**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.

**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.

**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)**

**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

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