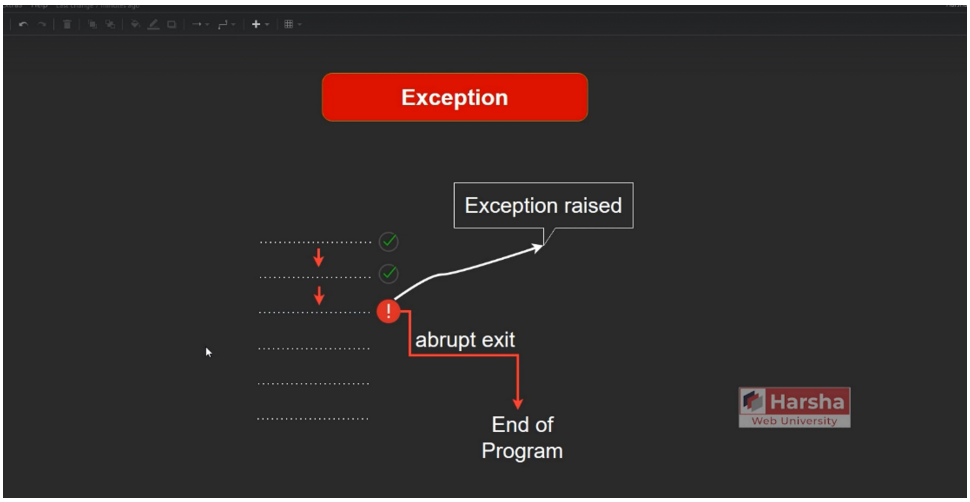
# Section 28: Exception Handling

03 August 2024 12:22


The goal of exception handling is executing some code whenever an exception rises while executing the program.
At first you require to understand what is the meaning of exception. Generally an exception represents a
specific runtime error that can occur while executing your code. For example there is a statement that
converts the string value into a number but what if that string value cannot be converted into number
for example it contains a space or alphabet so cannot be converted into number so exactly at that point of time the CLR treats that this statement is impossible to execute,

so it causes an exception.so that runtime error is represented as a specific exception

For example format exception then clr automatically raises or throws an exception

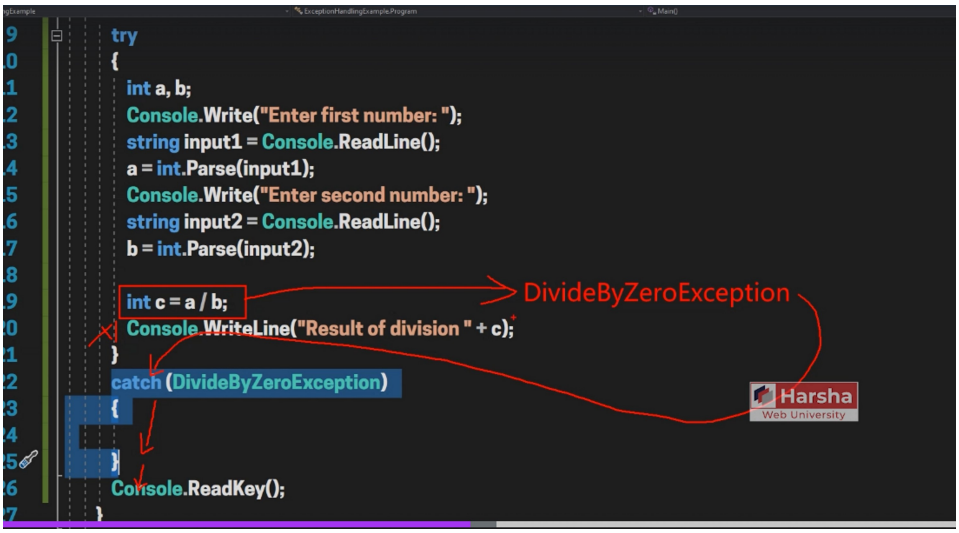
in case if you don't catch that exception in your program by using the try catch block
it leads to abrupt termination of the application

for example see there is a statement here this statement
executed successfully no error

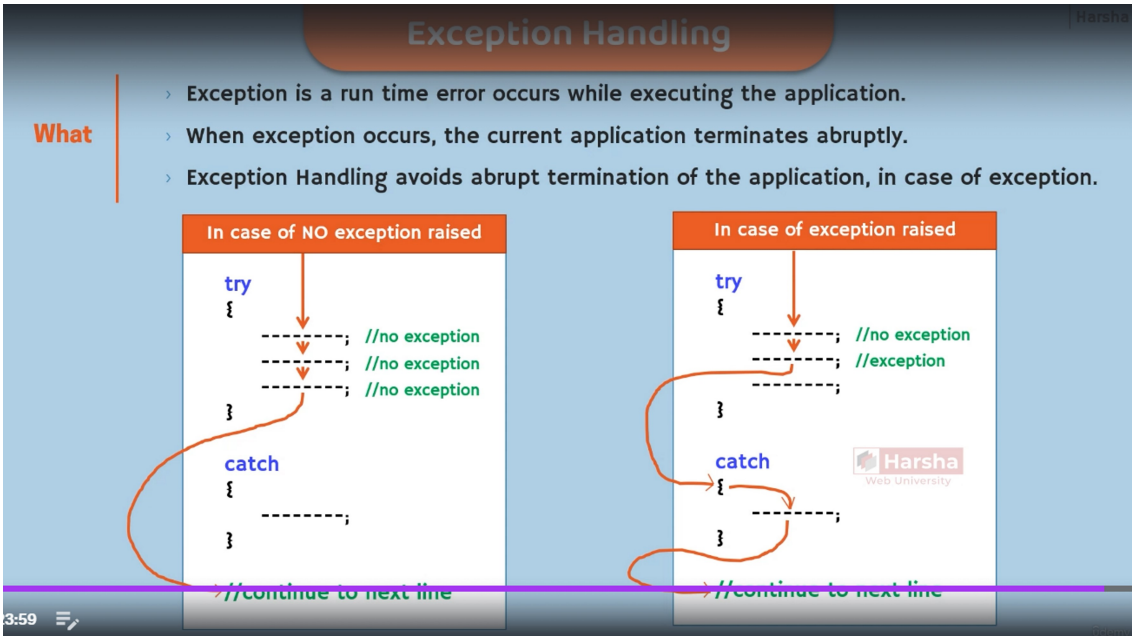
Suppose second statement also executed successfully

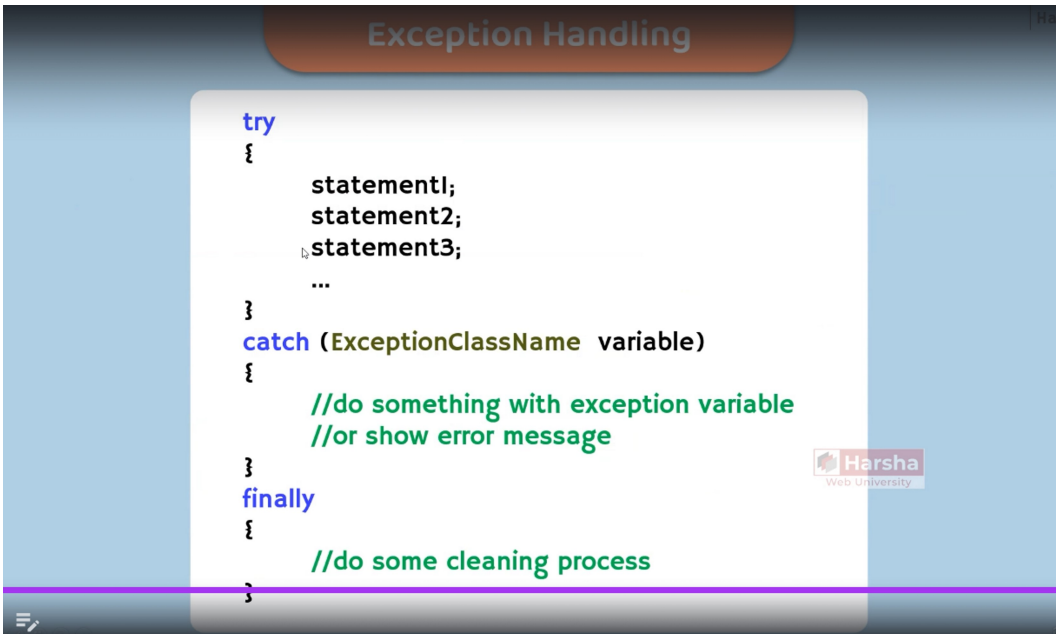
but assume the third statement is a statement that is used to convert the string value
into a number but the input value itself is wrong. you cannot convert your number into string
Because that string contains some alphabet or space, so it is impossible to execute this
particular statement. According to the rulesof .net so that is the root cause of the error

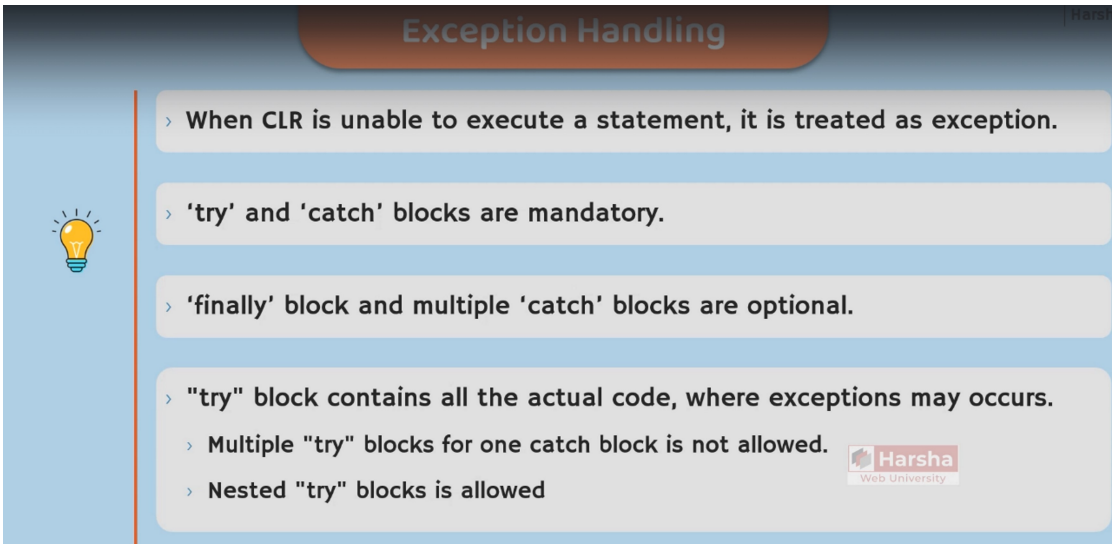
so as soon as the clr is unable to execute this particular line number 3
it automatically creates an exception object, that is called throwing or raising the exception and as soon as that exception is thrown by clr, the application will be abruptly terminated, means it will exit from the application.



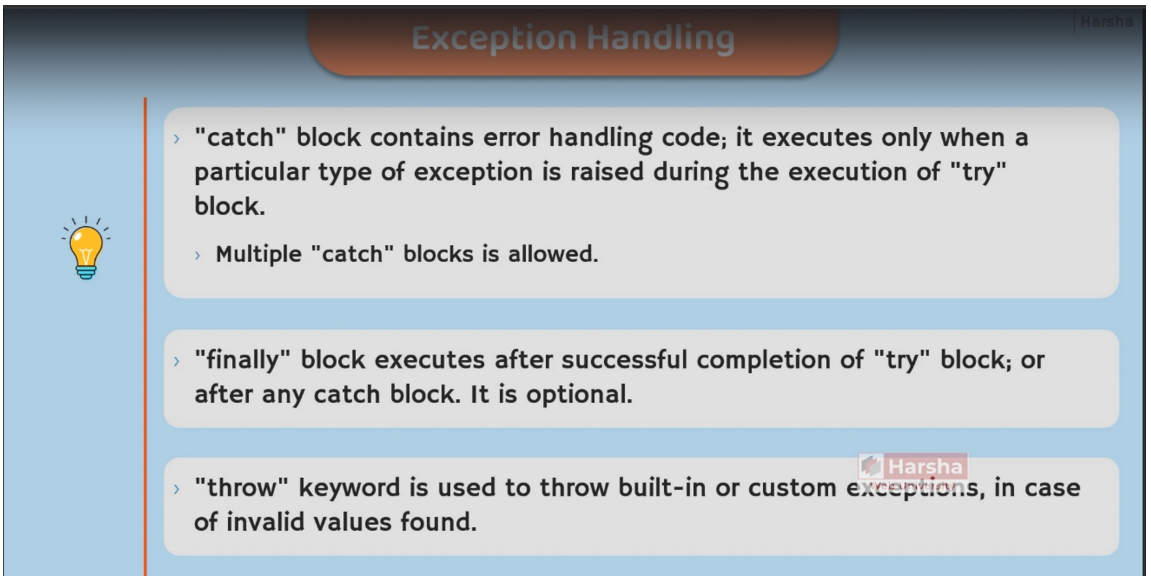

In real life example, a suppose a browser can not download a file, it throws an Exception. but the browser does not close. it says 'unable to download' the file.



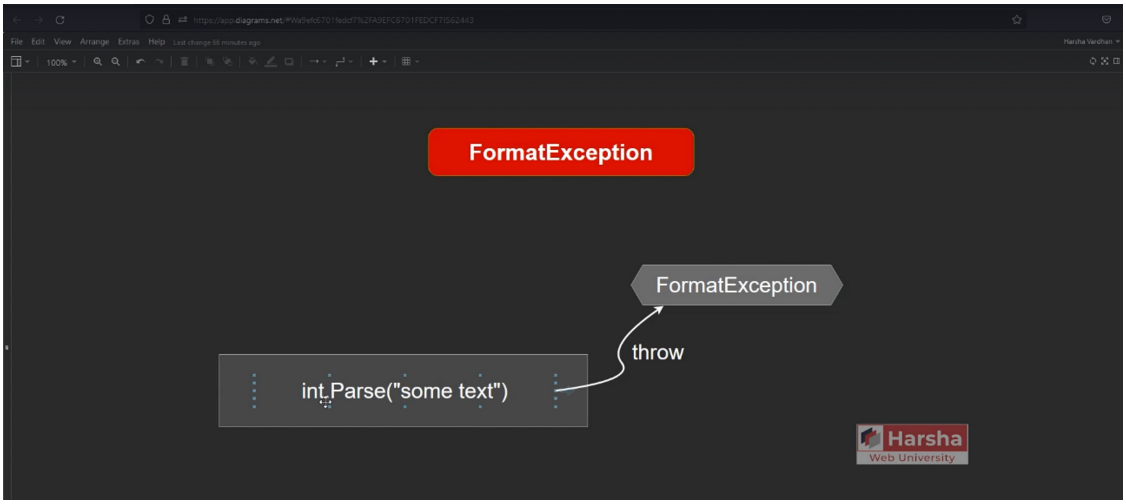


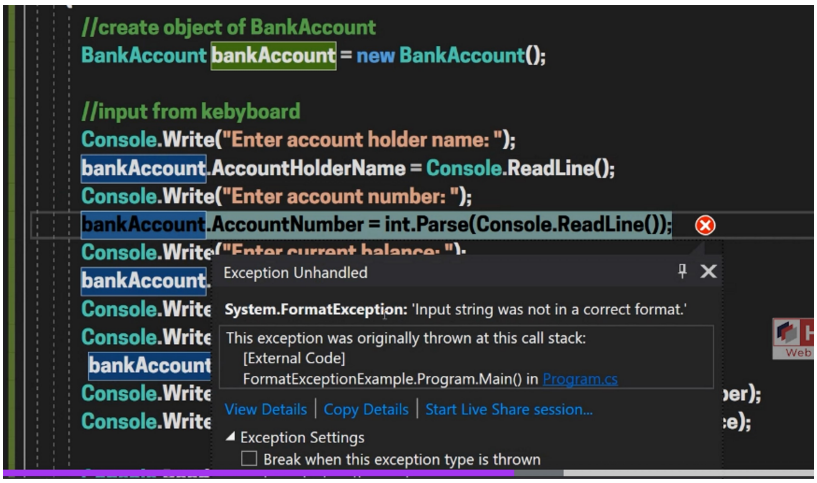

Although, it's a bad practice to write nested try block.

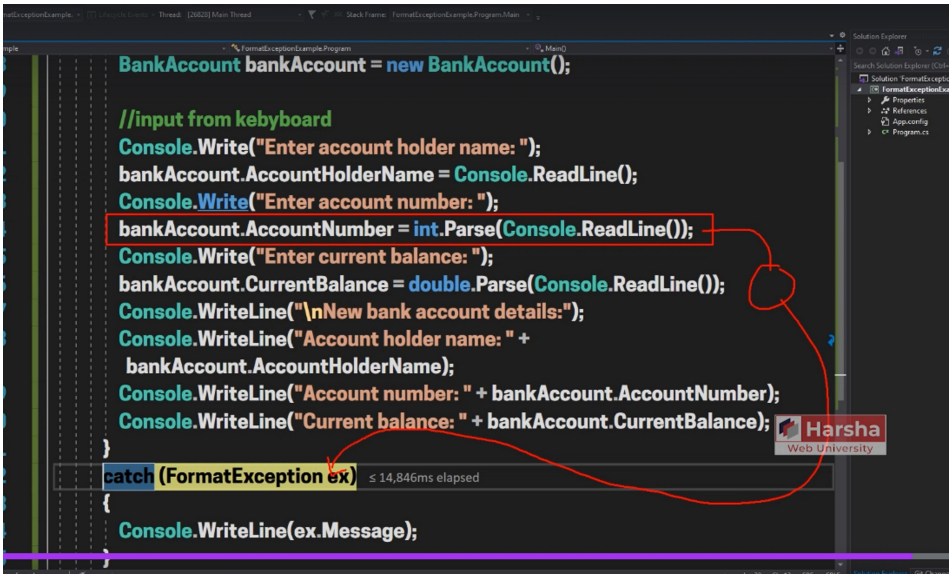


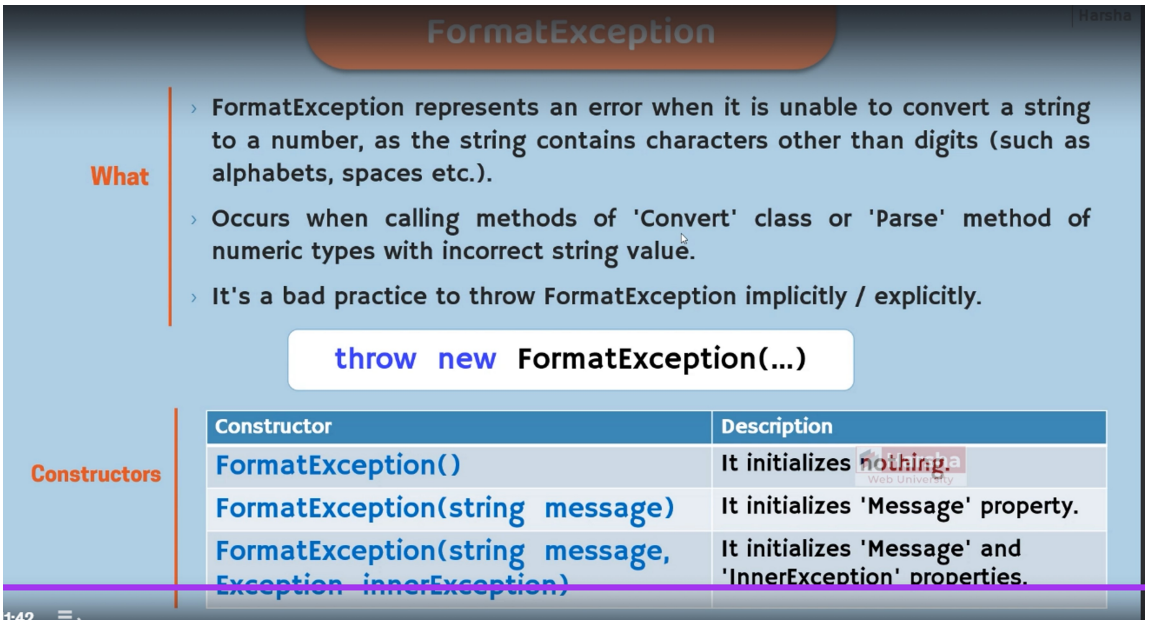
# FormatException



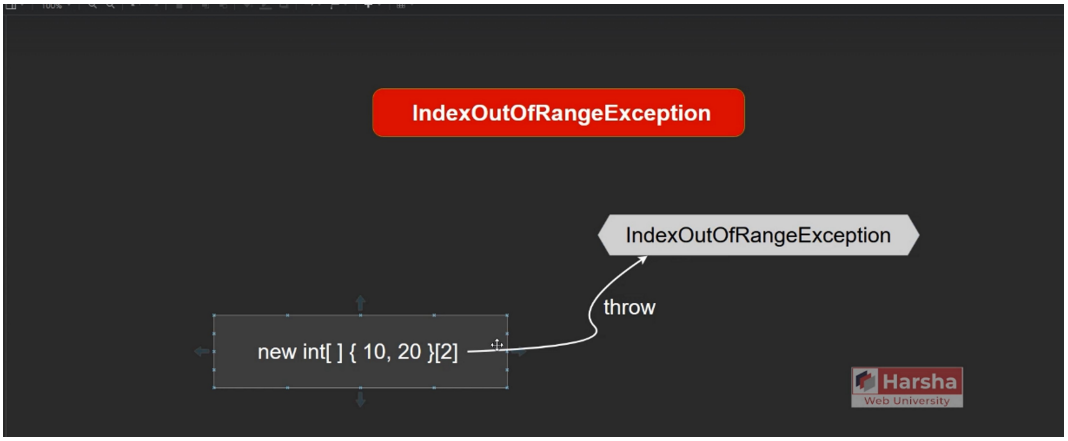

In case while passing the string value into numbers. In case if the given string value cannot be converted into number, for example it might contain a space or alphabet then it cannot be converted right?
so in that case it automatically throws an exception called format exception.




BankAccount bankAccount = new BankAccount();
//input from kebyboard
Console.Write("Enter account holder name: ");
bankAccount.AccountHolderName = Console.ReadLine();
Console.Write("Enter account number: ");
bankAccount.AccountNumber = int.Parse(Console.ReadLine());
Console.Write("Enter current balance: ");
bankAccount.CurrentBalance = double.Parse(Console.ReadLine());
Console.WriteLine("\nNew bank account details:");
Console.WriteLine("Account holder name: " +
bankAccount.AccountHolderName);
Console.WriteLine("Account number: " + bankAccount.AccountNumber);
Console.WriteLine("Current balance: " + bankAccount.CurrentBalance);
}
catch (FormatException ex)
≤ 14,846ms elapsed
{
Console.WriteLine(ex.Message);
}
Harsha
Web University


FormatException
What
› FormatException represents an error when it is unable to convert a string to a number, as the string contains characters other than digits (such as alphabets, spaces etc.).
› Occurs when calling methods of 'Convert' class or 'Parse' method of numeric types with incorrect string value.
› It's a bad practice to throw FormatException implicitly / explicitly.
throw new FormatException(...)
Constructors
Constructor
Description
FormatException()
It initializes nothing.
FormatException(string message)
It initializes 'Message' property.
FormatException(string message, Exception innerException)
It initializes 'Message' and 'InnerException' properties.

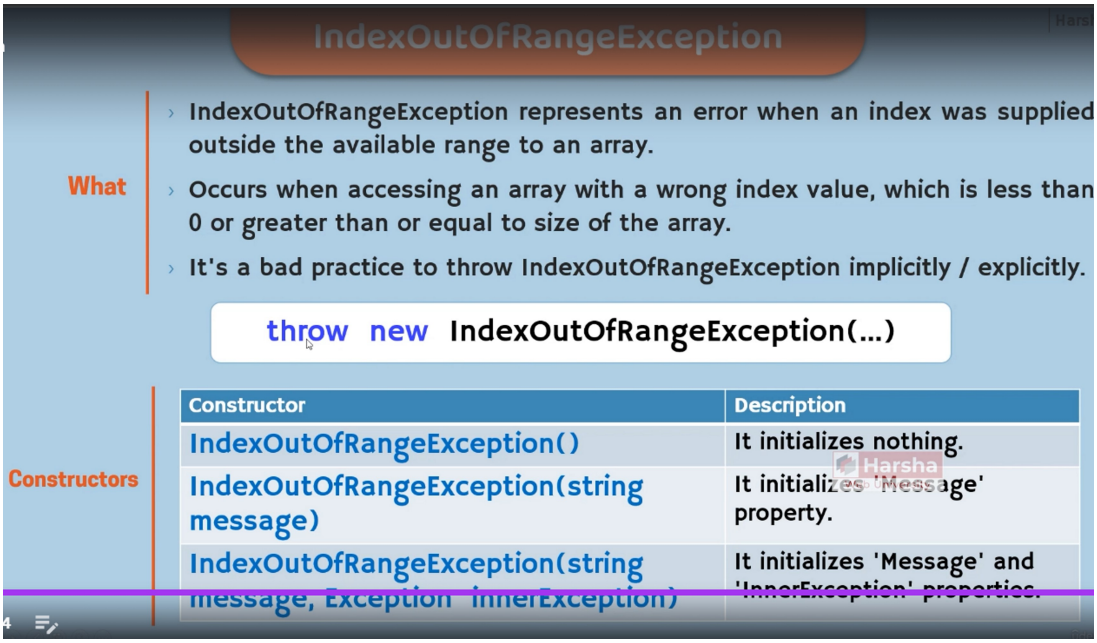

IndexOutOfRangeException



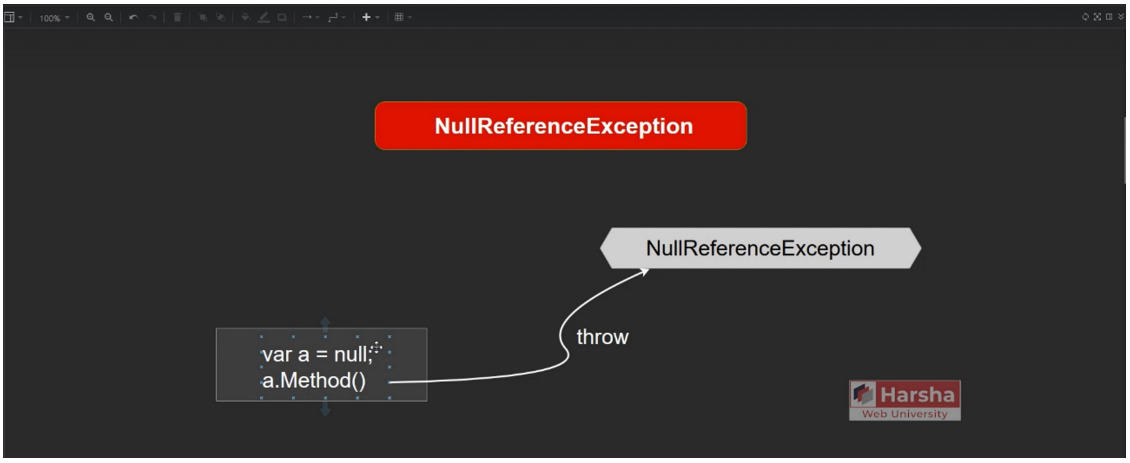

Suppose there is an array and you are trying to retrieve the particular value based on the index but what if you supply some wrong index which is greater than the last index
then it automatically creates and throws an exception

that is index out of range exception in other words this exception represents
an error where the index is incorrect or wrong.

It's advisable to avoid exception in code.

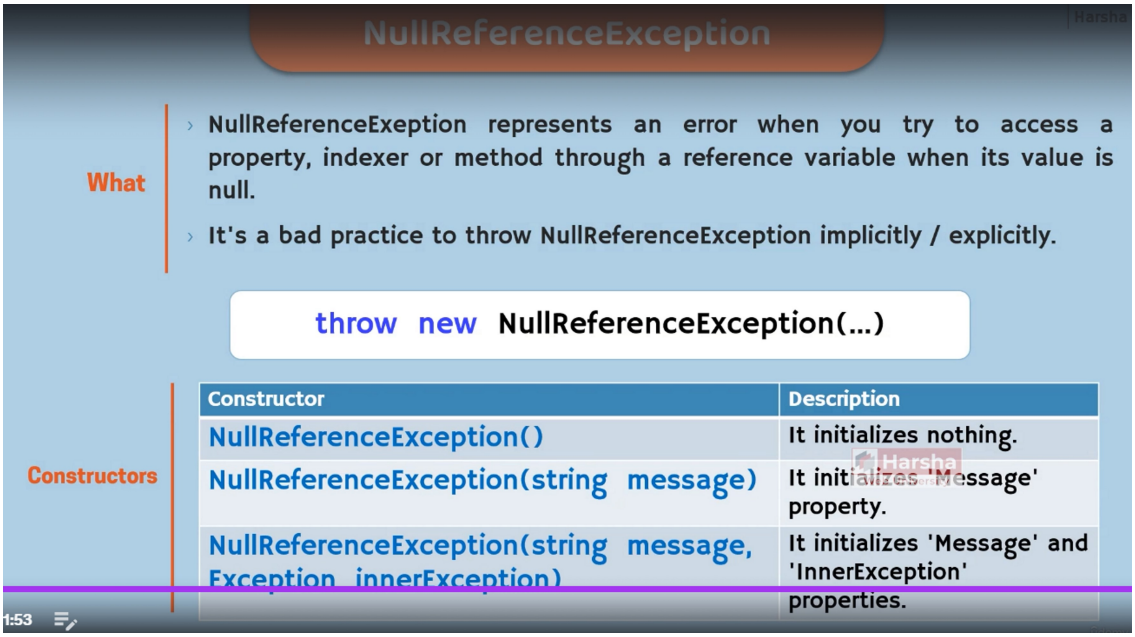


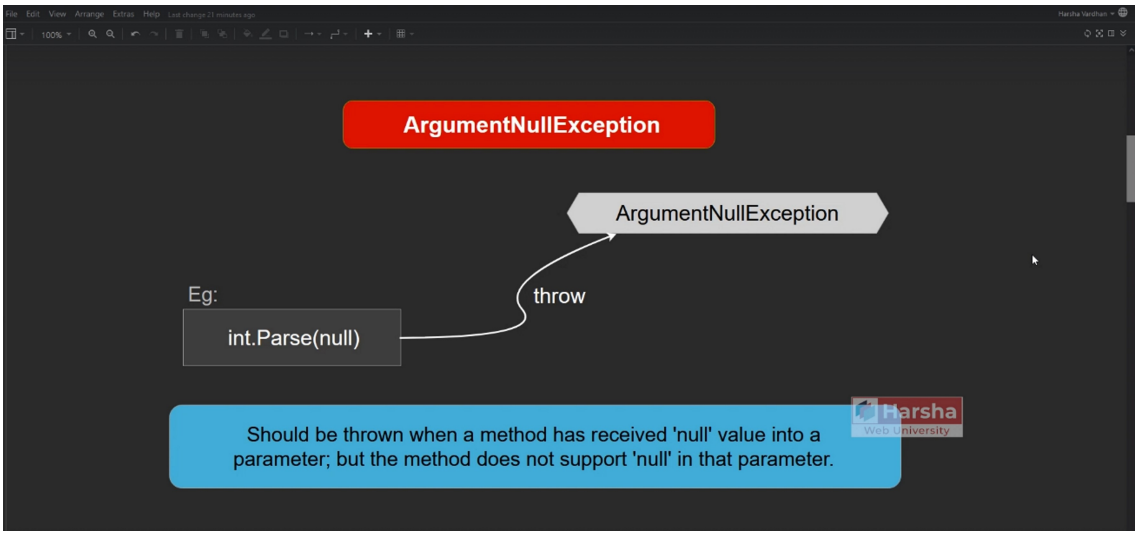
# NullReferenceException

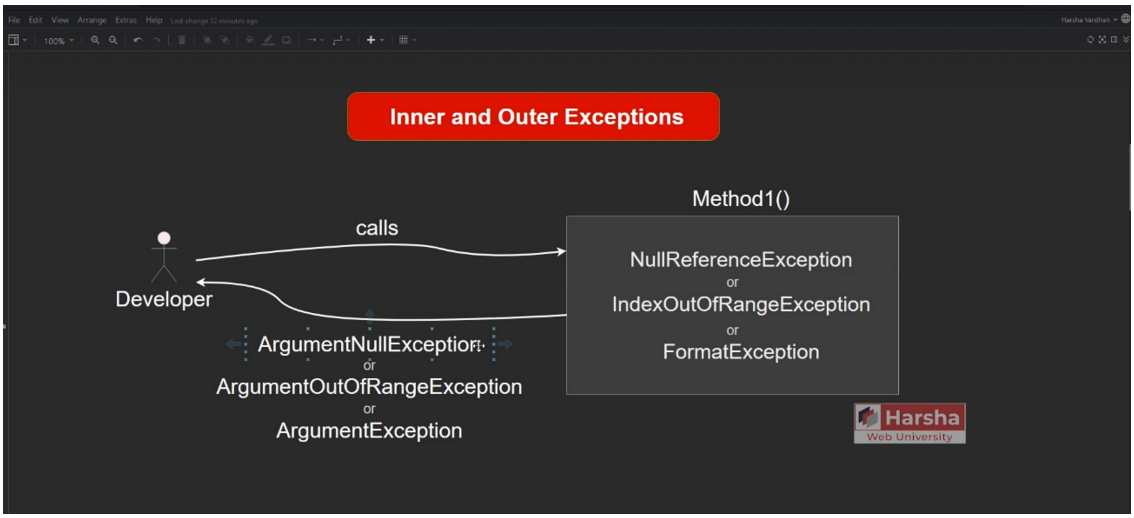



whenever you try to access a method property or indexer through a reference variable and when its value is null it is not allowed in c sharp to represent that error

it automatically creates and throws an exception called null reference exception




For example you can assume your Fund Transfer method, The developer calls that fund transfer
method that means he wants to transfer the money but due to some reason as a part of the progress of this method execution we are unable to complete the process because a variable has a null value, so we cannot do that
in this case it automatically raises a null reference exception

let's say for example we are writing a statement destinationAccount.Balance equal to something
suppose the variable destination account is null, so it automatically raises null reference exception immediately.

.NET automatically raises this exception but in general it is not advisable to report the same exception back to the
calling portion that means if the developer who is the outside person of this method

if he is getting null reference exception from this method that indicates the flawness of the
method, in other words the developer might think that this method was developed with flaws because null reference exception, index out of range exception, format exception

***this type of exceptions in general or internal errors***, they can be raised and catched within the same method
it is not good idea to report the same exceptions back to the calling portion whoever is calling this method.

so that is the reason if the developer of this method wants to design this method in a good way
instead of throwing the same exceptions as it is to the caller instead he can raise a different exceptions
other than these three

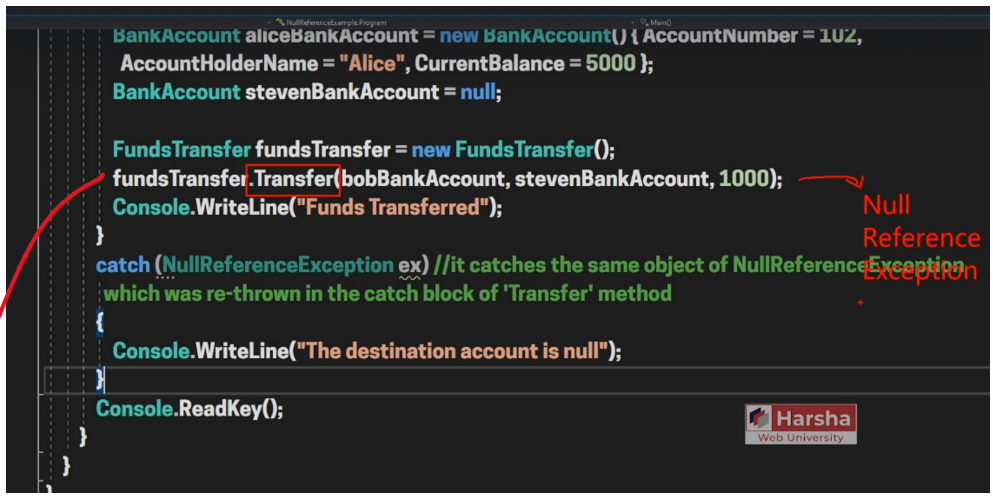
**that is instead of null reference exception he might raise Argument Null exception, the difference between these two is that the null reference exception says that there is somewhere null value in a**

**variable so it is unable to execute some statement. but the meaning of argument null exception is that
the method1 talks to the developer who is the caller of that method, hey developer you have supplied a null value as
argument so that is the reason this method one is unable to execute so the meaning of argument null
exception will be rather than blaming the method one itself**

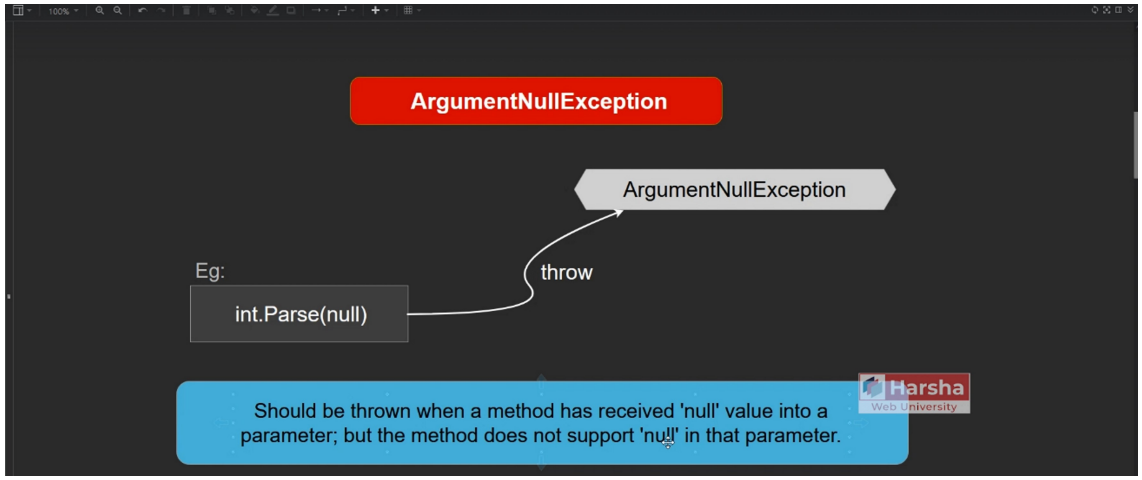
the method1 reports that you have supplied the null value incorrectly, so go and check that
this is the report given from method one to the developer

that indicates the mistake is by the developer that means the caller but it is not the actual mistake of the
method one, so this is the difference between null reference exception and argument null exception and moreover in addition in the argument null exception you can include the variable
which is the exact null value
but null reference exception is not it doesn't tell you which argument or
which variable is exactly null, so that is why rather than throw in the

null reference exception from the method it is always better to throw argument null exception.




Instead of throwing **Null Reference Exception** from that method, It's better to convert the same as **'Argument Null Exception'**




## ArgumentNullException

**What**

- ArgumentNullException represents an error when a null value is passed as argument to a method; and that method doesn't accept null's into that parameter.
- It's a good practice to throw ArgumentNullException implicitly / explicitly.

```
throw new ArgumentNullException(...)
```

Harsha Web University

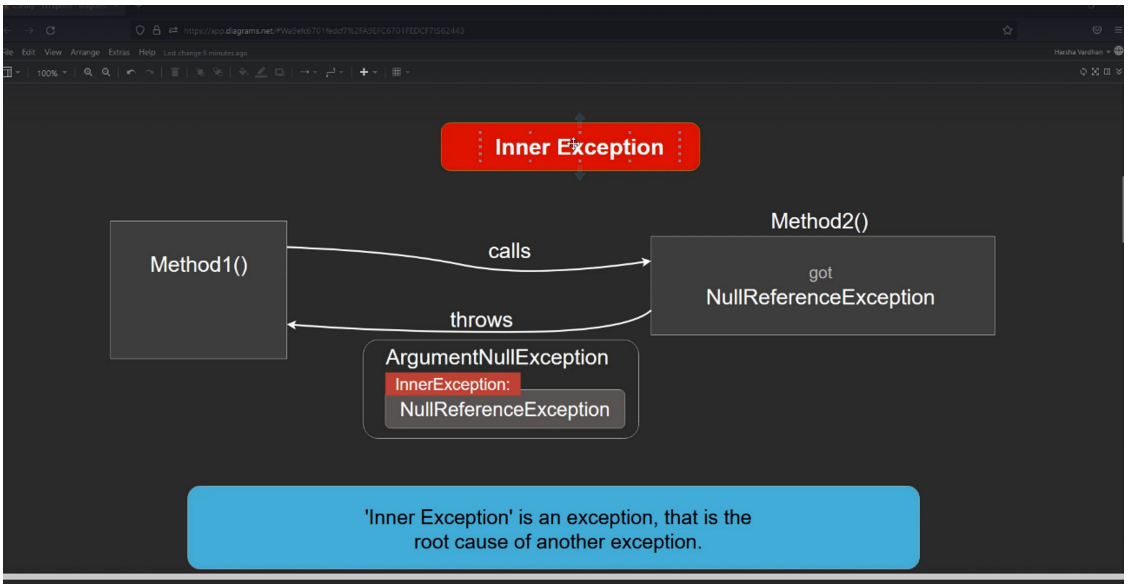
## ArgumentNullException

**Constructors**

| Constructor | Description |
|---|---|
| ArgumentNullException() | It initializes nothing. |
| ArgumentNullException(string paramName) | It initializes 'ParamName' property. |
| ArgumentNullException(string message, Exception innerException) | It initializes 'Message' and 'InnerException' properties. |
| ArgumentNullException(string paramName, string message) | It initializes 'ParamName' and 'Message' properties. |

Harsha Web University

# Inner Exception




The inner exception is an exception which is a root cause or at least a cause of another exception.

What is an Inner Exception? An inner exception is an exception that is nested within another exception. When an exception occurs, it can be caused by various factors. Sometimes, the root cause of the exception lies within a different part of the code or a deeper layer of the call stack. In such cases, an inner exception provides additional context about what went wrong.

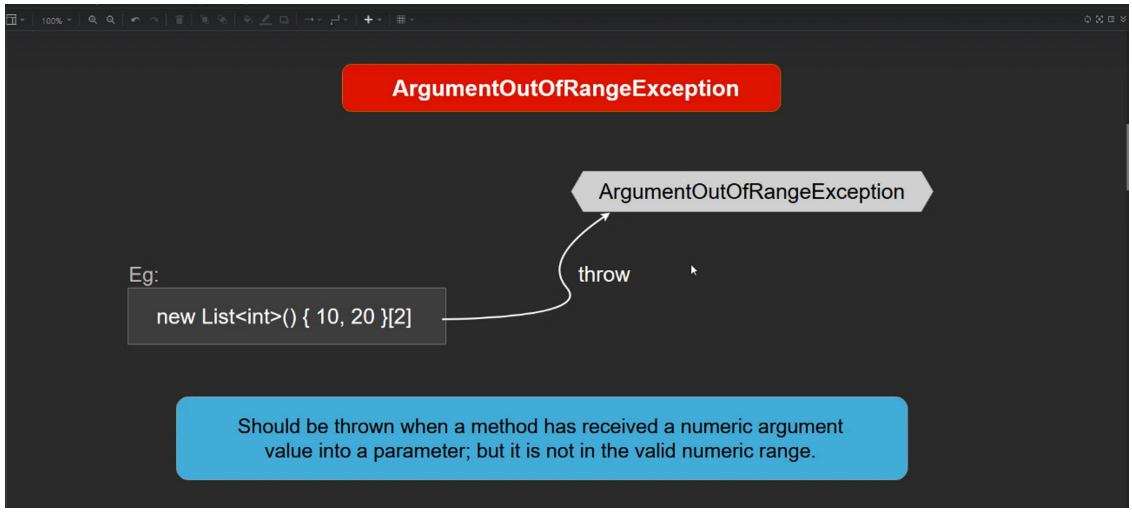
Here's why inner exceptions are useful:

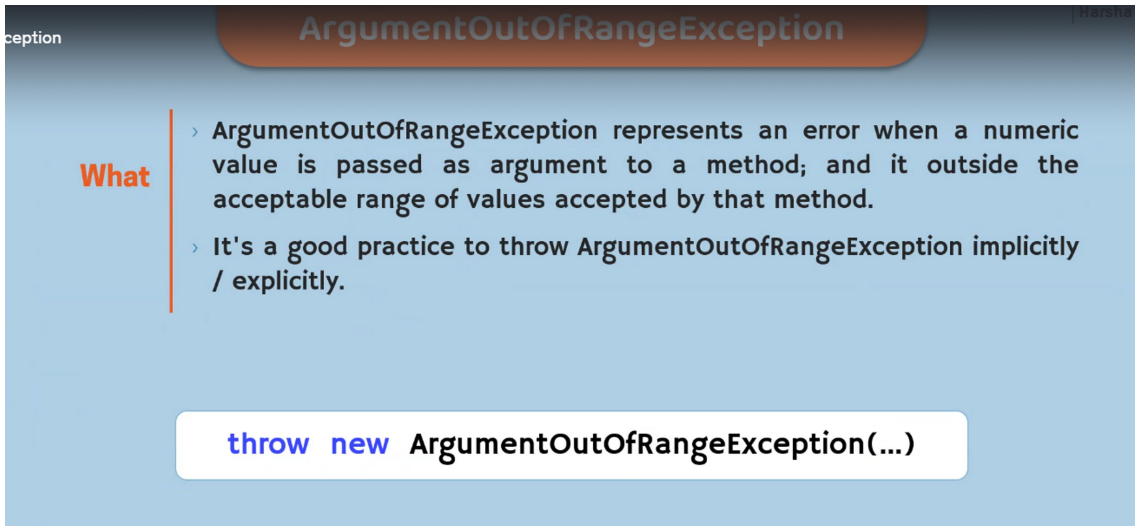
Layered Error Information: Imagine you have a method that calls another method, which in turn calls yet another method. If an exception occurs deep down the call stack, the outermost method (the one initially invoked) might not have enough information to handle the error effectively. By using inner exceptions, you can propagate detailed error information up the stack.
Preserving the Original Cause: When an exception occurs, it's essential to preserve the original cause. An inner exception allows you to wrap the original exception while adding more specific details. This way, you don't lose track of the root issue.
Debugging and Troubleshooting: When debugging, examining the inner exceptions can help pinpoint the exact location of the problem. It allows developers to trace back through the layers of code to find the source of the issue.
Now, let's see an example in C#:

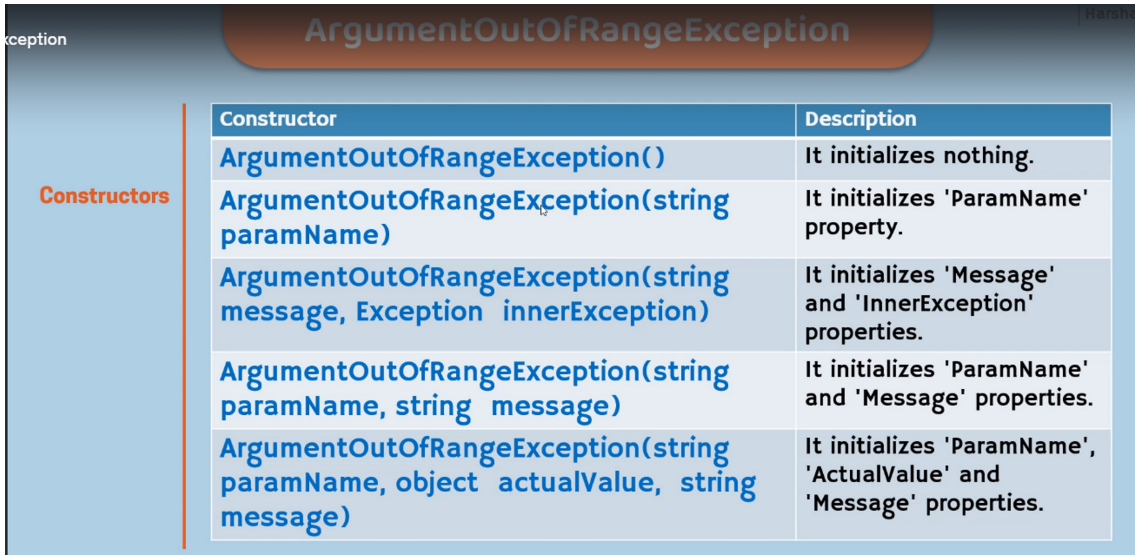
Suppose we have a simple scenario where we're reading data from a file. If an error occurs during file reading, we want to handle it gracefully and provide meaningful information to the user.

```
using System;
```

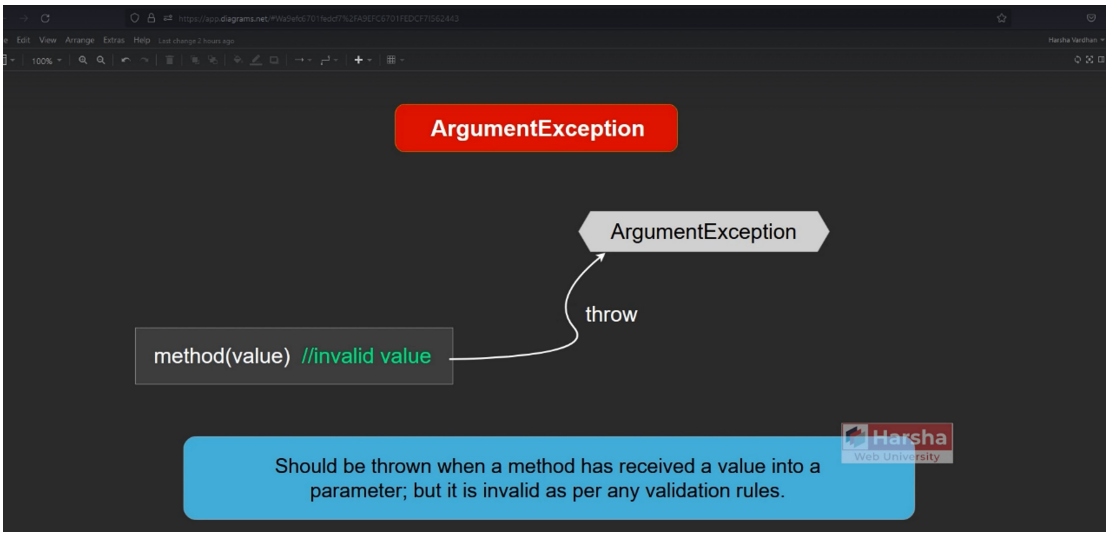
```
class Program
{
    static void Main()
    {
        try
        {
            ReadDataFromFile();
        }
        catch (FileReadException ex)
        {
            Console.WriteLine($"Error reading data: {ex.Message}");
            if (ex.InnerException != null)
            {
                Console.WriteLine($"Inner exception: {ex.InnerException.Message}");
            }
        }
    }

    static void ReadDataFromFile()
    {
        try
        {
            // Simulate reading data from a file (this could throw an exception)
            throw new FileReadException("File not found", new UnauthorizedAccessException("Access denied"));
        }
        catch (Exception ex)
        {
            throw new FileReadException("Error reading data", ex);
        }
    }
}

class FileReadException : Exception
{
    public FileReadException(string message, Exception innerException)
        : base(message, innerException)
    {
    }
}
```

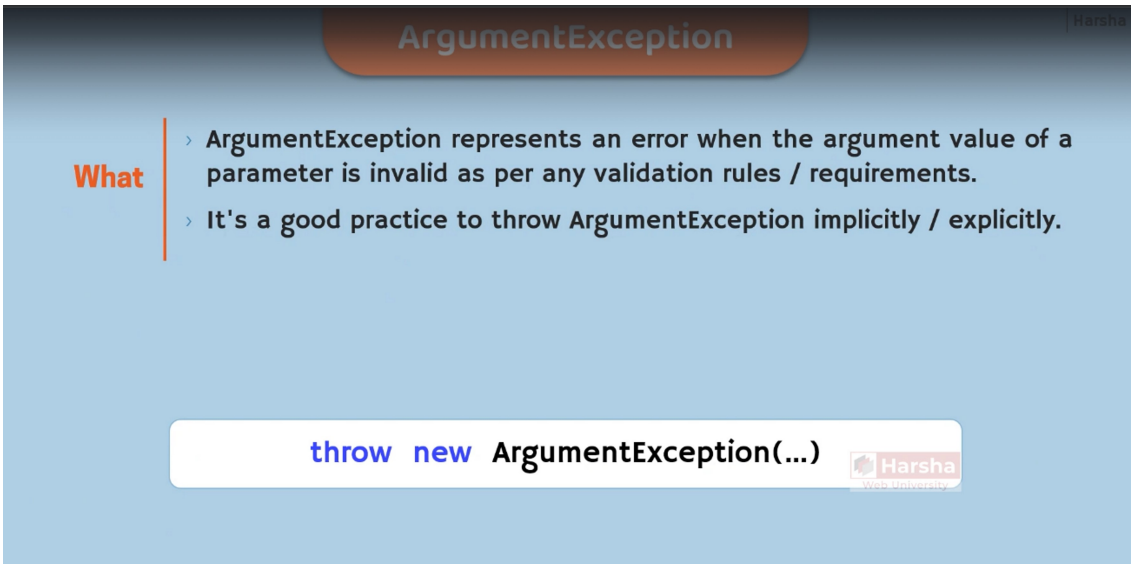

ArgumentOutOfRangeException
ArgumentOutOfRangeException
Eg:
new List<int>() { 10, 20 }[2]
throw
Should be thrown when a method has received a numeric argument value into a parameter; but it is not in the valid numeric range.


ArgumentOutOfRangeException
What
› ArgumentOutOfRangeException represents an error when a numeric value is passed as argument to a method; and it outside the acceptable range of values accepted by that method.
› It's a good practice to throw ArgumentOutOfRangeException implicitly / explicitly.
throw new ArgumentOutOfRangeException(...)


ArgumentOutOfRangeException
Constructors
Constructor
Description
ArgumentOutOfRangeException()
It initializes nothing.
ArgumentOutOfRangeException(string paramName)
It initializes 'ParamName' property.
ArgumentOutOfRangeException(string message, Exception innerException)
It initializes 'Message' and 'InnerException' properties.
ArgumentOutOfRangeException(string paramName, string message)
It initializes 'ParamName' and 'Message' properties.
ArgumentOutOfRangeException(string paramName, object actualValue, string message)
It initializes 'ParamName', 'ActualValue' and 'Message' properties.

# ArgumentException



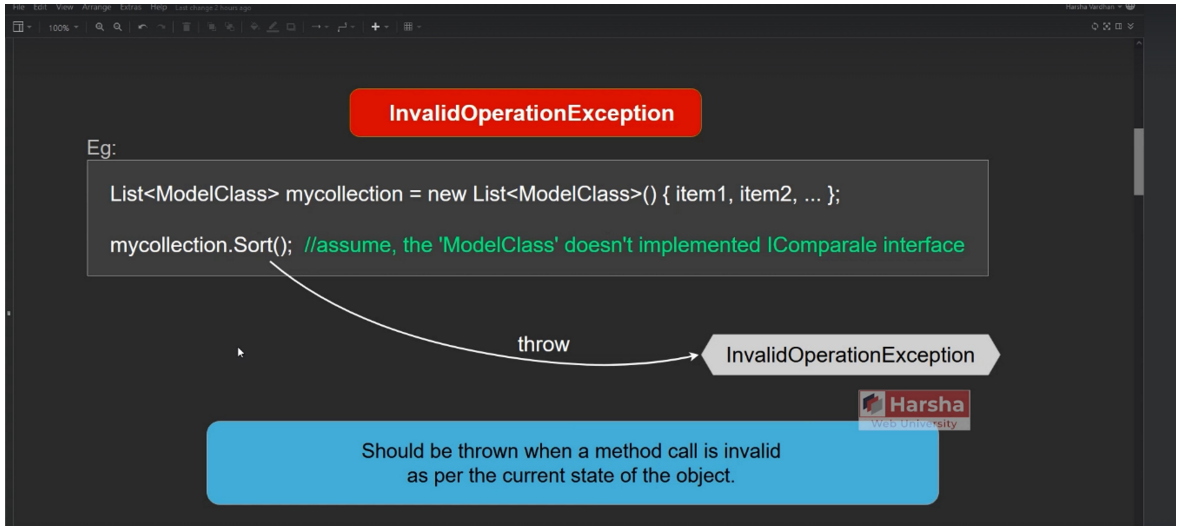



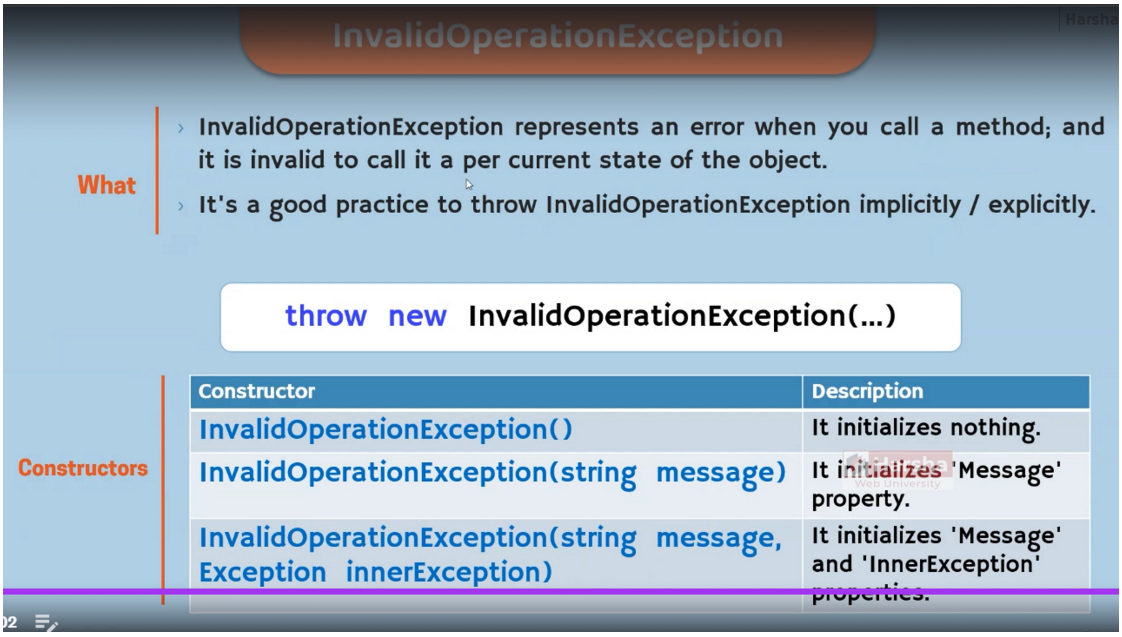
## ArgumentException

Constructors

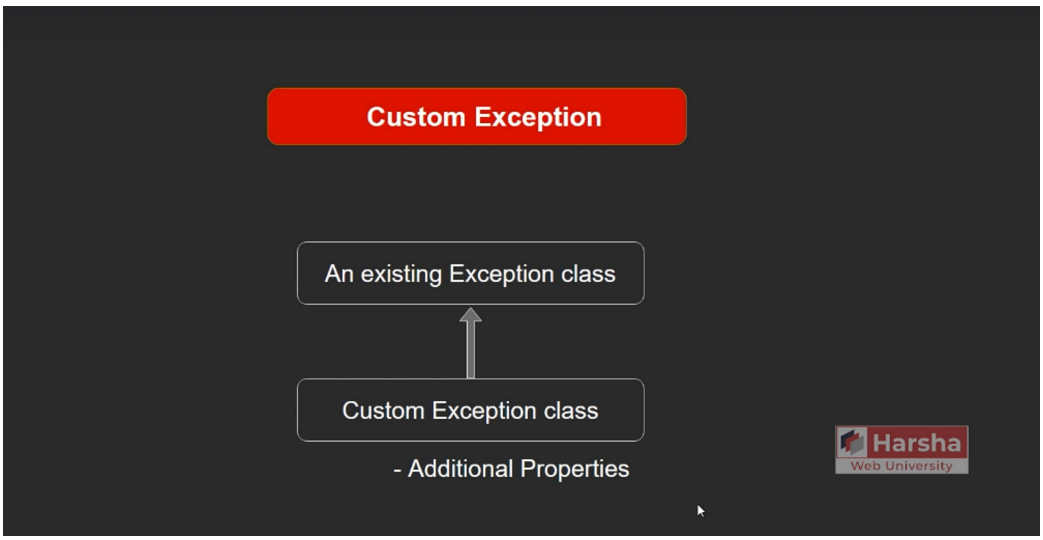
| Constructor | Description |
|---|---|
| ArgumentException() | It initializes nothing. |
| ArgumentException(string message) | It initializes 'Message' property. |
| ArgumentException(string message, Exception innerException) | It initializes 'Message' and 'InnerException' properties. |
| ArgumentException(string message, string paramName) | It initializes 'Message' and 'ParamName' properties. |
| ArgumentException(string message, string paramName, string innerException) | It initializes 'Message', 'ParamName' and 'InnerException' properties. |

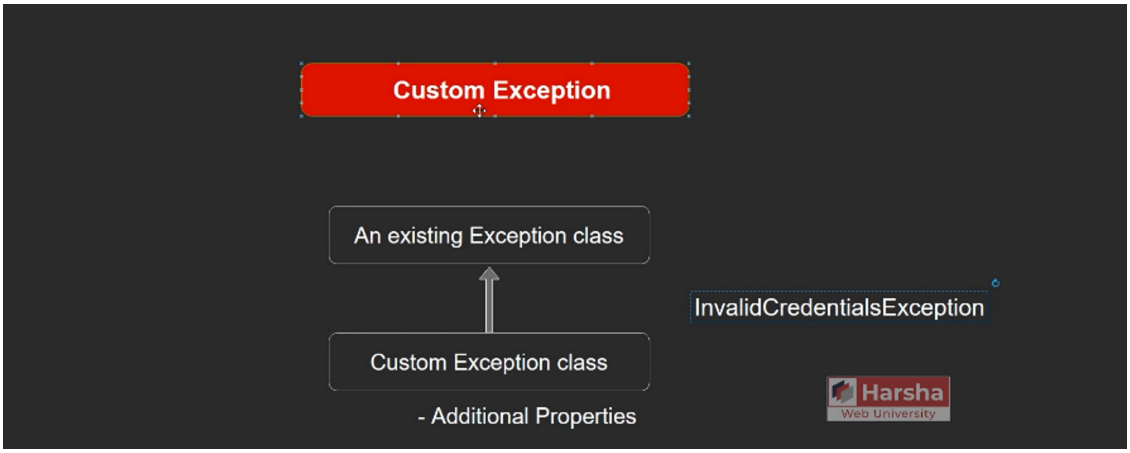
# InvalidOperationException




InvalidOperationException
What
› InvalidOperationException represents an error when you call a method; and it is invalid to call it a per current state of the object.
› It's a good practice to throw InvalidOperationException implicitly / explicitly.
throw new InvalidOperationException(...)
Constructors
Constructor
Description
InvalidOperationException()
It initializes nothing.
InvalidOperationException(string message)
It initializes 'Message' property.
InvalidOperationException(string message, Exception innerException)
It initializes 'Message' and 'InnerException' properties.


Custom Exception


Custom Exception
An existing Exception class
Custom Exception class
- Additional Properties
Harsha
Web University




we have understood **index out of range** null reference and several other type of exceptions

so it most of the common situations you will be using either of these exceptions. for example let's say when the argument value is passed as null incorrectly then
you will use the **argument null exception,** for example the incoming value or argument value is passed as incorrectly maybe format is wrong or the value itself is wrong in that case you will use **argument exception**

like that we have seen several types of exceptions

*but in the real-world development there may be some situations where you would like to represent a different type of error other than these which are mentioned so far or you would like to represent a*
*specific type of error in your application which is much important.*

let's say for example there is an application in the login page the user has entered the username and password for login and that login is failed

so to represent that particular error there is no any built-in predefined exception class

so exactly for those situations you have to create your own exception class and that is called **custom exception**. so you will create your user defined exception class which is also called as custom exception class inheriting from a predefined exception
class.

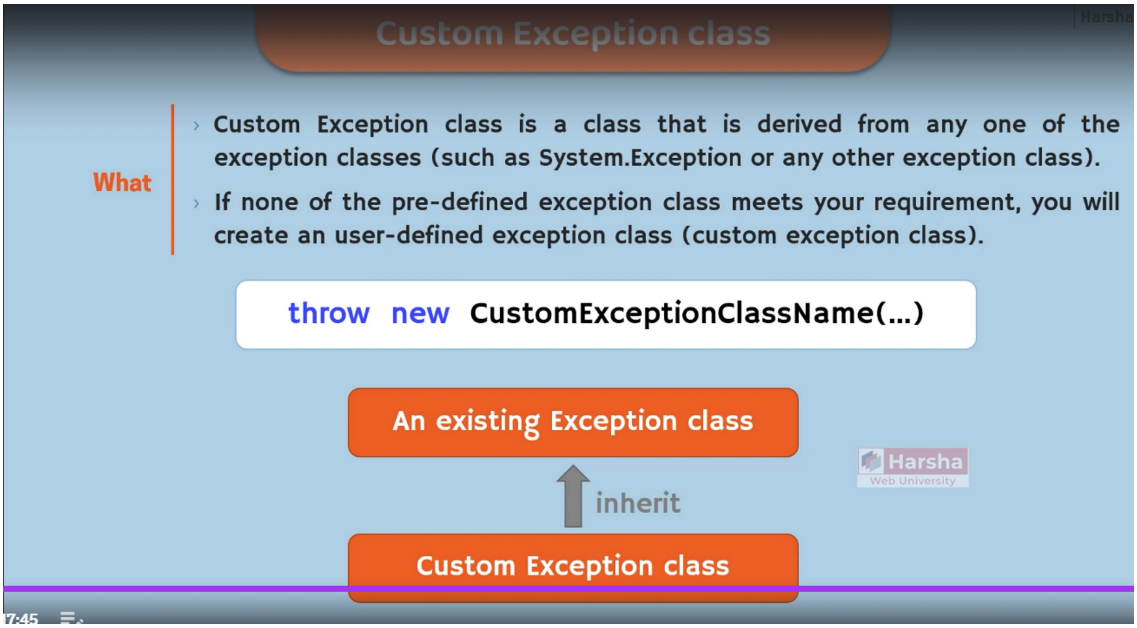
it can be either of these exceptions what we have learned so far. for example you can inherit from

argument exception or argument null exception or invalid operation exception or any other exception class what we have learned so far.

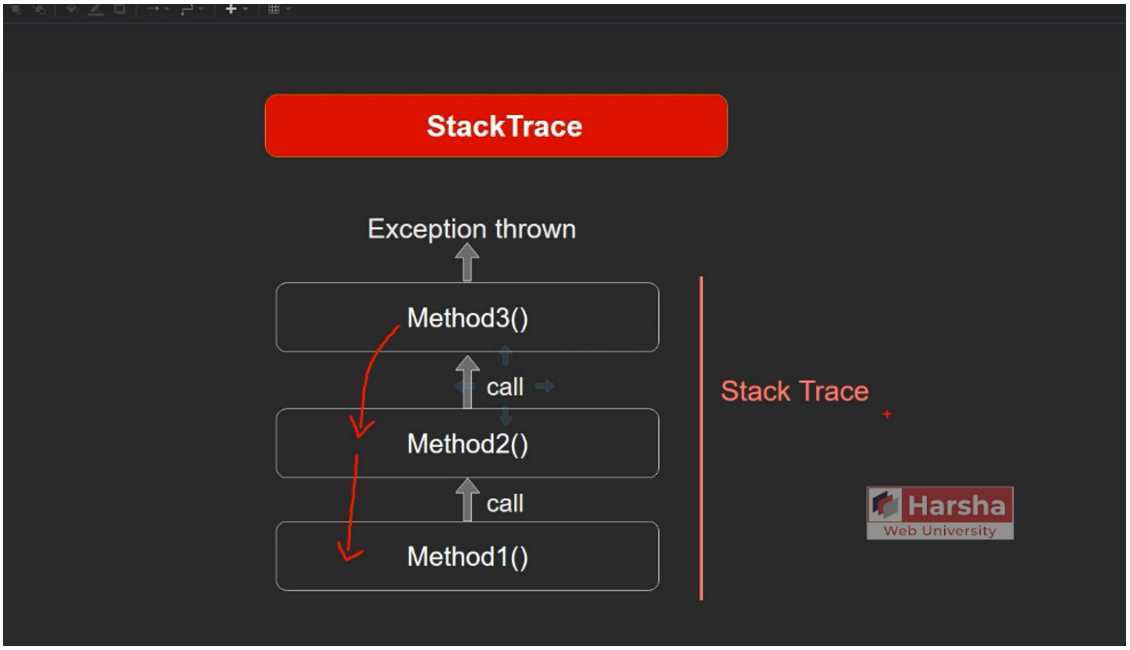
so for that login example you might be creating an exception class saying that invalid credentials exception which indicates the username or password or both of them are incorrect

so what is the benefit of creating the user defined exception you will have your own exception class
that can represent a specific application related error, for example the login is failed or the registration form values are passed incorrectly

otherwise unable to connect to server or any other specific application related error so for representation of a specific type of error it is best to create a custom exception class. so that you can log the same exception name in the exception logs or also you can ask the other co-developers to handle that specific exception.



# Stack Trace




suppose there is a method called method one and in that method one we are calling the method two and in the method two we are calling the method 3.

while executing the method 3 somewhere in the middle some exception was thrown for example null reference exception is thrown
then that null reference exception which was actually thrown as a part of method 3
will be reported back to the caller

so who is the caller of that method 3? method 2 right?

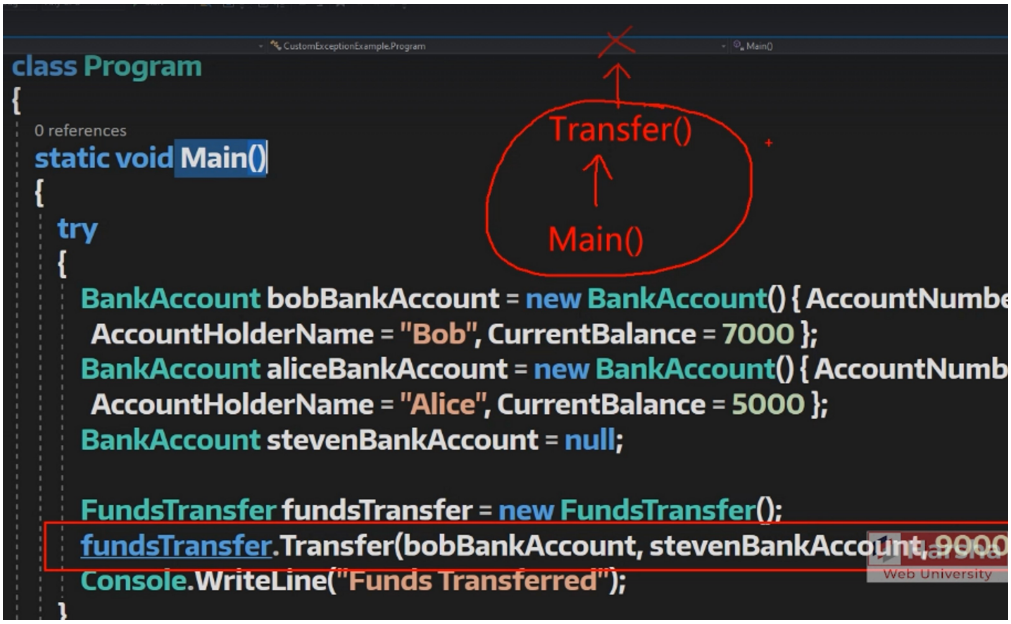
so that method 3 will report that null reference exception back to the method 2
and method 2 also will not keep quiet; it reports the same exception back to the method 1.

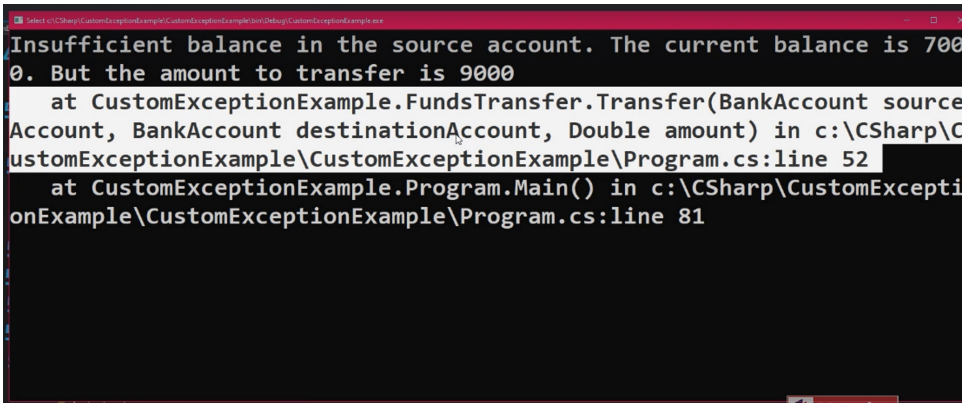
so this same order of calling the methods is called as stacked rise
it is generally stored in the exception log

or printed in the console so that the other developers can
identify the reason or when exactly the exceptions are being
raised

for example someone calls you and ask you that
your application is not working so then you can investigate this strac trace

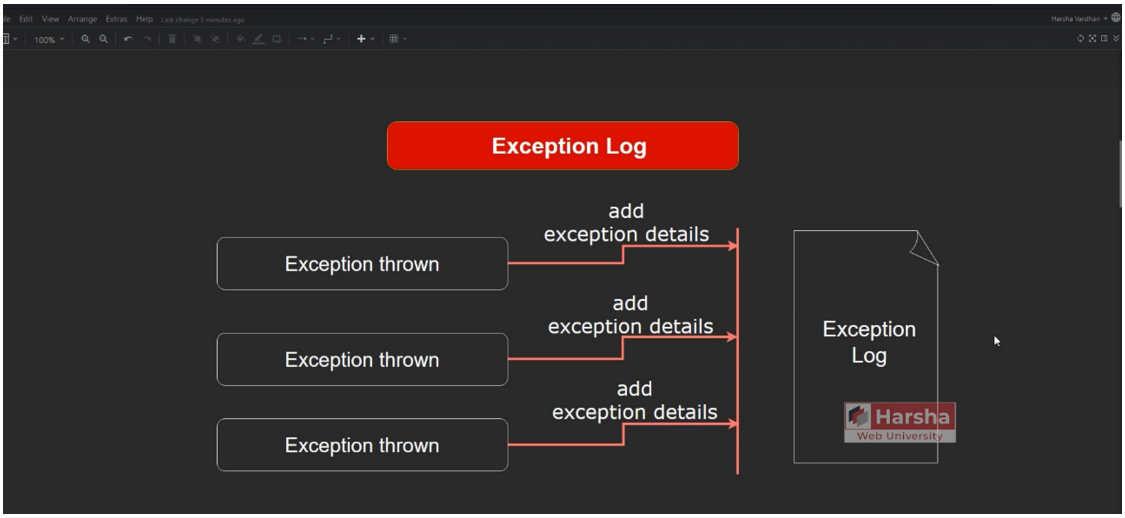
so that you can see the methods sequence of calling and then you will identify exactly at
which method the exception was originally raised and you will go to that particular
method and you will try to identify the problem or cause of that actual error






in general no developer wants to provide the exception details or method names are class names to the end users. so there should be a mechanism where
only developers can see that exception details but not the normal end users that is why instead of printing that stack trace directly in the console we will try to write them into a log file.

so that only developers can open up that log file but not the end users but how do you lock the errors or exceptions
we will try that in the next lecture.




suppose one of your users complains that your application crashes frequently
and he is getting lot of errors at runtime but as a developer you require more details about that runtime errors right

you require to know when that runtime errors or exceptions are being raised
and at what method and what line number and what is the reason behind it
you require to know all those details but physically you cannot sit with him right
he may be using his application somewhere else

so instead of asking him for more details about what exactly happened means what are the inputs given at the time of running
instead of that you can know the runtime exception details in exception log

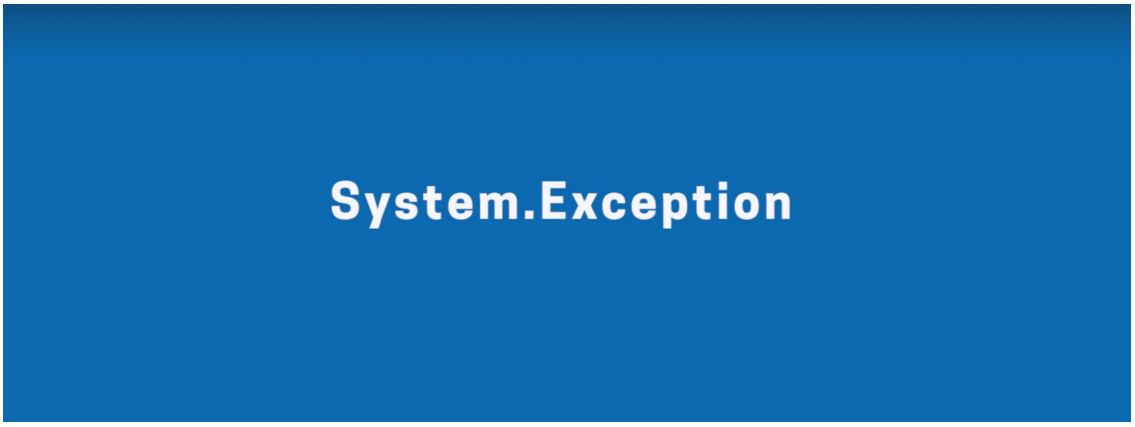
but

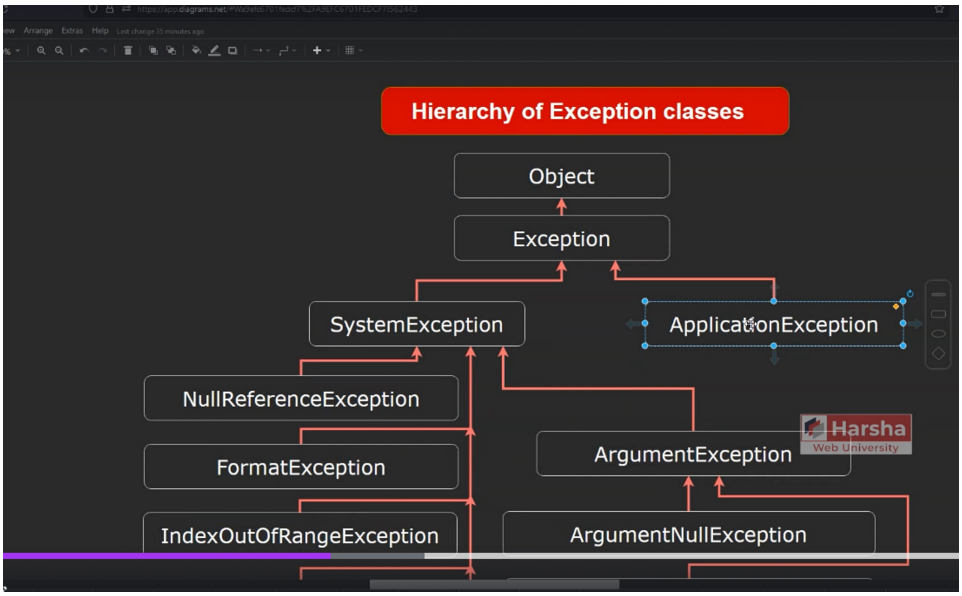
any time when the exception is raised you require to store those exception details in log for example in a text file then later when you want to investigate
about the errors you can open that exception log
and identify the details the complete details about all exceptions raised at different situations or times

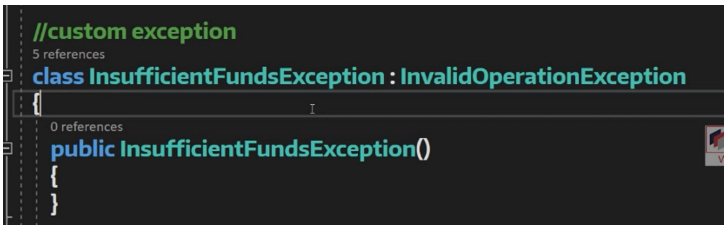
so before that that you require to write essential code whenever any exception is thrown you require to write those exception details in exception log
then only as a developer you can find this exception log in order to know the exception details

which are raised at different situations while running the application

so in this lecture we are going to learn how do you write exception details in
the exception log here the exception log is a normal text file.




This is the hierarchy of the predefined exception classes in .NET
here you can see all the predefined exception classes that we have learned
so far, at first as you know there is a system.object class which is the ultimate parent class of all types in dotnet that means it is the parent of all the
classes structures enumerations and interfaces in .NET

so there is a predefined class called system.exception; it is the parent of all exception classes in dot net. for predefined exception classes and also for user defined exception classes.
so the predefined properties such as message stack trace and inner exception
these properties are already exist in the system.exception class
and under this you have two predefined classes that is system exception and application exception

both are predefined classes actually the usage of application exception is not recommended in these days. initially at the beginning of dot net framework development the microsoft thought that that developers will create custom exception classes based on this application exception

that means if you are creating any custom exception class you will inherit from this application exception; which indirectly indicates that

that exception is the application specific exception for example previously we have created insufficient Fund exception right

so you are supposed to inherit from application exception
like this

but later microsoft changed the mind now according to the best practices of

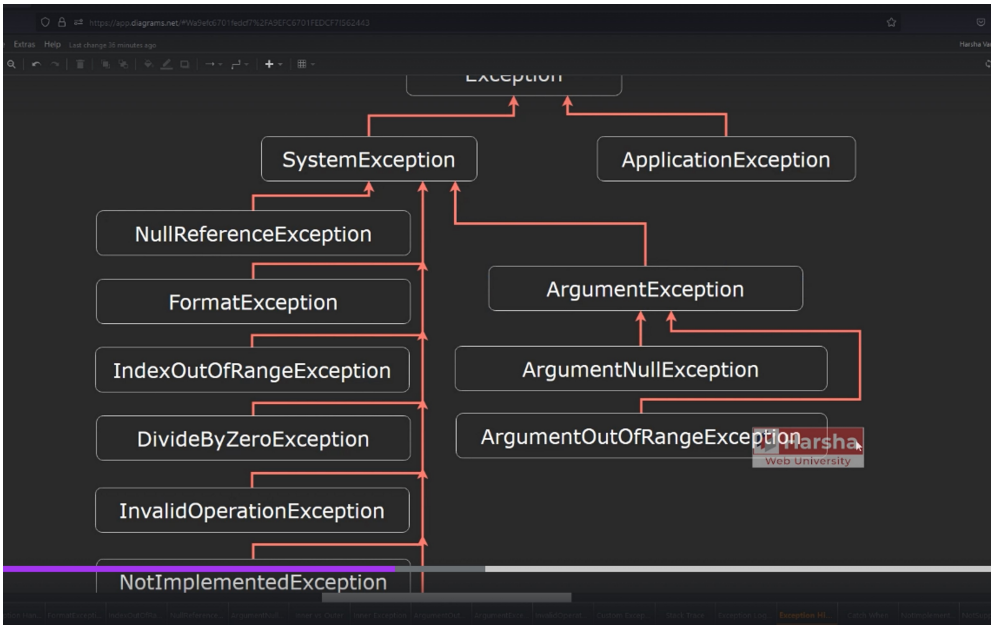
dot net, they are not suggesting this application exception

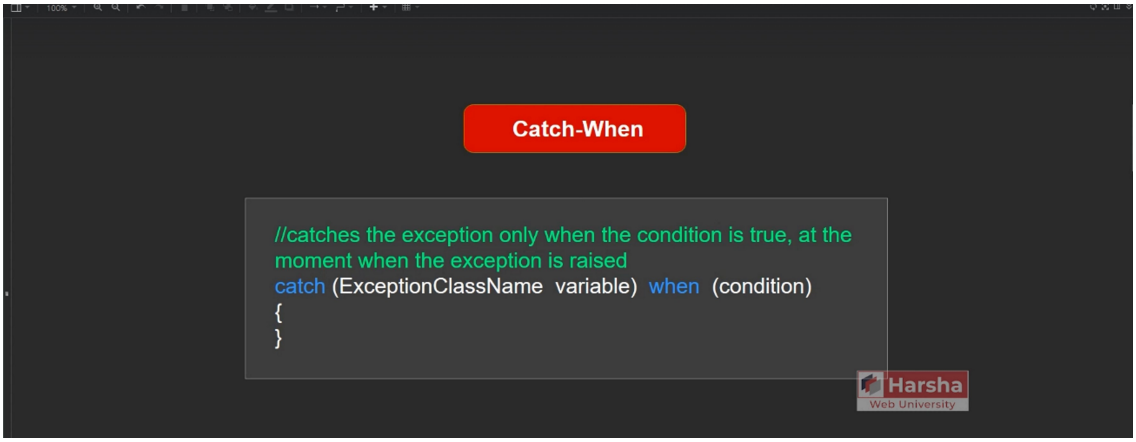
so they did not remove the same but they are clearly saying that don't
use it because the reason is that instead of inheriting from application
exception inheriting from the predefined

exceptions such as argument exception or invalid operation exception
or null reference exception. these are more meaningful
more informative and you can better relate to this exception
for example by inheriting from invalid operation exception into this class

you can indirectly saying that yes it is an invalid operation
because you are trying to transfer the funds at a wrong situation or wrong object
where there are no sufficient funds. so this is more meaningful for the developers

that is why inheriting from application exception is not recommended by microsoft
so you never use it ignore it




You can write a condition while creating the catch block
so that when that condition is true then only this particular catch block will execute otherwise not