The anonymous function, or lambda

The anonymous function was introduced into Java in version 8. It is also called *lambda* because the idea and a notation for it was first used in Alonzo Church's *lambda calculus*, or *λ-calculus*, in the 1930s as formal system for expressing computations. Here is a discussion of the lambda calculus:

https://en.wikipedia.org/wiki/Lambda_calculus

Introduction to anonymous functions

Consider these two declarations:

int sum(**int** c, **int** d) {**return** c + d;}

(c, d) -> c + d

The first is a conventional function declaration. The second is an equivalent *anonymous* function —anonymous because the function is not given a name. The list of parameters is there (though their types need not be given), the function body is there (though in this case braces are not needed), but the function name is not there. Also note that there is no need for the keyword **return**. The body is an expression, and its value is returned.

In certain circumstances, you could assign the anonymous function to a variable:

add= (c, d) -> {c + d};

and then call it, using

add.apply(3, 4);

Use of anonymous functions in Java

In functional programming languages, anonymous functions are easy to write and understand. Because of the way anonymous functions are added to Java in version 8, their introduction is more complex. For example, variable add in the above paragraphs has type:

BiFunction<Integer, Integer, Integer>

which means that it is a function that has two Integer parameters and returns an Integer. It takes a while to learn how to write and use anonymous functions.

However, their use in some places makes programming easier and simpler. For example, In the tutorial on Eclipse, we showed you how to write a statement to test whether the new-expression

new P(null)

threw an an AssertionError. We said to use the statement

assertThrows(AssertionError.class, () -> {new P(null);});

We will write more explanation of the anonymous function later.