Java 8 Features

**Java Streams** :

Java streams are similar to sreams of data that I did in GCP or big data analytics. Data can either be batched or streamed.A batched data arrives at certain time and in batch . Stream just follows along, never gets stored.

*In functional programming, a monad is a structure that represents computations defined as sequences of steps. A type with a monad structure defines what it means to chain operations, or nest functions of that type together.*

Most stream operations accept some kind of lambda expression parameter, a functional interface specifying the exact behavior of the operation. Most of those operations must be both***non-interfering*and*stateless***. What does that mean?

**A function is non-interfering** when it does not modify the underlying data source of the stream, e.g. in the above example no lambda expression does modify myList by adding or removing elements from the collection.

**A function is stateless** when the execution of the operation is deterministic, e.g. in the above example no lambda expression depends on any mutable variables or states from the outer scope which might change during execution.

Difference between filter and map?

Filter return truer or false based on certain condition and lets us know if the item added to list or not

Map performs a function on collection element

**Functional References** :

Functional references are used within lambda expression wherein we can refer to a function of a class/ object.

Eg :

list.forEach(System.out::println);

**Functional Interfaces** :

Functional Interfaces have been introduced in java8. This is an important step towards functional programming.

A functional Interface can have 1 and only 1 adstract method.

It can have as many default methods as required

Usually functional interfaces definition is done in line using Lambda expression

Functional Interfaces can override Object methods.

In java8 the functional Interfaces were extended to Runnable, ActionListener(actionPerformed) etc.

Usage We can use Lambda expressions to instantiate the functional interfaces and avoid bulky inner class implemenations.

Question - Isnt it voilating code reusability if we are providing implemenation at runtime

Advantages :

Forces developer to go down the functional route

Forces developer to have compact and consise interfaces which only do 1 job/task(S of Solid)

**Default Methods in Interfaces** :

Since java 8 bwe can have default methods other than abstracty methods in Interfaces.

Default methods are used so that functionality can be added to existing interfaces / implementations without the existing implementations to implement those.

Helped with the evolution of existing interfaces such as List etc where foreach was added so that the interfaces remain backward compatible and the new implementation can be used as well

**Date/Time API Changes** :

**Lambdas** :

Lambdas make code consise and are very much used with Streams , Functional Interfaces.

Predicates makes a lot of usage for Lambdas

**Predicate** :

In mathematics, a predicate is commonly understood to be a boolean-valued function 'P: X? {true, false}', called the predicate on X. Informally, a strong.

It can be thought of as an operator or function that returns a value that is either true or false.

Since Predicate is a functional interface we can pass lambda expression wherever predicate is expected.

Create a predicate and use it with Stream to filter, return and collect

**Changes to comparator** :

**Java Boxed Streams** :

List<Integer> ints = IntStream.of(1,2,3,4,5)

.boxed()

.collect(Collectors.toList());

If primitives stream needs to be converted to Collection we need to box them

How does Foreach work in Java8?

What and how does predicate work in Java8?