What is a lambda expression?

Java is an object-oriented language. By introducing lambdas in Java 8, the authors of Java tried to add elements of functional programming in Java. Now you might be wondering what the difference between object-oriented programming and functional programming is?

In **object-oriented programming**, objects and classes are the main entities. If we create a function then it should exist within a class. A function has no meaning outside the scope of the class object.

In **functional programming**, functions can exist outside the scope of an object. We can assign them to a reference variable and we can also pass them to other methods as a parameter.

A **lambda expression** is just an anonymous function, i.e., a function with no name and that is not bound to an identifier. We can pass it to other methods as parameters, therefore, using the power of functional programming in Java.

How to write a lambda expression

It might be difficult to understand what lambda is and how to write a lambda without looking at an example. So, let's look at an example first, and then we will revisit what we just read.

In the below example, we have a functional interface called `Greeting`. There are two classes `HindiGreeting` and `EnglishGreeting` that implement this interface.

```
@FunctionalInterface
public interface Greeting {
    void greet();
}
```

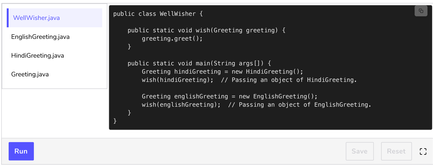
```
public class HindiGreeting implements Greeting {

    // Overriding the greet() method from Greeting interface.
    @Override
    public void greet() {
        System.out.println("Namaste");
    }
}
```

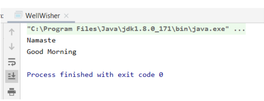
```
public class EnglishGreeting implements Greeting {

    // Overriding the greet() method from Greeting interface.
    @Override
    public void greet() {
        System.out.println("Good Morning");
    }
}
```

Here, we have another class called `WellWisher`. This class has a method called `wish(Greeting greeting)` which takes `Greeting` as a parameter and based on the type of object passed, prints the greeting.



When we run the `WellWisher` class, we get the below output.



This is a classic object-oriented programming example. Now, what if we want our `WellWisher` class to greet in all the languages available?

Do we need to create a class for each language, e.g., `SpanishGreeting`, `FrenchGreeting`, etc?

Isn't it possible that we don't create any class and just send a function to the `wish(Greeting greeting)` method?

Our `wish(Greeting greeting)` method will directly execute the function that is provided to it and print the greeting.

This is possible through anonymous classes. We will quickly see how this can be done through an anonymous class, and then jump straight back into lambdas.

In the below example, we will change the `WellWisher` class to use an anonymous class.

In the above example, we don't need to create a class for each language. We can use an anonymous class, and that does the trick for us in the example above. However, don't you think that this code is still cumbersome? Although the class is anonymous, we are still creating a class.

To make our code less cumbersome, let's remove all the unnecessary code step-by-step and create our first lambda expression.

Step 1 -> The compiler knows that the `wish(Greeting greeting)` method takes in a parameter of type `Greeting`. So, we don't need to specifically create an anonymous class of type greeting.

```
public class WellWisher {

    public static void wish(Greeting greeting) {
        greeting.greet();
    }

    public static void main(String args[]) {
        wish(
            public void greet() {
                System.out.println("Namaste");
            }
            );
    }
}
```

Step 2 -> We know that the Greeting interface has only one method. So, we don't need to provide the method name. We are only concerned with the method body.

```
public class WellWisher {

    public static void wish(Greeting greeting) {
        greeting.greet();
    }

    public static void main(String args[]) {
        wish(

            public void () {
                System.out.println("Namaste");
            }
            );
    }
}
```

Step 3 -> The compiler can understand that the body does not return anything. So, mentioning the return type is redundant. We can also remove the public declaration.

```
public class WellWisher {

    public static void wish(Greeting greeting) {
        greeting.greet();
    }

    public static void main(String args[]) {
        wish(
                () -> {
                    System.out.println("Namaste");
                }
        );
    }
}
```

> Please note that we add a `->` between the empty brackets and the method body. This is how a lambda expression is declared.

There still is one more improvement we can make. Since the method body contains only a single line, the curly braces are also unnecessary.

Congratulations!! You have written your first lambda. This is how simple it is to write a lambda expression.

To recap, when we write a lambda expression, we are basically sending a function as a method parameter, and it is directly getting executed.

In the next few lessons, you will see some more examples of writing lambdas. We will also discuss some of the inbuilt functional interfaces and how we can use them to make coding easier.