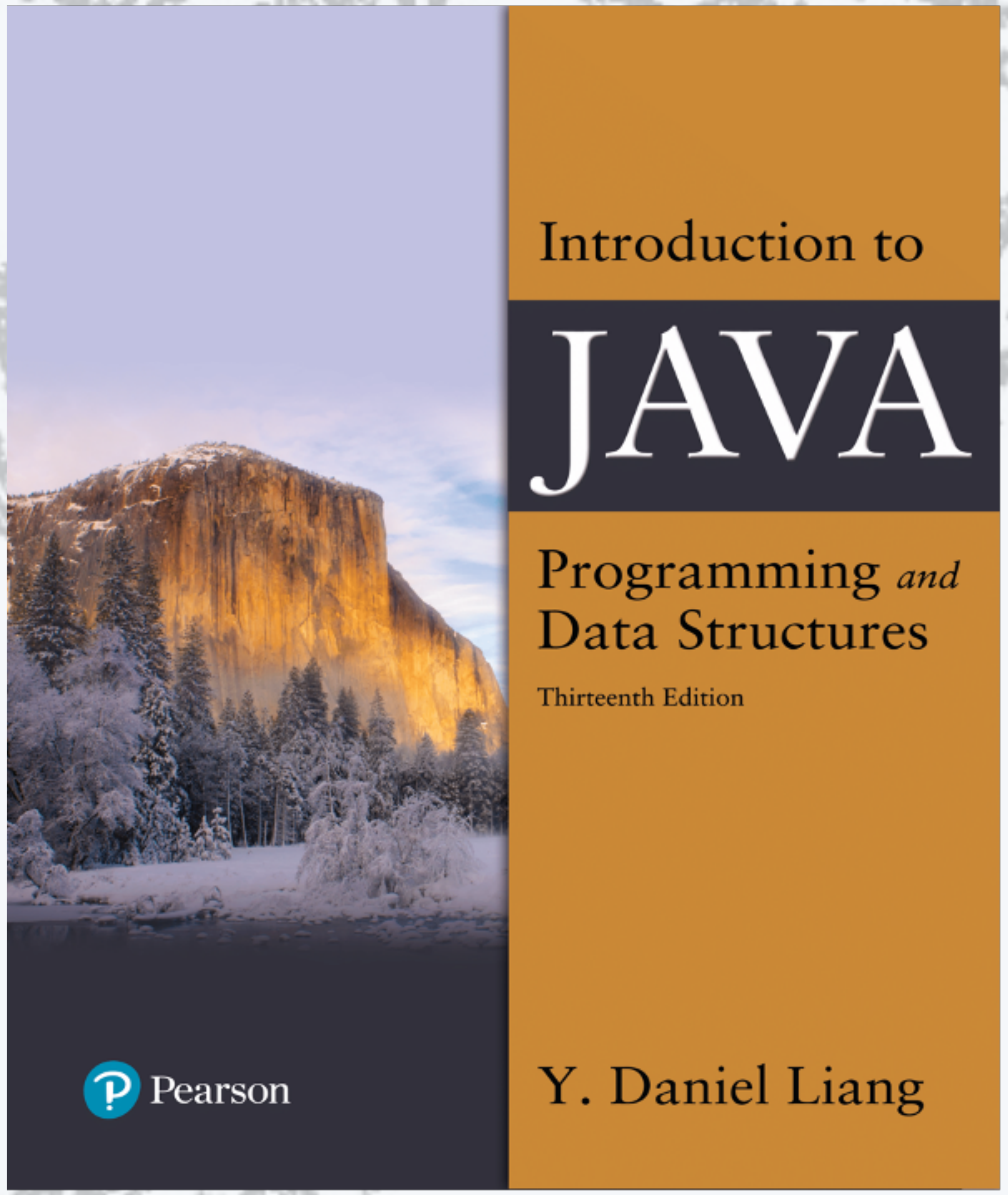


Exception Handling

Exception Handling

- Exception Handling
-

Motivations

When a program runs into a ***runtime error***, the program terminates abnormally. How can you handle the runtime error so that the program can continue to run or terminate gracefully? This is the subject we will introduce in this chapter.

Exception-Handling Overview

Example of Runtime Error Quotient

Fix it using an if statement QuotientWithIf

With a Method QuotientWithMethod

QuotientWithException

Advantages of using exception handling

It enables a method to throw an exception to its caller. Without this capability, a method must handle the exception or terminate the program.

Handling InputMismatchException

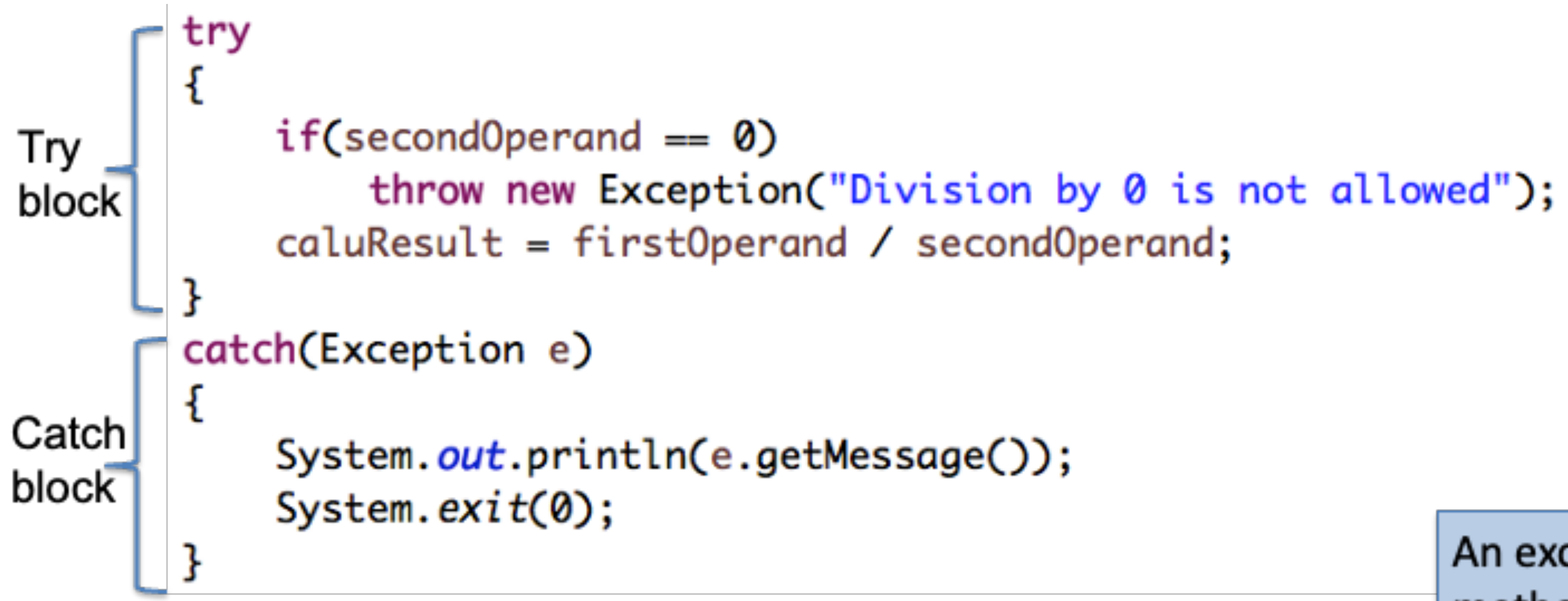
InputMismatchExceptionDemo

By handling InputMismatchException, our program will continuously read an input until it is correct.

Try-Throw-Catch

- In Java, the mechanism is called “**Exception Handling**”
  - Try to execute some actions
  - **Throw an exception**: report a problem and asks for some code to handle it properly
  - **Catch an exception**: a piece of code dedicated to handle one or more specific types of problem

Try-Throw-Catch Example



```
try
{
    if(secondOperand == 0)
        throw new Exception("Division by 0 is not allowed");
    caluResult = firstOperand / secondOperand;
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    System.exit(0);
}
```

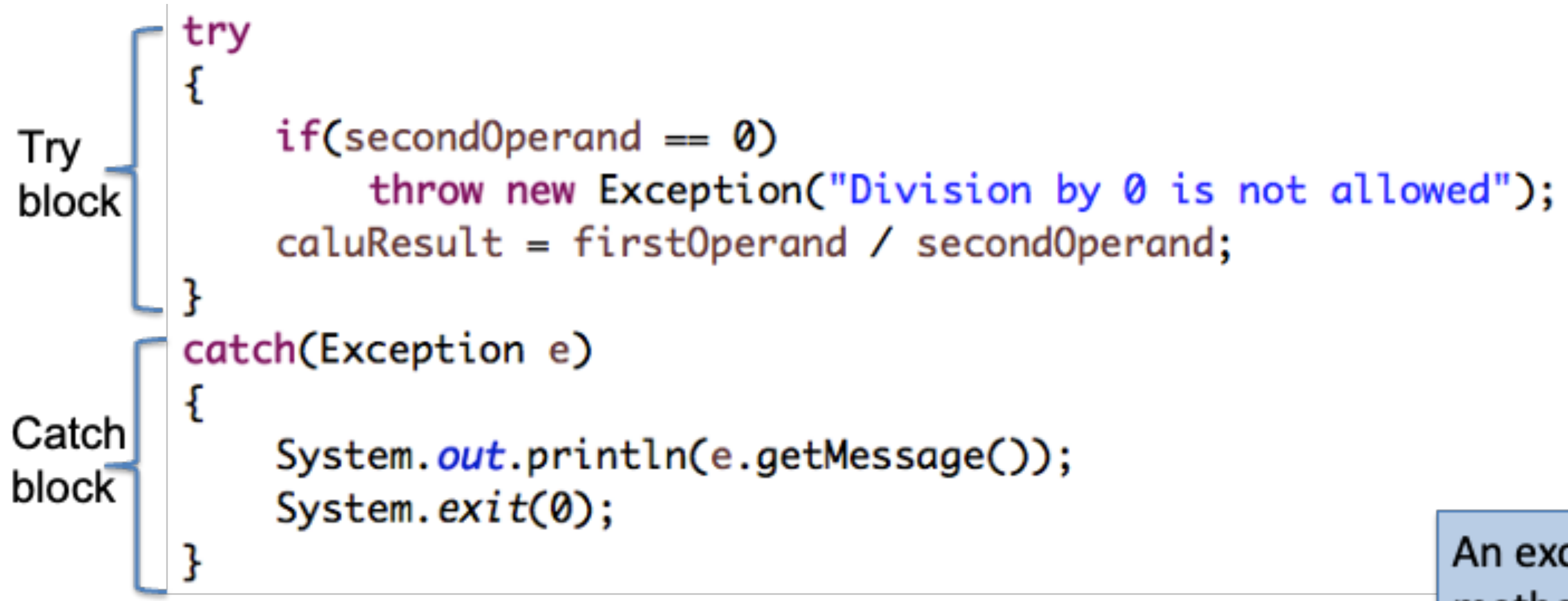
An exception's getMessage method returns a description of the exception

A try bock detects an exception

A throw statement throws an exception

A catch block deals with a particular exception

Try-Throw-Catch Example



```
try
{
    if(secondOperand == 0)
        throw new Exception("Division by 0 is not allowed");
    caluResult = firstOperand / secondOperand;
}
catch(Exception e)
{
    System.out.println(e.getMessage());
    System.exit(0);
}
```

An exception's getMessage method returns a description of the exception

If an exception **occurs** within a try block, the rest of the block is ignored

If **no** exception occurs within a try block, the catch blocks are ignored

An exception is an object of the class Exception

Syntax for Handling Exceptions

Syntax for the try and catch statements

```
try
{
    Code_To_Try
    Possibly_Throw_An_Exception
    More_Code
}
catch (Exception_Class_NameCatch_Block_Parameter)
{
    Process_Exception_Of_Type_Exception_Class_Name
}
Possibly_Other_Catch_Blocks
```

Syntax for the throw statement

```
throw new Exception_Class_Name(Possibly_Some_Arguments);
```

Predefined Exception Classes

- Java provides several exception classes
  - The names are designed to be self-explanatory
      - *BadStringOperationException,*
      - *ClassNotFoundException,*
      - *IOException,*
      - *NoSuchMethodException,*
      - *InputMismatchException*
  - Use the try and catch statements

An Example

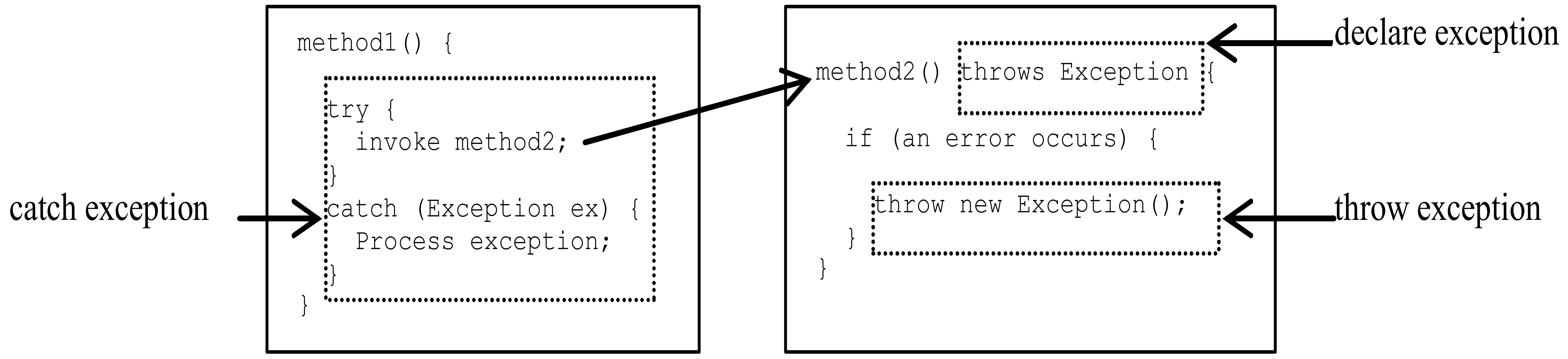
```
SampleClass object = new SampleClass();
try
{
    <Possibly some code>
    object.doStuff(); //may throw IOException
    <Possibly some more code>
}
catch(IOException e)
{
    <Code to deal with the exception, probably including the
     following:>
    System.out.println(e.getMessage());
}
```

If you think that continuing with program execution is infeasible after the exception occurs, use `System.exit(0)` to end the program in the catch block

Declaring Exceptions

When we want to delay handling of an exception

A method might not catch an exception that its code throws



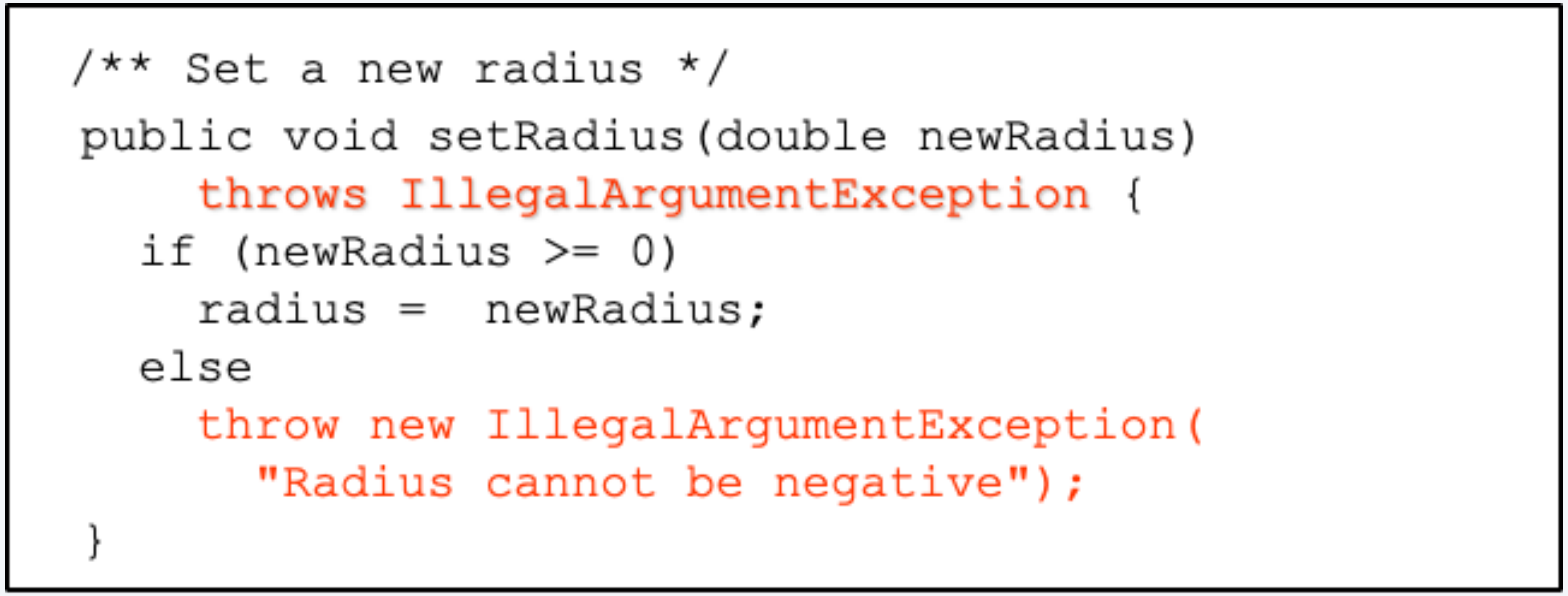
```
method1() {

  try {
    invoke method2;
  }
  catch (Exception ex) {
    Process exception;
  }
}
```

```
method2() throws Exception {

  if (an error occurs) {

    throw new Exception();
  }
}
```

Throwing Exception Example

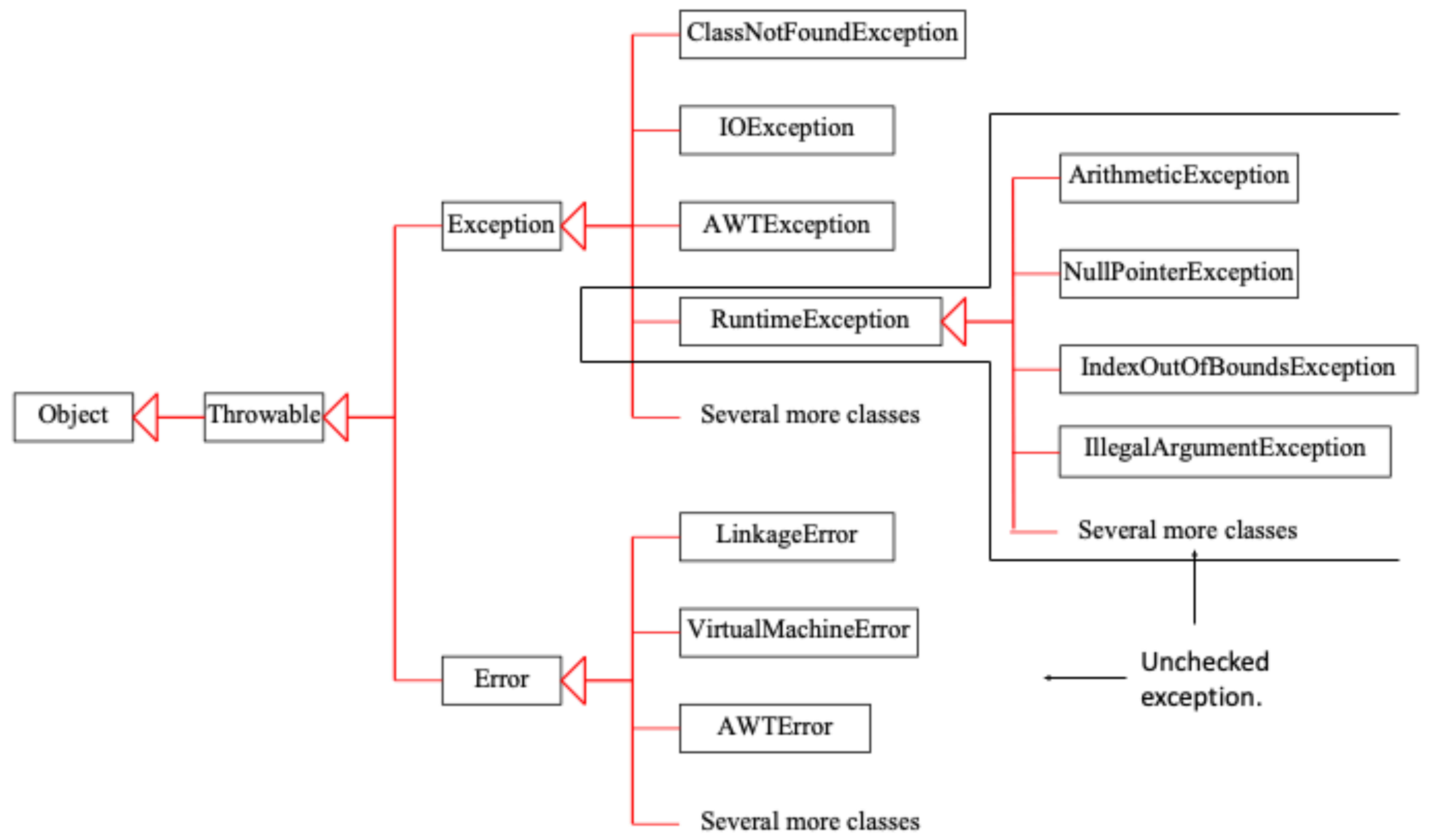
Step 1: add throws clause, "`throws ExceptionType`", in the method's heading

Step 2: when problem occurs, use a throw statement throws an exception, "`throw new ExceptionType( …. );` "



```
/** Set a new radius */
public void setRadius(double newRadius)
    throws IllegalArgumentException {
  if (newRadius >= 0)
    radius =  newRadius;
  else
    throw new IllegalArgumentException(
      "Radius cannot be negative");
}
```

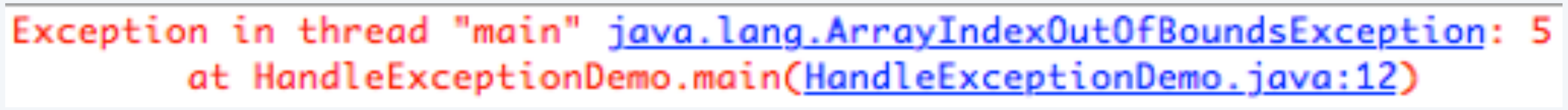
Java Exception Hierarchy



Checked Exceptions vs. Unchecked Exceptions

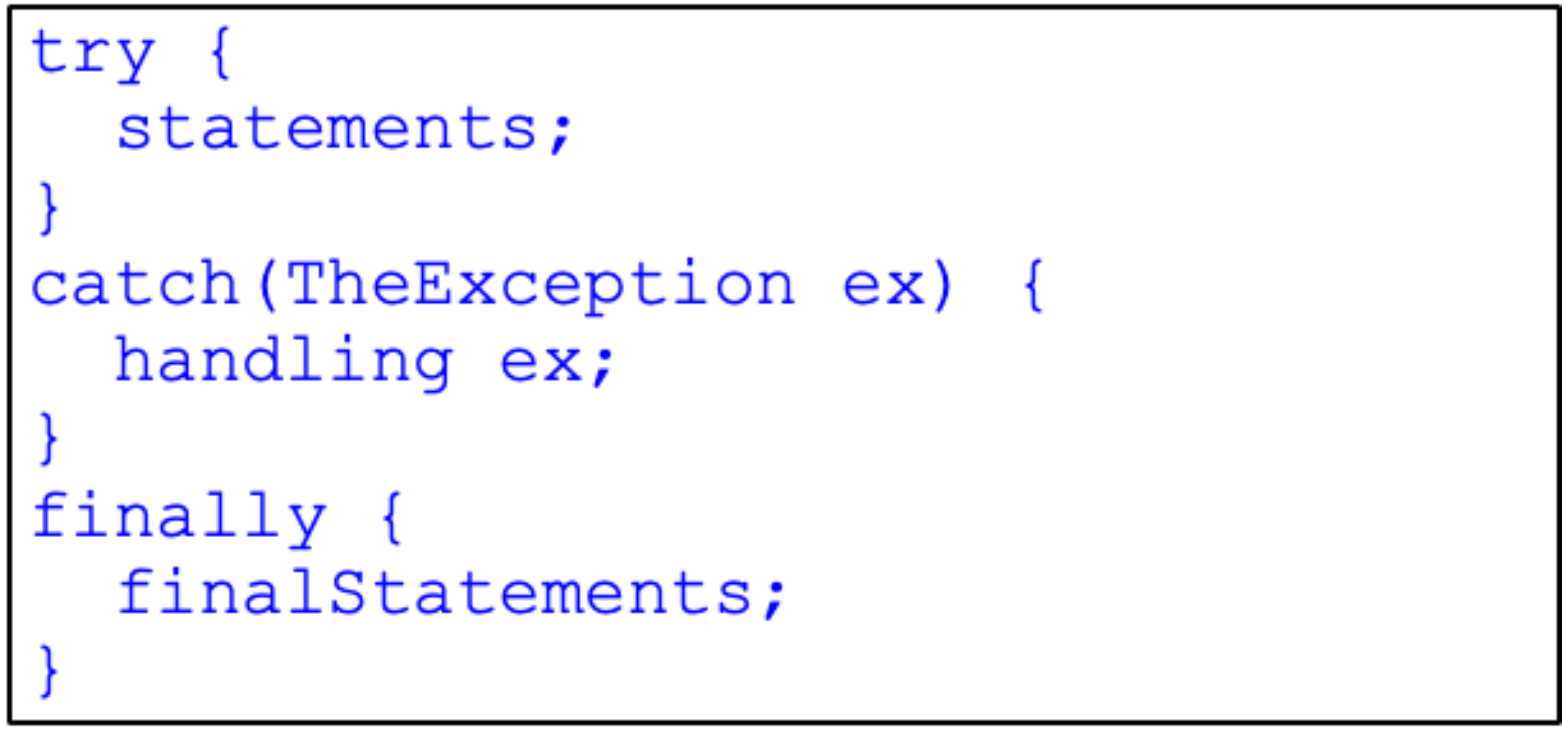
- **RuntimeException, Error** and **their subclasses** are known as *unchecked exceptions*
  - no need to be caught or declared in a throws clause of a method's heading
- All other exceptions are known as *checked exceptions*
  - must be either caught or declared in a throws clause

- In most cases, unchecked exceptions reflect programming logic errors that are not recoverable
  - a *NullPointerException* is thrown if you access an object through a reference variable before an object is assigned to it
  - an *ArrayIndexOutOfBoundsException* is thrown if you access an element outside the bounds of the array
- Logic errors that should be corrected in the program, Java does not mandate you to write code to catch unchecked exception



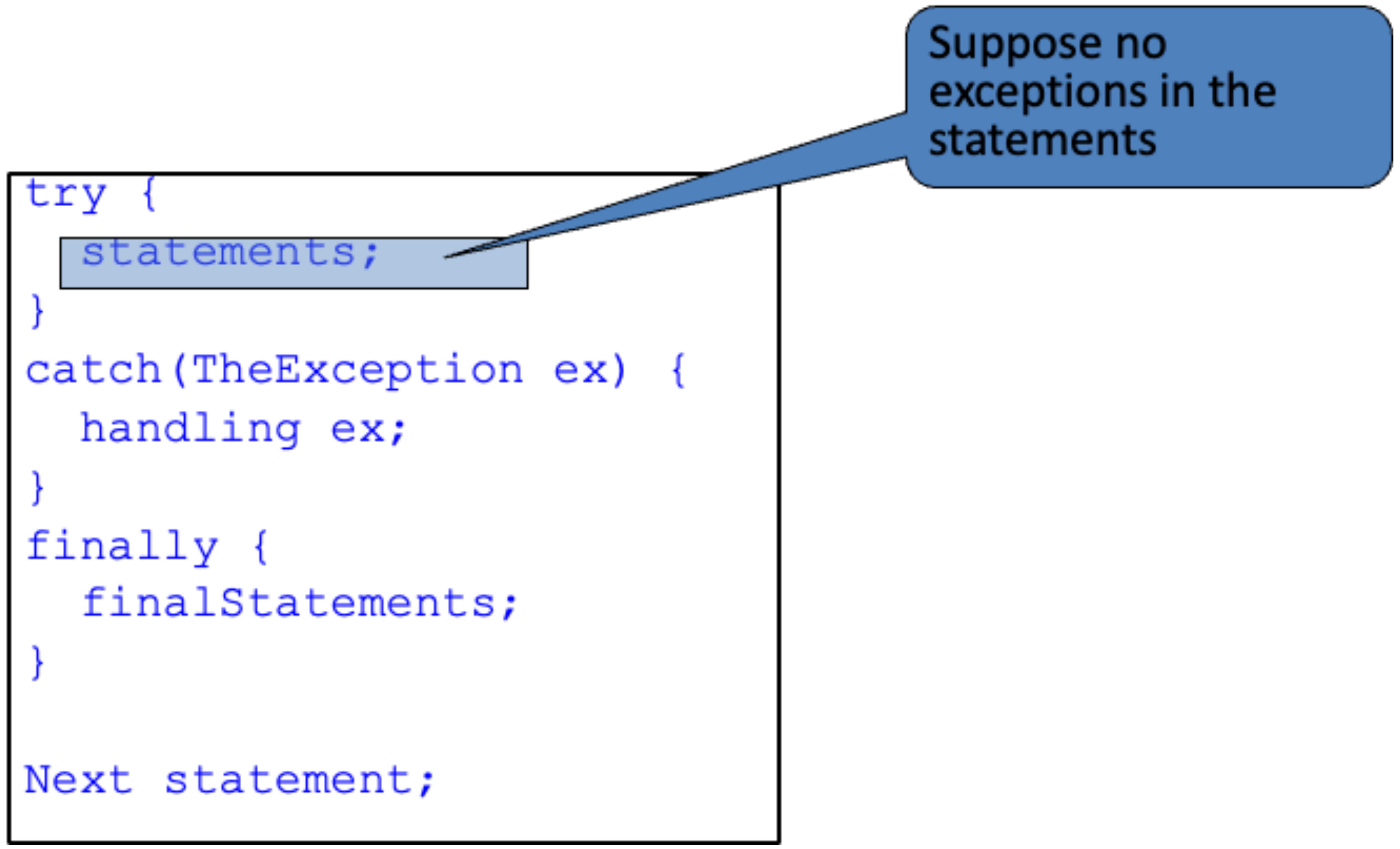
```
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
        at HandleExceptionDemo.main(HandleExceptionDemo.java:12)
```

The `finally` block

- A finally block always executes
- Put cleanup code in a finally block, e.g., closing a file
- 

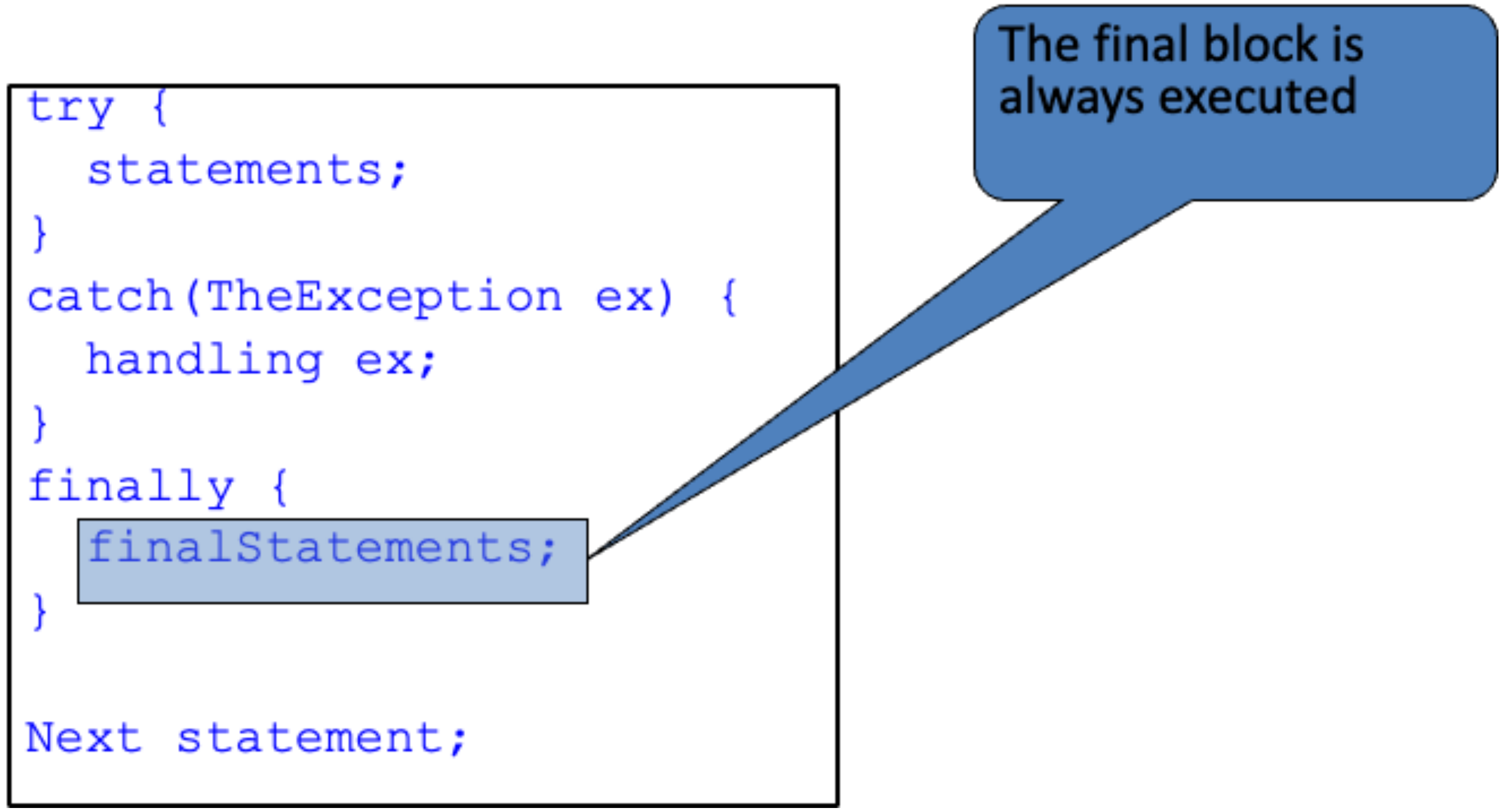
```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

Trace a Program Execution



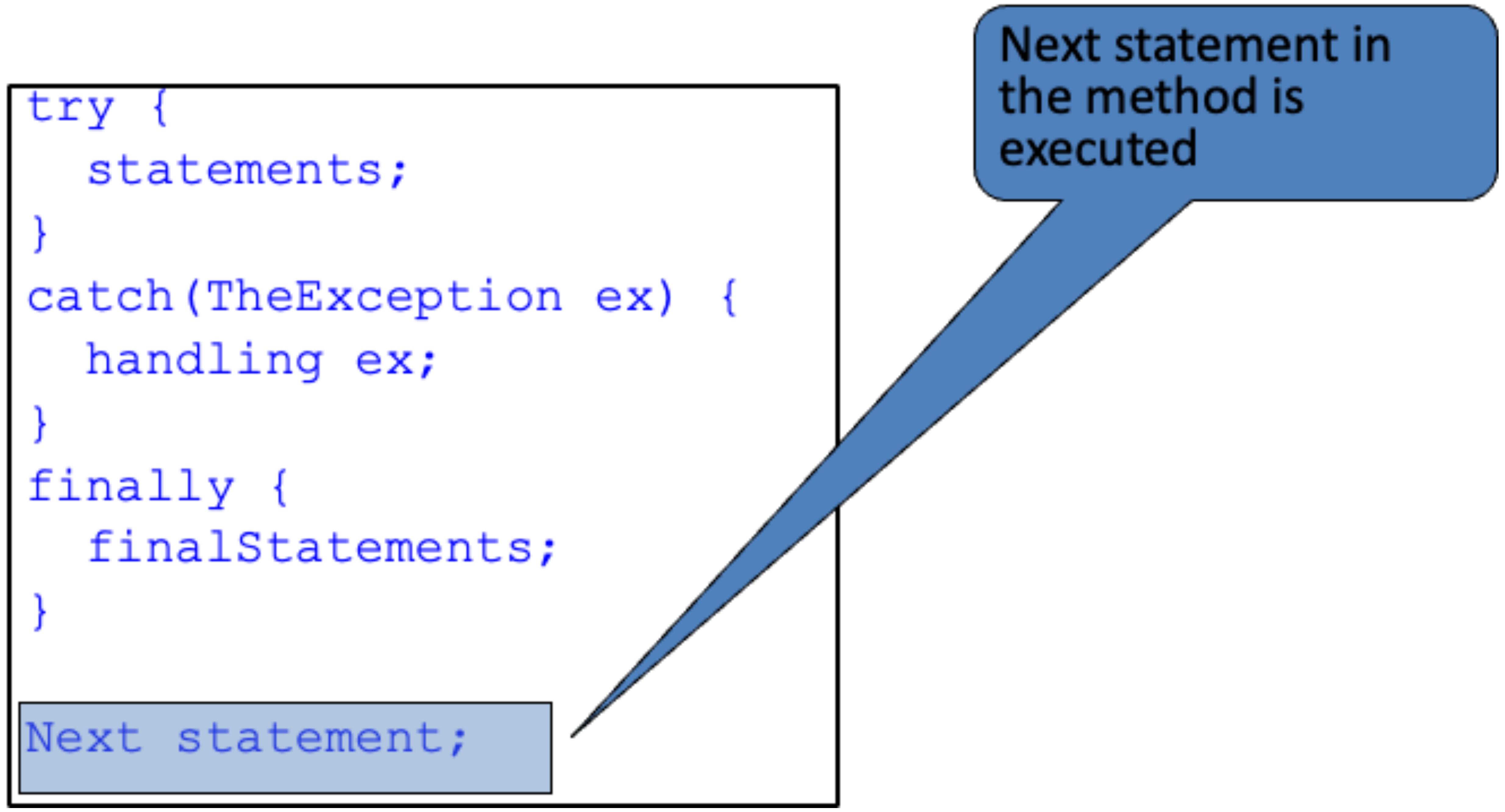
```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution



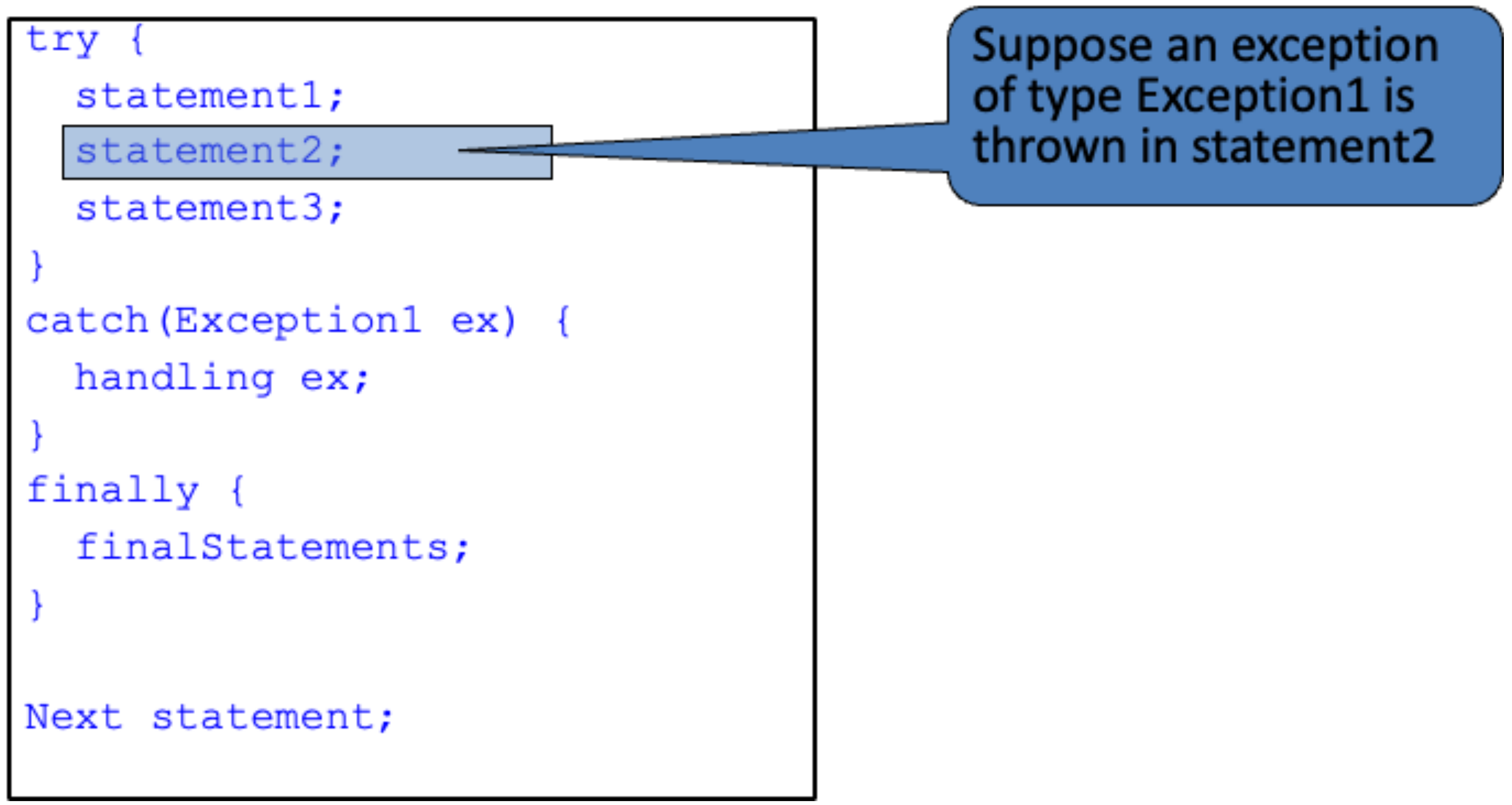
```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution



```
try {
  statements;
}
catch(TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

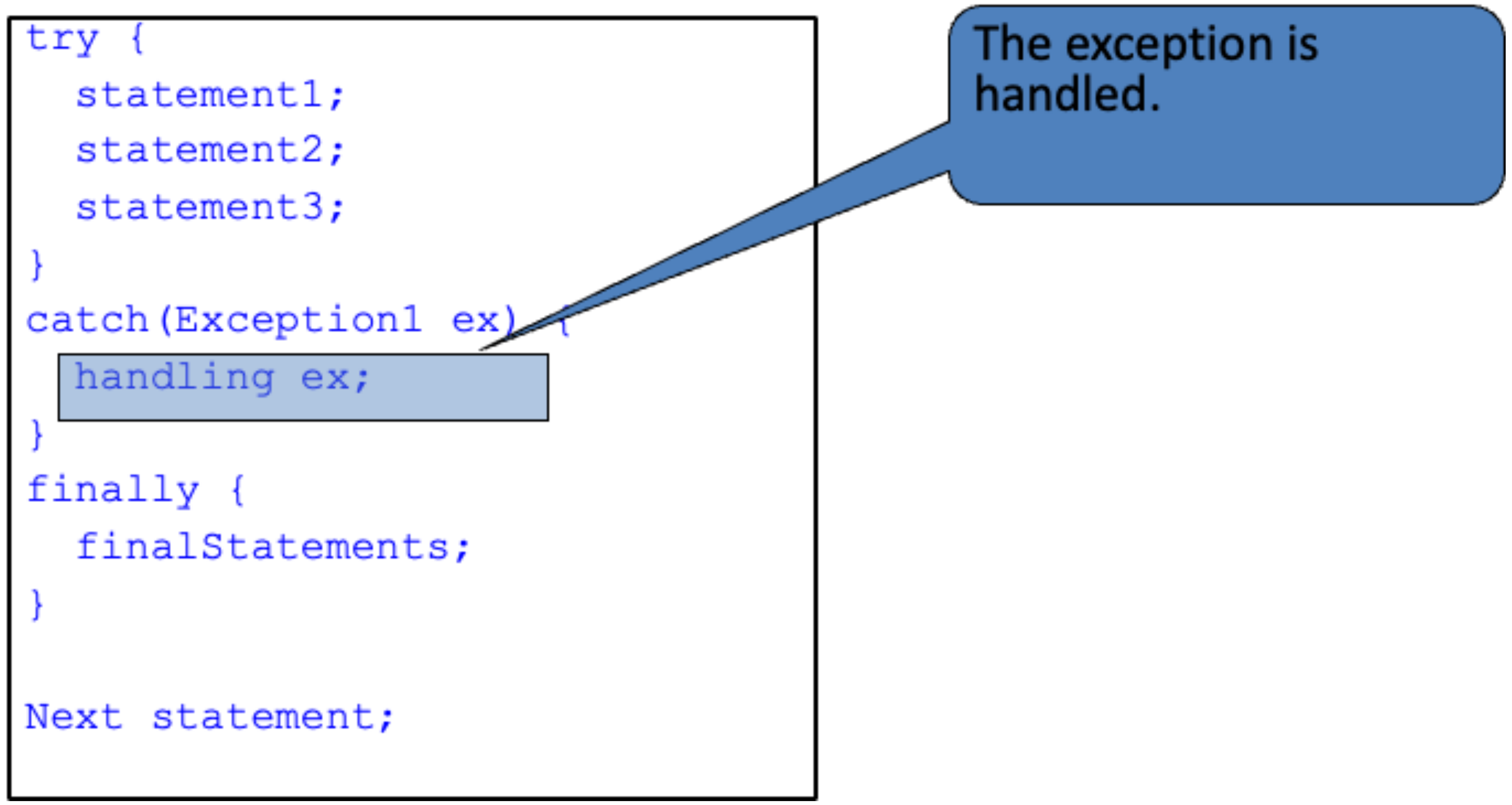
Trace a Program Execution



```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

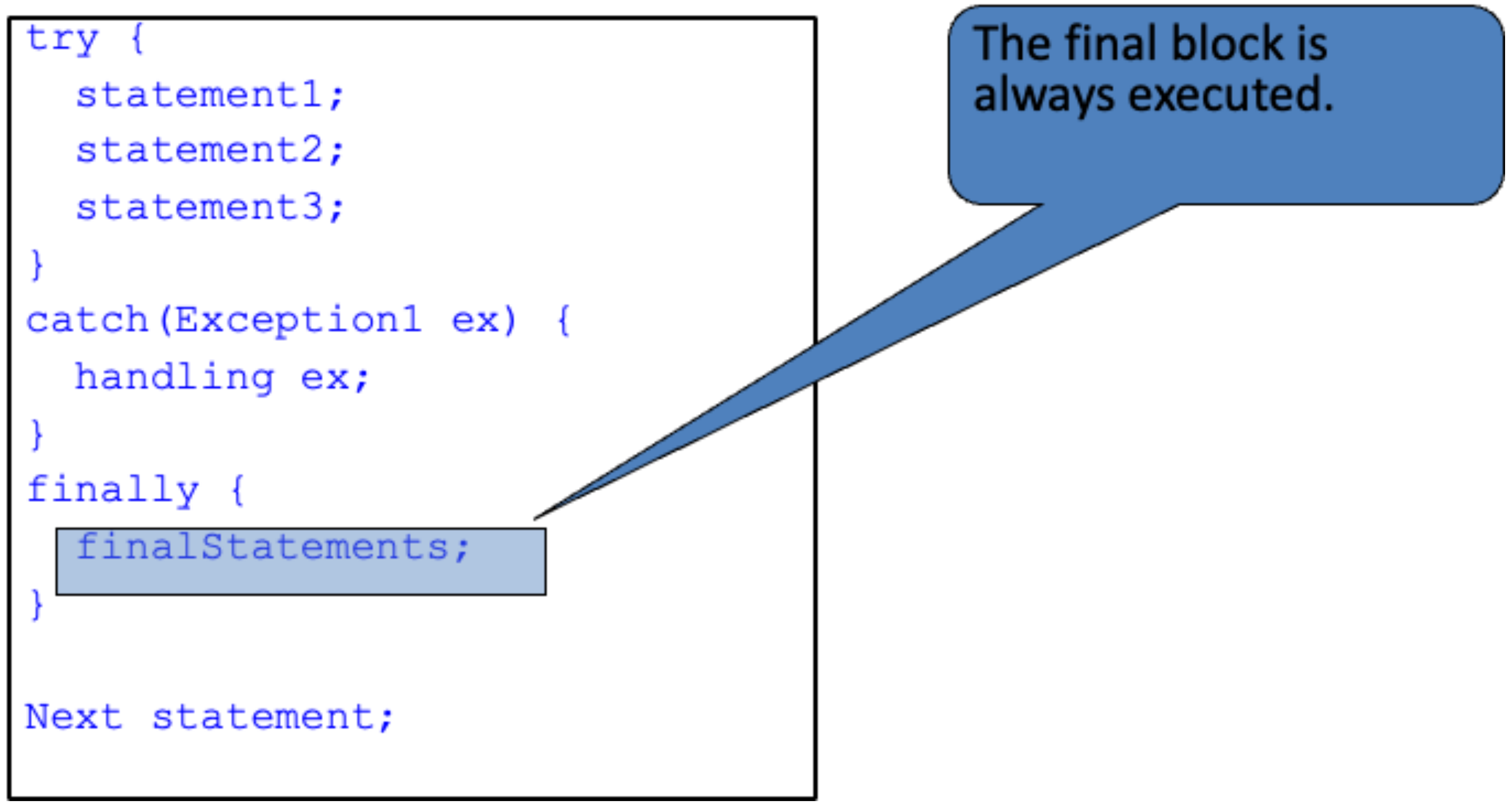
Suppose an exception of type Exception1 is thrown in statement2

Trace a Program Execution



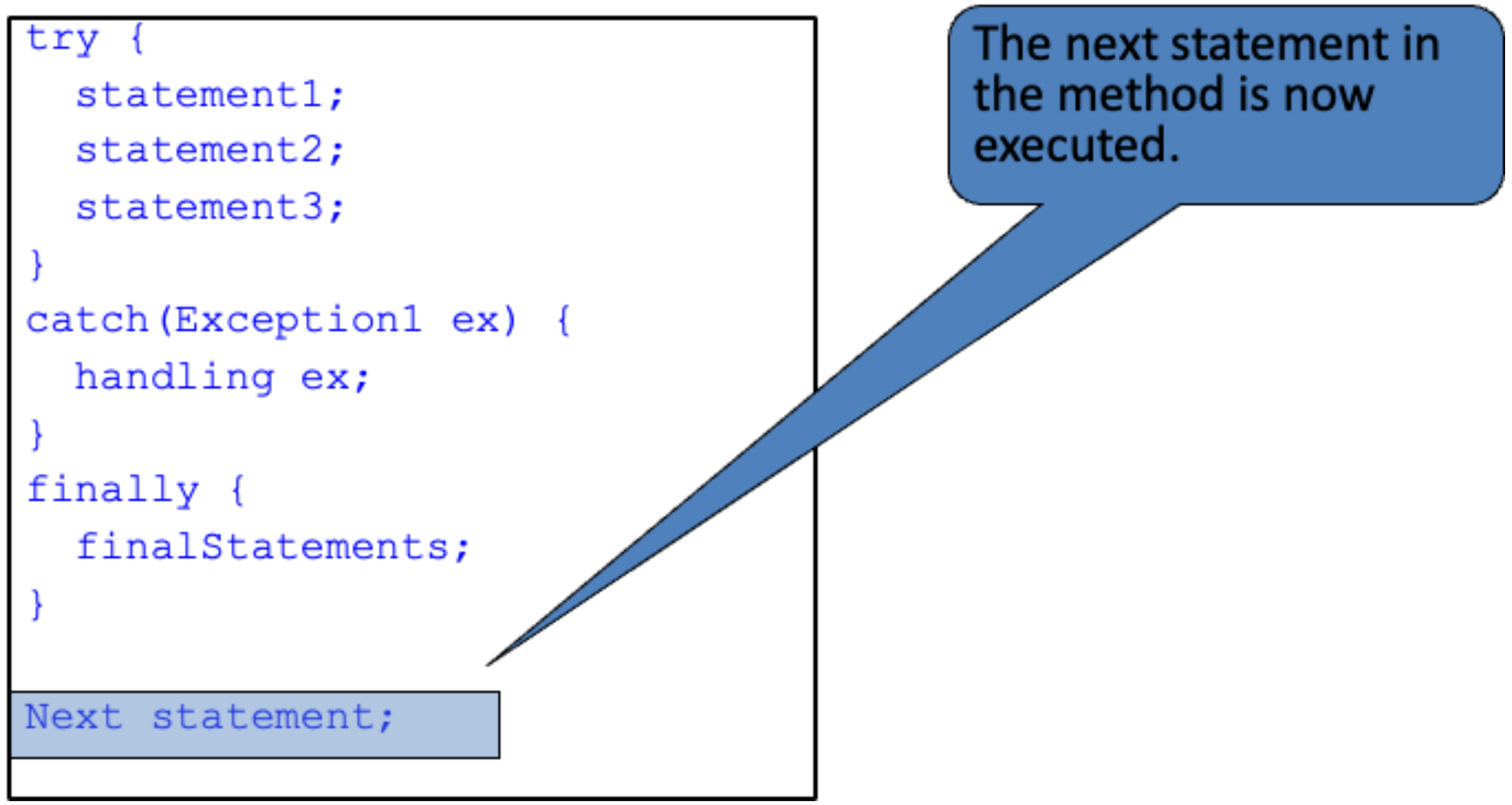
```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution



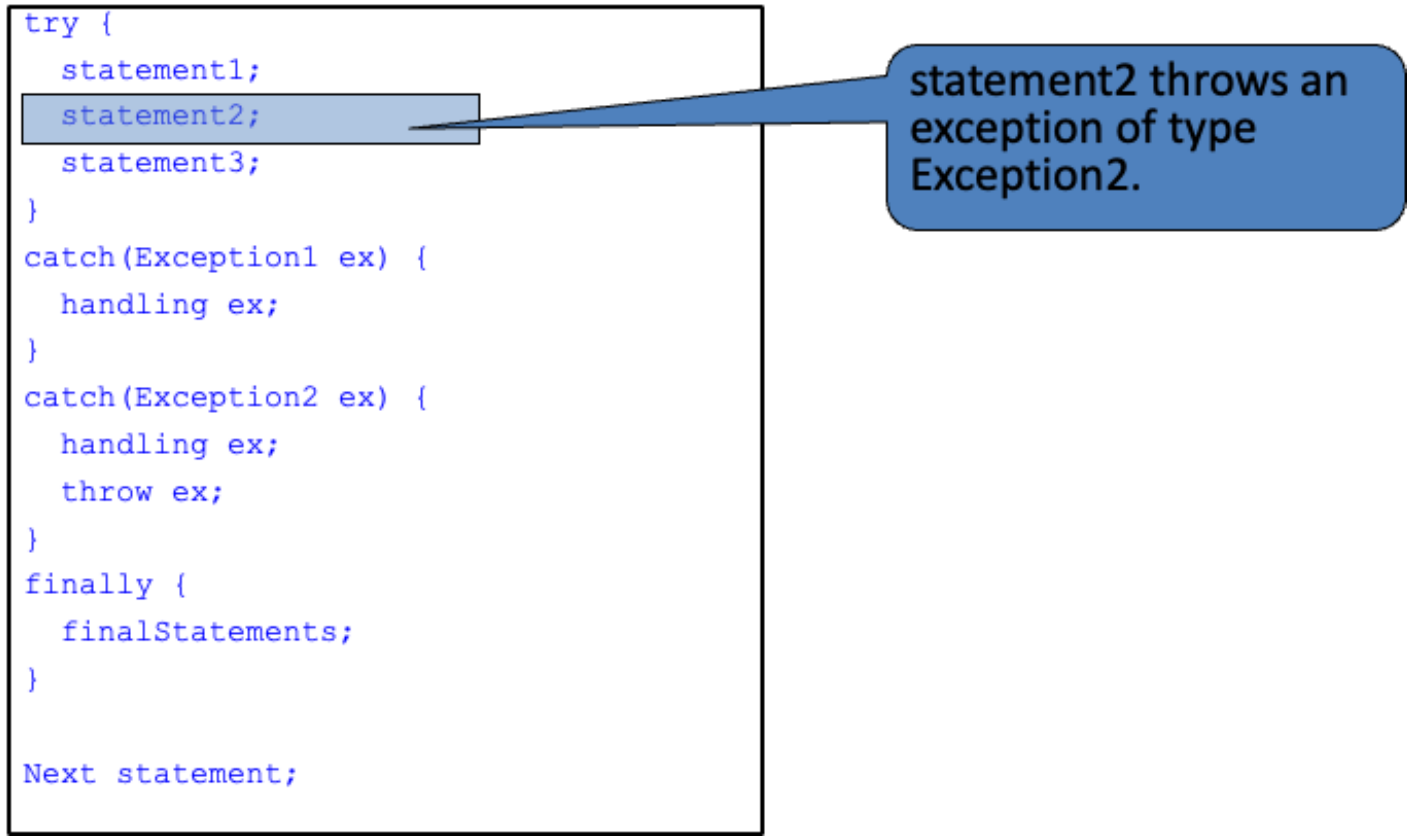
```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution



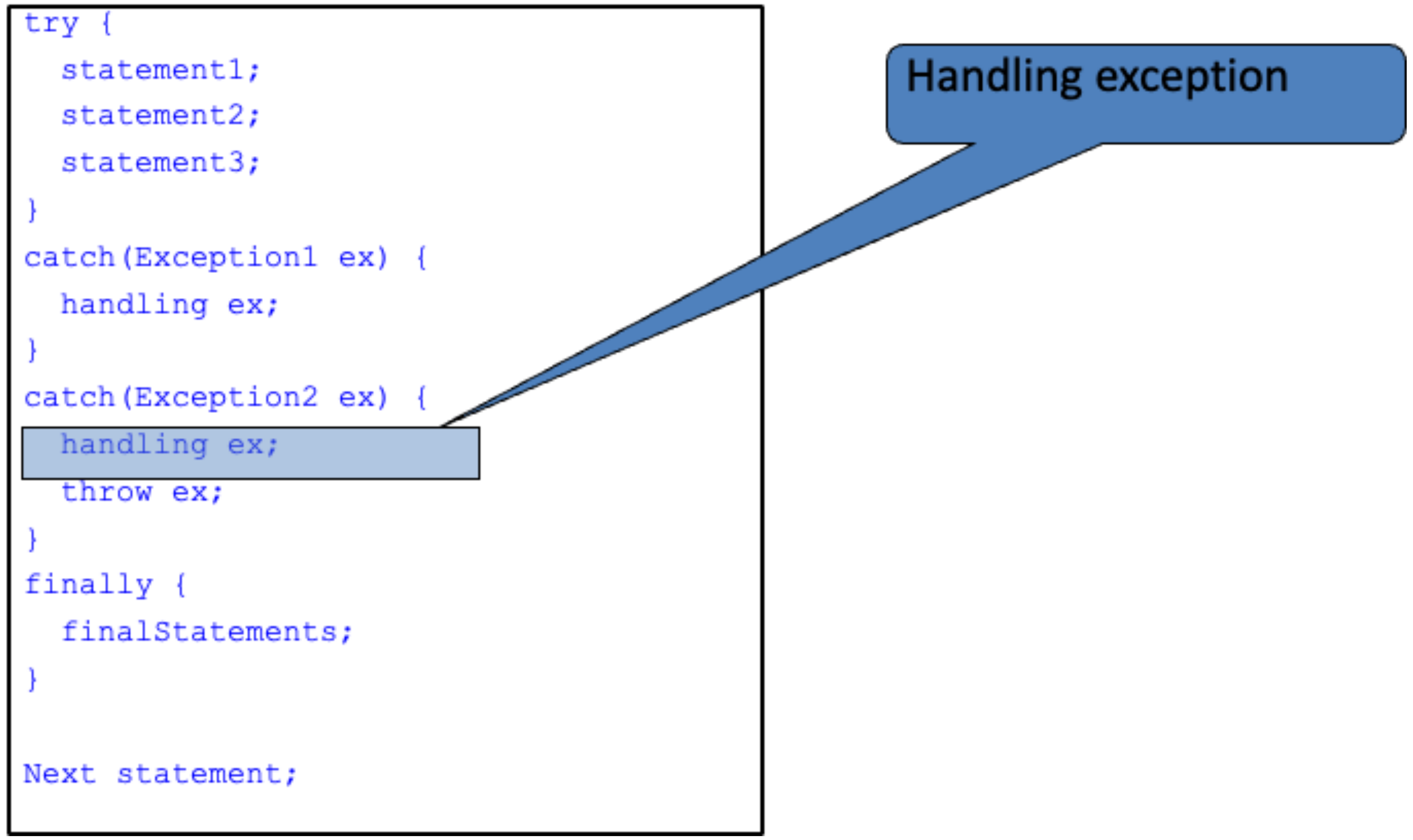
```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution



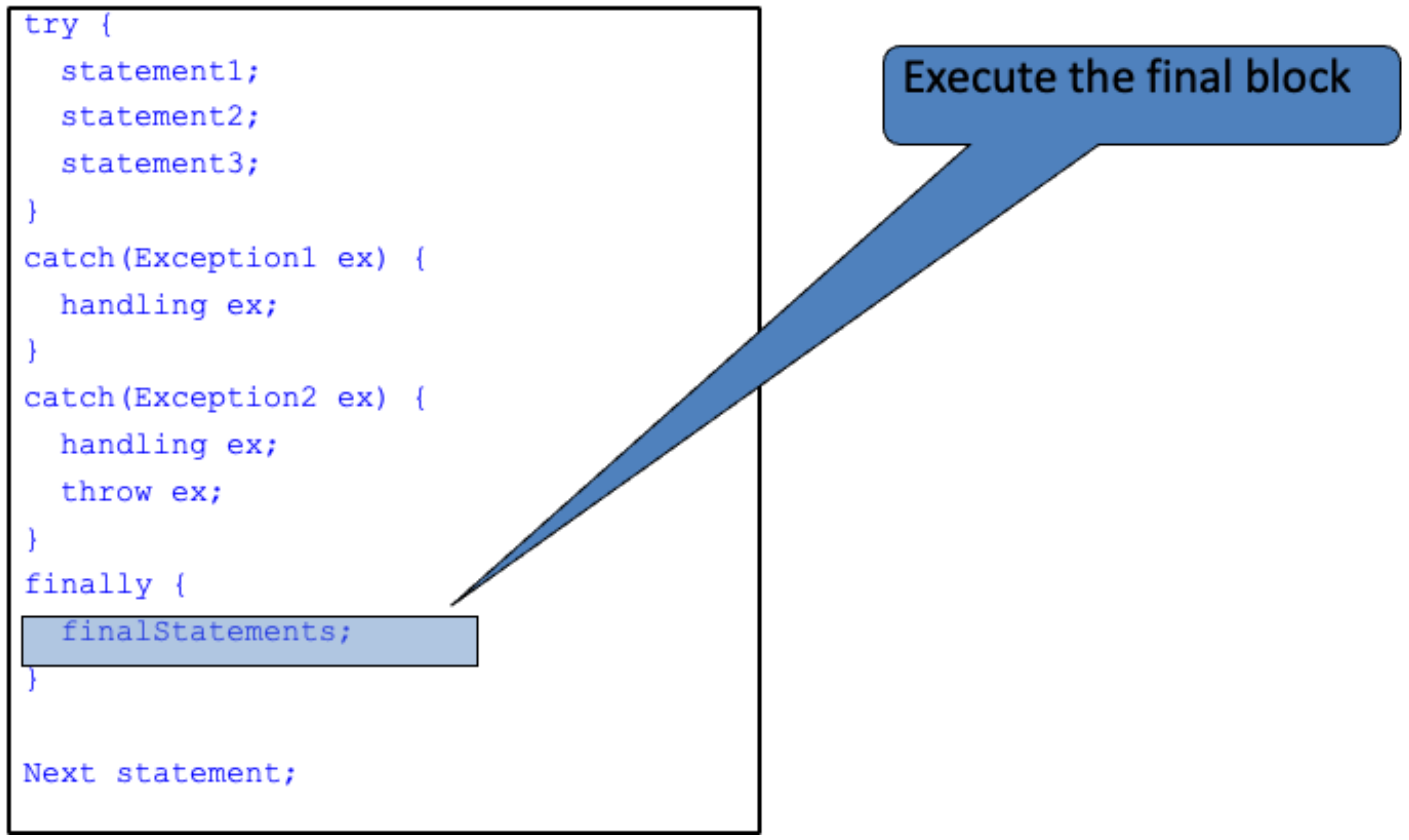
```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution

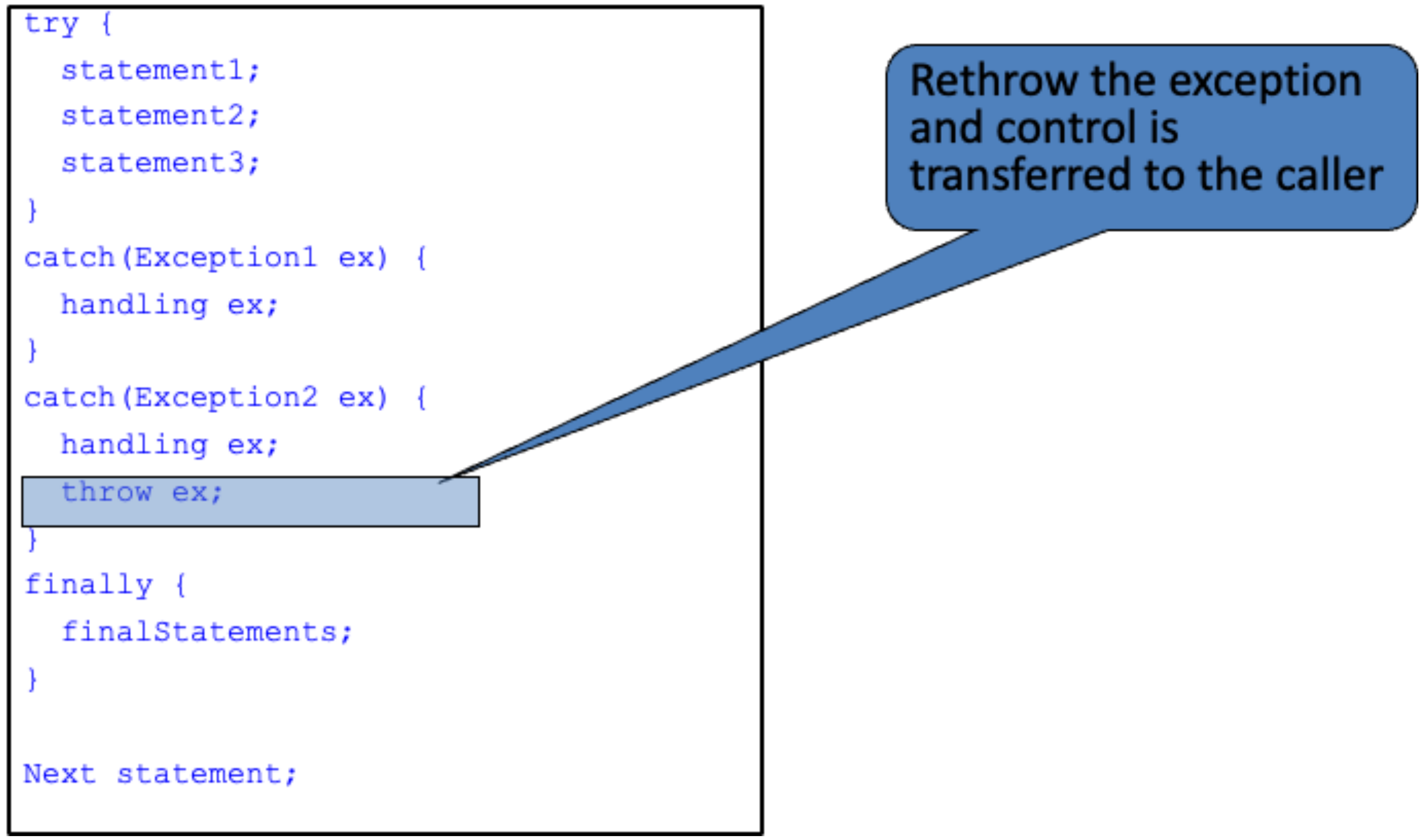


```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution


```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Trace a Program Execution



```
try {
  statement1;
  statement2;
  statement3;
}
catch(Exception1 ex) {
  handling ex;
}
catch(Exception2 ex) {
  handling ex;
  throw ex;
}
finally {
  finalStatements;
}

Next statement;
```

Rethrow the exception and control is transferred to the caller