**What is Lambda Expression in Java 8?**

Lambda expression facilitates the functional programming and simplifies the development.

Lambda expression is a shorter way of writing an implementation of a single abstract method interface. Lambda expressions are used to create anonymous functions.

**Syntax:** (parameters) -> expression or (parameters) -> {statements ;}

Java 8 provides support for lambda expressions only with functional interfaces.

We will implement the Runnable interface with anonymous class and with lambda expression.

**private** **static** **void** simpleConceptWithRunnable() {

// boilerplate code which needs to be written everytime

Runnable runnable = **new** Runnable() {

@Override

**public** **void** run() {

System.***out***.println("Inside annonymos inner class");

}

};

Runnable runnable2 = () -> { // Lambda expression

System.***out***.println("Inside lambda expression");

};

Thread thread = **new** Thread(runnable);

thread.start();

thread = **new** Thread(runnable2);

thread.start();

}

We wrote a boilerplate code of creating an anonymous class and override the run method. Lambda takes care of these things and gives us a simple syntax to create a anonymous class. Lambdas are more than anonymous classes.

1. **No Parameters –**

Runnable runnable2 = () -> System.out.println("Inside lambda expression");

**2. Single Parameter –**

interface SingleParam {

public void print(String param);

}

SingleParam singleParam = (param) ->

{System.out.printf("Hello s",param);};

singleParam.print("saurabh");

**3. Multiple parameters –**

interface MultipleParam {

public void print(String param1, String param2);

}

MultipleParam multipleParam = (param1, param2) ->

System.out.printf("param1 : %s, param2: %s\n", param1, param2);

multipleParams.print("OnlyFullstack", "Development");

**Lambda vs Anonymous class in Java 8**

**1) Syntax**

Lambda expressions looks neat as compared to Anonymous Inner Class (AIC)

**1. Syntax**  
**Anonymous class:**

package com.onlyfullstack;

public class LambdaVsAnonymousClass {

public static void main(String[] args) {

Runnable runnable = new Runnable() {

@Override

public void run() {

System.out.println("Anonymous class");

}

};

Thread thread = new Thread(runnable);

thread.start();

}

}

**Lambda:**

package com.onlyfullstack;

public class LambdaVsAnonymousClass {

public static void main(String[] args) {

Runnable runnable = () -> System.out.println("Lambda Expression");

Thread thread = new Thread(runnable);

thread.start();

}

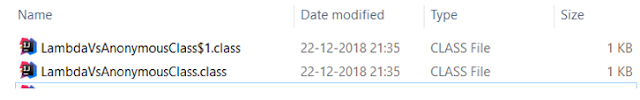
}

**2. Implementation**

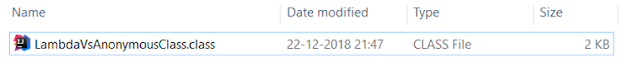
**Anonymous class** can be used to implement any interface with any number of abstract methods.

**Lambda Expression** will only work with SAM (Single Abstract Method) types. That is interfaces with only a single abstract method which is also called as Functional Interface. It would fail as soon as our interface contains more than 1 abstract method.

**3. Compilation**  
**Anonymous class:**Java creates two class files for LambdaVsAnonymousClass java file as  
LambdaVsAnonymousClass.class     – contains the main program  
LambdaVsAnonymousClass$1.class – contains an anonymous class

[](https://1.bp.blogspot.com/-ZuYFmoSq8gQ/XELpVm-qaqI/AAAAAAAAAQ8/zFbVHWRHD00k88jlurggF_M2yyWQNt_NQCLcBGAs/s1600/Untitled.png)

**Lambda Expression:**With Lambda expression compiler will create only 1 class file as below.

[](https://1.bp.blogspot.com/-yE6tXdPMOTc/XELpzBdGmQI/AAAAAAAAARE/LblQCkRTD2ArVIoj28nOUL66BO3_4n1TACLcBGAs/s1600/Untitled1.png)

So Java will create the new class file for each Anonymous class used.

**4. Performance**

At runtime anonymous inner classes require class loading, memory allocation and object initialization and invocation of a non-static method while lambda expression is pure compile time activity and don’t incur extra cost during runtime. So performance of lambda expression is better as compare to anonymous inner classes.\*\*

[**https://javaconceptoftheday.com/interface-vs-abstract-class-after-java-8/**](https://javaconceptoftheday.com/interface-vs-abstract-class-after-java-8/)