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 lines of Java code:

```
int sum(int c, int d) {return c + d;}
(c, d) -> c + d
```

The first is a conventional function declaration. The second is an expression whose evaluation yields 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, keyword **return** is not needed. 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`.

Before we go into full detail on anonymous functions, we show how to use them in a few simple but powerful ways. The first is in a JUnit testing class to test whether a statement throws an exception, shown below. For other ways, look at JavaHyperText entry "anonymous function".

Using an anonymous function to test whether an exception is thrown

Suppose you want to test in a JUnit testing class whether this new-expression throws an `AssertionError`:

```
new P(null)
```

To make that test, use the following call on procedure `assertThrows`:

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

Here, the second argument to `assertThrows` is an anonymous function with no parameters, and its body is the expression to be evaluated followed by a semicolon.

Another example: The following call on `assertThrows` tests whether dividing by 0 throws an `ArithmeticException`. Again, we write the anonymous function in red.

```
assertThrows(ArithmeticException.class, () -> {int b= 5 / 0;});
```

You can generalize this yourself to testing whether any expression or statement throws any exception.