Here are 100+ questions and answers related to Java 8 features, without any website content:

1. What are the major new features introduced in Java 8?

Key new features in Java 8 include:

- Lambda expressions
- Stream API
- Default methods in interfaces
- Functional interfaces
- Method references
- Optional class
- New Date and Time API
- Nashorn JavaScript engine
- Base64 encoding/decoding
- Parallel array sorting
- Annotations on Java types

2. What are lambda expressions in Java 8?

Lambda expressions are anonymous functions that can be passed as arguments or returned from method calls. They enable functional programming in Java. The syntax is:

(parameters) -> expression

or

(parameters) -> { statements; }

3. What is the Stream API in Java 8?

The Stream API provides a functional approach to processing collections of objects. It allows operations like filtering, mapping, reducing etc. to be performed on collections in a declarative way.

4. What are default methods in interfaces?

Default methods allow interfaces to have method implementations. This enables adding new methods to interfaces without breaking existing implementations.

5. What is a functional interface?

A functional interface is an interface that contains exactly one abstract method. It can be used as the assignment target for a lambda expression.

6. What are method references in Java 8?

Method references provide a way to refer to methods or constructors without executing them. They can be used as a more compact alternative to certain lambda expressions.

7. What is the Optional class used for?

The Optional class is a container object which may or may not contain a non-null value. It is used to represent null with absent values and avoid null pointer exceptions.

8. What are the key classes in the new Date and Time API?

Key classes include:

- LocalDate
- LocalTime
- LocalDateTime
- ZonedDateTime
- Period
- Duration

9. What is the Nashorn JavaScript engine?

Nashorn is a JavaScript engine developed by Oracle for the Java platform. It allows embedding JavaScript in Java applications.

10. What are the advantages of the new Date and Time API?

- Immutable classes
- Clear separation between human readable time and machine time
- Better handling of time zones
- More comprehensive API for date manipulations

11. What is the purpose of the @FunctionalInterface annotation?

It is used to indicate that an interface is intended to be a functional interface. The compiler will generate an error if the interface does not have exactly one abstract method.

12. How do you define a lambda expression in Java 8?

The basic syntax is:

```
(parameters) -> expression
```

For example:

```
(int x, int y) -> x + y
```

13. What are the benefits of using lambda expressions?

- More concise code
- Enables functional programming
- Improves readability
- Allows passing behavior as method arguments
- Enables parallel processing of collections

14. What is the difference between intermediate and terminal operations in streams?

Intermediate operations return a new stream and are lazy. Terminal operations trigger the processing of the stream and produce a result.

15. Give some examples of intermediate stream operations.
- filter()
- map()
- flatMap()
- distinct()
- sorted()
- peek()
- limit()
- skip()

16. Give some examples of terminal stream operations.
- forEach()
- toArray()
- reduce()
- collect()
- min()
- max()
- count()
- anyMatch()
- allMatch()
- noneMatch()
- findFirst()
- findAny()

17. What is method reference in Java 8?
Method reference is a shorthand notation of a lambda expression to call a method. It is denoted by :: symbol.

18. What are the types of method references in Java 8?
There are four types:

- Reference to a static method
- Reference to an instance method of a particular object
- Reference to an instance method of an arbitrary object of a particular type
- Reference to a constructor

19. What is the purpose of the default keyword in interface methods?
The default keyword allows adding new methods to interfaces without breaking existing implementations. It provides a default implementation for the method.
20. Can a functional interface extend another interface?
Yes, a functional interface can extend another interface as long as it doesn't add any new abstract methods.

21. What is the purpose of the java.util.function package?

It provides a set of functional interfaces that can be used as assignment targets for lambda expressions and method references.

22. What are some common functional interfaces in the java.util.function package?

- Predicate`<T>`
- Function<T,R>
- Consumer`<T>`
- Supplier`<T>`
- UnaryOperator`<T>`
- BinaryOperator`<T>`

23. What is the purpose of the Optional class?

Optional is a container object used to represent nullable values and avoid null pointer exceptions. It forces developers to explicitly handle the case when a value is not present.

24. How do you create an Optional object?

You can create an Optional using:

- Optional.empty()
- Optional.of(value)
- Optional.ofNullable(value)

25. What methods does Optional provide to retrieve the value?

- get()
- orElse(other)
- orElseGet(supplier)
- orElseThrow(exceptionSupplier)

26. What is the difference between Collection and Stream?

Collections are data structures that hold elements, while Streams are a view of data that allows performing operations on it. Streams don't store data and can't be reused.

27. What is the forEach() method in Iterable interface?

The forEach() method performs an action for each element of the Iterable. It accepts a Consumer functional interface as an argument.

28. What is the purpose of the peek() method in streams?

The peek() method is used to perform an action on each element of the stream without modifying it. It's mainly used for debugging.

29. What is the difference between map() and flatMap() methods in Stream?

map() transforms each element of the stream, while flatMap() transforms each element of the stream into another stream and then flattens all the streams into one.

30. What is a parallel stream?

A parallel stream is capable of running operations in parallel by splitting the data into multiple chunks and processing them concurrently.

31. How do you create a parallel stream?

You can create a parallel stream using:

- parallelStream() method on a collection
- parallel() method on an existing stream

32. What is the purpose of the Collector interface in Stream API?

The Collector interface is used to combine the results of processing elements of a stream. It encapsulates the functions used as arguments in the collect() method of streams.

33. What are some built-in collectors in the Collectors class?

- toList()
- toSet()
- toMap()
- joining()
- counting()
- summarizingDouble/Long/Int()
- groupingBy()
- partitioningBy()

34. What is the purpose of the reduce() method in streams?

The reduce() method is used to reduce the elements of a stream to a single value. It takes a BinaryOperator as an argument.

35. What is the difference between reduce() and collect() methods?

reduce() combines stream elements into a single result, while collect() can produce any result, including collections.

36. What is the purpose of the Spliterator interface?

Spliterator is used for traversing and partitioning elements of a source. It's designed to support parallel processing.

37. What are the new features in Java 8 for working with Arrays?

- Parallel sort methods
- New methods for creating streams from arrays
- New methods for comparing arrays

38. What is the purpose of the StringJoiner class?

StringJoiner is used to construct a sequence of characters separated by a delimiter. It can also add a prefix and suffix to the result.

39. What is the purpose of the Objects class?

The Objects class provides static utility methods for operating on objects. Many of these methods are null-safe.

40. What are the new features in Java 8 for working with Maps?

- New methods like forEach(), replaceAll(), compute(), merge()
- Support for parallel operations
- Performance improvements for the HashMap class

41. What is a default method in an interface?

A default method is a method with an implementation defined in an interface. It allows adding new methods to interfaces without breaking existing implementations.

42. Can a class implement two interfaces with the same default method?

Yes, but the class must override the default method to specify which one to use or provide its own implementation.

43. What is a static method in an interface?

Java 8 allows defining static methods in interfaces. These methods belong to the interface itself, not to the implementing classes.

44. What is the diamond problem in multiple inheritance?

The diamond problem occurs when a class inherits from two classes that have a common ancestor, potentially leading to ambiguity.

45. How does Java 8 solve the diamond problem?

Java 8 provides rules for resolving conflicts in multiple inheritance scenarios with default methods. The class must override the conflicting method.

46. What is the @FunctionalInterface annotation used for?

It's used to indicate that an interface is intended to be a functional interface. The compiler will generate an error if the interface doesn't meet the requirements of a functional interface.

47. What is the purpose of the java.time package?

The java.time package provides classes for date, time, duration, and time zones. It's a replacement for the older java.util.Date and java.util.Calendar classes.

48. What is the LocalDate class?

LocalDate represents a date without a time or time zone in the ISO-8601 calendar system, such as 2007-12-03.

49. What is the LocalTime class?

LocalTime represents a time without a date or time zone in the ISO-8601 calendar system, such as 10:15:30.

50. What is the LocalDateTime class?

LocalDateTime represents a combination of date and time without a time zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30.

51. What is the ZonedDateTime class?

ZonedDateTime represents a date-time with a time zone in the ISO-8601 calendar system, such as 2007-12-03T10:15:30+01:00 Europe/Paris.

52. What is the Period class?

Period represents a date-based amount of time in the ISO-8601 calendar system, such as '2 years, 3 months and 4 days'.

53. What is the Duration class?

Duration represents a time-based amount of time, such as '34.5 seconds'.

54. What is the Instant class?

Instant represents a point in time on the timeline, typically used to record event timestamps in applications.

55. How do you parse a string into a LocalDate?

You can use LocalDate.parse(String) method. For example:
LocalDate date = LocalDate.parse("2007-12-03");

56. How do you format a LocalDateTime into a string?

You can use the DateTimeFormatter class. For example:
String formatted = localDateTime.format(DateTimeFormatter.ISO_LOCAL_DATE_TIME);

57. What is the purpose of the TemporalAdjusters class?

TemporalAdjusters provides a set of standard adjustments for dates, such as finding the first or last day of the month, the next Tuesday, etc.

58. What is the Clock class used for?

The Clock class provides access to the current instant, date and time using a time-zone. It's mainly used for testing or when you need more control over the source of time.

59. What is the MonthDay class?

MonthDay represents a combination of month and day-of-month, such as '--12-03' for "3rd December".

60. What is the YearMonth class?

YearMonth represents a combination of year and month, such as '2007-12' for "December 2007".

61. What is the difference between Instant and LocalDateTime?

Instant represents a point on the time-line in UTC, while LocalDateTime is a date-time without a time zone.

62. How do you convert between the old and new date-time APIs?

You can use methods like toInstant(), toLocalDateTime(), etc., on the old date-time classes, and fromInstant(), from(), etc., on the new date-time classes.

63. What is the purpose of the ChronoUnit enum?

ChronoUnit is an enum of date periods and time units. It provides a standard set of units for measuring time, like DAYS, HOURS, MINUTES, etc.

64. What is the purpose of the CompletableFuture class?

CompletableFuture is used for asynchronous programming. It can be explicitly completed by calling its complete() method, and can be used as a Future, or as a CompletionStage.

65. What are the advantages of CompletableFuture over Future?

- Can be explicitly completed
- Supports functional style operations
- Supports chaining of computations
- Provides better exception handling

66. What is the purpose of the thenApply() method in CompletableFuture?

thenApply() is used to process and transform the result of a CompletableFuture when it completes normally.

67. What is the difference between thenApply() and thenCompose() in CompletableFuture?

thenApply() is used when the transformation returns a simple value, while thenCompose() is used when the transformation returns another CompletableFuture.

68. What is the purpose of the allOf() method in CompletableFuture?

allOf() is used to combine multiple CompletableFutures. It returns a new CompletableFuture that completes when all of the given CompletableFutures complete.

69. What is the purpose of the anyOf() method in CompletableFuture?

anyOf() is used to combine multiple CompletableFutures. It returns a new CompletableFuture that completes when any of the given CompletableFutures complete.

70. What is a Nashorn engine?

Nashorn is a JavaScript engine developed by Oracle for the Java platform. It allows you to embed JavaScript in Java applications.

71. How do you execute JavaScript code using Nashorn?

You can use the ScriptEngineManager and ScriptEngine classes:

```
ScriptEngine engine = new ScriptEngineManager().getEngineByName("nashorn");
engine.eval("print('Hello, World!');");
```

72. What is the jjs command?

jjs is a command-line tool introduced in Java 8 that allows you to execute JavaScript code directly from the command line using Nashorn.

73. What is the purpose of the Base64 class?

The Base64 class provides static methods for encoding and decoding using the Base64 encoding scheme as specified in RFC 4648.

74. How do you encode a string to Base64?

You can use the Base64.getEncoder().encodeToString() method:

```
String encoded = Base64.getEncoder().encodeToString("Hello, World!".getBytes());
```

75. How do you decode a Base64 encoded string?

You can use the Base64.getDecoder().decode() method:

```
byte[] decoded = Base64.getDecoder().decode(encoded);
String decodedString = new String(decoded);
```

76. What is the purpose of the StampedLock class?

StampedLock is a capability-based lock with three modes for controlling read/write access. It's designed as an alternative to ReadWriteLock for certain use cases.

77. What are the advantages of StampedLock over ReadWriteLock?

- Supports optimistic read locking
- Can achieve higher throughput in certain scenarios
- Allows downgrading from write lock to read lock

78. What is the purpose of the LongAdder class?

LongAdder is used for concurrent addition operations. It's designed to reduce contention when multiple threads are updating a common sum.

79. What are the advantages of LongAdder over AtomicLong?

LongAdder can provide higher throughput under high contention, at the cost of higher space consumption.

80. What is the purpose of the LongAccumulator class?

LongAccumulator is a more generalized version of LongAdder. It allows you to perform any associative operation, not just addition.

81. What is the purpose of the ThreadLocalRandom class?

ThreadLocalRandom is used for generating random numbers. It's designed to be more efficient in concurrent environments than using shared Random objects.

82. What is the purpose of the Arrays.parallelSort() method?

Arrays.parallelSort() is used to sort arrays in parallel, potentially providing better performance on multi-core systems for large arrays.

83. What is the purpose of the Files.lines() method?

Files.lines() is used to read all lines from a file as a Stream. It's a convenient way to process large files line by line.

84. What is the purpose of the Files.walk() method?

Files.walk() is used to traverse a file tree. It returns a Stream of all files in a directory and its subdirectories.

85. What is the purpose of the Files.find() method?

Files.find() is used to search for files in a file tree. It allows you to specify a search depth and a predicate to match files.

86. What is the purpose of the SecureRandom.getInstanceStrong() method?

SecureRandom.getInstanceStrong() returns a SecureRandom object that's suitable for sensitive or long-lived keys. It may block to gather additional system entropy if necessary.

87. What is the purpose of the String.join() method?

String.join() is used to create a new string by joining multiple strings together with a specified delimiter. It's a convenient way to concatenate strings, especially when working with collections.

88. What is the purpose of the String.chars() method?

String.chars() returns an IntStream of the char values in the string. This allows you to perform various stream operations on the characters of a string.

89. What is the purpose of the Comparator.reverseOrder() method?

Comparator.reverseOrder() returns a comparator that imposes the reverse of the natural ordering on a collection of objects. It's useful for sorting collections in descending order.

90. What is the purpose of the Math.toIntExact() method?

Math.toIntExact() converts a long to an int, throwing an ArithmeticException if the value overflows an int. It's useful for safely converting between numeric types.

91. What is the purpose of the Objects.requireNonNull() method?

Objects.requireNonNull() checks if the given object reference is null and throws a NullPointerException if it is. It's useful for validating method parameters.

92. What is the purpose of the Collections.checkedCollection() method?

Collections.checkedCollection() returns a dynamically typesafe view of the specified collection. It's useful for ensuring that a collection contains only elements of a specific type at runtime.

93. What is the purpose of the Stream.iterate() method?

Stream.iterate() returns an infinite sequential ordered Stream produced by iterative application of a function to an initial element. It's useful for generating sequences.

94. What is the purpose of the IntStream.range() method?

IntStream.range() returns a sequential ordered IntStream from startInclusive (inclusive) to endExclusive (exclusive). It's useful for generating a range of numbers.

95. What is the purpose of the Collectors.groupingBy() method?

Collectors.groupingBy() is used to group objects by some property and store the results in a Map. It's commonly used with the Stream.collect() method.

96. What is the purpose of the Collectors.partitioningBy() method?

Collectors.partitioningBy() is used to partition objects into two groups based on a predicate and store the results in a Map. It's similar to groupingBy(), but always results in a Map with boolean keys.

97. What is the purpose of the Optional.flatMap() method?

Optional.flatMap() is similar to map(), but the provided mapper function returns an Optional. It's useful for chaining together operations that may or may not return a value.

98. What is the purpose of the Stream.concat() method?
Stream.concat() is used to create a lazily concatenated stream whose elements are all the elements of the first stream followed by all the elements of the second stream. It's useful for combining multiple streams.

99. What is the purpose of the IntSummaryStatistics class?
IntSummaryStatistics is a state object for collecting statistics such as count, min, max, sum, and average. It's often used with streams to calculate multiple statistics in a single pass over the data.

100. What is the purpose of the Collectors.toMap() method?
Collectors.toMap() is used to collect stream elements into a Map. It allows you to specify how to determine the keys and values of the map.
These questions and answers cover a wide range of Java 8 features and provide a comprehensive overview of the new capabilities introduced in this version of Java.