Chapter 11

Debugging and Handling Exceptions

At a Glance

Instructor's Manual Table of Contents

Lecture Notes

Chapter Overview

This chapter introduces you to one of the tools available in Visual Studio, the Debugger, which can be used to observe the run-time environment and locate logic errors. Using the Debugger, you can stop program execution and inspect the values stored in memory.

This chapter also introduces a special type of error, called an exception. Exceptions are usually associated with error conditions that cause abnormal terminations if they are not handled. The try…catch…finally structured exception-handling techniques are demonstrated for dealing with these unexpected conditions.

Chapter Objectives

In this chapter, students will:

- Learn about exceptions, including how they are thrown and caught
- Gain an understanding of the different types of errors that are found in programs
- Look at debugging methods available in Visual Studio
- Discover how the Debugger can be used to find run-time errors
- Become aware of and use exception-handling techniques to include try...catch...finally clauses
- Explore the many exception classes and learn how to write and order multiple catch clauses

Instructor Notes

ERRORS

Visual Studio integrated development environment (IDE) reports errors in your program as soon as it is able to detect a problem. Compiler errors, associated with a language rule violation, are the easiest to discover and eliminate.

Run-Time Errors
Sometimes a program stops during execution. Other times, output is produced, but the output might not be correct all the time. Logic errors are normally associated with programs that run but produce incorrect results.

Debugging in C#
Visual Studio has a Debugger that enables you to observe the run-time behavior of your program and help locate logic errors. The Debugger lets you break, or halt, the execution of the program to examine the code, evaluate variables in the program, and view the memory space used by your application.

You can step through an application, checking the values of variables as each line is executed, set breakpoints that when reached, cause the program to be suspended so you can examine, and even change, the value of variables.

To use the Debugger, run your program by selecting the **Start Debugging** option from the **Debug** menu.

Breakpoints
Breakpoints are markers that are placed in an application, indicating where the program should halt execution. When the break occurs, the program and the Debugger are said to be in break mode. To set a breakpoint, select **Toggle Breakpoint** from the **Debug** menu or use the F9 keyboard shortcut.

Continue
After reviewing the variables and expressions, pressing F5 or selecting **Continue** from the **Debug** menu takes the program out of break mode and restores it to a run-time mode. If more than one breakpoint is set, selecting **Continue** causes the program to execute from the halted line until it reaches the second breakpoint.

Stepping Through Code
The Debug menu offers Step Into (F11), Step Over (F10), and Step Out (Shift+F11) commands for stepping through code while you are in break mode. The Step Into command halts at the first line of code inside a called method. The Step Over command executes the entire method called before it halts. If you are executing statements inside a method and want to return to the calling method, this third command, Step Out, is useful. Step Out causes the rest of the program statements in the method to be executed and then returns control to the method that called it.

Watches
The Watch window lets you type in one or more variables or expressions that you want to observe while the program is running. Unlike the Locals window, which shows all variables currently in scope, you selectively identify the variables or expressions for the Watch window.

Quick Quiz

1. True or False: Run-time errors are detected by the compiler.
   Answer: False

2. A _____ is a marker that is placed in an application, indicating where the program should halt execution.
   Answer: breakpoint

3. True or False: Unlike a Locals window, a Watch window can examine any variable.
   Answer: True

4. Identify two of the three ways you can step through an application using the Debugger.
   Answer: Step Into and Step Over

EXCEPTIONS

Some circumstances are beyond the control of the programmer. Given perfect situations for running applications, programs may perform beautifully. But, some circumstances are beyond the control of the programmer, and unless provisions are made for handling exceptions, your program may crash or produce erroneous results.

There are a number of things you can do to keep your program from crashing. For example, you can include if statements that check values used as input to ensure the value is numeric, prior to parsing or converting the string value to its numeric equivalent.

You may receive an error message indicating "Unhandled exception" during run time with the entire application halting. Usually no errors are detected when the program is compiled. It is only when the application runs that the program crashes and the message is displayed. Sometimes the message will indicate what caused the problem. If the program is running within Visual Studio, the problem code may be highlighted in yellow with buttons available to stop, break or stop debugging, or continue.

Raising an Exception
When a program encounters an error that it cannot recover from, it raises or throws an exception. Execution halts in the current method and the Common Language Runtime (CLR) attempts to locate an exception handler to handle the exception. An exception handler is a block of code that is executed when an exception occurs.

If an exception handler is found in the current method, control is transferred to that code. If no method is found in the current method, that method is halted and the exception is thrown back to the parent method to handle it. If more than two methods are used, the exception continues to be thrown backwards until it reaches the topmost method. If none of the methods includes code to handle the error, the entire application is halted. This can be very abrupt.

Bugs, Errors, and Exceptions
In some instances, you can use selection statements, such as if…else, to programmatically prevent your programs from crashing. Bugs differ from exceptions in that they are normally labeled “programmer mistakes” that should be caught and fixed before an application is released. In addition to bugs, programs may experience errors because of user actions. These actions may cause exceptions to be thrown. Entering the wrong type of data from the keyboard is an example of a common user mistake.

A program starts execution in the Main( ) method. From Main( ), it calls on other methods, which can also call on other methods, and so on. When execution reaches the bottom of a given method, control is returned to the method that called it. This continues until control eventually returns to the end of the Main( ) method, where the program finishes its execution. A stack is used to keep up with the execution chain. When an unhandled exception message is displayed in Visual Studio, click the Details button to view a stack trace of methods with the method that raised the exception listed first. A stack trace is a listing of all the methods that are in the execution chain when the exception is thrown. If none of the methods listed in the stack trace include code to handle the type of exception that is encountered, the unhandled exception message is displayed and the program is halted.

Quick Quiz

1. True or False: Bugs are unexpected conditions that should be treated as exceptions.
   Answer: False

2. A listing of all the methods that are in the execution chain when the exception is thrown is called a _____.
   Answer: stack trace

3. True or False: All program errors should be treated as exceptions.
   Answer: False

4. How might you write program statements to keep division from zero from crashing your program?
   Answer: Use the if…else statement to check the divisor. If it is zero, do not do the division.

EXCEPTION-HANDLING TECHNIQUES

If an event that creates a problem happens frequently, it is best to write program statements, using conditional expressions, to check for the potential error and provide program instructions for what to do when that problem is encountered. Exceptions are those events from which your program would not be able to recover, such as attempting to read from a data file that does not exist.

Try…Catch…Finally Blocks
The code that may create a problem is placed in the try block. The code to deal with the problem (the exception handler) is placed in catch blocks, which are also called catch clauses. The code that you want executed whether an exception is thrown or not is placed in the finally block. More than one catch clause may be included. One is required. If you include an exception type as part of the argument list, only exceptions that match the type listed are handled by that catch clause. The finally clause is also optional.

Control is not returned back into the try block after an exception is thrown. The statement that creates a problem in the try block is the last one tried in the try clause.

Generic Catches
If you omit the argument list with the catch clause, it is considered a generic catch. Any exception that is thrown is handled by executing the code within that catch block. The problem with using a generic catch is you are never quite sure what caused the exception to be thrown. You can debug more easily if you know what caused the exception to be thrown.

Exception Object
When an exception is thrown, an object is created. The base class for exceptions is Exception; it is part of the System namespace. Exception objects have properties and methods.

The catch clause may list an exception class name and an object identifier inside parentheses following the catch keyword. In order to use any of the properties of the exception object that is created, you must have an object name. Using the catch { } without an exception type does not give you access to an object.

A property of the base class, Message, returns a string describing the exception. The Message property, associated with the object name identified in the catch clause's parenthesized argument list, is used inside the catch clause to display a message describing the exception. By specifying more than one exception filter, you can write code in the catch clauses that is specific to the particular exception thrown.

Quick Quiz

1. The _____ property of the base exception class can be used to display a string describing the exception.
   Answer: Message

2. True or False: The base class for all exception classes is Exception.
   Answer: True

3. True or False: The argument to the catch clause is the exception class identifier and an object of that type.
   Answer: True

4. What is the disadvantage of using generic catch clauses?
   Answer: You may not know exactly what caused the exception to be thrown. No object is available to display information about the exception.

EXCEPTION CLASSES

There are a number of different types of exceptions that can be thrown.

Derived Classes of the Base Exception Class
User-defined exceptions derive from ApplicationException exception class and system supplied exceptions derive from the SystemException class.

SystemException Class
SystemException adds no functionality to classes. Except for its constructor, the SystemException class has no additional properties or methods other than those derived from the Exception and Object classes. Over 70 classes derive from the SystemException class. Review Table 11-2 for a short list of some of the more common exceptions that are thrown. In addition to these 70 classes, a number of other classes derive from these classes. For example, one of the derived classes of the System.ArithmeticException class is the System.DivideByZeroException class.

Filtering Multiple Exceptions
Multiple catch clauses can be included with a single try clause. If more than one is included, the order of placement of these clauses is important. They should be placed from the most specific to the most generic. Because all exception classes derive from the Exception class, if you are including the Exception class, it should always be placed last.

Custom Exceptions
You can write your own exception classes. The only requirement is that custom exceptions must derive from the ApplicationException class. Creating an exception class is no different from creating any other class.

Throwing an Exception
An exception object is instantiated when "an exceptional condition occurs." With user-defined exceptions, the CLR does not throw the exception. Instead, the exception is thrown by the program using the throw keyword. When the object is thrown in a method, the exception object propagates back up the call chain, first stopping at the method that called it to see if a catch is available to handle it.

Input Output (IO) Exceptions
Exceptions are extremely useful for applications that process or create stored data. The primary exception class for files is System.IO.IOException. System.IO.IOException derives from the SystemException class. An IO.IOException exception is thrown when a specified file or directory is not found, if you attempt to read beyond the end of a file, or if there are problems loading or accessing the contents of a file. Review Table 11-3 for a list of classes derived from the IO.IOException class.

Quick Quiz

1. True or False: User-defined exceptions derive from SystemException class.
   Answer: False

2. True or False: To filter more than one exception, include the most general first, followed by the most specific.
   Answer: True

3. The primary exception class for deal with input/output files is the _____class.
   Answer: System.IO.IOException

4. How does throwing an exception that derives from the SystemException class differ from throwing one that derives from the ApplicationException class?
   Answer: With user-defined exceptions derived from the ApplicationException, the CLR does not throw the exception. Instead, the exception is thrown by the program using the throw keyword. The CLR throws exceptions derived from the SystemException class.

PROGRAMMING EXAMPLE: ICW WATERDEPTH APPLICATION

This example demonstrates exception-handling techniques. Three classes are constructed for the application. One of the classes is a programmer-defined custom exception class. It inherits methods and properties from the ApplicationException class and is included to illustrate throwing an exception using program statements. The graphical user interface enables the user to enter location name, state where it is located, mile number, and four separate days of water depth at low and high tide. When invalid data is entered, an exception is thrown.

Two additional classes are defined for the application. The business logic for the application is separated from the presentation details. The class that defines the graphical user interface makes use of a try...catch block with multiple catch clauses. This class is used to input the data. After the data is retrieved, it is used to instantiate an object of the third class, the ShoalArea class. Output from the application is displayed in a Windows dialog box.

Discussion Questions

Some interesting topics of discussion in this chapter include:

- How can you use the Debugger to help you desk check a solution?
- When should you write program statements such as if…else to deal with potential problems versus writing try…catch…finally clauses?
- Why would you ever need to write a custom exception class?

Projects to Assign

All of the Multiple Choice Exercises, Problems 1-20
Odd numbered Short Answer Exercises, Problems 21 - 25
Programming Exercises, Problems 2, 6, 8, and 9

Key Terms

- **Breakpoint**: selected line in your program that when reached, the program is suspended or placed in break mode
- **bug**: normally labeled “programmer mistakes” that should be caught and fixed before an application is released
- **catch clause**: code written to deal with an exception; also called the exception handler or catch blocks
- **checked exception**: type of exception that must include exception handling techniques if you use a specific construct
- **debugging**: methodical process of finding and reducing bugs or defects in a computer program
- **errors**: problems created for the program due to user actions
- **exception handler**: a block of code that is executed when an exception occurs

- **exceptions**: unexpected conditions usually associated with error conditions that cause abnormal terminations if they are not handled
- **generic catch clause**: any exception that is thrown is handled by executing the code within that catch block
- **logic error**: form of run-time error normally associated with programs that run but produce incorrect results.
- **raise an exception**: program encounters an error such as division by zero during run time that the program cannot recover from
- **stack trace**: a listing of all the methods that are in the execution chain when an exception is thrown
- **Step Into command**: halts at the first line of code inside a called method
- **Step Out command**: returns control to the method that called it when you are executing statements inside a method
- **Step Over command**: executes the entire method called before it halts
- **thrown back**: when the current method does not contain an exception handler, that method is halted, and the parent method that called the method gets the exception to see if it can handle the exception
- **throws an exception**: program encounters an error such as division by zero during run time that the program cannot recover from
- **unhandled exception**: when the CLR handles the exception by halting the application