**Lambda’s and functional Interfaces**

**What is Lambda?**

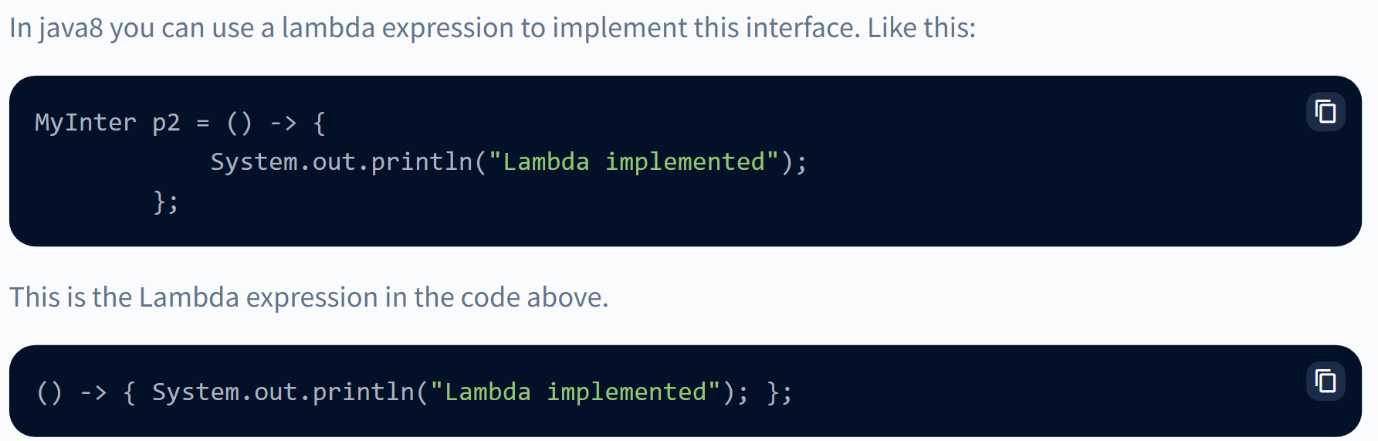
Lambda expression in java is an anonymous (no name) function that does not need to define the data type of input parameters and does not need to have a return type. Lambda expression in java implements the functional interface and it can be treated as any other java object. It can be used to create threads, comparators and can be used to add event Listeners.

The difficulty of utilising anonymous inner classes to implement one-function interfaces (functional interface in java8)?

Take a look at the one-function interface:



Before Java8, the best way to implement this example interface was to use anonymous inner classes, as shown in the code below. (similar concept we have practiced in threads)



Lambda expression in java is a feature introduced in Java 8 that adds functional programming techniques to Java, making it easier to write Java code in specific situations as with Java anonymous inner classes. Lambda expressions combine functional programming features with Java's object-oriented programming characteristics, resulting in more powerful concurrency features that are easier to use. A lambda expression is an anonymous function, which means it doesn't have a name and isn't bound by any return type constraints.

**A class that has no name is known as an anonymous inner class in Java. It should be used if you have to override a method of class or interface.**

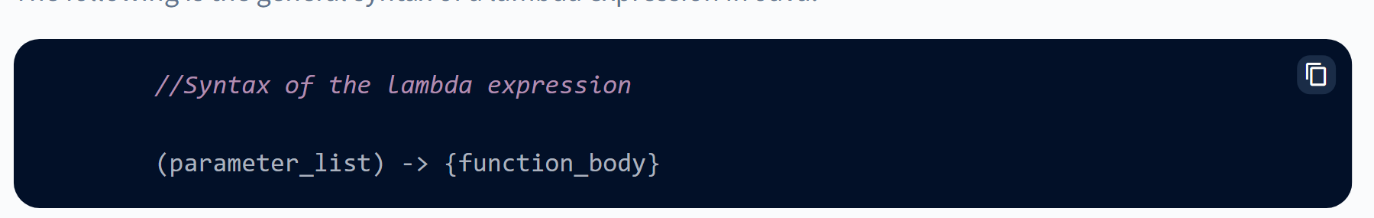
**You must either design your functional interface or use the default functional interface given by Java to use a lambda expression. Functional interfaces (or Single Abstract Method Interfaces) are interfaces that have only one abstract method, such as Runnable, Callable, and ActionListener.**

**Java Lambda Expression Syntax**

The Syntax of Lambda Expression in Java consists of three components.

1. **Arguments-list:** It is the first portion of the Lambda expression in Java. It can also be empty or non-empty. In the expression, it is enclosed by a round bracket.
2. **Lambda operator ->:** It's an arrow sign that appears after the list of arguments. It connects the arguments-list with the body of the expression.
3. **Body:** It contains the function body of lambda expression. It is surrounded by curly brackets.

The following is the general syntax of a lambda expression in Java



**No Parameter Syntax**

It is the case when the function does not require any input parameter, so don't provide anything in the parameters and the rest of the things are the same.



**One Parameter Syntax**

We only need to pass one input parameter in this situation; however, while using lambda, we don't need to specify the data type of input parameters because the compiler discovers it automatically.



**Two Parameter Syntax**

The case, as the name implies, refers to a circumstance in which two input parameters are necessary.

