**Lab # 9**

**Objective**

***Exception Handling***

An exception is a problem that arises during the execution of a program. A C# exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions are unforeseen errors that happen in your programs. Most of the time, you can, and should, detect and handle program errors in your code. For example, validating user input, checking for null objects, and verifying the values returned from methods are what you expect, are all examples of good standard error handling that you should be doing all the time.

However, there are times when you don't know if an error will occur. For example, you can't predict when you'll receive a file I/O error, run out of system memory, or encounter a database error. These things are generally unlikely, but they could still happen and you want to be able to deal with them when they do occur. This is where exception handling comes in.

Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try, catch, finally, and throw.

* **try**: A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
* **catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
* **finally**: The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
* **throw**: A program throws an exception when a problem shows up. This is done using a throw keyword.

**Syntax**

**try**

**{**

**// statements causing exception**

**}**

**catch( ExceptionName e1 )**

**{**

**// error handling code**

**}**

**catch( ExceptionName e2 )**

**{**

**// error handling code**

**}**

**catch( ExceptionName eN )**

**{**

**// error handling code**

**}**

**finally**

**{**

**// statements to be executed**

**}**

**Exception Classes**

C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the System.Exception class. Some of the exception classes derived from the System.Exception class are the System.ApplicationException andSystem.SystemException classes.

The System.Application Exception class supports exceptions generated by application programs. Hence the exceptions defined by the programmers should derive from this class.

|  |  |
| --- | --- |
| **Exception Class** | **Description** |
| System.IO.IOException | Handles I/O errors. |
| System.IndexOutOfRangeException | Handles errors generated when a method refers to an array index out of range. |
| System.ArrayTypeMismatchException | Handles errors generated when type is mismatched with the array type. |
| System.NullReferenceException | Handles errors generated from deferencing a null object. |
| System.DivideByZeroException | Handles errors generated from dividing a dividend with zero. |
| System.InvalidCastException | Handles errors generated during typecasting. |
| System.OutOfMemoryException | Handles errors generated from insufficient free memory. |
| System.StackOverflowException | Handles errors generated from stack overflow. |

**Example-1**

using System;

using System.Collections;

namespace ConsoleApplication1

{

class Program

{

static void Main(string[] args)

{

int[] numbers = new int[2];

numbers[0] = 23;

numbers[1] = 32;

numbers[2] = 42;

foreach(int i in numbers)

Console.WriteLine(i);

Console.ReadLine();

}

}

}

In previous example, we have defined an array of integers with room for 2 items, yet we try to use 3 spaces in it. Obviously, this leads to an error, which you will see if you try to run this example. If you know that an error might occur, you should handle it. This is where exceptions are used. Here is a slightly modified version of the code from above:

int[] numbers = new int[2];

try

{

numbers[0] = 23;

numbers[1] = 32;

numbers[2] = 42;

foreach(int i in numbers)

Console.WriteLine(i);

}

catch

{

Console.WriteLine("Something went wrong!");

}

Console.ReadLine();

One more thing you should know about concerning exceptions is the finally block. The finally block can be added to a set of catch blocks, or be used exclusively, depending on your needs. The code within the finally block is always run - exception or no exception. It's a good place if you need to close file references or dispose objects you won't need anymore. Since our examples have been pretty simple so far, we haven't really been in need of any cleanup, since the garbage collector handles that. But since will likely run into situations where you need the finally block, here is an extended version of our example:

int[] numbers = new int[2];

try

{

numbers[0] = 23;

numbers[1] = 32;

numbers[2] = 42;

foreach(int i in numbers)

Console.WriteLine(i);

}

catch(IndexOutOfRangeException ex)

{

Console.WriteLine("An index was out of range!");

}

catch(Exception ex)

{

Console.WriteLine("Some sort of error occured: " + ex.Message);

}

finally

{

Console.WriteLine("It's the end of our try block. Time to clean up!");

}

Console.ReadLine();

If you run the code, you will see that both the first exception block and the finally block is executed. If you remove the line that adds the number 42 to the array, you will see that only the finally block is reached.  
Another important part you should know about exceptions, is how they impact the method in which the exceptions occur. Not all unhandled exceptions are fatal for your application, but when they aren't, you should not expect the remaining code of the method to be executed. On the other hand, if you do handle the exception, only the lines after the try block will be executed. In our example, the loop that outputs the values of the array is never reached, because the try block goes straight to the catch/finally block(s) once an exception is thrown. However, the last line, where we read from the console to prevent the application from exiting immediately, is reached. You should always have this in mind when you construct try blocks.

**TASK :**

1. Write a program of throwing an exception when dividing by zero condition occurs?
2. Write a program that if the user program wants to issue a book in library. The user program then can decide to generate a message and tell the user to check if the book is already issued etc?