Sir Syed University of Engineering & Technology, Karachi

EXCEPTION HANDLING

Course Code : CS-127

Course Title : Object Oriented Programming

Semester : 2nd

Batch - 2020

Department of Computer Science / Information Technology

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.

- C# provides built-in support to handle the exception using try, catch & finally blocks.

keywords: **try**, **catch**, **finally**, and **throw**

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

Assuming a block raises an exception, a method catches an exception using a combination of the try and catch keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch looks like the following:

```
try
{
// statem ents causing exception
}
catch( ExceptionNam e e1 )
{
// error handling code
}
catch( ExceptionNam e e2 )
{
// error handling code
}
catch( ExceptionNam e eN )
{
// error handling code
}
finally
{
// statem ents to be executed
}
```

C# Exception Classes

All the exception classes in C# are derived from **System.Exception** class.

The list of C# most common exception classes :

Exception	***Description***
System.DivideByZeroException	handles the error generated by dividing a number with zero.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.IO.IOException	handles the Input Output errors.
System.FieldAccessException	handles the error generated by invalid private or protected field access.

Handling Exceptions

- C# provides a structured solution to the exception handling in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

- These error handling blocks are implemented using the **try**, **catch**, and **finally** keywords.

Examples1

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication62
{
    class withoutexception
    {
        public void insert_data_in_array()
        {
            int[] numbers = new int[3];

            numbers[0] = 92;
            numbers[1] = 83;
            numbers[2] = 84;
            numbers[3] = 63;
            foreach(int i in numbers)
            {
                Console.WriteLine(i);
            }

        }

    }
```

Examples1

```
•       class withexception
•         {
•            public void insert_data_in_array()
•            {
•               int[] numbers = new int[3];

•               try
•               {
•                  numbers[0] = 92;
•                  numbers[1] = 83;
•                  numbers[2] = 84;
•                  numbers[3] = 63;
•                  foreach (int i in numbers)
•                  {
•                     Console.WriteLine(i);
•                  }
•               }
•               catch(IndexOutOfRangeException ex)
•               {
•                  Console.WriteLine(" Some Error has occurred  : "+ex.Message);

•               }

•            }

•         }
```

Examples1

```
•  class Program
•     {
•         static void Main(string[] args)
•         {
•             //withoutexception woe = new withoutexception();
•             //woe.insert_data_in_array();
•             withexception we = new withexception();
•             we.insert_data_in_array();
•
•             Console.ReadLine();
•         }
•     }
• }
```

Examples2

```
using System;
using System.Collections.Generic;
using System.Text;

namespace ConsoleApplication62
{

    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine(" Enter value of X ");
                int x = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine(" Enter value of Y ");
                int y = Convert.ToInt32(Console.ReadLine());
                Console.WriteLine(x / y);
            }
            catch (DivideByZeroException ex)
            {
                Console.WriteLine(" Message :" + ex.Message);


            }
```

Examples2

```
•    catch (FormatException ex)
•            {
•                Console.WriteLine(" Message :" + ex.Message);
•
•
•            }
•            finally
•            {
•                Console.WriteLine("  Thanks for using .NET APP......  ");
•            }

•            Console.ReadLine();
•        }
•    }
•  }
```