**JAVA 1.8 NEW Features: -**

1. lambda Expressions
2. Functional Interfaces
3. Default methods in Interface
4. Static Methods in Interfaces.
5. Predicate --- Pre defined functional Interface
6. Function --- Pre defined functional Interface
7. Consumer --- Pre defined functional Interface
8. Supplier
9. Method reference & constructor reference by Double colon(::) operator.
10. Stream API
11. Date & Time API ( Joda API)

**What is the objective of Java 1.8 New Features: -**

1. To simplify programming.
2. To utilize functional programming benefits in JAVA.
3. To enable parallel programming.

**Lambda Expressions: -**

LISP is the first programming language which uses lambda expressions.

To enable functional programming in JAVA

Write more readable, maintainable and concise code

To use API very easily and effectively

To enable parallel processing also

**What is Lambda Expression?**

It is anonymous function with not having name, not having modifiers, not having any return type.

( ) -> {System.out.println(“Hello”);}

(int a, int b) -> System.out.ptintln(“Hello”); -- for One line statement, no need of curly brace. Also, no need of input data type

(String str) -> {return str.length();}

**Functional Interface: -**

An Interface which have only one abstract method (SAM).

Example :- 1. Runnable --- > contains only run() method.

2. callable ---> vb only call() method.

3. ActionListener ---> contains only actionPerformed() method.

4. comparable --- > contains only compareTo method.

Anotation --- > @FunctionalInterface

**Anonymous inner classes vs Lambda Expressions**

Wherever we are using anonymous inner classes there may be a chance of using Lambda expression to reduce length of the code and to resolve complexity.

**What are the advantages of Lambda expression?**

* We can reduce length of the code so that readability of the code will be improved.
* We can resolve complexity of anonymous inner classes.
* We can provide Lambda expression in the place of object.
* We can pass lambda expression as argument to methods.

**Note:**

* Anonymous inner class can extend concrete class, can extend abstract class, can implement interface with any number of methods but
* Lambda expression can implement an interface with only single abstract method (Functional Interface).
* Hence if anonymous inner class implements Functional Interface in that particular case only, we can replace with lambda expressions. Hence wherever anonymous inner class concept is there, it may not possible to replace with Lambda expressions.
* Anonymous inner class! = Lambda Expression
* Inside anonymous inner class we can declare instance variables.
* Inside anonymous inner class “this” always refers current inner class object (anonymous inner class) but not related outer class object

**Ex:**

* Inside lambda expression we can’t declare instance variables.
* Whatever the variables declare inside lambda expression are simply acts as local variables
* Within lambda expression ‘this” keyword represents current outer class object reference (that is current enclosing class reference in which we declare lambda expression)
* From lambda expression we can access enclosing class variables and enclosing method variables directly.
* The local variables referenced from lambda expression are implicitly final and hence we can’t perform re-assignment for those local variables otherwise we get compile time error

**Differences between anonymous inner classes and Lambda expression: -**

|  |  |
| --- | --- |
| Anonymous Inner class | Lambda Expression |
| It’s a class without name | It’s a method without name (anonymous function) |
| Anonymous inner class can extend Abstract and concrete classes | lambda expression can’t extend Abstract and concrete classes |
| Anonymous inner class can implement an interface that contains any number of Abstract methods | lambda expression can implement an Interface which contains single abstract method (Functional Interface) |
| Inside anonymous inner class we can Declare instance variables. | Inside lambda expression we can’t Declare instance variables, whatever the variables declared are simply acts as local variables. |
| Anonymous inner classes can be Instantiated | lambda expressions can’t be instantiated |
| Inside anonymous inner class “this” Always refers current anonymous Inner class object but not outer class Object. | Inside lambda expression “this” Always refers current outer class object. That is enclosing class object. |
| Anonymous inner class is the best choice If we want to handle multiple methods. | Lambda expression is the best Choice if we want to handle interface with single abstract method (Functional Interface). |
| In the case of anonymous inner class at the time of compilation, a separate Dot class file will be generated (outerclass$1.class) | At the time of compilation, no dot Class file will be generated for Lambda expression. It simply converts in to private method outer class. |
| Memory allocated on demand Whenever we are creating an object | Reside in permanent memory of JVM (Method Area). |