Suppose this call on method m is supposed to throw an ArithmeticException:

```
P.m(0);
```

You want to check in a JUnit 5 testing procedure that the call throws the exception. Do it like this:

```
@Test
public void test() {
   assertThrows(ArithmeticException.class, () -> {P.m(0);});
}
```

The first argument to method assertThrows is the name of any Throwable class followed by ".class". The second argument is an *anonymous function* with no parameters (look up *anonymous function* in the JavaHyperText). The body of the function is {P.m(0);}. You can see that it contains the call P.m(0);

You can use any statement as the body of the anonymous function.

Note: When you first write a call on assertThrows, you may get a message saying that it is not available. In that case, insert this import statement:

```
import static org.junit.jupiter.api.Assertions.*;
```

Testing using a JUnit 4 testing procedure

JUnit 4 does not include method assertThrows, so you have to fall back on a more cumbersome way of testing whether an exception was thrown.

To check that the call P.m(0) throws the exception, use this procedure:

```
@Test
public void test() {
   try {
      P.m(0);
      fail();
   }
   catch (ArithmeticException e) {}
}
```

In analyzing execution, we consider three cases:

1. The call throws an ArithmeticException. The exception is caught by the catch-block and catch-block and the try-statement terminate normally.
2. The call throws some other exception. The exception is *not* caught and is thrown out further, so the procedure fails. That is what it is supposed to do.
3. The call does not throw an exception. The statement fail() is executed, which throws an AssertionError, which is *not* caught by the catch-block and is thrown out further, so the procedure fails. That is what it is supposed to do.

We make two points about this code.

1. You may need to use this pattern many times, and since it is standard, a paradigm, it's OK to scrunch it up to take two lines:

   ```
   try {P.m(0);  fail();}
   catch (ArithmeticException e) {}}
   ```

   That makes it easy to place it among all the other tests that are being performed on method A.m.

2. You can use this pattern to test that a method throws *any* exception (just replace ArithmeticException by another exception). But *don't* use it for the exception AssertionError. Here's the reason. The call fail() always throws an AssertionError, which will then be caught by the catch-block, and it doesn't work right. See the JavaHyperText entry for "Junit testing" to see how to check whether an assert statement does it job.