104 Java 8 interview questions to ask your applicants

Questions

1. What new features did Java 8 bring to the table, and which one excites you the most?

2. Can you explain what a lambda expression is in simple terms, and provide a basic example?

3. What are functional interfaces, and how are they related to lambda expressions?

4. What is the purpose of the Stream API in Java 8, and how does it make working with collections easier?

5. Explain the difference between forEach and stream().forEach().

6. What are the benefits of using the Stream API, and can you give a practical scenario where you would use it?

7. How can you sort a list of objects using lambda expressions and the Stream API?

8. What is the purpose of Optional in Java 8, and how does it help avoid NullPointerExceptions?

9. Can you explain method references in Java 8, and provide an example of how they simplify lambda expressions?

10. What is the difference between map and flatMap operations in the Stream API?

11. How do you group elements of a collection using the Stream API?

12. What are the parallel streams in Java 8, and when would you use them?

13. Explain the differences between Collection and Stream interfaces.

14. How can you filter elements from a collection using the Stream API?

15. What are the advantages of using immutable collections in Java 8?

16. What are the key differences between the filter and map operations in Java 8 streams?

17. How do you convert a Stream back into a List or Set?

18. Can you explain the concept of 'lazy evaluation' in the context of Java 8 Streams?

19. How does Java 8's LocalDate and LocalDateTime improve upon the old Date class?

20. Explain what default methods are in interfaces and why they were introduced in Java 8.

21. How can you use the reduce operation in the Stream API to find the sum of numbers in a list?

22. What is the significance of the @FunctionalInterface annotation?

23. In what ways did Java 8 improve the handling of date and time, and why was it needed?

24. Describe a scenario where you would use a lambda expression instead of an anonymous class.

25. How would you find the maximum value in a list of integers using streams?

26. Explain how to use the Collectors.groupingBy method in Java 8.

27. What are the potential drawbacks or considerations when using parallel streams?

28. Can you give an example of using a method reference to call a constructor?

29. Explain the difference between intermediate and terminal operations in Java 8 streams. Give examples.

30. What is the difference between Collection and Stream in Java 8?

31. Can you explain what Lambda Expressions are and give a simple example?

32. What are functional interfaces? Can you name a few built-in functional interfaces in Java 8?

33. How do you use the forEach method with Lambda Expressions to iterate over a list?

34. What is the purpose of the Optional class? How can you avoid NullPointerExceptions using it?

35. What are the key advantages of using streams in Java 8?

36. Describe the difference between intermediate and terminal operations in Java Streams. Give examples of each.

37. How would you use the map operation in a Stream to transform a list of strings to uppercase?

38. Explain how you can use the filter operation in a Stream to select only even numbers from a list of integers?

39. What is method referencing in Java 8 and when would you use it?

40. How do you sort a list of objects using Lambda Expressions in Java 8?

41. Explain the difference between Predicate, Function, Consumer, and Supplier functional interfaces.

42. How can you convert a Collection to a Stream in Java 8?

43. What are the limitations of using parallel streams? When is it not a good idea to use them?

44. Explain how to use the reduce operation to find the sum of all elements in a stream.

45. How can you group elements of a stream based on a certain criteria using groupingBy?

46. What is the purpose of the flatMap operation in Java 8 streams?

47. Explain what is the difference between findFirst and findAny methods in Java Streams.

48. How do you handle exceptions inside a Lambda Expression?

49. What is the purpose of default methods in interfaces in Java 8? Give an example.

50. How can you create a stream from an array in Java 8?

51. Explain how to use the peek operation in a Stream for debugging purposes.

52. What is the difference between anyMatch, allMatch, and noneMatch in Java Streams?

53. How can you use multiple filter operations in a single Stream pipeline?

54. Explain how you can use Optional to handle situations where a stream might return no elements.

55. What is the purpose of Collectors in Java 8 streams? Give some examples of common collectors.

56. How would you find the maximum or minimum value in a stream of numbers using Java 8?

57. Describe a scenario where you would use a parallel stream to improve performance.

58. Explain how the distinct operation works in a Java 8 stream.

59. How does the flatMap operation in Java 8 Streams differ from map, and when would you choose one over the other? Explain with a simple use case.

60. Explain the concept of 'lazy evaluation' in the context of Java 8 Streams. Why is it beneficial, and how does it impact performance?

61. Describe the purpose of the Optional class in Java 8. How does it help prevent NullPointerExceptions, and what are some best practices for using it?

62. Can you explain how method references (::) work in Java 8, and provide examples of different types of method references (static, instance, constructor)?

63. What are the key differences between Collection.stream() and Collection.parallelStream() in Java 8? When would you prefer one over the other, and what are the potential pitfalls of using parallelStream()?

64. Explain the use of Collectors.groupingBy() in Java 8 Streams. Provide an example of how you would group a list of objects based on a specific property.

65. How does Java 8's CompletableFuture class simplify asynchronous programming compared to traditional Future objects? Explain with an example.

66. Describe how you would use Java 8's LocalDate, LocalTime, and LocalDateTime classes for handling dates and times. What are the advantages of using these classes over the older java.util.Date class?

67. Explain the concept of 'functional interfaces' in Java 8. What is the @FunctionalInterface annotation, and why is it used?

68. How can you create custom functional interfaces in Java 8, and what are some practical use cases for them?

69. Describe the differences between forEach and forEachOrdered methods in Java 8 Streams. When should you use one over the other?

70. How does the reduce operation in Java 8 Streams work, and how can you use it to perform calculations on a stream of elements? Give an example.

71. Explain how to use Collectors.toMap() to convert a Java 8 Stream into a Map. What are the potential issues with duplicate keys, and how can you handle them?

72. How can you use Java 8 Streams to efficiently process large files, and what are some techniques for optimizing performance?

73. Describe the purpose of the peek operation in Java 8 Streams. How can it be useful for debugging or logging stream operations?

74. Explain the use of Collectors.partitioningBy() in Java 8 Streams. How does it differ from groupingBy() and when is it useful?

75. How can you chain multiple CompletableFuture instances together to create complex asynchronous workflows in Java 8?

76. Describe how to use Java 8's Duration and Period classes for measuring time intervals and date differences. Provide examples.

77. Explain how to use default methods in interfaces in Java 8. What problem do they solve, and what are some potential conflicts that can arise when using them?

78. How do you resolve conflicts when a class inherits multiple default methods with the same signature from different interfaces in Java 8?

79. How would you design a system using Java 8 features to process large datasets in parallel, ensuring optimal CPU utilization and minimal memory footprint?

80. Explain how you would use the 'CompletableFuture' API to handle asynchronous operations and combine the results of multiple independent services in a reactive application?

81. Describe a scenario where using 'StampedLock' would be more advantageous than 'ReentrantReadWriteLock', and explain the potential risks associated with using 'StampedLock' incorrectly.

82. How can you leverage Java 8's functional interfaces and lambda expressions to implement a custom retry mechanism for handling transient failures in a distributed system?

83. Explain how you would use the Stream API to efficiently process a collection of objects and group them based on multiple complex criteria, while also applying aggregations to each group.

84. Describe how you can use Optional to avoid NullPointerExceptions in a legacy codebase and what are the potential drawbacks of overusing Optional.

85. How do you use the new Date and Time API to handle different time zones and daylight saving time in a global application?

86. Explain the difference between map() and flatMap() operations in Java 8 streams and provide a use case for each.

87. Describe how you would implement a custom collector to perform a specific aggregation operation that is not provided by the standard collectors.

88. How can you use method references to make your code more readable and maintainable, and when should you avoid using them?

89. Explain how you would use the Stream API to perform complex data transformations and aggregations in a parallel and efficient manner.

90. Describe a scenario where you would use the 'computeIfAbsent' method of the ConcurrentHashMap and explain the potential performance implications.

91. How can you use the Java 8's 'Files' API to efficiently read and process large text files, and what are the best practices for handling character encoding?

92. Explain the difference between the 'peek' and 'forEach' methods in Java 8 streams and provide a use case for each.

93. Describe how you would implement a custom spliterator to efficiently process a large dataset in parallel using the Stream API.

94. How can you use the Java 8's Nashorn engine to execute JavaScript code within a Java application, and what are the potential security risks?

95. Explain how you would use the 'reduce' method of the Stream API to perform complex aggregations on a collection of objects.

96. Describe a scenario where you would use the 'merge' method of the ConcurrentHashMap and explain the potential concurrency issues.

97. How can you use the Java 8's Base64 API to encode and decode binary data, and what are the different encoding options available?

98. Explain how you would use the 'anyMatch', 'allMatch', and 'noneMatch' methods in Java 8 streams to perform complex data validation.

99. Describe how you would implement a custom stream pipeline to perform a specific data processing task that is not easily achieved with the standard stream operations.

100. What are the advantages and disadvantages of using lambda expressions versus anonymous classes?

101. How does the '::' operator work, and can you give examples of when it is best used?