**Lambda Expressions**

Think of a Lambda expression as a shorthand for an anonymous class that implements a functional interface – an interface that contains only a single abstract method. This makes your code more concise and easier to understand.

Lambda expressions allow you to pass blocks of code as parameters, offering a powerful and flexible way to write cleaner and more functional code with minimal effort. These compact and expressive constructs can greatly simplify your code.

Records are static like enums and interfaces.

Lambdas can be assigned to a variable. But assigning it isn’t going to print it. We need to call the method that prints the lambda and assign it there for example:

BiConsumer<Double,Double> p1 =

(lat, lng) -> System.***out***.printf(**"[lat:%.3f lon:%.3f]%n"**, lat,lng);  
**var** firstPoint = coords.get(0);  
*processPoint*(firstPoint[0], firstPoint[1], p1);//print 1st coordinate

System.***out***.println(**"--------"**);  
  
coords.forEach(s -> *processPoint*(s[0],s[1], p1));//prints all the coordinates

*//creating a generic method without the return type with BiConsumer***public static** <T> **void** processPoint (T t1, T t2, BiConsumer<T,T> consumer) {  
 consumer.accept(t1,t2);  
}

**The Lambda Expression**

(o1, o2) -> o1.lastName().compareTo(o2.lastName()));

Syntax: (paramter1, paramter2,…) -> expression;

For a lambda expression, **the method is inferred by Java**. Java takes its clue from the reference type, in the context of the Lambda expression usage.

**Functional Interface:**

A functional Interface is an interface that has one, and only one, abstract method. This is how java can infer the method to derive the parameters and return type, for the Lambda expression.

We can use @FuntionalInterface for functional methods. Java provides a library of functional interfaces in the java.util.function package. One of them is Consumer interface. The other one is the Binary Operator.

**The Consumer Interface:**

The Consumer interface is in the Jave.util.function Package.

It has one abstract method that takes a single argument and doesn’t return anything.

**Void accept (T t)**

The two most common Cosnumer interfaces, and the functional method are the following

The Consumer interface takes one argument of any type.

The BiConsumer interface takes two arguments of two different types

|  |  |
| --- | --- |
| Interface Name | Method Signature |
| Consumer | Void accept (T t) |
| BiConsumer | Void accept (T t, U u) |

**Example Lambda Expression for Consumer and Consumer Method:**

S -> System.out.println(S); void accept (T t);

**Lambda use in foreach loop:**

List.foreach (element -> System.out.println(element)) //valid

List.foreach ((element) -> System.out.println(element)) //valid

List.foreach ((String element) -> System.out.println(element)) //valid

List.foreach ((var element) -> System.out.println(element)) //valid

**int** result = *calculator*((**var** a, **var** b) -> {**return** a + b;}, 5, 2);// valid

**int** result = *calculator*((**var** a, **var** b) -> {**var** c = a + b; **return** c;}, 5, 2);

**Use of Lmabda expression variations, the Lambda body:**

list.forEach((**var** myString) -> System.***out***.println(myString));//valid

list.forEach((**var** myString) -> {  
 **char** first = myString.charAt(0);  
 System.***out***.println(myString + **" means "** + first);  
}); //valid

**The Four categories of Functional Interfaces:**

It’s a good idea to know the four basic types of functional interfaces in the java.util.function package. There are over forty interfaces in this package. The following table shows the four categories, with the simplest method shown.

|  |  |  |
| --- | --- | --- |
| **Interface Category** | **Basic Method Signature** | **Purpose** |
| Consumer | Void accept (T t) | Execute code without returning data |
| Function | R apply (T t) | Return a list of an operation or function |
| Predicate | Boolean test (T t) | Test if a condition is true or false |
| Supplier | T get () | Return an instance of something |