C# Exception Handling

Exception Handling in C# is *a process to handle runtime errors*. We perform exception handling so that normal flow of the application can be maintained even after runtime errors.

In C#, exception is an event or object which is thrown at runtime. All exceptions the derived from *System.Exception* class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

Advantage

It *maintains the normal flow* of the application. In such case, rest of the code is executed event after exception.

C# Exception Classes

All the exception classes in C# are derived from **System.Exception** class. Let's see the list of C# common exception classes.

Exception	Description
System.DivideByZeroException	handles the error generated by dividing a number with zero.
System.NullReferenceException	handles the error generated by referencing the null object.
System.InvalidCastException	handles the error generated by invalid typecasting.
System.IO.IOException	handles the Input Output errors.
System.FieldAccessException	handles the error generated by invalid private or protected field access.

C# Exception Handling Keywords

In C#, we use 4 keywords to perform exception handling:

- try
- catch
- finally, and
- throw

C# try/catch

In C# programming, exception handling is performed by try/catch statement. The **try block** in C# is used to place the code that may throw exception. The **catch block** is used to handled the exception. The catch block must be preceded by try block.

C# example without try/catch

```
using System;
public class ExExample
{
   public static void Main(string[] args)
   {
      int a = 10;
      int b = 0;
      int x = a/b;
      Console.WriteLine("Rest of the code");
   }
}
```

Output:

```
Unhandled Exception: System.DivideByZeroException: Attempted to divide by zero.
```

C# try/catch example

```
using System;
public class ExExample
{
   public static void Main(string[] args)
   {
      try
      {
         int a = 10;
         int b = 0;
         int x = a / b;
      }
      catch (Exception e) { Console.WriteLine(e); }

      Console.WriteLine("Rest of the code");
   }
}
```

Output:

```
System.DivideByZeroException: Attempted to divide by zero.
Rest of the code
```

C# finally

C# finally block is used to execute important code which is to be executed whether exception is handled or not. It must be preceded by catch or try block.

C# finally example if exception is handled

```
using System;
public class ExExample
{
   public static void Main(string[] args)
   {
      try
      {
         int a = 10;
         int b = 0;
         int x = a / b;
      }
      catch (Exception e) { Console.WriteLine(e); }
      finally { Console.WriteLine("Finally block is executed"); }
      Console.WriteLine("Rest of the code");
   }
}
```

Output:

```
System.DivideByZeroException: Attempted to divide by zero.
Finally block is executed
Rest of the code
```

C# finally example if exception is not handled

```
using System;
public class ExExample
{
   public static void Main(string[] args)
   {
      try
      {
         int a = 10;
         int b = 0;
         int x = a / b;
      }
      catch (NullReferenceException e) { Console.WriteLine(e); }
      finally { Console.WriteLine("Finally block is executed"); }
      Console.WriteLine("Rest of the code");
   }
}
```

Output:

```
Unhandled Exception: System.DivideBy
```

C# User-Defined Exceptions

C# allows us to create user-defined or custom exception. It is used to make the meaningful exception. To do this, we need to inherit Exception class.

C# user-defined exception example

```
using System;
public class InvalidAgeException : Exception
{
   public InvalidAgeException(String message)
      : base(message)
   {


   }
}
public class TestUserDefinedException
{
   static void validate(int age)
   {
      if (age < 18)
      {
         throw new InvalidAgeException("Sorry, Age must be greater than 18");
      }
   }
   public static void Main(string[] args)
   {
      try
      {
         validate(12);
      }
      catch (InvalidAgeException e) { Console.WriteLine(e); }
      Console.WriteLine("Rest of the code");
   }
}
```

Output:

```
InvalidAgeException: Sorry, Age must be greater than 18
Rest of the code
```

C# SystemException class

The SystemException is a predefined exception class in C#. It is used to handle system related exceptions. It works as base class for system exception namespace. It has various child classes like:

ValidationException, ArgumentException, ArithmeticException, DataException, StackOverflowException etc.

It consists of rich constructors, properties and methods that we have tabled below.

C# SystemException Signature

```
[SerializableAttribute]
[ComVisibleAttribute(true)]
public class SystemException : Exception
```

C# SystemException Constructors

Constructors	Description
SystemException()	It is used to initialize a new instance of the SystemException class.
SystemException(SerializationInfo,StreamingContext)	It is used to initialize a new instance of the SystemException class with serialized data.
SystemException(String)	It is used to initialize a new instance of the SystemException class with a specified error message.
SystemException(String,Exception)	It is used to initialize a new instance of the SystemException class with a specified error message and a reference to the inner exception that is the cause of this exception.

C# SystemException Properties

Property	Description
Data	It is used to get a collection of key/value pairs that provide additional user-defined information about the exception.
HelpLink	It is used to get or set a link to the help file associated with this exception.
HResult	It is used to get or set HRESULT, a coded numerical value that is assigned to a specific exception.
InnerException	It is used to get the Exception instance that caused the current exception.
Message	It is used to get a message that describes the current exception.
Source	It is used to get or set the name of the application that causes the error.

StackTrace	It is used to get a string representation of the immediate frames on the call stack.
TargetSite	It is used to get the method that throws the current exception.

C# SystemException Methods

Method	Description
Equals(Object)	It is used to check that the specified object is equal to the current object or not.
Finalize()	It is used to free resources and perform cleanup operations.
GetBaseException()	It is used to get root exception.
GetHashCode()	It is used to get hash code.
GetObjectData(SerializationInfo,StreamingContext)	It is used to get object data.
GetType()	It is used to get the runtime type of the current instance.
MemberwiseClone()	It is used to create a shallow copy of the current Object.
ToString()	It is used to create and return a string representation of the current exception.

C# SystemException Example

This class can be used to handle exception of subclasses. Here, in the following program, program throws an IndexOutOfRangeException that is subclass of SystemException class.

```
using System;
namespace CSharpProgram
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            int[] arr = new int[5];
```

```
            arr[10] = 25;
        }
        catch (SystemException e)
        {
            Console.WriteLine(e);
        }
    }
  }
}
```

Output:

System.IndexOutOfRangeException: Index was outside the bounds of the array.