Java Lambda Expression

1. **What is the lambda expression of Java 8?**

   Ans: As its name suggests it's an expression which allows you to write more succinct code in Java 8. For
   example (a, b) -> a + b is a lambda expression (look for that arrow ->).

   Which is equal to following code:

   public int value (int a, int b) {
   return a + b;
   }

   It's also called an anonymous function because you are essentially writing the code you write in function but
   without name.

2. **Can you pass lambda expressions to a method? When?**

   Ans: Yes, you can pass a lambda expression to a method provided it is expecting a functional interface. For
   example, if a method is accepting a Runnable, Comparable or Comparator then you can pass a lambda
   expression to it because all these are functional interfaces in Java 8.

```
package Interface_Ex;

interface TestInterface {
   boolean test(int a);
}
class Test {
   // lambda expression can be passed as first argument in the check() method
   static boolean check(TestInterface ti, int b) {
      return ti.test(b);
   }
}
public class LambdaExpressionPassMethod {
   public static void main(String arg[]) {
      // lambda expression
      boolean result = Test.check((x) -> (x%2) == 0, 10);
      System.out.println("The result is: "+ result);
   }
}
```

3. **What is the functional interface in java 8?**

   Ans: A functional interface in Java 8 is an interface with a single abstract method. For example, Comparator which has just one abstract method called compare() or Runnable which has just one abstract method called run(). There are many more general purpose functional interfaces introduced in JDK on java.util.function package. They are also annotated with @FunctionalInterface but that's optional.

4. **Why do we use lambda expressions in java?**

   Ans:

   1. To provide the implementation of Functional interface.
   2. Less coding.

A **lambda expression** can implement a **functional interface** by defining an **anonymous function** that can be passed as an argument to some method.

- **Enables functional programming**: All new JVM based languages take advantage of the functional paradigm in their applications, but programmers forced to work with **Object-Oriented Programming (OOPS)** till lambda expressions came. Hence lambda expressions enable us to write **functional code**.
- **Readable and concise code**: People have started using lambda expressions and reported that it can help to remove a huge number of lines from their code.
- **Easy-to-Use APIs and Libraries**: An API designed using lambda expressions can be easier to use and support other API.
- **Enables support for parallel processing**: A lambda expression can also enable us to write **parallel processing** because every processor is a multi-core processor nowadays.

5. **Is it mandatory for a lambda expression to have parameters?**

   Ans: No, it's not mandatory for a lambda expression to have parameters, you can define a lambda expression without parameters as shown below:
   () -> System.out.println("lambdas without parameter"); You can pass this code to any method which accepts a functional interface.