**Java8**

Java 8 is a major release in 2014

**Difference between java 7 and java 8**

Java 7 brings JVM support for dynamically-typed languages

Java 8 brings the most anticipated feature for the programming language called Lambda Expressions, a new language feature which allows users to code local functions as method arguments.

Users can catch multiple exception types in one catch block which could be impossible before JDK 7.  
  
Java 8 brings its own new specialized API for Date and Time manipulation.  
  
Small improvements have been made to simplify common programming tasks such as automated management of resources, flipping of string objects, better handling exceptions, etc.

New and improved JavaScript engine, Nashorn that enables developers to run the JVM script. The idea was to introduce a simple JavaScript runtime in the native JVM language.

**Features of java 8**

**Functional Interfaces And Lambda Expressions**

 Functional Interface is an interface that has exactly one abstract method. To designate an interface as a Functional Interface, we don’t need to use the @FunctionalInterface annotation.

The @FunctionalInterface annotation prevents abstract methods from being accidentally added to functional interfaces.

An anonymous function may be defined as a Lambda Expression (or function) (a function with no name and an identifier).

**forEach() Method In Iterable Interface**

Java.lang interface now supports a “forEach” function. Iterable that can iterate over the collection’s items. The Iterable interface has a default method called “forEach.” Collection classes use it to iterate items, which extends the Iterable interface.

List<String> subList = new ArrayList<String>();

subList.add("C");

subList.add("P");

subList.add("B");

subList.add("L");

subList.add("T");

subList.forEach(sub -> System.out.println(sub));

**Optional Class**

the “java.util” package included an optional class. The public final class “Optional” is used to handle NullPointerException in a Java program. You may give other code or values to execute using Optional.

String[] str = new String[10];

Optional<String>checkNull =

Optional.ofNullable(str[5]);

if (checkNull.isPresent()) {

String word = str[5].toLowerCase();

System.out.print(str);

} else

System.out.println("string is null");

}

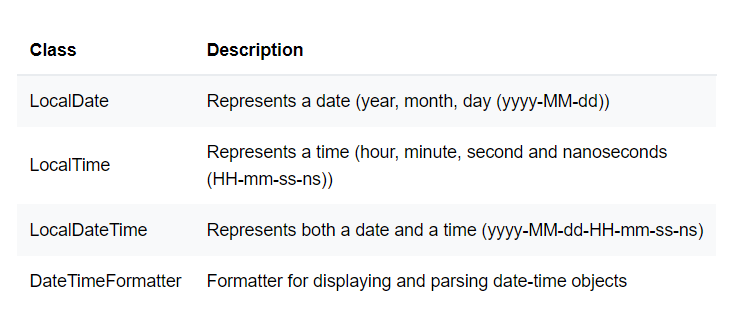
**Java Method References**

Java provides a new feature called method reference in Java 8. Method reference is used to refer method of functional interface. It is compact and easy form of lambda expression.

**Default And Static Methods In Interfaces**

you may add non-abstract methods to interfaces, allowing you to create interfaces with method implementation. To construct interfaces with method implementation, use the Default and Static keywords**.**

**Java Date Time API**

****

**Why need default methods**

The reason we have default methods in interfaces is to allow the developers to add new methods to the interfaces without affecting the classes that implements these interfaces.

**Static methods**

We can define utility methods in interface, if all are static methods then we can keep in interface rather than class

static method helps us in **providing security** by not allowing implementation classes to override them.

**Functional Interface**

 A Functional Interface is an interface that has exactly one abstract method,

If an interface annotated with @FunctionalInterface annotation. It is also a functional inteface

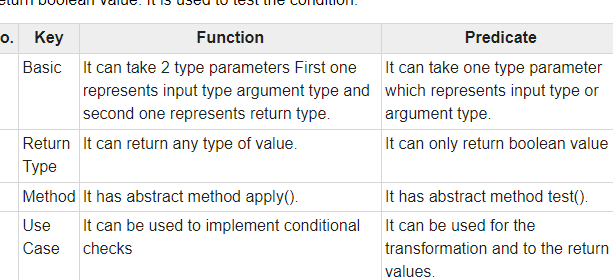
**Streams and different from collection**

Stream API is used to process collections of objects. A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

A **Collection**is an data structure, which holds all the values that the data structure currently has. Every element in the Collection has to be computed before we add it to the Collection. Operations such as searching,sorting, insertion, manipulation, and deletion can be performed on a Collection.

 A stream is a sequence of objects that supports various methods which can be pipelined to produce the desired result.

**Difference Between Function And Predicate**



**Consumer Functional Interface**

Represents an operation that accepts a single input argument and returns no result

method is accept(Object)..

**Function**

 It represents a function which takes in one argument and produces a result. Hence this functional interface takes in 2 generics namely as follows:

apply()

  Function<Integer, Double> half = a -> a / 2.0;

        // Applying the function to get the result

        System.out.println(half.apply(10));

**What is The use of Optional Class**

To verify whether the string is null in this application, we utilize the Optional class’s “ofNullable” attribute. If it is, the user receives the relevant message