**Lab 12**

**Implementing Exception Handling**

In this lab we will learn how to handle exceptions using the try-catch construct, how to pass them to the calling methods and how to throw standard or our own exceptions using the throw construct. We will give various examples for using exceptions. We will look at the types of exceptions and the exceptions hierarchy in the .NET Framework. At the end, we will look at the advantages of using exceptions, best practices and how to apply them in different situations.

**What are Exceptions**

Exceptions are a type of error that occurs during the execution of an application. Errors are typically problems that are not expected. Whereas, exceptions are expected to happen within the

application’s code for various reasons.

Applications use exception handling logic to explicitly handle the exceptions when they

happen. Exceptions can occur for a wide variety of reasons. From the infamous NullReferenceException to a database query timeout.

**The Anatomy of C# Exceptions**

Exceptions allow an application to transfer control from one part of the code to another. When an exception is thrown, the current flow of the code is interrupted and handed back to a parent try catch block. C# exception handling is done with the follow keywords: try, catch, finally, and throw

* **try** – A try block is used to encapsulate a region of code. If any code throws an exception within that try block, the exception will be handled by the corresponding catch.
* **catch** – When an exception occurs, the Catch block of code is executed. This is where you are able to handle the exception, log it, or ignore it.
* **finally** – The finally block allows you to execute certain code if an exception is thrown or not. For example, disposing of an object that must be disposed of.
* **throw** – The throw keyword is used to actually create a new exception that is the bubbled up to a try catch finally block.

Your exception handling code can utilize **multiple C# catch statements for different types of exceptions**.

**Common .NET Exceptions**

Proper exception handling is critical to all application code. There are a lot of standard exceptions that are frequently used. The most common being the dreaded null reference exception. These are some of the common C# Exception types that you will see on a regular basis.

The follow is a list of common .NET exceptions:

* **System.NullReferenceException** – Very common exception related to calling a method on a null object reference
* **System.IndexOutOfRangeException** – Occurs attempting to access an array element that does not exist
* **System.IO.IOException** – Used around many file I/O type operations
* **System.Net.WebException** – Commonly thrown around any errors performing HTTP calls
* **System.Data.SqlClient.SqlException** – Various types of SQL Server exceptions
* **System.StackOverflowException** – If a method calls itself recursively, you may get this exception
* **System.OutOfMemoryException** – If your app runs out of memory
* **System.InvalidCastException** – If you try to cast an object to a type that it can’t be cast to
* **System.InvalidOperationException** – Common generic exception in a lot of libraries
* **System.ObjectDisposedException** – Trying to use an object that has already been

**Sample File Code 1:**

using System; using System.IO; using System.Text; public class Writer

{

static string ans="y";

public static void Main(String[] args)

{

Writing();

}

static void Writing()

{

if (ans=="y" || ans=="Y")

{

Console.Write ("Enter the file name: "); string Filename = Console.ReadLine();

if (!File.Exists(Filename))

{

Console.WriteLine("{0} does not exist!",Filename); return;

}

StreamWriter sr = File.AppendText(Filename); Console.Write ("Enter a string to be written to the file: "); String str = Console.ReadLine();

sr.WriteLine(str); sr.Close();

Console.Write ("Do you want to continue [Y/N]: "); ans= Console.ReadLine();

Writing();

}

}

}

**Sample File Code 2:**

static void Main()

{

// These examples assume a "C:\Users\Public\TestFolder" folder on your machine.

// You can modify the path if necessary.

// Example #1: Write an array of strings to a file.

// Create a string array that consists of three lines.

string[] lines = { "First line", "Second line", "Third line" };

// WriteAllLines creates a file, writes a collection of strings to the file,

// and then closes the file. You do NOT need to call Flush() or Close(). System.IO.File.WriteAllLines(@"C:\Users\Public\TestFolder\WriteLines.txt", lines);

}

# Time Boxing

|  |  |  |
| --- | --- | --- |
| **Activity Name** | **Activity Time** | **Total Time** |
| Login Systems + Setting up Visual Studio  Environment | 3 mints + 5 mints | 8 mints |
| Walk through Theory & Tasks | 60 mints | 60 mints |
| Implement Tasks | 80 mints | 80 mints |
| Evaluation Time | 30 mints | 30 mints |
|  | Total Duration | 178 mints |

# Objectives/Outcomes

This Lab exercise delivers the idea/concept of:

* + Creation and implementation of Exception Handling

# Lab Tasks/Practical Work

1. Write a program that takes a positive integer from the console and prints the square root of this integer. If the input is negative or invalid print "Invalid Number" in the console. In all cases print "Good Bye".
2. Write a method ReadNumber(int start, int end) that reads an integer array of 10 values from the console in the range [start…end]. In case the input integer is not valid, or it is not in the required range throw appropriate exception.
3. Write a method that takes as a parameter the name of a text file then, reads the file and returns its content as string. What should the method do if an exception is thrown?