Exception handling in C# and .NET framework, suppoted by the try catch and finaly block is a mechanism to detect and handle run-time errors in code. The .NET framework provides built-in classes for common exceptions. The exceptions are anomalies that occur during the execution of a program. They can be because of user, logic or system errors. If a user (programmer) does not provide a mechanism to handle these anomalies, the .NET runtime environment provide a default mechanism, which terminates the program execution.

C# provides three keywords try, catch and finally to implement exception handling. The try encloses the statements that might throw an exception whereas catch handles an exception if one exists. The finally can be used for any cleanup work that needs to be done.

Try..catch..finally block example:

1. try
2. {
3. // Statement which can cause an exception.
4. }
5. catch(Type x)
6. {
7. // Statements for handling the exception
8. }
9. finally
10. {
11. //Any cleanup code
12. }

**Uncaught Exceptions**

The following program will compile but will show an error during execution. The division by zero is a runtime anomaly and program terminates with an error message. Any uncaught exceptions in the current context propagate to a higher context and looks for an appropriate catch block to handle it. If it can't find any suitable catch blocks, the default mechanism of the .NET runtime will terminate the execution of the entire program.

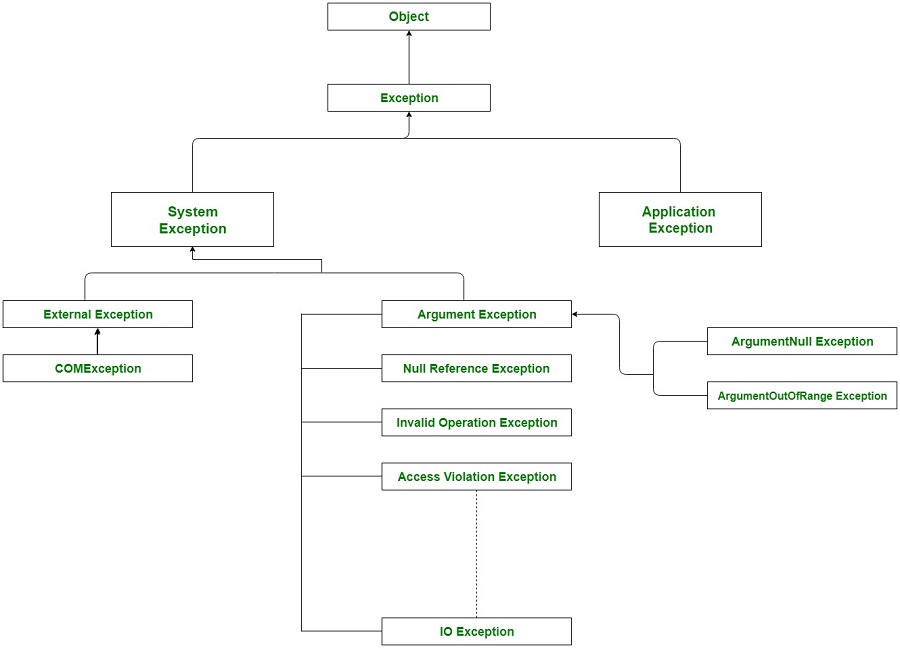
1. //C#: Exception Handling
2. //Author: rajeshvs@msn.com
3. using System;
4. class MyClient
5. {
6. public static void Main()
7. {
8. int x = 0;
9. int div = 100/x;
10. Console.WriteLine(div);
11. }
12. }

The modified form of the above program with exception handling mechanism is as follows. Here we are using the object of the standard exception class DivideByZeroException to handle the exception caused by division by zero.

1. //C#: Exception Handling
2. using System;
3. class MyClient
4. {
5. public static void Main()
6. {
7. int x = 0;
8. int div = 0;
9. try
10. {
11. div = 100 / x;
12. Console.WriteLine("This linein not executed");
13. }
14. catch (DivideByZeroException)
15. {
16. Console.WriteLine("Exception occured");
17. }
18. Console.WriteLine($"Result is {div}");
19. }
20. }

**Exception Hierarchy**

In C#, all the exceptions are derived from the base class **Exception** which gets further divided into two branches as **ApplicationException** and another one is **SystemException**. *SystemException* is a base class for all CLR or program code generated errors. *ApplicationException* is a base class for all application related exceptions. All the exception classes are directly or indirectly derived from the Exception class. In case of *ApplicationException*, the user may create its own exception types and classes. But SystemException contains all the known exception types such as *DivideByZeroException* or *NullReferenceException* etc.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/ExceptionClassHierarchy-1.jpg)

**Different Exception Classes:** There are different kinds of exceptions which can be generated in C# program:

* **Divide By Zero exception**: It occurs when the user attempts to divide by zero
* **Out of Memory exceptions**: It occurs when then the program tries to use excessive memory
* **Index out of bound Exception**: Accessing the array element or index which is not present in it.
* **Stackoverflow Exception**: Mainly caused due to infinite recursion process
* **Null Reference Exception** : Occurs when the user attempts to reference an object which is of NULL type.

**Properties of the Exception Class:** The Exception class has many properties which help the user to get information about the exception during exception.

* **Data:** This property helps to get the information about the arbitrary data which is held by the property in the key-value pairs.
* **TargetSite:** This property helps to get the name of the method where the exception will throw.
* **Message:** This property helps to provide the details about the main cause of the exception occurrence.
* **HelpLink:** This property helps to hold the URL for a particular exception.
* **StackTrace:** This property helps to provide the information about where the error occurred.
* **InnerException:** This property helps to provide the information about the series of exceptions that might have occurred.
* try
* {
* int value = 1 / int.Parse("0");
* }
* catch (Exception ex)
* {
* Console.WriteLine("HelpLink = {0}", ex.HelpLink);
* Console.WriteLine("Message = {0}", ex.Message);
* Console.WriteLine("Source = {0}", ex.Source);
* Console.WriteLine("StackTrace = {0}", ex.StackTrace);
* Console.WriteLine("TargetSite = {0}", ex.TargetSite);
* }

Nested try-catch

static void Main(string[] args)

{

Student std = null;

try

{

try

{

// following throws NullReferenceException

std.StudentName = "";

}

catch (InvalidOperationException innerEx)

{

Console.WriteLine("Inner catch");

}

}

catch

{

Console.WriteLine("Outer catch");

}

}

Nested try-catch

static void Main(string[] args)

{

Student std = null;

try

{

try

{

std.StudentName = "";

}

catch

{

Console.WriteLine("Inner catch");

}

}

catch

{

Console.WriteLine("Outer catch");

}

}

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();

/\*

\* C# Program to Demonstrate IndexOutOfRange Exception

\*/

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace differnce

{

class arrayoutofindex

{

public void calculatedifference()

{

int difference=0;

int [] number= new int[5] {1,2,3,4,5};

try

{

for (int init =1; init <=5; init++)

{

difference= difference - number[init];

}

Console.WriteLine("The difference of the array is:" + difference);

}

catch (IndexOutOfRangeException e)

{

Console.WriteLine(e.Message);

}

}

}

class classmain

{

static void Main(string [] args)

{

arrayoutofindex obj = new arrayoutofindex();

obj.calculatedifference();

Console.ReadLine();

}

}

}

1. Use the try, catch and finally blocks to handle exceptions in C#.
2. The try block must be followed by a *catch* or *finally* block or both.
3. A multiple catch block is allowed with different exception filters. General catch{..} block must come last.
4. catch{..} and catch(Exception ex){ } both cannot be used.
5. The finally block must come after the try or catch block.
6. The finally block will always execute irrespective of whether an exception occured or not.
7. The finally block is appropriate place for disposing objects.
8. The finally block cannot have a return or break because it isn't allow to leave the control.
9. Nested try-catch blocks are allowed in C#.
10. An Exception will be catched in the inner catch block if appropriate filter found, otherwise will be catched by outer catch block.