

JAVA

m2formation.fr

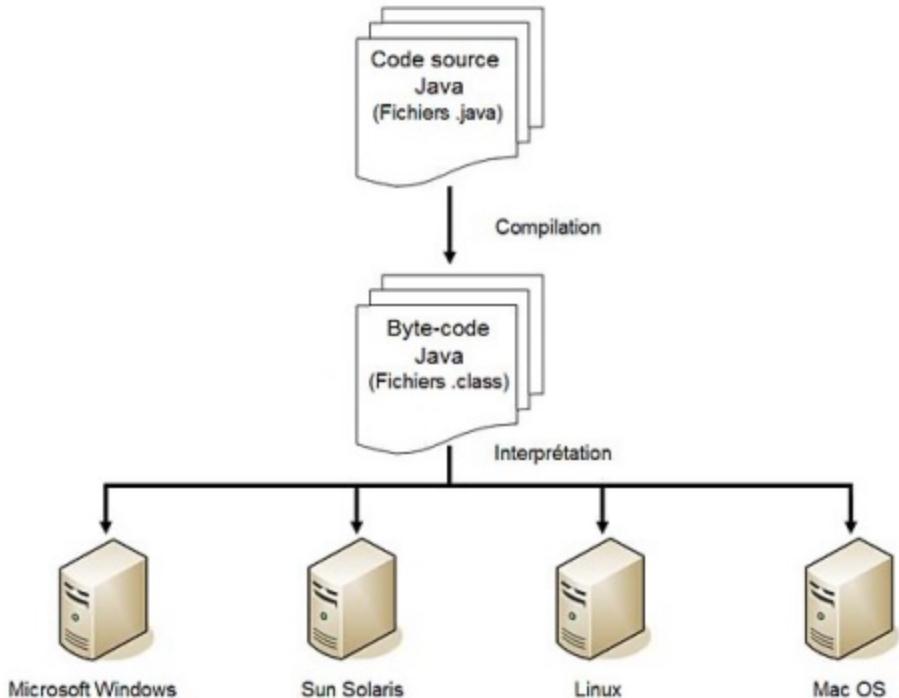


Sommaire

1. L'écosystème Java
2. Les IDE : Eclipse, IntelliJ.
3. Variables
4. Opérateurs
5. Instructions
6. Tableaux

1. L'écosystème Java

La plateforme java



WORA (*Write Once, Run Anywhere* : écrire une fois, exécuter partout).

Historique

- En 1991, la société Sun Microsystems démarre un projet d'informatique embarquée.
- Le langage utilisé par les ingénieurs de Sun est le **C++**.
- Ce langage est inadapté au développement de leur projet (problème gestion mémoire, sécurité, etc.)
- Ils ont souhaités créer un nouveau langage plus adapté, orienté objet, inspiré du C++.
- Appellation c++-, puis Oak et enfin **Java**.
- En 1995, la première version du kit de développement logiciel en Java (JDK).
- La compilation du code source Java ne donne pas, comme c'est le cas avec beaucoup d'autres langages, un exécutable natif, mais un format de fichier spécifique, appelé **bytecode**.
- Il faut que chaque machine qui souhaite travailler avec du code java, installe une **JVM** (Java Virtuel Machine).
- La **JVM** va se charger d'interpréter le bytes-code et lancer le programme.

Le langage Java

Les caractéristiques :

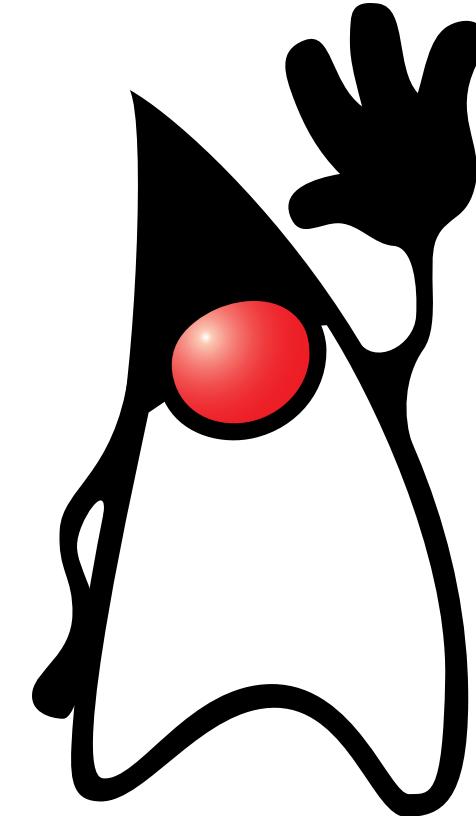


Le langage Java

Les dates importantes :

- **1995 : Mai** : premier lancement commercial du JDK 1.0
- **1996 : Janvier** : JDK 1.0.1 - **Septembre** : Lancement du JDC
- **1997 : Java Card 2.0** - **Février** : JDK 1.1
- **1998 : Décembre** : lancement de J2SE 1.2 et du JCP - Personal Java 1.0
- **1999 : Décembre** : lancement J2EE 1.2
- **2000 : Mai** : J2SE 1.3
- **2001 : J2EE 1.3**
- **2002 : Février** : J2SE 1.4
- **2003 : J2EE 1.4**
- **2004 : Septembre** : J2SE 5.0
- **2005 : Lancement du programme Java Champion**
- **2006 : Mai** : Java EE 5 - **Décembre** : Java SE 6.0
- **2008 : Décembre** : JavaFX 1.0
- **2009 : Février** : JavaFX 1.1 - **Juin** : JavaFX 1.2 - **Décembre** : Java EE 6
- **2010 : Janvier** : rachat de Sun Microsystems par Oracle - **Avril** : JavaFX 1.3
- **2011 : Juillet** : Java SE 7 - **Octobre** : JavaFX 2.0
- **2012 : Août** : JavaFX 2.2
- **2013 : Juin** : Java EE 7
- **2014 : Mars** : Java SE 8, JavaFX 8
- **2017 : Septembre** Java SE 9, Java EE 8
- **2018 : Mars** : Java SE 10 - **Septembre** : Java SE 11

Duke la mascotte de Java :



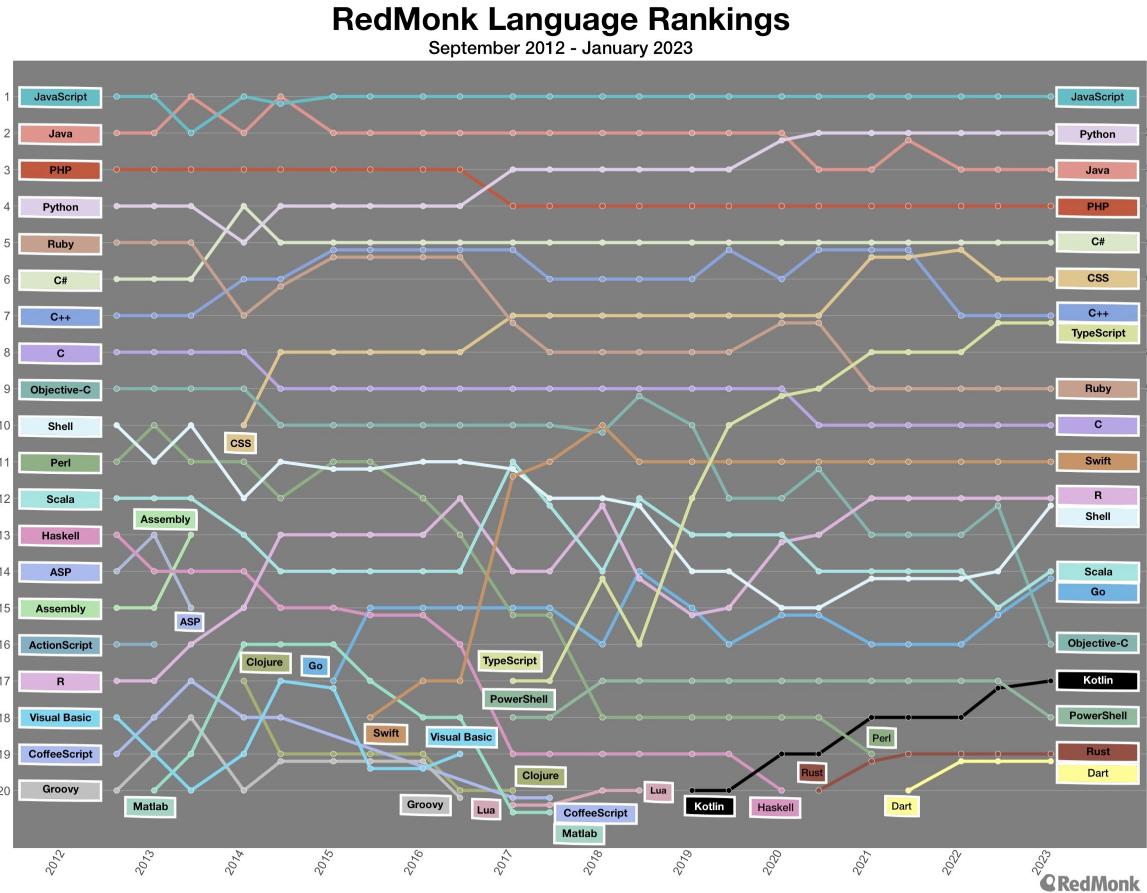
Le langage Java

Java est compilé	Le source est compilé en pseudo code ou bytecode puis exécuté par un interpréteur Java : la Java Virtual Machine (JVM). Ce concept est à la base du slogan de Sun pour Java : WORA (Write Once, Run Anywhere : écrire une fois, exécuter partout). En effet, le bytecode, s'il ne contient pas de code spécifique à une plate-forme particulière peut être exécuté et obtenir quasiment les mêmes résultats sur toutes les machines disposant d'une JVM.
Java est portable : il est indépendant de toute plate-forme	Il n'y a pas de compilation spécifique pour chaque plate forme. Le code reste indépendant de la machine sur laquelle il s'exécute. Il est possible d'exécuter des programmes Java sur tous les environnements qui possèdent une Java Virtual Machine. Cette indépendance est assurée au niveau du code source grâce à Unicode et au niveau du bytecode.
Java est orienté objet.	Comme la plupart des langages récents, Java est orienté objet. Chaque fichier source contient la définition d'une ou plusieurs classes qui sont utilisées les unes avec les autres pour former une application. Java n'est pas complètement objet car il définit des types primitifs (entier, caractère, flottant, booléen,...).
Java assure la gestion de la mémoire	L'allocation de la mémoire pour un objet est automatique à sa création et Java récupère automatiquement la mémoire inutilisée grâce au Garbage collector qui restitue les zones de mémoire laissées libres suite à la destruction des objets.
Java est fortement typé	Toutes les variables sont typées et il n'existe pas de conversion automatique qui risquerait une perte de données. Si une telle conversion doit être réalisée, le développeur doit obligatoirement utiliser un cast ou une méthode statique fournie en standard pour la réaliser.
Java est multitâche	Il permet l'utilisation de threads qui sont des unités d'exécutions isolées. La JVM, elle même, utilise plusieurs threads.
Java est sûr	La sécurité fait partie intégrante du système d'exécution et du compilateur. Un programme Java planté ne menace pas le système d'exploitation. Il ne peut pas y avoir d'accès direct à la mémoire. L'accès au disque dur est réglementé dans une applet. Les applets fonctionnant sur le Web sont soumises aux restrictions suivantes dans la version 1.0 de Java : <ul style="list-style-type: none"> • Aucun programme ne peut ouvrir, lire, écrire ou effacer un fichier sur le système de l'utilisateur • Aucun programme ne peut lancer un autre programme sur le système de l'utilisateur • Toute fenêtre créée par le programme est clairement identifiée comme étant une fenêtre Java, ce qui interdit par exemple la création d'une fausse fenêtre demandant un mot de passe • Les programmes ne peuvent pas se connecter à d'autres sites Web que celui dont ils proviennent.
Java est économique	Le pseudo code a une taille relativement petite car les bibliothèques de classes requises ne sont liées qu'à l'exécution.

Environnement de développement

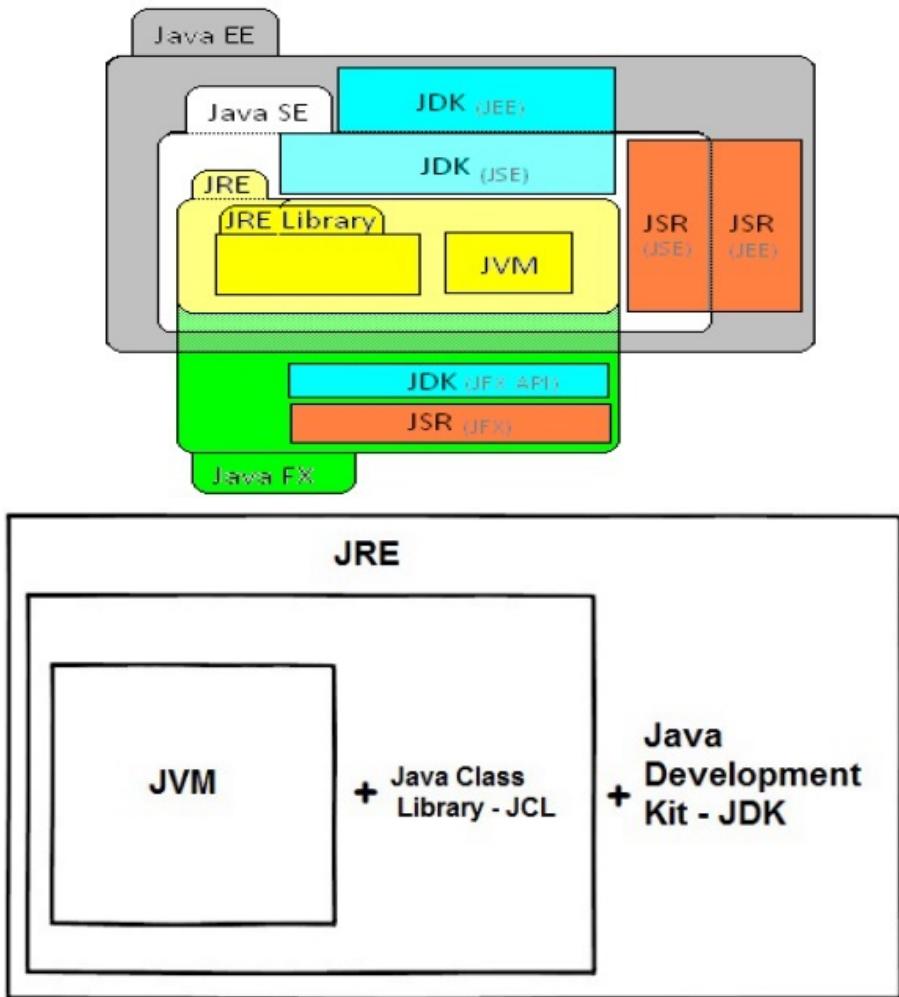
- L'environnement de développement Java est :
 - **Multiplateformes** (Linux, Windows, MacOS)
 - **Open Source**
 - **Gratuit**
 - Librement **redistribuable** dans vos applications
- Vous pouvez télécharger l'environnement de développement (JDK) à cette adresse :
 - <https://www.oracle.com/java/technologies/downloads/>
- Vous pouvez trouver l'ensemble des documentations pour les différentes version de JAVA (SE, EE...) sur le site d'oracle : <https://docs.oracle.com/en/java/> (Attention le JDK proposé par ce site est soumis à licence commerciale.)
 - [Java SE](#)
 - [Java EE](#)
 - [Java ME](#)

java de nos jours



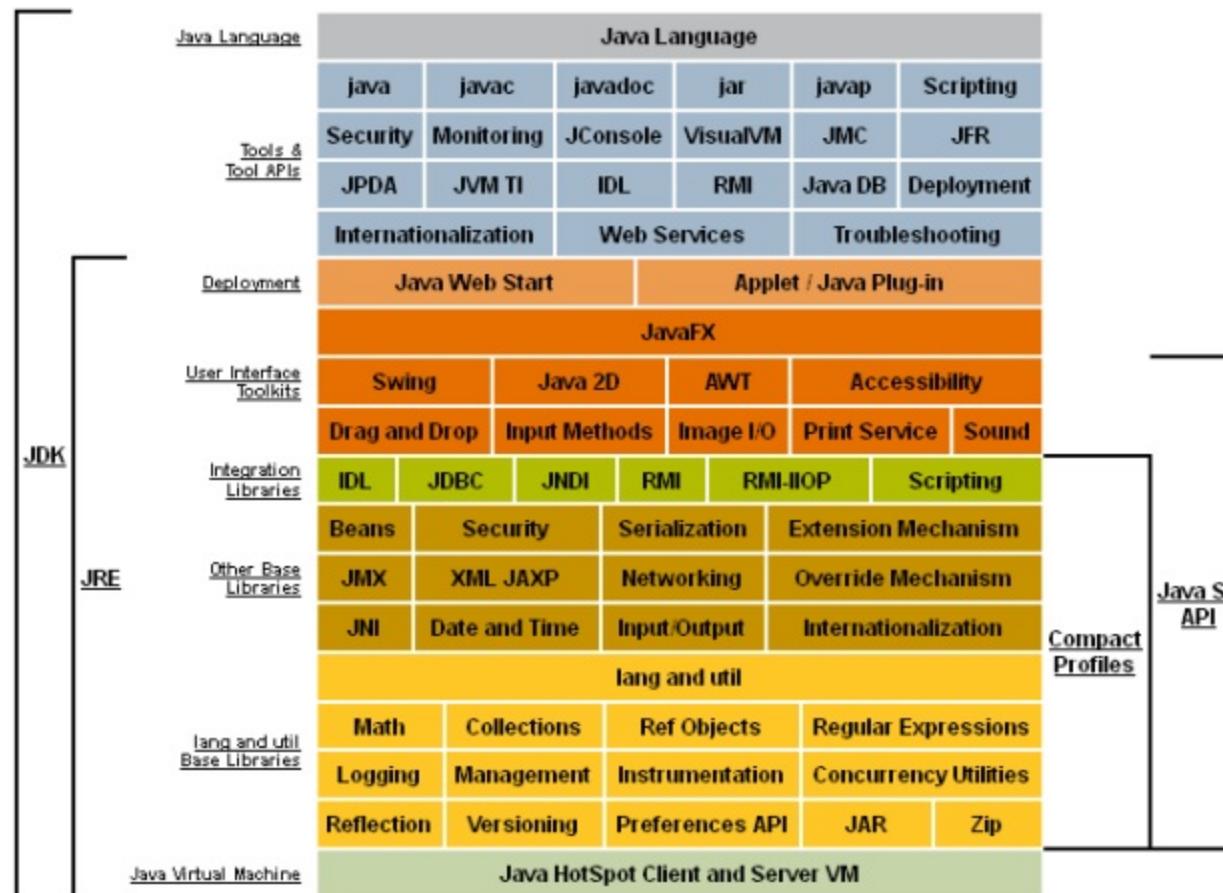
- La plateforme Java est une des plateformes de développement logiciel les plus adoptées par les entreprises.
- Plusieurs avantages :
 - Multithreading
 - Gestion de la mémoire
 - Évolutivité
 - Développement multiplateforme
 - Haute sécurité
- La plateforme Java se décompose en 3 plateformes :
 - **JSE (Java Standard Edition)** : **destinée aux ordinateurs de bureau**.
 - **JEE (Jakarta Entreprise Edition)** : **extension de JSE, destinée aux serveurs web**.
 - **JME (Java Micro Edition)** : **destinée aux appareils portables comme les smartphones**, partageant un noyau commun avec Java SE.

java de nos jours



- **JDK (Java Développement Kit)**: Le JDK désigne un ensemble de bibliothèques logicielles de base du langage de programmation Java, ainsi que les outils avec lesquels le code Java peut être compilé, transformé en bytecode destiné à la machine virtuelle Java. Il fournit les outils nécessaires au développement d'application java. Il comprend l'environnement d'exécution Java (JRE), un interpréteur / chargeur (Java), un compilateur (javac), un archiveur (jar), un générateur de documentation (Javadoc) et d'autres outils nécessaires au développement Java.
- **JRE (Java Runtime Environnement)** : Le JRE est l'environnement d'exécution Java. Il est intégré au JDK. Java RuntimeEnvironnement fournit la configuration minimale requise pour l'exécution d'une application Java. Il comprend la machine virtuelle Java (JVM), les classes principales et les fichiers de support.
- **JVM (Java Virtual Machine)**: La JVM est une partie très importante du JDK et du JRE car elle est contenue ou intégrée dans les deux. Quel que soit le programme Java que vous exécutez à l'aide de JRE ou de JDK, il est intégré à la machine virtuelle Java et celle-ci est chargée de l'exécution ligne par ligne du programme Java. Il est donc également appelé interpréteur.

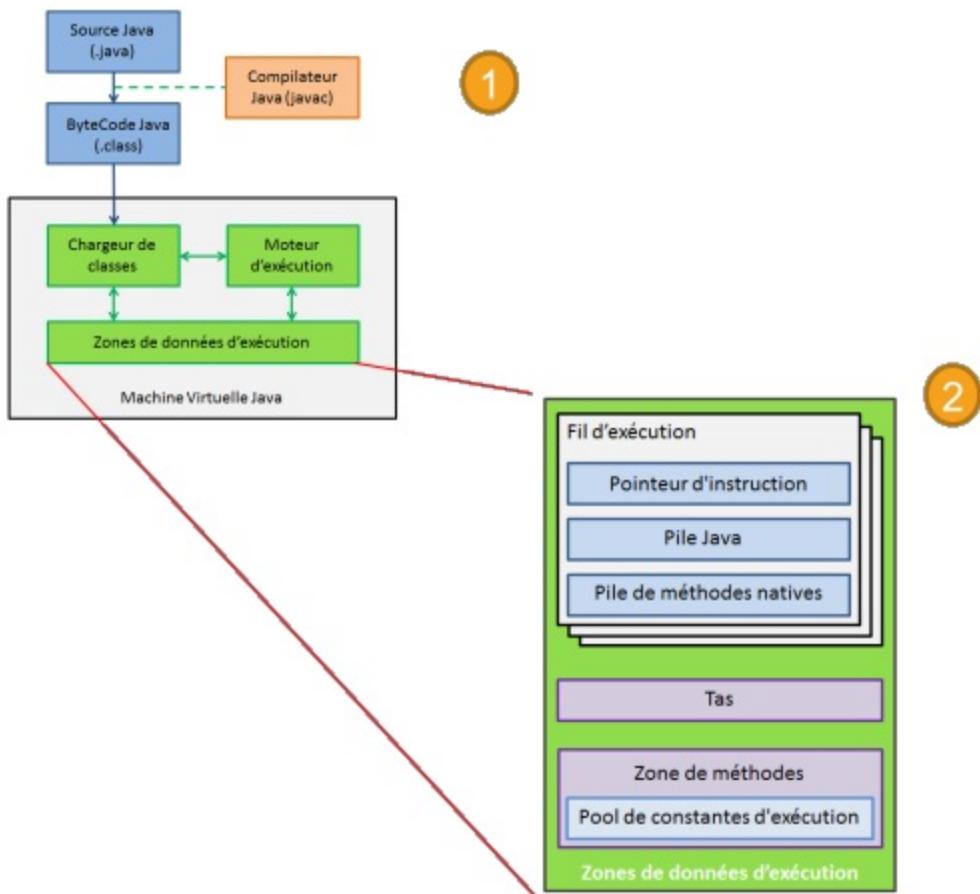
La plateforme java



JVM

- La **machine virtuelle Java** ou JVM (Java Virtual Machine) est un environnement d'exécution pour applications Java.
- La **machine virtuelle** permet notamment :
 - L'interprétation du bytecode
 - L'interaction avec le système d'exploitation
 - La gestion de sa mémoire grâce au ramasse-miettes
- **La machine virtuelle ne connaît pas le langage Java** : elle ne connaît que le bytecode qui est issu de la compilation de codes sources écrits en Java.
- Il existe de nombreuses implémentations de JVM dont les plus connues sont :
 - [HotSpot](#) de Sun Microsystem (la plus utilisée).
 - [Apache Harmony](#) par la fondation Apache
 - OpenJ9 initialement IBM (puis sous licence Eclipse).
 - MRJ licence propriétaire Apple.
 - Etc...

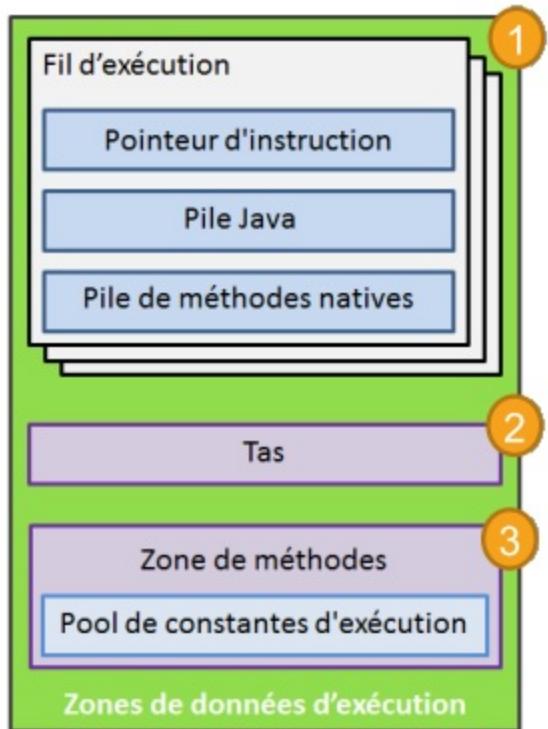
JVM



Lors de l'exécution de notre code Java :

1.
 - Le fichier .java va être compilé par le compilateur pour le transformer en byte Code avec une extension .class.
 - Le **chargeur de classe (ClassLoader)** se charge de retrouver l'emplacement des bibliothèques, de lire leur contenu et de charger les classes qu'elles contiennent .
 - Le **moteur d'exécution** est le Composant Central de la JVM. Il communique avec différentes zones mémoire de la JVM. Il exécute les fichiers .class.
 - Le **moteur d'exécution** exécute le code d'octet qui est affecté aux zones de données d'exécution dans JVM via le chargeur de classe.
 - Le **moteur d'exécution** lit le byte code et interprète (convertit) en code machine (code natif) et les exécute de manière séquentielle.
2.
 - La **JVM** définit différentes **zones de données d'exécution** qui sont utilisées durant l'exécution d'un programme.
 - Certaines de **ces zones de données** sont créées lorsque la **JVM** est lancée et sont détruites lorsqu'elle s'arrête.

JVM



Lors de l'exécution de notre code Java :

1.
 - Les autres **zones de données d'exécution** sont propres à chaque fil d'exécution (thread).
 - Les **zones de données par fils d'exécution** sont créées lorsque le fil d'exécution est créé et sont détruites lorsqu'il se termine .

2.
 - La **JVM** a un **tas** partagé par tous les **fils d'exécution**.
 - Le **tas** est une zone de données d'exécution dans laquelle toutes les instances de classes et les tableaux sont stockés .

3.
 - Lorsque le **chargeur de classes** charge une classe, il stocke sa structure dans la **Zone de Méthodes**.
 - La **Zone de Méthodes** d'un espace mémoire partagé par tous les fils d'exécution



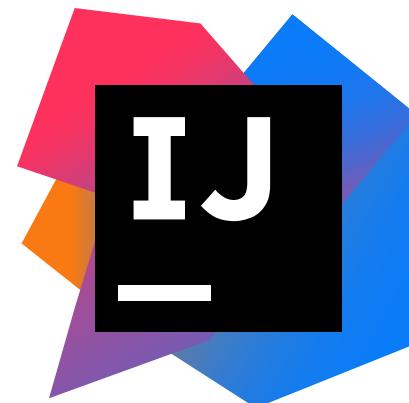
Les IDE

Les IDE

- **Un environnement de développement intégré** (IDE : Integrated Development Environnement) est un ensemble d'outils qui permet d'augmenter la productivité des programmeurs qui développent des logiciels.
- Les outils de développement incluent souvent :
 - des éditeurs de texte.
 - des bibliothèques de code.
 - des compilateurs.
 - un debugger.
 - des plates-formes de test.
- Certains IDE sont open source, tandis que d'autres sont des offres commerciales.
- L'objectif d'un IDE est d'augmenter la productivité des programmeurs en automatisant une partie des activités et en simplifiant les opérations.

Les IDE

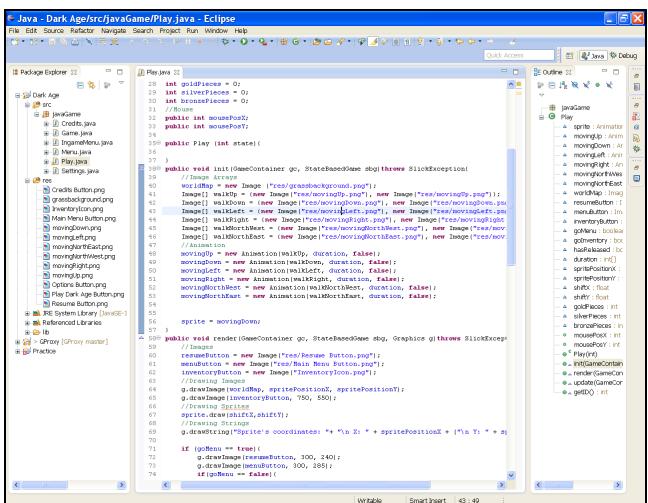
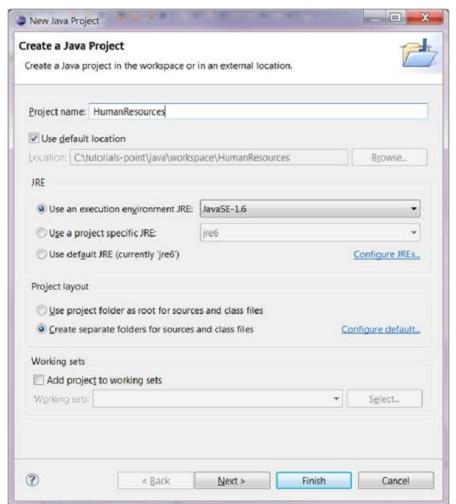
- Il existe presque autant d'IDE que de langage, ci-dessous quelques uns d'entre :
 - Visual studio code.
 - Intelij Idea.
 - Eclipse.
 - NetBeans.
 - Xcode, etc.
- Les 2 principaux pour Java :



Les IDE



- En 2021, l'**IDE Eclipse** compte environ un million de téléchargements par mois, ce qui en fait l'un des principaux IDE pour le développement Java.
- **Eclipse** peut être étendu avec des composants logiciels supplémentaires appelés plugins.
- La distribution **Eclipse IDE for Java Developers** est conçue pour prendre en charge le développement Java standard
 - Il inclut la prise en charge du système de construction Maven et Gradle et la prise en charge du système de contrôle de version Git.



- Téléchargez le programme d'installation via <https://eclipse.org/downloads/eclipse-packages/>.
- **Les éléments importants :**

1. Espace de travail et projets :

L'espace de travail est l'emplacement physique (chemin d'accès au fichier) pour stocker les métadonnées et (facultatif) vos artefacts de développement.

2. L'interface utilisateur :

Eclipse fournit des vues et des éditeurs pour naviguer et modifier le contenu. La vue et les éditeurs peuvent être regroupés en perspectives .

3. Projets Eclipse :

Un projet Eclipse contient des fichiers source, de configuration et binaires liés à une certaine tâche. Il les regroupe en unités constructibles et réutilisables. Un projet Eclipse peut avoir des natures qui lui sont assignées qui décrivent le but de ce projet.

Les IDE



- **IntelliJ IDEA** également appelé « IntelliJ », « IDEA » ou « IDJ » est un environnement de développement intégré destiné au développement de logiciels informatiques reposant sur la technologie [Java](#).
- Il est développé par [JetBrains](#) (anciennement « IntelliJ ») avec 2 versions :
 - L'une communautaire, open source, sous [licence Apache 2](#).
 - L'autre propriétaire, protégée par une licence commerciale.

Les IDE



- **IntelliJ IDEA** apporte une étroite intégration avec quelques-uns des outils de développement libres les plus répandus, tels que [Git](#), [CVS](#), [Subversion](#), [Ant](#) et [Maven](#), [JUnit](#) et [TestNG](#).
- **Les autres langages pris en charge :**
 - [Dart](#) - [CoffeeScript](#) - [Go](#) - [Groovy](#)
 - [HTML/XHTML/CSS](#)
 - [Java](#) - [JavaScript](#) - [JSON](#) - [Kotlin](#)
 - [Markdown](#) - [PHP](#) - [Python](#) - [Ruby/Jruby](#) - [Rust](#) - [Scala](#)
 - [SQL](#) - [XML/XSL](#) - [YAML](#)

<https://www.jetbrains.com/idea/>

Les Variables

Variables

- Une variable Java est un morceau de mémoire qui peut contenir une valeur de données.
- Java est un langage fortement typé : chaque variable a un type.
- Une variable est définie par la combinaison d'un identifiant et d'un type.
- Vous pouvez déclarer une variable avec le mot-clé 'var' - dans ce cas, le type de la variable sera défini par le compilateur.
- Toutes les variables ont une portée, qui définit leur visibilité.
- Vous ne pouvez pas donner un nom à la variable du mot-clé java
(https://en.wikipedia.org/wiki/List_of_Java_keywords)

Définition

- Les variables sont des symboles qui associent un nom (l'identifiant) à un type et une valeur :
 - « Type » « nomVariable » = « valeur »;
- Il existe deux grandes familles de variables :
 - Les variables primitives
 - Ces variables contiennent une valeur en mémoire une fois initialisée
 - Les variables de référence aux objets
 - Ces variables ne contiennent pas de valeur mais la référence mémoire de l'objet (pointeur mémoire)
- Il existe une infinité de type de variables allant des plus simple (entiers) aux plus complexes (objets)

```
int solde = 123; // Déclare une variable de type entier nommé solde de valeur 123
String nomFamille = "Durand" ;// Variable de type chaine de caractères (String)
Maison grandModel = new Maison("individuelle" , 95 , 4 , true); // Objet
```

Les Variables Primitives

- La famille des variables **primitives** est composée de **quatre sous-famille** :

Les nombres entiers

- Intéger (int)
- Le Byte (byte)
- Le Short (short)
- Le Long (long)

Les nombres décimaux

- Le Float (float)
- Le Double (double)

Les variables booléennes

- Le Booléen (boolean)

Les caractères

- Le Caractère (char)

-
- Depuis la **version 9** de Java, il est possible d'utiliser l'**auto-typage** pour déclarer les **variables** (var) :
 - Important, bien que le type ne soit plus explicite, la variable reste bien typé, elle prend le type de sa valeur une fois initialisée

```
// Exemple :  
var age = 15 ; // Auto-typage de la variable en int car la valeur associée est de ce type
```

Les Nombres Entiers

- Détails des différents types d'entiers et leurs plages d'utilisation:

	Taille (bits)	Etendue
byte	8	-128 à 127
short	16	- 32768 à 32767
int	32	- 214748648 à 214748647
long	64	- 9 223 372 036 854 775 808 à 9 223 372 036 854 775 807

- En Java, les **type numériques** sont **OBLIGATOIUREMENT** signés :
 - C'est pourquoi, à l'inverse du langage C, le Java emploie uniquement le char pour les caractères et le type numérique est confié au byte.

Les Nombres Entiers

- Il existe plusieurs bases numériques pouvant être employées en java :

- Base binaire (valeurs préfixées par 0b – à partir de Java SE 7) :

```
int valeur = 0b111; // 7 en base binaire
```

- Base octale (valeurs préfixées par 0) :

```
int valeur = 020; // 16 en base octale
```

- Base décimale :

```
int valeur = 10;
```

- Base Hexadécimale (valeur préfixées pas 0x) :

```
int valeur = 0x20; // 32 en base Hexadécimale
```

- Depuis Java SE 7, il est possible d'employer un séparateur de groupe de chiffre (le caractère _) :

- Fonctionne avec toutes les bases numériques (le nombre de chiffres regroupés est libre)
 - Le caractère _ ne doit pas être positionné ni avant le premier, ni après le dernier chiffre.

- ```
int count = 1_222_333_444 ;
int color = 0xff_ff_00_ff
```

## Les Nombres Entiers

- Déclaration d'une variable de type entier:

- **byte** :

```
byte variableByte; // Déclare une variable de type byte nommée variableByte
```

- **short** :

```
short variableShort; // Déclare une variable de type short nommée variableShort
```

- **int** :

```
int variableInt; // Déclare une variable de type Int nommée variableInt
```

- **long** :

```
long variableLong; // Déclare une variable de type long nommée variableLong
```

- Initialisation / assignation de valeur à la variable :

- **byte** :

```
variableByte = 7B; // B à la fin d'un byte
```

- **short** :

```
variableShort = 138;
```

- **int** :

```
int variableInt = 54321; // Déclaration & initialisation de sa valeur
```

- **long** :

```
long variableLong = 9876543210L; // Ne pas oublier le L à la fin d'un long
```

## Les Nombres Décimaux

- Détails des différents **types** de nombres décimaux :

|        | Taille (bits) | Exemple   |
|--------|---------------|-----------|
| float  | 32            | 3,25f     |
| double | 64            | 0,0000325 |

- Tout comme le langage C, il existe seulement **deux types** de variables **décimales** :
  - La seule différence entre ces deux types est leur précision (Grâce au nombre d'octets utilisés pour stocker une valeur flottante)
  - Par défaut, une variable décimale est typée double. Ne pas oublier le f à la fin pour les float sinon il sera pas compilé:

```
float f = 10.75F; // Compile
float f = 10.75; // Ne compile pas
```

# Les Nombres Décimaux

- Déclaration d'une variable de type décimal:

- **float** :

```
float variableFloat ; // Déclare une variable de type float nommé variableFloat
```

- **double** :

```
double variableDouble ; // Déclare une variable de type double nommé variableDouble
```

- Initialisation / modification de la valeur de la variable :

- **float** :

```
variableFloat = 10.75f ; // Assigne à variableFloat la valeur de 10,75
```

- **double** :

```
variableDouble = 999999.999 ; // Assigne à variableDouble la valeur de 999999.999
```

- Déclaration de la variable et initialisation de sa valeur à la création :

- **float** :

```
float variableFloat = 10.75f;
```

- **double** :

```
double variableDouble = 999999.999;
```

# Les Nombres Décimaux

## Utiliser flottant ou double ?

- La précision d'une valeur à virgule flottante indique le nombre de chiffres que la valeur peut avoir après la virgule décimale.
- La précision de float n'est que de six ou sept chiffres décimaux, tandis que les variables doubles ont une précision d'environ 15 chiffres.
- Par conséquent, il est plus sûr d'utiliser double pour la plupart des calculs.

## Les Booléens ( bool )

- Les variables de **types** Booléenne (boolean) ne peuvent avoir que deux états:

|         | Taille (bits) | Etat           |
|---------|---------------|----------------|
| boolean | 1             | True ( vrai )  |
| boolean | 1             | False ( faux ) |

- Important, à la différence du langage c et c# , une valeur **numérique n'est pas considérée** comme une valeur **booléenne** en **Java**.

  - Il est donc impossible d'écrire ce genre de vérification :

```
if (verif == 1) { . . . } ; // Ne compile pas
```

  - Il faudra plutôt écrire ce genre de vérification d'égalité :

```
if (verif == true) { . . . } ; // Compile parfaitement.
// Si verif est égal à true alors le code contenu dans la boucle sera exécuté
```

# Les Booléens ( bool )

- Déclaration d'une variable de type Booléenne ( bool ) :

```
boolean variableBool; // Déclare une variable de type boolean nommé variableBool
```

- Initialisation / modification de la valeur de la variable :

```
variableBool = true; // Assigne à variableBool la valeur true (vrai)
```

- Déclaration de la variable et initialisation de sa valeur à la création:

```
boolean variableBool = false; // Déclare et assigne à variableBool la valeur false
```

- L'état par défaut d'une variable booléenne est **false**.

# Les Caractères ( char )

- Les variables de types Caractère (char) ne peuvent avoir de valeur numérique:

|      | Taille (Octets) | Etendue                          |
|------|-----------------|----------------------------------|
| char | 2               | <a href="#">Unicode (UTF-16)</a> |

- La valeur d'un char doit être contenue OBLIGATOIUREMENT entre deux simples quotes ('')

```
'm' // La lettre m en minuscule
'\t' // Une tabulation
'\n' // Un retour à la ligne
'\\' // Un caractère \ (car il a un sens particulier en Java)
'\u03c0' // Un caractère Unicode (π) à partir de son code UTF-16
```

# Les Caractères ( char )

- Déclaration d'une variable de type Caractère ( char ) :

```
char variableChar ; // Déclare une variable de type boolean nommé variableChar
```

- Initialisation / modification de la valeur de la variable :

```
variableChar = 'a' ; // Assigne à variableChar la valeur 'a'
```

- Déclaration de la variable et initialisation de sa valeur à la création :

```
char variableChar = '\n' ; // Déclare et assigne à variableChar la valeur \n (retour à la ligne)
```

- L'état par défaut d'une variable caractère ( char ) est '\0'

# Les variables de référence aux objets

- Les **variables de référence aux objets** ont des types particulier faisant appel à des classes pour les définir :
  - Leur type est introduit par une majuscule

```
String phrase = "Ceci est une chaîne de caractères, elle est de type String";
```

- Elle peuvent exposer un certain nombre de méthodes propres à la classe d'origine

```
String phraseTitre = "Voici mon titre".toUpperCase(); // Méthode Mise en Majuscule
```

- Les **variables de référence aux objets** sont **immutable** :
  - Une fois **instanciée**, les **objets** issus d'une classe ne peuvent **plus** être **modifiés**
    - Il faut **obligatoirement** que la **classe** à l'origine de l'instance en **possède la méthode**.
    - Une **méthode** permettant la **transformation** d'un objet **renvoie** forcément à une **nouvelle instance** de celui-ci.

# Les Chaînes de Caractères (String)

- Parmi les variables de référence aux objets un des types les plus usités est le type String. Il est employé afin de manipuler les chaînes de caractères :
  - Il s'agit bien d'une classe et non d'un type primitif, donc introduite par une majuscule

```
String phrase = "Ceci est une chaîne de caractères, elle est de type String";
```

- A l'inverse de beaucoup de langage, en Java, un String n'est pas un tableau de (char) mais un objet
  - Comme tout objet, il hérite des méthodes de sa classe d'origine pour le manipuler
- Parmi les principales méthodes de la classe String, observons notamment :

| Méthode       | Etendue                                                                  |
|---------------|--------------------------------------------------------------------------|
| charAt(pos)   | Renvoie le caractère à la position indiquée (pos). Etendue               |
| Length()      | Renvoie la taille de la chaîne de caractères.                            |
| equals()      | Compare le contenus de deux chaînes de caractères.                       |
| toLowerCase() | Produit une chaîne entièrement en minuscule à partir de l'objet courant. |
| toUpperCase() | Produit une chaîne entièrement en majuscule à partir de l'objet courant. |

# Les Chaînes de Caractères (String)

- Déclaration d'une variable de type chaîne de caractères ( String ) :

```
String variableString ; // Déclare une variable de type String nommé variableString
```

- Initialisation / modification de la valeur de la variable :

```
variableString = "a" ; // Assigne à variableString la valeur 'a'
```

- Déclaration de la variable et initialisation de sa valeur à la création :

```
String variableString = "\n" ; // Déclare et assigne à variableString la valeur \n (retour à la ligne)
```

- L'état par défaut d'une variable chaine de caractères ( String ) est 'null'

# Les types énumérés (enum)

- Les **types énumérés** ( enum ) sont supportés depuis la version **Java SE 5.0** :
  - Un type **énuméré** propose **plusieurs état prédéfinis**
  - Une **variable** basée sur un type **énuméré** ne pourra prendre comme **valeur** qu'un des **états supportés**.
  - On définit souvent un **type énuméré** dans un **fichier** du **même nom** ( comme pour une classe ) et son nom commence donc par une **majuscule**.
- Exemple de définition d'un type énuméré :

```
Public enum SystemExploitation {
 Windows, Linux, MacOs
}
// Utilisation du type énuméré
SystemExploitation monOs = SystemExploitation.Linux;
if (monOs == SystemExploitation.Windows) {
 System.out.println(« Votre système est compatible ! »)
}
```

# Les variables

| Type                          | Size                      | Min Value                                                                                                   | Max Value                              | Wrapper Type |
|-------------------------------|---------------------------|-------------------------------------------------------------------------------------------------------------|----------------------------------------|--------------|
| <b>INTEGERS</b>               |                           |                                                                                                             |                                        |              |
| <b>byte</b>                   | 1 bytes                   | -128                                                                                                        | 127                                    | Byte         |
| <b>short</b>                  | 2 bytes                   | $-2^{15}$ (-32,768)                                                                                         | $-2^{15} - 1$ (-32,767)                | Short        |
| <b>int</b>                    | 4 bytes                   | $-2^{31}$ (-2,147,483,648)                                                                                  | $-2^{31}$ (-2,147,483,647)             | Integer      |
| <b>long</b>                   | 8 bytes                   | $-2^{63}$ (-9,223,372,036,854,755,808)                                                                      | $-2^{63}$ (-9,223,372,036,854,755,807) | Long         |
| <b>FLOATING-POINT NUMBERS</b> |                           |                                                                                                             |                                        |              |
| <b>float</b>                  | 4 bytes                   | approximately +3.40282347E+38F (6- 7 significant decimal digits)<br>Java implements IEEE 754 standard Float |                                        | Float        |
| <b>double</b>                 | 8 bytes                   | approximately +1.79769313486231570E+308 (15 significant decimal digits)                                     |                                        | Double       |
| <b>BOOLEAN</b>                |                           |                                                                                                             |                                        |              |
| <b>boolean</b>                | virtual machine dependent | True OR false                                                                                               |                                        | Boolean      |
| <b>CHARACTERS</b>             |                           |                                                                                                             |                                        |              |
| <b>char</b>                   | 2 bytes                   | 0                                                                                                           | 65,535                                 | Character    |

# Les Opérateurs

# Les Opérateurs de Valeur Numérique Entières

- Voici la liste des principaux opérateurs utilisable avec des valeurs numériques entières :

| Opérateurs Arithmétiques |                                         |
|--------------------------|-----------------------------------------|
| +                        | Addition                                |
| -                        | Soustraction                            |
| *                        | Multiplication                          |
| /                        | Division                                |
| %                        | Modulo ( Reste de la division entière ) |

| Opérateurs Binaires |                                                       |
|---------------------|-------------------------------------------------------|
| &                   | « Et » bit à bit : $0b010101 \& 0b101011 == 0b000001$ |
|                     | « Ou » bit à bit : $0b010101   0b101010 = 0b111111$   |
| ^                   | « Xor » bit à bit : $0b010101 ^ 0b111111 = 0b101010$  |

# Les Opérateurs de Valeur Numérique Décimale

- Voici la liste des principaux opérateurs utilisable avec des valeurs numériques décimale:

| <b>Opérateurs Arithmétiques</b> |                       |
|---------------------------------|-----------------------|
| <b>+</b>                        | <b>Addition</b>       |
| <b>-</b>                        | <b>Soustraction</b>   |
| <b>*</b>                        | <b>Multiplication</b> |
| <b>/</b>                        | <b>Division</b>       |

| <b>Opérateurs de comparaisons</b> |                                                    |
|-----------------------------------|----------------------------------------------------|
| <b>==</b>                         | <b>Egalité (Renvoie un booléen)</b>                |
| <b>!=</b>                         | <b>Différence (Renvoie un booléen)</b>             |
| <b>&lt;</b>                       | <b>Infériorité stricte (Renvoie un booléen)</b>    |
| <b>&lt;=</b>                      | <b>Infériorité ou égalité (Renvoie un booléen)</b> |
| <b>&gt;</b>                       | <b>Supériorité stricte (Renvoie un booléen)</b>    |
| <b>&gt;=</b>                      | <b>Supériorité ou égalité (Renvoie un booléen)</b> |

# Les Opérateurs de Valeur Numérique

| Opérateurs d'affectation |                                                                    |
|--------------------------|--------------------------------------------------------------------|
| =                        | <b>Ex : a = 2 ;</b>                                                |
| +=                       | <b>Ex : a+=2 ; Equivaut quasiment à a = a + 2 ;</b>                |
| -=                       | <b>Ex : a-= 2 ; Equivaut quasiment à a = a - 2 ;</b>               |
| *=                       | <b>Ex : a*= 2 ; Equivaut quasiment à a = a * 2 ;</b>               |
| /=                       | <b>Ex : a/= 2 ; Equivaut quasiment à a = a / 2 ;</b>               |
| %=                       | <b>Ex : a%= 2 ; Equivaut quasiment à a = a % 2 ;</b>               |
| &=                       | <b>Ex : a&amp;= 2 ; Equivaut quasiment à a = a &amp; 2 ;</b>       |
| =                        | <b>Ex : a = 2 ; Equivaut quasiment à a = a   2 ;</b>               |
| ^=                       | <b>Ex : a^= 2 ; Equivaut quasiment à a = a ^ 2 ;</b>               |
| <<=                      | <b>Ex : a&lt;&lt;= 2 ; Equivaut quasiment à a = a &gt;&gt; 2 ;</b> |
| >>=                      | <b>Ex : a&gt;&gt;= 2 ; Equivaut quasiment à a = a &gt;&gt; 2 ;</b> |

# Les Opérateurs de Valeur Booléenne

- Voici la liste des principaux opérateurs utilisable avec des valeurs Booléennes:

| Opérateurs Logiques |                                                |
|---------------------|------------------------------------------------|
| &&                  | « Et » logique ( Ex : true && false == false ) |
|                     | « Ou » logique ( Ex : true    false == true )  |
| !                   | « Not » logique                                |

# Les Opérateurs de Chaînes de Caractères (String)

- L'opérateur de comparaison == entre deux String requière une attention particulière :
  - Il compare les valeurs contenu dans les variables. Dans le cas d'un String, sa valeur est l'adresses (pointeur mémoire) de la chaîne de caractère (donc de l'objet).
    - L'opérateur == entre deux String reviens donc à demander s'il s'agit bien de la même instance
  - La méthode equals() compare les contenus des deux chaînes de caractères

```
String nom = "toto"; // Déclaration d'une variable de type String nommé nom de valeur toto
String part = "to"; ; // Déclaration d'une variable de type String nommé part de valeur to
String autreNom = part + part; // Déclaration String nommé autreNom de valeur « to » + « to »
System.out.println(nom == autreNom); // false
System.out.println(nom.equals(autreNom)); // true
String notANewObject = nom; // nom prend donc comme valeur le pointeur mémoire
System.out.println(nom == notANewObject); // true
System.out.println(nom.equals(notANewObject)); // true
```

# Les Opérateurs ++ et --

- Les opérateurs d'incrémentation ( ++ ) et de décrémentation ( -- ) permettent d'ajouter ou retirer la valeur 1 à une variable numérique.
  - Utilisation préfixée des opérateurs ++ et -- :

```
int value = 10 ; // On initialise value à 10
++value ; // On incrémente de 1
System.out.println(value) ; // Affiche 11
--value; // On décrémente de 1
System.out.println(value) ; // Affiche 10
```

- Cas d'une somme de deux variables préfixée (incrémentée avant l'emploi) :

```
int premier = 10 ; // On initialise premier à 10
int deuxième = 20 ; // On initialise deuxième à 20
int result = ++premier + ++deuxième; // Somme des deux variables préfixée
System.out.println(result); // Affiche 32 car incrémantation avant la somme
```

# Les Opérateurs ++ et --

- Utilisation postfixée des opérateurs ++ et -- :

```
int value = 10 ; // On initialise value à 10
value++ ; // On incrémente de 1
System.out.println(value) ; // Affiche 11
value--; // On décrémente de 1
System.out.println(value) ; // Affiche 10
```

- Cas d'une somme de deux variables postfixée (incrémentée après l'emploi) :

```
int premier = 10 ; // On initialise premier à 10
int deuxième = 20 ; // On initialise deuxième à 20
int result = premier++ + deuxième++ ; // Somme des deux variables postfixée
System.out.println(result) ; // Affiche 30 car incrémentation après la somme
```

- La variable préfixée est d'abord incrémentée/décrémentée avant d'être employée tandis que la postfixée est d'abord employée avant d'être incrémenté/décrémenté

# Les autres opérateurs Java

- Quelques autres opérateurs du langage Java :

| <b>Opérateurs ( Divers)</b> |                                                                               |
|-----------------------------|-------------------------------------------------------------------------------|
| <b>new</b>                  | <b>Opérateur d'allocation mémoire</b>                                         |
| <b>[ ]</b>                  | <b>Opérateur permettant l'accès aux éléments d'un tableau</b>                 |
| <b>?:</b>                   | <b>Opérateur conditionnel (Seul opérateur Java acceptant trois opérandes)</b> |
| <b>.</b>                    | <b>Opérateur de traversé</b>                                                  |
| <b>instanceof</b>           | <b>Permet de savoir si un objet est compatible avec un type de donné</b>      |



# Exercice(s)

# Exercice : Prénom nom

1. Créer une variable nom et une variable prenom
2. Afficher la phrase suivante "Bonjour {prenom} {NOM}."
3. Remplacer les valeurs entre chevrons par les variables créées précédemment

Saisir un prenom :

James

Saisir un nom :

Bond

Bonjour James BOND.

# Exercice : Permuter deux variables

1. Écrire un programme qui permet de permuter les valeurs entre deux variables

```
Saisir a :
5
Saisir b :
3
a = 3 et b = 5
```

# Exercice : Somme des carrés

1. Écrire un programme avec les variables suivantes : a, b
2. Afficher la somme des carrés de ces deux nombres

```
Saisir a :
2
saisir b :
3
La somme des carrés de a et b est : 13
```

## Exercice : majeur ou mineur

1. Créer une variable age et lui affecter une valeur
2. Vérifier si la personne est mineure ou majeure à l'aide des opérateurs logiques
3. Afficher le résultat
4. /!\ Ne pas utiliser de structure conditionnelle

```
Saisir un age : 17
false
```

```
Saisir un age : 25
true
```

# Exercice : volume d'un cercle

1. Créer un programme qui permet de calculer le volume d'un cône
2. La formule est la suivante :  $1/3 \times \pi \times r^2 \times h$

```
Saisir un rayon : 4
```

```
Saisir une hauteur : 7
```

```
Le volume du cône est de 117,29 cm3
```

## Exercice : Mise en forme de chaînes

1. Saisir une chaîne et la stocker dans une variable

2. Effectuer les traitements suivants :

1. Afficher la chaîne en minuscule

2. Afficher la chaîne en majuscule

3. **Bonus**

1. Convertir la chaîne en tableau puis afficher les caractères séparés d'une virgule

2. Afficher la première lettre de chaque mot en majuscule

Saisir une chaîne : CoMMenT çA vA ?

En minuscule : comment ça va ?

En majuscule : COMMENT ÇA VA ?

En tableau : c,o,m,m,e,n,t, ,ç,a, ,v,a, ,?

Première lettre en majuscule : Comment Ça Va ?

# Exercice : Périmètre et aire d'un carré

1. Saisir la longueur du côté du carré et la stocker dans une variable
2. Calculer l'aire et le périmètre du carré

```
Saisir la longueur d'un côté : 5
Le périmètre du carré est de : 20cm
L'aire du carré est de : 25cm carré
```

# Les Instructions

# Vue d'ensembles des instructions applicables en Java

- Définition d'une instruction :
  - Une instruction permet de lancer des actions (un test, une boucle, une assertion ... )
  - Toute les instructions se termine par le caractère « ; »
- Tout oublie d'un « ; » entraîne systématiquement une erreur à la compilation
- Le bloc d'instructions :
  - Un bloc d'instructions commence par « { » et se termine par « } »
  - Un bloc d'instructions peut contenir une ou plusieurs instruction(s)

```
if (age >= 18){ //Si condition remplie on rentre dans le bloc d'instructions
 System.out.println(" Vous êtes majeur ") ; // Exécution de la 1ere instruction
 System.out.println(" Vous pouvez voter ") ; // Exécution de la 2nd instruction
} // Fermeture et sortie du bloc d'instructions
```

# Les blocs d'instructions

- Une variable déclarée dans un bloc d'instruction voit sa durée de vie limité à la durée du bloc qui la contiens :
  - Quand on déclare une variable locale à un bloc d'instructions, un espace de mémoire est réservé sur la pile d'exécution afin d'y stocker la valeur de la variable
  - A la fin de l'exécution de ce bloc, la variable est supprimé de la pile d'exécution : On ne peux plus y accéder

```
{
int age = 10 ;
float taille = 1,80 ;
System.out.println(« Vous avez : » + age + « ans ») ;
System.out.println(« Vous mesurez : » + taille + « m») ;
} // Une fois le bloc terminé, l'ensemble des instructions sont supprimées. Ici taille et age sont supprimés
```

# Les instructions applicables en Java : if (else)

- Définition de l'instruction de condition if (Si) else (Sinon) :
  - La condition Si ( if ) est nécessaire afin de comparer une/des variable(s) et appliquer une/des instruction(s) si les conditions son remplies
  - Voici la syntaxe d'une instruction ( if ) en Java :

```
if («Condition») { // Si la condition est remplie
 InstructionAppliquee(); // Instruction appliquée si condition remplie
} // Sortie de la condition Si
else { // Sinon (la condition n'est pas remplie)
 AutreInstructionAppliquee(); // Instruction appliquée si condition non remplie
} // Sortie de la condition Sinon
```

- Notez que le bloc d'instruction else est facultatif mais s'il est présent, il est de bonne pratique qu'il contienne des instructions

## Les instructions applicables en Java : else if

- Définition de l'instruction de condition else if (Sinon Si) :
  - La condition else if est nécessaire afin de comparer une/des variable(s) à plusieurs conditions et appliquer une/des instruction(s) différentes pour chaque condition
  - Voici la syntaxe d'une instruction else if en Java :

```
if (« Condition1») { // Si la condition1 est remplie
 Instruction1Appliquee(); // Instruction appliquée si condition1 remplit
} // Sortie de la condition Si
else if (« Condition2») { // Sinon Si (établissement d'une autre condition)
 Instruction2Appliquee(); // Instruction appliquée si Condition2 remplit
} // Sortie de la condition Sinon Si
else { // Sinon (Aucune condition n'est remplie, cas par défaut)
 AutreInstructionAppliquee(); // Instruction appliquée si cas par défaut
} // Sortie de la condition Sinon
```

## Les instructions applicables en Java : Boucle for

- Définition de l'instruction de boucle for (pour) :
  - Les boucles for testent une condition et exécutent le bout de code attaché à la boucle tant que la condition est remplie.
  - Elles sont le plus souvent utilisées lorsque l'on sait combien de fois on souhaite exécuter le bout de code attaché à la boucle.
  - Voici la syntaxe d'une boucle for en Java :

```
for ("InitialisationVariableBoucle" ; "ConditionDeRebouclage" ; "IncremetationDeLaBoucle") {
 "Action à appliquer à chaque boucle"();
}
```

- Exemple :

```
for (int i = 0 ; i < 3 ; i++) {
 System.out.println(i + 1 + " Instruction(s) dans la boucle for ");
}
```

- Résultat dans la console :

```
1 Instruction(s) dans la boucle for
2 Instruction(s) dans la boucle for
3 Instruction(s) dans la boucle for
```

# Les instructions applicables en Java : Boucle for (each)

- Définition de l'instruction de boucle for (each) (Pour chaque) :
  - Les boucles for (each) permettent de traiter les éléments d'une collection sans avoir à gérer l'index de boucle.
  - Voici la syntaxe d'une boucle for (each) en Java :

```
« Type » [] « nomTableau » = { « valeur1 », « valeur2 », « valeur3 », ... } ;
for (« Type » « nomObjet » : « nomTableauObjet») {
 System.out.println(" nomObjet");
}
```

- Cette variante de l'instruction for fonctionne avec :
  - Les tableaux traditionnels (exemple ci-dessus)
  - Les collections basées sur l'interface java.util.List(ArrayList, Vector, ...)
  - Les collections basées sur l'interface java.util.set(HashSet,...)
  - T plus généralement tout type implémentant l'interface java.util.Iterable

## Les instructions applicables en Java : Boucle while

- Définition de l'instruction de boucle while (tant que) :
  - Le code est exécuté tant que le booléen est vrai. Si avant l'instruction while, le booléen est faux, alors le code de la boucle ne sera jamais exécuté. Voici la syntaxe :

```
"ExpressionInitialisationBoucle"
while ("ConditionDeRebouclage")
{
 Action à appliquer à chaque boucle() ;
 IncremetationDeLaBoucle() ; // Si besoin
}
```

Exemple :

```
int i = 0 ;
while (i < 5){
 System.out.println(i + " Instruction(s) dans la boucle while");
 i++ ;
}
```

## Les instructions applicables en Java : switch

- Définition de l'instruction switch (Switchcase) :
  - L'instruction switch est utile quand vous devez gérer beaucoup de if / else if / else.
  - Elle a une syntaxe plus courte et plus appropriée pour ce type de cas. Voici sa syntaxe ci-contre.
  - Ici dans notre exemple de fonctionnement de cette instruction, on remarque la ligne case 'valeur3' : 'valeur4' :
    - Cette ligne veut dire que si la variable 'variable' vaut soit 'valeur3' soit 'valeur4', alors on exécutera « Instruction3 ».
    - Vous pouvez mettre autant de valeurs que vous souhaitez.
    - Si aucune valeur ne correspond, les instructions contenues dans le bloc default s'exécutent.

```
switch (variable)
{
 case 'valeur1' :
 Instruction1;
 break ;
 case 'valeur2' :
 Instruction2;
 break ;
 case 'valeur3' : 'valeur4' :
 Instruction3;
 break ;
 default :
 DefaultInstruction1;
 break ;
}
```



# Exercice(s)

# Exercice : majeur ou mineur

1. Créer une variable age
2. Affecter une valeur à la variable age
3. Créer une condition qui permet d'afficher si la personne est majeure ou mineure

Saisir un âge : 23

Vous êtes majeur, vous pouvez rentrer dans le club.

## Exercice : Pair ou impair

1. Créer une variable nombre de type entier
  2. Affecter une valeur à la variable nombre
  3. Créer une condition qui permet d'afficher si le nombre est pair ou impair
- **Utiliser l'opérateur mathématique permettant de retourner le reste d'une division euclidienne**

Saisir un nombre : 10

Le nombre saisi est pair

Saisir un nombre : 3

Le nombre saisi est impair

# Exercice : Maximum de 3 nombres

1. Créer 3 variables : nombre1, nombre2, nombre3
2. Affecter des valeurs aux variables
3. Créer des structures conditionnelles permettant d'afficher le maximum parmis les 3 nombres créésprécédemment

```
Saisir le nombre 1 : 2
Saisir le nombre 2 : 5
Saisir le nombre 3 : 3
La valeur maximale est : 5
```

# Exercice : Voyelle ou consonne

1. Créer une variable caractere
2. Affecter une valeur à la variable caractere
3. A l'aide des structures conditionnelles, déterminer si le caractère est une consonne ou une voyelle

Saisir un caractère : c

Le caractère saisi est une consonne

# Exercice : Jour de la semaine

1. Créer une variable jour de type entier
2. Affecter une valeur à la variable jour
3. Afficher le jour de la semaine en lettre en fonction du nombre passé
4. Si le nombre est inférieur à 1 et supérieur à 7, afficher un message d'erreur

```
Saisir un jour de la semaine : 5
Vendredi
```

# Exercice : Jours dans le mois

1. Créer une variable mois de type entier
2. Affecter une valeur à la variable mois
3. Afficher le nombre de jours du mois
4. Si le nombre est inférieur à 1 et supérieur à 12, afficher un message d'erreur

```
Saisir un numéro de mois : 3
31 jours
```

# Exercice : Année bissextile

1. Créer une variable annnee de type entier
2. Affecter une valeur à la variable annnee
3. Une année est bissextile si elle est divisible par 4 mais non divisible par 100.
4. L'année est également bissextile si elle est divisible par 400

```
Saisir une année : 2020
2020 est une année bissextile
```

# Exercice : Lettre, nombre ou caractère spécial

1. Créer une variable caractere
2. Affecter une valeur à la variable caractere
3. Afficher un message en fonction du type du caractère (lettre, nombre, ou caractère spécial)

```
Saisir un caractère : h
h est une lettre de l'alphabet
```

## Exercice : Etat de l'eau

1. Définir une variable température
2. Affecter une valeur à la variable temperature
3. Selon la température, affiche l'état de l'eau :
  - SOLIDE : température inférieure à 0°C
  - LIQUIDE : température entre 0 et 100°C
  - GAZEUX : température supérieure à 100°C

```
Saisir une température : 5
```

```
LIQUIDE
```

```
Saisir une température : -3
```

```
SOLIDE
```

## Exercice : Candidature

1. Ecrire un programme qui permet de vérifier si un profil est valable pour une candidature
2. Le profil contient trois critères :
  - age : supérieur à 30 ans
  - salaire demandé : maximum 40 000€
  - années d'expériences : minimum 5 ans
3. Afficher un message pour chaque condition non respectée

```
Saisir un âge : 25
```

```
Saisir le salaire souhaité : 34000
```

```
Saisir le nombre d'années d'expériences : 4
```

```
Vous êtes trop jeune
```

```
Vous manquez d'expérience
```

# Exercice : De 1 à 10

1. Écrire un programme qui affiche les nombres de 1 à 10 à l'aide d'une boucle
2. Afficher la valeur à chaque itération

```
1
2
3
...
10
```

## Exercice : Chapitres

1. Créer un programme qui permet d'afficher un nombre de chapitres et de sous-parties
2. Le programme demandera le nombre de chapitres ainsi que le nombre de sous-parties à afficher

```
Saisir un nombre de chapitres : 3
Saisir un nombre de sous-partie : 2
Chapitre 1
 Sous-partie 1.1
 Sous-partie 1.2
Chapitre 2
 Sous-partie 2.1
 Sous-partie 2.2
Chapitre 3
 Sous-partie 3.1
 Sous-partie 3.2
```

# Exercice : Tables de multiplications

1. Créer un programme permettant d'afficher les tables de multiplications de 1 à 10

Table de 1

1 x 1 = 1

1 x 2 = 2

1 x 3 = 3

1 x 4 = 4

...

10 x 10 = 100

# Exercice : Population

1. L'accroissement de la population de Tourcoing est de 0.89%
2. En 2015 la ville comptait 96809 habitants
3. Combien d'années faut-il pour atteindre 120 000 habitants ?
4. Combien d'habitants y aura-t-il cette année-là ?
5. Écrire un programme permettant de résoudre ce problème

La population de Tourcoing dépassera les 120 000 habitants en 2040 pour environ 120815 habitants

# Exercice : Sommes consécutives

1. Déclarer une variable nombre
2. À l'aide de boucles, afficher les suites de nombres qui permettent d'arriver au nombre inscrit précédemment

```
Saisir un nombre : 45
45 = 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9
45 = 5 + 6 + 7 + 8 + 9 + 10
45 = 7 + 8 + 9 + 10 + 11
45 = 14 + 15 + 16
45 = 22 + 23
```

## Exercice : Notes

1. Créer une variable stockant le nombre de notes à saisir
2. Afficher la note la plus haute
3. Afficher la note la plus basse
4. Afficher la moyenne des notes
5. /\ la note doit être comprise entre 0 et 20

```
Combien de notes souhaitez-vous saisir : 3
```

```
note 1 : 10
```

```
note 2 : 20
```

```
note 3 : 15
```

```
La note la plus haute est : 20
```

```
La note la plus basse est : 10
```

```
La moyenne est de : 15.0
```

# Exercice : Nombres premiers

1. Un nombre premier est un nombre divisible par 1 et par soit-même uniquement
2. 1 n'est pas un nombre premier
3. Écrire un programme qui permet de savoir si un nombre est premier

```
Saisir un nombre (> 1) : 3
```

```
3 est un nombre premier
```

## Exercice : Nombre mystère

1. Générer un nombre aléatoire entre 1 et 100
2. Faire saisir un nombre à l'utilisateur
3. Si le chiffre saisi est plus grand, écrire : Le nombre est plus petit
4. Si le chiffre saisi est plus petit, écrire : Le nombre est plus grand
5. Si le chiffre saisi est égal au chiffre aléatoire, écrire : Vous avez gagné en X tentatives

```
Saisir un nombre : 12
Le nombre est plus grand
Saisir un nombre : 14
Vous avez gagné en 2 tentative(s)
```

# Exercice : Factorielle

1. La factorielle d'un nombre positif est le quotient cumulatif des nombres allant de 1 à ce nombre
2. Exemple : la factorielle de 3 est  $1 \times 2 \times 3 = 6$
3. Réaliser un programme qui affiche la factorielle d'un nombre

```
Saisir un nombre : 5
5! = 1 x 2 x 3 x 4 x 5
```

```
Saisir un nombre : 2
2! = 1 x 2
```

## Exercice : Nombre fort

1. Un nombre fort est un nombre dont la somme de la factorielle des chiffres qui le constituent est égale à ce nombre
2. Par exemple :  $145 = 1! + 4! + 5!$  soit  $145 = 1 + 24 + 120$
3. Écrire un programme qui permet de savoir si un nombre est fort

```
Saisir un nombre : 145
145 est un nombre fort
```

# Les Tableaux

# Comprendre la création et la gestion des tableaux

- Définition d'un tableau en Java :
  - Un tableau est une structure de données qui contient plusieurs éléments du même type
- Allocation d'un tableau en Java :
  - Un tableau doit être alloué dans la mémoire avec : "new"
  - Allocation d'un tableau de "n" éléments ayant pour type "type" :
  - ==> new "type"[ "nb" ];
  - Exemple :

```
new int[8] // Alloue un tableau de 8 entiers
```

- L'opérateur "new" renvoie une référence vers un objet "tableau", (une référence est un identifiant unique d'une structure de données)
  - L'exemple : new int[8] renvoie une référence vers ce tableau :
  - [0,0,0,0,0,0,0,0] // Java initialise à 0 chaque élément d'une allocation

## Comprendre la création et la gestion des tableaux

- Déclaration d'un tableau en Java :
  - Il n'y a pas de variable de type "tableau" en Java car ce sont des objets.
  - En revanche, on peut déclarer une variable de type référence vers un objet tableau :
  - ==> "type"[ ] var;
  - "var" est une variable de type référence vers un tableau contenant des éléments de type "type"

- Exemple :

```
int[] tab = new int[8] // tab est la variable référence d'un tableau de 8 entiers
```

- Avec cet exemple: tab = [0,0,0,0,0,0,0,0];
  - Java a référencé dans la variable "tab" un tableau de 8 entiers dont la valeur a été initialisée à 0.

- Déclaration avec allocation de valeur

- On peut aussi allouer un tableau et l'initialiser en même temps:  
==> "type"[ ] var = {X1, X2, X3 ...Xn};
  - Dans ce cas, la longueur du tableau est déduite du nombre "n" d'éléments initialisations.

# Comprendre la création et la gestion des tableaux

- Déclaration d'un tableau avec allocation de valeur (suite)
  - Exemple de déclaration avec initialisation:  
==> float[ ] tab = { 3.6, 2.89, 3.26, 23.42, 3.1416 } ;
    - Dans ce cas, Java alloue le tableau (fait un new) en déduisant la taille du nombre d'éléments [5]
    - Initialise la valeur des éléments du tableau à leurs valeurs respectives.
    - Donne à "tab" la valeur de référence vers le tableau
- Accéder et manipuler les valeurs d'un tableau
  - Connaitre la taille du tableau avec :  
==> « monTableau ».length ; // retourne le nombre d'éléments dans le tableau
  - Accéder à un élément du tableau avec :  
==> « monTableau »[indice] ; // retourne l'élément à cet indice dans le tableau

# Comprendre la création et la gestion des tableaux

- Accéder et manipuler les valeurs d'un tableau (suite)
  - Exemple d'accès à une valeur dans un tableau :  
==> int[ ] tab = { 1, 2, 3, 4, 5 } ;  
Dans ce cas, tab[ 3 ] = 4 ;
    - Les éléments sont indexés à partir de 0, dans notre cas de 0 ,1 , 2 , 3 , 4.
    - Un tableau possède des éléments allant de 0 à tab.length-1
  - Un accès en dehors des limites du tableau provoque systématiquement une erreur à l'exécution de type (ArrayOutOfBoundsException)



# Exercice(s)

## Exercice : 100 éléments

1. Déclarer un tableau de 100 éléments et l'initialiser
2. Afficher les éléments par dizaine séparés d'une virgule

```
0, 1, 2, 3, 4, 5, 6, 7, 8, 9
10, 11, 12, 13, 14, 15, 16, 17, 18, 19
20, 21, 22, 23, 24, 25, 26, 27, 28, 29
30, 31, 32, 33, 34, 35, 36, 37, 38, 39
40, 41, 42, 43, 44, 45, 46, 47, 48, 49
50, 51, 52, 53, 54, 55, 56, 57, 58, 59
60, 61, 62, 63, 64, 65, 66, 67, 68, 69
70, 71, 72, 73, 74, 75, 76, 77, 78, 79
80, 81, 82, 83, 84, 85, 86, 87, 88, 89
90, 91, 92, 93, 94, 95, 96, 97, 98, 99
```

# Exercice : Tableau positif

1. Déclarer deux tableaux de 10 éléments
2. Le premier tableau contiendra des nombres négatifs et positifs
3. Ajouter tous les éléments positifs du premier tableau dans le second tableau

```
tableauSource = [-5, 3, 24, -12, -10, 5, 100, -2, -71, 0]
tableauDestination = [3, 24, 5, 100, 0, 0, 0, 0, 0, 0]
```

# Exercice : Valeur maximale

1. Ecrire un programme qui permet de trouver la valeur maximale dans un tableau

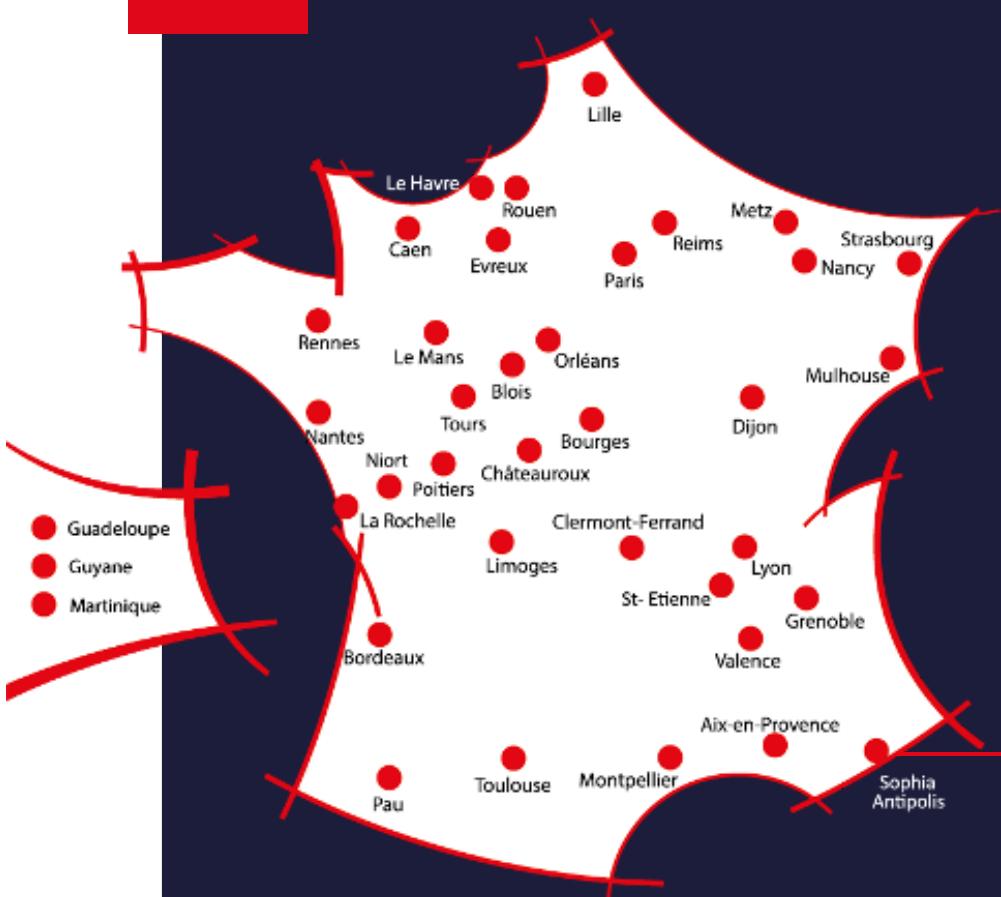
```
tableau = [23, 150, 12, 28, 59, 2]
La valeur maximale est 150 à l'indice 1
```

## Exercice : Tableaux égaux

1. Ecrire un programme qui permet de vérifier si 2 tableaux sont égaux
2. Vérifier la taille des tableaux et l'ordre des valeurs

```
tableau1 = [1, 4, 2, 3]
tableau2 = [1, 4, 2, 3]
les tableaux sont égaux
tableau1 = [1, 4, 2, 3]
tableau2 = [1, 4, 3, 2]
les tableaux ne sont pas égaux
tableau1 = [1, 4, 2, 3]
tableau2 = [1, 4, 3]
les tableaux ne sont pas égaux
```

**Merci pour votre attention  
Des questions ?**



Découvrez également  
l'ensemble des stages à votre disposition  
sur notre site [m2iformation.fr](http://m2iformation.fr)

[m2iformation.fr](http://m2iformation.fr)

