Lambda Expressions

A lambda expression is, essentially, an anonymous (that is, unnamed) method. However, this method is not executed on its own. Instead, it is used to implement a method defined by a functional interface. Thus, a lambda expression results in a form of anonymous class. Lambda expressions are also commonly referred to as closures.

A functional interface is an interface that contains one and only one abstract method. Normally, this method specifies the intended purpose of the interface. Thus, a functional interface typically represents a single action. For example, the standard interface Runnable is a functional interface because it defines only one method: run( ). Therefore, run( ) defines the action of Runnable.

Furthermore, a functional interface defines the target type of a lambda expression. Here is a key point: a lambda expression can be used only in a context in which its target type is specified. One other thing: a functional interface is sometimes referred to as a SAM type, where SAM stands for Single Abstract Method.

The lambda expression introduces a new syntax element and operator into the Java language. The new operator, sometimes referred to as the lambda operator or the arrow operator, is −>. It divides a lambda expression into two parts. The left side specifies any parameters required by the lambda expression. (If no parameters are needed, an empty parameter list is used.) On the right side is the lambda body, which specifies the actions of the lambda expression. The −> can be verbalized as “becomes” or “goes to.”

Java defines two types of lambda bodies. One consists of a single expression, and the other type consists of a block of code. We will begin with lambdas that define a single expression.

Example:

() -> 123.45

This lambda expression takes no parameters, thus the parameter list is empty. It returns the constant value 123.45. Therefore, it is similar to the following method:

double myMeth() { return 123.45; }

() -> Math.random() \* 100

This lambda expression obtains a pseudo-random value from Math.random( ), multiplies it by 100, and returns the result. It, too, does not require a parameter.

Parameterized Lambda Expression:

(n) -> (n % 2)==0 This lambda expression returns true if the value of parameter n is even. Although it is possible to explicitly specify the type of a parameter, such as n in this case, often you won’t need to do so because in many cases its type can be inferred.