**Java 8**

[**https://www.youtube.com/watch?v=j9nj5dTo54Q**](https://www.youtube.com/watch?v=j9nj5dTo54Q)

[**https://www.youtube.com/watch?v=f4QZ12wMQO8&list=PLd3UqWTnYXOk0y\_HFp2r1eMW\_lhZ7yP4w**](https://www.youtube.com/watch?v=f4QZ12wMQO8&list=PLd3UqWTnYXOk0y_HFp2r1eMW_lhZ7yP4w)

[**https://www.youtube.com/watch?v=gpIUfj3KaOc&list=PLqq-6Pq4lTTa9YGfyhyW2CqdtW9RtY-I3**](https://www.youtube.com/watch?v=gpIUfj3KaOc&list=PLqq-6Pq4lTTa9YGfyhyW2CqdtW9RtY-I3)

[**https://javabrains.thinkific.com/courses/java-8-lambda-basics**](https://javabrains.thinkific.com/courses/java-8-lambda-basics)

**Objective of Java 8:**

* To simplify programming
* To utilize functional programming benefits in Java
* To enable parallel programming in Java

1. **Lambda Expressions**

Lambda expressions which are functions don’t belong to a class or methods of a class.

These are just functions in isolation.

aBlockOfCode = public void perform() {

syso(“Hello World!”));

}

In Java 8 using Lambda:

aBlockOfCode = () -> {syso(“Hello World!”);}

1. **Functional Interfaces**

**It needs to have one abstract method.**

**What above line means is?** In Java 8 we have interfaces with implementation methods. But before Java 8, we have interface with only Abstract methods.

An Interface can have any default methods, but should have only 1 Abstract method is called a Functional Interface. This type of Interface is used for declaring Lambda types.

In Java 8, we have new way to represent an Interface as **Functional Interface.**

@FunctionalInterface

1. **Default methods in Interfaces**
2. **Static methods also inside Interfaces**
3. **Predicate – Predefined Functional Interface**
4. **Function – Predefined Functional Interface**
5. **Consumer – Predefined Functional Interface**
6. **Method reference and constructor reference by double colon (::) operator**

**Example:**

Public class MethodReferenceExample1 {

p.s.v.m() {

Thread t = new Thread (() -> printMessage());

}

public static void printMessage() {

syso(“Hello”);

}

}

**This is normal way of method reference calling but:**

Thread t = new Thread(MethodReferenceExample1 :: printMessage);

// MethodReferenceExample1 :: printMessage == (() -> printMessage()

p -> System.out.println(p) == System.out::Println; (The method which has been called will have type of Consumer<Person> it is)

1. **Stream API**

A sequence of elements supporting sequential and parallel aggregate operations.

A simple stream API for first scenario:

people.stream()

.filter(p -> p.getLastName.startsWith(“C”))

.forEach(p-System.out.println(p.getFirstName()));

**10) Date and Time API (Joda API)**

**What are lambda expressions?**

They are anonymous methods.

**Why Lambda?**

* Enables Funtional Programming
* Readable and concise code
* Easier-to-use APIs and libraries
* Enable support for Parallel Processing

**The foreach iteration:**

For-in and for loop iterators are external iterators i.e., you are managing the iteration.

Now with Java 8, we just need to define the for-each on collection. At runtime, it will automatically executes the iteration on collection.

people.forEach(p->System.out.println(p));

(or)

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