Solved Exercises

1 List five common examples of exceptions.

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

3 What is a “resource leak”?

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

5 Give a key advantage of using catch(Exception exceptionName).

6 Should a conventional application catch Error objects? Explain.

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

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

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

10 What is the key reason for using finally blocks?

11 What happens when a catch block throws an Exception?

12 What does the statement throw exceptionReference do in a catch block?

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

Answers to Solved Exercises

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

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

3 A “resource leak” occurs when an executing program does not properly release a resource when it’s no longer needed.

4 The catch blocks for that try statement are skipped, and the program resumes execution

after the last catch block. If there's a finally block, it's executed first; then the program resumes execution after the finally block.

5 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 handle the exception or rethrow it.

6 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.

7 This causes the search for a match to continue in the next enclosing try statement. If there's 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.

8 The first matching catch block after the try block is executed.

9 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.

10 The finally block is the preferred means for releasing resources to prevent resource leaks.

11 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).

12 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.

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

Exercises

14 (**Exceptional Conditions**) 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.

15 (**Exceptions and Constructor Failure**) 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.

16 (**Catching Exceptions with Superclasses**) Use inheritance to create an exception superclass (called ExceptionA) and exception subclasses ExceptionB and ExceptionC, where ExceptionB inherits from ExceptionA and ExceptionC inherits from ExceptionB. Write a program to demonstrate that the catch block for type ExceptionA catches exceptions of types ExceptionB and ExceptionC.

17 (**Catching Exceptions Using Class Exception**) Write a program that demonstrates how various exceptions are caught with

catch (Exception exception)

This time, define classes ExceptionA (which inherits from class Exception) and ExceptionB (which inherits from class ExceptionA). In your program, create try blocks that throw exceptions of types ExceptionA, ExceptionB, NullPointerException and IOException. All exceptions should be caught with catch blocks specifying type Exception.

18 (**Order of catch Blocks**) Write a program demonstrating that the order of catch blocks is important. If you try to catch a superclass exception type before a subclass type, the compiler should generate errors.

19 (**Constructor Failure**) Write a program that shows a constructor passing information about constructor failure to an exception handler. Define class **SomeClass**, which throws an **Exception** in the constructor. Your program should try to create an object of type **SomeClass** and catch the exception that's thrown from the constructor.

20 (**Rethrowing Exceptions**) Write a program that illustrates rethrowing an exception. Define methods **someMethod** and **someMethod2**. Method **someMethod2** should initially throw an exception. Method **someMethod** should call **someMethod2**, catch the exception and rethrow it. Call someMethod from method **main**, and catch the rethrown exception. Print the stack trace of this exception.

21 (**Catching Exceptions Using Outer Scopes**) Write a program showing that a method with its own try block does not have to catch every possible error generated within the **try**. Some exceptions can slip through to, and be handled in, other scopes.