User Defined Exceptions

.NET provides a hierarchy of exception classes ultimately derived from the base class Exception.

However, if none of the predefined exceptions meets your needs, you can create your own exception classes by deriving from the Exception class.

When creating your own exceptions, end the class name of the user-defined exception with the word "Exception," and implement the three common constructors, as shown in the following example.

The example defines a new exception class named AgeInvalidException.

The class is derived from Exception and includes three constructors.

```
using System;
public class AgeInvalidException : Exception
{
   public AgeInvalidException()
   {
   }
   public AgeInvalidException(string message): base(message)
   {
   }
   public AgeInvalidException(string message, Exception inner): base(message, inner)
   {
   }
}
```

Application :

Below application demonstrates the use of user defined exception.

In this application we accept value from user and if entered value is negative then we throw our user defined exception from try block using throw keyword.

```
using System;
public class InvalidNumberException : Exception
{
   public InvalidNumberException(String message): base(message)
   {
      Console.WriteLine("UserDefined Exception {0}",message);
```

```
    }
}
    class Marvellous
    {
        static void Accept(int no)
        {
            if (no < 0)
            {
                throw new InvalidNumberException("Input should be positive");
            }
        }

        static void Main(string[] args)
        {
            try
            {
                Console.WriteLine("Enter number");
                int no = Convert.ToInt32(Console.ReadLine());

                Accept(no);
            }

            catch (InvalidNumberException e)
            {
                Console.WriteLine(e);
            }

            Console.WriteLine("Successful termination of application");
        }
    }
```

Checked Unchecked in C#

C# statements can execute in either checked or unchecked context.

In a checked context, arithmetic overflow raises an exception.

In an unchecked context, arithmetic overflow is ignored and the result is truncated by discarding any high-order bits that don't fit in the destination type.

- checked Specify checked context.
- unchecked Specify unchecked context.

The following operations are affected by the overflow checking:

- Expressions using the following predefined operators on integral types: ++, --, unary -, +, -, *, /

- Explicit numeric conversions between integral types, or from float or double to an integral type.

If neither checked nor unchecked is specified, the default context for non-constant expressions (expressions that are evaluated at run time) is defined by the value of the -checked compiler option.

By default the value of that option is unset and arithmetic operations are executed in an unchecked context.

For constant expressions (expressions that can be fully evaluated at compile time), the default context is always checked.

Unless a constant expression is explicitly placed in an unchecked context, overflows that occur during the compile-time evaluation of the expression cause compile-time errors.

Application:

Application to demonstrates the use of checked and unchecked keyword.

In below application we pass largest value that we can store in integer and inside the function we increase the value to check the use of checked and unchecked keyword.

```
using System;

   class Marvellous
   {
      public static void Demo(int no)
      {
```

```
        Console.WriteLine(no + 10);
    }

    public static void CheckedDemo(int no)
    {
        checked
        {
            Console.WriteLine(no + 10);
        }
    }

    public static void UncheckedDemo(int no)
    {
        unchecked
        {
            Console.WriteLine(no + 10);
        }
    }

    static void Main(string[] args)
    {
        Demo(int.MaxValue);
        CheckedDemo(int.MaxValue);
        UncheckedDemo(int.MaxValue);
    }
}
```

For better understanding call only one function at a time.