**20. Enumerations**

* Enumeration is a user defines data type and used to create named constants.
* By using an enumeration we can group constants that are logically related t each other.
* Enum are used when we know all possible values of variable at compile time.
* By using Enum we can create set of constants and we can assign the meaningful name to constant value.
* If we want to restrict the choices we can do it using the Enum.
* Syntax:

<Access modifier> enum <enum name>

{

//enumeration list

}

* We used enum keyword to define an enumeration.
* The enumeration list’s identifiers can be assigned different values.
* If values are not specified the default values are taken as 0, 1, 2, …
* We can access enum items using dot syntax.
* An enumerataion can contain only integral data type items such as byte, sbyte, short, ushort, int, uint, long or ulong.
* It’s not possible to use an enum with string or any other data type to define enum elements .
* type of enumeration element is int.
* In case if we want to change type of enum to any other integral data type, then we need to mention a data type with a colon (:) .
* To get the values of enum elements, an explicit cast is necessary to convert from enum to int.

**21. Handling Exceptions**

* An exception is an event which occurs during the execution of a program that disrupts the normal flow of the program.
* An exception is a C# object that represents an error.
* When a C# raises an exception it must either handle the exception immediately otherwise it terminate the program and quit.
* The exception handling is one of the powerful mechanism to handle a runtime errors so that normal flow of the application can be maintained.
* To handle runtime or unexpected errors in application C# has provided a built-in exception handling support by using try, Catch and finally blocks.
* In c# when exception is thrown , then the CLR will look for the catch block In a program that handles the exception. In case, if the currently executing method does not contain such a catch block, then CLR will displayed an unhandled exception message to the user and stops the execution of the program.
* The try block will contain the code that may cause an exception so that if any error occurred in our code, then immediately the code execution will be moved to catch block to handle those exceptions.
* In case if no exception occurred in the try block then the catch block will be skipped.
* Syntax:

Try

{

//code that may cause exception

}

Catch (Exception e)

{

//exception handling

}

Finally

{

//code

}

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

**Throw**

* Now this is the handling of exception which are automatically raised by CLR We can alos raise an exception manually.
* We can raise exception manually by using Throw keyword.
* Any type of Exceptions which is derived from Exception class can be raised using the throe keyword.
* Syntax:

throw e

**Types of Exception**

* .NET includes build-in exception classes for every possible error. The Exception class is the base class of all the exception classes.
* The exception class is the base class of the SystemException and ApplicationException classes. The SystemException class is the base class for all exception that can occur during the execution of the program
* Here is the list of some important built-in exception classes in .net.

**DivideByZeroException**

* The Exception that is thrown when there is an attempt to divide an integral or decimal value by zero.
* To prevent the exception , ensure that the denominator in a division operation with integer or decimal values is non-zero.

**IndexOutOfRangeException**

* The exception that is thrown when we accessing the array or collections elements with an index that is outside of it’s bound.

**NullReferenceException**

* The exception that is thrown when we try to access member on a type whose value is null.

**FormatException**

* The exception occurs when a value is not in a appropriate format to be converted from a string by s conversion method such as Parse.

**InvalidCastException**

* Handles the error genrrated by invalid typecasting.

**FileNotFoundException**

* The exception that is thrown when a physical file does not exist at the specified location

**OutOfMemoryException**

* The exception that is thrown when program does not get enough memory to execute the code.

**22. Events**

* Event enable a class or object to notify other classes or objects when something of interest occurs.
* The class that sends(Raises) the event is called publisher and the classes that receives (Handle) the event are called subscriber.
* The publisher determines when an event is raised, The subscriber determines what action is taken in response to the event.
* An event can have a multiple subscribers. Subscriber can handle multiple events from multiple publishers.
* In C# events are encapsulated delegates so first, we need to declare delegate before we declare an event inside of a class by using event keyword.
* Event that have no subscriber are never raised.
* An event can be declared in two steps.

1. Declare a delegate
2. Declare the event using event keyword

* In C# to raise an event we need to invoke the evnt delegate and subscribe to the event using +=operator. In case , if you want to unsubscribe from an event, then use -= operator.

The event will enable a class or object to notify other class or object when something special happens.

**23. Basic File Operations**

* Generally, the file is used to store the data.
* The term file handling refers to the various operations like creating file, Reading from the file, appending the file, etc…
* There are two basic operations which is mostly used in file is reading and writing.
* The file becomes stream when we open file for writing and reading.
* Two streams can be formed from file one is input stream which is used to read the file and another is output stream is used to write in the file.
* There is System.IO namespace contains classes which handle input and output streams and provide information about file and directory structure.
* Here is some classes and description of these classes

.

|  |  |
| --- | --- |
| **Class Type** | **Description** |
| FileStream | Is used to read from and write within a file. |
| StreamReader | Is used to read characters from abyte stream. |
| StreamWriter | Is used to write characters to stream. |
| BinaryReader | Reads primitive data from binary stream. |
| BinaryWriter | Writes primitive data in binary format |
| DirectoryInfo | Used to perform operation on directories. |
| FileInfo | Use to perform operations on file. |

**FileStream Class**

* We need to create a FileStream object to open existing file or create new file .
* Syntax:

FileStream <object\_name> = new FileSream(<File\_name> <FileMode><FileShare>)

* FileMode enum have create, createnew ,open, openorcreate, append, Truncate.
* FileAccess enum have read write and readwritw.
* FileShare enum have Inheritable, none, read, write and readwrite members.

**StreamReader and StreamWriter**

* We can the StreamReader and StreamWriter class to read and write data in files.
* We can create an object of StreamWriter and StreamReader by paasing the filename to their constructor.

**StreamReader and StreamWriter**

* We can the StreamReader and StreamWriter class to read and write data in files.
* We can create an object of StreamWriter and StreamReader by paasing the filename to their constructor.
* There is some methods in StreamWriters class such as close, write, writwLine.
* And also some methods in STreamREader class such as close, Read, ReadLine and ReadToEnd.

**BinaryReader and BinaryWriter**

* The BinaryReader and BinaryWriter classes read and write data in binary format, rather than text.
* We can use the Write() method of the BinaryWriter class to write binary data in to a file.
* To read a binary data , we can use Read() method of BinaryReader class.

**FileInfo Class**

* This class provides method to create, copy, delete, move and open files.
* There is somemethods of FileInfo class such as CopyTo, Create, MoveTo, Delete and properties like Length, Nmae, CreationTime, LastAccessTime, LastWriteTime etc.

**DirectoryInfo Class**

* The DirectoryInfo class shares almost all of the same properties as the FileInfo class, except that they operate on directories not files.
* There is some methods of DirectoryInfo class such as Create, CreateSubdirectory, MoveTo, Delete, GetDirectories, GetFiles etc. and properties like CreationTime, Fullname, Name, LastAccessTime, LastWriteTime, Root etc.

**24. Interface & Inheritance**

**(a)Interface**

* An interface is contract between itself and any class that implements it.
* This contract states that any class that implements the interface will implement the interface’s properties, methods, and events.
* All the method which are declared inside the interface are abstract methods, It cannot have method body and can not be instantiated.
* It is used to achieve multiple inheritance which cannot be achieved by class.
* It is used to achieve fully abstraction because it have not method body.
* Interface cannot have variables(field) and also we cannot make the object of the interface.
* All members of the interface are public, they don’t allow explicit access modifier.
* We can declare it by interface keyword
* Syntax:

interface <Interface name>

{

Body//

}

* Syntax of implementing interface:

Class <class name> : <Interface name>

* A class has to explicitly implement multiple inheritance if these interfaces hav methods with same name and signature.

**(b)Inheritance**

* The mechanism of deriving a new class from an old one is known as inheritance.
* The old class is referred to as base class and new one is called the derived class or subclass
* A derived class extends its base class. That is, it contains the methods and data of it’s parent class, and it can also contain its own data members and methods.
* He derived class can not change the definition of an inherited member.
* A derived class can override inherited members.
* If we want to declare a class derives from another class , the syntax is:
* Class Derived class : Base class

{

//body

}

**Type of Inheritance**

1. **Single inheritance**: A derived class with only one base class, is called single inheritance.
2. **Multilevel inheritance:** The mechanism of deriving a class from another derived class is called multilevel inheritance.
3. **Hierarchical inheritance**: one class may be inherited by more than one classes , this process known as hierarchical inheritance.
4. **Hybrid inheritance:** There could be the situation where we need to apply one or more inheritance to design the program, so applying more then one inheritance is known as hybrid inheritance**.**
5. **Multiple inheritance:** In multiple inheritance, one class can have more than one super class and inherit features from all parent classes. We can achieve multiple inheritance only through interfaces.

**PROGRAMS**

1. **Enumeration**

using System;

namespace EnumDemo

{

class Program

{

public enum WeakDays

{

sunday, monday, tuesday, wednesday, thursday, friday, saturday

}

static void Main(string[] args)

{

// Accessing value by explicit conversion

Console.WriteLine("value at friday is " + (int)WeakDays.friday);

// Geting the name of enum

Console.WriteLine("Name of the weak is " + WeakDays.friday);

// Getting names using GetNames method

string[] values = Enum.GetNames(typeof(WeakDays));

foreach (var item in values)

{

Console.WriteLine(item);

}

// Getting values using GetValue method

int[] num = (int [])Enum.GetValues(typeof(WeakDays));

foreach (var item in num)

{

Console.WriteLine(item);

}

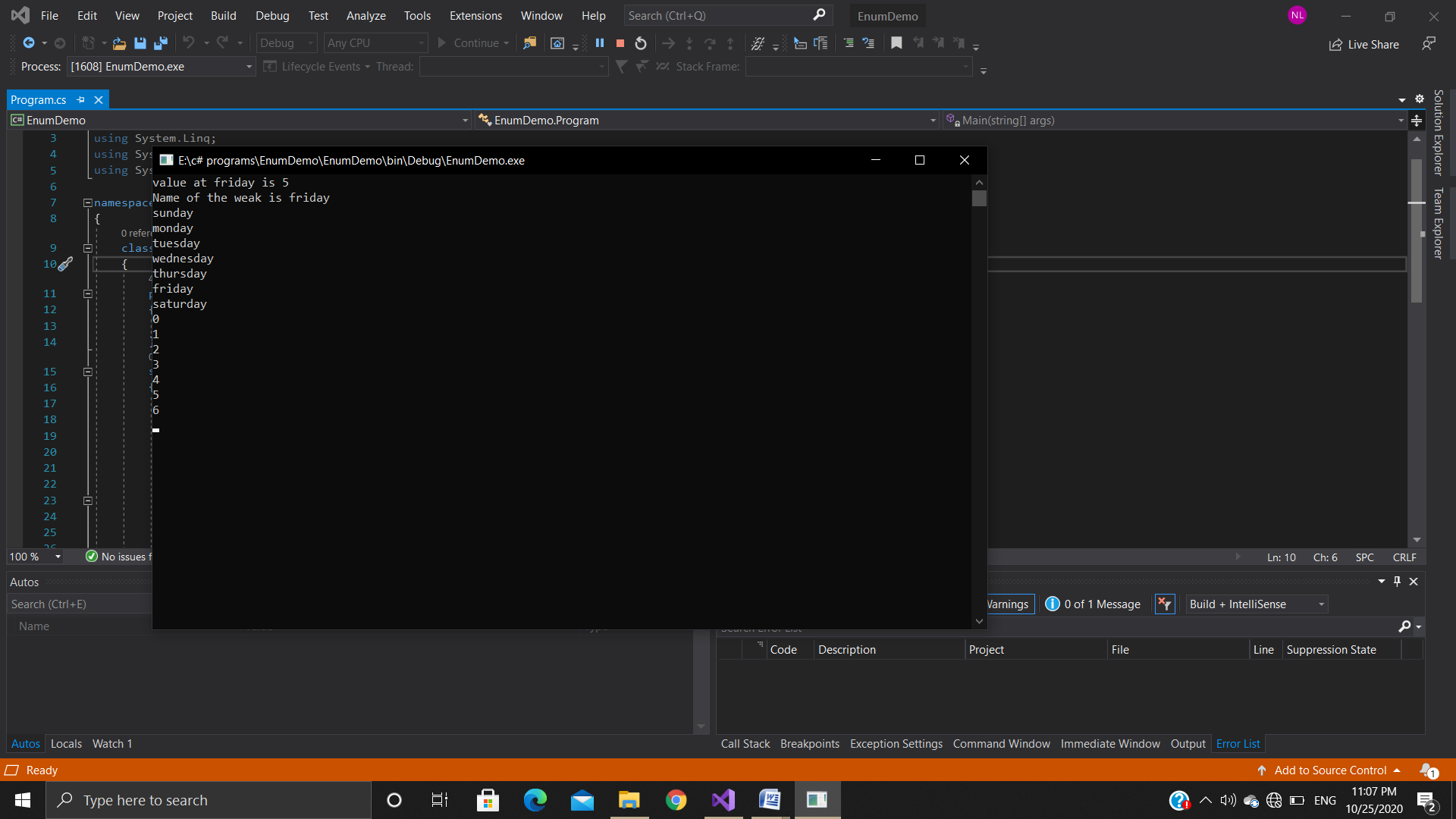
Console.ReadLine();

}

}

}

**Output**

****

1. **Interface**

using System;

namespace InterfaceDemo

{

interface i1

{

void show();

}

interface i2

{

void show();

}

class Program: i1 ,i2

{

void i1.show()

{

Console.WriteLine("This is the interface of i1");

}

void i2.show()

{

Console.WriteLine("This is the interface of i2");

}

static void Main(string[] args)

{

Program obj1 = new Program();

((i1)obj1).show();

i2 obj2 = new Program();

obj2.show();

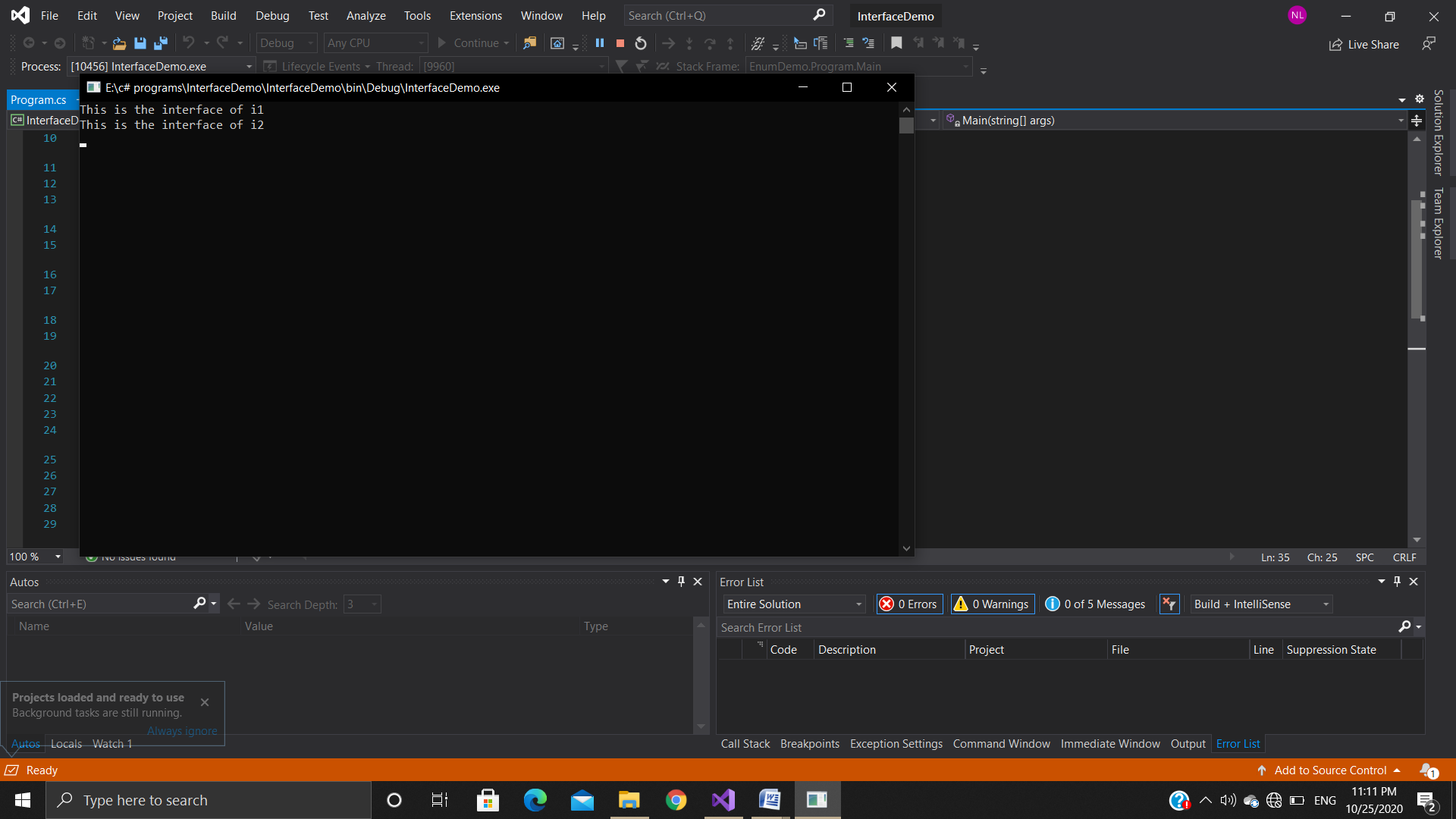
Console.ReadLine();

}

}

}

**Output**

****

1. **Inheritance**

using System;

namespace InheritenceDemo

{

class parent

{

public void msg()

{

Console.WriteLine("This is the parent class");

}

}

class child1 : parent

{

public void msg1()

{

Console.WriteLine("This is child1 class");

}

}

class child2 : parent

{

public void msg2()

{

Console.WriteLine("This is child2 class");

}

}

class Program

{

static void Main(string[] args)

{

child1 c1 = new child1();

c1.msg1();

c1.msg();

Console.WriteLine("----------------------------------------------------");

child2 c2 = new child2();

c2.msg2();

c2.msg();

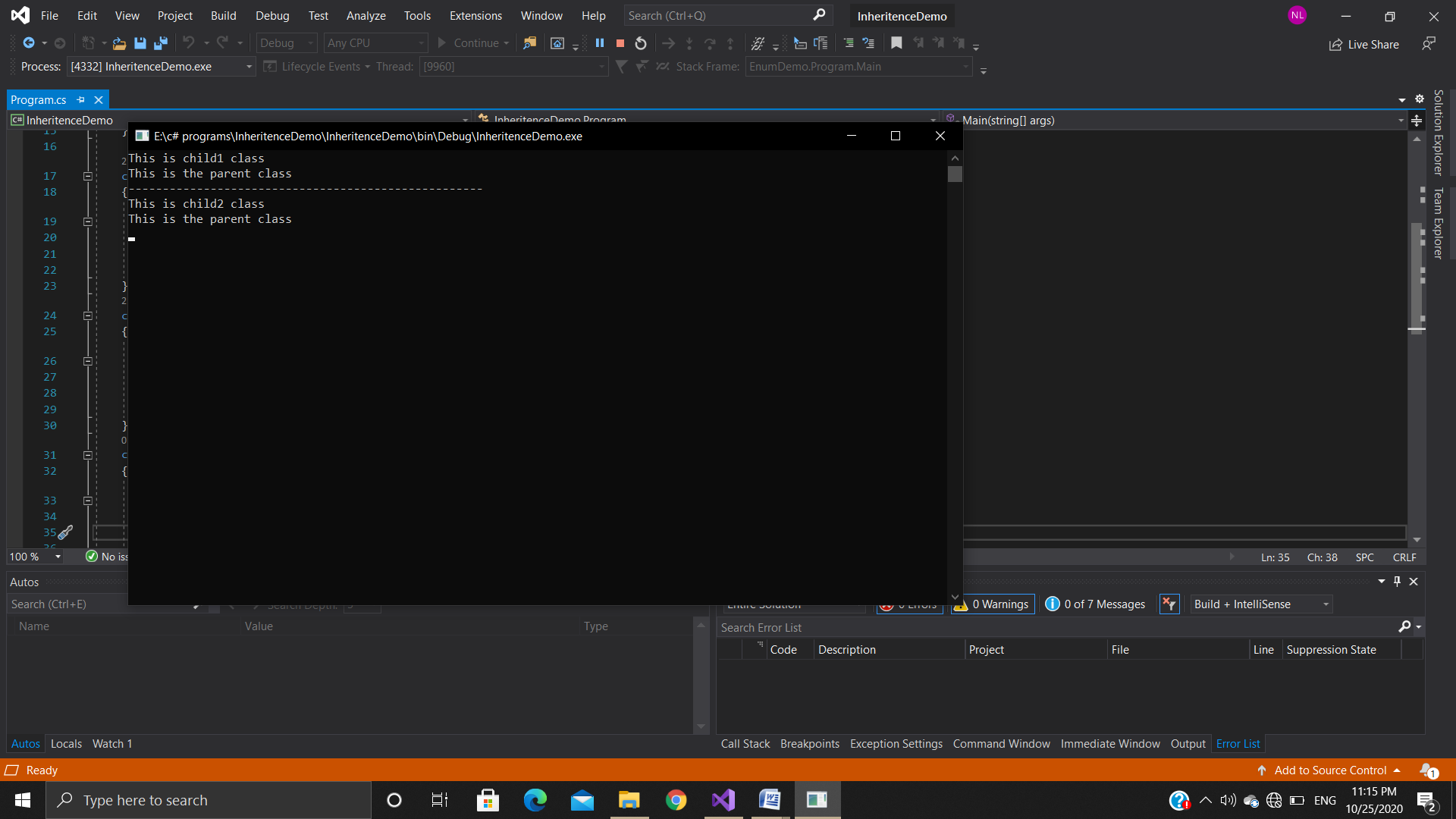
Console.ReadLine();

}

}

}

**Output**

****

1. **Exception Handling**

using System;

namespace ExceptionDemo

{

class Program

{

static void Main(string[] args)

{

try

{

int a = 10;

int b = 0;

int result = a / b;

Console.WriteLine(result);

}

catch(DivideByZeroException ex)

{

Console.WriteLine(ex.Message);

}

finally

{

Console.WriteLine("Finally block executed....");

}

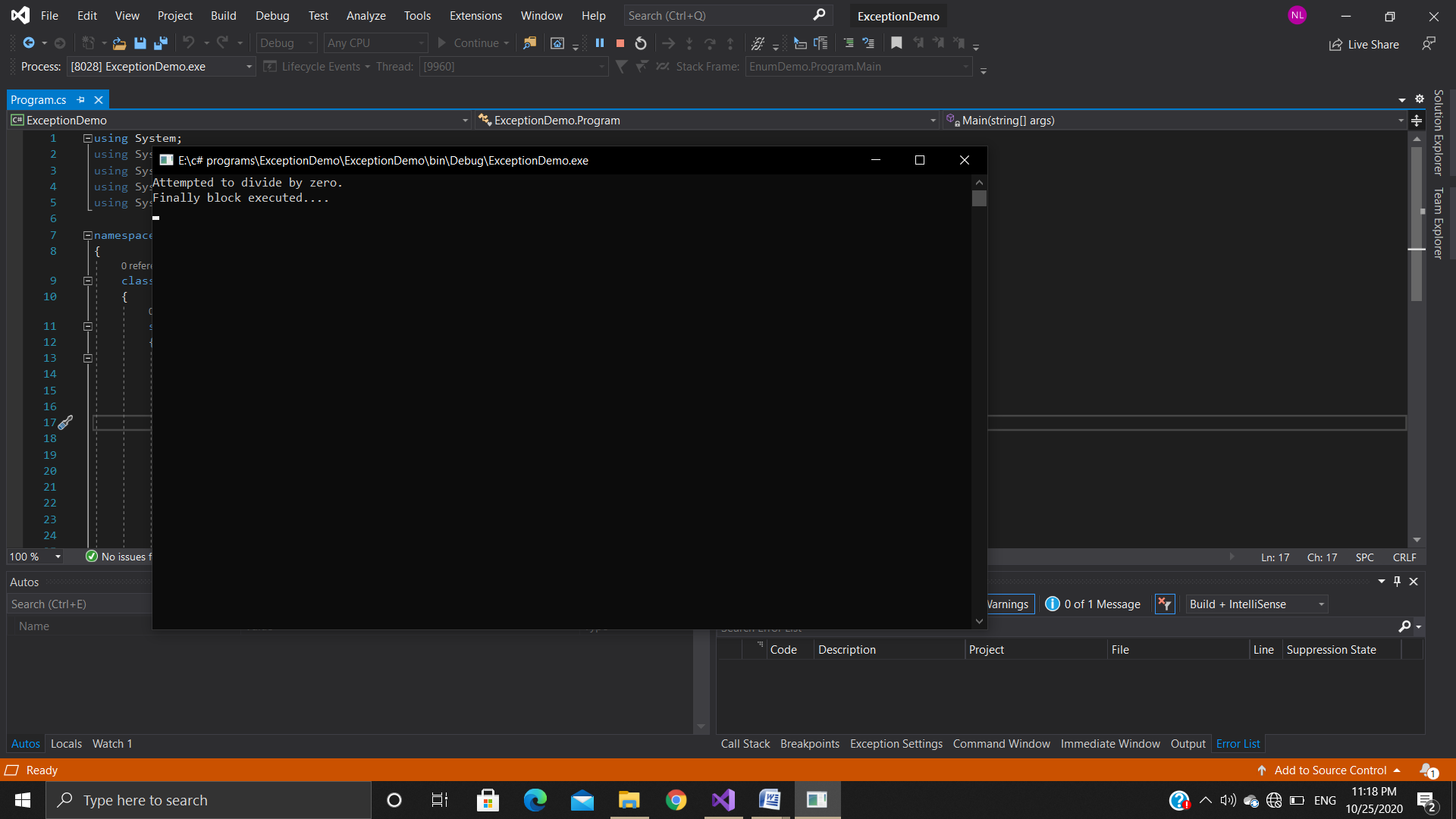
Console.ReadLine();

}

}

}

**Output**

****

1. **Use of Throw**

using System;

namespace ThroeDemo

{

class Program

{

static void Main(string[] args)

{

Console.WriteLine("Enter your age");

int age = int.Parse(Console.ReadLine());

if(age<18)

{

throw new Exception("Access Denied - You must be alteast 18");

}

else

{

Console.WriteLine("You are old Enough to access.");

}

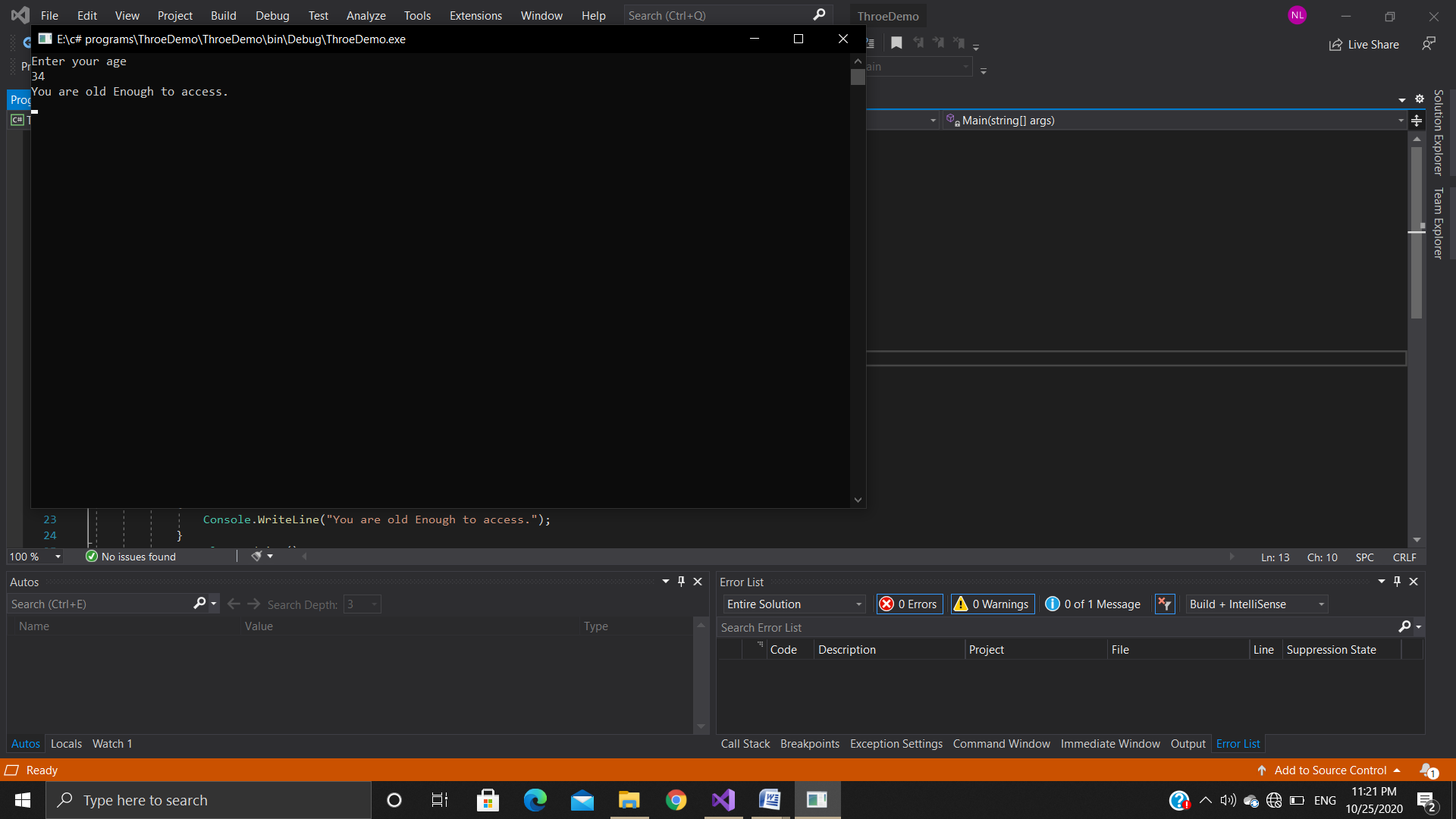
Console.ReadLine();

}

}

}

**Output**

****

1. **Creating a file**

using System;

using System.IO;

namespace filedemo

{

class Program

{

static void Main(string[] args)

{

string filepath = @"E:\MyFile.txt";

using (FileStream fs = new FileStream(filepath, FileMode.OpenOrCreate)) ;

{

if (File.Exists(filepath))

{

Console.WriteLine("File Created Successfully.......");

}

Console.ReadLine();

}

}

}

}

1. **Read write and Append file**

using System;

using System.IO;

namespace FileAppendDemo

{

class Program

{

static void Main(string[] args)

{

string filepath = @"E:\MyFile.txt";

using (FileStream fs = new FileStream(filepath, FileMode.OpenOrCreate, FileAccess.ReadWrite))

{

try

{

using (StreamWriter writer = new StreamWriter(fs))

{

writer.WriteLine("This is MyFile");

writer.WriteLine("My name is Nirali");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

using (FileStream fs = new FileStream(filepath, FileMode.OpenOrCreate, FileAccess.ReadWrite))

{

try

{

using (StreamWriter writer = new StreamWriter(fs))

{

writer.WriteLine("This is MyFile");

writer.WriteLine("My name is Nirali");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

using (FileStream fs1 = new FileStream(filepath, FileMode.Append, FileAccess.Write))

{

try

{

using (StreamWriter writer = new StreamWriter(fs1))

{

writer.WriteLine("This is append operation");

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

}

try

{

using (StreamReader reader = new StreamReader(filepath))

{

string line;

while ((line = reader.ReadLine()) != null)

{

Console.WriteLine(line);

}

}

}

catch (Exception e)

{

Console.WriteLine(e.Message);

}

Console.ReadLine();

}

}

}

}

1. **Events**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace simpleEvent

{

public delegate void EventHandler();

class Program

{

public static event EventHandler \_show;

static void Main(string[] args)

{

\_show += new EventHandler(Add);

\_show += new EventHandler(Sub);

\_show += new EventHandler(Div);

\_show.Invoke();

Console.ReadLine();

}

static void Add()

{

Console.WriteLine("This is Addition");

}

static void Sub()

{

Console.WriteLine("This is Subtraction");

}

static void Div()

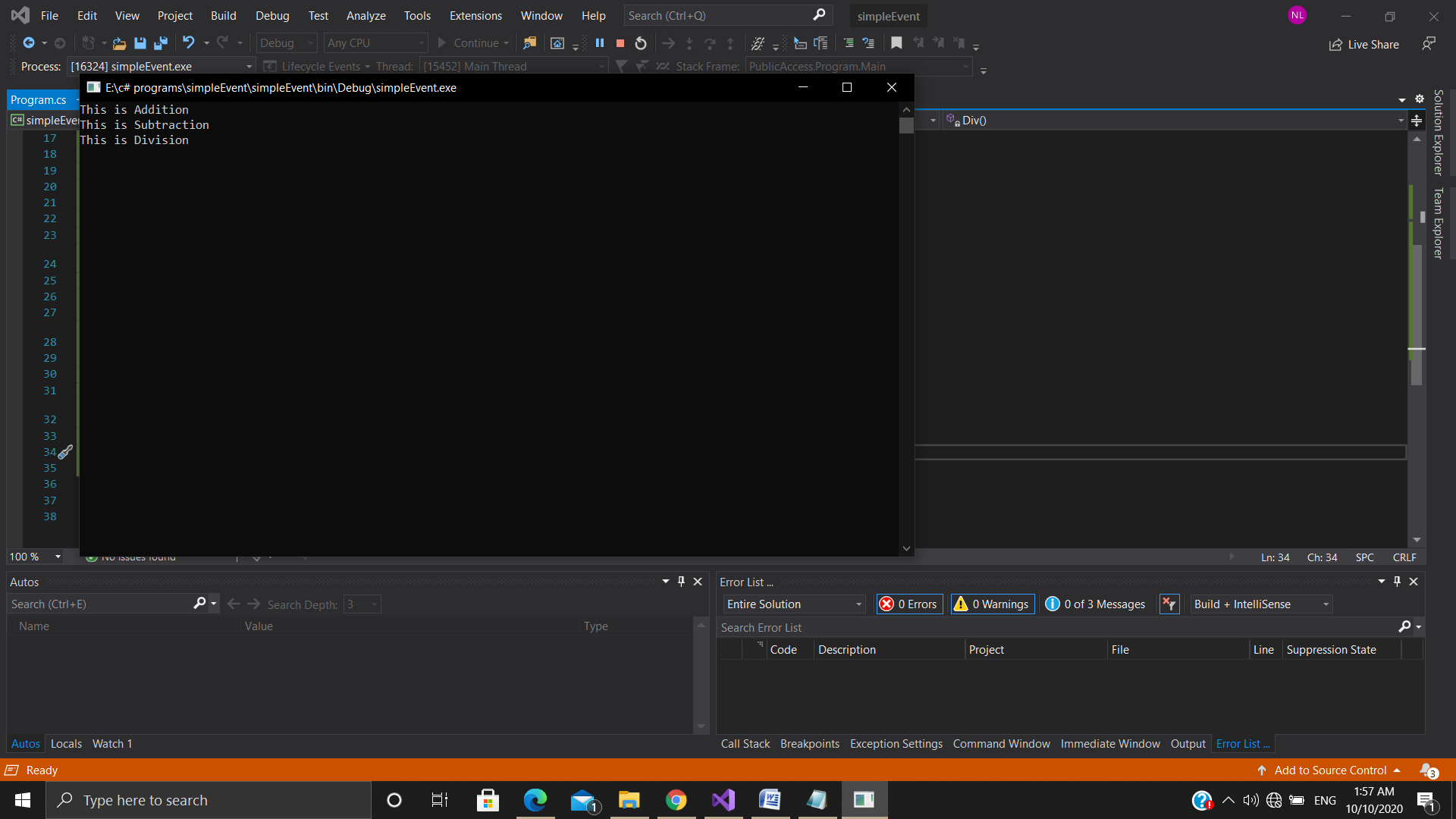
{

Console.WriteLine("This is Division");

}

}

**Output**

****