



Cours Java

Date : 28/04/2023

Sommaire

1. Introduction à Java : histoire, caractéristiques, installations, environnements de développement.
2. Les bases du langage Java : syntaxe, variables, types de données, opérateurs, boucles, conditions, fonctions.
3. Les objets et les classes : introduction aux objets, les classes, les constructeurs, les méthodes, les champs, les encapsulations.
4. Les tableaux et les collections : les tableaux en Java, les collections, les ArrayList, les LinkedList, les HashMap.
5. Les exceptions et les gestion d'erreurs : les exceptions en Java, les try-catch, les throw, les finally, les blocs de gestion d'erreurs.
6. Les fichiers et la gestion des entrées/sorties : la gestion des fichiers en Java, les entrées/sorties standard, les entrées/sorties de fichiers, les entrées/sorties binaires.
7. La programmation orientée objet avancée : les héritages, les interfaces, les polymorphismes, les références d'objets, les génériques.

-
- 8. Les frameworks Java : introduction aux frameworks Java populaires tels que Spring, Hibernate, Struts, etc.
 - 9. Les applications web avec Java : développement d'applications web en utilisant Java Servlets, JSP et frameworks web tels que Spring MVC.
 - 10. Projet final : Mettre en pratique les connaissances acquises pour développer une application Java complète.

Ce parcours de formation est conçu pour vous donner une base solide en Java et vous permettre de développer des applications simples à l'aide de ce langage de programmation populaire.



Introduction à Java

Java est un langage de programmation populaire qui a été créé par James Gosling en 1991. Au fil des ans, il est devenu un langage de programmation largement utilisé pour le développement de logiciels d'entreprise, d'applications mobiles et de jeux vidéo.

En 2009, Oracle Corporation a acquis Sun Microsystems, la société qui avait créé Java en 1995. Depuis lors, Oracle a continué à développer et à améliorer Java, en publiant régulièrement des mises à jour de la plateforme Java Standard Edition (Java SE) et de la plateforme Java Enterprise Edition (Java EE).

Java est devenu très populaire en raison de sa portabilité, de sa sécurité et de sa robustesse. Les programmes Java peuvent fonctionner sur n'importe quel système d'exploitation, ce qui permet aux développeurs de créer des applications une seule fois pour les exécuter sur de multiples plateformes. De plus, Java est équipé de fonctions de sécurité intégrées, telles que la vérification du bytecode, qui empêchent les programmes malveillants d'être exécutés. Enfin, la structure solide de Java et sa grande bibliothèque de classes prédéfinies facilitent la création d'applications robustes et fiables.

Java est également très populaire pour la création d'applications Web, de jeux vidéo, d'applications mobiles et de logiciels d'entreprise. Les bibliothèques Java sont nombreuses et diverses, couvrant des domaines allant des graphiques et de l'interface utilisateur à la communication en réseau et à la sécurité. Les développeurs peuvent facilement intégrer ces bibliothèques dans leurs applications pour accélérer leur développement et améliorer leur fonctionnalité.

L'une des principales caractéristiques de Java est sa portabilité, ce qui signifie que les programmes écrits en Java peuvent être exécutés sur n'importe quel système d'exploitation sans nécessiter de modifications supplémentaires. Cela est possible grâce à l'utilisation de la machine virtuelle Java (JVM), qui traduit le code Java en code binaire compréhensible par le système d'exploitation. Il faut aussi savoir que Java est un langage dit Objet, c'est-à-dire que contrairement à d'autres langages qui peuvent être multi-paradigme, Java est un langage haut niveau qui ne vous proposera pas plusieurs manières de développer vos applications. La POO est une caractéristique de Java

importante car elle permet aux développeurs de créer des programmes modulaires et évolutifs en utilisant des objets pour représenter des données et des fonctions. Nous aborderons les concepts clés de la POO tels que l'encapsulation, l'héritage et le polymorphisme, et comment ils sont utilisés dans Java pour créer des programmes plus robustes et plus faciles à maintenir. Cette formation aura donc en thème central la POO, autrement dit la programmation orientée objet.

Cependant, il est important de noter que Java est un langage de programmation très vaste, et qu'il est donc impossible de tout apprendre à travers une seule formation ou un seul cours. Les développeurs peuvent se spécialiser dans des domaines spécifiques de Java, tels que le développement d'applications mobiles Android, la création de jeux vidéo ou la programmation d'applications d'entreprise.

Voici 10 exemples d'applications réelles développées en Java :

1. Applications de trading en ligne : De nombreuses applications de trading en ligne sont développées en Java en raison de la sécurité, de la fiabilité et de la rapidité qu'offre le langage.
2. Jeux vidéo : Les jeux vidéo tels que Minecraft, Runescape et RuneScape 3 sont tous développés en Java.
3. Applications mobiles Android : Java est le langage de programmation standard pour le développement d'applications mobiles Android.
4. Systèmes de gestion de base de données : De nombreux systèmes de gestion de base de données tels que Apache Cassandra, HBase et OrientDB sont développés en Java.
5. Logiciels de traitement de données : Les logiciels de traitement de données tels que Apache Hadoop et Apache Spark sont tous deux développés en Java.
6. Applications de surveillance en temps réel : Les applications de surveillance en temps réel telles que Nagios et Zabbix sont développées en Java en raison de la rapidité et de la fiabilité du langage.
7. Applications de messagerie instantanée : Des applications de messagerie instantanée telles que Jabber et Jitsi sont développées en Java.

-
8. Systèmes de gestion de contenu : De nombreux systèmes de gestion de contenu tels que Magnolia CMS et Hippo CMS sont développés en Java.
 9. Applications de gestion de projet : Les applications de gestion de projet telles que GanttProject et OpenProj sont développées en Java.
 10. Applications de médias sociaux : De nombreux sites de médias sociaux tels que LinkedIn, Twitter et Facebook utilisent Java pour leur développement backend.

Ces exemples illustrent la diversité des domaines d'application de Java et montrent comment il est utilisé dans des industries variées telles que la finance, les jeux vidéo, les réseaux sociaux, la gestion de données et bien d'autres.

Les dernières versions de Java Standard Edition (Java SE) et Java Enterprise Edition (Java EE) ont apporté des améliorations significatives. La version la plus récente de Java SE, Java SE 17, est sortie en septembre 2021 avec plusieurs améliorations telles que des améliorations de performance, l'ajout de nouvelles fonctionnalités de sécurité et la mise à jour de bibliothèques importantes. De même, Java EE a été mis à jour pour inclure de nouvelles fonctionnalités telles que la prise en charge de l'Internet des objets (IoT) et une meilleure intégration avec les services cloud. Ces mises à jour garantissent que Java reste un langage de programmation pertinent et fiable pour les développeurs dans une variété de domaines.

En résumé, Java est un langage de programmation populaire qui offre de nombreux avantages, tels que la portabilité, la sécurité et la robustesse. Les bibliothèques Java sont vastes et diverses, ce qui facilite le développement d'applications de haute qualité. Cependant, Java est une technologie très vaste et les développeurs doivent se spécialiser dans des domaines spécifiques pour devenir des experts dans ce domaine.

Java est un langage de programmation puissant et portable qui offre de nombreuses fonctionnalités et avantages pour le développement de logiciels. Pour commencer à programmer en Java, vous devrez installer le JDK et utiliser un IDE pour écrire et tester votre code.

Comme pour tous les langages de programmation, la pratique est essentielle pour devenir compétent en Java. Plus on pratique et on développe des projets en Java, plus

on acquiert une expertise dans ce langage. C'est pourquoi mes trois conseils pour devenir compétent en Java sont les suivants :

- **PRATIQUEZ**
- **PRATIQUEZ**
- **PRATIQUEZ**

Pour commencer à programmer en Java, vous devrez d'abord installer le kit de développement Java (JDK) sur votre ordinateur. Le JDK comprend tous les outils nécessaires pour développer, tester et déboguer des applications Java. Il est disponible gratuitement sur le site Web d'Oracle.

Une fois que vous avez installé le JDK, vous pouvez utiliser un environnement de développement intégré (IDE) pour écrire et tester votre code. Les IDE les plus populaires pour le développement Java sont Eclipse, NetBeans et IntelliJ IDEA. Ces outils offrent des fonctionnalités telles que la coloration syntaxique, la complétion automatique de code et le débogage.

Installation du JDK

D'abord, je vais vous expliquer brièvement la différence entre le JSE, le JDK et le JRE.

Le JSE (Java Standard Edition) est l'environnement d'exécution standard de Java. Il fournit les bibliothèques et les outils nécessaires pour développer et exécuter des applications Java sur n'importe quelle plateforme. Le JSE est également appelé la "plateforme Java" ou "Java SE".

Le JDK (Java Development Kit) est un ensemble complet d'outils nécessaires pour développer des applications Java. Le JDK inclut le JSE ainsi que des outils supplémentaires tels que le compilateur Java (javac) et l'outil de création de JAR (jar). Le JDK est principalement destiné aux développeurs qui souhaitent créer des applications Java.

Le JRE (Java Runtime Environment) est l'environnement d'exécution pour les applications Java. Il inclut les bibliothèques et les fichiers nécessaires pour exécuter des applications Java sans avoir à installer le JDK. Le JRE est principalement destiné

aux utilisateurs qui souhaitent exécuter des applications Java sur leur machine, sans avoir besoin de les développer.

En résumé, le JSE est l'environnement d'exécution standard de Java, le JDK est un ensemble complet d'outils pour développer des applications Java et le JRE est l'environnement d'exécution pour les applications Java.

En cherchant Java se sur google on obtient comme premier résultat le site de Oracle.

Google search results for "java se". The first result is the Oracle Java SE page, titled "Java SE | Oracle Technology Network". The snippet describes Java Platform, Standard Edition (Java SE) as letting you develop and deploy Java applications on desktops and servers. The URL is https://www.oracle.com/java/technology/se/index.html.

Puis on rendez vous sur le site officiel et cliquez sur Java se :

The Oracle Java SE homepage features the Java logo (a steaming coffee cup) and the text "Java SE at a Glance". A red box highlights the text "Java Platform, Standard Edition (Java SE) lets you develop and deploy Java applications on desktops and servers. Java offers the rich user interface, performance, versatility, and security that today's applications require." Another red box highlights the "Java SE at a Glance" section title.

Là, cliquez sur “Download Java” et choisissez la version et sélectionnez votre os (Linux, Mac ou Windows). Enfin Cliquez sur le lien qui aura la bonne extension.

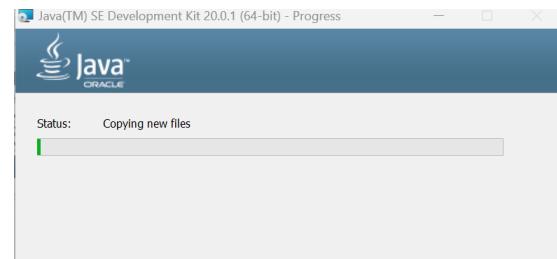
JDK Development Kit 20.0.1 downloads

JDK 20 binaries are free to use in production and free to redistribute, at no cost, under the terms of the Oracle Java License Agreement. JDK 20 will receive updates under these terms, until September 2023 when it will be succeeded by Java 21.

[Linux](#) [macOS](#) [Windows](#)

Product/file description	File size	Download
x64 Compressed Archive	180.81 MB	https://download.oracle.com/java/20/jdk/20.0.1/binaries/jdk-20.0.1_linux-x64_bin.tar.gz
x64 Installer	159.95 MB	https://download.oracle.com/java/20/jdk/20.0.1/binaries/jdk-20.0.1_linux-x64_bin.rpm
x64 MSI Installer	158.74 MB	https://download.oracle.com/java/20/jdk/20.0.1/binaries/jdk-20.0.1_windows-x64_bin.exe

Par exemple pour moi au moment où je fais ce support de formation nous en sommes à la version 20 de java, et je suis sur un ordinateur qui tourne sur Windows je choisi donc le fichier avec l'extension .exe. Ensuite j'installe java en exécutant le fichier que je viens de télécharger, et je clique sur next jusqu'à voir la barre de progression et le message de succès.





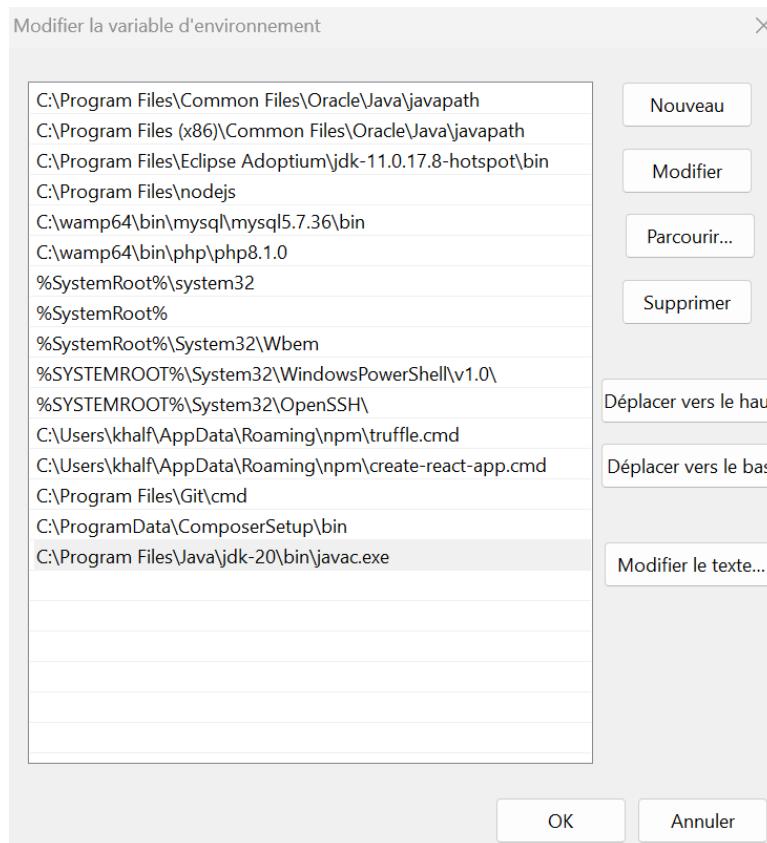
On appuie sur le bouton close est l'installation est finie!

Nous sommes prêts à faire du code en Java.

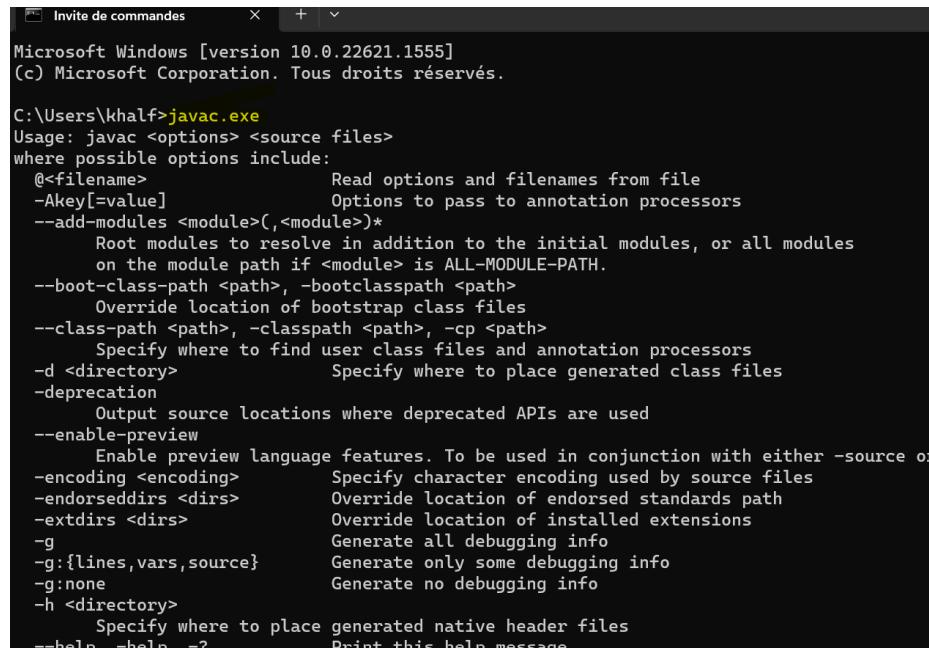
Nous pouvons nous attarder à télécharger un IDE dédié à Java comme Eclipse ou IntelliJ mais pour l'instant nous ne le ferons pas. À la place, nous allons plutôt faire en sorte que Java puisse s'exécuter en ligne de

commande, en l'ajoutant à nos variables d'environnement.

Pour cela vous devez chercher dans le dossier "Programme" -> "Java" -> "jdk-20" -> "bin" le fichier qui s'appelle javac.exe. Et c'est ce chemin que vous devez ajouter à votre variable d'environnement Path.



Maintenant en tapant dans un cmd la commande "javac.exe", vous constatez que cela fonctionne :



```

Invite de commandes
Microsoft Windows [version 10.0.22621.1555]
(c) Microsoft Corporation. Tous droits réservés.

C:\Users\khalf>javac.exe
Usage: javac <options> <source files>
where possible options include:
  @<filename>           Read options and filenames from file
  -Akey[=value]           Options to pass to annotation processors
  --add-modules <module>(<module>)*
    Root modules to resolve in addition to the initial modules, or all modules
    on the module path if <module> is ALL-MODULE-PATH.
  --boot-class-path <path>, -bootclasspath <path>
    Override location of bootstrap class files
  --class-path <path>, -classpath <path>, -cp <path>
    Specify where to find user class files and annotation processors
  -d <directory>         Specify where to place generated class files
  -deprecation
    Output source locations where deprecated APIs are used
  --enable-preview
    Enable preview language features. To be used in conjunction with either -source or
  -encoding <encoding>      Specify character encoding used by source files
  -endorseddirs <dirs>    Override location of endorsed standards path
  -extdirs <dirs>          Override location of installed extensions
  -g
  -g:{lines,vars,source}   Generate all debugging info
  -g:none                 Generate only some debugging info
  -g:none                 Generate no debugging info
  -h <directory>          Specify where to place generated native header files
  --help, -help, -?        Print this help message

```

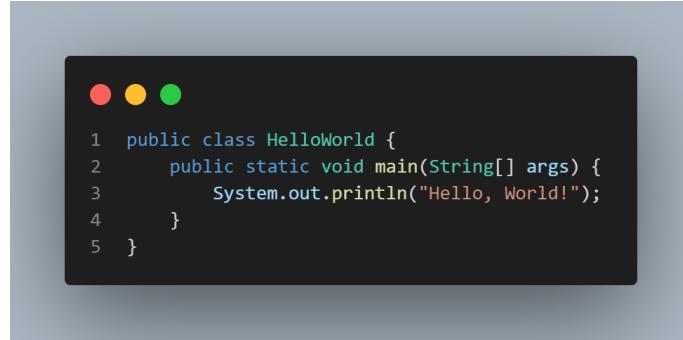
Bien passons maintenant à la pratique en voyant un peu la syntaxe et les bases de Java.

Les bases du langage: La syntaxe.

En Java, la syntaxe d'un programme est structurée et se base sur des classes. Chaque classe est définie dans un fichier portant le même nom que la classe.

Par exemple, supposons que nous voulions créer une classe simple appelée "HelloWorld" qui affiche le message "Hello, World!" à l'écran. Voici comment le fichier et la classe seraient nommés :

1. Ouvrez votre éditeur de texte préféré et créez un nouveau fichier nommé "HelloWorld.java". L'extension ".java" indique que c'est un fichier source Java.
2. À l'intérieur du fichier "HelloWorld.java", déclarez la classe "HelloWorld" de la manière suivante :



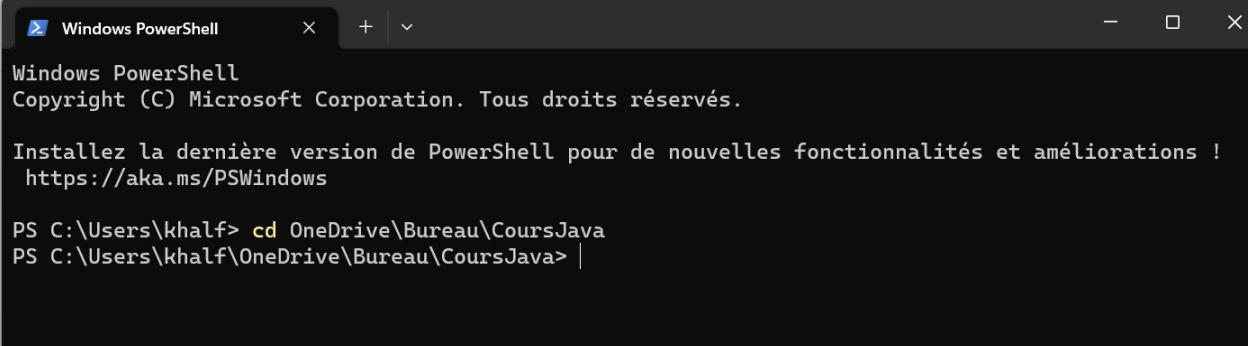
```
● ● ●
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Hello, World!");
4     }
5 }
```

Analysons cette syntaxe :

- La ligne '**public class HelloWorld**' définit la classe "HelloWorld". Le mot-clé '**public**' indique que la classe est accessible à partir d'autres classes. **Le nom de la classe est identique au nom du fichier et commence toujours par une majuscule.**
- À l'intérieur de la classe, nous avons une méthode un peu spéciale appelée '**main**'. C'est le point d'entrée de notre programme Java. La méthode '**main**' est toujours déclarée de cette façon.
- La ligne '**public static void main(String[] args)**' déclare la méthode '**main**'. La déclaration inclut plusieurs éléments : le modificateur d'accès '**public**', le mot-clé '**static**' qui indique que la méthode appartient à la classe elle-même et non à une instance, '**void**' qui spécifie que la méthode ne retourne pas de valeur, et '**(String[] args)**' qui définit les arguments que la méthode peut recevoir (dans ce cas, un tableau de chaînes de caractères).
- Enfin, à l'intérieur de la méthode '**main**', nous avons une seule instruction : '**System.out.println("Hello, World!");**'. Cette instruction affiche le message "Hello, World!" à l'écran. La méthode '**println**' est une méthode de la classe '**System**' qui permet d'afficher du texte sur la console. La ligne se termine par un point virgule pour signifier la fin de déclaration de l'instruction.

Pour exécuter ce programme, vous devez d'abord compiler le fichier source en bytecode, puis exécuter le bytecode généré. Vous pouvez utiliser la commande '**javac**' pour compiler et '**java**' pour exécuter.

Voici un exemple, vous allez d'abord ouvrir PowerShell et vous placer dans le bon dossier grâce à '**cd**' (commande directory) :

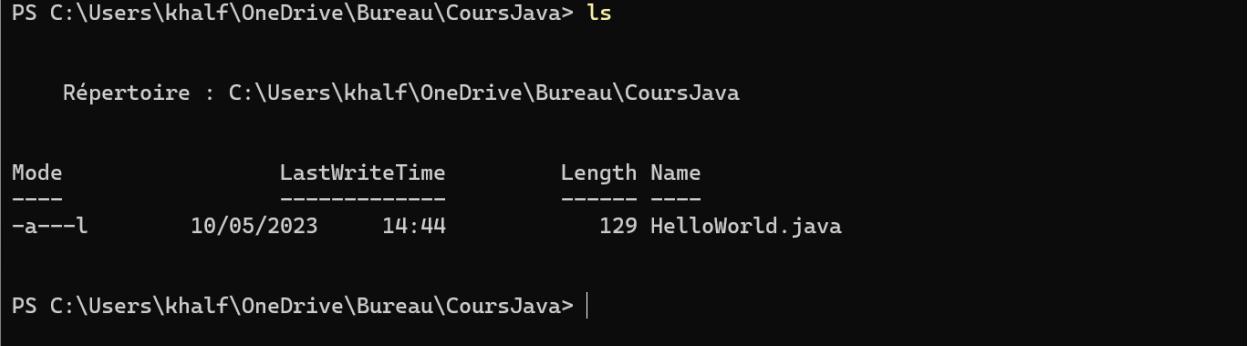


```
Windows PowerShell
Copyright (C) Microsoft Corporation. Tous droits réservés.

Installez la dernière version de PowerShell pour de nouvelles fonctionnalités et améliorations !
https://aka.ms/PSWindows

PS C:\Users\khalf> cd OneDrive\Bureau\CoursJava
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> |
```

En exécutant la commande '**ls**' (abréviation de list en anglais), qui permet de lister le contenu d'un répertoire, cela nous permet de vérifier que le fichier que nous venons de créer s'y trouve bien.



```
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> ls

Répertoire : C:\Users\khalf\OneDrive\Bureau\CoursJava

Mode          LastWriteTime    Length Name
----          -----        ---- 
-a---l       10/05/2023     14:44      129 HelloWorld.java

PS C:\Users\khalf\OneDrive\Bureau\CoursJava> |
```

Une fois le programme compilé avec '**javac**', vous pouvez constater qu'un nouveau fichier a été créé dans votre dossier et que celui-ci porte le même nom que la classe contenue dans le fichier.

```
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> javac .\HelloWorld.java
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> ls
```

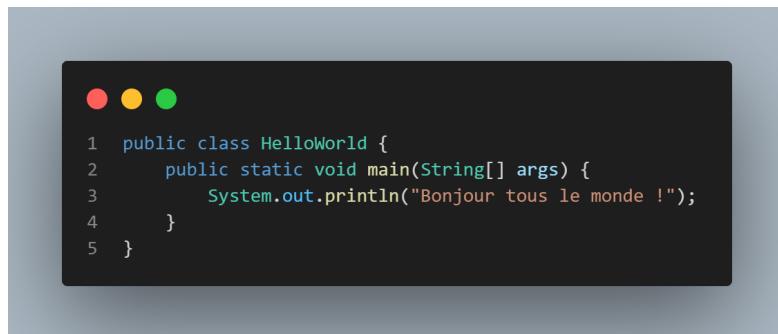
Répertoire : C:\Users\khalf\OneDrive\Bureau\CoursJava

Mode	LastWriteTime	Length	Name
-a---	10/05/2023 15:11	427	HelloWorld.class
-a--l	10/05/2023 14:44	129	HelloWorld.java

Il ne vous reste plus qu'à exécuter le programme en tapant '**java**' suivi du nom de la classe sans avoir besoin de préciser une extension.

```
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> java HelloWorld
Hello, World!
PS C:\Users\khalf\OneDrive\Bureau\CoursJava>
```

Bravo votre programme se lance sans problème d'exécution et affiche bien dans la console le message "Hello, World". Mais si nous changeons ce message dans le code et que nous ré-exécutons notre programme, nous ne verrons aucun changement.



```
1 public class HelloWorld {
2     public static void main(String[] args) {
3         System.out.println("Bonjour tous le monde !");
4     }
5 }
```

```
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> java HelloWorld
Hello, World!
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> java HelloWorld
Hello, World!
PS C:\Users\khalf\OneDrive\Bureau\CoursJava>
```

Ceci est dû au fait que nous devons recompiler le programme à chaque changement afin que la dernière version soit prise en compte.

```
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> java HelloWorld
Hello, World!
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> javac .\HelloWorld.java
PS C:\Users\khalf\OneDrive\Bureau\CoursJava> java HelloWorld
Bonjour tous le monde !
PS C:\Users\khalf\OneDrive\Bureau\CoursJava>
```

Voilà ! Vous avez maintenant un exemple simple qui illustre la syntaxe de base de Java, en soulignant l'importance de nommer le fichier avec le même nom que la classe qu'il contient.

Bonus syntaxe :



Les bases du langage: Les types de données.

En Java, les types de données définissent la nature des valeurs manipulées par un programme. Les types de données peuvent être classés en deux catégories principales : les types primitifs et les types de référence.

Types primitifs :

Les types primitifs sont les types de données de base en Java. Ils représentent des valeurs simples et ne sont pas des objets. Voici les types primitifs les plus couramment utilisés en Java :

Type Primitif	Type Objet Associé	Description	Place en mémoire	Valeurs min-max	Exemple
boolean	Boolean	Représente une valeur vraie ou fausse.	1 octet	true(1) false(0)	boolean estVrai = true;
byte	Byte	Représente un entier de 8 bits.	1 octet	-128 à 127	byte monByte = 100;
short	Short	Représente un entier de 16 bits.	2 octets	-32768 à 32767	short monshort = 1000;
int	Integer	Représente un entier de 32 bits.	4 octets	-2 147 483 648 à 2 147 483 647	int monInt = 100000;
long	Long	Représente un entier de 64 bits	8 octets	-2^{63} à $+2^{63}-1$	long monLong = 1000000000L;
float	Float	Représente un nombre à virgule flottante de 32 bits.	4 octets	1.4×10^{-45} à 3.4×10^{38}	float monFloat = 3.14f;
double	Double	Représente un nombre à virgule flottante de 64 bits.	8 octets	4.9×10^{-324} à 1.7×10^{308}	double monDouble = 3.14159;
char	Character	Représente un caractère Unicode de 16 bits.	2 octets	Unicode (65536 caractères disponibles)	char monChar = "A";

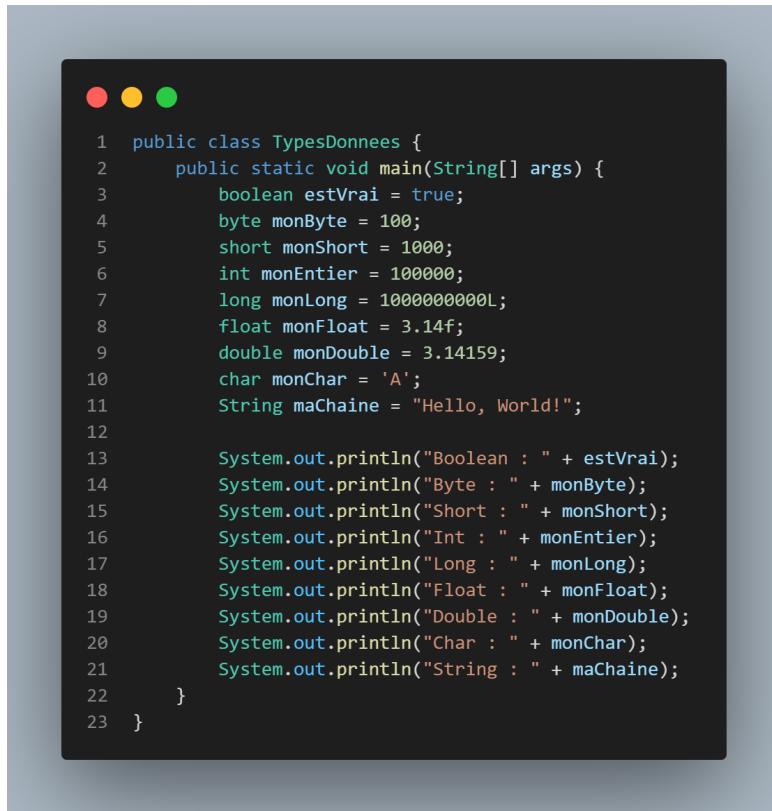
Il est important de noter que les noms des types primitifs commencent tous par une lettre minuscule.

Types de référence :

Les types de référence sont des types qui représentent des objets en Java. Ils sont créés à partir de classes ou d'interfaces définies par l'utilisateur ou fournies par le

langage. Les types de référence peuvent stocker des valeurs complexes et offrent des fonctionnalités avancées. Certains exemples de types de référence courants incluent String, Arrays et les objets personnalisés créés par les programmeurs.

Voici un exemple d'utilisation de différents types de données en Java :

A screenshot of a terminal window on a Mac OS X system. The window has red, yellow, and green close buttons at the top. The code inside the terminal is as follows:

```
1 public class TypesDonnees {
2     public static void main(String[] args) {
3         boolean estVrai = true;
4         byte monByte = 100;
5         short monShort = 1000;
6         int monEntier = 100000;
7         long monLong = 1000000000L;
8         float monFloat = 3.14f;
9         double monDouble = 3.14159;
10        char monChar = 'A';
11        String maChaine = "Hello, World!";
12
13        System.out.println("Boolean : " + estVrai);
14        System.out.println("Byte : " + monByte);
15        System.out.println("Short : " + monShort);
16        System.out.println("Int : " + monEntier);
17        System.out.println("Long : " + monLong);
18        System.out.println("Float : " + monFloat);
19        System.out.println("Double : " + monDouble);
20        System.out.println("Char : " + monChar);
21        System.out.println("String : " + maChaine);
22    }
23 }
```

The terminal output shows the values of each variable printed to the screen.

Dans cet exemple, nous avons déclaré différentes variables avec différents types de données. Nous avons ensuite utilisé la méthode '**println**' pour afficher les valeurs de ces variables à la console. Chaque ligne utilise la concaténation de chaînes (utilisation de l'opérateur '+' pour combiner des chaînes) pour afficher le type de données correspondant suivi de la valeur de la variable.

Il est essentiel de choisir le type de données approprié en fonction de la nature des valeurs que vous souhaitez stocker et manipuler dans votre programme. Les types de données primitifs offrent une efficacité de stockage et de calcul accrue, tandis que les types de référence fournissent des fonctionnalités plus avancées et permettent de travailler avec des structures de données complexes.

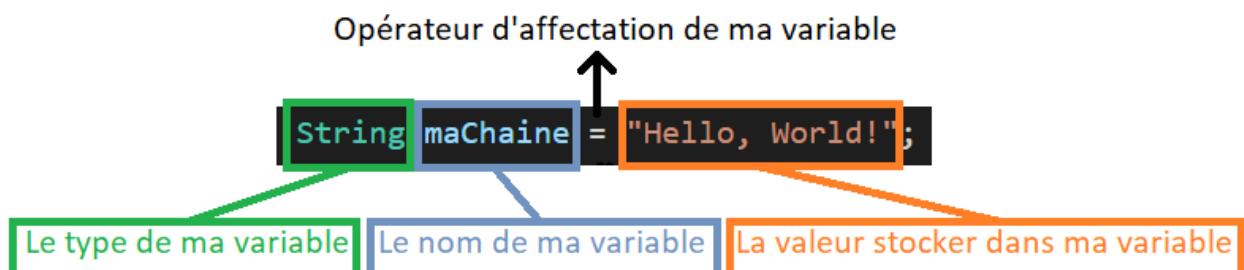
En résumé, les types de données en Java sont utilisés pour définir le comportement et la représentation des valeurs dans un programme. Les types primitifs offrent des valeurs simples, tandis que les types de référence sont utilisés pour manipuler des objets plus complexes. Comprendre et choisir les bons types de données est essentiel pour développer des programmes efficaces et fiables en Java.

Bonus syntaxe :



Les bases du langage: Les constantes et les variables.

Dans le dernier exemple que nous avons vu ensemble nous avons dit que nous avions déclaré différentes variables avec différents types de données. Mais qu'est-ce que cela signifie réellement ? Voici un petit schéma qui explique la déclaration d'une variable en Java :



Comme nous pouvons le voir la déclaration d'une variable suit cet ordre : le type, le nom puis la valeur. En ce qui concerne le type nous avons abordé le sujet dans le chapitre précédent, mais pour les noms des variables il existe quelques règles à suivre.

Voici les règles de nommage des variables en Java :

1. Les noms de variable doivent commencer par une lettre (a-z ou A-Z) ou un underscore: en français un caractère de soulignement (_). Ils ne peuvent pas commencer par un chiffre.
2. Les noms de variable peuvent contenir des lettres, des chiffres et des caractères de soulignement.
3. Les noms de variable sont sensibles à la casse, ce qui signifie que les majuscules et les minuscules sont distinctes. Par exemple, maVariable et mavariable sont considérées comme deux variables différentes.
4. Il est recommandé d'utiliser des noms de variables significatifs qui décrivent leur but ou leur contenu.
5. Les noms de variable ne doivent pas être des mots-clés réservés du langage Java, tels que int, if, for, etc.
6. Les noms de variable ne doivent pas contenir d'espaces ou de caractères spéciaux tels que @, !, #, \$, etc.
7. Il est courant d'utiliser la notation camelCase pour les noms de variable. Cela signifie que les mots sont concaténés et chaque mot commence par une majuscule, à l'exception du premier mot qui commence par une minuscule. Par exemple, maVariable, nombreEtudiants, etc.
8. Les noms de variable doivent être descriptifs et compréhensibles, afin d'améliorer la lisibilité du code.

Il est important de suivre ces règles de nommage pour écrire un code clair, compréhensible et maintenable en Java.

Une variable est un symbole qui représente une valeur stockée en mémoire dans un programme informatique. Elle peut être modifiée et peut prendre différentes valeurs pendant l'exécution du programme. Une variable est déclarée en spécifiant son type et son nom, et peut être assignée à une valeur.

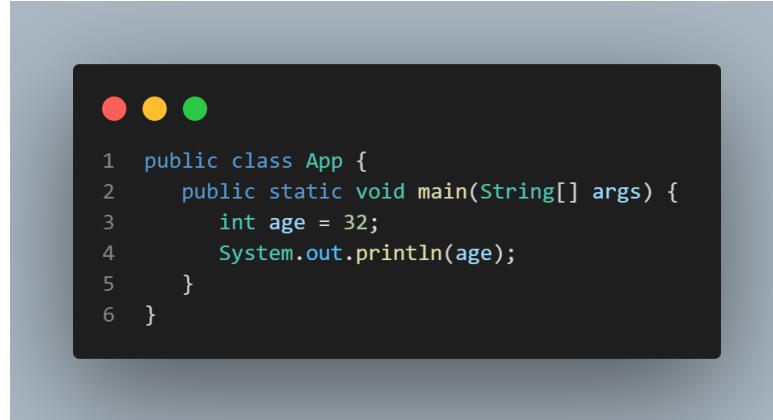
En revanche, une constante est un symbole qui représente une valeur fixe et immuable. Une fois définie, sa valeur ne peut pas être modifiée pendant l'exécution du programme. Les constantes sont souvent utilisées pour représenter des valeurs qui sont connues et ne doivent pas être modifiées, comme des valeurs mathématiques constantes (par exemple, π) ou des valeurs de configuration.

Il est important de bien comprendre la différence entre les variables et les constantes pour plusieurs raisons :

1. Clarté du code : En distinguant clairement les variables des constantes, il devient plus facile de comprendre l'intention du code. Les constantes fournissent des informations sur les valeurs fixes et immuables utilisées dans le programme, tandis que les variables indiquent les valeurs qui peuvent varier.
2. Maintenance du code : En utilisant des constantes pour des valeurs qui ne doivent pas être modifiées, le code devient plus maintenable. Si une valeur constante doit être modifiée à un moment donné, il suffit de changer sa définition à un seul endroit, plutôt que de rechercher toutes les occurrences de cette valeur dans le code.
3. Sécurité et stabilité : En utilisant des constantes pour des valeurs critiques et fixes, on s'assure qu'elles ne seront pas accidentellement modifiées, ce qui pourrait entraîner des erreurs ou des comportements indésirables du programme.

En comprenant la distinction entre les variables et les constantes, les développeurs peuvent écrire un code plus clair, plus maintenable et plus sûr. L'utilisation appropriée des variables et des constantes contribue à une meilleure compréhension du code par les autres développeurs et facilite les modifications ultérieures du programme.

Imaginons une variable `age` :



```
● ● ●

1 public class App {
2     public static void main(String[] args) {
3         int age = 32;
4         System.out.println(age);
5     }
6 }
```

Nous avons stocké dans la variable âge 32, ainsi lors de l'exécution de ce code la console affichera 32. Réaffectionons une autre valeur dans cette variable :



```
● ● ●

1 public class App {
2     public static void main(String[] args) {
3         int age = 32;
4         System.out.println(age);
5         age = 34;
6         System.out.println(age);
7     }
8 }
```

Dans ce code, nous avons une classe appelée "App". À l'intérieur de cette classe, nous avons une méthode principale appelée "main" qui est l'entrée principale du programme.

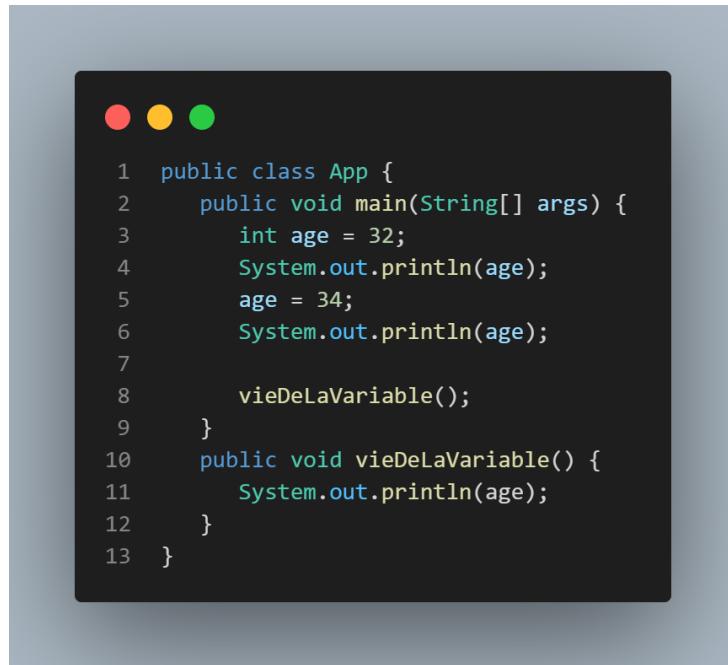
À l'intérieur de la méthode "main", nous déclarons une variable appelée "age" de type entier (int) et lui attribuons une valeur initiale de 32 en utilisant l'instruction "**int age = 32;**". Cela signifie que nous créons une variable appelée "age" qui peut contenir des valeurs entières et nous lui assignons la valeur initiale de 32.

Ensuite, nous utilisons l'instruction "**System.out.println(age);**" pour afficher la valeur de la variable "age" sur la console. Cela affichera "32" car c'est la valeur actuelle de la variable "age" à ce stade.

Ensuite, nous mettons à jour la valeur de la variable "age" en utilisant l'instruction "**age = 34;**". Cela signifie que nous changeons la valeur de la variable "age" de 32 à 34.

Enfin, nous utilisons à nouveau l'instruction "**System.out.println(age);**" pour afficher la nouvelle valeur de la variable "age" sur la console. Cette fois-ci, cela affichera "34" car nous avons modifié la valeur de la variable "age" précédemment.

En résumé, ce code déclare une variable "age", lui attribue une valeur initiale de 32, l'affiche sur la console, met à jour sa valeur à 34, puis l'affiche à nouveau. **Cela démontre comment une variable peut être modifiée et utilisée pour stocker différentes valeurs pendant l'exécution d'un programme.**



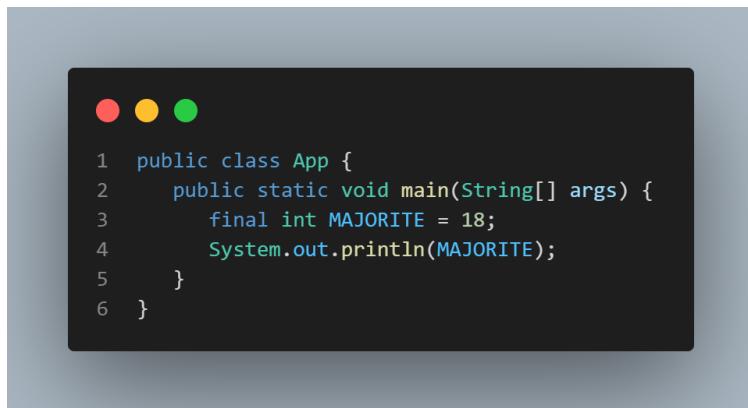
```
1 public class App {  
2     public void main(String[] args) {  
3         int age = 32;  
4         System.out.println(age);  
5         age = 34;  
6         System.out.println(age);  
7     }  
8     public void vieDeLaVariable() {  
9         System.out.println(age);  
10    }  
11 }
```

Nous appelons la méthode "vieDeLaVariable" depuis la méthode "main". Cette méthode "vieDeLaVariable" est une autre méthode de la classe "App". Cependant, lorsque nous essayons d'afficher la valeur de la variable "age" à l'intérieur de cette méthode, nous obtenons une erreur de compilation car la variable "age" n'est pas accessible à cet endroit. La portée de la variable "age" est limitée à la méthode "main" où elle a été déclarée (dans son contexte).

En résumé, la variable "age" a une durée de vie limitée à la méthode "main". Elle existe et peut être utilisée uniquement à l'intérieur de cette méthode. Lorsque la méthode "main"

se termine, la variable "age" cesse d'exister. La portée de la variable "age" est également limitée à la méthode "main", ce qui signifie qu'elle n'est pas accessible depuis d'autres méthodes de la classe "App" comme "vieDeLaVariable". C'est pourquoi nous obtenons une erreur lors de la tentative d'accès à la variable "age" à l'intérieur de la méthode "vieDeLaVariable".

Maintenant pour transformer une variable en constante on utilisera le mots clé "final" et le nom de celle-ci sera écrit en majuscule afin de facilement l'identifier en tant que constante.



```

1 public class App {
2     public static void main(String[] args) {
3         final int MAJORITE = 18;
4         System.out.println(MAJORITE);
5     }
6 }
```

En revanche, si on essaie de lui attribuer une autre valeur, nous obtiendrons une erreur lors de la compilation.



```

1 public class App {
2     public static void main(String[] args) {
3         final int MAJORITE = 18;
4         System.out.println(MAJORITE);
5         MAJORITE = 21;
6         System.out.println(MAJORITE);
7     }
8 }
```

Dans le code, nous déclarons la constante "MAJORITE" et lui attribuons une valeur de 18 en utilisant l'instruction "**final int MAJORITE = 18;**". Ensuite, nous utilisons

l'instruction "**System.out.println(MAJORITE);**" pour afficher la valeur de la constante sur la console, ce qui affiche "18".

Cependant, lorsque nous essayons de modifier la valeur de la constante "MAJORITE" en utilisant l'instruction "**MAJORITE = 21;**", une erreur de compilation se produit. Cela est dû au fait que les constantes en Java ne peuvent pas être réassignées avec une nouvelle valeur une fois qu'elles ont été définies. Elles conservent leur valeur initiale tout au long du programme.

Les bases du langage: Les opérateurs et les boucles.

Les opérateurs et les boucles sont des éléments essentiels du langage Java pour effectuer des opérations et contrôler le flux d'exécution d'un programme. Dans ce chapitre, nous explorerons les différents types d'opérateurs disponibles en Java ainsi que les boucles couramment utilisées pour répéter des instructions.

Les opérateurs :

1. Les opérateurs sont des symboles spéciaux utilisés pour effectuer des opérations sur les données. Java offre différents types d'opérateurs, notamment les opérateurs arithmétiques, les opérateurs de comparaison, les opérateurs logiques et les opérateurs d'assignation. En d'autres termes, les opérateurs en Java sont utilisés pour effectuer des opérations sur les données, telles que les calculs mathématiques, les comparaisons et les opérations logiques. Voici quelques types d'opérateurs couramment utilisés :

- Les opérateurs arithmétiques : Ils permettent d'effectuer des opérations mathématiques comme l'addition (+), la soustraction (-), la multiplication (*), la division (/) et le modulo (%).

Exemple :

```
● ● ●

1  public class App {
2      public static void main(String[] args) {
3          int a = 10;
4          int b = 5;
5
6          int addition = a + b;           // Addition
7          System.out.println(addition); // retourne 15
8
9          int soustraction = a - b;     // Soustraction
10         System.out.println(soustraction); // retourne 5
11
12         int multiplication = a * b;   // Multiplication
13         System.out.println(multiplication); // retourne 50
14
15         int division = a / b;        // Division
16         System.out.println(division); // retourne 2
17
18         int modulo = a % b;         // Modulo (reste de la division)
19         System.out.println(modulo); // retourne 0
20     }
21 }
```

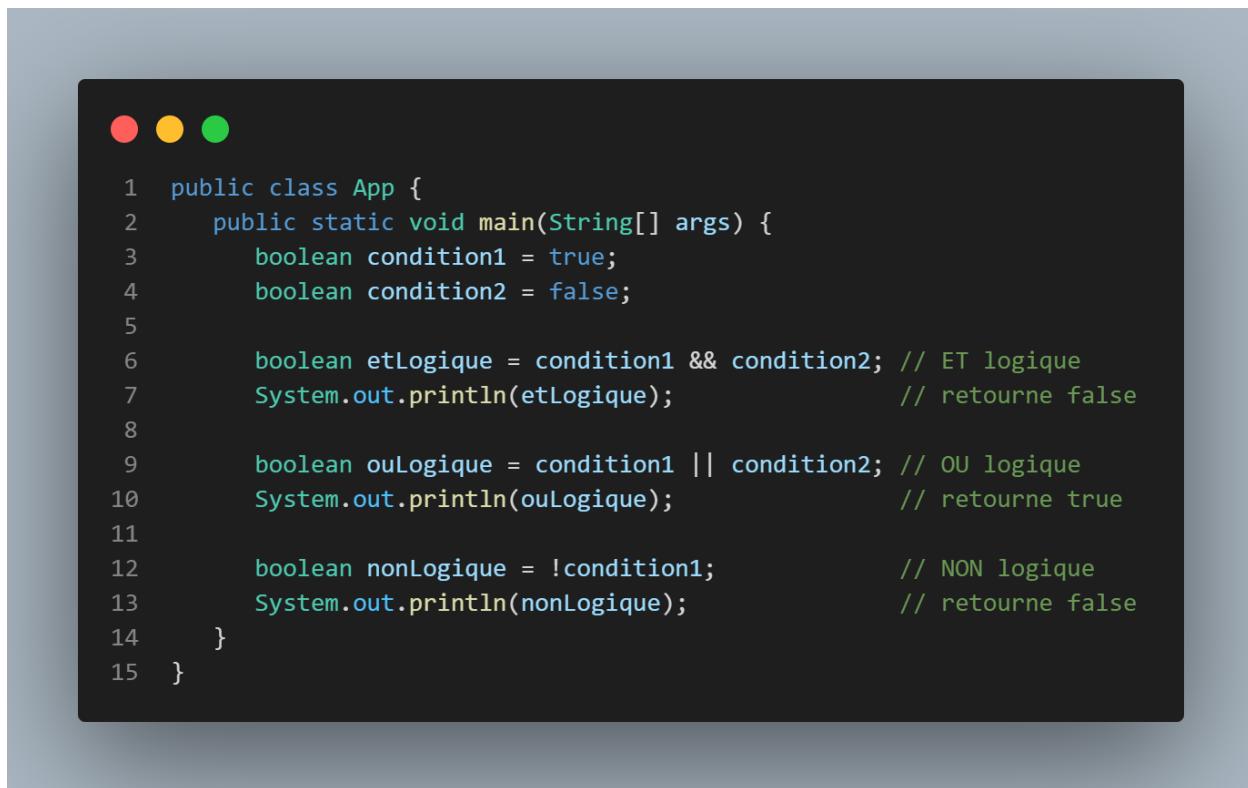
- Les opérateurs de comparaison : Ils sont utilisés pour comparer les valeurs et renvoient un résultat booléen (vrai ou faux). Les opérateurs de comparaison comprennent l'égalité (==), la différence (!=), l'infériorité (<), la supériorité (>), l'infériorité ou l'égalité (<=) et la supériorité ou égalité (>=).

Exemple :

```
● ● ●  
1  public class App {  
2      public static void main(String[] args) {  
3          int x = 5;  
4          int y = 10;  
5  
6          boolean estPlusGrand = x > y;           // Supérieur à  
7          System.out.println(estPlusGrand);         // retourne false  
8  
9          boolean estPlusPetit = x < y;            // Inférieur à  
10         System.out.println(estPlusPetit);          // retourne true  
11  
12         boolean estEgal = x == y;                 // Égal à  
13         System.out.println(estEgal);                // retourne false  
14  
15         boolean estDifferent = x != y;            // Différent de  
16         System.out.println(estDifferent);           // retourne true  
17  
18         boolean estPlusGrandOuEgal = x >= y;       // Supérieur ou égal à  
19         System.out.println(estPlusGrandOuEgal); // retourne false  
20  
21         boolean estPlusPetitOuEgal = x <= y;        // Inférieur ou égal à  
22         System.out.println(estPlusPetitOuEgal); // retourne true  
23     }  
24 }
```

- Les opérateurs logiques : Ils sont utilisés pour effectuer des opérations logiques sur des valeurs booléennes. Les principaux opérateurs logiques sont le "et" logique (`&&`), le "ou" logique (`||`) et la négation logique (`!`).

Exemple :



The screenshot shows a Java code editor with a dark theme. At the top left, there are three circular icons: red, yellow, and green. The code itself is as follows:

```
1 public class App {  
2     public static void main(String[] args) {  
3         boolean condition1 = true;  
4         boolean condition2 = false;  
5  
6         boolean etLogique = condition1 && condition2; // ET logique  
7         System.out.println(etLogique); // retourne false  
8  
9         boolean ouLogique = condition1 || condition2; // OU logique  
10        System.out.println(ouLogique); // retourne true  
11  
12        boolean nonLogique = !condition1; // NON logique  
13        System.out.println(nonLogique); // retourne false  
14    }  
15 }
```

- Les opérateurs d'assignation : Ils permettent d'attribuer une valeur à une variable. L'opérateur d'assignation de base est le signe égal (`=`), mais il existe également des opérateurs d'assignation combinée tels que l'addition et l'assignation (`+=`) ou la soustraction et l'assignation (`-=`).

Exemple :

```

● ● ●

1 public class App {
2     public static void main(String[] args) {
3         int nombre = 10;
4         nombre += 5; // Équivaut à : nombre = nombre + 5;
5         nombre -= 3; // Équivaut à : nombre = nombre - 3;
6         nombre *= 2; // Équivaut à : nombre = nombre * 2;
7         nombre /= 4; // Équivaut à : nombre = nombre / 4;
8         nombre %= 3; // Équivaut à : nombre = nombre % 3;
9     }
10 }
```

Les boucles :

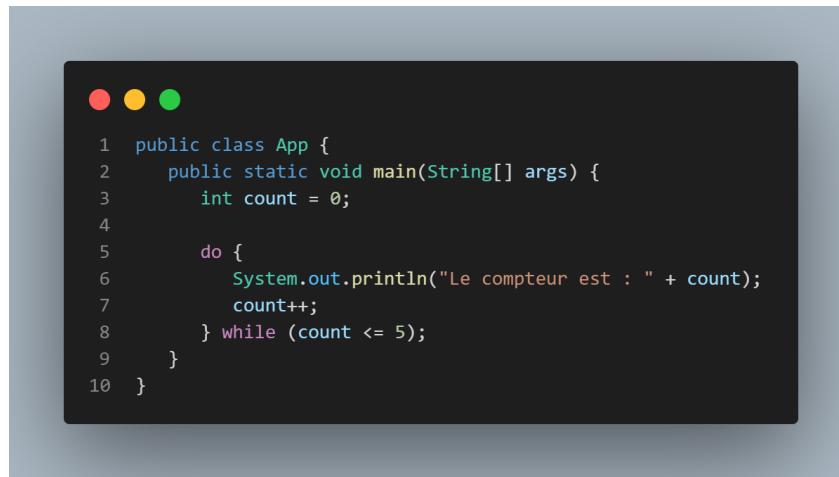
1. Les boucles sont utilisées pour répéter l'exécution d'un bloc de code tant qu'une condition est vraie. En Java, nous avons principalement trois types de boucles :
 - La boucle "while" : Elle répète l'exécution d'un bloc de code tant qu'une condition donnée est vraie. La condition est vérifiée avant chaque itération.

```

● ● ●

1 public class App {
2     public static void main(String[] args) {
3         int i = 0;
4         while (i < 5) {
5             System.out.println("Iteration " + i);
6             i++;
7         }
8     }
9 }
```

- La boucle "do-while" : Elle est similaire à la boucle "while", mais la condition est vérifiée après l'exécution du bloc de code. Cela garantit que le bloc de code est exécuté au moins une fois.



```
1 public class App {  
2     public static void main(String[] args) {  
3         int count = 0;  
4  
5         do {  
6             System.out.println("Le compteur est : " + count);  
7             count++;  
8         } while (count <= 5);  
9     }  
10 }
```

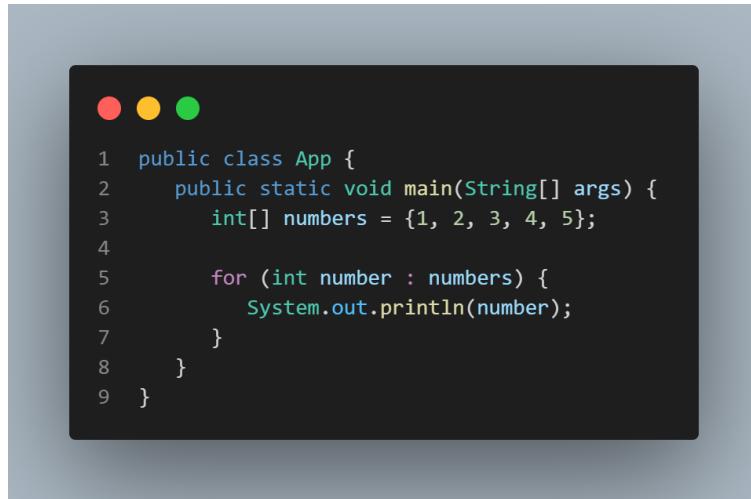
- La boucle "for" : Elle permet de répéter l'exécution d'un bloc de code un certain nombre de fois. Elle utilise une variable de contrôle, une condition de continuation et une expression d'itération pour définir le nombre d'itérations.



```
1 public class App {  
2     public static void main(String[] args) {  
3         for (int i = 0; i < 10; i++) {  
4             System.out.println(i);  
5         }  
6     }  
7 }
```

- La boucle foreach, également appelée boucle améliorée for, est une structure de contrôle utilisée pour itérer à travers les éléments d'une collection ou d'un tableau en Java. Elle simplifie le processus de parcours

des éléments d'une collection, en évitant la gestion manuelle des indices ou des compteurs.



```
● ● ●
1 public class App {
2     public static void main(String[] args) {
3         int[] numbers = {1, 2, 3, 4, 5};
4
5         for (int number : numbers) {
6             System.out.println(number);
7         }
8     }
9 }
```

Les bases du langage: Les conditions.

Les conditions jouent un rôle essentiel dans la programmation, car elles permettent d'exécuter différentes instructions en fonction de certaines conditions ou situations. En Java, les conditions sont mises en œuvre à l'aide des instructions if, else if et else.

La structure de base d'une condition est la suivante :



```
● ● ●
1 public class App {
2     public static void main(String[] args) {
3         if (condition) {
4             // Bloc d'instructions à exécuter si la condition est vraie
5         } else if (autreCondition) {
6             // Bloc d'instructions à exécuter si la condition précédente est fausse et cette condition est vraie
7         } else {
8             // Bloc d'instructions à exécuter si toutes les conditions précédentes sont fausses
9         }
10    }
11 }
```

La condition est une expression qui évalue à une valeur booléenne (vrai ou faux). Si la condition est vraie, le bloc d'instructions associé est exécuté. Si la condition est fausse,

le bloc d'instructions associé est ignoré, et l'exécution se poursuit avec la prochaine partie du code.

Voici quelques exemples d'utilisation des conditions en Java :

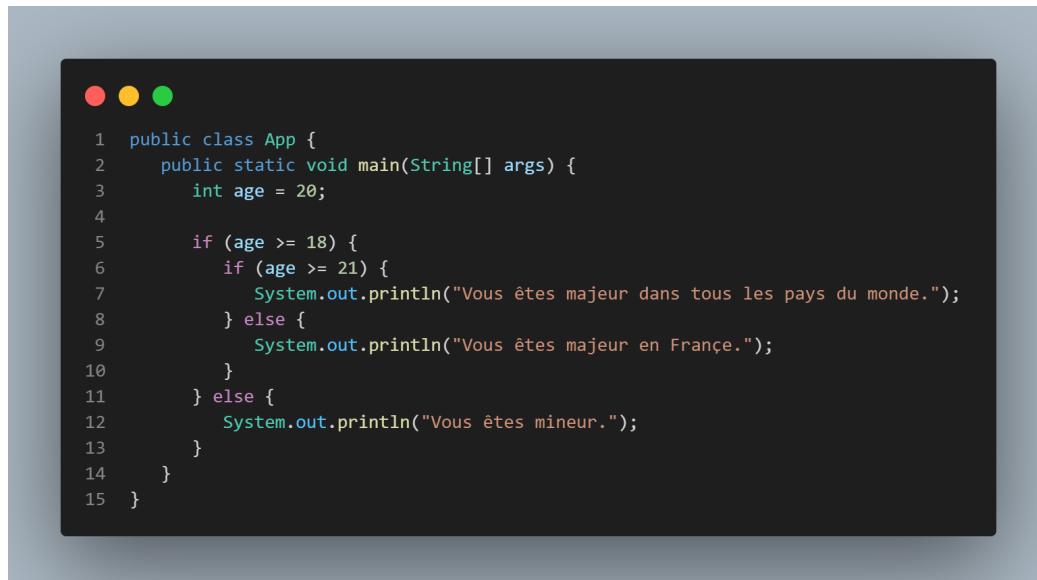
Exemple 1 : Vérification de l'âge d'une personne



```
1 public class App {
2     public static void main(String[] args) {
3         int age = 20;
4
5         if (age >= 18) {
6             System.out.println("Vous êtes majeur.");
7         } else {
8             System.out.println("Vous êtes mineur.");
9         }
10    }
11 }
```

Dans cet exemple, nous vérifions si l'âge est supérieur ou égal à 18. Si c'est le cas, le message "Vous êtes majeur." est affiché. Sinon, le message "Vous êtes mineur." est affiché. Mais nous pouvons aller plus loin en imbriquant les conditions.

Exemple 2 : Vérification de l'âge d'une personne



```
1 public class App {
2     public static void main(String[] args) {
3         int age = 20;
4
5         if (age >= 18) {
6             if (age >= 21) {
7                 System.out.println("Vous êtes majeur dans tous les pays du monde.");
8             } else {
9                 System.out.println("Vous êtes majeur en France.");
10            }
11        } else {
12            System.out.println("Vous êtes mineur.");
13        }
14    }
15 }
```

Dans cet exemple de code, nous utilisons des conditions imbriquées pour vérifier l'âge d'une personne et afficher un message en fonction de cette valeur.

Tout d'abord, nous déclarons et initialisons une variable "age" avec la valeur 20.

Ensuite, nous utilisons une instruction "if" pour vérifier si l'âge est supérieur ou égal à 18. Si cette condition est vraie, nous entrons dans le premier bloc d'instructions.

À l'intérieur de ce premier bloc, nous utilisons une autre instruction "if" pour vérifier si l'âge est supérieur ou égal à 21. Si cette condition est vraie, cela signifie que la personne est majeure dans tous les pays du monde, et nous affichons le message correspondant.

Si la condition du deuxième "if" n'est pas satisfaite, cela signifie que la personne est majeure en France, mais pas nécessairement dans tous les pays du monde. Nous affichons donc le message correspondant.

Si la condition du premier "if" n'est pas satisfaite, cela signifie que l'âge est inférieur à 18, et nous entrons dans le bloc "else". Dans ce cas, nous affichons le message indiquant que la personne est mineure.

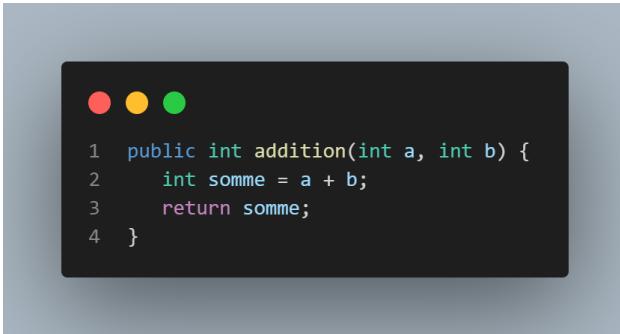
Les conditions imbriquées permettent de tester des conditions plus spécifiques à l'intérieur de conditions plus générales. Cela nous permet de prendre des décisions en fonction de multiples critères et de traiter différents cas de manière appropriée.

Les bases du langage: Les fonctions.

Les fonctions, également appelées méthodes en Java, sont des blocs de code réutilisables qui effectuent une tâche spécifique. Elles permettent de découper un programme en petites parties modulaires, ce qui facilite la compréhension, la maintenance et la réutilisation du code.

En Java, une fonction est déclarée à l'intérieur d'une classe et peut être appelée depuis d'autres parties du programme. Une fonction peut prendre des paramètres en entrée, effectuer des opérations sur ces paramètres, et renvoyer un résultat en sortie.

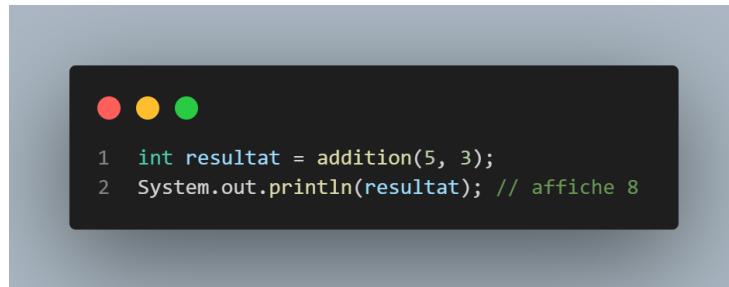
Voici un exemple de déclaration d'une fonction en Java :



```
● ● ●
1 public int addition(int a, int b) {
2     int somme = a + b;
3     return somme;
4 }
```

Dans cet exemple, nous déclarons une fonction appelée "addition" qui prend deux paramètres de type entier (a et b). À l'intérieur de la fonction, nous effectuons l'opération d'addition des deux paramètres et stockons le résultat dans une variable locale "somme". Enfin, nous renvoyons la valeur de la somme à l'aide du mot-clé "return".

Pour appeler cette fonction et utiliser son résultat, nous pouvons faire ce qui suit :



```
● ● ●
1 int resultat = addition(5, 3);
2 System.out.println(resultat); // affiche 8
```

Dans cet exemple, nous appelons la fonction "addition" avec les valeurs 5 et 3 comme arguments. La fonction effectue l'addition et renvoie le résultat 8, que nous stockons dans la variable "resultat". Nous affichons ensuite la valeur de "resultat" à l'aide de la fonction "println".

Les fonctions peuvent également être déclarées sans paramètres ou sans valeur de retour. Voici un exemple :



```
● ● ●
1 public void afficherMessage() {
2     System.out.println("Bonjour !");
3 }
```

Cette fonction nommée "afficherMessage" ne prend aucun paramètre et ne renvoie aucune valeur. Elle se contente d'afficher le message "Bonjour !" à l'écran. Pour l'appeler, nous pouvons simplement écrire :



En résumé, les fonctions en Java permettent de regrouper des morceaux de code qui effectuent des tâches spécifiques. Elles améliorent la modularité, la réutilisabilité et la lisibilité du code. Les fonctions peuvent prendre des paramètres en entrée, effectuer des opérations et renvoyer des valeurs en sortie, ou simplement effectuer des actions sans retourner de valeurs.



Exercice 1 :

Écrivez un programme Java qui affiche tous les chiffres pairs jusqu'à 10 sur la console.

Voici les étapes pour résoudre cet exercice :

1. Déclarez une variable entière nommée "nombre" et initialisez-la à 0.
2. Utilisez une boucle while pour itérer tant que "nombre" est inférieur ou égal à 10.
3. À l'intérieur de la boucle, utilisez une structure de contrôle if pour vérifier si "nombre" est pair.
4. Si "nombre" est pair, affichez sa valeur sur la console à l'aide de "System.out.println()".
5. Incrémentez la valeur de "nombre" de 1 à chaque itération pour passer au prochain nombre.

Exercice 2 :

Écrivez un programme Java qui demande à l'utilisateur d'entrer deux valeurs entières : une valeur de début et une valeur de fin. Ensuite, le programme vérifie si la valeur de début est inférieure à la valeur de fin. Si c'est le cas, le programme affiche tous les nombres pairs compris entre la valeur de début et la valeur de fin. Sinon, le programme affiche un message d'erreur indiquant que la valeur de début doit être plus petite que la valeur de fin.

Voici les étapes pour résoudre cet exercice :

1. Demandez à l'utilisateur d'entrer la valeur de début et stockez-la dans une variable entière nommée "debut".
2. Demandez à l'utilisateur d'entrer la valeur de fin et stockez-la dans une variable entière nommée "fin".
3. Utilisez une structure de contrôle if pour vérifier si la valeur de début est inférieure à la valeur de fin.
4. Si la condition est vérifiée, utilisez une boucle while pour itérer de la valeur de début à la valeur de fin.
5. À chaque itération, vérifiez si le nombre est pair en utilisant l'opérateur modulo (%) pour vérifier si le reste de la division par 2 est égal à 0.
6. Si le nombre est pair, affichez sa valeur sur la console à l'aide de "System.out.println()".

Les objets et les classes : introduction aux objets.

Les objets et les classes sont des concepts fondamentaux en programmation orientée objet. Ils permettent de modéliser des entités du monde réel ou des concepts abstraits sous forme de structures de données.

En Java, un objet est une instance d'une classe. Une classe définit les propriétés (attributs) et les comportements (méthodes) que les objets de cette classe peuvent avoir.

Pour illustrer ces concepts, prenons l'exemple d'une classe "Voiture" :

```
● ● ●
1 public class Voiture {
2     // Attributs
3     private String marque;
4     private String couleur;
5     private int annee;
6
7     // Méthode
8     public void demarrer() {
9         System.out.println("La voiture démarre !");
10    }
11 }
```

Dans cet exemple, nous avons déclaré une classe "Voiture" avec trois attributs (marque, couleur, annee) et une méthode (demarrer). Les attributs représentent les caractéristiques d'une voiture, tandis que la méthode représente un comportement associé à la voiture.

Pour créer un objet de la classe "Voiture", nous utilisons le mot-clé "new" suivi du nom de la classe et des parenthèses :

```
1 Voiture maVoiture = new Voiture();
```

Maintenant, nous pouvons accéder aux attributs et aux méthodes de l'objet "maVoiture". Par exemple, pour définir la marque de la voiture, nous pouvons écrire :

```
1 maVoiture.marque = "Toyota";
```

Pour appeler la méthode "demarrer" de l'objet "maVoiture", nous écrivons simplement :

```
1 maVoiture.demarrer();
```

Les objets nous permettent de créer des instances uniques avec leurs propres valeurs d'attributs. Par exemple, nous pouvons créer plusieurs objets de la classe "Voiture" avec différentes marques, couleurs et années :



Chaque objet possède ses propres valeurs d'attributs et peut appeler les méthodes définies dans la classe "Voiture".

En résumé, les objets et les classes sont des concepts clés de la programmation orientée objet. Les classes définissent la structure et les comportements des objets, tandis que les objets sont des instances spécifiques de ces classes. Les objets nous permettent de modéliser des entités du monde réel ou des concepts abstraits, et de manipuler leurs attributs et leurs comportements.

Les objets et les classes : Les classes et les constructeurs.

Comme nous venons de le voir dans la programmation orientée objet, les classes et les constructeurs sont des éléments fondamentaux pour la création d'objets. Les classes servent de modèle ou de plan pour créer des objets, tandis que les constructeurs permettent d'initialiser les attributs des objets lors de leur création.

Les classes :

Une classe est une structure qui définit les attributs (variables) et les méthodes (fonctions) communs à un groupe d'objets. Elle représente un concept, un objet ou une entité du monde réel. Les classes définissent les caractéristiques et les comportements que les objets de cette classe auront.

Reprendons l' exemple de la classe "Voiture" :

```
1 public class Voiture {
2     // Attributs
3     private String marque;
4     private String modèle;
5     private int année;
6
7     // Constructeur
8     public Voiture(String marque, String modèle, int année) {
9         this.marque = marque;
10        this.modèle = modèle;
11        this.année = année;
12    }
13
14    // Méthodes
15    public void démarrer() {
16        System.out.println("La voiture démarre.");
17    }
18
19    public void accélérer() {
20        System.out.println("La voiture accélère.");
21    }
22 }
```

Dans cet exemple, nous avons défini une classe "Voiture" avec trois attributs (marque, modèle et année) et deux méthodes (démarrer et accélérer).

Les constructeurs :

Un constructeur est une méthode spéciale qui est appelée lors de la création d'un nouvel objet. Il est responsable de l'initialisation des attributs de l'objet avec les valeurs passées en paramètres. Les constructeurs portent le même nom que la classe et peuvent avoir des paramètres ou être sans paramètre.

Dans notre exemple de classe "Voiture", nous avons défini un constructeur qui prend trois paramètres : marque, modèle et année. Le constructeur initialise les attributs de l'objet avec ces valeurs.

Pour créer des objets de la classe "Voiture" et les utiliser, nous pouvons faire ce qui suit :

```
● ○ ●  
1 Voiture voiture1 = new Voiture("Toyota", "Corolla", 2020);  
2 voiture1.démarrer(); // Affiche : La voiture démarre.  
3  
4 Voiture voiture2 = new Voiture("Honda", "Civic", 2018);  
5 voiture2.accélérer(); // Affiche : La voiture accélère.
```

Dans cet exemple, nous avons créé deux objets de la classe "Voiture" en utilisant le constructeur. Chaque objet a ses propres valeurs d'attributs. Nous avons ensuite appelé les méthodes de ces objets pour effectuer des actions spécifiques à la voiture.

Les constructeurs sont essentiels pour initialiser les objets avec des valeurs cohérentes et appropriées. Ils permettent de définir l'état initial des objets et de s'assurer qu'ils sont prêts à être utilisés.

En résumé, les classes servent de modèle pour créer des objets et définissent les attributs et les méthodes communs à ces objets. Les constructeurs sont des méthodes spéciales qui sont responsables de l'initialisation des attributs des objets lors de leur création. Ils permettent de créer des objets cohérents et prêts à être utilisés.

Les objets et les classes : Les champs et les encapsulations.

Dans la programmation orientée objet, les champs et l'encapsulation jouent un rôle essentiel dans la définition des attributs et dans le contrôle de l'accès à ces attributs au sein des classes. Ils permettent d'assurer l'encapsulation des données et de maintenir l'intégrité des objets.

Les champs :

Les champs, également appelés variables d'instance, sont des variables déclarées à l'intérieur d'une classe et qui représentent les caractéristiques ou les données

spécifiques à chaque objet créé à partir de cette classe. Chaque objet a sa propre copie des champs de la classe.

Voici un exemple de classe "Personne" avec des champs :



```
 1 public class Personne {  
 2     // Champs (variables d'instance)  
 3     private String nom;  
 4     private int age;  
 5     private String adresse;  
 6  
 7     // Méthodes (getters et setters)  
 8     // ...  
 9 }
```

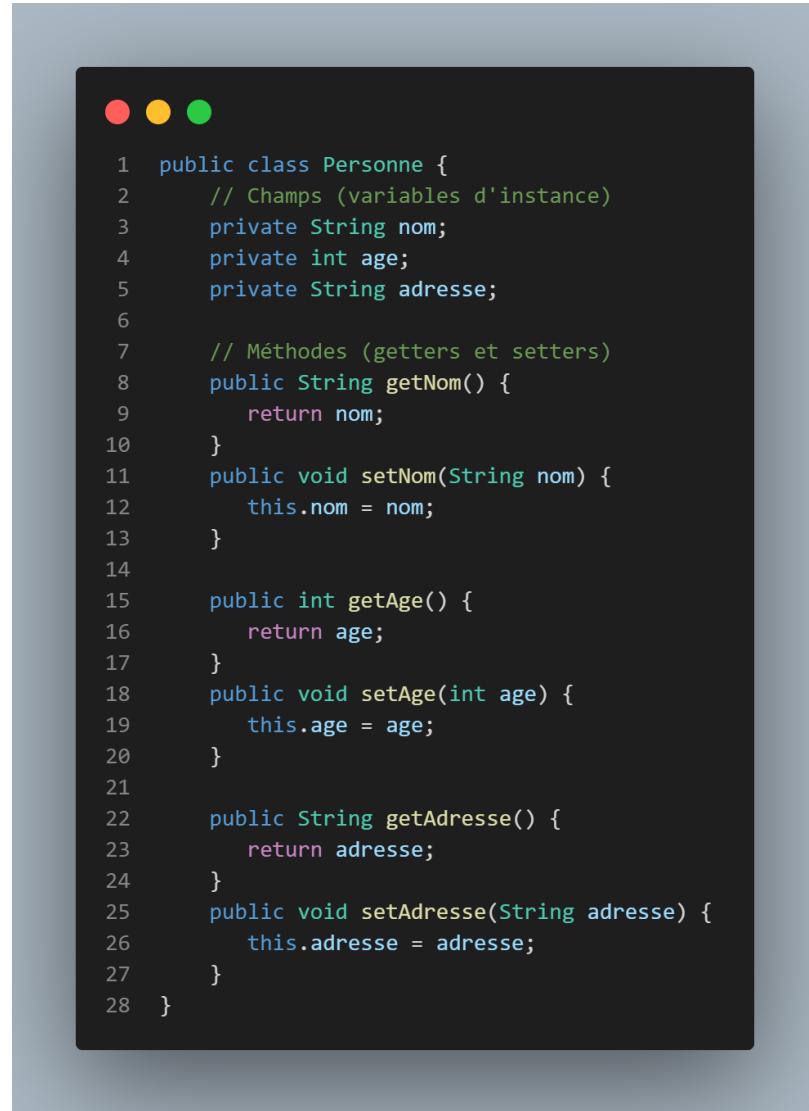
Dans cet exemple, nous avons déclaré trois champs : "nom", "âge" et "adresse". Ces champs représentent les données spécifiques à chaque objet de la classe "Personne".

L'encapsulation :

L'encapsulation est un concept clé de la programmation orientée objet qui consiste à cacher les détails internes d'une classe et à fournir un accès contrôlé aux données et aux fonctionnalités de cette classe. Cela se fait en définissant des méthodes spéciales appelées "getters" ou "accesseurs" et "setters" ou "mutateurs" pour accéder aux champs privés de la classe.

Les getters sont des méthodes qui permettent d'obtenir la valeur d'un champ, tandis que les setters sont des méthodes qui permettent de modifier la valeur d'un champ.

Voici comment nous pourrions ajouter des getters et des setters à notre classe "Personne" :



```
1 public class Personne {
2     // Champs (variables d'instance)
3     private String nom;
4     private int age;
5     private String adresse;
6
7     // Méthodes (getters et setters)
8     public String getNom() {
9         return nom;
10    }
11    public void setNom(String nom) {
12        this.nom = nom;
13    }
14
15    public int getAge() {
16        return age;
17    }
18    public void setAge(int age) {
19        this.age = age;
20    }
21
22    public String getAdresse() {
23        return adresse;
24    }
25    public void setAdresse(String adresse) {
26        this.adresse = adresse;
27    }
28 }
```

Maintenant, nous pouvons utiliser les getters et les setters pour accéder aux champs privés de la classe "Personne" depuis l'extérieur de la classe. Cela permet de contrôler l'accès aux données et de garantir leur intégrité.

Pour une meilleure lisibilité du code nous pouvons réunir chaque champ à son getter et son setter, puis ajouter le constructeur au début :



```
1 public class Personne {
2     //Constructeur
3     public Personne(String nom, int age, String adresse) {
4         this.nom = nom;
5         this.age = age;
6         this.adresse = adresse;
7     }
8     //Champs (variables d'instance) & Méthodes (getters et setters)
9     private String nom;
10    public String getNom() {
11        return nom;
12    }
13    public void setNom(String nom) {
14        this.nom = nom;
15    }
16
17    private int age;
18    public int getAge() {
19        return age;
20    }
21    public void setAge(int age) {
22        this.age = age;
23    }
24
25    private String adresse;
26    public String getAdresse() {
27        return adresse;
28    }
29    public void setAdresse(String adresse) {
30        this.adresse = adresse;
31    }
32 }
```

L'encapsulation offre plusieurs avantages, notamment :

- La possibilité de contrôler l'accès aux données en rendant certains champs privés et en fournissant des méthodes pour y accéder.
- La possibilité de valider les données avant de les affecter à un champ, en utilisant des conditions ou des règles spécifiques.
- La flexibilité de modifier la mise en œuvre interne d'une classe sans affecter le code externe qui utilise cette classe.

En résumé, les champs représentent les données spécifiques à chaque objet d'une classe. L'encapsulation permet de cacher les détails internes d'une classe et de contrôler l'accès aux données en utilisant des getters et des setters.

Exercice 3 :

1. Créez deux classes, "Chien" et "Chat", avec les attributs suivants :
 - nom (String)
 - age (int)
 - vacciné (boolean)
2. Assurez-vous de fournir des constructeurs, des getters et des setters appropriés pour chaque attribut.
3. Dans la classe "App", créez deux instances de la classe "Chat" avec des caractéristiques différentes. L'une des instances doit être vaccinée (vacciné = true) et l'autre ne doit pas l'être (vacciné = false).
4. Utilisez une condition ternaire pour afficher les informations appropriées pour chaque instance de chat. Si un chat est vacciné, affichez la phrase : "Bonjour, je m'appelle [nom du chat], j'ai [âge du chat] ans et je suis vacciné." Sinon, affichez la phrase : "Bonjour, je m'appelle [nom du chat], j'ai [âge du chat] ans et je ne suis pas vacciné."

Exercice 4 :

1. Ajoutez un nouvel attribut "action" de type String à la classe "Chien" et à la classe "Chat".
2. Pour la classe "Chien", définissez la valeur de l'attribut "action" comme "woof-woof".
3. Pour la classe "Chat", définissez la valeur de l'attribut "action" comme "miaw-miaw".
4. Dans la classe "Chat", ajoutez une méthode appelée "demanderMiaulement" qui demande à l'utilisateur s'il souhaite entendre le chat miauler.
 - Utilisez la classe "Scanner" pour lire l'entrée de l'utilisateur.
 - Si l'utilisateur entre "y" (oui), affichez "miaw-miaw" sur la console.

- Sinon, si l'utilisateur entre "n" (non) affichez "OK, peut-être une prochaine fois !".
 - Dans tous les autres cas, affichez "Je n'ai pas compris votre réponse" et reposez la question à l'utilisateur.
5. Dans la méthode "main" de la classe "App", créez une instance de la classe "Chat" et appelez la méthode "demanderMiaulement" sur cette instance.

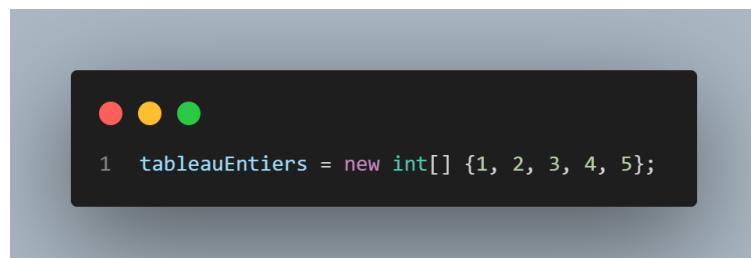
Les tableaux et les collections : les tableaux en Java.

Les tableaux sont des structures de données utilisées pour stocker des éléments de **même type** de manière séquentielle. En Java, les tableaux sont des objets qui peuvent contenir un **ensemble fixe** d'éléments. Voici comment déclarer, initialiser et accéder aux éléments d'un tableau en Java.

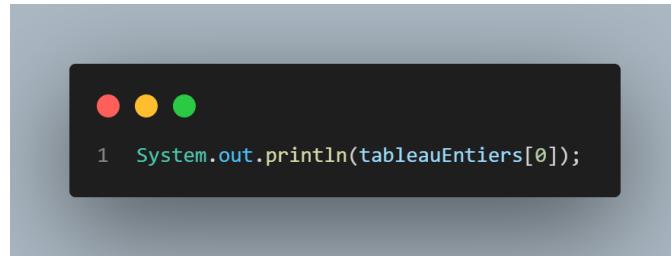
Pour commencer, déclarons un tableau d'entiers :



Cette déclaration indique que nous aurons un tableau d'entiers, mais il n'a pas encore été initialisé avec des valeurs. Pour initialiser le tableau et lui assigner des valeurs, nous pouvons utiliser la syntaxe suivante :

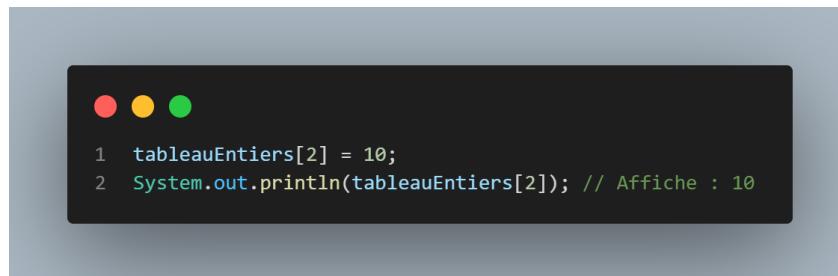


Maintenant, le tableau "tableauEntiers" contient les entiers 1, 2, 3, 4 et 5. Nous pouvons accéder aux éléments du tableau en utilisant l'index, qui commence à 0. Par exemple, pour accéder au premier élément du tableau, nous utilisons :



```
1 System.out.println(tableauEntiers[0]);
```

Nous pouvons également modifier la valeur d'un élément du tableau en utilisant son index :



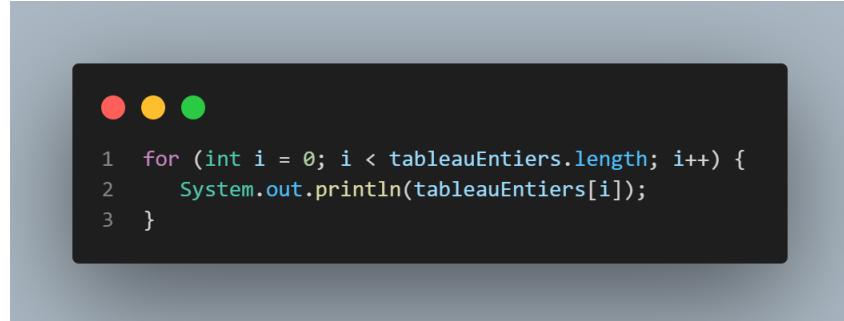
```
1 tableauEntiers[2] = 10;
2 System.out.println(tableauEntiers[2]); // Affiche : 10
```

De plus, nous pouvons obtenir la longueur du tableau à l'aide de la propriété "length" :



```
1 System.out.println(tableauEntiers.length);
```

Il est également possible de parcourir un tableau à l'aide de boucles. Par exemple, une boucle "for" peut être utilisée pour afficher tous les éléments du tableau :



```
● ● ●
1 for (int i = 0; i < tableauEntiers.length; i++) {
2     System.out.println(tableauEntiers[i]);
3 }
```

Ce code affiche tous les éléments du tableau "tableauEntiers" sur des lignes séparées.

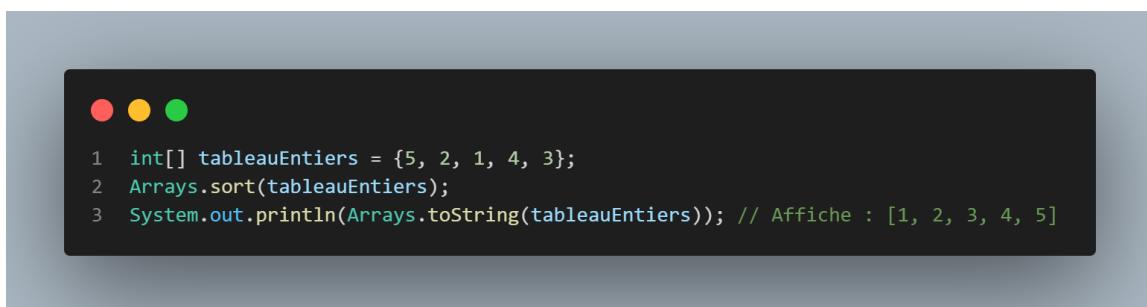
La méthode "**toString**" : Cette méthode renvoie une représentation sous forme de chaîne de caractères du tableau :



```
● ● ●
1 String tableauString = Arrays.toString(tableauEntiers);
2 System.out.println(tableauString); // Affiche : [1, 2, 3, 4, 5]
```

Notez que pour pouvoir utiliser l'objet "Arrays" il faut l'importer dans votre code avec "**import java.util.Arrays;**".

La méthode "**sort**" : Cette méthode trie les éléments du tableau dans l'ordre croissant :



```
● ● ●
1 int[] tableauEntiers = {5, 2, 1, 4, 3};
2 Arrays.sort(tableauEntiers);
3 System.out.println(Arrays.toString(tableauEntiers)); // Affiche : [1, 2, 3, 4, 5]
```

La méthode "**binarySearch**" : Cette méthode effectue une recherche binaire dans le tableau trié et renvoie l'index de l'élément recherché s'il est présent, sinon renvoie un index négatif :

```
● ● ●  
1 int index = Arrays.binarySearch(tableauEntiers, 3);  
2 System.out.println(index); // Affiche : 2 (index de l'élément 3 dans le tableau trié)
```

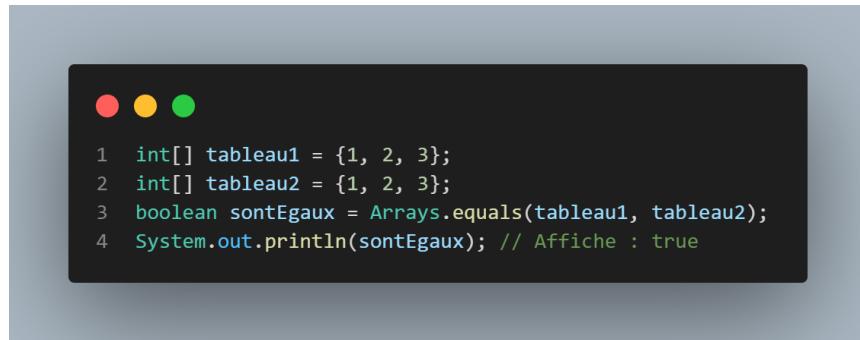
La méthode “**fill**” : Cette méthode remplit le tableau avec une valeur donnée :

```
● ● ●  
1 int[] tableauEntiers = new int[5];  
2 Arrays.fill(tableauEntiers, 10);  
3 System.out.println(Arrays.toString(tableauEntiers)); // Affiche : [10, 10, 10, 10, 10]
```

La méthode “**copyOf**” : Cette méthode crée une copie du tableau avec une nouvelle taille donnée :

```
● ● ●  
1 int[] tableauEntiers = {1, 2, 3};  
2 int[] copieTableau = Arrays.copyOf(tableauEntiers, 5);  
3 System.out.println(Arrays.toString(copieTableau)); // Affiche : [1, 2, 3, 0, 0]
```

La méthode “**equals**” : Cette méthode compare deux tableaux pour vérifier s'ils sont égaux :



En conclusion, les tableaux en Java sont des structures de données qui permettent de stocker des **éléments de même type** de manière séquentielle. Ils peuvent être déclarés, initialisés et accédés à l'aide d'index. Les tableaux offrent une façon pratique de gérer des collections d'éléments de **taille fixe**.

En Java, les tableaux traditionnels ont une taille fixe et ne peuvent pas être étendus ou réduits dynamiquement comme dans d'autres langages. Par conséquent, il n'y a pas d'équivalent direct des opérations de "push", "pop" ou "put" sur un tableau en Java. Cependant, vous pouvez utiliser des structures de données telles que les listes (ArrayList) et les piles (Stack) pour obtenir des fonctionnalités similaires.

Les tableaux et les collections : les collections en Java.

Les tableaux en Java ont une taille fixe et ne peuvent pas être étendus ou réduits dynamiquement. Cependant, Java offre une variété de structures de données appelées "collections" qui permettent de manipuler des ensembles d'éléments de manière plus flexible. Les collections offrent des fonctionnalités avancées telles que l'ajout, la suppression, la recherche et le tri des éléments. Voici quelques-unes des collections couramment utilisées en Java :

Les tableaux et les collections : les ArrayList en Java.

L'ArrayList est une collection dynamique qui permet de stocker des éléments de manière séquentielle. Elle peut être utilisée pour stocker des objets de n'importe quel type. Voici un exemple d'utilisation de l'ArrayList :

```
● ● ●  
1 ArrayList<String> liste = new ArrayList<>();  
2 liste.add("élément 1");  
3 liste.add("élément 2");  
4 liste.remove(0);  
5 System.out.println(liste.get(0));
```

Vous l'aurez compris les tableaux sont pratiques pour stocker des collections d'éléments de taille fixe, mais parfois nous avons besoin de structures de données qui peuvent grandir ou rétrécir dynamiquement. C'est là que les ArrayList entrent en jeu. Les ArrayList sont des collections dynamiques implémentées en utilisant des tableaux sous-jacents en Java.

Les ArrayList offrent des fonctionnalités avancées pour manipuler les données, telles que l'ajout, la suppression, la recherche et le tri des éléments. Ils sont flexibles et faciles à utiliser. Voyons ça un peu plus en détail.

Voici comment déclarer et utiliser un ArrayList en Java :

Déclaration et initialisation d'un ArrayList :

1. Pour créer un ArrayList, vous devez importer la classe “[java.util.ArrayList](#)”. Voici un exemple de déclaration et d'initialisation d'un ArrayList contenant des chaînes de caractères :

```
● ● ●  
1 import java.util.ArrayList;  
2  
3 public class App {  
4     public static void main(String[] args) {  
5         ArrayList<String> liste = new ArrayList<>();  
6     }  
7 }
```

Ajouter des éléments à un ArrayList :

2. Vous pouvez utiliser la méthode “**add**” pour ajouter des éléments à un ArrayList.

Voici un exemple qui ajoute des prénoms à la liste :

```
● ● ●  
1 ArrayList<String> liste = new ArrayList<>();  
2 liste.add("David");  
3 liste.add("Anne-Sophie");  
4 liste.add("Matthias");
```

Accéder aux éléments d'un ArrayList :

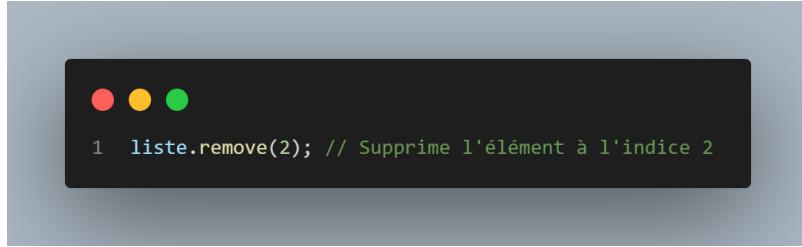
3. Vous pouvez accéder aux éléments d'un ArrayList en utilisant la méthode “**get**”.

Les indices des éléments commencent à 0. Voici comment accéder aux éléments de notre liste :

```
● ● ●  
1 String premierElement = liste.get(0);  
2 String deuxiemeElement = liste.get(1);
```

Supprimer des éléments d'un ArrayList :

4. Pour supprimer un élément d'un ArrayList, vous pouvez utiliser la méthode “**remove**”. Vous pouvez spécifier l'indice de l'élément à supprimer ou directement l'objet lui-même. Voici un exemple de suppression d'un élément de la liste :



Vérifier si un élément existe dans un ArrayList :

5. Vous pouvez utiliser la méthode “**contains**” pour vérifier si un élément existe dans un ArrayList. Elle renvoie “**true**” si l’élément est présent et “**false**” sinon. Voici un exemple :



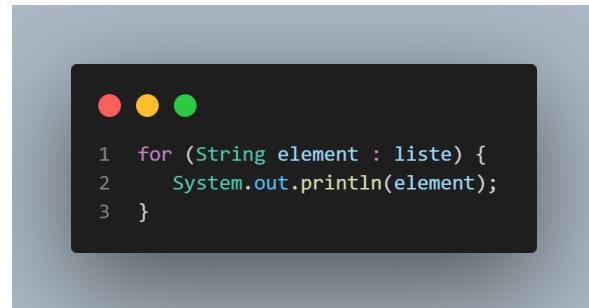
Taille d'un ArrayList :

6. Pour connaître le nombre d’éléments dans un ArrayList, vous pouvez utiliser la méthode “**size**”. Elle renvoie le nombre d’éléments présents dans la liste. Voici un exemple :



Parcourir un ArrayList avec une boucle :

7. Vous pouvez utiliser une boucle “**for-each**” pour parcourir tous les éléments d’un ArrayList. Voici un exemple :



En Java, List et ArrayList sont deux concepts différents mais étroitement liés. Voici la différence entre les deux :

1. List : List est une interface dans Java qui représente une collection ordonnée d'éléments. Elle définit un ensemble de méthodes pour manipuler et accéder aux éléments de la liste, telles que "add", "remove", "get", "size", etc. Les listes sont dynamiques, ce qui signifie qu'elles peuvent grandir ou rétrécir en fonction des besoins.
2. ArrayList : ArrayList est une classe qui implémente l'interface List. Cela signifie que ArrayList fournit une implémentation concrète de toutes les méthodes définies dans l'interface List. C'est l'une des implementations les plus couramment utilisées de l'interface List. Les ArrayLists sont des tableaux dynamiques, ce qui signifie qu'ils peuvent changer de taille automatiquement lorsqu'on ajoute ou supprime des éléments.

En résumé, List est une interface qui définit le comportement général d'une liste, tandis que ArrayList est une classe qui implémente cette interface et fournit une implémentation spécifique basée sur un tableau dynamique. En utilisant l'interface List, vous pouvez écrire un code qui est plus générique et qui peut être facilement adapté à d'autres implementations de listes, telles que LinkedList.

Exemple d'utilisation :

```
1 import java.util.List;
2 import java.util.ArrayList;
3
4 public class Main {
5     public static void main(String[] args) {
6         // Déclaration de la liste en utilisant l'interface List
7         List<String> maListe = new ArrayList<>();
8
9         // Ajout d'éléments à la liste
10        maListe.add("Java");
11        maListe.add("Python");
12        maListe.add("C++");
13
14        // Accès à un élément de la liste
15        String element = maListe.get(0);
16        System.out.println(element); // Affiche "Java"
17
18        // Suppression d'un élément de la liste
19        maListe.remove(1);
20
21        // Parcours de la liste
22        for (String item : maListe) {
23            System.out.println(item);
24        }
25    }
26 }
```

Dans cet exemple, nous déclarons une liste en utilisant l'interface List, mais nous l'instancions avec une instance de ArrayList. Nous ajoutons des éléments à la liste, accédons à un élément spécifique, supprimons un élément et parcourons la liste à l'aide d'une boucle foreach. Cette approche nous permet de changer facilement l'implémentation de la liste en utilisant une autre classe qui implémente l'interface List, comme LinkedList, sans modifier le reste du code.

Les tableaux et les collections : les LinkedList en Java.

Les LinkedList en Java sont une autre implémentation de l'interface List, tout comme ArrayList. Cependant, les LinkedList diffèrent d'ArrayList sur plusieurs aspects,

notamment la façon dont les éléments sont stockés en mémoire et les performances d'accès aux éléments.

Les `LinkedList` sont une structure de données linéaire qui stocke les éléments sous forme de nœuds liés les uns aux autres. Chaque nœud contient une référence vers l'élément suivant et éventuellement vers l'élément précédent, formant ainsi une chaîne de nœuds. Contrairement à `ArrayList` qui est basé sur un tableau dynamique, `LinkedList` ne nécessite pas un espace de stockage continu en mémoire.

Avantages des `LinkedList` :

1. Insertions et suppressions efficaces : L'ajout ou la suppression d'un élément dans une `LinkedList` peut être plus efficace que dans un `ArrayList`, car cela ne nécessite pas de déplacement des éléments suivants pour maintenir l'ordre.
2. Structure dynamique : Les `LinkedList` peuvent grandir ou rétrécir facilement, car elles n'ont pas besoin d'un espace de stockage continu. Cela les rend plus flexibles dans les scénarios où la taille de la liste peut varier fréquemment.

Inconvénients des `LinkedList` :

1. Accès moins efficace : L'accès direct à un élément dans une `LinkedList` est moins efficace que dans un `ArrayList`. Pour accéder à un élément, il faut parcourir les nœuds depuis le début ou la fin de la liste jusqu'à atteindre l'élément souhaité.
2. Utilisation de mémoire supplémentaire : En plus de stocker les éléments, les `LinkedList` doivent également stocker les références vers les nœuds suivants et éventuellement les nœuds précédents, ce qui nécessite une utilisation de mémoire supplémentaire par rapport à un `ArrayList`.

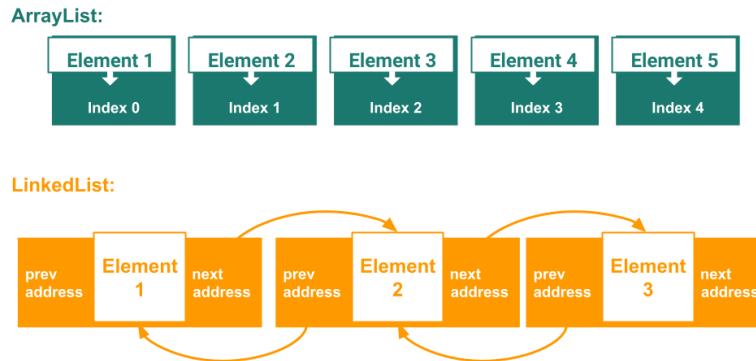
Exemple d'utilisation des `LinkedList` :

```
● ● ●

1 import java.util.LinkedList;
2 import java.util.List;
3
4 public class Main {
5     public static void main(String[] args) {
6         // Déclaration de la LinkedList
7         List<String> maListe = new LinkedList<>();
8
9         // Ajout d'éléments à la liste
10        maListe.add("Java");
11        maListe.add("Python");
12        maListe.add("C++");
13
14        // Accès à un élément de la liste
15        String element = maListe.get(0);
16        System.out.println(element); // Affiche "Java"
17
18        // Suppression d'un élément de la liste
19        maListe.remove(1);
20
21        // Parcours de la liste
22        for (String item : maListe) {
23            System.out.println(item);
24        }
25    }
26 }
```

Dans cet exemple, nous utilisons la classe `LinkedList` pour créer une liste chaînée. Nous ajoutons des éléments, accédons à un élément spécifique, supprimons un élément et parcourons la liste à l'aide d'une boucle `foreach`. Les `LinkedList` offrent une flexibilité accrue pour les opérations d'insertion et de suppression, mais peuvent être moins performantes pour l'accès direct aux éléments.

ArrayList vs LinkedList



Les tableaux et les collections : les HashMap et HashSet en Java.

En Java, les interfaces Map et Set font partie du framework de collections et offrent des fonctionnalités puissantes pour stocker, manipuler et organiser des données. Ces interfaces sont implémentées par différentes classes, telles que HashMap, HashSet, TreeMap et TreeSet, qui fournissent des structures de données spécifiques pour répondre à des besoins particuliers.

L'interface Map :

- L'interface Map représente une structure de données associant des clés à des valeurs. Chaque élément est une paire clé-valeur unique, où la clé est utilisée pour accéder à la valeur associée.
- Les clés dans une Map doivent être uniques, tandis que les valeurs peuvent être dupliquées.
- Les principales méthodes fournies par l'interface Map sont : put(key, value) pour ajouter une paire clé-valeur, get(key) pour récupérer une valeur à partir d'une clé, remove(key) pour supprimer une paire clé-valeur, containsKey(key) pour vérifier si une clé existe, et size() pour obtenir le nombre d'éléments dans la Map.

L'interface Set :

-
- L'interface Set représente une collection d'éléments uniques, sans doublons. Chaque élément dans un Set est unique et ne peut être présent qu'une seule fois.
 - Les Set ne garantissent pas un ordre spécifique des éléments.
 - Les principales méthodes fournies par l'interface Set sont : add(element) pour ajouter un élément, remove(element) pour supprimer un élément, contains(element) pour vérifier si un élément existe, et size() pour obtenir le nombre d'éléments dans le Set.

Ces interfaces offrent une flexibilité et une efficacité dans la gestion des données en fonction des besoins spécifiques. Les implémentations telles que HashMap, HashSet, TreeMap et TreeSet permettent d'utiliser ces fonctionnalités de manière optimale en fonction du contexte d'utilisation.

Il est important de noter que l'utilisation de ces interfaces nécessite une bonne compréhension de leurs spécificités, de leurs méthodes et de leurs performances. En comprenant les différences entre Map et Set, ainsi que leurs implémentations, vous pouvez choisir la structure de données la plus adaptée à vos besoins et garantir une manipulation efficace des données dans votre application Java.

Les HashMap et HashSet en Java sont des implémentations de l'interface Map et Set respectivement. Ils offrent des fonctionnalités avancées pour stocker et gérer des données de manière efficace.

HashMap :

- HashMap est une classe qui implémente l'interface Map en utilisant une structure de données appelée "table de hachage". Elle permet de stocker des paires clé-valeur, où chaque clé est unique. Les clés et les valeurs peuvent être de n'importe quel type d'objet.
- Les HashMap offrent une recherche et un accès rapides aux valeurs en utilisant des clés. Ils sont efficaces pour les opérations de recherche, d'insertion et de suppression.

- Les clés dans un HashMap doivent être uniques. Si une clé existante est utilisée pour ajouter une nouvelle valeur, la valeur existante sera remplacée.

Exemple d'utilisation d'un HashMap :



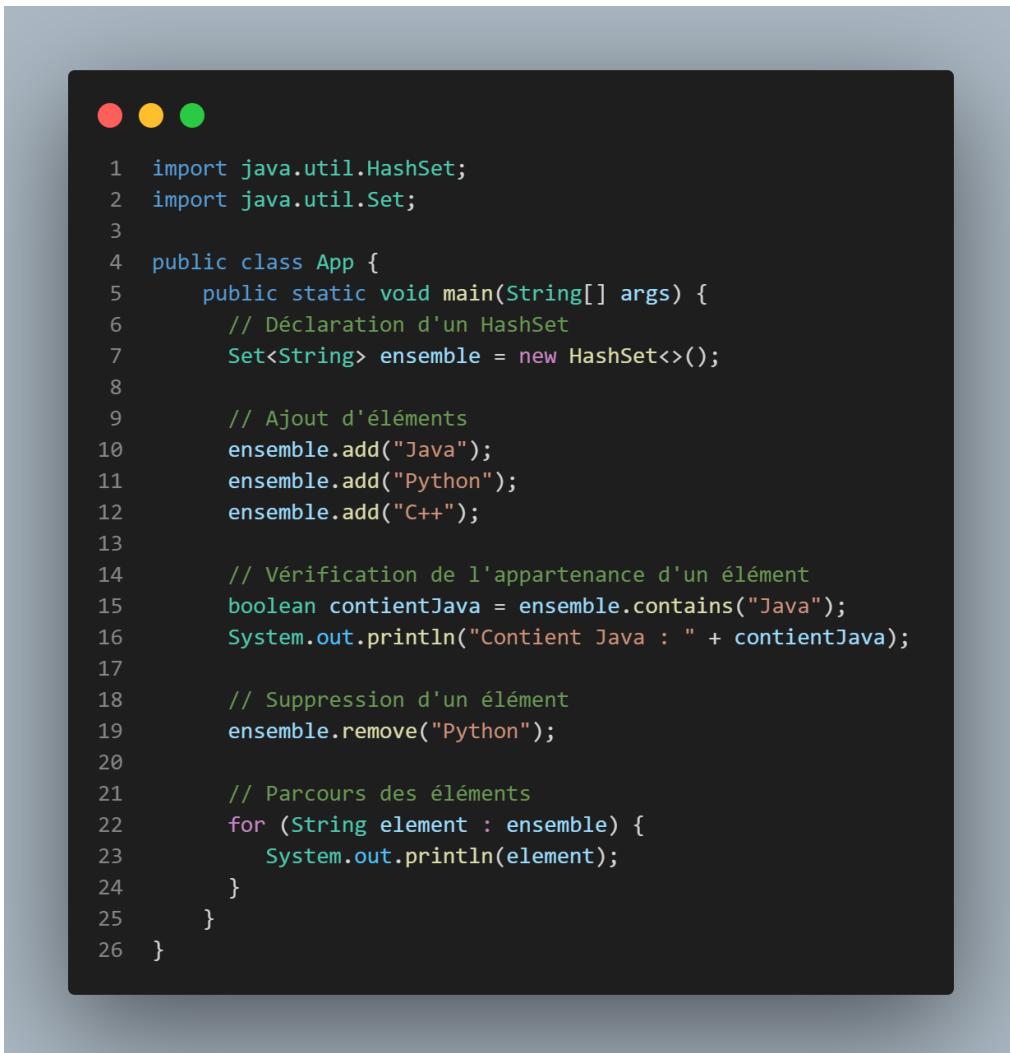
```
1 import java.util.HashMap;
2 import java.util.Map;
3
4 public class App {
5     public static void main(String[] args) {
6         // Déclaration d'un HashMap
7         Map<String, Integer> scores = new HashMap<>();
8
9         // Ajout de paires clé-valeur
10        scores.put("Remi", 85);
11        scores.put("Bessem", 92);
12        scores.put("Zuzanna", 78);
13
14        // Accès à une valeur à partir d'une clé
15        int scoreRemi = scores.get("Remi");
16        System.out.println("Score de Remi : " + scoreRemi);
17
18        // Suppression d'une paire clé-valeur
19        scores.remove("Zuzanna");
20
21        // Parcours des paires clé-valeur
22        for (Map.Entry<String, Integer> entry : scores.entrySet()) {
23            String nom = entry.getKey();
24            int score = entry.getValue();
25            System.out.println("Score de " + nom + " : " + score);
26        }
27    }
28 }
```

Dans cet exemple, nous utilisons la classe HashMap pour créer une structure de données associant des noms à des scores. Nous ajoutons des paires clé-valeur, accédons aux scores en utilisant les clés, supprimons une paire clé-valeur et parcourons toutes les paires clé-valeur à l'aide d'une boucle foreach.

HashSet :

- HashSet est une classe qui implémente l'interface Set en utilisant une table de hachage interne. Il permet de stocker des éléments uniques, sans doublons.
- Les HashSet offrent des opérations de recherche, d'insertion et de suppression rapides. Ils sont efficaces pour tester l'appartenance d'un élément à l'ensemble et pour éliminer les doublons dans une collection.
- Les éléments stockés dans un HashSet n'ont pas d'ordre spécifique. L'ensemble garantit uniquement l'unicité des éléments.

Exemple d'utilisation d'un HashSet :



```
● ● ●

1 import java.util.HashSet;
2 import java.util.Set;
3
4 public class App {
5     public static void main(String[] args) {
6         // Déclaration d'un HashSet
7         Set<String> ensemble = new HashSet<>();
8
9         // Ajout d'éléments
10        ensemble.add("Java");
11        ensemble.add("Python");
12        ensemble.add("C++");
13
14        // Vérification de l'appartenance d'un élément
15        boolean contientJava = ensemble.contains("Java");
16        System.out.println("Contient Java : " + contientJava);
17
18        // Suppression d'un élément
19        ensemble.remove("Python");
20
21        // Parcours des éléments
22        for (String element : ensemble) {
23            System.out.println(element);
24        }
25    }
26}
```

Dans cet exemple, nous utilisons la classe HashSet pour créer un ensemble d'éléments uniques. Nous ajoutons des éléments, vérifions l'appartenance d'un élément, supprimons un élément et parcourons tous les éléments à l'aide d'une boucle foreach.

Exercice 5 :

Utilisez un HashSet pour stocker une liste de noms d'étudiants. Demandez à l'utilisateur d'entrer les noms jusqu'à ce qu'il entre "q" pour quitter. Ensuite, affichez tous les noms d'étudiants enregistrés dans l'ensemble. Assurez-vous de gérer les doublons et d'afficher les noms dans l'ordre d'ajout.

Voici quelques exercices qui vous permettront de pratiquer l'utilisation des ArrayList, LinkedList et/ou Arrays :

Exercice 6:

Créez une méthode qui prend en entrée un tableau d'entiers et renvoie la somme de tous les éléments du tableau.

Exercice 7:

Créez une méthode qui prend en entrée une ArrayList de chaînes de caractères et renvoie la chaîne la plus longue de la liste.

Exercice 8:

Créez une méthode qui prend en entrée une LinkedList d'objets et renvoie la taille de la liste.

Exercice 9:

Créez une méthode qui prend en entrée un tableau d'entiers et un entier cible. La méthode doit renvoyer true si l'entier cible est présent dans le tableau, sinon elle doit renvoyer false.

Exercice 10:

Créez une méthode qui prend en entrée une ArrayList d'objets et un objet cible. La méthode doit renvoyer l'indice de la première occurrence de l'objet cible dans la liste, ou -1 s'il n'est pas présent.

Exercice 11:

Créez une méthode qui prend en entrée un tableau d'entiers trié par ordre croissant et un entier cible. La méthode doit renvoyer true si l'entier cible est présent dans le tableau en utilisant une recherche binaire, sinon elle doit renvoyer false.

Exercice 12:

Créez une méthode qui prend en entrée une LinkedList d'entiers et supprime tous les éléments pairs de la liste.

Exercice 13:

Créez une méthode qui prend en entrée un tableau d'entiers et renvoie un nouveau tableau qui contient uniquement les nombres premiers présents dans le tableau d'origine.

Ces exercices vous donneront l'occasion de manipuler différentes structures de données (ArrayList, LinkedList, Arrays) et d'appliquer des opérations courantes telles que la recherche, la modification et la suppression d'éléments. N'hésitez pas à adapter ces exercices en fonction de vos besoins spécifiques et à explorer d'autres fonctionnalités offertes par ces collections.











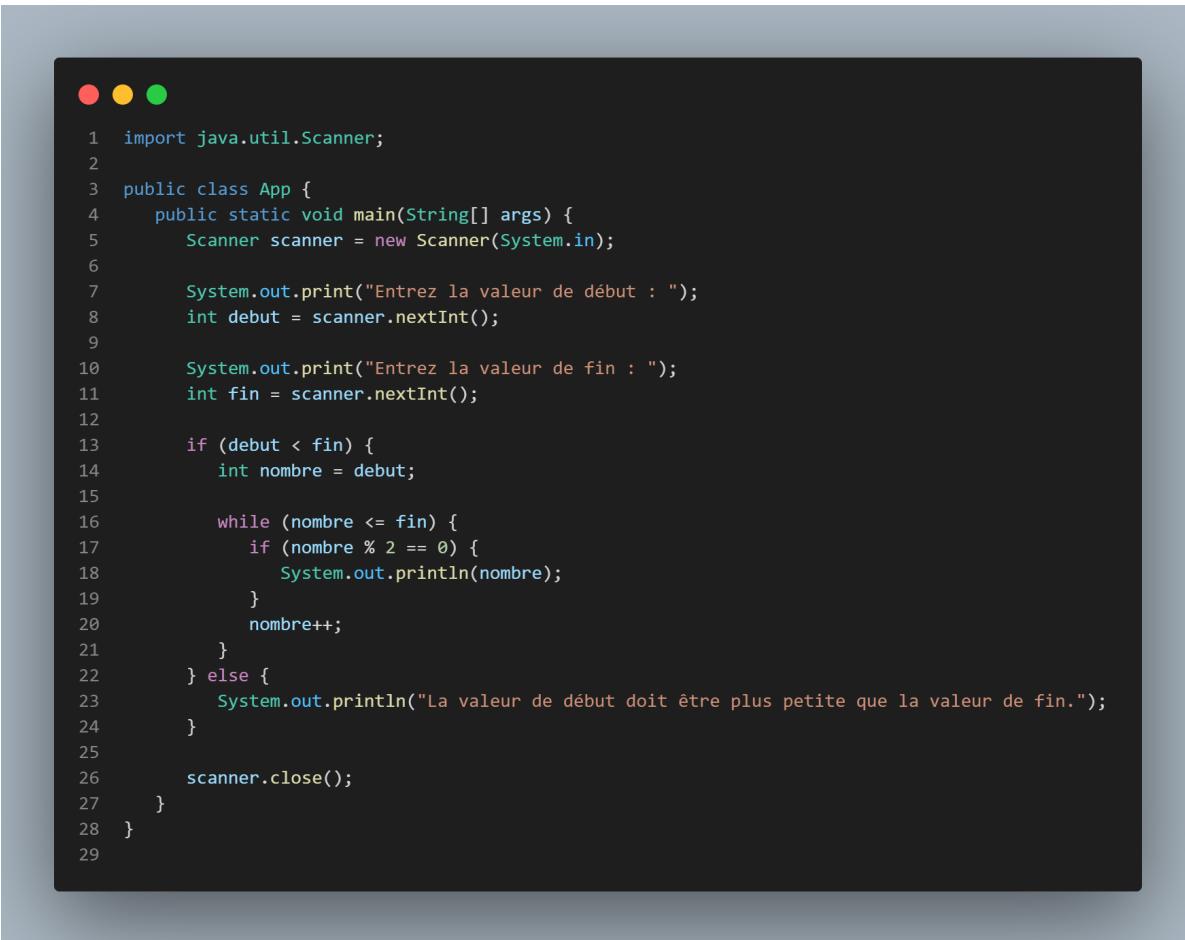
Correction des exercices.

Exercice 1 :

```
● ● ●
1 public class App {
2     public static void main(String[] args) {
3         int nombre = 0;
4
5         while (nombre <= 10) {
6             if (nombre % 2 == 0) {
7                 System.out.println(nombre);
8             }
9             nombre += 2;
10        }
11    }
12 }
```

Ce programme utilise une boucle while pour itérer de 0 à 10. À chaque itération, il vérifie si le nombre est pair en utilisant l'opérateur modulo (%) pour vérifier si le reste de la division par 2 est égal à 0. Si c'est le cas, il affiche le nombre sur la console à l'aide de "System.out.println()". Ensuite, il incrémente la valeur de "nombre" de 2 pour passer au prochain nombre pair. Le résultat affiché sera les chiffres pairs de 0 à 10 : 0, 2, 4, 6, 8, 10.

Exercice 2 :



```
1 import java.util.Scanner;
2
3 public class App {
4     public static void main(String[] args) {
5         Scanner scanner = new Scanner(System.in);
6
7         System.out.print("Entrez la valeur de début : ");
8         int debut = scanner.nextInt();
9
10        System.out.print("Entrez la valeur de fin : ");
11        int fin = scanner.nextInt();
12
13        if (debut < fin) {
14            int nombre = debut;
15
16            while (nombre <= fin) {
17                if (nombre % 2 == 0) {
18                    System.out.println(nombre);
19                }
20                nombre++;
21            }
22        } else {
23            System.out.println("La valeur de début doit être plus petite que la valeur de fin.");
24        }
25
26        scanner.close();
27    }
28 }
29
```

Ce programme demande à l'utilisateur d'entrer la valeur de début et la valeur de fin à l'aide de la classe Scanner. Ensuite, il vérifie si la valeur de début est inférieure à la valeur de fin. Si c'est le cas, il itère à travers tous les nombres entre la valeur de début et la valeur de fin à l'aide d'une boucle while. Il vérifie si chaque nombre est pair en utilisant l'opérateur modulo (%) et affiche les nombres pairs sur la console. Si la valeur de début est supérieure ou égale à la valeur de fin, il affiche un message d'erreur approprié.

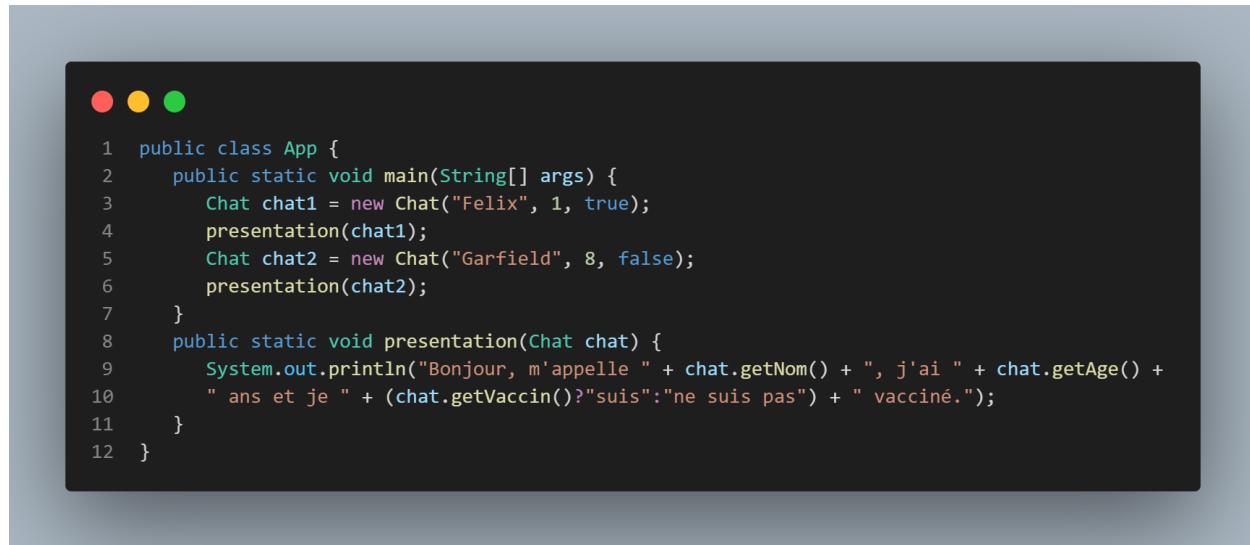
Exercice 3 :

Les classes Chien et Chat:

```
1 public class Chat {  
2     public Chat(String nom, int age, boolean vaccin) {  
3         this.nom = nom;  
4         this.age = age;  
5         this.vaccin = vaccin;  
6     }  
7  
8     private String nom;  
9     public String getNom() {  
10        return nom;  
11    }  
12    public void setNom(String nom) {  
13        this.nom = nom;  
14    }  
15  
16    private int age;  
17    public int getAge() {  
18        return age;  
19    }  
20    public void setAge(int age) {  
21        this.age = age;  
22    }  
23  
24    private boolean vaccin;  
25    public boolean getVaccin() {  
26        return vaccin;  
27    }  
28    public void setVaccin(boolean vaccin) {  
29        this.vaccin = vaccin;  
30    }  
31 }
```

```
1 public class Chien {  
2     public Chien(String nom, int age, boolean vaccin) {  
3         this.nom = nom;  
4         this.age = age;  
5         this.vaccin = vaccin;  
6     }  
7  
8     private String nom;  
9     public String getNom() {  
10        return nom;  
11    }  
12    public void setNom(String nom) {  
13        this.nom = nom;  
14    }  
15  
16    private int age;  
17    public int getAge() {  
18        return age;  
19    }  
20    public void setAge(int age) {  
21        this.age = age;  
22    }  
23  
24    private boolean vaccin;  
25    public boolean getVaccin() {  
26        return vaccin;  
27    }  
28    public void setVaccin(boolean vaccin) {  
29        this.vaccin = vaccin;  
30    }  
31 }
```

La classe App :



```
1 public class App {
2     public static void main(String[] args) {
3         Chat chat1 = new Chat("Felix", 1, true);
4         presentation(chat1);
5         Chat chat2 = new Chat("Garfield", 8, false);
6         presentation(chat2);
7     }
8     public static void presentation(Chat chat) {
9         System.out.println("Bonjour, m'appelle " + chat.getNom() + ", j'ai " + chat.getAge() +
10            " ans et je " + (chat.getVaccin()? "suis": "ne suis pas") + " vacciné.");
11    }
12 }
```

Dans cet exercice, nous créons deux classes, "Chien" et "Chat", avec les attributs demandés : nom, âge et vacciné. Nous fournissons des constructeurs pour initialiser les attributs, ainsi que des getters et des setters pour y accéder.

Dans la classe "App", nous créons deux instances de la classe "Chat" : "chat1" et "chat2". L'une des instances est vaccinée (vaccine = true) et l'autre ne l'est pas (vaccine = false). Nous utilisons ensuite une condition ternaire pour afficher les informations appropriées pour chaque instance de chat.

L'exécution du programme affiche les phrases correspondantes pour chaque chat, en fonction de leur statut de vaccination.

N'hésitez pas à personnaliser les noms, les âges et les statuts de vaccination des chats pour tester différentes situations.

Exercice 4 :

```
● ● ●  
1 import java.util.Scanner;  
2  
3 public class Chat {  
4     public Chat(String nom, int age, boolean vaccin) {  
5         this.nom = nom;  
6         this.age = age;  
7         this.vaccin = vaccin;  
8         this.action = "miauw-miauw";  
9     }  
10  
11    public void demanderMiaulement() {  
12        Scanner scanner = new Scanner(System.in);  
13        System.out.print("Voulez-vous entendre le chat miauler ? (y/n) ");  
14        String reponse = scanner.nextLine();  
15  
16        if (reponse.equals("y")) {  
17            System.out.println(action);  
18        } else if (reponse.equals("n")) {  
19            System.out.println("OK, peut-être une prochaine fois !");  
20        } else {  
21            System.out.println("Je n'ai pas compris votre réponse");  
22            demanderMiaulement();  
23        }  
24    }  
25  
26    private String nom;  
27    public String getNom() {  
28        return nom;  
29    }  
30    public void setNom(String nom) {  
31        this.nom = nom;  
32    }  
33  
34    private int age;  
35    public int getAge() {  
36        return age;  
37    }  
38    public void setAge(int age) {  
39        this.age = age;  
40    }  
41  
42    private boolean vaccin;  
43    public boolean getVaccin() {  
44        return vaccin;  
45    }  
46    public void setVaccin(boolean vaccin) {  
47        this.vaccin = vaccin;  
48    }  
49  
50    private String action;  
51    public String getAction() {  
52        return action;  
53    }  
54}
```

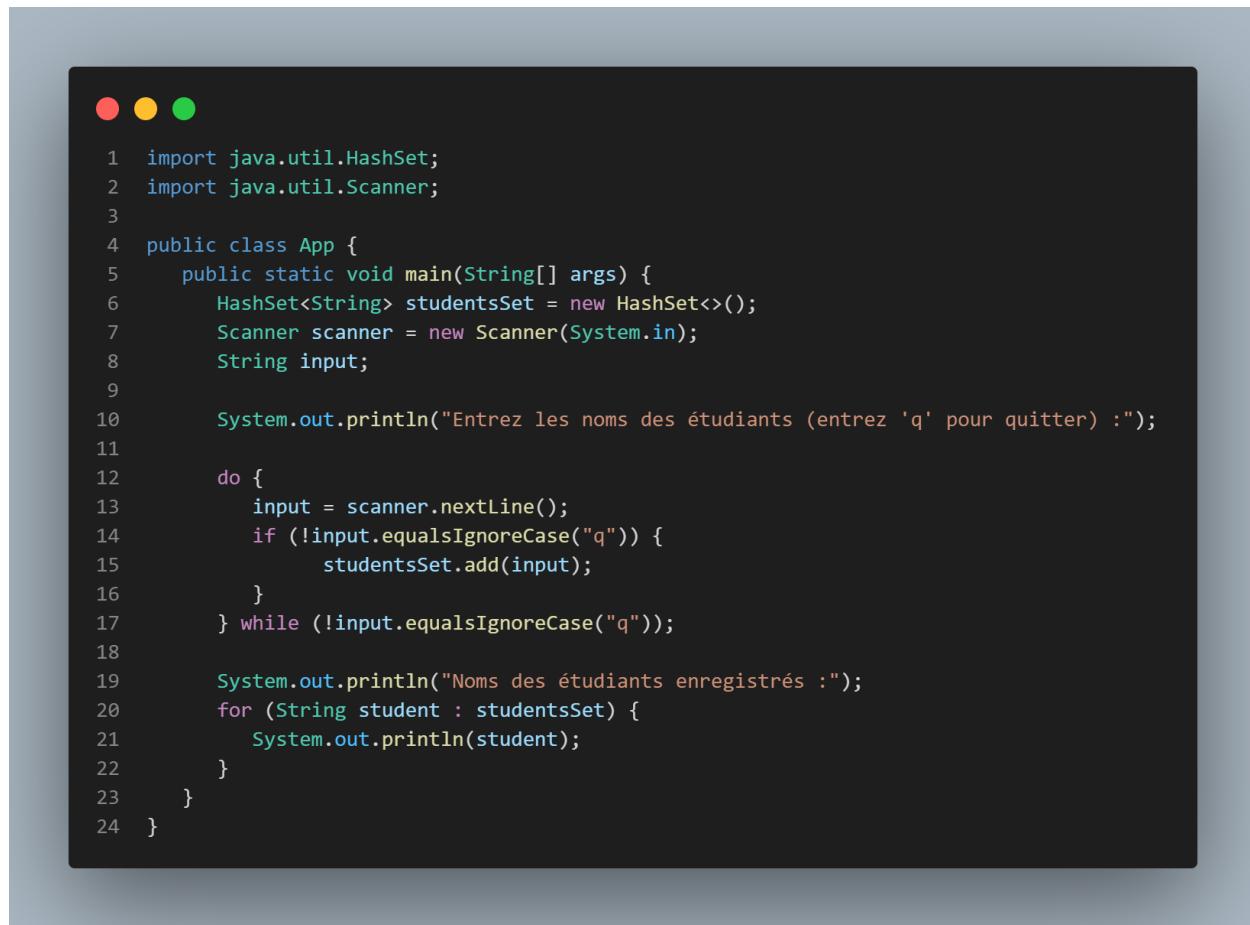
```
1 public class App {  
2     public static void main(String[] args) {  
3         Chat chat1 = new Chat("Felix", 1, true);  
4         presentation(chat1);  
5         Chat chat2 = new Chat("Garfield", 8, false);  
6         presentation(chat2);  
7     }  
8     public static void presentation(Chat chat) {  
9         System.out.println("Bonjour, m'appelle " + chat.getNom() + ", j'ai " + chat.getAge() +  
10            " ans et je " + (chat.getVaccin()?"suis":"ne suis pas") + " vacciné.");  
11         chat.demanderMiaulement();  
12     }  
13 }
```

Dans la classe "Chat", nous ajoutons une méthode "demanderMiaulement" qui demande à l'utilisateur s'il souhaite entendre le chat miauler. Nous utilisons la classe "Scanner" pour lire l'entrée de l'utilisateur. Selon la réponse de l'utilisateur, nous affichons "miauw-miauw" si la réponse est "y" (oui), "OK, peut-être une prochaine fois !" si la réponse est "n" (non), et nous affichons un message d'erreur si la réponse n'est ni "y" ni "n", puis nous reposons la question à l'utilisateur.

Dans la méthode "main" de la classe "App", nous créons une instance de la classe "Chat" et appelons la méthode "demanderMiaulement" sur cette instance.

L'exécution du programme demande à l'utilisateur s'il souhaite entendre le chat miauler. Selon la réponse de l'utilisateur, le programme affiche le miaulement du chat ou un message approprié.

Exercice 5 :



The screenshot shows a Java application running in a terminal window. The application prompts the user to enter student names, adds them to a HashSet, and then prints the names back out. The terminal window has three colored window control buttons (red, yellow, green) at the top.

```
1 import java.util.HashSet;
2 import java.util.Scanner;
3
4 public class App {
5     public static void main(String[] args) {
6         HashSet<String> studentsSet = new HashSet<>();
7         Scanner scanner = new Scanner(System.in);
8         String input;
9
10        System.out.println("Entrez les noms des étudiants (entrez 'q' pour quitter) :");
11
12        do {
13            input = scanner.nextLine();
14            if (!input.equalsIgnoreCase("q")) {
15                studentsSet.add(input);
16            }
17        } while (!input.equalsIgnoreCase("q"));
18
19        System.out.println("Noms des étudiants enregistrés :");
20        for (String student : studentsSet) {
21            System.out.println(student);
22        }
23    }
24 }
```

Dans cet exercice, nous utilisons un HashSet, qui est une implémentation de l'interface Set, pour stocker les noms d'étudiants. Le HashSet garantit que chaque nom est unique et gère automatiquement les doublons.

Nous utilisons un objet Scanner pour lire les entrées de l'utilisateur. La boucle do-while permet à l'utilisateur de saisir les noms jusqu'à ce qu'il entre "q" pour quitter. À chaque itération, si l'entrée n'est pas "q", le nom est ajouté à l'ensemble studentsSet à l'aide de la méthode add().

Une fois que l'utilisateur a terminé la saisie, nous parcourons l'ensemble studentsSet à l'aide d'une boucle for-each et affichons les noms des étudiants enregistrés.

L'utilisation de HashSet garantit que les noms seront affichés dans l'ordre d'ajout, sans doublons.

N'hésitez pas à modifier ou à étendre cet exemple selon vos besoins spécifiques.

