Exception Handling

PRATIAN
TECHNOLOGIES

PRATIAN TECHNOLOGIES

Objectives

- Understand what are errors?
- Understand why we use Exceptions?
- Difference between structured and non-structured exception handling
- What are Exception Objects
- How to use try, catch and finally blocks
- Multiple catch Blocks
- What are the best practices about exception handling?

Errors

Errors are part of any application, that we make knowingly or unknowingly. There are many types of errors that occur in our program, such as

- Compile-time error:
  - **Syntax errors:** Design-time errors.
- Runtime-error:
  - **Logical errors:** Occurs during run-time.
  - **System errors:** Occurs when the program is compiled and run.

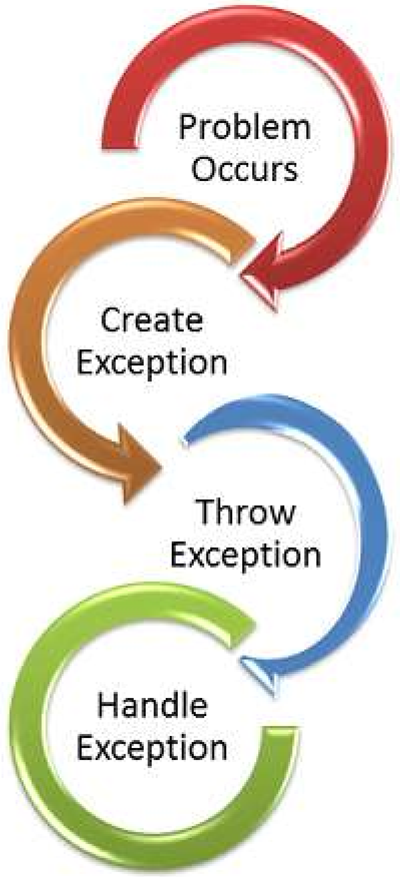
What is an Exception?

- **<u>Exception is not an error</u>**. Rather, <u>Exception is an abnormal condition that arises due to system error while executing a program</u>

- Effective exception handling will make your programs more robust and easier to debug. They help answer these three questions:
  - **What went wrong?**
    - Answered by the type of exception thrown.
  - **Where did it go wrong?**
    - Answered by exception stack trace.
  - **Why did it go wrong?**
    - Answered by exception message

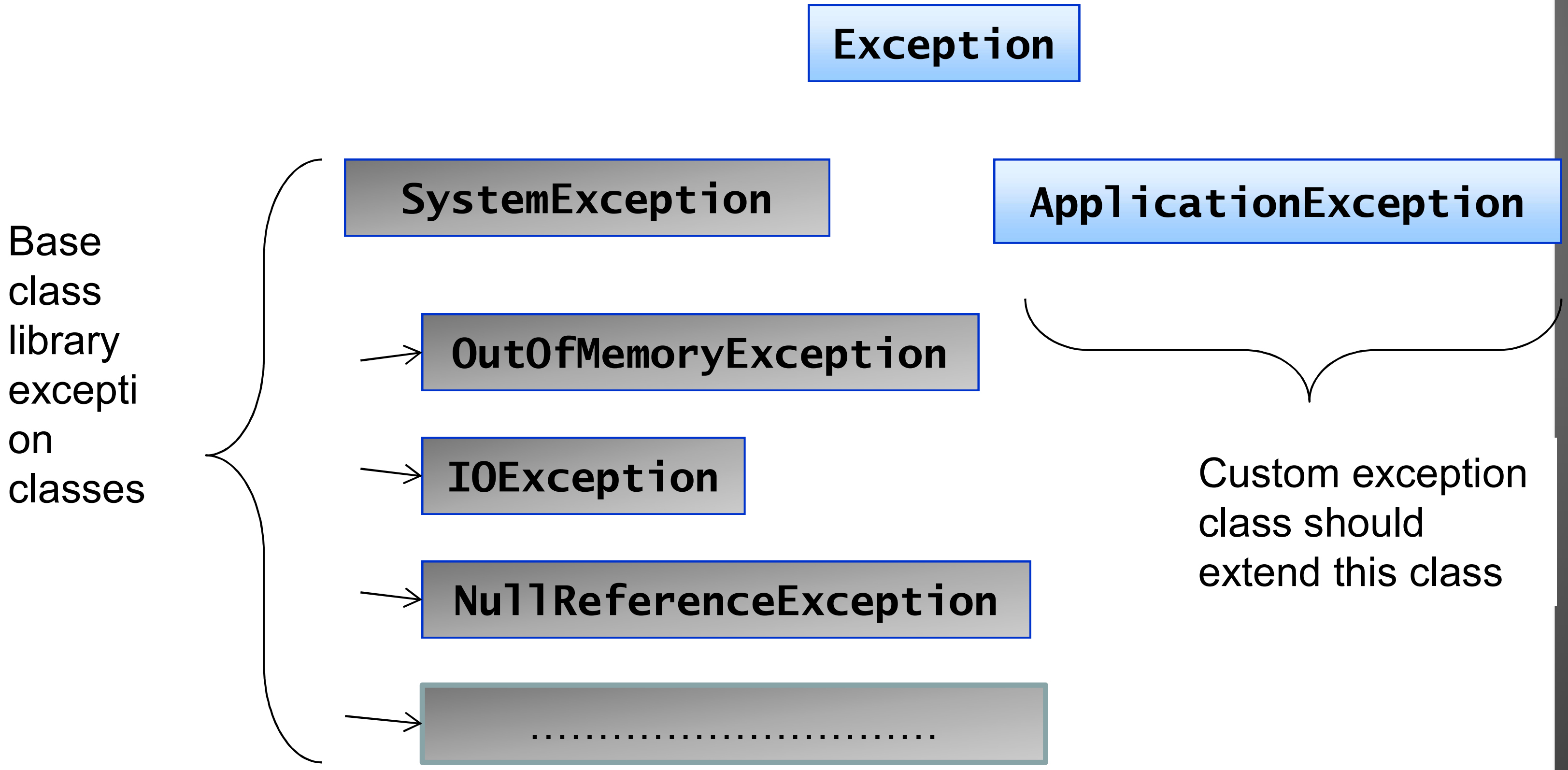
How an Exception is handled?

- Exception happens due to system's failure to execute code because of an abnormal condition
  - For example, when you write a code to divide zero or some other value by zero and then try to execute, system will not be able to perform that job
- CLR then creates an object to represent the unexpected error and then throws it to the method which caused it
  - That method may choose to handle the exception itself or pass it on
- Either way, at some point, the exception is caught and processed
- Sources for exceptions could be
  - Generated by CLR
  - Manually generated by programmer's code.



Exception Hierarchy

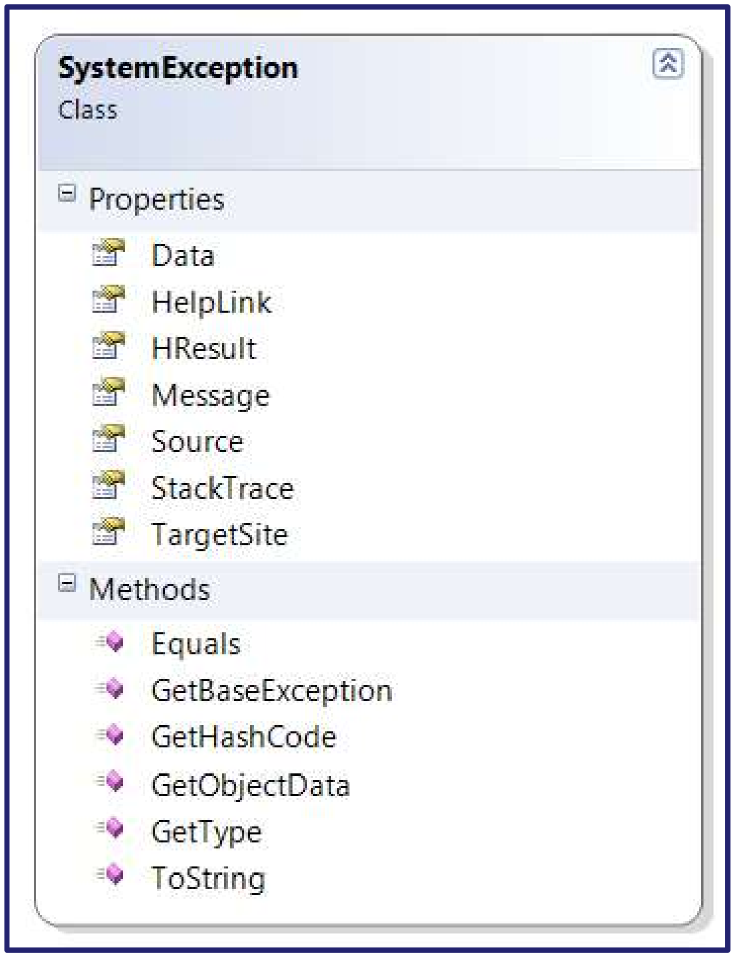
The base class for all the exception related classes is Exception. Two child classes that inherit from it are SystemException and ApplicationException class.



PRATIAN TECHNOLOGIES

System.SystemException class

•Defines the base class for predefined exceptions in the System namespace.
•This class is provided as a means to differentiate between exceptions defined by the system versus exceptions defined by applications


SystemException
Class

Properties
- Data
- HelpLink
- HResult
- Message
- Source
- StackTrace
- TargetSite

Methods
- Equals
- GetBaseException
- GetHashCode
- GetObjectData
- GetType
- ToString

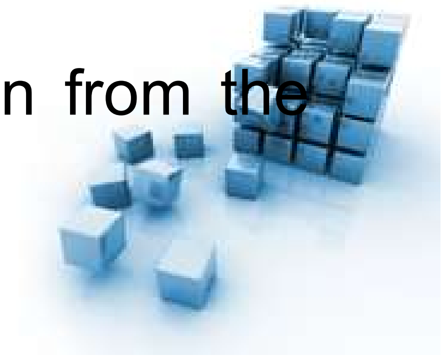
Important Methods and Properties of Exception class

- **Message property**: This property gets a message that describes the current exception. It returns the error message that explains the reason for the exception, or an empty string("").
- **Source Property**: This property gets or sets the name of the application or the object that causes the error. It returns the name of the application or the object that causes the error.
- **TargetSite Property**: Gets the method that throws the current exception. It returns the instance of MethodBase class, present in System.Reflection namespace, representing information of the method that threw the current exception
- **StackTrace Property**: it gets a string representation of the frames on the call stack at the time the current exception was thrown. It returns a string that describes the contents of the call stack, with the most recent method call appearing first.

Exception handling constructs

Four constructs are used in exception handling:

- try – a block surrounding program statements to monitor for exceptions
- catch – together with try, catches specific kinds of exceptions and handles them in some way
- finally – specifies any code that absolutely must be executed whether or not an exception occurs
- throw – used to throw a specific exception from the program



The try, catch and finally block

```
try {
   /*
    * some codes to test here
    */
} catch (SqlException sx) {
   /*
    * handle Exception1 here
    */
} catch (FileNotFoundException
   fx) {
   /*
    * handle Exception2 here
    */
} catch (Exception ex) {
   /*
    * handle Exception3 here
    */
} finally {
   /*
    * always execute codes here
    */
}
```

`try` block encloses the context where a possible exception can be thrown

each `catch()` block is an exception handler and can appear several times

An optional `finally` block is always executed before exiting the `try` statement.

Using try and catch Blocks

- Object-oriented solution to error handling
  - Put the normal code in a **try** block
  - Handle the exceptions in a separate **catch** block

```
try
{
          int x=0; int y=0; int z;
          z = x/y;
}
catch (DivideByzeroException caught)
{
          Console.WriteLine(caught);
}
```

Program logic

Error handling

Multiple catch Blocks

- Each catch block catches one class of exception
- A try block can have one general catch block

```
try
{
            Console.WriteLine("Enter first number");
            int i = int.Parse(Console.ReadLine());
            Console.WriteLine("Enter second number");
            int j = int.Parse(Console.ReadLine());
            int k = i / j;
}
catch (OverflowException caught) {…}
catch (DivideByZeroException caught) {…}
```

An important note about multiple catch block

- A catch block which catches all exceptions (catch block accepting Exception class object) should be placed as the last one if you are using multiple catch blocks.

```
try
{
    Console.WriteLine("Enter first number");
    int i = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter second
number");
    int j = int.Parse(Console.ReadLine());
    int k = i / j;

}
catch (OverflowException caught) {…}
catch (DivideByZeroException caught) {…}
catch (Exception caught) {…}
```

Correct approach

```
try
{
    Console.WriteLine("Enter first number");
    int i = int.Parse(Console.ReadLine());
    Console.WriteLine("Enter second
number");
    int j = int.Parse(Console.ReadLine());
    int k = i / j;

}
catch (Exception caught) {…}
catch (DivideByZeroException caught) {…}
catch (OverFlowException caught) {…}
```

Wrong approach

The finally Clause

- All of the statements in a finally block are always executed

```
Monitor.Enter(x);
try
{
    ...
}
catch(Exception ex)
{
//code
}
finally
{
    Monitor.Exit(x);
}
```

Any catch blocks are optional. Try can be followed by either catch or finally

Finally blocks are mainly used to clean up resources, such as if you have opened database connection or file connection in try block then close them in finally block, because due to some exception if the following codes are not executed, they are bound to get executed at least in finally block

PRATIAN TECHNOLOGIES

Why Use Exception?

Exception Handling: Traditional approach

- Method returns error code.
  - Problem: Forget to check for error code
    - Failure notification may go undetected
- Problem: Calling method may not be able to do anything about failure
  - Program must fail too and let its caller worry about it
  - Many method calls would need to be checked
- Instead of programming for success object.doSomething() you would always be programming for failure:

```
        if (!object.doSomething())
                return false;
```

Why Use Exceptions?

Traditional Approach vs. Structured Exception Handling

Traditional procedural error handling is cumbersome. Actual code is not separate from exception code.

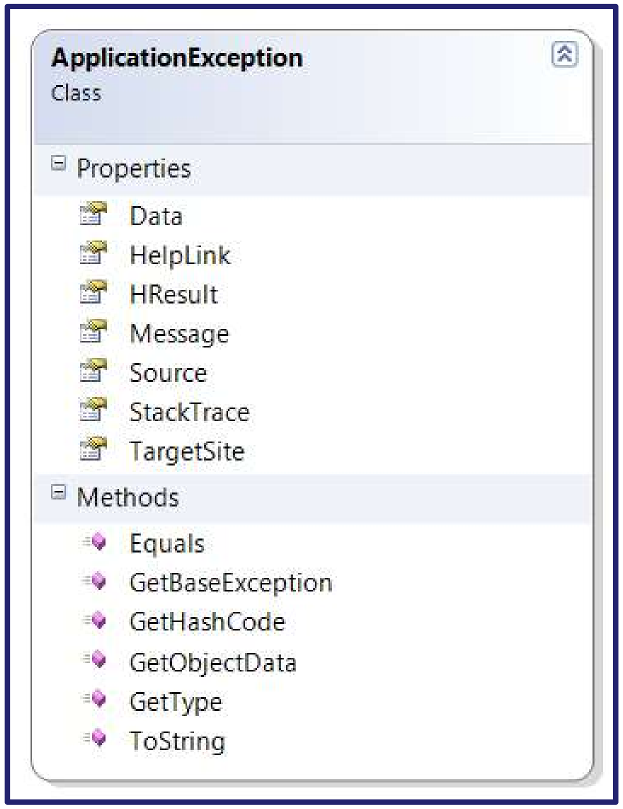
```
namespace ExceptionHandlingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
                int errorCode = 0;
                //programming logic
                FileInfo source = new FileInfo("code.cs");
               //error detection
                if (errorCode == -1)
                    goto Failed;
                //programming logic
                int length = (int)source.Length;
                //error detection
                if (errorCode == -2)
                    goto Failed;
                //programming logic
                char[] contents = new char[length];
                //error detection
                if (errorCode == -3)
                    goto Failed;
               //handling error
                Failed:
                    Console.WriteLine("failure..");
        }
    }
}
```

Structured Exception Handling makes it easy to separate exception code from actual code

```
namespace ExceptionHandlingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                FileInfo source = new FileInfo("code.cs");
                int length = (int)source.Length;
                char[] contents = new char[length];
            }
            catch (IOException ex)
            {
                Console.WriteLine(ex.Message);
            }
        }
    }
}
```

System.ApplicationException class

•The exception that is thrown when a non-fatal application error occurs.
•User applications, not the common language runtime, throw custom exceptions derived from the ApplicationException class.
•The ApplicationException class differentiates between exceptions defined by applications versus exceptions defined by the system.

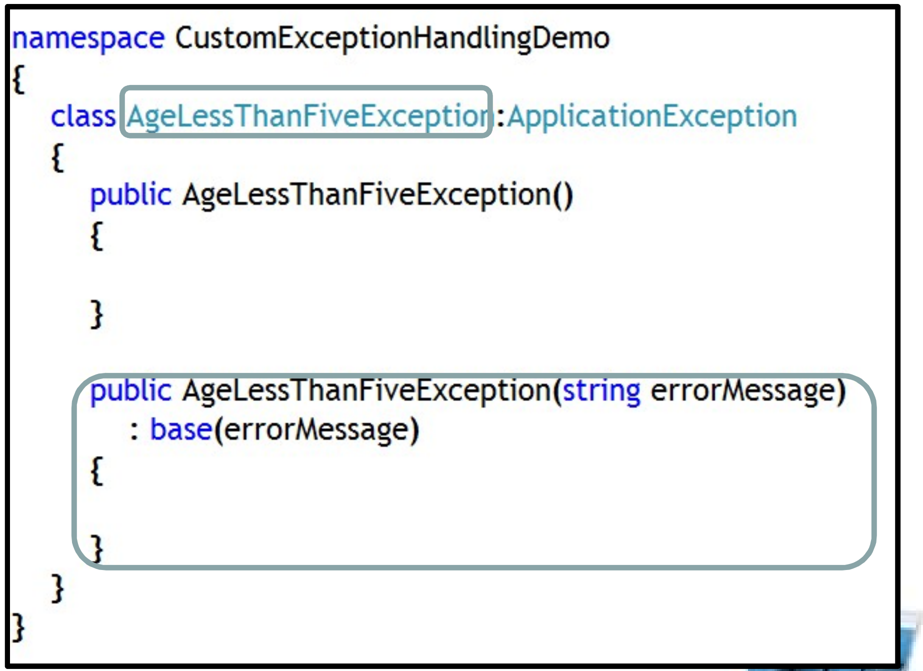


Custom Exception Class

- Exception class can be created by user.
- It is needed whenever you need to tackle a situation for which there is no system exception available
  - Example:
    - Insufficient Balance
    - Invalid Age
    - Invalid Card Number

How to create custom exception class?

- Create a custom exception class by extending from ApplicationException class, which inherits from base class Exception.
- Provide user-defined (overloaded constructors) which will accept error message as string data type and pass to base class using base keyword

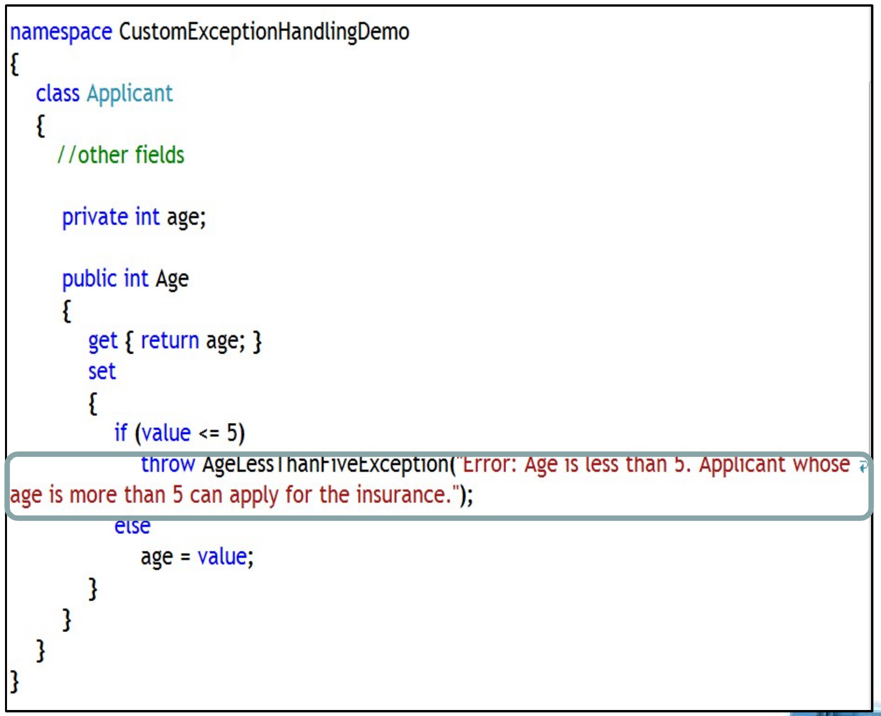


```
namespace CustomExceptionHandlingDemo
{
   class AgeLessThanFiveException:ApplicationException
   {
      public AgeLessThanFiveException()
      {

      }

      public AgeLessThanFiveException(string errorMessage)
         : base(errorMessage)
      {

      }
   }
}
```

How to throw custom exception?

User has to throw custom exception, since runtime is unaware about custom exception class

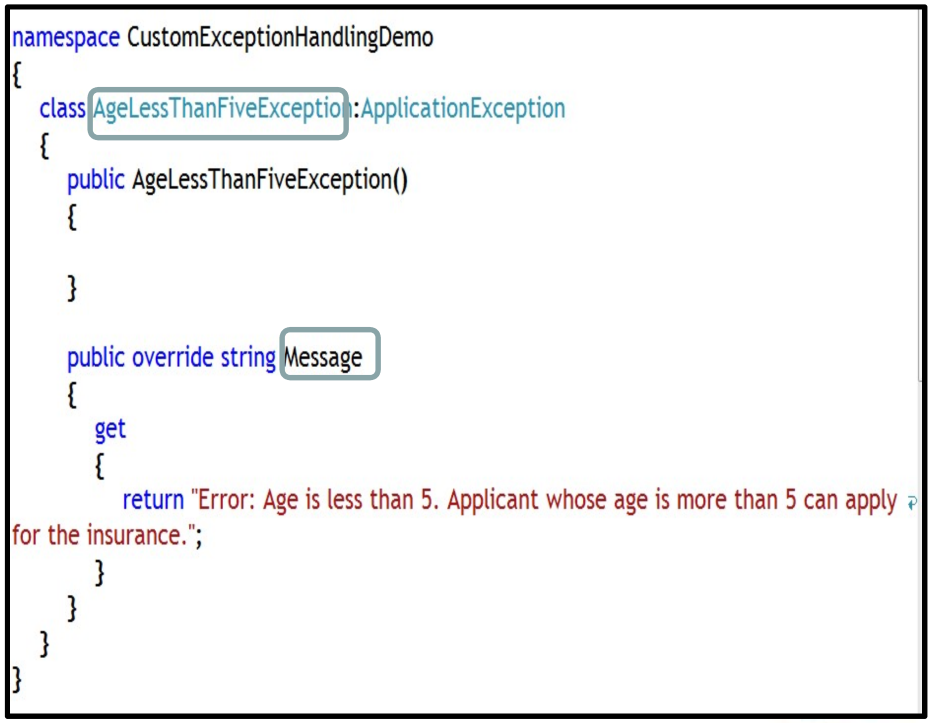
- Create an object of custom exception class wherever necessary.
- Use 'throw' keyword to throw the exception object



```
namespace CustomExceptionHandlingDemo
{
   class Applicant
   {
     //other fields

      private int age;

      public int Age
      {
         get { return age; }
         set
         {
            if (value <= 5)
               throw AgeLessThanFiveException("Error: Age is less than 5. Applicant whose
age is more than 5 can apply for the insurance.");
            else
               age = value;
         }
      }
   }
}
```

How to create custom exception class?

- Create a <u>custom exception class</u> by extending from ApplicationException class, which inherits from base class Exception.
- Override virtual, read-only 'Message' property from base class and return custom message from that property

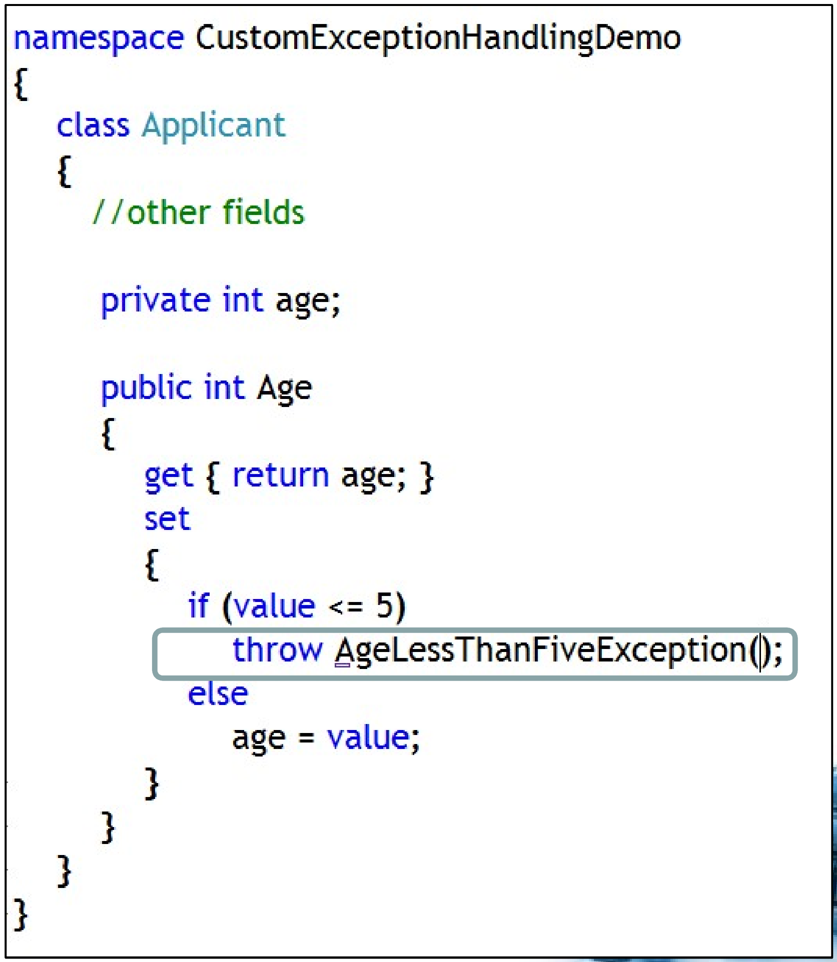


```
namespace CustomExceptionHandlingDemo
{
    class AgeLessThanFiveException:ApplicationException
    {
        public AgeLessThanFiveException()
        {

        }

        public override string Message
        {
            get
            {
                return "Error: Age is less than 5. Applicant whose age is more than 5 can apply 
for the insurance.";
            }
        }
    }
}
```

How to throw custom exception?

User has to throw custom exception, since runtime is unaware about custom exception class

- Create an object of custom exception class where ever necessary.
- Use ‘throw’ keyword to throw the exception object

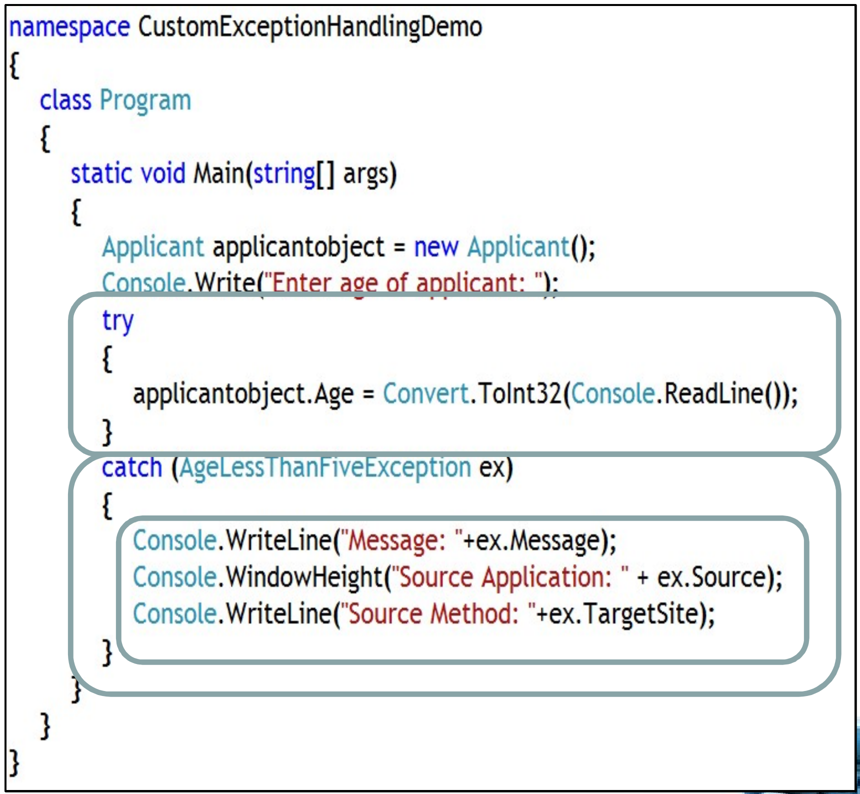


```
namespace CustomExceptionHandlingDemo
{
    class Applicant
    {
      //other fields

        private int age;

        public int Age
        {
            get { return age; }
            set
            {
                if (value <= 5)
                    throw AgeLessThanFiveException();
                else
                    age = value;
            }
        }
    }
}
```

How to catch custom exception?

Catching custom exception is in no way different from catching any system exception

- Put the suspected code in try block
- Use catch block with custom exception class variable to catch the custom exception
- Display necessary information



```
namespace CustomExceptionHandlingDemo
{
    class Program
    {
        static void Main(string[] args)
        {
            Applicant applicantobject = new Applicant();
            Console.Write("Enter age of applicant: ");
            try
            {
                applicantobject.Age = Convert.ToInt32(Console.ReadLine());
            }
            catch (AgeLessThanFiveException ex)
            {
                Console.WriteLine("Message: "+ex.Message);
                Console.WindowHeight("Source Application: " + ex.Source);
                Console.WriteLine("Source Method: "+ex.TargetSite);
            }
        }
    }
}
```

The throw Statement

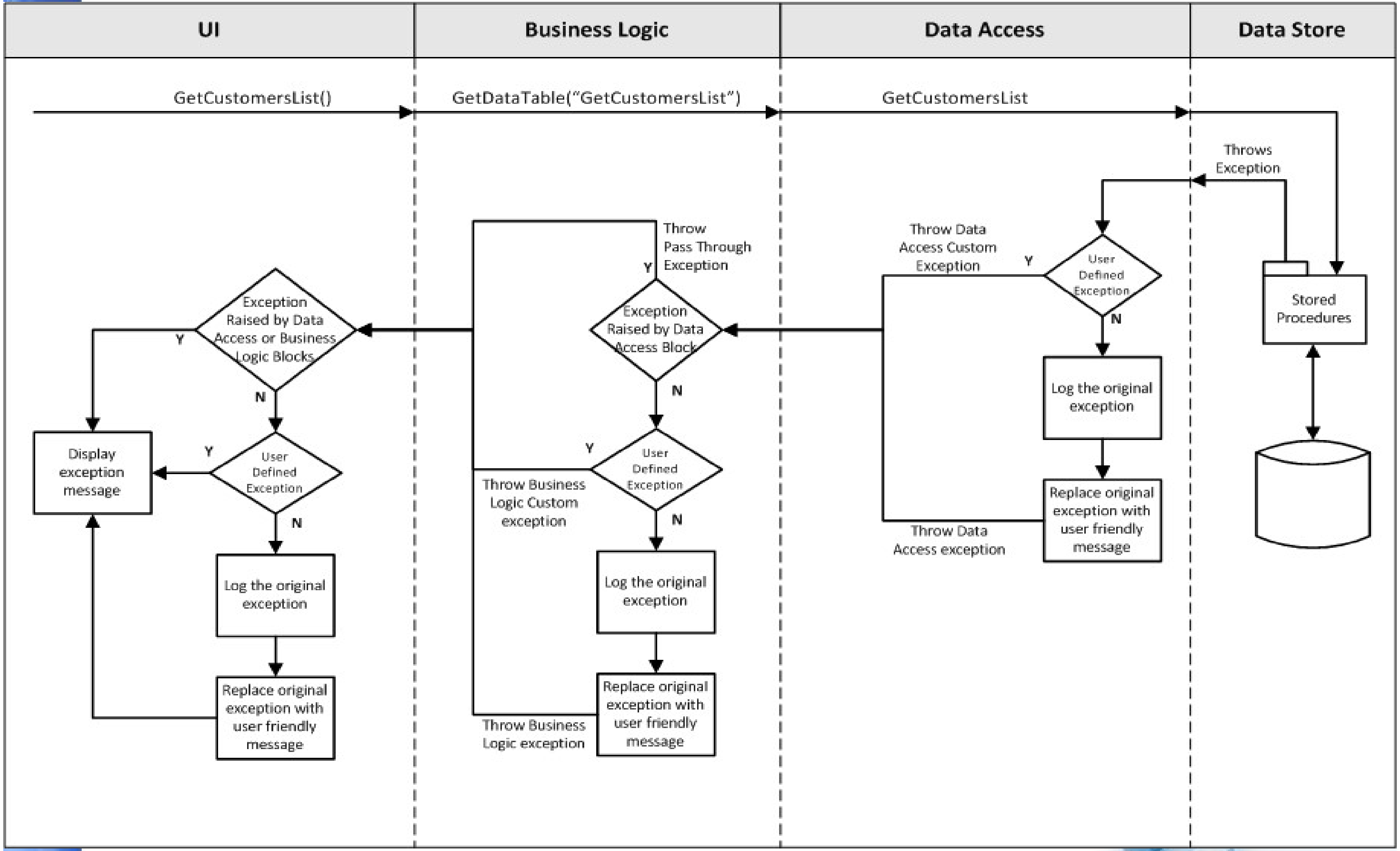
- Use ‘throw’ statement to throw an appropriate exception
- Generally used to throw custom exceptions
- Give the exception a meaningful message

```
throw expression ;
```

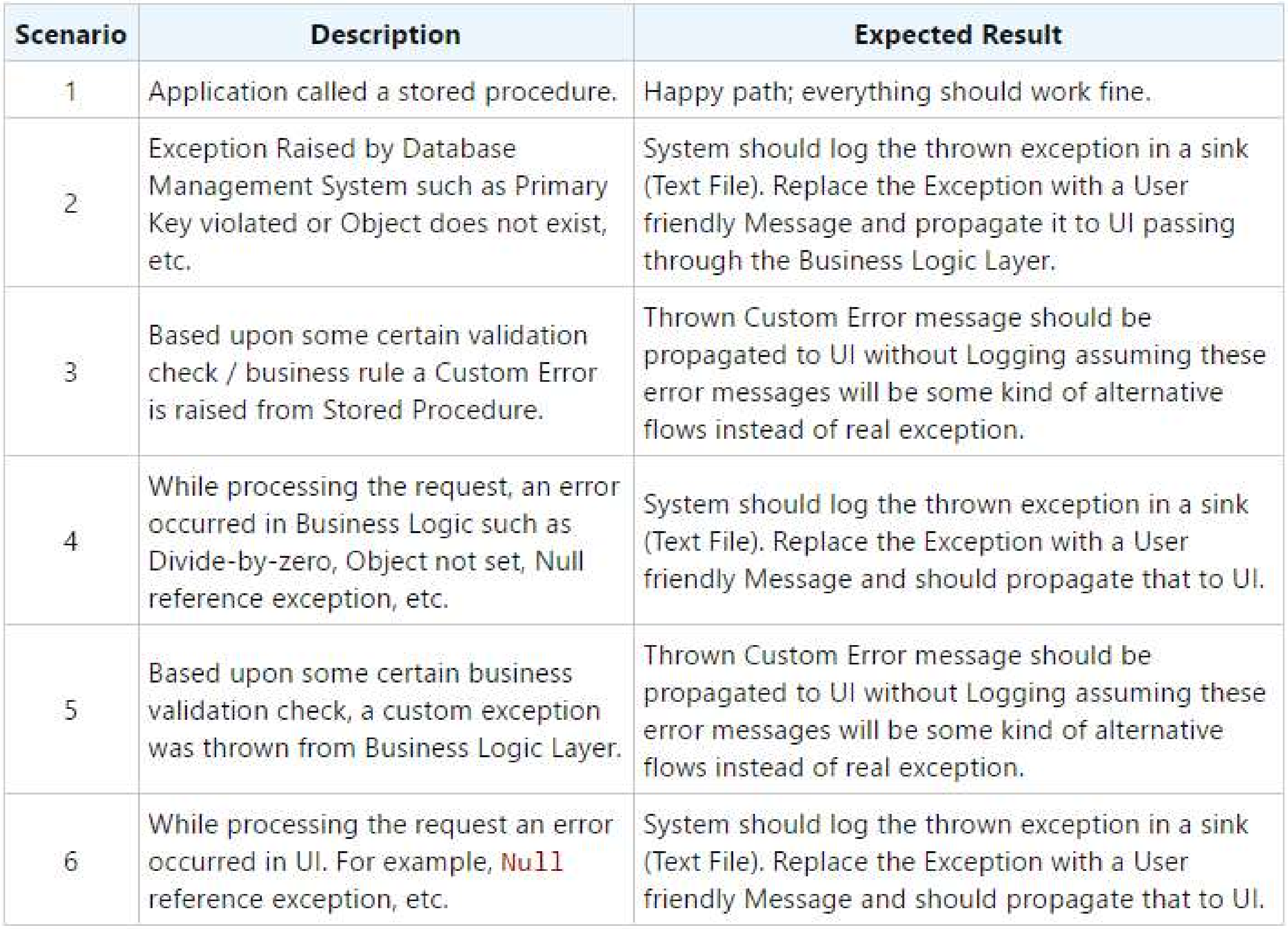
Custom Exception class

```
if (minute < 1 || minute >= 60) {
  throw new InvalidTimeException(minute +
                           " is not a valid minute“);
  // !! Not reached !!
}
```

Exception Handling in 3-Tier Architecture


UI
Business Logic
Data Access
Data Store
GetCustomersList()
GetDataTable("GetCustomersList")
GetCustomersList
Throws Exception
Stored Procedures
User Defined Exception
Y
N
Throw Data Access Custom Exception
Log the original exception
Replace original exception with user friendly message
Throw Data Access exception
Throw Pass Through Exception
Exception Raised by Data Access Block
Throw Business Logic Custom exception
Throw Business Logic exception
Exception Raised by Data Access or Business Logic Blocks
Display exception message

Exception Handling in 3-Tier Architecture



Scenario	Description	Expected Result
1	Application called a stored procedure.	Happy path; everything should work fine.
2	Exception Raised by Database Management System such as Primary Key violated or Object does not exist, etc.	System should log the thrown exception in a sink (Text File). Replace the Exception with a User friendly Message and propagate it to UI passing through the Business Logic Layer.
3	Based upon some certain validation check / business rule a Custom Error is raised from Stored Procedure.	Thrown Custom Error message should be propagated to UI without Logging assuming these error messages will be some kind of alternative flows instead of real exception.
4	While processing the request, an error occurred in Business Logic such as Divide-by-zero, Object not set, Null reference exception, etc.	System should log the thrown exception in a sink (Text File). Replace the Exception with a User friendly Message and should propagate that to UI.
5	Based upon some certain business validation check, a custom exception was thrown from Business Logic Layer.	Thrown Custom Error message should be propagated to UI without Logging assuming these error messages will be some kind of alternative flows instead of real exception.
6	While processing the request an error occurred in UI. For example, `Null` reference exception, etc.	System should log the thrown exception in a sink (Text File). Replace the Exception with a User friendly Message and should propagate that to UI.

Exception Handling in 3-Tier Architecture

- Exception should be logged into the Persistence Storage.
- It should be logged on the origin location (from where it is caught).
- Once an exception is caught:
  1. It should be wrapped into a Custom Exception class and then thrown further up the calling layers.
  2. Exception details should be logged into the database only once.
  3. If the exception is thrown from some other layer, then it should be forwarded to a further layer without logging to the database (since it's already logged in the origin).
  4. If it's a UI layer then a user friendly message should be displayed to the user.

QUIZ

Exception Handling

1. Which among the following is NOT considered as .NET Exception class?

a) Exception

[✓ b) StackUnderflow Exception]

[✓ c) File Found Exception]

d) Divide By zero Exception

Exception Handling

2. Select the statements which describe the correct usage of exception handling over conventional error handling approaches?

✓ a) As errors can be ignored but exceptions cannot be ignored

✓ b) Exception handling allows separation of program’s logic from error handling logic making software more reliable and maintainable

✓ c) try – catch – finally structure allows guaranteed clean-up in event of errors under all circumstances

d) All of the above mentioned

Exception Handling

3. Which of these keywords is not a part of exception handling?

a) try

b) finally

✓ c) thrown

d) catch

Exception Handling

4. Which of the keyword is used to manually throw an exception?

a) try

b) finally

[✓ c) throw]

d) catch

Exception Handling

5. Which of the following is the correct statement about exception handling in C#.NET?

[✓ a) finally clause is used to perform cleanup operations of closing network and database connections]

b) a program can contain multiple finally clauses

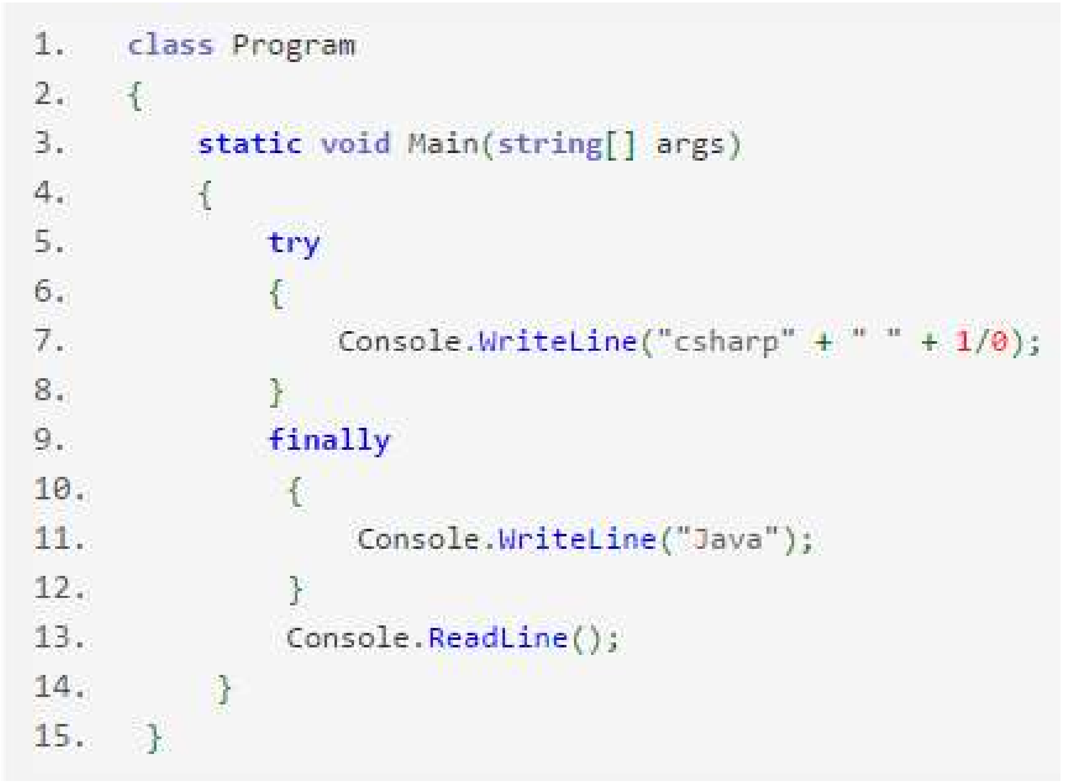
[✓ c) The statement in finally clause will get executed no matter whether an exception occurs or not]

d) All of the above mentioned

PRATIAN TECHNOLOGIES

Exception Handling

6. Choose the correct output for given set of code:



```
class Program
{
    static void Main(string[] args)
    {
        try
        {
            Console.WriteLine("csharp" + " " + 1/0);
        }
        finally
         {
             Console.WriteLine("Java");
         }
         Console.ReadLine();
     }
 }
```

a) csharp 0

[✓ b) Run time Exception generation]

c) Compile time error

d) Java

Exception Handling

7. Choose the correct statement which makes exception handling work in C#.NET?

✓ a) .Net runtime makes search for the exception handler where exception occurs

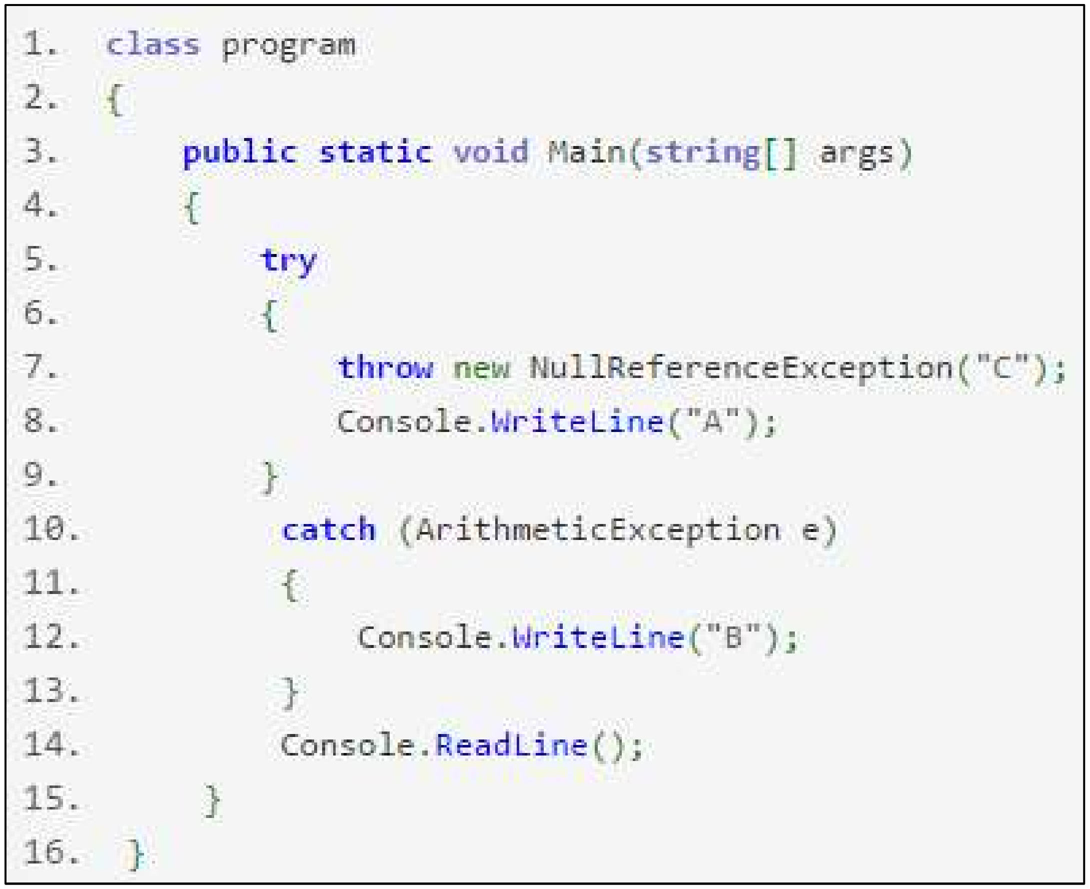
✓ b) If no exception is matched, exception handler goes up the stack and hence finds the match there

✓ c) If no match is found at the highest level of stack call, then unhandledException is generated and hence termination of program occurs

d) None of the mentioned

Exception Handling

8. What will be the output of given code snippet?



```
class program
{
    public static void Main(string[] args)
    {
        try
        {
            throw new NullReferenceException("C");
            Console.WriteLine("A");
        }
         catch (ArithmeticException e)
         {
             Console.WriteLine("B");
         }
         Console.ReadLine();
    }
}
```

a) A

b) B

c) Compile time error

d) Runtime error ✓