**Lambda**: Lambda expression facilitates functional programming, and simplifies the development a lot.

Syntax:

A lambda expression is characterized by the following syntax −

parameter -> expression body

Following are the important characteristics of a lambda expression −

* **Optional type declaration** − No need to declare the type of a parameter. The compiler can inference the same from the value of the parameter.
* **Optional parenthesis around parameter** − No need to declare a single parameter in parenthesis. For multiple parameters, parentheses are required.
* **Optional curly braces** − No need to use curly braces in expression body if the body contains a single statement. The body of the lambda expression can contain zero, one or more statements.
* **Optional return keyword** − The compiler automatically returns the value if the body has a single expression to return the value. Curly braces are required to indicate that expression returns a value.

Example:

n -> n % 2 != 0;

(char c) -> c == 'y';

(x, y) -> x + y;

(int a, int b) -> a \* a + b \* b;

() -> 42-no paramenters are given it will return 42

() -> { return 3.14 };

() -> { System.out.println("Hello World!"); };

(String s) -> { System.out.println(s); };->prints the string to the main output and returns void.

Lambda expressions are used primarily to define inline implementation of a functional interface

A Java lambda expression is thus a function which can be created without belonging to any class.

A lambda expression can be passed around as if it was an object and executed on demand.

The type of the lambda expression would be a *function;* but in Java, the lambda expressions are represented as objects, and so they must be bound to a particular object type known as a functional interface. This is called the *target type*.

Since a functional interface can only have a single abstract method, the types of the lambda expression parameters must correspond to the parameters in that method, and the type of the lambda body must correspond to the return type of this method. Additionally, any exceptions thrown in the lambda body must be allowed by the throws clause of this method in the functional interface.

The following are examples of lambda expressions bound to a target type:

Predicate<Integer> isOdd = n -> n % 2 != 0;

BinaryOperator<Integer> sum = (x, y) -> x + y;

Callable<Integer> callMe = () -> 42;

Block<String> printer -> (String s) -> { System.out.println(s); };

Runnable runner = () -> { System.out.println("Hello World!"); }