A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

Syntax

1. parameter -> expression

2. (parameter1,parameter2) -> expression

3. (parameter1,parameter2) -> { code block }

Example

import java.util.ArrayList;

public class Main {

public static void main(String[] args) {

ArrayList<Integer> numbers = new ArrayList<Integer>();

numbers.add(5);

numbers.add(9);

numbers.add(8);

numbers.add(1);

numbers.forEach( (n) -> { System.out.println(n); } );

}

}

Lambda expression provides implementation of functional interface. An interface which has only one abstract method is called functional interface. Java provides an anotation @FunctionalInterface, which is used to declare an interface as functional interface.

Comparison with ancestor

In case of Functional interface, before lambda expression, Anonymous inner class was used. Which requires object creation and method definition.

Which replaced by Lambda expression in one line.

Runnable interface is a functional Interface.

Lamda Expression in Multithreading :

Runnable Interface is a functional interface where run() is an abstract method, so Lambda expression can help here to create threads.

Runnable task = () ->

        {

            // Used to set custom name to the current thread

            Thread.currentThread().setName("myThread");

            System.out.println(

                Thread.currentThread().getName()

                + " is running");

        };

The above code replaces

class mytask implements Runnable {

mytask() { }

public void run() {

Thread.currentThread().setName(“one”);

Thread.currentThread().getName()

                + " is running");

}

public static void main(String[] args) {

mytask task=new mytask();

Thread thread=new Thread(task);

}