Ex. No. 5	**Exception Handling**		
Date of Exercise	17.08.2016	Date of Upload	21.10.2016

Aim

To develop **Integer Stack Application** using C# by handling various catching various exceptions relating to size and format specification thereby handling them.

Description

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.

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:

```
try

{// statements causing exception}

catch( ExceptionName e1 )

{// error handling code}

catch( ExceptionName e2 )

{// error handling code}
```

```
catch( ExceptionName eN )

{// error handling code}

finally

{// statements to be executed}
```

Exception Classes in C#

C# exceptions are represented by classes. The exception classes in C# are mainly directly or indirectly derived from the **"System.Exception"** class. Some of the exception classes derived from the System.Exception class are the **"System.ApplicationException"** and **"System. SystemException"** classes. The "**System.ApplicationException**" class supports exceptions generated by application programs. Hence the exceptions defined by the programmers should derive from this class. The "**System.SystemException"** class is the base class for all predefined system exception.

The following table provides some of the predefined exception classes derived from the **Sytem.SystemException class**:

Exception Class	**Description**
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.

System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

User - Defined Exception Classes

There are two types of exceptions:

- exceptions generated by an executing program
- exceptions generated by the CLR

The CLR throws **SystemException.**

The **ApplicationException** is thrown by a user program rather than the runtime where it is possible to create our own exception class. Exception must be the ultimate base class for all exceptions in C#. User-defined exception classes must inherit from either Exception class or one of its standard derived classes

Example

```
using System;
class MyException : ApplicationException
{ public MyException(string str) //constructor
{ Console.WriteLine ("User Defined Exception"); }
}
class MyClient
{
public static void Main()
{
try
{
throw new MyException ("Rajesh");
}
catch(MyException e)

{

Console.WriteLine ("Exception caught here" + e.ToString ());

}

Console.WriteLine("Last Statement");

}

}
```

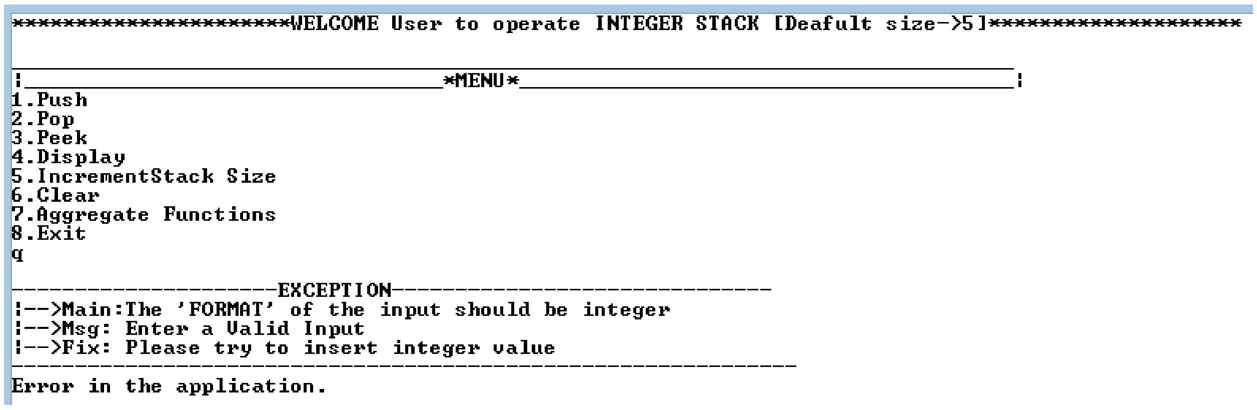
Program

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

namespace Integer_Stack_Exception_Handling_5
{
    //To raise an Exception if the input in non-integer
    class NonInteger_Exception:ApplicationException {

        public NonInteger_Exception(String msg) {
            Console.WriteLine("\n---------------------EXCEPTION--------------------------
----");
            Console.WriteLine("|-->Main:The \'FORMAT\' of the input should be integer\n|-
->Msg: {0}",msg);
            Console.WriteLine("|-->Fix: Please try to insert integer value");
            Console.WriteLine("----------------------------------------------------------
----");
        }
    }
    //To raise an Exception if the input if Stack is empty
    class StackEmpty_Exception : ApplicationException {
        public StackEmpty_Exception(String msg)
        {
            Console.WriteLine("\n---------------------EXCEPTION--------------------------
----");
            Console.WriteLine("|-->Main:For \'POP\' or \'DISPLAY\' the \'SIZE\' of the
stack should be > 0\n|-->Msg: {0}", msg);
            Console.WriteLine("|-->Fix: Please try to \'PUSH\' values");
            Console.WriteLine("----------------------------------------------------------
----");
        }
    }

    class Program
    {
        //initialize the size of stack(default is 5)
        static int Stack_Size = 5;
        //Main Control Method
        static void Main(string[] args)
        {
            //Design
            Console.WriteLine("*********************WELCOME User to operate INTEGER
STACK [Deafult size->5]********************");
            Console.WriteLine();
            //Initialising Starting default capacity to 5
            Stack<int> stackinteger = new Stack<int>(5);
            //Calling Menu Function
            Initialize_Menu(stackinteger);
        }
        //To Display Menu
        public static void Initialize_Menu(Stack<int> stackinteger)
        {

Console.WriteLine("_________________________________________________________________________________________
_________");

Console.WriteLine("|_________________________________*MENU*______________________________
_________|");
            Console.WriteLine("1.Push\n2.Pop\n3.Peek\n4.Display\n5.IncrementStack
Size\n6.Clear\n7.Aggregate Functions\n8.Exit");

            try
            {
                string value = Console.ReadLine();
                int choice;
                //to check if the input is integer
                if (int.TryParse(value, out choice))
                {

                switch (choice)
                {
                    case 1:
                        Push_Integer(stackinteger);
                        break;
                    case 2:
                        Pop_Integer(stackinteger);
                        break;
                    case 3:
                        Peek_Integer(stackinteger);
                        break;
                    case 4:
                        Display_Integer(stackinteger);
                        break;
                    case 5:
                        Increment_Stack_Size(stackinteger);
                        break;
                    case 6:
                        Clear_Integer(stackinteger);
                        break;

                    case 8:
                        Console.WriteLine("The program is terminated");
                        Environment.Exit(0);
                        break;

                    default:
                        Console.WriteLine("Invalid Choice!\nPlease Try again :(");
                        Initialize_Menu(stackinteger);
                        break;
                    }

                }
                else
                {
                    throw new NonInteger_Exception("Enter a Valid Input");
                }
            }
            catch (NonInteger_Exception formatexception)
            {
                Console.WriteLine(formatexception.Message);
            }
            finally
            {
                Initialize_Menu(stackinteger);
            }
        }
        //To Increment Stack Size
        public static void Increment_Stack_Size(Stack<int> stackinteger)
        {
            Console.WriteLine("Current Size is {0}", Stack_Size);
            Console.WriteLine("New Size should be greater than the Current Size");
            Console.WriteLine("Enter the new Stack Size ?");
            try
            {
                string value = Console.ReadLine();
                int intvalue;
                //to check if the input is integer
                if (int.TryParse(value, out intvalue))
                {
                    //Exception if the user tries to decrement the stack size
                    if (intvalue <= Stack_Size) { throw new
InvalidOperationException("Lesser Size or equal Size is entered"); }
                    else { Stack_Size = intvalue; }
                }
                else
                {
                    throw new NonInteger_Exception("Not an Integer");
                }
            }
            catch (NonInteger_Exception formatexception)
            {
                Console.WriteLine(formatexception.Message);
            }
            catch (InvalidOperationException invalidsize) {
                Console.WriteLine("\n----------------------EXCEPTION----------------------
--------");
                Console.WriteLine("|-->Main:The \'SIZE\' of the stack entered should be
greater than {0}\n|-->Msg: {1}", Stack_Size, invalidsize.Message);
```

```
                Console.WriteLine("|-->Fix: Please Enter greater size to increment");
                Console.WriteLine("-------------------------------------------------------
--------");
            }
            finally
            {
                Initialize_Menu(stackinteger);
            }
        }
        //To Push an Integer
        public static void Push_Integer(Stack<int> stackinteger) {
            try
            {
                //To check if it reached the max size limit
                if (stackinteger.Count == Stack_Size)
                {
                    throw new StackOverflowException();
                }
                else
                {
                    Console.WriteLine("Enter the integer value ?");
                    //reading the value
                    string value = Console.ReadLine();
                    int intvalue;
                    //to check if the input is integer
                    if (int.TryParse(value, out intvalue))
                    {
                        stackinteger.Push(intvalue); Display_Integer(stackinteger);
                    }
                    else
                    {
                        throw new NonInteger_Exception("Not an Integer");
                    }
                }
            }
            catch (StackOverflowException sizeexception)
            {
                Console.WriteLine("\n---------------------EXCEPTION----------------------
--------");
                Console.WriteLine("|-->Main:The \'SIZE\' of the stack cannot be greater
than {0}\n|-->Msg: {1}", Stack_Size, sizeexception.Message);
                Console.WriteLine("|-->Fix: Please POP the elements");
                Console.WriteLine("-------------------------------------------------------
--------");
            }
            catch (NonInteger_Exception formatexception) {
                Console.WriteLine(formatexception.Message);
            }
            finally
            {
                Initialize_Menu(stackinteger);
            }

        }
        //To Pop an the top element
        public static void Pop_Integer(Stack<int> stackinteger) {
            try
            {
                //if the stack has no elements
                if (stackinteger.Count == 0)
                {
                    throw new StackEmpty_Exception("Stack is empty");
                }
                else {
                    Console.WriteLine("The Popped Value is : " + stackinteger.Pop());
                }
            }
            catch(StackEmpty_Exception stackempty) {
                Console.WriteLine(stackempty.Message);
            }
            finally
            {
                Initialize_Menu(stackinteger);
            }
        }
        //To Peek the top element
        public static void Peek_Integer(Stack<int> stackinteger)
        {
            Console.WriteLine("The Peeked Value is : " + stackinteger.Peek());
            Initialize_Menu(stackinteger);
        }
        //To Display the elements in a Stack
        public static void Display_Integer(Stack<int> stackinteger)
        {
            try
            {
                //if the stack has no elements
                if (stackinteger.Count == 0)
                {
                    throw new StackEmpty_Exception("Stack is empty");
                }
                else
                {
                    Console.WriteLine("The stack elements are : ");
                    Console.Write("[ ");
                    foreach (int val in stackinteger)
                    {
                        Console.Write(val + ", ");
                    }
                    Console.WriteLine("]");
                }
            }
            catch (StackEmpty_Exception stackempty)
            {
                Console.WriteLine(stackempty.Message);
            }
            finally
            {
                Initialize_Menu(stackinteger);
            }
        }
        //To Clear the Stack elements
        public static void Clear_Integer(Stack<int> stackinteger) {
            Console.WriteLine("After clearing the Stack :");
            stackinteger.Clear();
            Display_Integer(stackinteger);
        }

    }

}
```

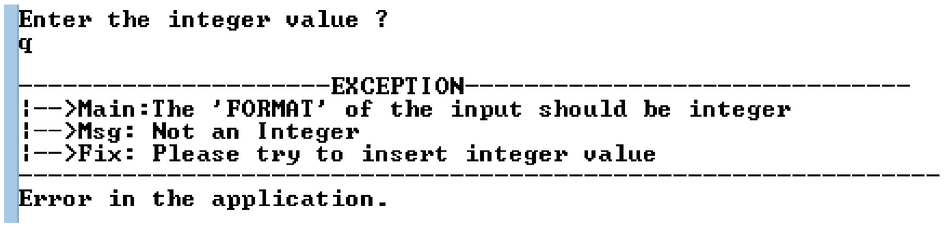
Output

Invalid Input



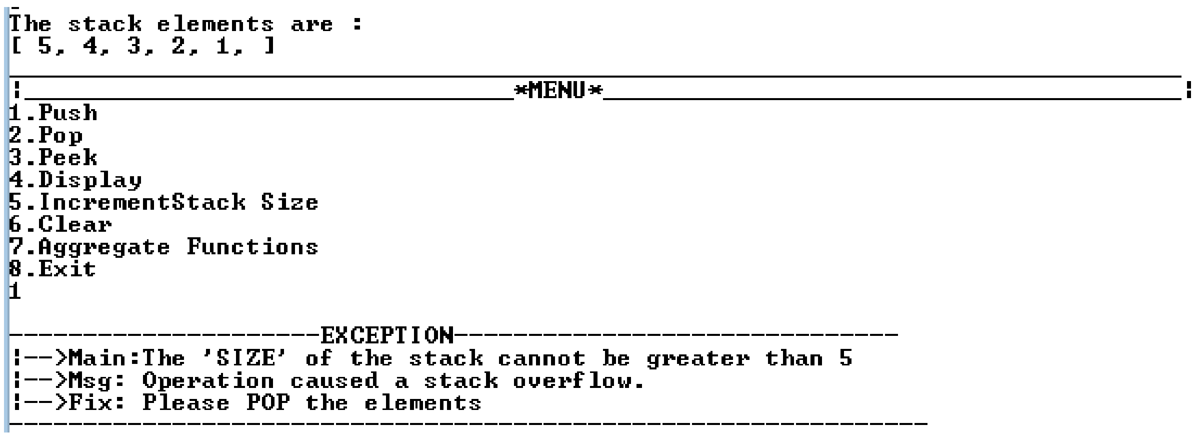
```
**********************WELCOME User to operate INTEGER STACK [Deafult size->5]********************

 ___________________________________________________________________________________
|__________________________________*MENU*__________________________________________|
1.Push
2.Pop
3.Peek
4.Display
5.IncrementStack Size
6.Clear
7.Aggregate Functions
8.Exit
q

---------------------EXCEPTION------------------------------
|-->Main:The 'FORMAT' of the input should be integer
|-->Msg: Enter a Valid Input
|-->Fix: Please try to insert integer value
--------------------------------------------------------------
Error in the application.
```

Non-Integer Value



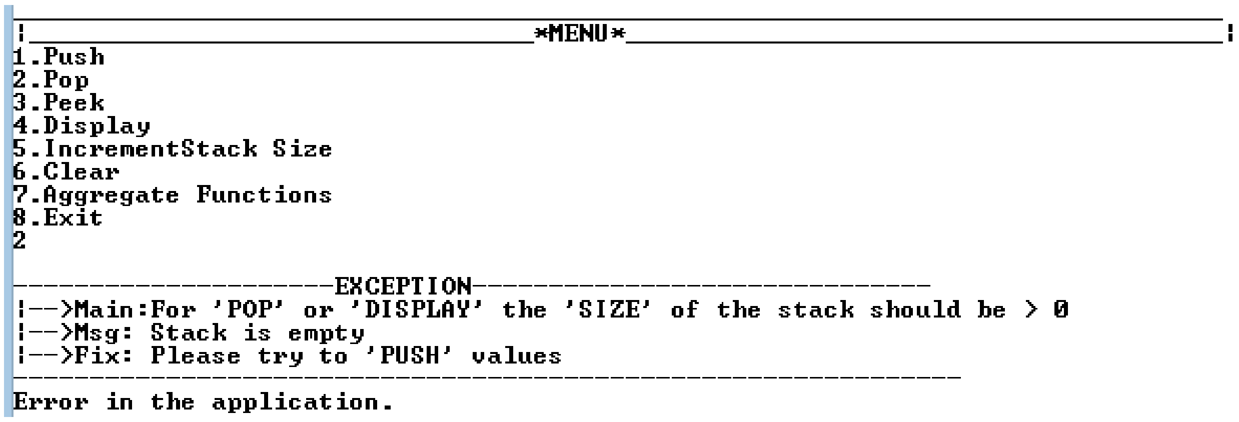
```
Enter the integer value ?
q

---------------------EXCEPTION------------------------------
|-->Main:The 'FORMAT' of the input should be integer
|-->Msg: Not an Integer
|-->Fix: Please try to insert integer value
--------------------------------------------------------------
Error in the application.
```

Max Stack Size limit



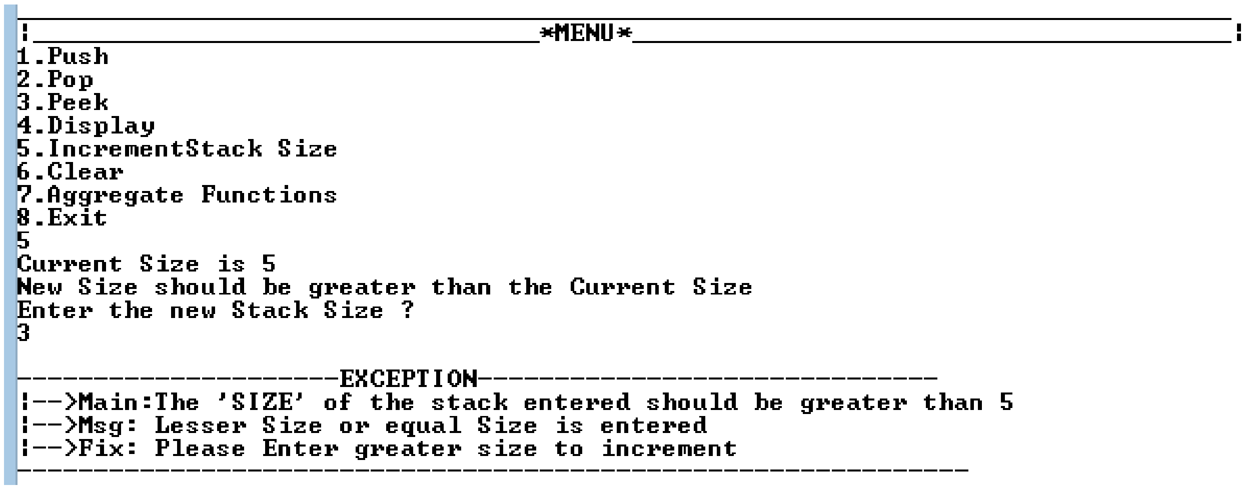
```
The stack elements are :
[ 5, 4, 3, 2, 1, ]
 ___________________________________________________________________________________
|__________________________________*MENU*__________________________________________|
1.Push
2.Pop
3.Peek
4.Display
5.IncrementStack Size
6.Clear
7.Aggregate Functions
8.Exit
1

---------------------EXCEPTION------------------------------
|-->Main:The 'SIZE' of the stack cannot be greater than 5
|-->Msg: Operation caused a stack overflow.
|-->Fix: Please POP the elements
--------------------------------------------------------------
```

Stack Empty



```
|__________________________________*MENU*_______________________________________|
1.Push
2.Pop
3.Peek
4.Display
5.IncrementStack Size
6.Clear
7.Aggregate Functions
8.Exit
2

---------------------EXCEPTION------------------------------
|-->Main:For 'POP' or 'DISPLAY' the 'SIZE' of the stack should be > 0
|-->Msg: Stack is empty
|-->Fix: Please try to 'PUSH' values
--------------------------------------------------------------
Error in the application.
```

Invalid Operation



```
|__________________________________*MENU*_______________________________________|
1.Push
2.Pop
3.Peek
4.Display
5.IncrementStack Size
6.Clear
7.Aggregate Functions
8.Exit
5
Current Size is 5
New Size should be greater than the Current Size
Enter the new Stack Size ?
3

---------------------EXCEPTION------------------------------
|-->Main:The 'SIZE' of the stack entered should be greater than 5
|-->Msg: Lesser Size or equal Size is entered
|-->Fix: Please Enter greater size to increment
--------------------------------------------------------------
```

Result

The above programmed is compiled successfully and the screenshots are well described with successful outputs and constraints.

[**Dr. S.P. Jeno Lovesum**]