While lambdas are the most prominent addition to Java 8, there are many other new features, such as **functional interfaces, virtual methods, class and method references, new time and date API, JavaScript support**, and so on.

Lambda Expression = **input arguments -> body**In Java, lambda expression is **SAM (Single Abstract Method) type i.**e a interface with Single method.

As Java is a strongly typed language, it is usually mandatory to declare types; otherwise, the compiler laughs at us. Note, however, that we omitted types when declaring the above lambda expressions. Left side part will be taken as type.

Another Example is List<String> list = new ArrayList< >(); // Type inference

Each lambda expression has two parts separated by an arrow token: the left hand side is our method arguments, and the right hand side is what we do with those arguments (that is, applying business logic). **The body can either be an expression or block of code, returning a result or void**.

**What is Functional Interface?** An interface which has exactly one abstract method and annotated with @FunctionalInterface.

**What will happen if 2 abstract methods are there in Functional Interface?**  Compile Time Error.

**What will happen if 1 abstract method and 1 or more default method are there in Functional Interface?** No Issues. It can act as Functional Interface.

**Name any java Pre-built functional interface which can be used to ease our life?** There are many functional interface written under “java.util.function”

**java.util.function.Predicate:** We need a function for checking a condition. A Predicate is one such function accepting a single argument to evaluate to a boolean result.

@FunctionalInterface

public interface Predicate<T> {

boolean test(T t);

}

### A java.util.function.Function return any result by working on a single input argument.

@FunctionalInterface  
public interface Function<T, R> {

R apply(T t);

}

The java.util.function.**Consumer** accepts a single argument but does not return any result:  
**Example** of Consumer is “list.stream().forEach(System.out::println);” Here forEach is Consumer. It accepts argument and prints result and don’t return anything.

@FunctionalInterface  
public interface Consumer<T> {  
 void accept(T t);  
}

The java.util.function.**Supplier**, as the name suggests, supplies us with a result; here’s the method signature:

@FunctionalInterface  
public interface Supplier<T> {

T get();

}

Virtual method or Default method

<http://zeroturnaround.com/rebellabs/java-8-explained-default-methods/>