What is Java8 -

The latest Java version contains important enhancements to improve performance, stability and security of the Java applications that run on your machine

What are Lambda expressions -

Lambda expressions are anonymous functions which are used to instantiate interfaces with single method

Replace more verbose class declarations

Significantly reduces the amount of code and makes it lot more readable

One of the goals of Lambda expression is to eliminate unnecessary syntax

It doesn't provide any performance benefit, as the underlying functionality remains exactly the same

Where to use Lambda Expressions -

Lambda expressions can only appear in places where they will be assigned to a variable whose type is a functional interface

Functional Interfaces -

A Functional Interface has a single abstract method (not inherited from object class i.e., equals)

Functional Interfaces included with Java runtime are -

Runnable, Callable, Comparator, TimerTask

Prior to Java 8 they are known as "Single Abstract Method" (SAM) Types

If an interface declares an abstract method overriding one of the public methods of java.lang.Object, that also does not count toward the interface's abstract method count since any implementation of the interface will have an implementation from java.lang.Object or elsewhere

Descriptions -

```
Runnable r =() -> System.out.println("Runnable inner class @JAVA8 : Hello World!\n");
```

() is the Method Signature - here it’s a no argument method, otherwise simply needs to give argument's names and no need to declare their types as for a single abstract method the datatypes are already known

-> is the Lambda operator

... is the Method implementation

To implement a method body with multiple statements, the codes have to be wrapped within a pair of curly braces.

Lambda expression can significantly reduce the amount of code you need to write and the number of custom classes you have to create and maintain. If you are implementing an interface for one-time use, it doesn't always make sense for creating another code file or another named class.

foreach method -

In Java 8, we can use lambda expression to traverse collection of items - mainly lists and sets - implements iterable.

foreach(Consumer<>) - here Consumer is a Functional interface

default void forEach(Consumer<? super T> action)

accept(T t)

Predicate FI -

In addition to the new lambda syntax, Java 8 added a number of new functional interfaces

Predicate FI has a single abstract method named test(T t) , which can be used to wrap up conditional processing and make conditional code much cleaner

java.util.function package contains new FIs of Java 8

Method References -

A method reference gives us a way of naming a method that we want to call instead of calling it directly

Goal is to make the code more concise and more readable

Default Methods -

Prior to Java 8, interfaces can only contain abstract methods and constant declarations, and we cannot provide fully implemented methods that would be inheritable.

Java 8 has significantly added support for default and static methods to interfaces

A default method is an instance method defined in an interface whose method header begins with the default keyword; it also provides a code body

A static method is a method that's associated with the class in which it's defined, rather than with any object created from that class. Every instance of the class shares the static methods of the class

specify that a method definition in an interface is a default method with the default keyword at the beginning of the method signature

JAVA 8 Stream -

In Java 8 we have a new way of managing, traversing and aggregating collections with the Stream APIs. A Collection based stream is not like an i\o stream, instead it’s a new way of working with data instead dealing with each item individually.

A stream represents a sequence of elements and supports different kind of operations to perform computations upon those elements

When we use stream we don’t have to worry about the details of looping or traversing. We need to create a stream object directly from a collection and then we can do all sorts of things with it including traversing, filtering and aggregating its values.

In Java 8 there are two kinds of collection streams known as Sequential stream and Parallel stream.

In Java 8 Stream API is designed to help us manage Collections data i.e., object that are members of the Collection framework such as ArrayList or HashMap, but we can also create steams directly from array.

Till now we have seen how to use Streams to iterate over collections, but we can also use Streams to calculate Sums, Averages, Counts and so on. To do such kind of operations it is important to know the nature of parallel streams.

mapTo* - takes a complex object and extracts a simple primitive value from it

To avoid divide by zero problem in case of averaging, we can use a type of Optional variable.

JAVA 8 Date and Time APIs -

Java 8 includes a complete new api for managing date and time values. The classes that actually hold the data in new apis are all immutable and thread-safe. So we don’t have to worry about passing the object around multi-threading environment and while using them with parallel stream, everything will work perfectly. All classes of this new API are members of the package java.time.

Duration.between() api takes two Temporal values and Instant class is a sub-class of Temporal.

LocalDate - uses 1 for Jan, 12 for Dec unlike older date time apis that uses 0 for Jan, 1 for Feb.

ZonedDateTime is immutable and thread-safe, same as LocalDateTime.

Nashorn -

Nashorn is a JavaScript engine developed in the Java programming language

Nashorn's goal is to implement a lightweight high-performance JavaScript runtime in Java with a native JVM

Java 8 includes a brand new library, a javascript engine named nashorn. It allows to code in javascript instead of java using the dynamic coding capabilities that language supports. Nashorn is a completely new engine, it replaces the older rainho engine. It can be used from command line as well as java code files.

Using the new nashorn engine, we can also build javascript into our java code. We can write complete code blocks using javascript and then execute them in the context of java class.

We can execute any sorts of javascript code in this manner, but as we can see creating a script inside java is very hard to maintian. So it’s a better pracitce to put all the javascript code into a separate file.

The Nashorn javascript engine can either be used programmatically from java programs or by utilizing the command line tool jjs, which is located in $JAVA_HOME/bin

Join method and StringJoiner class -

join - static method of Sting class

StringJoiner is used to construct a sequence of characters separated by a delimiter and optionally starting with a supplied prefix and ending with a supplied suffix

new String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter

String.join is an abbreivated call in a sense but if you look under the covers it delegates to StringJoiner

String composed of copies of the CharSequence elements joined together with a copy of the specified delimiter

ALL THE BEST