# Mots-clés en JAVA

53 mots…

## Les objets (11 mots)

|  |  |
| --- | --- |
| **ABSTRACT** | Méthode ou class abstraite, c’est à dire sans implémentation   * Ne peut être instanciée * La class définit des méthodes qui seront héritées par des class filles * Les méthodes abstraites sont s/corps mais surchargées dans la class fille * Utilisée pour définir un comportement par défaut |
| **CLASS** | Définit un objet de type class |
| **INTERFACE** | Définit un objet qui spécifie un comportement mais pas son code   * Méthodes et attributs toujours public |
| **IMPLEMENTS** | Indique qu’une class implémente une ou des interfaces |
| **ENUM** | Définit un objet de type énumération   * Utilisé comme constantes * Peut être utilisé dans un switch |
| **EXTENDS** | Indique qu’une class ou une interface étend une autre class ou d’autres interfaces   * Une class extends une autre class * Une Interface peut extends plusieurs interfaces * Quand pas de précision, une class hérite automatiquement de la class Object |
| **IMPORT** | Référence des class, packages entiers ou des méthodes pour utiliser dans les programmes sans écrire le nom complet => Raccourci   * Possibilité de faire des « Import Static » d’une méthode ou d’un attribut |
| **THIS** | Référence à l’objet courant   * Souvent utilisé dans les setters pour différencier la variable de class de la variable passée en paramètre * Possibilité d’appeler la méthode this faisant référence au constructeur de l’objet à partir d’un autre constructeur * Première instruction dans un constructeur * Permet d’accéder à l’instance « englobante » dans les classes anonymes |
| **SUPER** | Référence à la class mère de l’objet courant   * Si pas d’héritage fait référence à la class Object * Peut être appelé dans une méthode et fait référence à l’objet parent à partir d’un autre constructeur * Première instruction dans un constructeur |
| **PACKAGE** | Indique un package auquel appartient l’objet   * Nom des packages en minuscule |
| **NATIVE** | Indique qu’une méthode est écrite de manière native dans un autre langage que Java et dans un autre fichier   * Pour compléter le code, il faut charger le fichier |

## Les types (9 mots)

8 types primitifs + void. Chaque type primitif est codé sur un nombre de bits qui lui est propre et qui définit son intervalle de valeur.

|  |  |  |  |
| --- | --- | --- | --- |
| Type | Correspondance | Taille (bits) | Intervalle de valeurs |
| **BOOLEAN** | Booléan | 1 | True/False   * Boolean peut recevoir le résultat d’une opération booléenne |
| **CHAR** | Caractère UNICODE | 16 non signé | 0 à 65536   * Spécifié à l’aide d’apostrophes ‘\_’ * Un caractère non présent sur le clavier peut être saisis à l’aide de sa valeur hexadécimale en unicode : ‘\u039A’ * Un char peut recevoir une valeur entière qui sera convertie (pas d’apostrophes) |
| **BYTE** | Entier | 8 signé | -128 à127 |
| **SHORT** | Entier | 16 signé | -32768 à 32767 |
| **INT** | Entier | 32 signé | -2147483648 à 2147483647   * Le signe \_ peut être utilisé pour facilité la lecture 21\_356\_564 |
| **LONG** | Entier | 64 signé | -9223372036854775808 à 9223372036854775807   * Le signe \_ peut être utilisé pour facilité la lecture |
| **FLOAT** | Décimal | 32 signé | -1.40239846E-45 à 3.40282347E38 |
| **DOUBLE** | Décimal | 64 signé | 4.9406564584124654E-324 à 1.797693134862316E308 |
| **VOID** | Indique qu’une méthode ne renvoie pas de valeur   * Ex : méthode main ou les setters | | |

## Les Etats (10 mots)

|  |  |
| --- | --- |
| **CONST** | Mot réservé mais non utilisé à l’heure actuelle |
| **FALSE** | Représente la valeur booléenne négative |
| **TRUE** | Représente la valeur booléenne positive |
| **NEW** | Opérateur permettant d’instancier un objet   * Possibilité d’instancier une class anonyme à l’aide d’une interface par exemple, sous réserve d’implémenter toutes les méthodes définies |
| **NULL** | Valeur spéciale indiquant qu’un objet ne référence rien   * La valeur par défaut d’un objet est null * On ne peut pas réaliser d’opération sur un objet null (tombe dans une exception) |
| **STATIC** | Indique qu’une variable, une méthode ou un bloc n’appartient pas à une instance particulière d’une class   * Utilisation classique : design pattern * Une méthode static n’accède qu’à des variables static * Peut être utilisé pour un bloc code |
| **STRICTFP** | Veut dire Strict floating point. Garantit la précision et les arrondis dans les calculs flottants pour assurer la portabilité   * Interprétation de la JVM |
| **TRANSIENT** | Indique qu’une variable ne peut être sérialisée |
| **FNAL** | Indique qu’un élément ne peut être modifié, ni étendu   * Si utilisé sur une class : elle n’aura ni fille, ni héritage * Si utilisé sur une méthode : les class filles ne pourront pas surcharger la méthode * Si utilisé sur une variable : devient une constante * Final permet d’accéder à une variable depuis une class anonyme * Une variable final d’est instanciée qu’une seule fois (y compris null) |
| **VOLATILE** | Oblige la JVM à rafraîchir son conteny chaque fois qu’elle est utilisée |

## Les modificateurs (3 mots)

Détermine la visibilité de tous les éléments.

|  |  |
| --- | --- |
| **PRIVATE** | Indique qu’un élément n’est accessible que dans la class où il est défini   * Les attributs d’une class sont souvent en private puis ajout de getters et setters |
| **PROTECTED** | Indique qu’un élément n’est accessible que dans la class où il est défini ou dans les class filles ou encore dans les class du même package   * Le plus souvent utilisé pour rendre la visibilité depuis les class filles |
| **PUBLIC** | Indique qu’un élément est accessible depuis partout |
| **Vide** | Visibilité par défaut à mi-chemin entre protected et public |

## Les boucles (5 mots)

|  |  |
| --- | --- |
| **FOR** | Crée une boucle en spécifiant une initialisation de variable, une condition d’arrêt et une règle d’incrémentation   * Les trois parties initialisation, condition d’arrêt et incrémentation peuvent être vides * Possibilité de boucler sur des éléments itérables avec une for-each |
| **WHILE** | Crée une boucle avec une condition d’arrêt   * La condition d’arrêt est évaluée avant chaque itération |
| **DO** | Crée une boucle dont la condition d’arrêt est évaluée après l’itération   * Utilisé avec un while |
| **CONTINUE** | Stoppe l’exécution d’une boucle et reprend l’itération suivante |
| **BREAK** | Mot réservé mais non utilisé |

## Les branchements (10 mots)

|  |  |
| --- | --- |
| **IF** | Exécute un bloc si la condition booléenne est satisfaite   * Permet de faire un test. On entre dans la boucle si la condition est remplie * Si plusieurs instructions, ouvrir des accolades |
| **ELSE** | Exécute un bloc quand la condition du IF n’est pas remplie   * Possibilité de tester plusieurs conditions en enchainant des if et des else |
| **SWITCH** | Exécute un bloc, désigné par un label case, en fonction d’une valeur   * L’exécution des blocs commence à partir du case correspondant. Les case suivants sont aussi exécutés, dans l’ordre d’apparition, sauf si une instruction break fait sortir du switch * SWITCH accepte des valeurs convertibles en int, comme un char * Possibilité d’utiliser un enum * Possibilité d’utiliser un string (mais déconseillé) |
| **CASE** | Définit un label cible dans un bloc switch   * La valeur de case est une constante |
| **DEFAULT** | Définit un label par défaut quand aucun label d’un bloc switch ne correspond pas |
| **BREAK** | Saut hors d’une boucle, hors d’une instruction (switch) ou vers un label   * Il est utilisé pour sortir facilement d’une boucle et éviter de nombreux tests imbriqués * Break ne fait sortir que d’un seul niveau de boucles si plusieurs boucles imbriquées * Break peut être utilisé en combinaison avec un label * Structure un switch |
| **ASSERT** | Vérifie qu’une condition est remplie, sinon lève une erreur avec le message spécifié   * Surtout utilisé pour du débogage * L’erreur lancée est une ASSERTIONERROR |
| **INSTANCEOF** | Test si une variable est une instance d’un objet spécifique   * INSTANCEOF prend en compte la hiérarchie de l’objet * Généralement dans utilisé dans un IF |
| **RETURN** | Stoppe l’exécution d’une méthode d’une méthode et renvoie la valeur facultative spécifiée   * Peut être utilisée plusieurs fois dans une méthode * Ne peut être utilisé dans une méthode void |
| **SYNCHRONIZED** | Indique qu’un seul thread à la fois peut accéder à une méthode ou à un bloc   * Souvent utilisé pour le design pattern (ex : ouverture d’une ressource) |

## Les exceptions (5 mots)

|  |  |
| --- | --- |
| **THROW** | Lève une exception   * Si une exception est levée, l’exécution de la méthode est arrêtée |
| **THROWS** | Indique qu’une méthode peut lever une exception   * Une méthode peut lever plusieurs exceptions distinctes dont les noms sont séparés par des virgules * Les exceptions de type « runtime » n’ont pas besoin d’être indiquées dans la définition de la méthode |
| **TRY** | Ouvre un bloc pour intercepter d’éventuelles exceptions   * Un try peut gérer la fermeture d’une ressource (Autoclosable) * Try est toujours associé à un catch et/ou un finally |
| **CATCH** | Ouvre un bloc permettant de gérer une exception survenue dans un try   * Il peut y avoir plusieurs catch d’affilé, mais un seul sera exécuté * On peut spécifier plusieurs exceptions dans un catch |
| **FINALLY** | Ouvre un bloc qui sera toujours exécuté après un try   * Sert généralement à fermer une ressource ouverte dans un try * S’il y a un catch, il sera exécuté avant le finally * Le bloc finally est toujours exécuté même si il y a un return dans le try. Le return est exécuté après l’exécution du finally |