main features of Java 8 which make Java as a functional programming language

Lambda expressions, functional interfaces and Stream API are the three main features of Java 8 which enables developers to write functional style of programming in Java also.

lambda expressions

Lambda Expressions can be defined as methods without names i.e anonymous functions. Like methods, they also have parameters, a body, a return type and possible list of exceptions that can be thrown. But unlike methods, neither they have names nor they are associated with any particular class.

Lambda expressions are used where an instance of functional interface is expected. Before Java 8, anonymous inner classes are used for this purpose. After Java 8, you can use lambda expressions to implement functional interfaces.

Below code shows how Comparator interface is implemented using anonymous inner class before Java 8.

Comparator<Student> idComparator = new Comparator<Student>() {

@Override

public int compare(Student s1, Student s2) {

return s1.getID()-s2.getID();

}

};

With java8-

Comparator<Student> idComparator = (Student s1, Student s2) -> s1.getID()-s2.getID();

Can we use non-final local variables inside a lambda expression?

No. Only final local variables are allowed to use inside a lambda expressions just like anonymous inner classes.

Functional interface

Functional interfaces are the interfaces which has exactly one abstract method. Functional interfaces provide only one functionality to implement.

There were functional interfaces exist before Java 8. It is not like that they are the whole new concept introduced only in Java 8. Runnable, ActionListener, Callable and Comaprator are some old functional interfaces which exist even before Java 8.

Predicate:

Predicate is a functional interface which represents a boolean operation which takes one argument.

Method references

Java 8 method references can be defined as shortened versions of lambda expressions calling a specific method. Method references are the easiest way to refer a method than the lambdas calling a specific method. Method references will enhance the readability of your code.

Default & Static methods of an interface

Default methods of an interface are the concrete methods for which implementing classes need not to give implementation. They inherit default implementation.

Default methods are introduced to add extra features to current interfaces without disrupting their existing implementations. For example, stream() is a default method which is added to Collection interface in Java 8. If stream() would have been added as abstract method, then all classes implementing Collection interface must have implemented stream() method which may have irritated existing users. To overcome such issues, default methods are introduced to interfaces from Java 8.

Static:

Java API developers have followed the pattern of supplying an utility class along with an interface to perform basic operations on such objects.

For example, Collection and Collections. Collection is an interface and Collections is an utility class containing only static methods which operate on Collection objects.

But from Java 8, they have break this pattern by introducing static methods to interfaces. With the introduction of static methods to interface, such utility classes will disappear gradually and methods to perform basic operations will be kept as static methods in interface itself.

What are streams? Why they are introduced?

streams can be defined as operations on data. They are the sequence of elements from a source which support data processing operations. Using Java 8 Streams, you can write most complex data processing queries without much difficulties.

Almost every Java application use Collections API to store and process the data. Despite being the most used Java API, it is not easy to write the code for even some common data processing operations like filtering, finding, matching, sorting, mapping etc using Collections API . So, there needed Next-Gen API to process the data. So Java API designers have come with Java 8 Streams API to write more complex data processing operations with much of ease.

The operations which return stream themselves are called intermediate operations. For example – filter(), distinct(), sorted() etc.

The operations which return other than stream are called terminal operations. count(). min(), max() are some terminal operations.

difference between map() and flatMap()?

Java 8 map() and flatMap() are two important methods of java.util.stream.Stream interface used for transformation or mapping operations. Both are intermediate operations. The only difference is that map() takes Stream<T> as input and return Stream<R> where as flatMap() takes Stream<Stream<T> as input and return Stream<R> i.e flatmap() removes extra layer of nesting around input values.

Three important classes of Java 8 Date and Time API?

java.time.LocalDate, java.time.LocalTime and java.time.LocalDateTime