100 Exercices Java – Du Débutant au Hardcore

Organisation :

Niveau 1–20 : débutant

Niveau 21–50 : intermédiaire

Niveau 51–80 : avancé

Niveau 81–100 : hardcore (structures, threads, sockets, algorithmes, IA, etc.)

1. Écrire un programme Java qui affiche « Bonjour, Monde! » à l'écran.

2. Déclarer des variables de type int, double, char et boolean et afficher leurs valeurs.

3. Demander à l'utilisateur son nom et l'afficher avec un message de bienvenue.

4. Écrire un programme qui calcule la somme de deux nombres saisis par l'utilisateur.

5. Écrire un programme qui calcule la moyenne de trois nombres.

6. Écrire un programme qui convertit des degrés Celsius en Fahrenheit.

7. Écrire un programme qui affiche les nombres de 1 à 10 en utilisant une boucle for.

8. Écrire un programme qui calcule la factorielle d'un petit entier ($n \leq 10$) avec une boucle.

9. Écrire un programme qui affiche les nombres pairs entre 1 et 50.

10. Écrire un programme qui teste si un nombre donné est pair ou impair.

11. Écrire un programme qui détermine si un entier est positif, négatif ou zéro.

12. Écrire un programme qui trouve le plus grand de trois nombres donnés.

13. Écrire un programme qui inverse une chaîne de caractères fournie par l'utilisateur.

14. Écrire un programme qui compte le nombre de voyelles dans une chaîne donnée.

15. Écrire un programme qui remplace tous les espaces d'une chaîne par des underscores (_).

16. Écrire un programme qui calcule la somme des éléments d'un tableau d'entiers.

17. Écrire un programme qui recherche un élément dans un tableau (recherche linéaire).

18. Écrire un programme qui trie un petit tableau d'entiers avec l'algorithme de tri par insertion.

19. Écrire un programme qui utilise une méthode pour calculer le carré d'un nombre.

20. Écrire un programme qui montre l'utilisation d'une classe simple avec un constructeur et une méthode toString().

21. Écrire une classe ComplexNumber (réel, imaginaire) avec opérations add, sub, mul, div et tests unitaires simples.

22. Implémenter une pile (Stack) à l'aide d'un tableau et fournir push, pop, peek et isEmpty.

23. Implémenter une file (Queue) avec une LinkedList personnalisée (sans utiliser java.util).

24. Écrire un programme qui lit un fichier texte et compte le nombre de mots.

25. Écrire un programme qui calcule le PGCD (gcd) de deux entiers avec l'algorithme d'Euclide.

26. Écrire un programme qui génère les n premiers nombres de Fibonacci (itératif et récursif) et comparer les performances.

27. Écrire une méthode qui vérifie si une chaîne est un palindrome (ignorer espaces et casse).

28. Écrire une méthode qui supprime les doublons d'une liste d'entiers (retourner une nouvelle liste).

29. Implémenter le tri rapide (quicksort) récursif sur un tableau d'entiers.

30. Implémenter la recherche binaire sur un tableau trié et gérer les cas d'erreur.

31. Écrire une application console qui gère une petite liste de contacts (CRUD) en mémoire.

32. Écrire une fonction pour fusionner deux tableaux triés en un seul tableau trié (merge).

33. Écrire un programme qui convertit un nombre décimal en binaire, hexadécimal et octal.

34. Écrire un parseur simple qui valide si une expression arithmétique (digits +/-/*/) est bien formée.

35. Écrire une classe Money avec validation de devise et opérations d'addition/soustraction en tenant compte des centimes.

36. Utiliser les Collections Java : Map pour compter la fréquence des mots dans un texte.

37. Écrire un programme qui utilise les API de dates de Java (java.time) pour calculer l'âge à partir d'une date de naissance.

38. Écrire une méthode qui effectue une rotation circulaire d'un tableau vers la droite de k positions.

39. Écrire un programme qui simule un lancer de dés n fois et calcule les probabilités empiriques.

40. Écrire une simple application qui utilise les exceptions personnalisées pour valider l'entrée utilisateur.

41. Écrire un programme qui transforme une liste d'objets en JSON et lit un JSON pour recréer les objets (utiliser une bibliothèque comme Gson).

42. Écrire une classe générique Pair et montrer son usage.

43. Écrire un programme qui calcule la distance de Levenshtein entre deux chaînes (édition).

44. Écrire une méthode qui vérifie si deux chaînes sont des anagrammes l'une de l'autre.

45. Écrire un programme qui simule un jeu de pile ou face et affiche une statistique des séries de succès.

46. Écrire une méthode pour évaluer la notation postfixée (Reverse Polish Notation).

47. Écrire un programme qui sérialise et désérialise un objet Java (Serializable).

48. Écrire un petit benchmark pour comparer ArrayList et LinkedList pour insertions en début/milieu/fin.

49. Écrire une méthode utilitaire pour diviser un tableau en chunks (sous-tableaux) de taille k.

50. Écrire une API REST minimale avec Spark Java ou un micro-framework similaire (endpoints CRUD simples).

51. Implémenter un arbre binaire de recherche (BST) avec insert, delete, search, et parcours inorder/preorder/postorder.

52. Écrire une méthode pour équilibrer un arbre (conversion en AVL ou red-black simple).

53. Implémenter un graphe avec listes d'adjacence et effectuer un parcours BFS et DFS.

54. Implémenter l'algorithme de Dijkstra pour plus court chemin sur un graphe pondéré.

55. Implémenter l'algorithme de Kruskal ou Prim pour trouver l'arbre couvrant minimal (MST).

56. Écrire une solution pour le problème du sac à dos (knapsack) — version dynamique (DP) et version gloutonne.

57. Implémenter une table de hachage (HashMap) simplifiée avec gestion des collisions (chaînage).

58. Écrire un pool de threads simple et exécuter des tâches en parallèle (Executor-like).

59. Écrire un programme concurrent sûr qui utilise synchronized, wait/notify pour un producteur-consommateur.

60. Implémenter une structure de données LRU Cache avec complexité O(1) pour get/put (Hash + double linked list).

61. Écrire un programme qui utilise les flux (Streams) Java 8 pour traiter un dataset et produire des statistiques.

62. Écrire un parser récursif-descendant simple pour une mini-langue arithmétique avec priorité d'opérateurs.

63. Écrire une application qui interagit avec une base de données SQLite via JDBC (CRUD + requêtes préparées).

64. Implémenter un algorithme de backtracking pour résoudre le Sudoku.

65. Écrire une classe qui implémente l'interface Comparable et fournir différents comparators.

66. Implémenter l'algorithme A* pour la recherche de chemin avec heuristique admissible.

67. Écrire un outil qui calcule la similarité cosine entre vecteurs de textes (TF-IDF simplifié).

68. Implémenter un mini interpréteur de commandes (shell) capable d'exécuter commandes système et redirections.

69. Écrire une implémentation de mergesort parallèle (utiliser ForkJoinPool).

70. Écrire des tests unitaires complets pour une librairie donnée en utilisant JUnit et Mockito.

71. Écrire une application qui consomme une API REST externe et stocke les résultats dans une base locale.

72. Écrire une bibliothèque utilitaire pour gestion d'événements (EventEmitter pattern) avec listeners asynchrones.

73. Écrire un sérialiseur binaire personnalisé pour objets simples et sa désérialisation.

74. Écrire une application qui crypte/décrypte des fichiers avec AES en mode GCM et gestion d'IV.

75. Écrire un programme qui profile mémoire et détecte fuites simples (simulation et diagnostic).

76. Implémenter un algorithme d'ordonnancement (scheduling) simple pour jobs avec priorités.

77. Écrire un compilateur très simple qui convertit des expressions en bytecode fictif et exécute ce bytecode.

78. Écrire une application Swing ou JavaFX basique pour visualiser un graphe et effectuer zoom/pan.

79. Écrire un plugin Maven simple qui exécute une tâche personnalisée pendant le build.

81. Écrire un serveur TCP multithreadé capable de gérer plusieurs clients (chat simple) et assurer la synchronisation des ressources.

82. Implémenter une version thread-safe d'un compteur distribué en utilisant sockets et protocoles simples.

83. Écrire un serveur HTTP minimal conforme (parsing requêtes, headers, réponses statiques).

84. Écrire un client/serveur pour transfert de fichiers avec reprise (resume) et vérification d'intégrité (checksum).

85. Implémenter une mémoire partagée entre processus Java via mapped files (FileChannel) et synchronisation.

86. Écrire un système de logging distribué simple où plusieurs processus envoient des logs à un collector via UDP/TCP.

87. Écrire un mini moteur de règles qui évalue des règles conditionnelles sur des objets en entrée (pattern matching).

88. Implémenter un algorithme de compression simple (LZW ou Huffman) et tester la performance sur fichiers texte.

89. Écrire un programme qui construit un index inversé et permet des recherches booléennes sur un corpus (mini-moteur de recherche).

90. Implémenter une version simplifiée de MapReduce pour compter mots sur plusieurs fichiers en parallèle.

91. Écrire un réseau neuronal feedforward simple (backpropagation) pour résoudre un problème de classification binaire (sans bibliothèques ML).

92. Implémenter un algorithme d'optimisation (gradient descent) pour ajuster les poids d'un modèle simple.

93. Écrire une application qui utilise JNI pour appeler du code natif (C) et échanger des données simples.

94. Implémenter un système basique d'authentification JWT pour une API Java sécurisée.

95. Écrire un programme qui détecte et corrige dynamiquement les deadlocks dans un ensemble de threads (détection + restitution simple).

96. Implémenter une base de données clé-valeur en mémoire avec snapshotting et persistence sur disque.

97. Écrire un compilateur JIT très simple qui génère du bytecode pour une expression et l'exécute via ClassLoader dynamique.

98. Implémenter un algorithme de clustering (k-means) et visualiser les clusters (export CSV pour plotting).

99. Écrire un agent qui fait du web-scraping distribué et respecte robots.txt, gère proxy et politeness delays.

100. Résoudre un problème complexe d'algorithme : implémenter l'algorithme de suffix array + LCP et l'utiliser pour recherche de motifs efficace.

Fin des 100 exercices. Bon coding !

Si tu veux les solutions ou une version PDF avec corrections détaillées, demande « version corrigée ».