

## Exception Handling in C#

It is very common for **software applications to get errors** and **exceptions** when executing code. If these **errors are not handled properly**, the **application may crash** and you may not know the root cause of the problem.

**Exceptions** abnormally terminate the execution flow of the program instructions, we need to handle those exceptions.

**Exception handling** is the method of **catching** and **recording these errors** in code so you can fix them. Usually, **errors** and **exceptions** are stored in **log files** or **databases**.

In C#, the exception handling method is implemented using the following statements

- try
- catch
- finally

### **try and catch block**

The **try** encloses the code that **might throw** an exception, whereas the **catch handles** an exception if one exists.

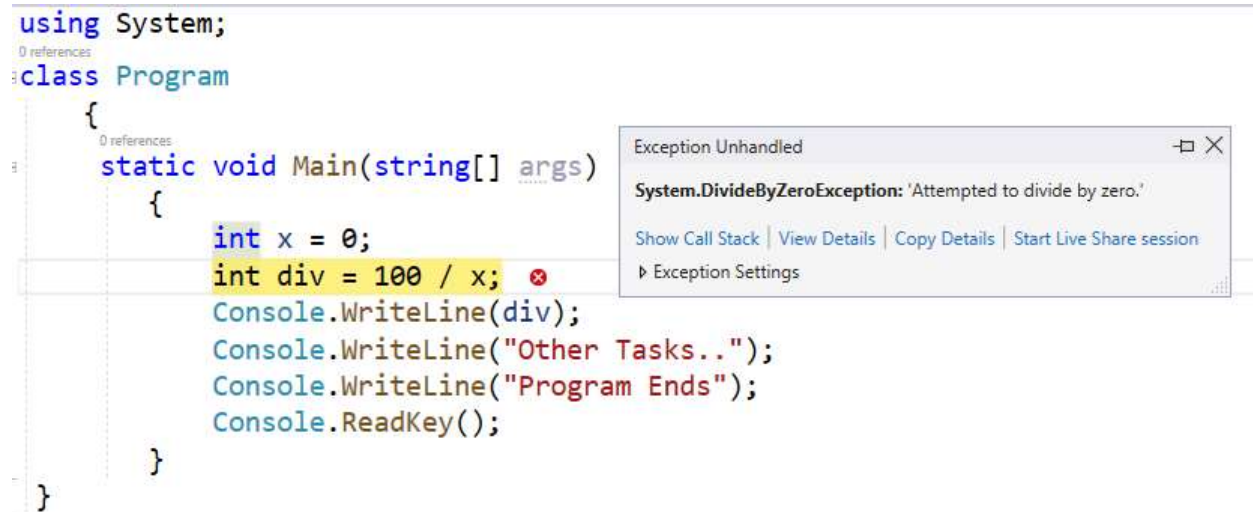
```
try
{
    // code that may raise an exception
}
catch (ExceptionType e)
{
    // code that handles the exception
}
```

Here, we place the code that might generate an exception inside the **try block**. The **try block** then **throws the exception** to the **catch block** which handles the raised exception.

**Let's learn this by example:-** Consider the following programming logic

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int x = 0;
        int div = 100 / x;
        Console.WriteLine(div);
        Console.WriteLine("Other Tasks..");
        Console.WriteLine("Program Ends");
        Console.ReadKey();
    }
}
```

The above program will compile but will show an error during execution. The **division by zero** is a runtime exception, and the program terminates with an error message as given below.



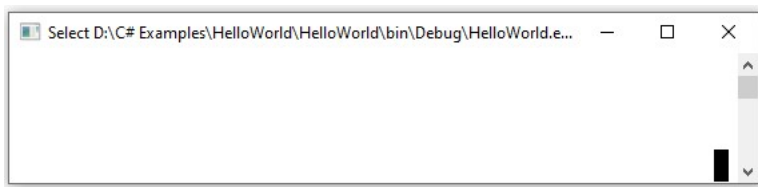
The screenshot shows a C# program in an IDE. The code is as follows:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int x = 0;
        int div = 100 / x;
        Console.WriteLine(div);
        Console.WriteLine("Other Tasks..");
        Console.WriteLine("Program Ends");
        Console.ReadKey();
    }
}
```

An "Exception Unhandled" dialog box is displayed over the code. The message reads: "System.DivideByZeroException: 'Attempted to divide by zero.'" Below the message are links for "Show Call Stack", "View Details", "Copy Details", and "Start Live Share session". There is also an "Exception Settings" link.

Hence, any **uncaught exceptions** in the current context propagate to a higher context and look for an **appropriate catch block** to handle it.

If it can't find suitable catch blocks, the default mechanism of the .NET runtime will **terminate the execution of the entire program** with no output as given below.



### Example 1: Handling the **DivideByZeroException**

The modified form of the above program with an exception-handling mechanism is as follows

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int x = 0;
        int div = 0;
        try
        {
            div = 100 / x;
        }
        catch(DivideByZeroException ex)
        {
            Console.WriteLine("Divide By Zero Error Handled!");
        }
    }
}
```

```

        Console.WriteLine(div);
        Console.WriteLine("Other Tasks..");
        Console.WriteLine("Program Ends");
        Console.ReadKey();
    }
}

```

Here we are using the object of the standard exception class `DivideByZeroException` to handle the exception caused by division by zero.



## Example 2: Handling the `IndexOutOfRangeException`

`IndexOutOfRangeException` will be generated when we access invalid index position in an array or a string.

### In Array...

```

using System;
class Program
{
    static void Main(string[] args)
    {
        string[] colors = { "Red", "Blue", "Green" };
        try
        {
            Console.WriteLine(colors[5]);
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine("Invalid Array Index - Handled!");
        }
        Console.WriteLine("Other Tasks..");
        Console.WriteLine("Program Ends");
        Console.ReadKey();
    }
}

```

Since, there is no element at **index 5** of the **colors array**, the above code **raises an exception**. So we have enclosed this code in the try block.

When the program encounters this code, `IndexOutOfRangeException` occurs. And, the exception is caught by the catch block and executes the code inside the catch block.



Accessing characters from String also generates this `IndexOutOfRangeException`

```
using System;
class Program
{
    static void Main(string[] args)
    {
        String msg = "Machine";
        try
        {
            Console.WriteLine(msg[8]);
        }
        catch(IndexOutOfRangeException ex)
        {
            Console.WriteLine("Invalid String Index - Handled!");
        }
        Console.WriteLine("Other Tasks..");
        Console.WriteLine("Program Ends");
        Console.ReadKey();
    }
}
```

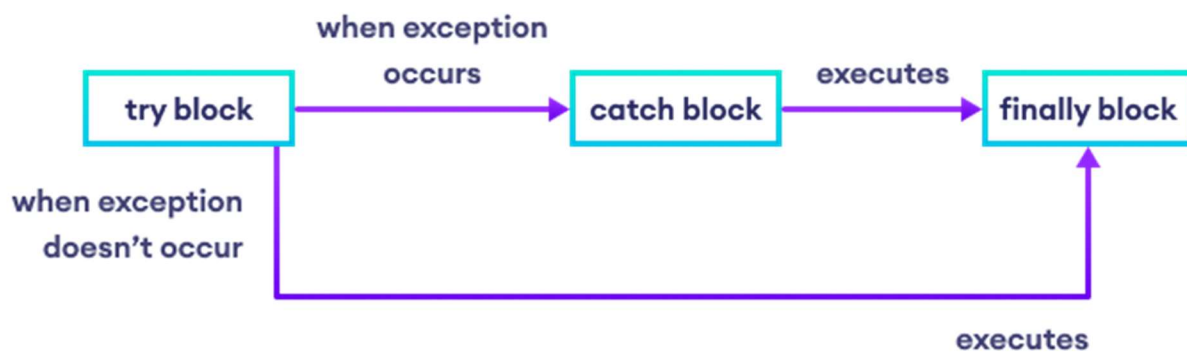
Since, there is no character at index 8 of the string msg, the above code generates `IndexOutOfRangeException`. And, the exception is caught by the catch block and executes the code inside the catch block.



## try...catch...finally block

We can also use **finally block** with try and catch block. The **finally block** is always **executed** whether there is an exception or not.

```
try
{
    // code that may raise an exception
}
catch (ExceptionType e)
{
    // code that handles the exception
}
finally
{
    // this code is always executed
}
```



We can see in the above image that the **finally block** is executed in both cases.

The **finally block** is executed:

- **after try and catch block** - when exception has occurred
- **after try block** - when exception doesn't occur

**Example 3:** Handling **NullReferenceException** with try, catch and finally block.

```
using System;
class Calculator
{
    public int sum(int x, int y)
    {
        return x + y;
    }
}
```

```

}
class Program
{
    static void Main(string[] args)
    {
        Calculator obj = null;
        try
        {
            Console.WriteLine(obj.sum(5, 8));
        }
        catch (NullReferenceException e)
        {
            Console.WriteLine("Null Reference");
        }
        finally
        {
            Console.WriteLine("Finally Block Statements");
        }
        Console.ReadKey();
    }
}

```

**Case I** - When exception occurs in try, the catch block is executed followed by the finally block.

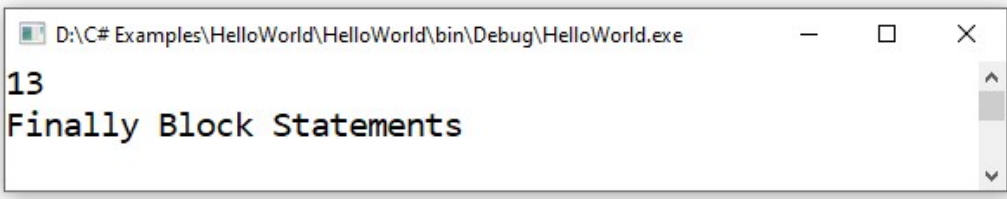


**Case II** - The finally block is directly executed after the try block if an exception doesn't occur. For example, if we create the calculator obj properly as given below.

```

Calculator obj = new Calculator();
try
{
    Console.WriteLine(obj.sum(5, 8));
}
catch (NullReferenceException e)
{
    Console.WriteLine("Null Reference");
}
finally
{
    Console.WriteLine("Finally Block Statements");
}

```



## Catching all Exceptions in C#

**Technique 1:** catch block without brackets or arguments

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        int result = 0;
        try
        {
            result = a / 0;
        }
        catch
        {
            Console.WriteLine("Divide By Zero Error");
        }
        Console.ReadKey();
    }
}
```

**Technique 2:** catch block with brackets and **Exception** class reference parameter.

**Exception class** is a base class of all built in exceptions, which can be used to catch all types of Exceptions in C#.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 10;
        int result = 0;
        try
        {
            result = a / 0;
        }
        catch(Exception ex)
        {
            Console.WriteLine("Divide By Zero Error");
        }
        Console.ReadKey();
    }
}
```

```
    }  
}
```

## Throwing Exceptions in C#

In C#, it is possible to **throw an exception** programmatically. The **'throw' keyword** is used for this purpose. The general form of throwing an exception is as follows.

**throw** exception\_obj;

**Example:** throwing DivideByZeroException

```
using System;  
class Program  
{  
    static void Main(string[] args)  
    {  
        try  
        {  
            throw new DivideByZeroException("Invalid Division");  
        }  
        catch(DivideByZeroException)  
        {  
            Console.WriteLine("Exception");  
        }  
        Console.ReadKey();  
    }  
}
```

## Types of Exceptions

**System.Exception** is the base class for all exceptions in C#. Several exception classes inherit from this class, including

- ApplicationException
- SystemException

## Examples of Built-in Exceptions

- System.DivideByZeroException
- System.IndexOutOfRangeException
- System.ArithmeticException
- System.NullReferenceException
- System.InvalidCastException
- System.OutOfMemoryException
- System.OverflowException



## User-defined Exceptions in C#

In C#, it is possible to create user defined exception classes. But **Exception** must be the ultimate base class for all exceptions in C#. So the **user-defined exception classes must inherit from either the Exception class.**

**Example:**

```
using System;

class MyException : Exception
{
    public MyException(string msg) : base(msg)
    {
    }
}

class Program
{
    static void Main(string[] args)
    {
        try
        {
            throw new MyException("Error Occured!!");
        }
        catch(MyException e)
        {
            Console.WriteLine(e.Message);
        }
        Console.ReadKey();
    }
}
```

**Example 1:** Write a program to get an Age of a person as input and check if it is greater than 18, if not throw an user defined exception called **InvalidAgeException**.

```
using System;

class InvalidAgeException : Exception
{
    public InvalidAgeException(string msg) : base(msg)
    {
    }
}

class Program
{
    static string checkAge(int age)
    {
        if (age < 18)
            throw new InvalidAgeException("Error! Age must be greater than 18");
        else
    }
}
```

```

        return "Yes! You are Eligible";
    }

    static void Main(string[] args)
    {
        Console.Write("Enter your age: ");
        int age = int.Parse(Console.ReadLine());
        try
        {
            Console.WriteLine(checkAge(age));
        }
        catch(InvalidAgeException e)
        {
            Console.WriteLine(e.Message);
        }
        Console.ReadKey();
    }
}

```

**Exercise:** Write a program to check the given Number contains space or not. If the number contains space, throw an user defined exception called “InvalidNumberException”.