# Lista de Questões — Collections e Streams em Java

Do nível básico ao avançado. Tente resolver antes de olhar as respostas no final.

1. Explique a diferença entre List, Set e Map no Java Collections Framework.

2. Qual a diferença entre ArrayList e LinkedList? Quando usar cada uma?

3. Implemente um código que insira 5 strings em um ArrayList e itere imprimindo-as.

4. Qual a diferença entre HashSet e TreeSet?

5. O que acontece se uma classe usada como chave em HashMap não sobrescreve equals e hashCode corretamente?

6. Implemente um HashMap que associe nomes de alunos às suas idades e itere imprimindo cada par.

7. Qual é a complexidade média de busca em HashMap?

8. Explique a diferença entre HashMap e Hashtable.

9. O que é um Iterator e como usá-lo em uma Collection?

10. Qual diferença entre fail-fast e fail-safe iterators?

11. Implemente um código que remova elementos pares de um ArrayList de inteiros usando removeIf.

12. Qual a diferença entre LinkedHashMap e HashMap? Dê exemplo de uso.

13. Implemente uma fila com PriorityQueue que priorize números menores.

14. Explique as diferenças entre ArrayDeque e Stack.

15. O que é um ConcurrentHashMap e quando usá-lo?

16. Mostre com código como agrupar uma lista de strings pelo seu tamanho usando Streams.

17. Explique o funcionamento de distinct() em Streams.

18. Qual a diferença entre map e flatMap em Streams?

19. Mostre como converter uma lista de objetos em um Map usando toMap collector.

20. Qual a diferença entre findFirst e findAny?

21. Implemente uma redução com reduce() para somar os elementos de uma lista de inteiros.

22. Explique a diferença entre operações intermediárias e terminais em Streams.

23. O que são Collectors.groupingBy e partitioningBy? Mostre exemplos.

24. Como funciona o Collector.collectingAndThen?

25. Escreva um código que conte palavras em uma lista usando Collectors.groupingBy.

26. Explique a diferença entre parallelStream e stream.

27. Implemente uma soma em paralelo de 1 a 1.000.000 usando IntStream.

28. O que é um Spliterator e qual sua importância em streams paralelos?

29. Qual diferença entre Optional.orElse, orElseGet e orElseThrow?

30. Escreva um Collector customizado que some números.

31. Qual a diferença entre CopyOnWriteArrayList e ArrayList?

32. Explique o funcionamento de WeakHashMap e um caso de uso.

33. Mostre como implementar um cache simples com LinkedHashMap e LRU.

34. Implemente uma ordenação customizada em TreeMap usando Comparator.

35. Explique as vantagens de usar EnumMap.

36. Escreva um código que calcule estatísticas (min, max, média) de uma lista de inteiros usando summaryStatistics.

37. Mostre como paralelizar processamento com ForkJoinPool e parallelStream.

38. Qual armadilha comum ao usar parallelStream com I/O?

39. Implemente uma pipeline que filtre, transforme e colete resultados em lista imutável.

40. Explique o uso de Collectors.mapping e Collectors.flatMapping.

41. Mostre como combinar múltiplos Collectors com teeing (Java 12+).

42. Qual diferença entre Collectors.toUnmodifiableList e Collectors.toList?

43. Mostre um exemplo de redução mutável customizada.

44. Como funciona a característica IDENTITY_FINISH em Collector?

45. Escreva código que use computeIfAbsent para agrupar itens por chave.

46. Qual diferença entre merge() e compute() em Map?

47. Mostre como usar replaceAll em Map para transformar valores.

48. Escreva código que faça flatMap em um Stream de listas de inteiros.

49. Como implementar uma ordenação estável em Streams?

50. Mostre um exemplo de uso de partitioningBy para separar pares e ímpares.

# Respostas — gabarito resumido

1. List permite duplicatas e ordem, Set não permite duplicatas, Map é chave→valor.

2. ArrayList: acesso rápido, ruim para inserções; LinkedList: bom para inserções, acesso lento.

3. new ArrayList<>(List.of("a","b","c")); forEach(System.out::println).

4. HashSet: não ordena, TreeSet: ordena.

5. O comportamento do mapa quebra, entradas se perdem.

6. for(Map.Entry e: map.entrySet())...

7. O(1) em média.

8. Hashtable é sincronizado, HashMap não.

9. Iterator permite percorrer removendo com segurança.

10. Fail-fast lança ConcurrentModificationException, fail-safe não.

11. list.removeIf(n -> n%2==0);

12. LinkedHashMap mantém ordem de inserção/acesso.

13. PriorityQueue pq = new PriorityQueue<>();

14. ArrayDeque é mais eficiente, Stack é legada.

15. Map concorrente e escalável.

16. stream.collect(groupingBy(String::length));

17. distinct usa equals().

18. map transforma, flatMap achata.

19. stream.collect(toMap(o->o.id, o->o));

20. findFirst garante ordem, findAny pode pegar qualquer em paralelo.

21. stream.reduce(Integer::sum).get();

22. Intermediárias são lazy, terminais disparam execução.

23. groupingBy agrupa, partitioningBy divide boolean.

24. Permite pós-processamento do collector.

25. stream.collect(groupingBy(w->w, counting()));

26. parallelStream usa ForkJoinPool.

27. IntStream.rangeClosed(1,1_000_000).parallel().sum();

28. Spliterator divide fonte em partes.

29. orElse sempre avalia, orElseGet é lazy, orElseThrow lança exceção.

30. Collector.of(...).

31. CopyOnWriteArrayList é thread-safe, cópia a cada modificação.

32. WeakHashMap remove entradas quando chave sem referência.

33. LRU: sobrescreve removeEldestEntry em LinkedHashMap.

34. TreeMap<>(Comparator.reverseOrder());

35. EnumMap é rápido e eficiente para enums.

36. list.stream().mapToInt(...).summaryStatistics();

37. ForkJoinPool.submit(...).

38. Pode travar threads.

39. stream.filter(...).map(...).collect(toUnmodifiableList());

40. mapping aplica transformação dentro de collector.

41. teeing combina dois collectors.

42. toUnmodifiableList retorna lista imutável.

43. Collector customizado com accumulator e combiner.

44. IDENTITY_FINISH evita função final.

45. map.computeIfAbsent(k, v->new ArrayList<>()).add(...);

46. merge combina valores, compute recalcula geral.

47. map.replaceAll((k,v)->v*2);

48. stream.flatMap(List::stream);

49. sorted com Comparator é estável.

50. partitioningBy(n->n%2==0).