**Java 8 Interview Questions :**

The type of “Journal Dev” is java.lang.String. The type of “true” is Boolean. In the same way, what is the type of a Lambda Expression?  
The Type of a Lambda Expression is a [Functional Interface](https://www.journaldev.com/2763/java-8-functional-interfaces).

Example:- What is the type of the following Lambda Expression?

**() -> System.out.println("Hello World");**

This Lambda Expression does not have parameters and does return any results. So it’s type is “java.lang.Runnable” Functional Interface.

**When do we go for Java 8 Stream API? Why do we need to use Java 8 Stream API in our projects?**

When our Java project wants to perform the following operations, it’s better to use [Java 8 Stream](https://www.journaldev.com/2774/java-8-stream) API to get lot of benefits:

* When we want perform Database like Operations. For instance, we want perform groupby operation, orderby operation etc.
* When want to Perform operations Lazily.
* When we want to write Functional Style programming.
* When we want to perform Parallel Operations.
* When want to use Internal Iteration
* When we want to perform Pipelining operations.
* When we want to achieve better performance.

**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”.

### What is Type Inference? Is Type Inference available in older versions like Java 7 and Before 7 or it is available only in Java SE 8?

Type Inference means determining the Type by compiler at compile-time.

It is not new feature in Java SE 8. It is available in Java 7 and before Java 7 too.

**Before Java 7:-**  
Let us explore Java arrays. Define a String of Array with values as shown below:

String str[] = { "Java 7", "Java 8", "[Java 9](https://www.journaldev.com/13121/java-9-features-with-examples)" };

Here we have assigned some String values at right side, but not defined it’s type. Java Compiler automatically infers it’s type and creates a String of Array.

**Java 7:-**  
Oracle Corporation has introduced “Diamond Operator” new feature in Java SE 7 to avoid unnecessary Type definition in Generics.

Map<String,List<Customer>> customerInfoByCity = new HashMap<>();

Here we have not defined Type information at right side, simply defined Java SE 7’s Diamond Operator “”.

**Java SE 8:-**  
Oracle Corporation has enhanced this Type Inference concept a lot in Java SE 8. We use this concept to define Lambda Expressions, Functions, Method References etc.

ToIntBiFunction<Integer,Integer> add = (a,b) -> a + b;

Here Java Compiler observes the type definition available at left-side and determines the type of Lambda Expression parameters a and b as Integers.

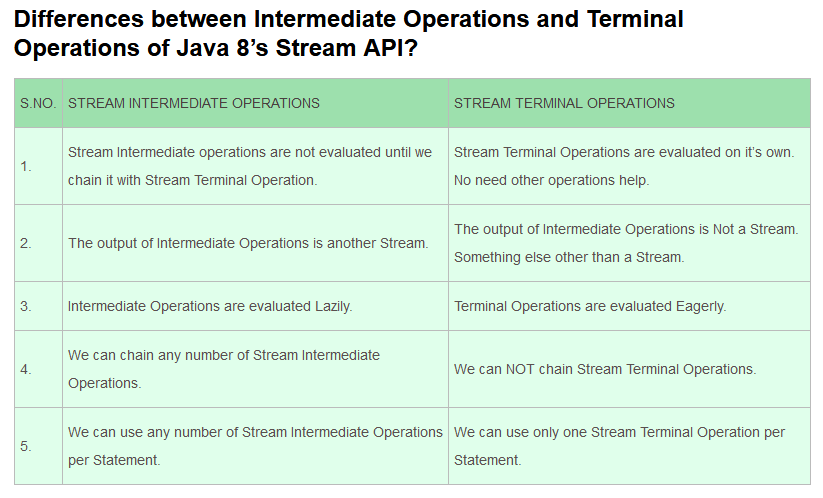
### What is Internal Iteration in Java SE 8?

Internal Iteration means “Iterating an Aggregated Object elements one by one internally by Java API”. Instead of Java Application do iteration externally, We ask Java API to do this job internally.

**When do we need to use Internal Iteration(forEach method)? When do we need to use External Iteration(Iterator,ListIterator,Enumaration)?**

We need to understand the situations to use either Internal Iteration or External Iteration.

* When we need more control over Iteration, we can use External Iteration.
* When we do NOT need more control over Iteration, we can use Internal Iteration.
* When we need to develop Highly Concurrency and Parallel applications and we , we should use Internal Iteration.



**Explain issues of Old Java Date API? What are the advantages of Java 8’s Date and Time API over Old Date API and Joda Time API?**

Java’s OLD Java Date API means Date API available before Java SE 8 that is Date, Calendar, SimpleDateFormat etc.

Java’s Old Date API has the following Issues or Drawbacks compare to Java 8’s Date and Time API and Joda Time API.

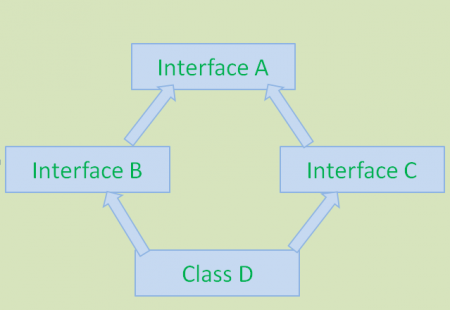
* Most of the API is deprecated.
* Less Readability.
* java.util.Date is Mutable and not Thread-Safe.
* java.text.SimpleDateFormat is not Thread-Safe.
* Less Performance.

Java SE 8’s Date and Time API has the following Advantages compare to Java’s OLD Date API.

* Very simple to use.
* Human Readable Syntax that is More Readability.
* All API is Thread-Safe.
* Better Performance.

### What is Diamond Problem in Inheritance? How Java 8 Solves this problem?

A Diamond Problem is a Multiple Inheritance problem. In Java, It occurs when a Class extends more than one Interface which have same method implementation (Default method).



This above diagram shows Diamond Problem. To avoid this problem, Java 7 and Earlier versions does not support methods implementation in interface and also doesn’t support Multiple Inheritance. Java 8 has introduced new feature: Default methods to support Multiple Inheritance with some limitations.

Sample Java SE 8 Code to show this Diamond Problem:

public interface A{

default void display() { //code goes here }

}

public interface B extends A{ }

public interface C extends A{ }

public class D implements B,C{ }

In the above code snippet, class D gives compiltime errors because Java Compiler will get bit confusion about which display() has to provide in class D. Class D inherits display() method from both interfaces B and C. To solve this problem, Java SE 8 has given the following remedy:

public interface A{

default void display() { //code goes here }

}

public interface B extends A{ }

public interface C extends A{ }

public class D implements B,C{

void display() {

B.super.display();

}

}

This **B.super.display();** will solve this Diamond Problem.