**Java 8 Interview Questions**

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

* Compact, readable, and reusable code.
* Less boilerplate code.
* Parallel operations and execution.
* Can be ported across operating systems.
* High stability.
* Stable environment.
* Adequate support

.java 8 features

Ans : java 8 provides following features :

o Lambda expressions,

o Method references,

o Functional interfaces,

o Stream API,

o Default methods,

o Base64 Encode Decode,

o Static methods in interface,

o Optional class,

o Collectors class,

o ForEach() method,

o Parallel array sorting,

o Nashorn JavaScript Engine,

o Parallel Array Sorting,

o Type and Repating Annotations

, o IO Enhancements,

o Concurrency Enhancements,

o JDBC Enhancements etc.

### What is the default method, and why is it required?

A method in the interface that has a predefined body is known as the default method. It uses the keyword default. default methods were introduced in Java 8 to have 'Backward Compatibility in case JDK modifies any interfaces. In case a new abstract method is added to the interface, all classes implementing the interface will break and will have to implement the new method. With default methods, there will not be any impact on the interface implementing classes. default methods can be overridden if needed in the implementation. Also, it does not qualify as synchronized or final.

### What are static methods in Interfaces?

Static methods, which contains method implementation is owned by the interface and is invoked using the name of the interface, it is suitable for defining the utility methods and cannot be overridden.

### 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.

**Runnable:** use to execute the instances of a class over another thread with no arguments and no return value.

**Callable:** use to execute the instances of a class over another thread with no arguments and it either returns a value or throws an exception.

**Comparator:** use to sort different objects in a user-defined order

**Comparable:** use to sort objects in the natural sort order.

### 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 the lambda expression in Java and How does a lambda expression relate to a functional interface?

Lambda expression is a type of function without a name. It may or may not have results and parameters. It is known as an anonymous function as it does not have type information by itself. It is executed on-demand. It is beneficial in iterating, filtering, and extracting data from a collection.

As lambda expressions are similar to anonymous functions, they can only be applied to the single abstract method of Functional Interface. It will infer the return type, type, and several arguments from the signature of the abstract method of functional interface

**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

### 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 are Java 8 streams?

A stream is an abstraction to express data processing queries in a declarative way.

A Stream, which represents a sequence of data objects & series of operations on that data is a data pipeline that is not related to Java I/O Streams does not hold any data permanently.  
The key interface is java.util.stream.Stream<T>. It accepts Functional Interfaces so that lambdas can be passed. Streams support a fluent interface or chaining. Below is the basic stream timeline marble diagram:

**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 the sources of data objects a Stream can process?**

A Stream can process the following data:

* A collection of an Array.
* An I/O channel or an input device.
* A reactive source (e.g., comments in social media or tweets/re-tweets)
* A stream generator function or a static factory.

**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 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 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 are the advantages of the Java SE 8?

The introduction of Java SE 8 has been determined to be extremely valuable for programmers in several ways. These are:

1. It makes the code more concise and understandable. Functional Interfaces and Lambda Expressions in Java SE 8 made the code more reusable.
2. In Java 8, code can be maintained and tested more quickly as compared to previous versions.
3. It helps in writing extraordinarily interfaces and highly scalable code.
4. It helps in writing parallel code.
5. It made writing database-like operations easy.
6. It provides better performance to different applications
7. Code can be made more productive in Java SE 8.

### What are the Java Lambda Expressions?

Java lambda expressions were Java's initial step towards functional programming. Java Lambda Expression or Lambda function is simply an unnamed expression or function that is rewritten as a parameter for any other function. Lambda Expressions in Java are the functions that are used to be shared as an object. It can also be used for an object to be referenced. Lambda Expressions need more concise coding, include and it also implements a method of executing the functional interfaces of Java 8. Lambda expressions in Java allow the users to express one functioning unit to carry throughout to different code.

The syntax of Lambda Expressions in Java consists of three main components. These components are shown below:

* **Argument List:** The argument list is written inside the brackets in the syntax of Java lambda expressions. The argument list can be empty and can contain parameter or parameters as well.
* **Arrow Token:** The arrow token is represented by an arrow (->) in Java lambda expressions. It is used to link the parameters in the argument list to the expression of the body.
* **Body:** Java Lambda expression's body is written just after the arrow token. The body comprises the expressions and statements of the lambda expressions in Java.

**Syntax:**

1. (Argument List) :> {body;}

### What do you mean by Functional Interfaces in Java?

Functional interfaces are included in Java SE 8 with Lambda expressions and Method references in order to make code more readable, clean, and straightforward. These are the interfaces which ensure that they include precisely only one abstract method. It is used and executed by representing the interface with an annotation called **@FunctionalInterface**. As described earlier, functional interfaces can contain only one abstract method. However, they can include any quantity of default and static methods.

Functional Interface is additionally recognized as Single Abstract Method Interfaces. In short, they are also known as **SAM** interfaces.

Describe the guidelines that the users should follow while using the Java Functional Interfaces?

In the Java programming language, the user needs to follow many guidelines while using the Functional Interfaces of Java. These guidelines are:

* The functional interface should contain only one abstract method. However, it can contain any number of static and default methods.
* In Java functional interfaces, only one abstract method is allowed. We cannot use more than one abstract method in Functional interfaces.
* The user should use the annotation @Functionalinterface while defining the functional interface in Java.
* In functional interfaces, if we override the Java.lang.object class's method in the interface, it will not be counted as an abstract method.
* We can use any method in order to define a number.

Name any four built-in Java Functional Interfaces?

Since Java SE 1.8 onwards, there are many interfaces that are converted into functional interfaces. All these interfaces are annotated with @FunctionalInterface. These interfaces are as follows:

* **Runnable:** The interface only contains the run() method.
* **Comparable:** This interface only contains the compareTo() method.
* **ActionListener:** This interface only contains the actionPerformed() method.
* **Callable:** This interface only contains the call() method.

### Explain some of the functional interfaces which are used in Java?

There are many functional interfaces that are used in Java. The following are commonly used by the developers.

**Function:** Function is the most common functional interface out of all. It accepts only one argument and returns a specified result

**Consumer:** It is the same as Function interface, Consumer is also a functional interface that takes only one argument, and no result is returned by it.

**Supplier:** Supplier is a type of functional interface in Java that does not accept any argument and still returns the desired result.

**Predicate:** The type of functional interface in Java that accepts one argument and returns a boolean value is known as Predicate functional interface.

**BiFunction**: The Bi-Function is substantially related to a Function. Besides, it takes two arguments, whereas Function accepts one argument. Just like Function, it also returns a specified result.

**Unary Operator and Binary Operator**: There are also two other function interfaces that are named as Unary Operator and Binary Operator. They both extend the Function and Bi-Function, respectively. In simple words, Unary Operator extends Function, and Binary Operator extends Bi-Function.

The Unary Operator accepts only one argument and returns a single argument only. Still, in Unary Operator, both the input and output values must be identical and of the same type. On the other way, Binary Operator takes two values and returns one value comparable to Bi: Function, but similarly, like Unary Operator, the input and output value type must be identical and of the same type.

### Describe in brief about the method references in Java?

Method references were included in the Java programming language in the most advanced variant of Java, i.e., Java SE 8. Method references in Java are an extra unquestionably admirable trait that is introduced to Java SE 8. Lambda Expressions are simply gained extra vanity and flexibility with the guidance of Method references. Whereas Method references in Java SE 8 are a specific brand of the lambda expression, which invoke (reference) the methods by applying a method name.

In plain words, Method references in Java have typically reduced Lambda Expressions that are used for invoking methods. Method references are used when the lambda expression is declaring a function and doing nothing more. Two Integer colons (::) means method reference in Java.

### In Java SE 8, what do you mean by Streams?

Java implements a newly added package in Java SE 8 known as **java.util.stream**. The package contains many different interfaces, classes, and enum, that allows the user to perform functional-style operations on the elements. The entire idea of the Java streams is to facilitate functional-style processes on streams of elements.

A Java stream cannot be said to be a data structure. However, we can say that Java Stream is an abstraction. However, Java Streams is not a collection or set where we can save elements and store data. The primary relevant difference between a structure and a stream is that a stream doesn't store the data and components. A stream is a reflection of a non-changeable compilation of processes that are applied in any classification of the data.

In simple words, a Java Stream is a package that contains various classes and interfaces which are capable of doing the iteration of its own elements internally. Conversely, when we are working with iteration characteristics of the Java Collections, then we have to perform the iteration of the elements personally. (For example - we can use a Java Iterator or the Java for each loop in order to work with a Java Iterable).

### What are the two types of common operations of Java Streams?

Java Stream holds two different types of operations that are:

**Intermediate operations:** Intermediate operations are the operations that return a stream so that the user can chain various intermediate operations without using semicolons, as we do in other programming languages like Scala.

**Terminal operations:** The terminal operations are the operations that are mainly void and null, and if not null, these operations return a non-stream as a result.

 What is an ideal situation where the user can use the Streams API in Java?

The Stream API in Java 8 can be efficiently utilized in many scenarios. These scenarios are:

* Streams API are used where the user has to perform database operations.
* Streams API are used to execute operations lazily.
* IT helps in writing functional-style programming.
* It is used in performing parallel processing.
* Streams API are used in projects where the project demands the use of pipeline operations.
* In order to achieve internal iteration, Streams API is very useful.

### Using Java 8 Date and Time API, how can a user get the current date and time?

LocalDate/LocalTime and LocalDateTime classes analyze the growth where timezones are not needed. With Java SE 8, an innovativeDate-Time API is included to satisfy the subsequent shortcomings of the past date-time API. The underneath program is addressed with the guidance of the latest date-time API launched in Java SE 8. In order to get the current date and time, we have to use the built utilization of LocalDate, LocalTime, and LocalDateTime API to get the actual date and time.

### What do you mean by StringJoiner class in Java SE 8? How can we achieve joining multiple Strings using StringJoiner Class?

In Java SE 8, a new class was advanced and practiced in the java.util package which is called as StringJoiner class. With the help of this class, a user can merge various strings by separating them with some delimiters. StringJoiner class also helps in adding prefix and suffix to the strings.

In this program, we will add delimiter "," in between the various distinguished given strings. According to the question, we have to join many different strings. For this, we will take the help of add() method in order to add them. Finally, after that we will print the String Joiner.

#### **How is Collection API different from Stream API?**

The difference between Collection API and Stream API are:

|  |  |
| --- | --- |
| **Collection API** | **Stream API** |
| It is available since the introduction of Java 1.2 | It is made available after the introduction of Java SE 8 |
| It helps in storing Data which are set of objects. | It helps in computing data wherein computations of a set of data objects are made. |
| It can be used for storing a limited number of elements. | It can be used for storing an unlimited number of elements. |
| The construction of the Collection object is done Eagerly. | The construction of the Stream object is done Lazily. |

#### **What is the similarity between Map and Flat map stream operation?**

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

#### **What are some examples of Intermediate Operations?**

Examples of Intermediate Operations include the following.

1. Limit(long n)
2. skip (long n)
3. Distinct ()
4. Filter (Predicate)
5. Map (Function)

#### **What are some examples of Terminal Operations?**

Examples of terminal operations are:

1. Max
2. Min
3. AnyMatch
4. AllMatch
5. Reduce
6. ToArray
7. Count