;).

L’essentiel

- Tous les programmes Java sont composés d'au moins une classe.
- Le point de départ de tout programme Java est la méthode suivante :

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

- On peut afficher des messages dans la console grâce à :

```
System.out.println(“Hello World !”);
```

qui affiche un message avec un saut de ligne à la fin.

Identificateur

- Permet de nommer les classes, les variables, les méthodes, …
- Un identificateur Java
  - Est de longueur quelconque ;
  - Commence par une lettre Unicode (https://www.unicode.org);
  - Peut ensuite contenir des lettres ou des chiffres ou le caractères souligné ;
  - Ne doit pas être un mot réservé au langage (mot clef), par exemple if,

    switch, class, true, ...

Quelques convention

- Pour faciliter la réutilisation, il y a une convention de nommage. En fait, c'est une façon d'appeler les classes, les variables, etc. Il faut essayer de respecter cette convention:
  - 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 .
  - Les constantes sont en majuscules .

Quelques mots clés JAVA

abstract else instanceof static try boolean false assert enum

interface strictfp volatile byte true break extends native

super while char case final new switch double catch finally

package synchronized float class for private this int const

goto protected throw long continue if public throws short

default implements return transient void null do import

Commentaires

- sur une ligne :

```
int i ; // commentaire  sur une ligne
```

- sur plusieurs lignes :

```
/* comme  en C commentaires
sur plusieurs lignes */
```

- commentaires pour l'outil javadoc :

```
/**
 * pour l'utilisation  de javadoc
*/
```

- Les conseils/consignes utilisés pour les autres langages de programmation sont bien évidemment toujours d'actualité
- commenter le plus possible et judicieusement
- chaque déclaration (variable, méthode ou classe)

app.wooclap.com/CTHOEU

1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
CTHOEU

1 Envoyez @CTHOEU au 06 44 60 96 62

2 Vous pouvez participer

Désactiver les réponses par SMS

Les variables

et

types de données

Déclarations

- Toute variable doit être déclarée avant d'être utilisée .
- La déclaration d'une variable consiste à associer un type à un identificateur, la syntaxe générale est : <type> variable.
- Peut être déclarée à n'importe quel niveau dans le code .
- Il est possible d'initialiser une variable lors de sa déclaration
- Une variable est accessible (visible) dans le bloc où elle a été déclarée jusqu'à la fin du bloc où elle a été déclarée .

Affectation

- Syntaxe identificateur = expression .
- En java, il est possible de réaliser des affectations multiples.

Déclaration de constante

- Par convention, les noms des constantes (« variable » avec static final) sont écrites en majuscules .
- Elle ne peut plus être modifiée une fois qu'elle a été initialisée.

Exemple :

```
public static final int VITESSE_LIMITE_AUTOROUTE_FR  = 130;
```

Mot clé static (à retenir)

- En Java, le mot-clé static est utilisé pour indiquer qu'un élément appartient à la classe elle-même, et non aux objets (ou instances) de cette classe.

- Variable static :
  - Une variable static est partagée par toutes les instances de la classe.
  - Plutôt que chaque objet ait sa propre copie de la variable, il n'y a qu'une seule copie pour toute la classe.
  - Exemple : une variable static est utile lorsqu'on veut suivre quelque chose de global pour la classe, comme un compteur qui enregistre combien d'objets ont été créés.

- Méthode static :
  - Une méthode static appartient à la classe et peut être appelée sans créer d'instance (objet) de cette classe.
  - Par exemple : Math.sqrt() est une méthode statique de la classe Math. Vous n'avez pas besoin de créer un objet Math pour l'utiliser.
  - Une méthode static ne peut pas accéder aux variables non static de la classe car elle n'est pas liée à une instance particulière.

Mot clé final (à retenir)

- Le mot-clé final signifie "non modifiable" ou "définitif". Selon le contexte, il a plusieurs effets différents :

- Variable final :
  - Une variable final est une constante. Une fois assignée, elle ne peut plus être modifiée.
  - Elle est souvent utilisée pour des valeurs fixes, comme PI, afin d'éviter toute modification accidentelle.

- Méthode final :
  - Une méthode final ne peut pas être redéfinie par une sous-classe.
  - Cela garantit que le comportement de cette méthode reste le même dans toute la hiérarchie de classes.

- Classe final :
  - Une classe final ne peut pas être héritée.
  - C'est utile pour éviter qu'une classe soit modifiée ou étendue par d'autres classes, comme c'est le cas pour la classe String en Java.

Variable d'instance et variable locales

- Variables d'instance: Les variables d'instance sont des variables qui sont accessibles par toutes les méthodes de la classe. Elles sont déclarées en dehors des méthodes et à l'intérieur de la classe. Ces variables décrivent les propriétés d'un objet et restent liées à celui-ci à tout prix.
- Variables locales : Les variables locales sont des variables présentes dans un bloc, une fonction ou un constructeur et ne sont accessibles qu'à l'intérieur de ceux-ci. L'utilisation de la variable est limitée à la portée du bloc. Lorsqu'une variable locale est déclarée dans une méthode, les autres méthodes de la classe n'ont pas connaissance de la variable locale.

```
class Person {
        public Int id; //variable d'instance
        public String name; //variable d'instance
        public String address; //variable d'instance
        public int age; //variable d'instance
}
```

```
public void createPerson() {
        int id; //variable locale
        String name; //variable locale
        String address; //variable locale
        int age; //variable locale
}
```

Types de données en JAVA

- 2 grands groupes de types de donnée :
  1. Types primitifs .
  2. Types références / compelxe (objet=> instances de classe) .

Java manipule différemment les valeurs des types primitifs et les objets :

- des valeurs de types primitifs .
- ou des références aux objets .

Types primitifs

- La durée de vie d'un type primitif est l'accolade "parente".

- Logique / Booléen .
  - boolean (true/false) .
  - Opérateur booléens : ! **& &** **||**
  - Comparaison : ==; !=; <; <=; >; >=
- Nombre entiers .
  - byte (1 octet), short (2 octets), int (4 octets), long (8 octets) .
  - nous utiliserons les int (attention les entiers sont toujours signés) .
  - format : 17, -4, 42, 0xb0, 0b01110
  - opérateurs : +, -, *, /, % (et d'autres) .

Types primitifs

- Valeurs réels .
  - float (4 octets), double (8 octets) .
  - nous utilisons les double .
  - format : 17.0, -4.5, 1.2e3
  - opérateurs : +, -, *, /
- Caractère .
  - char (2 octets). jeu de caractère unicode : https://www.unicode.org .

  entre 2 apostrophes ; Tous les caractères accentués du français sont présents.
  - – format : 'a', '1', '\u03c0' → p, ...
  - – '\t' tabulation .
  - – '\n' retour à la ligne .

Types primitifs

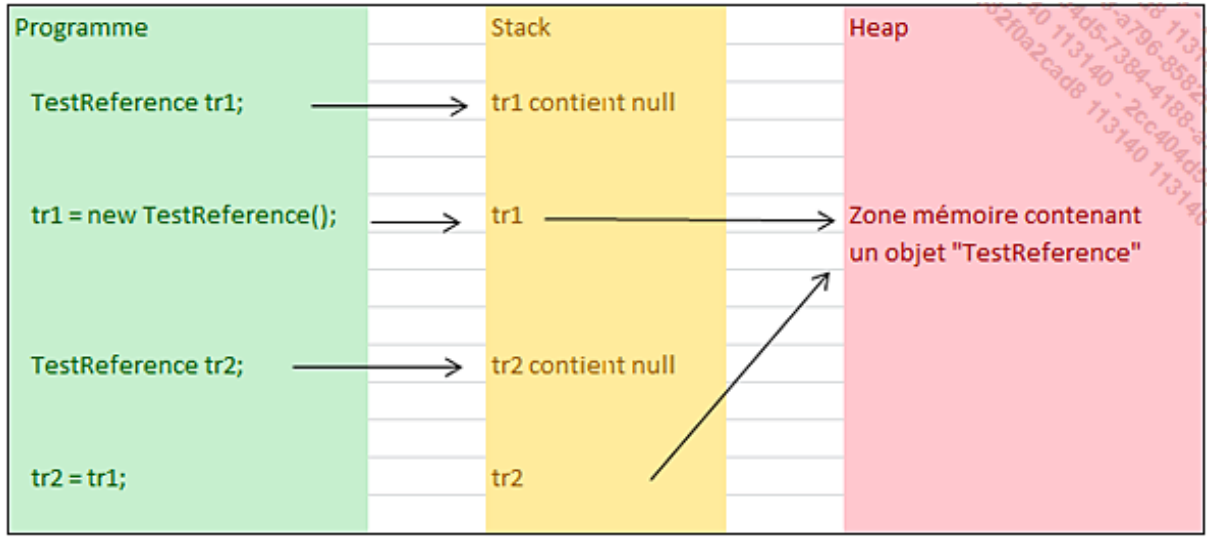
Type élémentaire	Intervalle de variation	Nombre de bits
boolean	false , true	1 bit
byte	[-128 , +127]	8 bits
char	Caractères	16 bits
double	Virgule flottante double précision [1.7e-308, 1.7e308]	64 bits
float	Virgule flottante simple précision [3.4e-38, 3.4e+38]	32 bits
int	entier signé : [-231, +231 - 1], i.e. [-2147483 648…2147483647]	32 bits
long	entier signé long : [-263, +263- 1], i.e. [-92233720368547758 08, 92233720368547758 07]	16 bits
short	entier signé court : [-215, +215 -1], i.e. [-32768, 32767]	64 bits

Types références

- Boolean, Byte, Character, Short, Integer, Long, Float et Double. Une variable de ce type est accessible via des méthodes :
  - Ce n'est pas un type primitif, c'est **une classe** (d'où la présence de la majuscule) .
  - Un type référence peut ne rien référencer (ou pas encore). Dans ce cas, il contient null.
  - La déclaration d'une variable de type classe d'objet se fait par le mot clé new qui permet d'instancier une classe. Par exemple :
    - ✓ Integer un = new Integer(1);
  - La Méthode xxxValue(), où xxx est l'un des noms du type primitif correspondant, permet d'obtenir une variable du type primitif correspondant.
  - L'opérateur instanceOf permet de tester si un objet est une instance d'une classe donnée (ou de l'une de ses sous-classes). L'opérateur instanceOf ne permet pas de tester le type d'une primitive.
- Remarque : Java ne dispose pas de type pointeur.

Types références

```
// Création d'une variable référence nommée tr1  sur un objet de type TestReference
TestReference tr1;
// pour l'instant tr1 vaut null
// Allocation d'un objet et stockage de son adresse  (donc de sa référence) dans la variable tr1
tr1 = new TestReference();
// Création d'une variable référence nommée tr2  sur un objet de type TestReference
TestReference tr2;
// pour l'instant tr2 vaut null
// Copie le contenu de tr1 donc de l'adresse de l'objet  dans tr2
tr2 = tr1;
```



Types références

- Référence : Un type référence stocke une adresse pointant vers des données situées dans le heap, pas les données elles-mêmes.
- Gestion mémoire : Les données sont conservées tant qu'une référence y pointe. Sinon, le garbage collector libère la mémoire.
- Valeur null : Une référence non assignée contient null.
- Instanciation avec new : Le mot-clé new crée l'objet et assigne son adresse à la variable de référence.
- Assignation et comparaison : L'assignation copie l'adresse (pas l'objet). La comparaison (==) compare les adresses, pas le contenu.
- Passage de référence : Lorsqu'une référence est passée ou retournée dans une méthode, c'est l'adresse de l'objet qui est utilisée.

La superclasse java.lang.Object

- La classe Object est la classe racine en Java, dont toutes les autres classes héritent implicitement.
- Les méthodes suivantes peuvent être redéfinies pour adapter leur comportement aux besoins de chaque classe.
- Les méthodes suivantes permettent de personnaliser le comportement des objets en Java et de gérer efficacement leur comparaison, leur identification et leur destruction.

La superclasse java.lang.Object

- equals() : Cette méthode compare deux objets pour déterminer s'ils sont égaux.
  - Par défaut, elle compare les références des objets (comme ==). En redéfinissant equals, on peut comparer les attributs d'objets.
- hashCode() : Elle génère un identifiant unique pour chaque instance. Lorsqu'on redéfinit equals, il est conseillé de redéfinir hashCode pour maintenir une cohérence dans les comparaisons d'objets.
- toString() : Cette méthode retourne une représentation textuelle de l'objet, utile pour le débogage.
- finalize() : Méthode appelée par le ramasse-miettes (garbage collector) avant que l'objet soit supprimé. Il est conseillé d'y inclure le code de nettoyage nécessaire.

Casts entre les types primitifs

- Java est fortement typé , on ne peut utiliser un opérateur sur deux types différents sans appliquer le CAST .
- La conversion CAST est d'écrire le nouveau type entre parenthèse avant la variable à « caster ». Il est parfois impossible de faire la conversion entre certains types complexe.
- Conversion automatique (implicite) :

  entier → réel ou petit entier → grand entier (float → double, par rapport au nombre d'octets)

- Pas de conversion automatique en booléen (tout ce qui est différent de 0 n'est pas vrai) .

CAST entre les types primitifs

a) double -> int

```
double nbre1 = 10, nbre2 = 3;
~~int resultat = nbre1 / nbre2;~~                    //impossible
int resultat = (int)(nbre1 / nbre2);                 //affiche 3
```

b) int -> float

```
int i = 2;
float t= (float)i;
```

c) int -> double

```
int i = 123;
double z = (double)i;
```

d) Et inversement

Casts entre les types primitifs

- Attention à la perte de précision dans l'autre sens (du grand au petit) .

  Exemple : conversion d'un réel → entier , ou d'un float en int.

- Les affectations entre types primitifs peuvent utiliser un **cast implicite**, **s'il n'y a pas de perte de précision** .

  int i = 180 ; double v = 4.0*i ;

- Sinon le **cast** doit être **explicite** .

  short s = 65 ;

  s = 10000 ; erreur de compilation

  s = (short) 100000 ; compilation = ok mais valeur erronée (perte de précision)

Casts entre les types primitifs

- Priorité des opérations
  - Si on déclare deux entiers et qu'on mette le résultat dans un double, l'opérateur s'applique sur ces entiers avant d'être affecté .

    int nbre1 = 3, nbre2 = 2;

    double resultat1 = nbre1 / nbre2; // resultat = 1

    ->résultat1 est 1 au lieu de 1,5 car la division est entière selon les opérandes.
  - Le cast après le résultat ne servira à rien :

    double resultat2 = (double)(nbre1 / nbre2); // resultat2= 1
  - Le cast sur l'un des opérandes entiers servira à avoir un résultat réel exacte.

    double resultat3 = (double)nbre1 / nbre2; // resultat= 1.5

Règle générale :
Pour avoir un résultat correcte, il faut CASTER chaque type au type approprié avant de faire l'opération .

Conversions entre les types primitifs

- Du numérique vers String
  - La méthode statique valueOf apparait dans beaucoup de classe comme la classe String et sert à convertir.
  - Il suffit de mettre la variable numérique à convertir comme paramètre de la méthode.
  - On reparlera de cette méthode pour la conversion des classes Color, Date…..

```
int i = 12;

String js = new String();

js = String.valueOf(i);
```

Conversions entre les types primitifs

- Du String vers numérique

Les méthodes clés pour les conversions entre types simple et complexe sont intValue(), floatValue(), doubleValue() et valueOf.

- Pour les autres types simples (respectivement float et double), nous utilisons respectivement Float, Double qui sont des types complexes (classes) comme Integer et qui contiennent les méthodes appropriées (valueOf() pour la conversion et floatValue(), doubleValue() pour extraire la valeur simple) .

Conversions entre les types primitifs

Voici un exemple :

```
int i = 12;

String js = String.valueOf(i);

int k = Integer.valueOf(j).intValue();  // car Integer.valueOf(j) retourne un
                                        Integer(objet) et non pas un int
```

Aussi on peut écrire

```
int k=Integer.parseInt(j) ;
```

Types énumérés

- Type défini en énumérant toutes ses valeurs possibles .

Exemple :

```
public enum CouleurCarte {TREFLE, CARREAU, COEUR, PIQUE}
```

- Utilisation

```
CouleurCarte couleur; . . .
couleur = CouleurCarte.PIQUE;
```

Nombre aléatoire

- Utilisation de la classe Random

```
import java.util.Random;
public class Test {
	public static void main(String  [] args){
	Random r = new Random();
	 }
}
```

app.wooclap.com/HYGZPZ

1 Allez sur wooclap.com

2 Entrez le code d'événement dans le bandeau supérieur

Code d'événement
HYGZPZ

1 Envoyez @HYGZPZ au 06 44 60 96 62

2 Vous pouvez participer

Désactiver les réponses par SMS

Bloc d’instruction

&

Opérations arithmétiques

et logiques

Bloc d’instructions

- permet de grouper un ensemble d'instructions en lui donnant la forme syntaxique d'une seule instruction .

syntaxe { } -> { est comme un début (begin), et } est comme un fin (end)

Exemple

```
int k;
{
int i = 1 ;
int j = 12 ;
j = j + i ;
k = j
}
```

Les opérateurs Java

- Affectation
  - = , += , -= , /= , *=
- Arithmétiques
  - + , - , * , / , % //modulo
  - ++, -- (pré ou post incrémentation/décrémentation)
- Opérateurs arithmétiques binaires
  - & , | , ^ , ~
  - << , >> , >>>
- Logique
  - & & (et) || (ou) ! (négation)
- Relationnels
  - >= , <= , == , != , < , >

Affectation

- Syntaxe identificateur = expression .
- En java, il est possible de réaliser des affectations multiples. Par exemple : x = y = z = 2.

Symbole	Description	Exemple	Opération équivalente
=	affectation	x = 2	x = 2
-=	soustraction et affectation	x-= 2	x = x - 2
+=	addition et affectation	x += 2	x = x + 2
*=	multiplication et affectation	x *= 2	x = x * 2
/=	division et affectation	x /= 2	x = x / 2
%=	modulo et affectation	x %= 2	x = x % 2

Les opérateurs Java

- Arithmétiques

Symbole	Description		Exemple	Opération Équivalente
+	adition		x+y	
-	x-y	soustraction	x-y	
	-x	opposé	-x	X*(-1)
*	multiplication		x*y	
/	division		x/y	
%	modulo (reste de la division)		x % y	

Expression	Interprétation
x++	Utiliser x puis Incrémenter
x--	Utiliser x puis Décrémenter
++x	Incrémenter x puis Utiliser
-- x	Décrémenter x puis Utiliser
x+=y	x=x+y
x-=y	x=x-y

Les opérateurs Java

- Raccourcis arithmétiques

Expression	Interprétation
i++	Utiliser i puis Incrémenter
i--	Utiliser i puis Décrémenter
++i	Incrémenter i puis Utiliser
-ii	Décrémenter i puis Utiliser
i+=j	i=i+j
i-=j	i=i-j

Les opérateurs Java

- Opérateurs arithmétiques binaires

Symbole	Description	Exemple
&	Et	a & b
|	Ou	a | b
^	Ou exclusif	a ^ b
~	Non	~ a
<<	Décalage à gauche	a << 2
>>	Décalage à droite	b >> 2
>>>	Décalage à droite sans extension du signe	b >>> 2

a	b	a & b	a | b	a ^ b	~ a
1	1	1	1	0	0
1	0	0	1	1	0
0	1	0	1	1	1
0	0	0	0	0	1

int a été = -1;

a = a >>> 24;

donne en forme binaire :

11111111111111111111111111111111 -1

>>>24

00000000000000000000000011111111 255

Les opérateurs Java

- Logique

Symbole	Description	Exemple
&&	ET logique	(x>10) && (x<15)
‖	OU logique	(x>10) || (y<10)
!	NON logique	! (x >= 0)

- Relationnels

Symbole	Description	Exemple
= =	équivalent	x == 0
<	plus petit que	x < 2
>	plus grand que	x > 2
<=	plus petit ou égal	x <= 3
>=	plus grand ou égal	x >= 3
!=	non équivalent	a != b

Quelle est la différence entre la méthode equals()

et

l'opérateur d'égalité (==) en Java

equals()	Opérateur d'égalité (==)
Il s'agit d'une méthode définie dans la classe Object.	C'est un opérateur binaire en Java.
Cette méthode est utilisée pour vérifier l'égalité de contenu entre deux objets selon la logique métier spécifiée.	Cet opérateur est utilisé pour comparer des adresses (ou des références), c'est-à-dire pour vérifier si les deux objets pointent vers le même emplacement mémoire.

Priorités entre les opérateurs Java

Postfixés	Appel de méthode [], (params), expr ++, expr--	Les opérations d'égales priorités sont évalués de gauche à droite, sauf les opérations d'affectation de droite à gauche.
Unaires	++expr, --expr +expr, -expr , !	
Création et cast	new, (type)expr	
Multiplicatifs	*, /, %	
Additifs	+, -	
Décalage bits	<<, >>, >>>	
Relationnels	<, >, <=, >=, instanceOf	
Egalité	==, !=	
Bits, et	&	
Bits, ou exclusif	^	
Bits, ou inclusif	|	
Logique, et	&&	
Logique, ou	||	
Conditionnel	? :	
Affecation	=, +=, *=, /=, %=, &=, ^=, |=,<<=, >>=	

Structures de contrôle

- Branchements conditionnels

- Boucles

Branchement Conditionnel

- Instruction conditionnel : If (expression booléen) {bloc d'instructions A} .

- Instruction conditionnel avec clause else :

If (expression booléen) {bloc d'instructions A}
else {bloc d'instructions B}

  - Opérateur à trois opérandes :

(expression booléen) ? {bloc d'instructions A} : {bloc d'instructions B}

Remarque:

- La condition est une expression booléenne qui retourne true ou false.
- Les parenthèses sont obligatoires en Java autour de la condition.
- Les accolades sont obligatoires pour un ensemble d'instructions et facultatives pour une seule instruction.

- Instruction conditionnel multiple :

```
Switch (expression)
{
cas 1 : {bloc d'instruction, break}  // break sera détaillé dans les prochaine slide
cas 2 : {bloc d'instruction, break}  // break sera détaillé dans les prochaine slide
      ..
cas n : {bloc d'instruction, break}  // break sera détaillé dans les prochaine slide
default : {bloc d'instruction  //équivalent à else (si aucun cas n'est vérifier), Default est facultative
}
```

Les boucles

- En général : une boucle à un état initial, état final exprimé par un test d'arrêt et un bloc d'instructions qui mènent de l'état initial vers l'état final qu'on appel la convergence vers test d'arrêt, si l'opération de convergence vers test d'arrêt est mal conçu, il y a un risque d'avoir une boucle infinie !!
  - Répétition inconditionnelle : For (expression1, expression2, expression3)
    {bloc d'instruction}
    - Expression1 : état initial, se compose d'une ou plusieurs initialisations (séparées par des virgules) .
    - Expression2 : état final (test d'arrêt) est une expression booléenne .
    - Expression3 : convergence vers test d'arrêt, peut contenir plusieurs instructions séparées par des virgules .

Remarque:

- for (; ;); est une boucle infinie.
- Dans une boucle for (int i=0 ;... ;…), la portée de la variable i est limitée au bloc du for. Une fois la boucle finie, la variable i n'est plus accessible.

- Répétition conditionnelle
  - (la boucle tant que) : While (expression booléen) {bloc d'instruction}
  - (la boucle répéter .. Jusqu'à) : Do {bloc d'instruction} While (expression booléen)

  L'expression booléen est le test d'arrêt, l'état initial est généralement défini avant la boucle, la convergence vers le test d'arrêt doit être dans le bloc d'instruction .

- Structures de contrôle imbriqués .

Remarque :

- while (true) {bloc d'instruction}, do {bloc d'instruction} while (true), sont des boucles infinies
- La partie initialisation se compose d'une ou plusieurs initialisations (séparées par des virgules).
- La partie test est une expression booléenne. La partie incrémentation peut contenir plusieurs instructions séparées par des virgules.

break et continue

- Instructions liées aux boucles .
  - **break** : Interruption de l'itération en cours et passage à l'instruction qui suit la boucle.
  - **continue** : Interruption de l'itération en cours et retour au début de la boucle avec exécution de la partie incrémentation.

break et continue avec étiquette

- L'instruction break et continue ordinaire à l'inconvénient de ne sortir que du niveau (boucle ou switch) le plus interne. Dans certain cas, cela s'avère insuffisant.

- Java permet de faire suivre break/continue d'une étiquette qui doit alors figurer devant la structure dont on souhaite sortir.

```
repet : while (.............)
{
        for (.........)
        {
                ....
                break(continue) repet
                ....
        }
}
```

Boucle infini

- Les boucles infinies sont des boucles qui s'exécutent à l'infini sans aucune condition de rupture.

- Utilisation d'une boucle For :

```
for (;;)
{
// code...
}
```

- Utilisation d'une boucle While :

```
while(true){
// code...
}
```

- Utilisation d'une boucle Do-while :

```
Do { } while (true)
```

INPUT / OUTPUT

Affichage sur la console

- System.out.println(chaîne) ou System.out.print(chaîne) .
  - System: C'est une classe présente dans le package java.lang.
  - Out: est la variable statique de type PrintStream présente dans la classe System.
  - println(): est la méthode présente dans la classe PrintStream.
- Println : print + saut de ligne .
- chaîne est ici :
  - une constante chaîne de caractères (String) .
  - une expression convertie automatiquement en String si c'est possible .
  - une concaténation d'objets de type String ou convertis en String séparés par le symbole + .

Lecture au clavier

- Il n'y a pas un moyen simple pour lire à partir de clavier !
- Il faut utiliser la classe Scanner :
  - classe comprenant beaucoup de méthodes .

Illustration de lecture au clavier

```
import java.util.Scanner

public class CalculatorPanel {

        public static void main(String[] args) {

                int i;

                System.out.println("Entez  un entier: ");

                Scanner  clavier = new Scanner(System.in);

                i = clavier.nextInt();

                System.out.println("Vous  avez entré : "+i);

                }

}
```

Méthode de Scanner

- nextInt() - gets the next integer
- nextBoolean() - gets the next Boolean
- nextDouble() - gets the next double
- nextFloat() - gets the next float
- nextShort() - gets the next short
- next() - gets the next string (a line can have multiple strings separated by space)
- nextLine() - gets the next line

Exercice 01

- Écrire un programme qui convertit les degrés Celsius en Fahrenheit
- Demande à l'utilisateur de rentrer un nombre en degré celsius
- Calcul en utilisant la formule

  y=32+(9/5)*x (x est la température en ° celsius)
- Affiche la valeur convertie à l'écran

```
import java.util.Scanner

public class CalculatorPanel {

        public static void main(String[] args) {

                int i;

                System.out.println("Entez un entier: ");

                Scanner clavier = new Scanner(System.in);

                i = clavier.nextInt();

                System.out.println("Vous avez entré : "+i);

                }

}
```

Structures de données

Tableaux et matrices

- Principe assez proche du C++ .
- Plus faciles à manipuler qu'en C .
- Les tableaux peuvent être des ensembles ordonnées de :
  - types de bases (int, char, …) .
  - d'objets !

Tableaux et matrices

- Les tableaux sont des objets .
- Créés par new :
  - Ils sont enregistrés dans le tas et les variables contiennent des références aux tableaux .
- Une variable d'instance : public final int length .
- Mais des objets particuliers, donc syntaxe particulière pour :
  - la déclaration des tableaux .
  - l'initialisation .

Tableaux et matrices

1. Déclaration

- syntaxe : type [ ] nomDuTableau .

  Exemple : int [ ] tableau .

  - La déclaration ne fait que déclarer le tableau mais ne crée pas le tableau en mémoire.
  - La variable déclarée permet de référence tout tableau de même type.

L'opérateur new permet de créer le tableau et d'allouer la mémoire

int [] tableau .

tableau = new int[50] .

- La taille donnée est fixe, elle ne peut plus être modifié, sauf par écrasement de tableau (réinstancier le tableau, mais il y aura une perte d'information) .

Tableaux et matrices

2. Création

- La création d'un tableau par new :
  - Alloue la mémoire nécessaire en fonction :
    - Du type de tableau .
    - De la taille spécifiée .

- Initialise le contenu du tableau :
    - Type numérique : 0 .
    - Type complexe : null .

Tableaux et matrices

3. Accès aux éléments

- principe : tableau[expression] .
- Les éléments d'un tableau de taille n sont numérotés de 0 à n-1 .
- expression est une expression donnant une valeur entière .
- Java vérifie dynamiquement que la valeur de l'élément existe sinon

erreur : ArrayIndexOutOfBoundsException

Tableaux et matrices

4. Attribut du tableau .

- Il y a des attributs puisque c'est un objet !
- tableau.length donne la taille du tableau .

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

monTableau.length → 10 taille du tableau

monTableau.length – 1 → indice max du tableau

Tableaux et matrices

5. Autres possibilité pour crée un tableau

- on peut donner explicitement la liste de ses éléments à la déclaration (liste de valeurs entre accolades) .

```
int [ ] t = {17, 8, 9, 23, 17} ;

char [ ] voyelles = { 'a', 'e', 'i', 'o', 'u', 'y' } ;
```

- L'allocation mémoire (équivalent de new) est réalisée implicitement .

Tableaux et matrices

5. Matrices (tableau à 2 dimensions) et tableaux multi dimensions .

- Déclaration et création .

syntaxe : type [ ] …. [ ] nomDuTableau ;

n fois (dimension)

- Accès aux éléments .

syntaxe : nomDuTableau [expression][expression]…[expression]

n fois (dimension)

expression est une expression donnant une valeur entière .

Tableaux et matrices (Bonus)

- Java dispose d'une classe Arrays qui propose des méthodes pour trier et rechercher des éléments dans des tableaux .

- import java.util.Arrays ; pour l'utiliser .

Chaînes de caractères

- Chaîne de caractère **String** :
  - Ce n'est pas un type primitif, c'est **une classe** (d'où la présence de la majuscule) .
  - Nous reviendrons plus tard sur les classes .
  - Ce sont des objets immutables (impossible à modifier) .
  - Comme c'est une classe, on utilise le mot-clef new pour créer un objet, on parle d'une instance de classe de type String .

    String str; str = "ceci est une phrase "; // 1er méthode .

    String str = new String() ; str = "ceci est une phrase" ; // 2ième méthode .

    String str = "ceci est une phrase "; // 3ième méthode .

    String str = new String ("ceci est une phrase "); // 4ième méthode .
  - la chaîne de caractères sont mis entre guillemets .

Chaînes de caractères

- Opération sur les chaînes de caractères :
  - On utilise chaine1.equals(chaine2) pour vérifier l'égalité .
  - Concaténation : opérateur + .
  - Taille d'une chaîne chaine.length() .
  - Affichage System.out.println(chaine) .
- Opération sur les chaînes de caractères (suite) :
  - Accéder à un caractère de rang fixé chaine.charAt(rang) .
  - position d'une sous-chaîne à l'intérieur d'une chaîne donnée chaine.indexOf(sous-chaine) .
  - Convertir une chaîne de caractère en tableau de carctère : chr_tab[]= chaine.toCharArray() .
  - Convertir Caractère -> String => chaine = String.ValueOf(chr) .

Chaînes de caractères

- La méthode toString est implicite lorsque :

  - on affiche un objet : System.out.println(object) .

  - on concatène un objet à une chaîne :

    String s = objet + "123" ;

    est équivalent à

    String s = objet.toString() + "123" ;

- En règle générale, dès que Java s'attend à une chaîne de caractère, il convertit automatiquement l'objet en chaîne de caractère .

Types références

Séquance	Description
\012	Caractère en code octal
\uxxxx	Caractère en code héxadécimal (unicode)
\’	‘
\”	“
\\	\
\r	Retour chariot
\n	Saut de ligne
\f	Saut de page
\t	Tabulation horizontale
\b	Backspace

String : .equals vs « == »

```
String s1,s2,s3,ch;

   ch = "abcdef";

   s1 = ch;

   s2 = "abcdef";

   s3 = new String("abcdef".toCharArray( ));

System.out.println("s1="+s1);

   System.out.println  ("s2="+s2);

   System.out.println  ("s3="+s3);

   System.out.println  ("ch="+ch);
```

```
if( s1 == ch ) System.out.println ("s1=ch");

  else System.out.println ("s1<>ch");

if( s1 == s3 ) System.out.println ("s1=s3");

  else System.out.println ("s1<>s3");

if( s1.equals(s2) ) System.out.println ("s1 même val. que s2");

  else System.out.println ("s1 différent de s2");

if( s1.equals(s3) ) System.out.println ("s1 même val. que s3");

  else System.out.println ("s1 différent de s3");

if( s1.equals(ch) ) System.out.println ("s1 même val. que ch");

  else System.out.println ("s1 différent de ch");
```

Affichage sur console

```
s1= abcdef
s2= abcdef
s3= abcdef
ch= abcdef
s1=ch
s1<>s3
s1 même val. que s2
s1 même val. que s3
s1 même val. que ch
```

Vector (Vecteur)

- Nous pouvons dire que c’est un tableau dynamique : taille dynamique .
- Utilisation de paquetage : java.util.Vector .
- Un objet de classe Vector peut "grandir" automatiquement d'un certain nombre de cellules pendant l'exécution.
- Vector est utilisé pour les variable références (Object), donc même String .

Opération Vector (Vecteur) : méthodes

- void addElement(Object obj)
- void clear( )
- Object elementAt(int index)
- int indexOf(Object elem)
- Object remove(int index)
- void setElementAt(Object obj, int index)
- int size( )

```
static void afficheVector(Vector  vect)
//affiche un vecteur de String
 {
   System.out.println(  "Vector taille = " + vect.size( ) );
   for ( int i = 0; i<= vect.size( )-1; i++ )
        System.out.println(  "Vector[" + i + "]=" + (String)vect.elementAt(  i ) );
 }

static void VectorInitialiser( )
// initialisation du vecteur de String
{    Vector table = new Vector( );
      String str = "val:";
      for ( int i = 0; i<=5; i++ )
            table.addElement(str  + String.valueOf( i ) );
      afficheVector(table);
}
```

Affichage sur console

```
Vector taille = 6
Vector[0] = val:0
Vector[1] = val:1
Vector[2] = val:2
Vector[3] = val:3
Vector[4] = val:4
Vector[5] = val:5
```

Les listes chaînées : LinkedList

- Une liste chaînée est un ensemble ordonné d'éléments de même type (structure de donnée homogène) .
- L'accès aux éléments est réalisé d'une façon séquentielle .
- Une liste chaînée peut être unidirectionnel(simplement chaînée) bidirectionnel (doublement chaînée) .
- Les opérations de base dans une liste chaînée : avancer, (reculer si bidirectionnel), ajouter, modifier, supprimer .

Les listes chaînées : LinkedList

- C’est de la gestion dynamique de mémoire .
- Utilisation de paquetage: java.util.LinkedList .
- en Java une implémentation de la liste chaînée bi-directionnelle .
- Contrairement au Vector, LinkedList est utilisé pour les variable primitif et références (Object), donc même String .

Opérations sur LinkedList (Vecteur) : Méthodes

- void addFirst(Object obj)
- void addLast(Object obj)
- void clear( )
- Object get(int index)
- int indexOf(Object elem)
- Object remove(int index)
- Object set(int index , Object obj)
- int size( )

```
import java.util.LinkedList;
class ApplicationLinkedList {
 static void afficheLinkedList (LinkedList  liste )
 //affiche une liste de chaînes
 {
  System.out.println("liste  taille = "+liste.size());
  for ( int i = 0 ; i <= liste.size( )-1 ; i++ )
       System.out.println("liste("  + i + ")=" + (String)liste.get(i));
 }
 static void LinkedListInitialiser( )
 {    LinkedList liste = new LinkedList( );
     String str = "val:";
     for ( int i = 0 ; i <= 5 ; i++ )
     liste.addLast(  str + String.valueOf( i ) );
     afficheLinkedList(liste);
 }
 static void main(String[] args)
 {
  LinkedListInitialiser( );
 }
}
```

Affichage sur console

liste taille = 6
liste(0) = **val:**0
liste(1) = **val:**1
liste(2) = **val:**2
liste(3) = **val:**3
liste(4) = **val:**4
liste(5) = **val:**5

Exercice 2

- Jeu de la devinette :
  - L'ordinateur choisit un nombre entier entre 1 et 1000 .
  - Le joueur doit le deviner en maximum 10 essais .
  - Pour chaque tentative, on indique si le nombre à trouver est plus grand ou plus petit que le nombre rentré par l'utilisateur .

Exercice 3

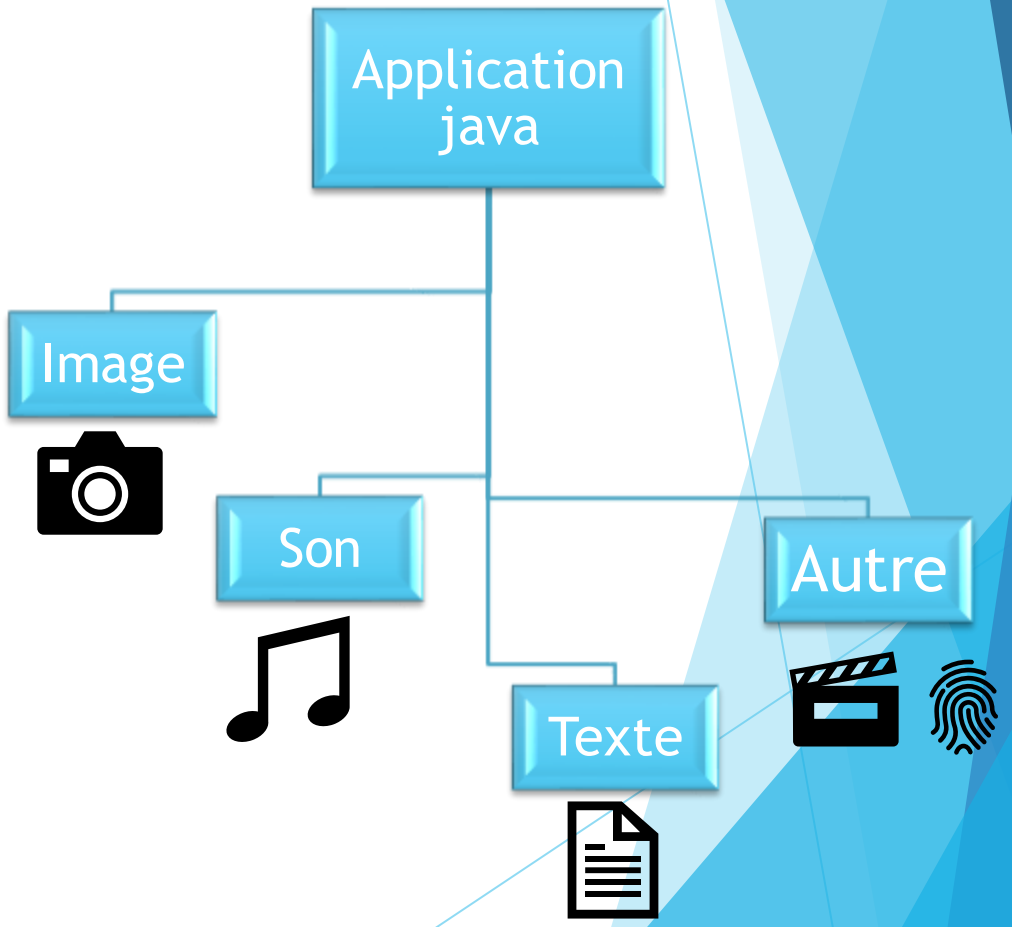
```
int somme = 0;

for (int i = 0; i < tab.length; i++)

    {

    if (tab[i] == 0) break;

    if (tab[i] < 0) continue; somme += tab[i];

    }

System.out.println(somme);
```

Qu'affiche ce code avec le tableau [1 ; -2 ; 5 ; -1 ; 0 ; 8 ; -3 ; 10 ?]

Flux de données

Flux et fichiers

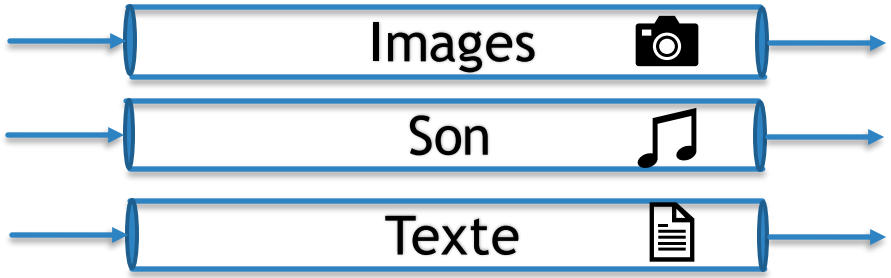
- Un programme utilise ses données internes, mais il est très nécessaire d'en chercher d'extérieur en INPUT (entrée) , on dit **lire** .
- Les données peuvent être des son, des images des texte provenant de plusieurs sources (périphériques, autres applications) .
- En plus de la console, le programme peut après traitement avoir besoin de faire son OUTPUT (sortie) (fichier, périphérique..), on dit **écrire** .



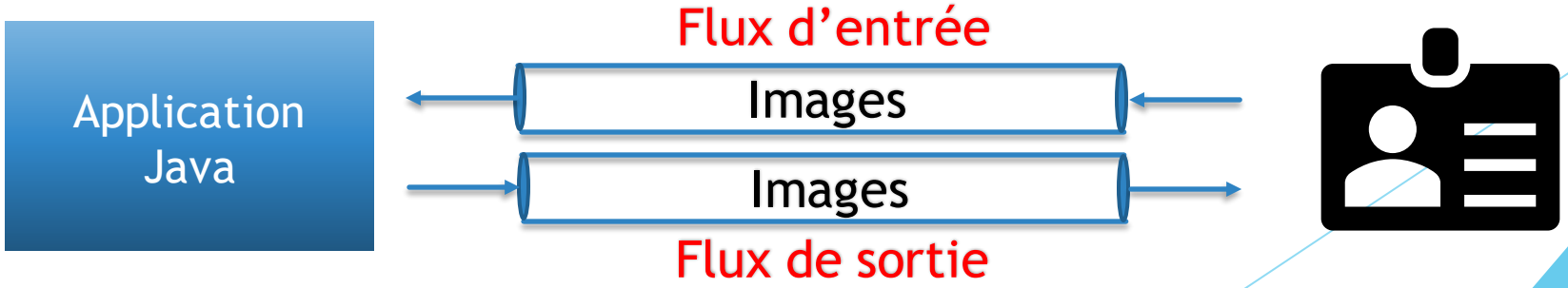
Flux et fichiers

- En java , En Java, toutes ces données sont échangées en entrée et en sortie à travers des flux (Stream).

- Un flux est une sorte de tuyau de transport séquentiel de données.



- Un flux est unidirectionnel : il y a donc des flux d'entrée et des flux de sortie .



Flux et fichiers

- Utilisation de package : java.io.* (* indique que toutes les classes du package sont concerné) .
- Il y a deux familles de flux :
  - La famille des flux de caractères (caractères 16 bits) .
  - La famille des flux d'octets (information binaires sur 8 bits) .
- Java est un LOO (Langage Orienté Objet) les différents flux d'une famille sont des classes dont les méthodes sont adaptées au transfert et à la structuration des données selon la destination de celles-ci.
- Pour lire ou écrire des données binaires (sons, images,...) => une des classes de la famille des flux d'octets.
- Des données de type caractères => systématiquement l'un des classes de la famille des flux de caractères.

Les flux de données prédéfinis

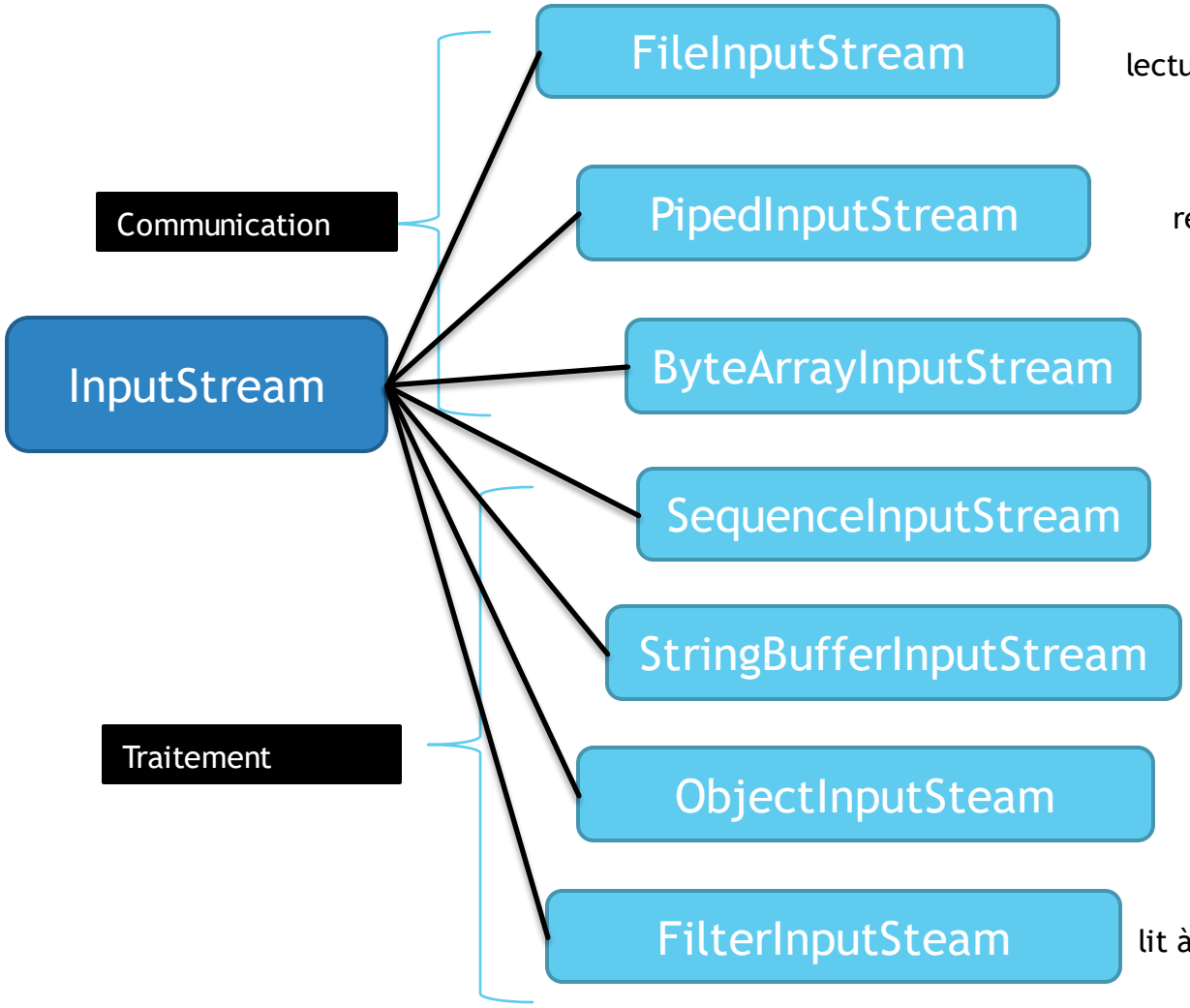
Il existe 3 flux prédéfinis :

- entrée standard System.in (instance de InputStream) .
- sortie standard System.out (instance de PrintStream) .
- sortie standard d'erreurs System.err(instance de PrintStream) .

Flux et fichiers

- Les flux sont décomposés en **deux grandes familles** .
- Les flux d'octets :
  - classes abstraites **InputStream** et **OutputStream** et leurs sous-classes respectives .
- Les flux de caractères :
  - classes abstraites **Reader** et **Writer** et leurs sous-classes respectives.
- Il existe de nombreuses classes représentant les flux :
  - il n'est pas toujours aisé de se repérer.
- Certains types de flux agissent sur la façon dont sont traitées les données qui transitent par leur intermédiaire :
  - E / S bufferisées, traduction de données, …
- Il est possible de combiner ces différents types de flux afin d'avoir une gestion souhaitée pour les E / S.

Flux et fichiers



lecture de fichiers octets par octets.

récupère des données provenant d'un flux de sortie (cf. PipedOutputStream).

lit des données dans un tampon structuré sous forme d'un array.

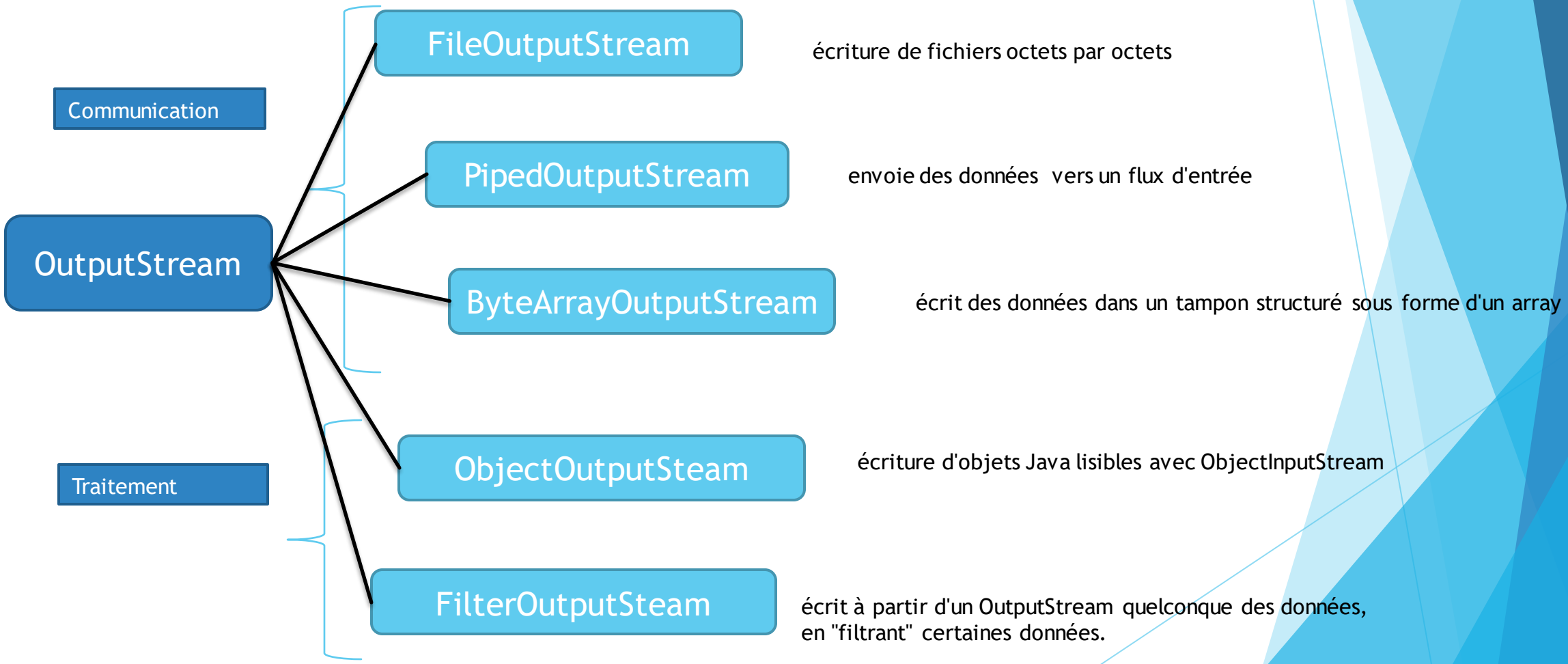
concaténation d'une énumération de plusieurs flux d'entrée en un seul flux d'entrée

lecture d'une String (Sun déconseille son utilisation et préconise son remplacement par StringReader).

lecture d'objets Java

lit à partir d'un InputStream quelconque des données, en "filtrant" certaines données

Flux et fichiers



Les flux d’octets :
La classe InputStream

- Un **InputStream** est un flux de lecture d’octets.
- **InputStream** est une classe abstraite.
  - Ses sous-classes concrètes permettent une mise en œuvre pratique.
  - Par exemple, **FileInputStream** permet la lecture d'octets dans un fichier.
- Les méthodes principales qui peuvent être utilisées sur un InputStream sont :
  - public abstract int read () throws IOException qui retourne l’octet lu ou -1 si la fin de la source de données est atteinte.
  - int read (byte[ ] b) qui emplit un tableau d’octets et retourne le nombre d’octets lus .
  - int read (byte [] b, int off, int len) qui emplit un tableau d’octets à partir d ’une position donnée et sur une longueur donnée .
  - void close () qui permet de fermer un flux, Il faut fermer les flux dès qu'on a fini de les utiliser. Un flux ouvert consomme des ressources du système d'exploitation.
  - int available () qui retourne le nombre d’octets prêts à être lus dans le flux, cette fonction permet d'être sûr qu'on ne fait pas une tentative de lecture bloquante. Au moment de la lecture effective, il se peut qu’il y ait plus d’octets de disponibles.
  - long skip (long n) qui permet d’ignorer un certain nombre d’octets en provenance du flot. Cette fonction renvoie le nombre d'octets effectivement ignorés .

La classe OutputStream (1)

- Un **OutputStream** est un flot d'écriture d'octets.
- La classe **OutputStream** est abstraite.
- Les méthodes principales qui peuvent être utilisées sur un **OutputStream** sont :
  - public abstract void write (int) throws IOException qui écrit l'octet passé en paramètre .
  - void write (byte[] b) qui écrit les octets lus depuis un tableau d'octets .
  - void write (byte [] b, int off, int len) qui écrit les octets lus depuis un tableau d 'octets à partir d 'une position donnée et sur une longueur donnée .
  - void close () qui permet de fermer le flux après avoir éventuellement vidé le tampon de sortie .
  - flush () qui permet de purger le tampon en cas d'écritures bufferisées.

Les flux d’octets

- Classe DataInputStream :
  - sous classes de InputStream permet de lire tous les types de base de Java.
- Classe DataOutputStream :
  - sous classes de OutputStream permet d'écrire tous les types de base de Java.
- Classes ZipOutputStream et ZipInputStream :
  - permettent de lire et d'écrire des flux dans le format de compression zip.

Empilement de flux filtrés (1)

- En Java, chaque type de flux est conçu pour la réalisation une tâche spécifique.
- Lorsque le programmeur souhaite un flux qui ait un comportement plus complexe, il "empile", à la façon des poupées russes, plusieurs flux ayant des comportements élémentaires.
- On parle de « flux filtrés ».
- Concrètement, il s'agit de passer, dans le constructeur d'un flux, un autre flux déjà existant pour combiner leurs caractéristiques.

Exemple :

- FileInputStream : permet de lire depuis un fichier mais ne sait lire que des octets.
- DataInputStream :permet de combiner les octets pour fournir des méthodes de lecture de plus haut niveau (pour lire un double par exemple), mais ne sait pas lire depuis un fichier.
- Une combinaison des deux permet de combiner leurs caractéristiques .

Empilement de flux filtrés (2)

```
FileInputStream fic = new FileInputStream ("fichier");
DataInputStream din = new DataInputStream (fic);
double d = din.readDouble ();
```

Lecture bufferisée de nombres depuis un fichier

```
DataInputStream din = new DataInputStream(new BufferedInputStream(
                                    new FileInputStream ("monfichier")));
```

Lecture de nombre dans un fichier au format zip

```
ZipInputStream zin = new ZipInputStream (
                                    new FileInputStream ("monfichier.zip"));
DataInputStream din = new DataInputStream (zin);
```

Flux de fichiers à accès direct (1)

- La classe **RandomAccessFile** :
  - permet de lire ou d'écrire dans un fichier à n'importe quel emplacement (par opposition aux fichiers à accès séquentiels).
- Elle implémente les interfaces **DataInput** et **DataOutput** .
  - permettent de lire ou d'écrire tous les types Java de base, les lignes, les chaînes de caractères ascii ou unicode, etc ...
- Un fichier à accès direct peut être :
  - ouvert en lecture seule (option "r") ou
  - en lecture / écriture (option "rw").
- Ces fichiers possèdent un pointeur de fichier qui indique constamment la donnée suivante.
  - La position de ce pointeur est donnée par long getFilePointer () et celui-ci peut être déplacé à une position donnée grâce à seek (long off).

Les flux de caractères

- Ce sont des sous-classes de Reader et Writer.
- Ces flux utilisent le codage de caractères Unicode.
- Exemples:
  - conversion des caractères saisis au clavier en caractères dans le codage par défaut

```
InputStreamReader in = new InputStreamReader (System.in);
```

  - Conversion des caractères d ’un fichier avec un codage explicitement indiqué

```
InputStreamReader in = new InputStreamReader (
    new FileInputStream ("chinois.txt"), "ISO2022CN");
```

Les flux de caractères

- Pour écrire des chaînes de caractères et des nombres sous forme de **texte**
  - on utilise la classe PrintWriter qui possède un certain nombre de méthodes print (...) et println (...).
- Pour lire des chaînes de caractères sous forme texte, il faut utiliser, par exemple,
  - BufferedReader qui possède une méthode readLine() .
    - Pour la lecture de nombres sous forme de texte, il n'existe pas de solution toute faite : il faut par exemple passer par des chaînes de caractères et les convertir en nombres.

Exemple lecture de fichier

```
import java.io.*;
public class LireLigne
{
  public static void main(String[] args)
    {
      try
       {
         FileReader fr=new
FileReader("c:\\windows\\system.ini");
         BufferedReader br= new BufferedReader(fr);
         while (br.ready())
            System.out.println(br.readLine());
         br.close();
        }
     catch (Exception e)
       {System.out.println("Erreur "+e);}
    }
}
```

A partir du chemin d'un dossier ou d'un fichier, on peut créer un objet FileReader puis à partir ce celui-ci, on crée un BufferedReader

Dans l'objet BufferedReader on dispose d'une méthode readLine()

Exemple ecriture dans un fichier

```
import java.io.*;
public class Ecrire
{
  public static void main(String[] args)
    {
      try
       {
         FileWriter fw=new FileWriter("c:\\temp\\essai.txt");
         BufferedWriter bw= new BufferedWriter(fw);
         bw.write("Ceci est mon fichier");
         bw.newLine();
         bw.write("Il est à moi...");
         bw.close();
       }
      catch (Exception e)
       { System.out.println("Erreur "+e);}
}}
```

A partir du chemin d'un dossier ou d'un fichier, on peut créer un objet FileWriter puis à partir ce celui-ci, on crée un BufferedWriter

Attention, lorsque l'on a écrit, il ne faut pas oublier de fermer le fichier

La sérialisation

- La sérialisation consiste à prendre un objet en mémoire et à en sauvegarder l'état sur un flux de données (vers un fichier, par exemple).
- Ce concept permet aussi de reconstruire, ultérieurement, l'objet en mémoire à l'identique de ce qu'il pouvait être initialement.
- La sérialisation peut donc être considérée comme une forme de persistance des données.
- Deux classes ObjectInputStream et ObjectOutputStream proposent, respectivement, les méthodes readObject et writeObject .
- Par défaut, les classes ne permettent pas de sauvegarder l'état d'un objet sur un flux de données. Il faut implémenter l'interface java.io.Serializable.
- Il faut que la classe n’ait pas supprimé le constructeur par défaut .

Exemple de sérialisation

```
void sauvegarde(String s) {
try  {FileOutputStream  f = new FileOutputStream(new  File(s));
       ObjectOutputStream  oos = new ObjectOutputStream(f);
       oos.writeObject(this);
        oos.close();}
     catch (Exception e)
     { System.out.println("Erreur  "+e);}
}

static Object relecture(String  s) {
try  {FileInputStream  f = new FileInputStream(new  File(s));
       ObjectInputStream  oos = new ObjectInputStream(f);
       Object o=oos.readObject();
        oos.close();
       return o;}
     catch (Exception e)
     { System.out.println("Erreur  "+e);
       return null;}
}
```

La gestion des fichiers

- La gestion de fichiers se fait par la classe File.
- Cette classe implémente des méthodes qui permettent d'interroger le système de gestion de fichiers du système d'exploitation.
- Un objet de la classe File peut représenter un fichier ou un répertoire.
  - File (String name)
  - File (String path, String name)
  - File (File dir, String name)
  - boolean isFile( ) / boolean isDirectory( )
  - boolean mkdir( )
  - boolean exists( )
  - boolean delete( )
  - boolean canWrite( ) / boolean canRead( )
  - File getParentFile( )
  - long lastModified( )

La gestion des fichiers

```
import java.io.*;
public class Listeur
{
 public static void main(String[] args)
  {
   litrep(new File("."));
  }
 public static void litrep(File rep)
 {
   if (rep.isDirectory())
    { //liste les fichier du répertoire
      String t[]=rep.list();
      for (int i=0;i<t.length;i++)
        System.out.println(t[i]);
    }
  }
}
```

Les objets et classes relatifs à la gestion des fichiers se trouvent dans le package java.io

A partir du chemin d'un dossier ou d'un fichier, on peut créer un objet File : ici on va lister le répertoire courant (« . »)

Les méthodes isFile() et isDirectory() permettent de déterminer si mon objet File est une fichier ou un répertoire

La gestion des fichiers

```
import java.io.*;
public class Listeur
{
  public static void main(String[] args)
  { litrep(new File( "c:\\"));}

  public static void litrep(File rep)
  {
    File r2;
    if (rep.isDirectory())
    {String t[]=rep.list();
      for (int i=0;i<t.length;i++)
      {
        r2=new File(rep.getAbsolutePath()+"\\"+t[i]);
        if (r2.isDirectory())  litrep(r2);
        else System.out.println(r2.getAbsolutePath());
      }}
 }}
```

Pour chaque fichier, on regarde s'il est un répertoire.

Le nom complet du fichier est rep\fichier

Si le fichier est un répertoire litrep s'appelle récursivement elle-même

JDBC : Java DataBase Connectivity

- **JDBC** est une API (Application Programming Interface) qui permet d'exécuter des instructions SQL
- JDBC fait partie du JDK (Java Development Kit)
- Toutes les classes et interfaces sont dans le paquetagejava.sql :

```
import java.sql.*
```

- Structure d'un programme
  1. On établit une connexion avec une source de données
  2. On effectue des requêtes
  3. On utilise les données obtenues pour des affichages, des traitements statistiques etc.
  4. On met à jour les informations de la source de données
  5. On termine la connexion