### Describe the newly added features in Java 8?

| **Feature Name** | **Description** |
| --- | --- |
| Lambda expression | A function that can be shared or referred to as an object. |
| Functional Interfaces | Single abstract method interface. |
| Method References | Uses function as a parameter to invoke a method. |
| Default method | It provides an implementation of methods within interfaces enabling 'Interface evolution' facilities. |
| Stream API | Abstract layer that provides pipeline processing of the data. |
| Date Time API | New improved joda-time inspired APIs to overcome the drawbacks in previous versions |
| Optional | Wrapper class to check the null values and helps in further processing based on the value. |
| Nashorn, JavaScript Engine | An improvised version of JavaScript Engine that enables JavaScript executions in Java, to replace Rhino. |

**What are the significant advantages of Java 8?**

* More compact code
* Less boiler plate code
* More readable and reusable code
* More testable code
* Parallel operations

## What are functional interfaces?

Functional interfaces are those interfaces which can have only one abstract method.It can have static method, default methods or can override Object’s class methods.

There are many functional interfaces already present in java such as Comparable, Runnable

## How lambda expression and functional interfaces are related?

Lambda expressions can only be applied to abstract method of functional interface.  
**For example**

Runnable has only one abstract method called run, so it can be used as below:

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | // Using lambda expression  Thread t1=new Thread(  ()->System.out.println("In Run method")  ); |

Here we are using Thread constructor which takes Runnable as parameter. As you can see we did not specify any function name here, as Runnable has only one abstract method, java will implicitly create anonymous Runnable and execute run method.

## Can you create your own functional interface?

Yes, you can create your own functional interface. Java can implicitly identify functional interface but you can also annotate it with @FunctionalInterface.

### Can a functional interface extend/inherit another interface?

A functional interface cannot extend another interface with abstract methods as it will void the rule of one abstract method per functional interface.

It can extend other interfaces which do not have any abstract method and only have the default, static, another class is overridden, and normal methods.

### What are some standard Java pre-defined functional interfaces?

Some of the famous pre-defined functional interfaces from previous Java versions are Runnable, Callable, Comparator, and Comparable. While Java 8 introduces functional interfaces like Supplier, Consumer, Predicate, etc.

### What are the various categories of pre-defined function interfaces?

**Function:** To transform arguments in returnable value.

**Predicate:** To perform a test and return a Boolean value.

**Consumer:** Accept arguments but do not return any values.

**Supplier:** Do not accept any arguments but return a value.

**Operator:** Perform a reduction type operation that accepts the same input types.

## What is lambda expression?

Lambda expression is anonymous function which have set of parameters and a lambda (->) and a function body .You can call it function without name.

#### Structure of Lambda Expressions

|  |  |
| --- | --- |
| 1  2  3  4 | (Argument List) ->{expression;} or  (Argument List) ->{statements;} |

**What are the features of a lambda expression?**

Below are the two significant features of the methods that are defined as the lambda expressions:

* Lambda expressions can be passed as a parameter to another method.
* Lambda expressions can be standalone without belonging to any class.

### In Java 8, what is Method Reference?

Method reference is a compact way of referring to a method of functional interface. It is used to refer to a method without invoking it. :: (double colon) is used for describing the method reference. The syntax is class::methodName

For e.g.:

Integer::parseInt(str) \\ method reference

str -> Integer.ParseInt(str); \\ equivalent lambda

### What does the String::ValueOf expression mean?

It is a static method reference to method Valueof() of class String. It will return the string representation of the argument passed

### What is an Optional class?

Optional is a container type which may or may not contain value i.e. zero(null) or one(not-null) value. It is part of java.util package. There are pre-defined methods like isPresent(), which returns true if the value is present or else false and the method get(), which will return the value if it is present.

### What are the advantages of using the Optional class?

Below are the main advantage of using the Optional class:

It encapsulates optional values, i.e., null or not-null values, which helps in avoiding null checks, which results in better, readable, and robust code It acts as a wrapper around the object and returns an object instead of a value, which can be used to avoid run-time NullPointerExceptions.

## **What is Stream?**

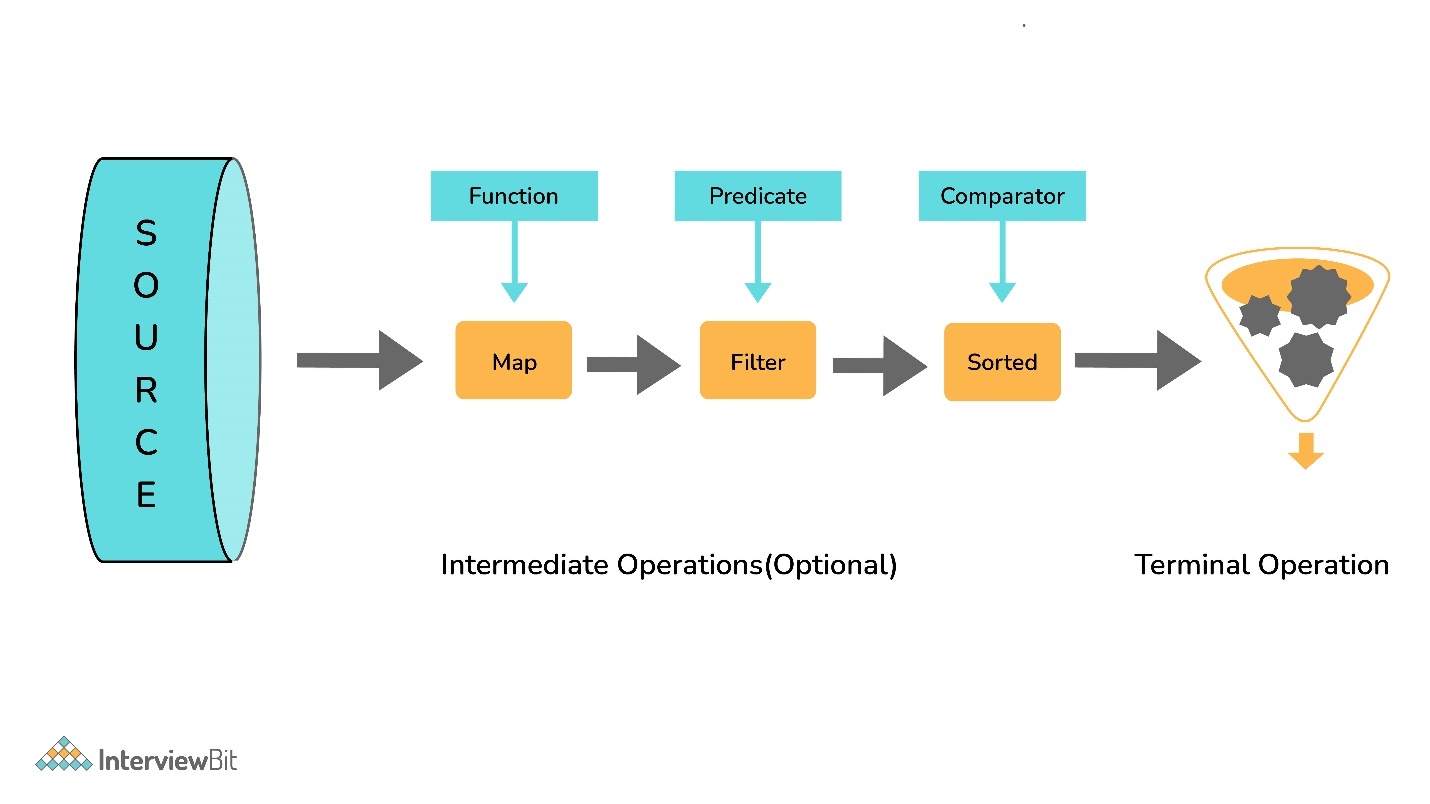
Stream represents a sequence of objects from a source, which supports aggregate operations. Following are the characteristics of a Stream −

* **Sequence of elements** − A stream provides a set of elements of specific type in a sequential manner. A stream gets/computes elements on demand. It never stores the elements.
* **Source** − Stream takes Collections, Arrays, or I/O resources as input source.
* **Aggregate operations** − Stream supports aggregate operations like filter, map, limit, reduce, find, match, and so on.
* **Pipelining** − Most of the stream operations return stream itself so that their result can be pipelined. These operations are called intermediate operations and their function is to take input, process them, and return output to the target. collect() method is a terminal operation which is normally present at the end of the pipelining operation to mark the end of the stream.
* **Automatic iterations** − Stream operations do the iterations internally over the source elements provided, in contrast to Collections where explicit iteration is required.

**What are the main components of a Stream?**

Components of the stream are:

* A data source
* Set of Intermediate Operations to process the data source
* Single Terminal Operation that produces the result



**What are Intermediate and Terminal operations?**

**Intermediate Operations:**

* Process the stream elements.
* Typically transforms a stream into another stream.
* Are lazy, i.e., not executed till a terminal operation is invoked.
* Does internal iteration of all source elements.
* Any number of operations can be chained in the processing pipeline.
* Operations are applied as per the defined order.
* Intermediate operations are mostly lambda functions.

**Terminal Operations:**

* Kick-starts the Stream pipeline.
* used to collect the processed Stream data.

**int** count = Stream.of(1, 2, 3, 4, 5)

.filter(i -> i <4) // Intermediate Operation filter

.count(); // Terminal Operation count

### What are the most commonly used Intermediate operations?

**Filter(Predicate<T>)** - Allows selective processing of Stream elements. It returns elements that are satisfying the supplied condition by the predicate.

**map(Funtion<T, R>)** - Returns a new Stream, transforming each of the elements by applying the supplied mapper function.= sorted() - Sorts the input elements and then passes them to the next stage.

**distinct()** - Only pass on elements to the next stage, not passed yet.

**limit(long maxsize)** - Limit the stream size to maxsize.

**skip(long start)** - Skip the initial elements till the start.

**peek(Consumer)** - Apply a consumer without modification to the stream.

**flatMap(mapper)** - Transform each element to a stream of its constituent elements and flatten all the streams into a single stream

### What is the stateful intermediate operation? Give some examples of stateful intermediate operations.

To complete some of the intermediate operations, some state is to be maintained, and such intermediate operations are called stateful intermediate operations. Parallel execution of these types of operations is complex.

For Eg: sorted() , distinct() , limit() , skip() etc.

Sending data elements to further steps in the pipeline stops till all the data is sorted for sorted() and stream data elements are stored in temporary data structures

**What is the most common type of Terminal operations?**

* collect() - Collects single result from all elements of the stream sequence.
* reduce() - Produces a single result from all elements of the stream sequence
  + count() - Returns the number of elements on the stream.
  + min() - Returns the min element from the stream.
  + max() - Returns the max element from the stream.
* Search/Query operations
  + anyMatch() , noneMatch() , allMatch() , ... - Short-circuiting operations.
  + Takes a Predicate as input for the match condition.
  + Stream processing will be stopped, as and when the result can be determined.
* Iterative operations
  + forEach() - Useful to do something with each of the Stream elements. It accepts a consumer.
  + forEachOrdered() - It is helpful to maintain order in parallel streams.

### What is the difference between findFirst() and findAny()?

| **findFirst()** | **findAny()** |
| --- | --- |
| Returns the first element in the Stream | Return any element from the Stream |
| Deterministic in nature | Non-deterministic in nature |

### How are Collections different from Stream?

Collections are the source for the Stream. Java 8 collection API is enhanced with the default methods returning Stream<T> from the collections.

| **Collections** | **Streams** |
| --- | --- |
| Data structure holds all the data elements | No data is stored. Have the capacity to process an infinite number of elements on demand |
| External Iteration | Internal Iteration |
| Can be processed any number of times | Traversed only once |
| Elements are easy to access | No direct way of accessing specific elements |
| Is a data store | Is an API to process the data |

**What is the feature of the new Date and Time API in Java 8?**

* Immutable classes and Thread-safe
* Timezone support
* Fluent methods for object creation and arithmetic
* Addresses I18N issue for earlier APIs
* Influenced by popular joda-time package
* All packages are based on the ISO-8601 calendar system

**30. What are the important packages for the new Data and Time API?**

* java.time
  + dates
  + times
  + Instants
  + durations
  + time-zones
  + periods
* Java.time.format
* Java.time.temporal
* java.time.zone

## What is method reference in java 8?

[Method reference](https://java2blog.com/java-8-method-reference/) is used refer method of functional interface. It is nothing but compact way of lambda expression. You can simply replace lambda expression with method reference.  
**Syntax:**  
class::methodname

## What is Optional? Why and how can you use it?

Java 8 has introduced new class Called Optional. This class is basically introduced to avoid NullPointerException in java.  
Optional class encapsulates optional value which is either present or not.

# Interface default methods in java 8

Have you ever faced a situation, when you created an interface and many classes implemented that interface. Now you need to add new methods to interface. After adding new methods, your java project will be full of compilation errors because you need to add these new methods to all classes which are implementing that interface

The oneliner for this could be “backward compatibility”.If JDK modifies an interface, then all classes which implements this interface will break.

For adding lambda expression in Java 8, JDK needs to add methods(such as foreach) to List or collections Interface, but if you add this method to these interface, it will break millions lines of code as class which implements the interface, need to implement all its methods

By adding default method in interface, you can provide default implementation of it without affecting implementing classes as it includes implementation of that method and any implementing class which needs that method can override it.

## Difference between default methods and abstract class

Introduction of default methods to interface bridge gap between interface and abstract class.Now interface looks very similar to abstract classes but there are still differences. Lets list them

|  |  |  |
| --- | --- | --- |
| **Parameter** | **Abstract class** | **Interface with default methods** |
| State of objects | Abstract class can hold state of object | Interface with default methods can not hold state of objects |
| Access Modifier | Abstract class methods can have public ,protected,private and default modifier | Interface methods are by default public. you can not use any other access modifier with it |
| Constructor | Abstract class can have constructor | Interface  can not have constructor |
| Member variables | It can have member variables |  |

**What was wrong with the old date and time?**

**Answer:** **Enlisted below are the drawbacks of the old date and time:**

* Java.util.Date is mutable and is not thread-safe whereas the new Java 8 Date and Time API are thread-safe.
* Java 8 Date and Time API meets the ISO standards whereas the old date and time were poorly designed.
* It has introduced several API classes for a date like LocalDate, LocalTime, LocalDateTime, etc.
* Talking about the performance between the two, Java 8 works faster than the old regime of date and time.

**What is the difference between the Collection API and Stream API?**

**Answer:** **The difference between the Stream API and the Collection API can be understood from the below table:**

| **Stream API** | **Collection API** |
| --- | --- |
| It was introduced in Java 8 Standard Edition version. | It was introduced in Java version 1.2 |
| There is no use of the Iterator and Spliterators. | With the help of forEach, we can use the Iterator and Spliterators to iterate the elements and perform an action on each item or the element. |
| An infinite number of features can be stored. | A countable number of elements can be stored. |
| Consumption and Iteration of elements from the Stream object can be done only once. | Consumption and Iteration of elements from the Collection object can be done multiple times. |
| It is used to compute data. | It is used to store data. |

**What is a SAM Interface?**

**Answer:** Java 8 has introduced the concept of FunctionalInterface that can have only one abstract method. Since these Interfaces specify only one abstract method, they are sometimes called as SAM Interfaces. SAM stands for “Single Abstract Method”.

**What is a Stream API? Why do we require the Stream API?**

**Answer:** Stream API is a new feature added in Java 8. It is a special class that is used for processing objects from a source such as Collection.

**We require the Stream API because,**

* It supports aggregate operations which makes the processing simple.
* It supports Functional-Style programming.
* It does faster processing. Hence, it is apt for better performance.
* It allows parallel operations.
* **What is the difference between Iterator and Spliterator?**
* **Answer:** Below is the differences between Iterator and Spliterator.

| **Iterator** | **Spliterator** |
| --- | --- |
| It was introduced in Java version 1.2 | It was introduced in Java SE 8 |
| It is used for Collection API. | It is used for Stream API. |
| Some of the iterate methods are next() and hasNext() which are used to iterate elements. | Spliterator method is tryAdvance(). |
| We need to call the iterator() method on Collection Object. | We need to call the spliterator() method on Stream Object. |
| Iterates only in sequential order. | Iterates in Parallel and sequential order. |

**What is the Consumer Functional Interface?**

**Answer:** Consumer Functional Interface is also a single argument interface (like Predicate<T> and Function<T, R>). It comes under java.util.function.Consumer. This does not return any value.

**What is the Supplier Functional Interface?**

**Answer:** Supplier Functional Interface does not accept input parameters. It comes under java.util.function.Supplier. This returns the value using the get method.

**What is the Difference Between Map and flatMap Stream Operation?**

**Answer:** Map Stream operation gives one output value per input value whereas flatMap Stream operation gives zero or more output value per input value.

Map Stream operation is generally used for simple operation on Stream.

FlatMap Stream operation is used for more complex Stream operation.

**What is ChronoUnits in Java 8?**

**Answer:** ChronoUnits is the enum that is introduced to replace the Integer values that are used in the old API for representing the month, day, etc.

**Explain StringJoiner Class in Java 8? How can we achieve joining multiple Strings using StringJoiner Class?**

**Answer:** In Java 8, a new class was introduced in the package java.util which was known as StringJoiner. Through this class, we can join multiple strings separated by delimiters along with providing prefix and suffix to them.

In the below program, we will learn about joining multiple Strings using StringJoiner Class. Here, we have “,” as the delimiter between two different strings. Then we have joined five different strings by adding them with the help of the add() method. Finally, printed the String Joiner.

In the next question #35, you will learn about adding prefix and suffix to the string.

|  |
| --- |
| import java.util.StringJoiner;    public class Java8 {      public static void main(String[] args) {            StringJoiner stj = new StringJoiner(",");          // Separated the elements with a comma in between.            stj.add("Saket");          stj.add("John");          stj.add("Franklin");          stj.add("Ricky");          stj.add("Trevor");            // Added elements into StringJoiner “stj”            System.out.println(stj);      }  } |

**Output:**

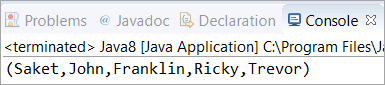
[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/04/34th-Question.png)

**Write a Java 8 program to add prefix and suffix to the String?**

**Answer:** In this program, we have “,” as the delimiter between two different strings. Also, we have given “(” and “)” brackets as prefix and suffix. Then five different strings are joined by adding them with the help of the add() method. Finally, printed the String Joiner.

|  |
| --- |
| import java.util.StringJoiner;    public class Java8 {      public static void main(String[] args) {            StringJoiner stj = new StringJoiner(",", "(", ")");            // Separated the elements with a comma in between.          //Added a prefix "(" and a suffix ")"            stj.add("Saket");          stj.add("John");          stj.add("Franklin");          stj.add("Ricky");          stj.add("Trevor");            // Added elements into StringJoiner “stj”            System.out.println(stj);      }  } |

**Output:**

[](https://www.softwaretestinghelp.com/wp-content/qa/uploads/2020/04/35th-Question.png)