It is common sense to take a method and try it. If it fails, admit it frankly and try another. But above all, try something.
—Franklin Delano Roosevelt

O! throw away the worser part of it,
And live the purer with the other half.
—William Shakespeare

If they're running and they don't look where they're going
I have to come out from somewhere and catch them.
—Jerome David Salinger

O infinite virtue! com'st thou smiling from the world's great snare uncaught?
—William Shakespeare

11

Exception Handling

Objectives

In this Chapter you'll learn:

- What exceptions are.
- How exception and error handling works.
- To use `try`, `throw` and `catch` to detect, indicate and handle exceptions, respectively.
- To use the `finally` block to release resources.
- How stack unwinding enables exceptions not caught in one scope to be caught in another.
- How stack traces help in debugging.
- How exceptions are arranged in an exception-class hierarchy.
- To declare new exception classes.
- To create chained exceptions that maintain complete stack-trace information.

Self-Review Exercises

11.1 List five common examples of exceptions.

ANS: Memory exhaustion, array index out of bounds, arithmetic overflow, division by zero, invalid method parameters.

11.2 Give several reasons why exception-handling techniques should not be used for conventional program control.

ANS: (a) Exception handling is designed to handle infrequently occurring situations that often result in program termination, not situations that arise all the time. (b) Flow of control with conventional control structures is generally clearer and more efficient than with exceptions. (c) The additional exceptions can get in the way of genuine error-type exceptions. It becomes more difficult for you to keep track of the larger number of exception cases.

11.3 Why are exceptions particularly appropriate for dealing with errors produced by methods of classes in the Java API?

ANS: It's unlikely that methods of classes in the Java API could perform error processing that would meet the unique needs of all users.

11.4 What is a "resource leak"?

ANS: A "resource leak" occurs when an executing program does not properly release a resource when it's no longer needed.

11.5 If no exceptions are thrown in a `try` block, where does control proceed to when the `try` block completes execution?

ANS: The `catch` blocks for that `try` statement are skipped, and the program resumes execution after the last `catch` block. If there is a `finally` block, it's executed first; then the program resumes execution after the `finally` block.

11.6 Give a key advantage of using `catch( Exception` *exceptionName* `)`.

ANS: The form `catch( Exception` *exceptionName* `)` catches any type of exception thrown in a `try` block. An advantage is that no thrown `Exception` can slip by without being caught. You can then decide to handle the exception or possibly rethrow it.

11.7 Should a conventional application catch `Error` objects? Explain.

ANS: `Errors` are usually serious problems with the underlying Java system; most programs will not want to catch `Errors` because they will not be able to recover from them.

11.8 What happens if no `catch` handler matches the type of a thrown object?

ANS: This causes the search for a match to continue in the next enclosing `try` statement. If there is a `finally` block, it will be executed before the exception goes to the next enclosing `try` statement. If there are no enclosing `try` statements for which there are matching `catch` blocks and the exceptions are declared (or unchecked), a stack trace is printed and the current thread terminates early. If the exceptions are checked, but not caught or declared, compilation errors occur.

11.9 What happens if several `catch` blocks match the type of the thrown object?

ANS: The first matching `catch` block after the `try` block is executed.

11.10 Why would a programmer specify a superclass type as the type in a `catch` block?

ANS: This enables a program to catch related types of exceptions and process them in a uniform manner. However, it's often useful to process the subclass types individually for more precise exception handling.

11.11 What is the key reason for using `finally` blocks?

ANS: The `finally` block is the preferred means for releasing resources to prevent resource leaks.

11.12 What happens when a `catch` block throws an `Exception`?

ANS: First, control passes to the `finally` block if there is one. Then the exception will be processed by a `catch` block (if one exists) associated with an enclosing `try` block (if one exists).

11.13 What does the statement `throw` *exceptionReference* do in a `catch` block?

ANS: It rethrows the exception for processing by an exception handler of an enclosing `try` statement, after the `finally` block of the current `try` statement executes.

11.14 What happens to a local reference in a `try` block when that block throws an `Exception`?

ANS: The reference goes out of scope. If the referenced object becomes unreachable, the object can be garbage collected.

Exercises

NOTE: Solutions to the programming exercises are located in the `ch11solutions` folder. Each exercise has its own folder named `ex11_##` where ## is a two-digit number representing the exercise number. For example, exercise 11.17's solution is located in the folder `ex11_17`.

11.15 List the various exceptional conditions that have occurred in programs throughout this text so far. List as many additional exceptional conditions as you can. For each of these, describe briefly how a program typically would handle the exception by using the exception-handling techniques discussed in this chapter. Typical exceptions include division by zero and array index out of bounds.

ANS: A few examples are: Division by zero—catch the exception, inform user of the attempt to divide by zero. Array subscript out of bounds—catch the exception, print an error message telling the user what index was being referenced incorrectly, and exit the program in a controlled manner. Bad cast—catch the exception and either cast it to the proper type if that can be determined, or print an error message indicating what the bad cast was, and exit the program. Invalid input—catch the exception and inform the user that the input cannot be converted to the proper type.

11.16 Until this chapter, we've found dealing with errors detected by constructors to be a bit awkward. Explain why exception handling is an effective means for dealing with constructor failure.

ANS: A thrown exception passes to the outside world the information about the failed constructor and the responsibility to deal with the failure. Exceptions thrown in constructors cause objects built as part of the object being constructed to be marked for eventual garbage collection.