InterviewBit

Java 8 Interview Questions

Contents

Java 8 Interview Questions for Freshers

Java 8 Interview Questions for Experienced

(.....Continued)

19. What are the advantages of using the Optional class?
20. What are Java 8 streams?
21. What are the main components of a Stream?
22. What are the sources of data objects a Stream can process?
23. What are Intermediate and Terminal operations?
24. What are the most commonly used Intermediate operations?
25. What is the stateful intermediate operation? Give some examples of stateful intermediate operations.
26. What is the most common type of Terminal operations?
27. What is the difference between findFirst() and findAny()?
28. How are Collections different from Stream?
29. What is the feature of the new Date and Time API in Java 8?
30. What are the important packages for the new Data and Time API?
31. Explain with example, LocalDate, LocalTime, and LocalDateTime APIs.
32. Define Nashorn in Java 8
33. What is the use of JJS in Java 8?

Let's get Started

Java, originally evolved from the Oak language, was born in early 1996 with its major version as Java 1 or JDK 1.0. Java was initially designed and developed by Sir James Gosling at Sun Microsystems. Java 8 or JDK 8.0 is one of the major releases of the Java programming language in 2014. It is also known by the codename Spider. Java is an open-source project and is currently managed by Oracle Corporation.

This article would walk you through the Java 8 interview questions for freshers and experienced, scope, and opportunities.

Java 8 Interview Questions for Freshers

1. Describe the newly added features in Java 8?

Here are the newly added features of 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.

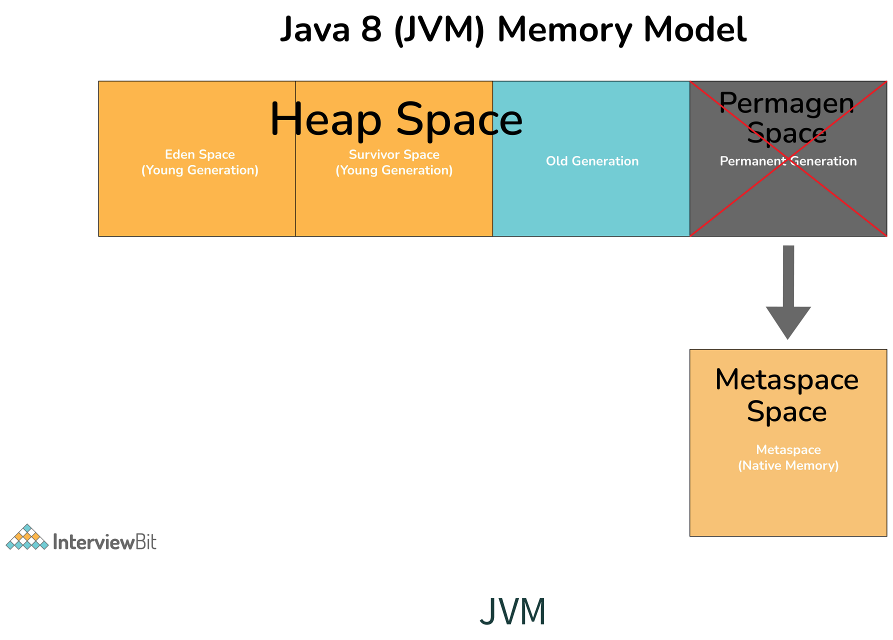
2. In which programming paradigm Java 8 falls?

- Object-oriented programming language.
- Functional programming language.
- Procedural programming language.
- Logic programming language

3. 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

4. What is MetaSpace? How does it differ from PermGen?


Java 8 (JVM) Memory Model
Heap Space
Eden Space (Young Generation)
Survivor Space (Young Generation)
Old Generation
Permagen Space
Permanent Generation
Metaspace Space
Metaspace (Native Memory)
InterviewBit
JVM

PremGen: MetaData information of classes was stored in PremGen (Permanent-Generation) memory type before Java 8. PremGen is fixed in size and cannot be dynamically resized. It was a contiguous Java Heap Memory.

MetaSpace: Java 8 stores the MetaData of classes in native memory called 'MetaSpace'. It is not a contiguous Heap Memory and hence can be grown dynamically which helps to overcome the size constraints. This improves the garbage collection, auto-tuning, and de-allocation of metadata.

5. What are functional or SAM interfaces?

Functional Interfaces are an interface with only one abstract method. Due to which it is also known as the Single Abstract Method (SAM) interface. It is known as a functional interface because it wraps a function as an interface or in other words a function is represented by a single abstract method of the interface.

Functional interfaces can have any number of default, static, and overridden methods. For declaring Functional Interfaces @FunctionalInterface annotation is optional to use. If this annotation is used for interfaces with more than one abstract method, it will generate a compiler error.

```
@FunctionalInterface // Annotation is optional
public interface Foo() {
// Default Method - Optional can be 0 or more
public default String HelloWorld() {
return "Hello World";
}
// Static Method - Optional can be 0 or more
public static String CustomMessage(String msg) {
return msg;
}
// Single Abstract Method
public void bar();
}

public class FooImplementation implements Foo {
// Default Method - Optional to Override
@Override
public default String HelloWorld() {
return "Hello Java 8";
}
// Method Override
@Override
public void bar() {
    System.out.println(“Hello World”);
}
}

public static void main(String[] args) {

FooImplementation fi = new FooImplementation();
System.out.println(fi.HelloWorld());
System.out.println(fi.CustomMessage(“Hi”));
fi.bar();
}
```

6. 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. E.g:

```
interface Parent {
public int parentMethod();
}
@FunctionalInterface // This cannot be FunctionalInterface
interface Child extends Parent {
public int childMethod();
// It will also extend the abstract method of the Parent Interface
// Hence it will have more than one abstract method
// And will give a compiler error
}
```

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. For eg:

```
interface Parent {
public void parentMethod(){
System.out.println("Hello");
}
}
@FunctionalInterface
interface Child extends Parent {
public int childMethod();
}
```

7. 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.

```
@FunctionalInterface // Annotation is optional
public interface Foo() {
// Default Method - Optional can be 0 or more
public default String HelloWorld() {
return "Hello World";
}
// Single Abstract Method
public void bar();
}
```

8. 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.

9. 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. Please refer to the java.util.function doc for other predefined functional interfaces and its description introduced in Java 8.

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

10. 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.

11. 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.

Java 8 Interview Questions for Experienced

12. What is the basic structure/syntax of a lambda expression?

```
FunctionalInterface fi = (String name) -> {
System.out.println("Hello "+name);
return "Hello "+name;
}
```

Lambda expression can be divided into three distinct parts as below:

1. List of Arguments/Params:

(String name)

A list of params is passed in () round brackets. It can have zero or more params. Declaring the type of parameter is optional and can be inferred for the context.

2. Arrow Token:

->
Arrow token is known as the lambda arrow operator. It is used to separate the parameters from the body, or it points the list of arguments to the body. 3. Expression/Body:

```
{
System.out.println("Hello "+name);
return "Hello "+name;
}
```

A body can have expressions or statements. {} curly braces are only required when there is more than one line. In one statement, the return type is the same as the return type of the statement. In other cases, the return type is either inferred by the return keyword or void if nothing is returned.

13. 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.

14. What is a type interface?

Type interface is available even in earlier versions of Java. It is used to infer the type of argument by the compiler at the compile time by looking at method invocation and corresponding declaration.

15. What are the types and common ways to use lambda expressions?

A lambda expression does not have any specific type by itself. A lambda expression receives type once it is assigned to a functional interface. That same lambda expression can be assigned to different functional interface types and can have a different type.

For eg consider expression `s -> s.isEmpty() :`

```
Predicate<String> stringPredicate = s -> s.isEmpty();
Predicate<List> listPredicate = s -> s.isEmpty();
Function<String, Boolean> func = s -> s.isEmpty();
Consumer<String> stringConsumer = s -> s.isEmpty();
```

Common ways to use the expression

Assignment to a functional Interface —> `Predicate<String> stringPredicate = s -> s.isEmpty();`
Can be passed as a parameter that has a functional type —> `stream.filter(s -> s.isEmpty())`
Returning it from a function —> `return s -> s.isEmpty()`
Casting it to a functional type —> `(Predicate<String>) s -> s.isEmpty()`

16. 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

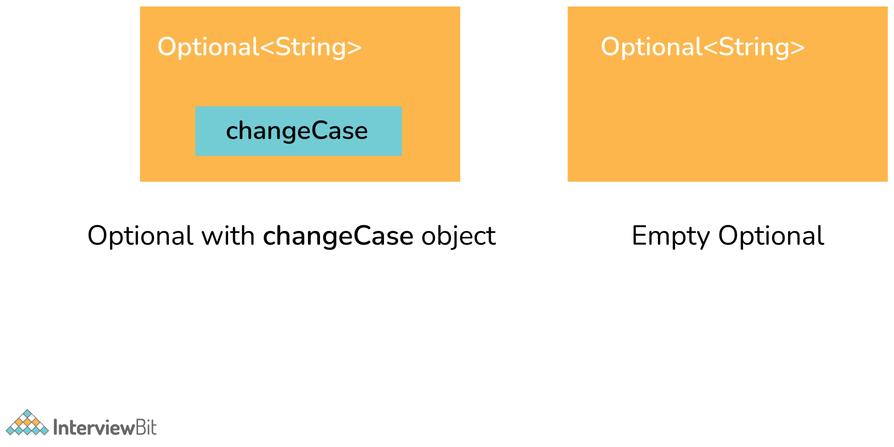
17. What does the String::ValueOf expression mean?

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

18. 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.

```
static Optional<String> changeCase(String word) {
if (name != null && word.startsWith("A")) {
   return Optional.of(word.toUpperCase());
  }
else {
return Optional.ofNullable(word); // someString can be null
}
}
```



Optional Class

19. What are the advantages of using the Optional class?

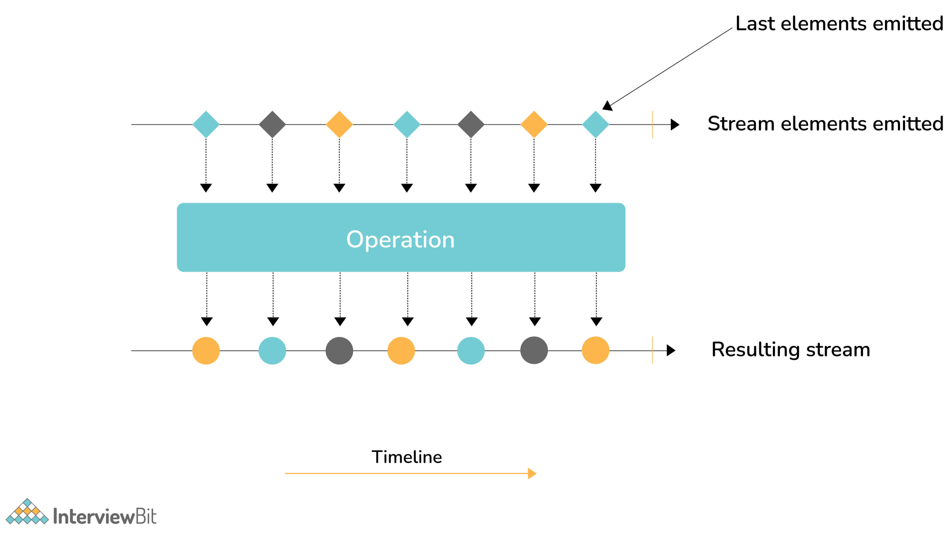
Below are the main advantage of using the Optional class:

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.

20. 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:

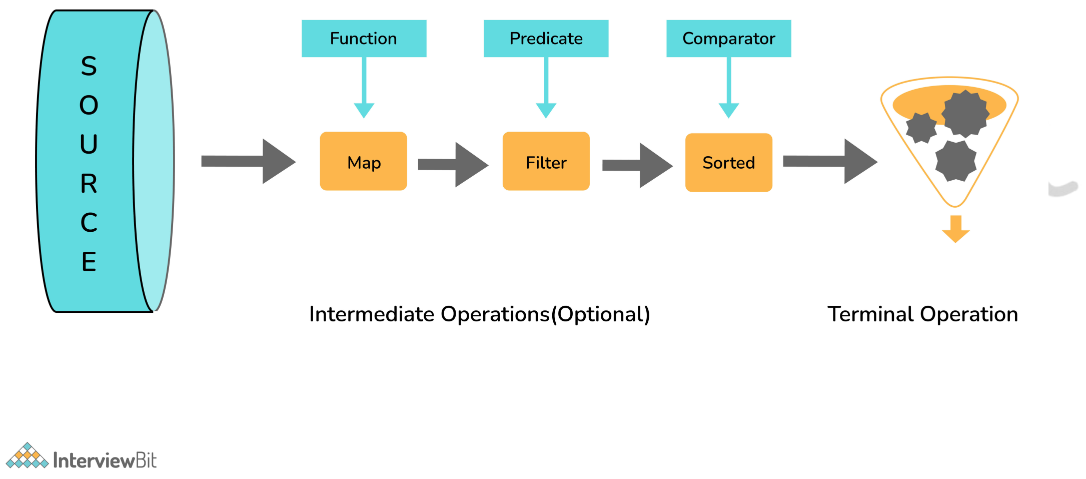


Java 8 Streams

21. 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



Components of Stream

22. 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.

23. 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
```

24. 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.

25. 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.

26. 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.

27. 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

28. 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

29. 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

31. Explain with example, LocalDate, LocalTime, and LocalDateTime APIs.

LocalDate

- Date with no time component
- Default format - yyyy-MM-dd (2020-02-20)
- LocalDate today = LocalDate.now(); // gives today's date
- LocalDate aDate = LocalDate.of(2011, 12, 30); //(year, month, date)

LocalTime

- Time with no date with nanosecond precision
- Default format - hh:mm:ss:zzz (12:06:03.015) nanosecond is optional
- LocalTime now = LocalTime.now(); // gives time now
- LocalTime aTime2 = LocalTime.of(18, 20, 30); // (hours, min, sec)

LocalDateTime

- Holds both Date and Time
- Default format - yyyy-MM-dd-HH-mm-ss.zzz (2020-02-20T12:06:03.015)
- LocalDateTime timestamp = LocalDateTime.now(); // gives timestamp now
- //(year, month, date, hours, min, sec)
- LocalDateTime dt1 = LocalDateTime.of(2011, 12, 30, 18, 20, 30);

32. Define Nashorn in Java 8

Nashorn is a JavaScript processing engine that is bundled with Java 8. It provides better compliance with ECMA (European Computer Manufacturers Association) normalized JavaScript specifications and better performance at run-time than older versions.

33. What is the use of JJS in Java 8?

As part of Java 8, JJS is a command-line tool that helps to execute the JavaScript code in the console. Below is the example of CLI commands:

```
JAVA>jjs
jjs> print("Hello, Java 8 - I am the new JJS!")
Hello, Java 8 - I am the new JJS!
jjs> quit()
>>
```

Conclusion

All in all, Java is a prevalent programming language securing the second rank in popularity in both TIOBE and PYPL programming language ranking. The world's leading tech giants like Twitter, LinkedIn, Amazon, PayPal, etc., use Java to build their web apps and backend web systems. Java is also one of the primary languages used to develop Android apps; an operating system backed and promoted by Google.

As of today, there are 1,751,661 questions around Java on StackOverflow and 123,776 Java public repositories on GitHub and continuously increasing. Considering Java 8 to be one of the most stable versions, there are immense career opportunities and scope in the same. Just understand the concepts, implement them and get ready for the interviews!

Additional Resources

[Practice Coding]

[Java Tutorials]

[Java Interview Questions]

Spring Boot Interview Questions

Hibernate Interview Questions

How To Become A Java Developer

Links to More Interview Questions

C Interview Questions	Php Interview Questions	C Sharp Interview Questions
Web Api Interview Questions	Hibernate Interview Questions	Node Js Interview Questions
Cpp Interview Questions	Oops Interview Questions	Devops Interview Questions
Machine Learning Interview Questions	Docker Interview Questions	Mysql Interview Questions
Css Interview Questions	Laravel Interview Questions	Asp Net Interview Questions
Django Interview Questions	Dot Net Interview Questions	Kubernetes Interview Questions
Operating System Interview Questions	React Native Interview Questions	Aws Interview Questions
Git Interview Questions	Java 8 Interview Questions	Mongodb Interview Questions
Dbms Interview Questions	Spring Boot Interview Questions	Power Bi Interview Questions
Pl Sql Interview Questions	Tableau Interview Questions	Linux Interview Questions
Ansible Interview Questions	Java Interview Questions	Jenkins Interview Questions