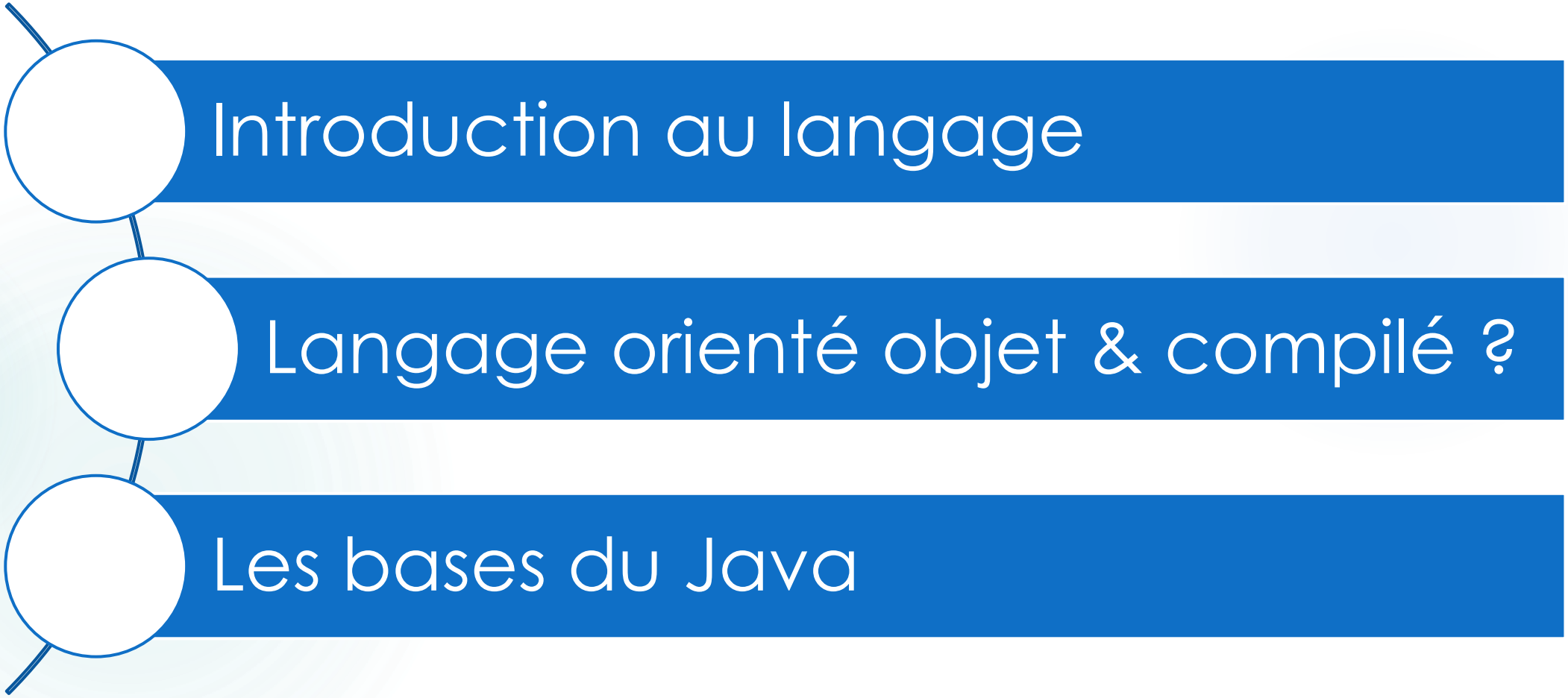
JAVA

PLAN DE LA FORMATION



INTRODUCTION AU LANGAGE

- Programmation ?

- Créé en 1995 par SUN
- Puis J2SE à partir de 1.4 et enfin JSE à partir de 1.**6**
- On parle couramment de Java **6**

- **Java 15** depuis mars 2020 mais nous utiliserons **JAVA 11 pour le cours**

INTRODUCTION AU LANGAGE

Version	Dénomination	Date de sortie	Statut	Période de maintenance	Support étendu
1.0	Java 1.0	1996	N'est plus utilisé	1996-2000	
1.1	Java 1.1	1997	N'est plus utilisé	1997-2000	
1.2	J2SE 1.2	1998	N'est plus utilisé	2000-2006	
1.3	J2SE 1.3	2000	Obsolète	2000-2001	
1.4	J2SE 1.4	2002	Obsolète	2000-2008	
1.5	J2SE 5.0	2004	Obsolète	2002-2009	Mai 2015
1.6	Java SE 6	2006	Obsolète	2005-2013	Décembre 2018
1.7	Java SE 7	2011	Stable	2011-2015	Juillet 2022
1.8	Java SE 8	2014	Stable	2014-2018	Mars 2025
1.9	Java SE 9	2017	Stable	2017-2018	Non LTS
…	…	…	…	…	…

INTRODUCTION AU LANGAGE

Version	Dénomination	Date de sortie	Période de maintenance	Support étendu
10	JAVA 10	Mars 2018	03/2018-09/2018	Non LTS
11	JAVA 11	Septembre 2018	Septembre 2023	Septembre 2026
12	JAVA 12	19/03/2019	Septembre 2019	Non LTS
13	JAVA 13	Septembre 2019	Mars 2020	Non LTS
14	JAVA 14	Septembre 2020	…	Non LTS
15	JAVA 15	Mars 2020	…	Non LTS
…	…	…	…	…

D'après le site d'Oracle, les dates décrites ci-dessus sont susceptibles d'être modifiées.
LTS : Support à Long Terme

http://www.oracle.com/technetwork/java/eol-135779.html (décembre 2020)

INTRODUCTION AU LANGAGE

Il existe **trois** environnements permettant de créer des programmes pour des plates-formes différentes :

- **J2SE**(Java 2 Standard Edition) : permet de développer des applications dites « client lourd », par exemple Word, Excel, JMerise… Toutes ces applications sont des « clients lourds » , elles s'exécutent sur la machine directement.

- **J2EE**(Java 2 Enterprise Edition) : permet de développer des applications web en Java. On parle aussi de clients légers.

- **J2ME**(Java 2 Micro Edition) : permet de développer des applications pour appareils portables, comme des téléphones portables, des PDA…

INTRODUCTION AU LANGAGE

Lors de la création du langage Java, il avait été décidé que ce langage devait répondre à cinq objectifs :

- simple, **orienté objet** et familier ;
- robuste et sûr ;
- indépendant de la machine employée pour l'exécution (« **Write once, run anywhere** »);
- très performant ;
- **compilé**, **multi-tâches** et dynamique.

2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

Orienté **objet** ?

La programmation orientée objet (P.O.O) est une programmation basée sur des objets qui possèdent des attributs et des méthodes.

Cela permet d'avoir une programmation proche de la réalité.

2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

Orienté **objet** ?

- En P.O.O, nous pourrions utiliser un objet de type ordinateur.




2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

Orienté **objet** ?

2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

Compilé ?

Besoin d'un **JDK** : Java Development Kit

Besoin d'un **JRE** : Java Runtime Environment

2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

Multi-tâches ?

- En Java, nous pouvons exécuter plusieurs tâches en même temps mais attention à ne pas faire deux tâches différentes sur un même objet !

2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

La P.O.O tourne autour des objets. Chaque objet est unique et défini par :

- Une ..... (nom de l'objet)
- Ses .....
- Un ..... (actions qu'il est capable de réaliser, ses .....).

2 LANGAGE ORIENTÉ OBJET & COMPILÉ ?

Une …….. est un moule pour représenter des objets

qui ont des caractéristiques communes.

Exemple : Ordinateur est le 'moule' pour créer un Mac Pro, un Mac, un Asus Chromebook…

Ce sont ces …….. que nous représentons en UML.

LES BASES DU JAVA :
LES VARIABLES ET CONSTANTES

- **Variable** ?

.....

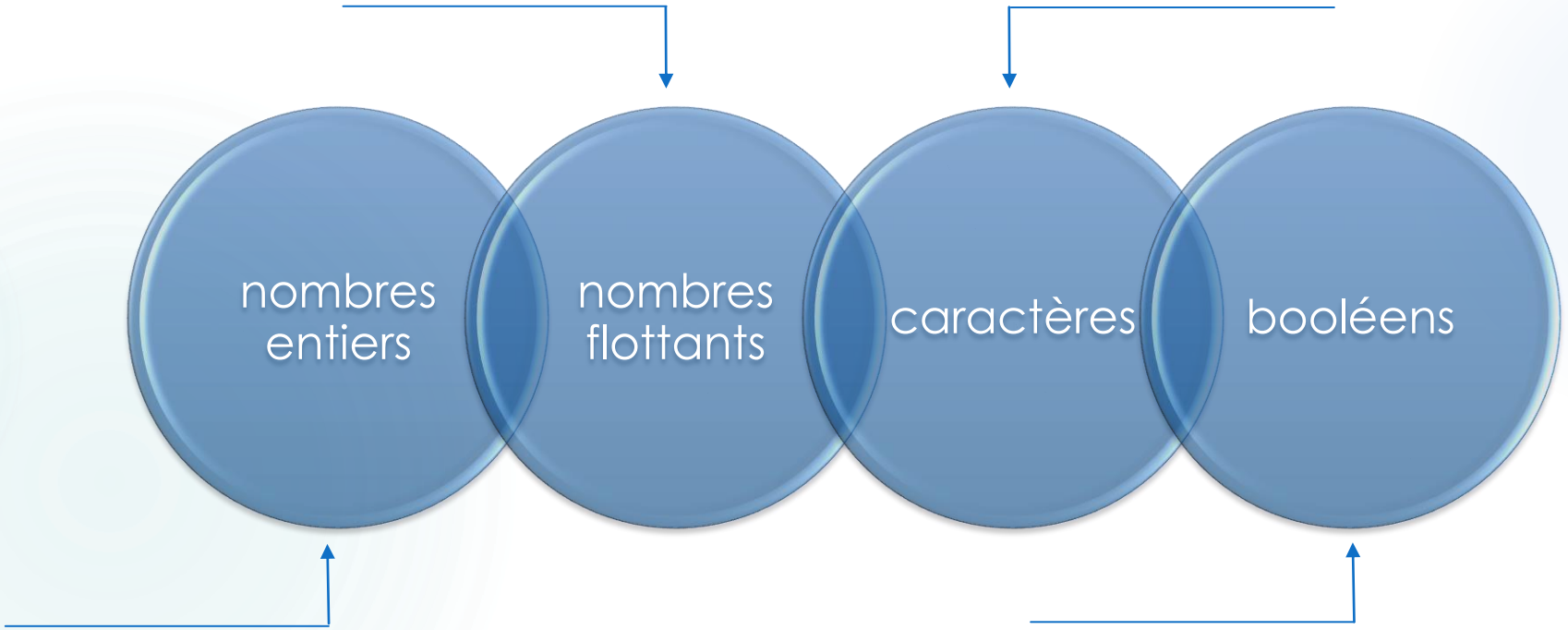
- **Constante** ?

Conventionnellement en MAJUSCULES.

Commence par le mot clef **final**

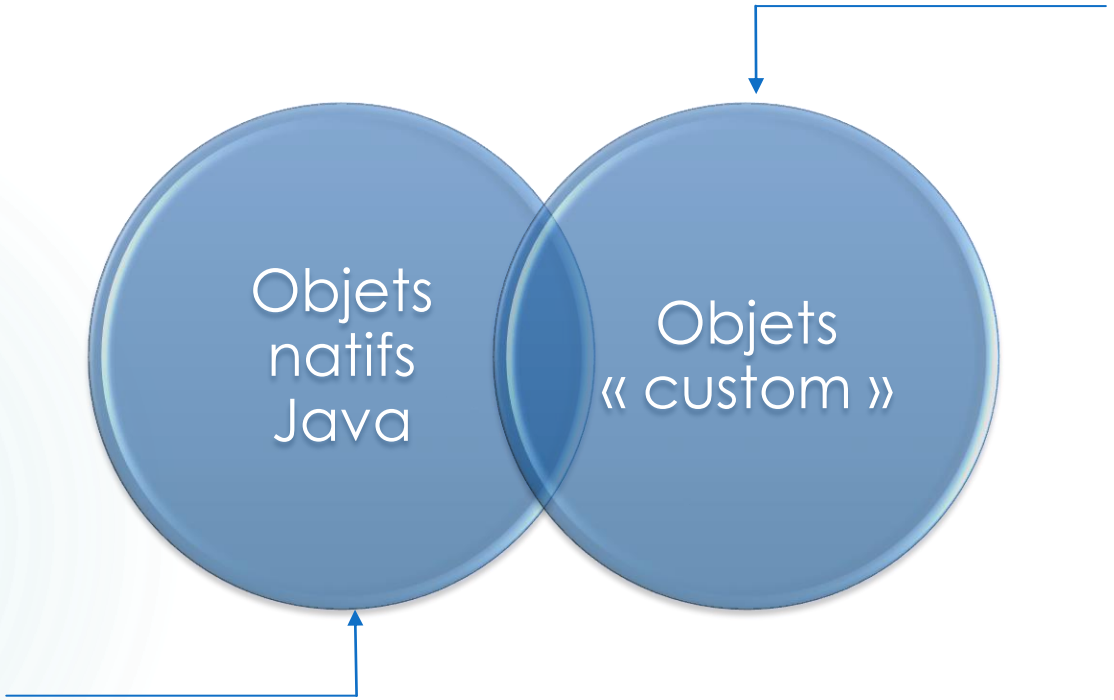
LES BASES DU JAVA :

LES TYPES PRIMITIFS

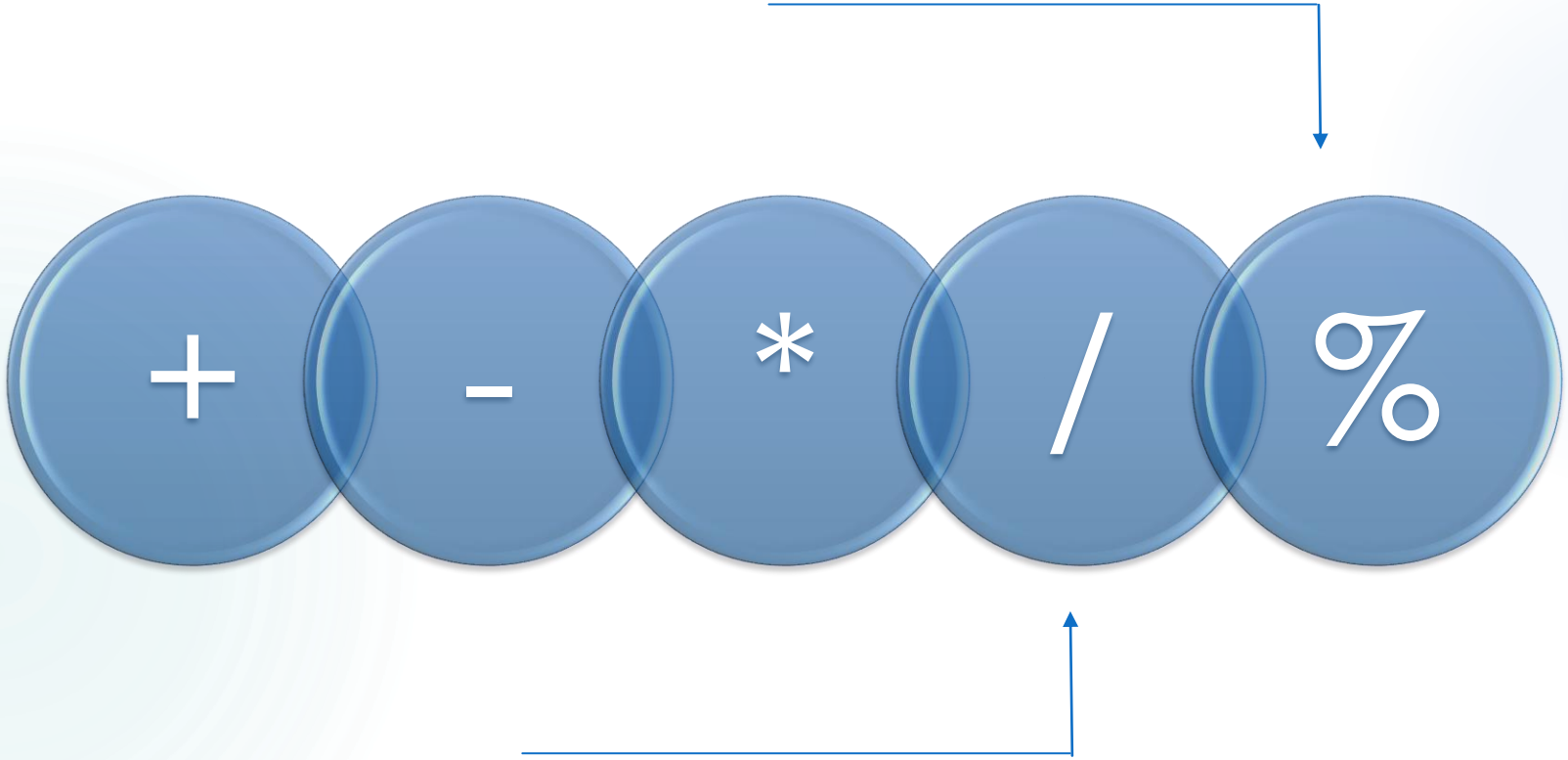


LES BASES DU JAVA :

LES AUTRES TYPES



LES BASES DU JAVA :
LES OPÉRATEURS ARITHMÉTIQUES



LES BASES DU JAVA :
LES OPÉRATEURS RELATIONNELS

- Une comparaison renvoie toujours une valeur de type **booléen**.

Comparaison exacte	Inférieur	Supérieur
== !=	< <=	> >=

/!\ ATTENTION :

Ne pas confondre == et **=** !

= sert à l'**affectation** d'une valeur à une variable.

LES BASES DU JAVA :
LES OPÉRATEURS LOGIQUES

- Une comparaison renvoie toujours une valeur de type **booléen**.

Négation	Et	Ou
!	& &&	| ||

Quelle est la différence entre (& et &&) ou (| et ||) ?

.....

.....

LES BASES DU JAVA : INCRÉMENTATION & DÉCRÉMENTATION

Il existe en java de nombreuses manières d'incrémenter ou décrémenter une variable mais **attention aux subtilités** !

Soit i, une variable de type int :

Incrémenter	Décrémenter
i = i +1 i++ ++i	i = i - 1 i-- --i

Quelle est la différence entre mettre le ++ ou -- **devant** ou **derrière** la variable?

……

……

LES BASES DU JAVA :
LE CAST

- Le cast permet de forcer le type d'une variable mais attention à la perte de données.

```
long m = 10L;
int n = (int) m; //Pour caster, il suffit de mettre le type désiré entre parenthèses avant la valeur à caster
```

- Cela peut-être très utile lors des divisions par exemple :

```
int a = 5;
int b = 2;
int c = 5/2; //c=2;
double d = (double) (5/2); // d=2.5;
```

LES BASES DU JAVA :
LE CAST

- Le cast permet de forcer le type d'une variable.

```
String m = « 56 »;
int n = Integer.parseInt(m);
Idem pour les classes Double, Float,…
```

- Et inversement :

```
int a = 59;
String b = String.valueOf(a);
```

LES BASES DU JAVA :
LES TABLEAUX

- Création

```
<type du tableau> [] <nom du tableau> = { <contenu du tableau>};
int[] tableauEntier = new int[6];
```

- Exemple

```
int [] tableauEntier = {0,1,2,3,4,5,6,7,8,9};
String [] tableauChaine = {"chaine1", "chaine2", "chaine3" ,
"chaine4"};
```

LES BASES DU JAVA :
LES TABLEAUX

- Accéder à une valeur

```
int [] tableauEntier = {0,1,2,3,4,5,6,7,8,9};
int val3 = tableauEntier[2];
```

/!\ Les indices des tableaux commencent à 0.

Le dernier élément du tableau sera donc à l'indice tailleDuTableau-1

LES BASES DU JAVA :
LES LISTES

- Une liste est un **ensemble d'objets du même type**. Celles-ci peuvent être modifiées comme bon vous semble à la différence des énumérations.

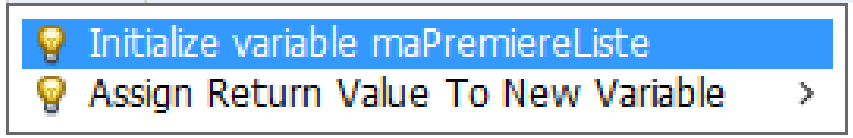
- L'objet liste est implémentée via la classe **ArrayList**:

```
List<String> maPremiereListe = new ArrayList<>();
```

LES BASES DU JAVA :
LES LISTES

- Puisque ArrayList est un objet, nous sommes obligés de l'instancier ! Instancié à l'aide de new. Si nous ne le faisons pas, notre objet sera toujours null.

```
List<String> maPremiereListe;
maPremiereListe.add("ma premiere valeur"); //Retourne NPE
```



LES BASES DU JAVA :

LES LISTES

- Puisque ArrayList est un objet, nous sommes obligés de l'instancier ! Instancié à l'aide de new. Si nous ne le faisons pas, notre objet sera toujours null.

```
List<String> maPremiereListe = new ArrayList<>();
maPremiereListe.add("ma premiere valeur"); //Correct
```

LES BASES DU JAVA :
LES LISTES

Quelques méthodes utiles :

- **add()** : permet d'ajouter un élément à la liste;
- **get(i)** : permet de récupérer un objet dont l'indice dans la liste est i;
- **remove(i)** : permet de supprimer un objet dont l'indice dans la liste est i;
- **isEmpty()** : renvoie si la liste est vide ou non;
- …

EXERCICE

JAVA_Partie 1

LES BASES DU JAVA : LA CLASSE MAIN

- La méthode **main** est une méthode particulière. 
- Elle définit le **point d'entrée du programme** et se déclare comme ceci :

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

LES BASES DU JAVA : QUELQUES CONVENTIONS D'ÉCRITURE

- Les noms des classes commencent par des majuscules.
- Les noms des variables et attributs commencent par des minuscules.
- Le choix des noms de classes, d'attributs et de variables est très important ! Cela doit être explicite.
- Si un nom de variable est long, nous utilisons le **camelCase**.
- On **indente** toujours son code.

LES BASES DU JAVA :
CRÉATION DES MÉTHODES

Pour créer les méthodes :

public/private Type nomDeLaMethode(**TypeArg arguments**);

Si **Type** == **void** alors on ne retourne **rien**.

Si **Type** != void alors on n'oublie pas le **return** dans la méthode !

TypeArg est le type correspondant à l'argument passé en paramètre de la méthode.

Le **return** permet de renvoyer à la méthode qui appelle la seconde méthode la valeur désirée.

La valeur désirée doit absolument être du même **Type** que celui déclaré.

LES BASES DU JAVA :
CRÉATION DES MÉTHODES

Pour créer les méthodes :

public/private Type nomDeLaMethode(**TypeArg arguments**);

Les arguments sont séparés par des virgules.

public/private Type nomDeLaMethode(**TypeArg arg1, TypeArg arg2**);

LES BASES DU JAVA :
CRÉATION DES MÉTHODES

Pour appeler la méthode, il suffit de faire :

nomDeLaMethode(**arguments**)

S'il y a un return dans la méthode, on peut affecter la valeur de retour à une variable pour l'utiliser par la suite :

Type variable = nomDeLaMethode(**arguments**);

LES BASES DU JAVA :
LES CONDITIONS

- Si… alors… sinon

```
if (condition)
{
    instruction;
}
else if {
    instruction;
}
else
{
    instruction;
}
```

LES BASES DU JAVA :
LES CONDITIONS – OPÉRATEURS LOGIQUES

- **&&** : et

Théorème de De Morgan : &&			
0	0		0
0	1		0
1	0		0
1	1		1

LES BASES DU JAVA :
LES CONDITIONS – OPÉRATEURS LOGIQUES

- || : ou

Théorème de De Morgan : ||			
0	0		0
0	1		1
1	0		1
1	1		1

/!\ Si la première condition est vraie,

on ne prend même pas la peine de tester la seconde !

LES BASES DU JAVA :
LES CONDITIONS

- Switch… case

```
switch (condition)
{
     case 1 :
          instruction;
          break;
     case 2 :
     case 3 :
          instruction;
          break;
     default :
          instruction;
}
```

LES BASES DU JAVA :
LES CONDITIONS

- Switch… case (depuis Java 12)

```
switch (condition)
{
    case 1        -> instruction;
    case 2, 3     -> instruction;
    default       -> instruction;
}
```

LES BASES DU JAVA :
LES BOUCLES

- Do… While

```
do
{
    instruction;
} while (condition);
```

/!\ On fait au moins une fois l'instruction et on teste la condition après.

Assurez-vous que la condition sera vérifiée au moins une fois !

LES BASES DU JAVA :
LES BOUCLES

- While

```
while(condition)
{
    instruction;
}
```

/!\ On ne passe pas forcément dans la condition contrairement à do…while.

Attention au while(true) !

LES BASES DU JAVA :
LES BOUCLES

- For

```
for(int i=0; i<nb ; i++)
{
    instruction;
}
```

- break : .....
- continue : .....

LES BASES DU JAVA :
LES BOUCLES

- For each

```
for(TypeObjet obj : listeObjet)
{
    instruction;
}
```

- Impossible d'arrêter la boucle en cours de route

EXERCICE

JAVA_Partie 2

BIBLIOGRAPHIE

- Programmer en Java, C. DELANNOY, EYROLLES, 2016
- https://portail.greyc.fr/sites/default/files/documentation/developpement/bonnes-pratiques.pdf, C.COURONNE, 2016
- Cours de JAVA pour l'IUT d'Allier, Paul CHECCHIN, 2011-2013
- https://openclassrooms.com/courses/programmez-en-oriente-objet-avec-c/introduction-a-la-programmation-orientee-objet-1