General

1. What New Features Were Added in Java 8? ↑

Java 8 ships with several new features but the most significant are the following:

- Lambda Expressions − a new language feature allowing treating actions as objects
- Method References − enable defining Lambda Expressions by referring to methods directly using their names
- Optional − special wrapper class used for expressing optionality
- Functional Interface – an interface with maximum one abstract method, implementation can be provided using a Lambda Expression
- Default methods − give us the ability to add full implementations in interfaces besides abstract methods
- Nashorn, JavaScript Engine − Java-based engine for executing and evaluating JavaScript code
- Stream API − a special iterator class that allows processing collections of objects in a functional manner
- Date API − an improved, immutable JodaTime-inspired Date API Along with these new features, lots of feature enhancements are done under-the-hood, at both compiler and JVM level.

2. Describe Some of the Functional Interfaces in the Standard Library. ↑

There are a lot of functional interfaces in the java.util.function package, the more common ones include but not limited to:

- Function – it takes one argument and returns a result
- Consumer – it takes one argument and returns no result (represents a side effect)
- Supplier – it takes not argument and returns a result
- Predicate – it takes one argument and returns a boolean

- BiFunction – it takes two arguments and returns a result
- BinaryOperator – it is similar to a BiFunction, taking two arguments and returning a result. The two arguments and the result are all of the same types
- UnaryOperator – it is similar to a Function, taking a single argument and returning a result of the same type

3. What Is a Lambda Expression and What Is It Used for? ↑

In very simple terms, a lambda expression is a function that can be referenced and passed around as an object.

Lambda expressions introduce functional style processing in Java and facilitate the writing of compact and easy-to-read code.

Because of this, lambda expressions are a natural replacement for anonymous classes as method arguments. One of their main uses is to define inline implementations of functional interfaces.

4. What Is Nashorn in Java8? ↑

Nashorn is the new Javascript processing engine for the Java platform that shipped with Java 8. Until JDK 7, the Java platform used Mozilla Rhino for the same purpose. as a Javascript processing engine. Nashorn provides better compliance with the ECMA normalized JavaScript specification and better runtime performance than its predecessor.

5. What Is JJS? ↑

In Java 8, jjs is the new executable or command line tool used to execute Javascript code at the console.

6. What Is a Stream? How Does It Differ from a Collection? ↑

In simple terms, a stream is an iterator whose role is to accept a set of actions to apply on each of the elements it contains.

The stream represents a sequence of objects from a source such as a collection, which supports aggregate operations. They were designed to make collection processing simple and concise. Contrary to the collections, the logic of iteration is implemented inside the stream, so we can use methods like map and flatMap for performing a declarative processing.

Another difference is that the Stream API is fluent and allows pipelining. And yet another important distinction from collections is that streams are inherently lazily loaded and processed.

7. What Is Stream Pipelining in Java 8? ↑

Stream pipelining is the concept of chaining operations together. This is done by splitting the operations that can happen on a stream into two categories: intermediate operations and terminal operations.

Each intermediate operation returns an instance of Stream itself when it runs, an arbitrary number of intermediate operations can, therefore, be set up to process data forming a processing pipeline.

There must then be a terminal operation which returns a final value and terminates the pipeline.

8. What is a Functional Interface? What is SAM Interface? ↑

A Functional Interface is an interface, which contains one and only one abstract method. Functional Interface is also know as SAM Interface because it contains only one abstract method.

SAM Interface stands for Single Abstract Method Interface. Java SE 8 API has defined many Functional Interfaces.

9. What is Optional in Java 8? What is the use of Optional?Advantages of Java 8 Optional? ↑

Optional: Optional is a final Class introduced as part of Java SE 8. It is defined in java.util package.

It is used to represent optional values that is either exist or not exist. It can contain either one value or zero value. If it contains a value, we can get it. Otherwise, we get nothing.

It is a bounded collection that is it contains at most one element only. It is an alternative to “null” value.

Main Advantage of Optional is: - It is used to avoid null checks. - It is used to avoid “NullPointerException”.

10. What is the difference between Collections and Stream in Java8? ↑

Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.

11. What is the purpose of filter method of stream in java 8? ↑

The 'filter' method is used to eliminate elements based on a criteria.

12. What does the flatmap() function do? why you need it? ↑

The flatmap function is an extension of the map function. Apart from transforming one object into another, it can also flatten it.

For example, if you have a list of the list but you want to combine all elements of lists into just one list. In this case, you can use flatMap() for flattening. At

the same time, you can also transform an object like you do use map() function.

13. What is the difference between intermediate and terminal operations on Stream? ↑

The intermediate Stream operation returns another Stream, which means you can further call other methods of Stream class to compose a pipeline.

For example after calling map() or flatMap() you can still call filter() method on Stream.

On the other hand, the terminal operation produces a result other than Streams like a value or a Collection.

Once a terminal method like forEach() or collect() is called, you cannot call any other method of Stream or reuse the Stream.

14. What does the peek() method does? When should you use it? ↑

The peek() method of Stream class allows you to see through a Stream pipeline. You can peek through each step and print meaningful messages on the console. It's generally used for debugging issues related to lambda expression and Stream processing.

15. What is difference between findFirst() and findAny() method? ↑

The findFirst() method will return the first element meeting the criterion i.e. Predicate, while the findAny() method will return any element meeting the criterion, very useful while working with a parallel stream.

16. What is the difference between PermGenSpace and MetaSpace? ↑

In jdk 8 onwards PermGenSpace is removed. Earlier PermGenSpace is used for storing the metadata. Metadata means storing the information about classes

like bytecodes, names and JIT information. Java classes metadata now stored in native heap and this space is called MetaSpace. Metaspace grows automatically by default and will be garbage collected.

So the major difference between PermGenSpace and MetaSpace is that PermGenSpace was fixed in size and did not grow automatically, but MetaSpace does not have any size constraints.

17. What is a default method in Java 8 ? When to use it? ↑

Default method is also known as defender methods or virtual extension methods. It is a non abstract method i.e have body, which can be declared inside interface. Default method is introduced in Java 8 for backward compatibility. That is if you add a new abstract method to the interface, all the implementing classes shall break. Implementing classes need to implement the added abstract method. This problem is solved by default method of java 8.

18. What is the difference and similarities between Function and Predicate in java 8? ↑

Difference: - Return Type : Function returns an Object and it is a single argument function. Predicate return type is boolean (i.e true or false) and it is also a single argument function.

Similarities: - Both are functional interfaces i.e both contain single abstract method.

19. What is the difference between Internal iteration and External iteration? ↑

Java 8 has introduced the new concept "internal iteration". Prior to java 8 there is only external iteration. Let's dive into the differences between internal iteration and external iteration.

- Availability: Internal iteration is added in jdk 8 while external iteration is there before jdk 8.
- Iteration behavior: Internal iterator iterating an Aggregated Object elements like Collections, Arrays internally. External iterator iterating an Aggregated Object elements externally.
- Approach: Internal iterator follows functional programming approach that is declarative style. Meanwhile, External iterator follows OOP approach i.e imperative style.

20. Is it possible to define our own Functional Interface? Explain the rules to define a functional interface. ↑

It is possible to define our own functional interfaces. A user can use Java SE 8's @FunctionalInterface annotation to mark an interface as Functional Interface. The following rules need to be kept in mind when creating a functional interface.

- Only one interface must be defined having only one abstract method
- More than on abstract methods cannot be defined
- A user should make use of @FunctionalInterface annotation in the interface definition.
- Any number of different methods like the default method, static method, etc. can be defined.
- We can override java.lang.Object class's method as an abstract method and this will not be counted as an abstract method.

21. What is StringJoiner? ↑

StringJoiner is a util method which is used to construct different strings with desired delimiters. It can also help in creating sequences of different characters separated by delimiters. This was introduced in Java 8. The different constructors are Public StringJoiner(CharSequence delimiter) and Public StringJoiner(CharSequence delimiter,CharSequence prefix,CharSequence suffix).

22. Why was a new version of Java needed in the first place? ↑

There are two main reasons: - Dramatic changes in hardware created the need for Java to use current multi-core CPUs more efficiently - Enable users to use new Functional Programming (FP) features

23. What is Type Inference? ↑

Type inference helps the compiler determine the argument types by looking at each method invocation and corresponding declaration.

24. What is a stream, and how does it differ from a collection? ↑

A stream is an iterator whose function is to accept a set of actions and apply them to each of the elements it contains. A stream represents an object sequence from a collection or other source that supports aggregate operations. Unlike collections, iteration logic implements inside the stream.

Also, streams are inherently lazily loaded and processed, unlike collections.

25. Explain local datetime API in Java8? ↑

In new data-time API, one of the class is Local Date-Time API where there is no problem of handling of the time-zones. Programmers use this API where time zones are not required. Local Date-Time API is defined in the package java.time

26. What do you mean by chromounits in java8? ↑

The Chrono unit was added in Java 8 to replace those integer value that was used in old API to represent the month, day, year, etc. unit is defined in the java.time.temporal.ChronoUnit

27. What is type inference in Java8? ↑

Type inference is a feature of Java that gives the capability to the compiler to seem at each method invocation and corresponding announcement to determine the type of arguments. Java provides multiplied model of type inference in Java eight

28. What is :: (double colon) operator-Method References in Java 8? ↑

Usually we use lambda expressions to create anonymous methods which return us the desired output. But sometimes lambda expressions do nothing but call an existing method. Because this lambda expression calls an existing method, method reference can be used here instead of Lambda function. Method reference is described using :: (double colon) symbol.

29. What is Optional in Java 8? ↑

Java 8 introduced a new container class java.util.Optional. It wraps a single value, if that value is available. If the value is not available an empty optional should be returned. Thus it represents null value with absent value. This class has various utility methods like isPresent() which helps users to avoid making use of null value checks. So instead of returning the value directly, a wrapper object is returned thus users can avoid the null pointer exception.

30. What is the distinct feature of the Block of Code? ↑

A Block of Code has the distinct feature of getting executed on only demand.

31. How is the Parameter List of Lambda Expression different from the Lambda Arrow Operator? ↑

Lambda Expression can carry zero, one or even more parameters at one time. On the other hand, the Lambda Arrow Operator separates these parameters from the list and body using the icon "->".

32. What are the guidelines that are needed to be followed in Functional Interface? ↑

There are several guidelines stated below which are needed to be followed in Functional Interface.

- The interface should be defined with only one abstract method.
- Not more than one abstract can be defined.
- Making use of @Functionalinterface annotation in the interface definition.
- The override of the Java.lang.object class's method will not be considered as an abstract method.
- Any methods can be used for defining a number.

33. What is the similarity between Map and Flat map stream operation? ↑

Both the Map and FlatMap stream operation is intermediate stream operations that receive a function and also apply these functions to different elements of the stream.

34. What is the major difference between Map and FlatMap stream operation? ↑

The major difference between Map and FlatMap stream operation is that the earlier wraps its return value inside its ordinal type while the latter does not.

35. Can we list the numbers and remove the duplicate elements in the list using Java SE 8 features? ↑

Yes, we can list the numbers and remove the duplicate elements in the list by applying stream and then collecting it to set using Collections.toSet() method.

36. What are the defining rules of a functional interface? ↑

A functional interface meets the following requirements. It contains only one abstract method and cannot define additional ones. It uses the

@FunctionalInterface annotation. A functional interface can have other types of methods, outside of a single abstract method, like static or default methods.

37. What is Diamond Problem in Inheritance? How does Java 8 solve this problem? ↑

=HYPERLINK("https://www.cs.cornell.edu/courses/JavaAndDS/abstractInterface/05diamond.pdf","Read up on the Diamond problem here.")

38. What is StringJoiner? ↑

StringJoiner belongs to java.util package and is used to combine different strings into a single string with delimiters, and also with prefixes and suffixes. It uses two different constructors for this purpose. The constructors are Public StringJoiner(CharSequence delimiter) and Public StringJoiner(CharSequence delimiter, CharSequence prefix, CharSequence suffix).

39. What is the need of static method in Interface? ↑

Java 8 has added static methods in interface to provide utility methods on interface level without creating the object of the interface. Some of the static methods of the java 8 are as below : - Stream.of() - Stream.iterate() - Stream.empty()

40. What is Method Reference in Java 8? ↑

Java provides new feature called Method reference to call the single method of functional interface. Method reference is a short form of lambda expressions used on Functional Interface. We can replace our lambda expression with Method reference to clean the code.

41. What are different ways to create Optional? ↑

- Optional.empty() – This method will return an empty Optional object. Optional card = Optional.empty();
- Optional.of() – This method will return an Optional of object passed as an argument to the of method. Returns an Optional with the specified present non-null value. Optional card = Optional.ofNullable(new GraphicsCard());
- Optional. ofNullable() - Returns an Optional describing the specified value, if non-null, otherwise returns an empty Optional Optional card = Optional.ofNullable(null);

42. Can we have a default method definition in the interface without specifying the keyword "default"? ↑

No. Compiler complains that its an abstract method and hence shouldn't have the body.

43. If there is a conflict between Base Class Method definition and Interface Default method definition, which definition is picked? ↑

Base Class Definition.

44. What is the @FunctionalInterface annotation? ↑

This is an informative annotation that specify that the interface is a functional interface. A Function Interface has only one abstract method and many default methods. Compiler generates an error if the interface specified with the annotation doesn't abide by the specifications for functional interface.

45. What do you mean by Default Methods? ↑

With Java 8, an interface can have default execution of a function in interfaces.

46. What is statistics collector in Java 8? ↑

With Java 8, statistics collectors are introduced to calculate all statistics when stream processing is being done.

47. How are the functional interface and Lambda expressions related? ↑

Lambda expressions are functional only to the user interface's abstract method.

48. What are repeating annotations? ↑

Java 8.0 introduced a new Java language feature regarding annotations. You can now repeat an annotation multiple times on a type declaration.

49. What are type annotations? Name some common type annotations. ↑

ava 8.0 introduced a new Java language feature regarding annotations. In addition to using annotations on type declarations, you can now apply annotations whenever you use types.

Following are some examples of Type annotations - @NonNull - @ReadOnly - @Regex - @Tainted - @Untainted

50. How do you use lambda expression with functional interface? ↑

Lambda expression provides implementation of the abstract method defined by the functional interface.

51. What is block lambda expression? ↑

A block lambda is the lambda expression where the right side of the lambda expression is a block of code.

52. Why lambda expression is called a poly expression? ↑

The type of a lambda expression is inferred from the target type thus the same lambda expression could have different types in different contexts.

53. Can we have a generic functional interface? ↑

Since lambda expression doesn't have type parameters of its own so it can't be generic. But the functional interface that specifies the target type for the lambda expression can be generic.

54. Comparator method is a functional interface but I see a lot of other methods in Comparator method then how is it a Single Abstract method interface? ↑

From Java 8 it is possible for an interface to have default methods and static methods so, in a functional interface there may be other default and static methods but there must be only one abstract method.

A functional interface can specify Object class public methods too in addition to the abstract method. That interface will still be a valid functional interface. The public Object methods are considered implicit members of a functional interface as they are automatically implemented by an instance of functional interface.

Advanced

1. What is a Spliterator? ↑

The term is a blend of “splitable” and “iterator” and is a new feature in Java SE 8. It is used in Stream API to iterate streams in a parallel or sequential order by internal iteration.