**Java 8 Features:**

1) Support for lambda exression in APIs

This is the most anticipated feature of the update. The introduction of Lambda Expressions allows us to write functional style code.

The Java Lambda syntax is:

(Argument-list) -> {body}

Here the argument list can be null or empty. The arrow(->) is used to link the arguments to the expression contained in the body.

@FunctionalInterface //optional

interface MyInterface{

public void myMethod();

}

public class LambdaExpressionEx{

public static void main(String args[])

{

int variable = 10;

//implemenmtation with lambda expression

MyInterface myInterface = ()->{

System.out.println("variable = "+variable);

};

myInterface.myMethod();

}

}

2) Functional Interface:

An interface with a single abstract method is a Functional Interface. All such interfaces must have *@FuntionalInterface* annotation. The implementation of any such method may be treated as a lambda expression. This, like Lambda expressions, enables Functional Programming.

import java.util.arrays;

import java.util.List;

import java.util.function.Predicate

public class FuncInterfaceEx {

public static void main(String args[]) {

List<Integer> intList = Arrays.asList(1,2,3,4,5,6,7,8,9);

System.out.println("Printing the whole list : ");

//Passing n as parameter

eval(intList, n-> true);

System.out.println("Printing even numbers : ");

eval(intList, n-> n%2 == 0);

}

public static void eval(List<Integer> intList, Predicate<Integer> predicate)

{

for(Integer n : intList)

{

if(predicate.test(n)){

System.out.println(n + " ");

}

}

}

}

#### 3)Default and static methods in Interfaces:

With Java 8’s introduction of default methods, the new method can be added to an interface without the need to implement them in all classes.  
This has basically given us a way to add new code without breaking the existing one.  
  
Similarly, even static methods do not need to be implemented in all the classes. The only difference is that as static methods, they cannot be overridden.

interface MyInterface{

default void newDefaultMethod(){

System.out.println("New default method");

}

static void newStaticMethod(){

System.out.println("New static method");

}

void existingMethod(String str);

}

public class Example implements MyInterface {

//existing method implementation

public void existingMethod(String str){

System.out.println("String : "+str);

}

//There is no need to implement the other methods

public static void main(String args[]) {

Example obj = new Example();

obj.newDefaultMethod();

MyInterface.newStaticMethod();

obj.existingMethod("This is the new java 8 feature");

}

}

4) Collection API Improvements forEach:

import java.util.ArrayList;

import java.util.List;

public class ForEachEx {

public static void main(String args[]) {

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

gamesList.add("Football");

gamesList.add("Cricket");

gamesList.add("Baseball");

System.out.println("----------------Passing lambda expression----------------------")

gamesList.forEach(games -> System.out.println(games));

}

}

5) Method Reference

interface Sayable{

    void say();

}

public class MethodReference {

    public static void saySomething(){

        System.out.println("Hello, this is static method.");

    }

    public static void main(String[] args) {

        // Referring static method

        Sayable sayable = MethodReference::saySomething;

        // Calling interface method

        sayable.say();

    }

}

6) Streams

With Java 8, Collection interface has two methods to generate a Stream.

* stream() − Returns a sequential stream considering collection as its source.
* parallelStream() − Returns a parallel Stream considering collection as its source.

List<String> strings = Arrays.asList("abc", "", "bc", "efg", "abcd","", "jkl");

List<String> filtered = strings.stream().filter(string -> !string.isEmpty()).collect(Collectors.toList());