**PRACTICAL # 08**

**OBJECT:**

C# Exception Handling

**THEORY:**

Some errors are not detectable at compile time. Such errors are usually caught at runtime and called Exceptions.

C# supports exception-handling to handle such runtime errors. Exception handling automates much of the error-handling code that otherwise had to be entered manually into a program. For example, in a computer language without exception handling, error codes must be returned when a method fails, and these values must be checked manually each time the method is called.

Exception handling allows your program to define an exception handler, that is executed automatically when an error occurs. It is not necessary to manually check the success or failure of each specific operation or method call.

C# defines standard exceptions for common runtime errors, such as divide-by-zero and index-out-of-range etc. All the exception classes in C# inherit a common base class System.Exception. An important subclass of Exception is SystemException, from which all exceptions generated by the C# runtime system (that is, the CLR) are derived. SystemException does not add anything to Exception. It simply defines the top of the standard exceptions hierarchy.

Program statements that may generate exceptions are contained within a ***try*** block. If an exception occurs within the try block, it is thrown. Your code can catch this exception using ***catch*** and handle it in some rational manner. System-generated exceptions are automatically thrown by the runtime system. To manually throw an exception, use the ***throw*** keyword. Any code that absolutely must be executed upon exiting from a try block is put in a finally block.

**Program:**

**Using try and catch**

The try and catch keywords work together to catch and handle an exception. Here is the general form of the try/catch exception-handling blocks:

*try {*

*// block of code to monitor for errors*

*}*

*catch (ExcepType1 exOb) {*

*// handler for ExcepType1*

*}*

*catch (ExcepType2 exOb) {*

*// handler for ExcepType2*

*}*

ExcepType is the type of exception that has occurred. When an exception is thrown, it is caught and processed by its corresponding catch clause. The type of the exception determines which catch is executed. That is, if the exception type specified by a catch matches that of the exception, then that catch is executed and the exception variable exOb will receive its value.

The following program demonstrates an index-out-of-range exception.

*static void Main() {*

*int[] nums = new int[4];*

*try {*

*Console.WriteLine("Before exception is generated.");*

*// Generate an index out-of-bounds exception.*

*for(int i=0; i < 6; i++) {*

*nums[i] = i;*

*Console.WriteLine("nums[{0}]: {1}", i, nums[i]);*

*}*

*Console.WriteLine("this won't be displayed");*

*}*

*catch (IndexOutOfRangeException ex) {*

*// Catch the exception.*

*Console.WriteLine("Index out-of-bounds!");*

*}*

*Console.WriteLine("After catch block.");*

*}*

In this program, nums is an int array of four elements. However, the for loop tries to index nums from 0 to 5, which causes an IndexOutOfRangeException to occur when an index value of 4 is used. Once the exception occurs, the catch block of IndexOutOfRangeException gets executed.

**Uncaught Exception**

When an exception is thrown, it must be caught by some piece of code, somewhere. In general, if your program does not catch an exception, it will be caught by the runtime system. The runtime system will report an error and terminate the program abruptly. For instance, in this example, the index out-of-bounds exception is not caught by the program:

*static void Main() {*

*int[] nums = new int[4];*

*Console.WriteLine("Before exception is generated.");*

*for(int i=0; i < 10; i++) {*

*nums[i] = i;*

*Console.WriteLine("nums[{0}]: {1}", i, nums[i]);*

*}*

*}*

When the array index error occurs, execution is halted with the following error message:

*Unhandled Exception: System.IndexOutOfRangeException:*

*Index was outside the bounds of the array. at NotHandled.Main()*

**ACTIVITIES**

**Activity 1**

Write a program for divide-by-zero exception.

**Activity 2**

Write an example program for file not found exception.

**REVIEW QUESTIONS**

1. What is exception handling?
2. Which is the base class for all exception classes?
3. Can a try block have multiple catch blocks?
4. What is the purpose of finally block?
5. What happens if you don’t catch an exception?