Advance Programming Techniques (APT)

Lecture # 10

Ehtisham Rasheed

Department of Computer Science
University of Gurjat, Gujrat

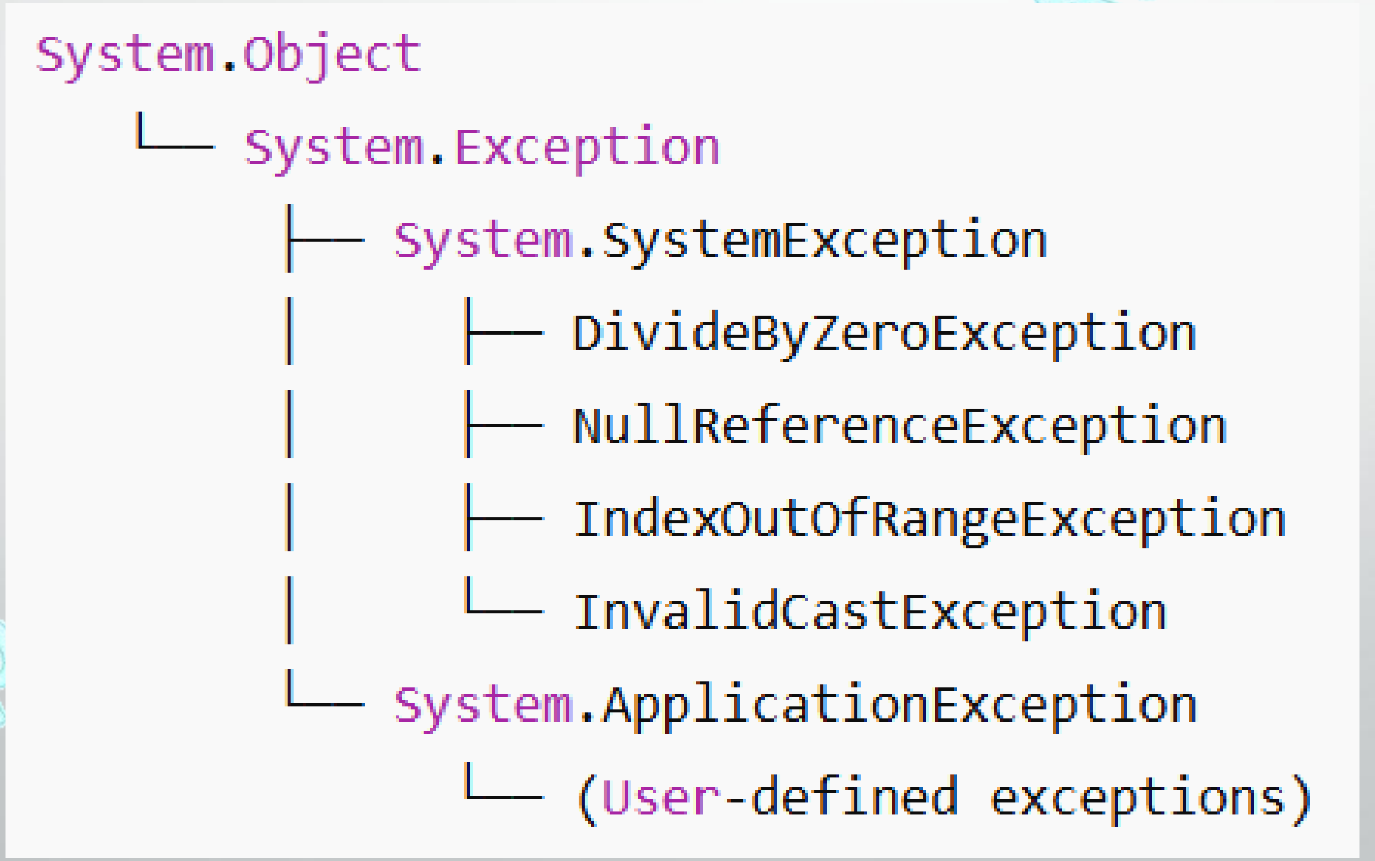
UNIVERSITY OF GUJRAT

Exceptions

- An exception is an **unexpected event** during program execution
- Exception **disturb** the normal flow of instructions
- In C#, all exceptions are **objects** derived from the base class `System.SystemException`
- Examples
  - Dividing a number by zero
  - Accessing an invalid array index
  - Opening a file that doesn’t exist
  - Database connection failure

Exceptions

- All exceptions in C# derive from the **System.Exception** class

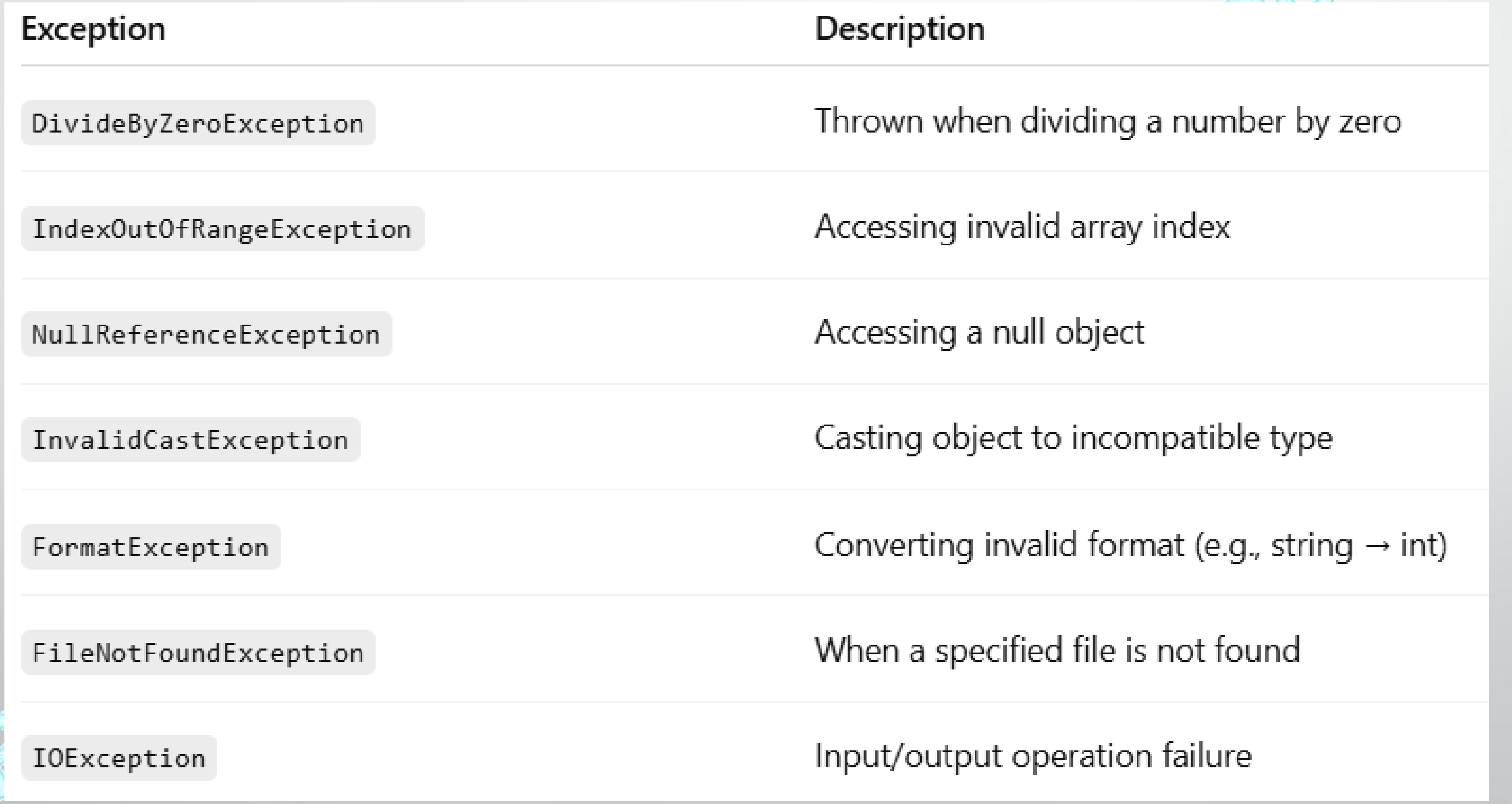


```
System.Object
    └── System.Exception
          ├── System.SystemException
          │     ├── DivideByZeroException
          │     ├── NullReferenceException
          │     ├── IndexOutOfRangeException
          │     └── InvalidCastException
          └── System.ApplicationException
                └── (User-defined exceptions)
```

Types of Exceptions

1. System-defined (built-in) exceptions
   - System exceptions are **built-in exception classes** provided by the **.NET Framework** inside the **System namespace**
   - They represent **common runtime errors** that occur due to incorrect logic or system-level issues
2. User-defined (custom or application) exceptions
   - User-defined exceptions are **custom exception classes** that you create yourself, derived from the System.Exception class.
   - They are used when you want to handle **application-specific errors** — things that the system cannot automatically detect

Common System-Defined Exceptions



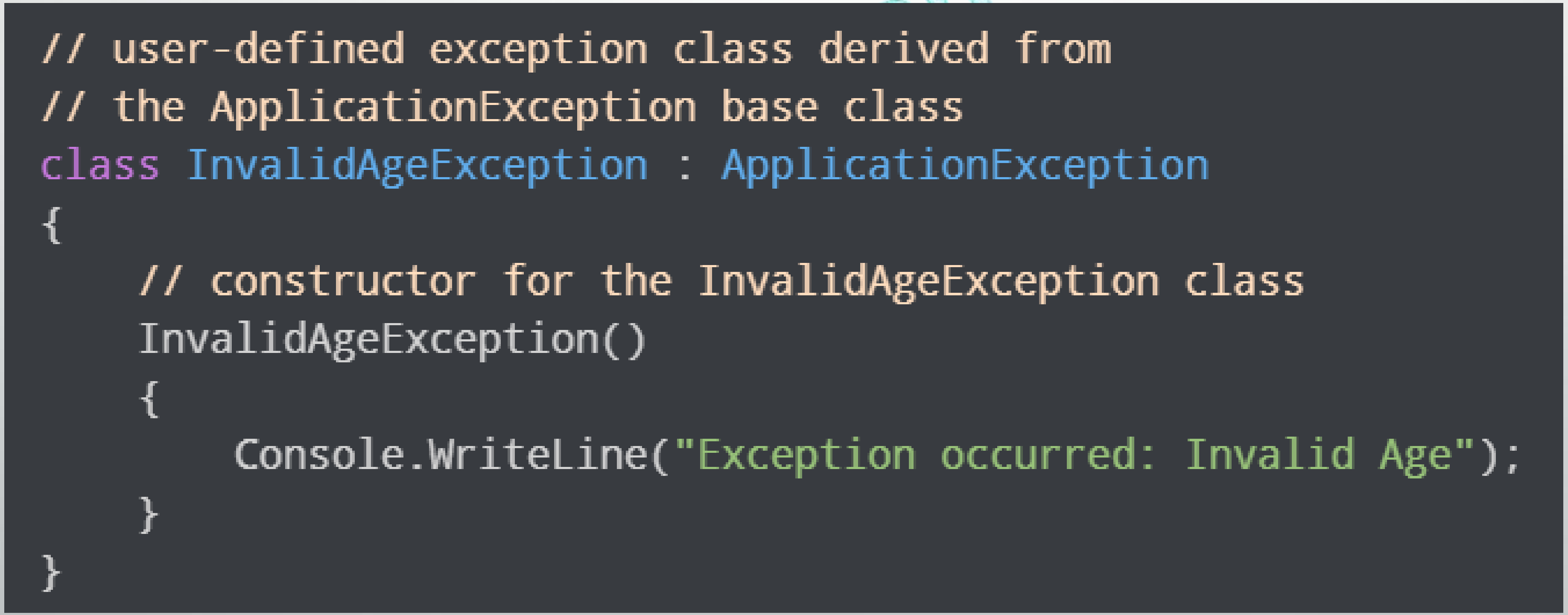
Exception	Description
`DivideByZeroException`	Thrown when dividing a number by zero
`IndexOutOfRangeException`	Accessing invalid array index
`NullReferenceException`	Accessing a null object
`InvalidCastException`	Casting object to incompatible type
`FormatException`	Converting invalid format (e.g., string → int)
`FileNotFoundException`	When a specified file is not found
`IOException`	Input/output operation failure

User-Defined Exceptions

- We create custom exceptions when:
  - We need to enforce business rules (e.g., invalid marks, negative balance, age limit)
  - We want to give a **meaningful message** for custom errors
  - We want to control **how** exceptions are thrown and handled in your application
- Example
  - Suppose we want to create an exception in an application that does not allow the age of a participant to be more than **18**
  - Let's name that exception – InvalidAgeException
  - To create InvalidAgeException exception, we derive it from Exception

Application Exceptions

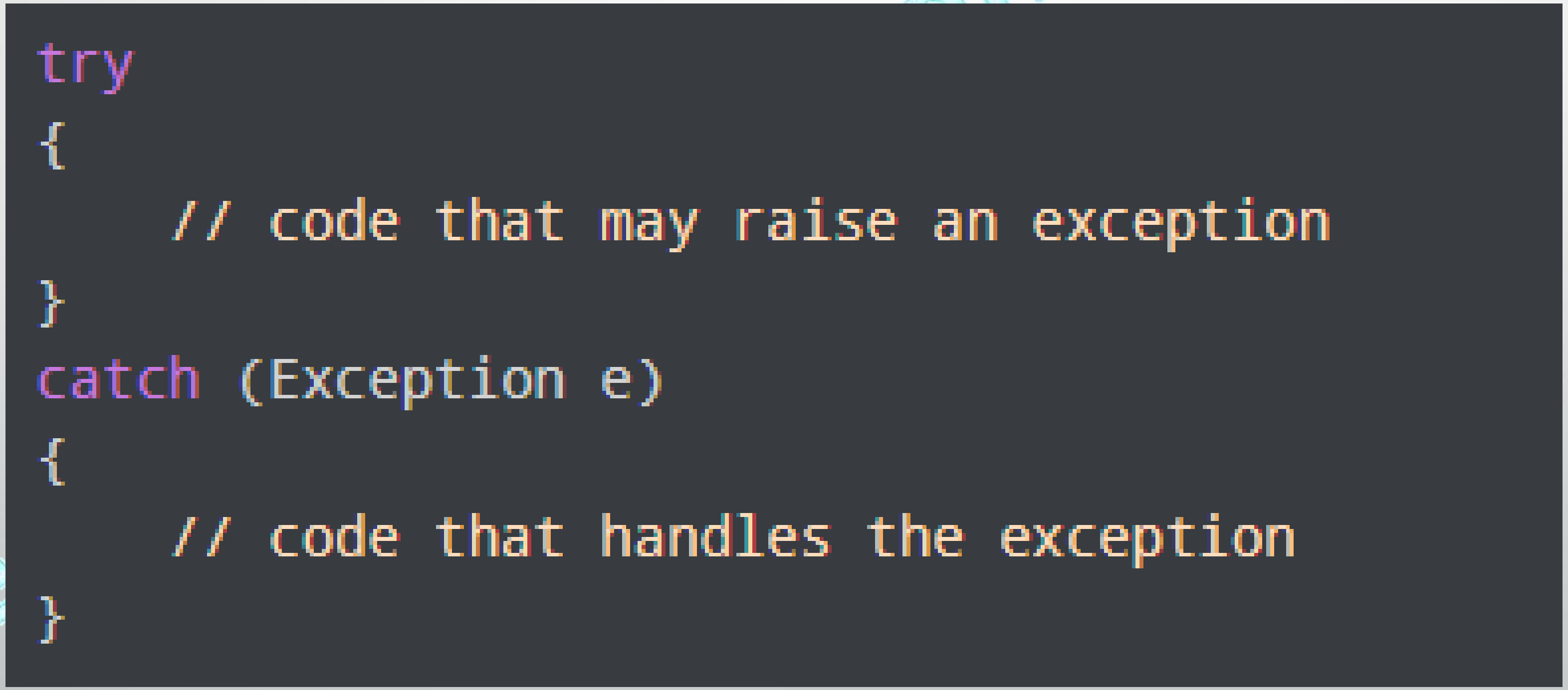
- The throw keyword is used to manually raise an exception
- The modern professional way (recommended by Microsoft) is to inherit from Exception class rather than ApplicationException class



```
// user-defined exception class derived from
// the ApplicationException base class
class InvalidAgeException : ApplicationException
{
    // constructor for the InvalidAgeException class
    InvalidAgeException()
    {
        Console.WriteLine("Exception occurred: Invalid Age");
    }
}
```

Exception Handling

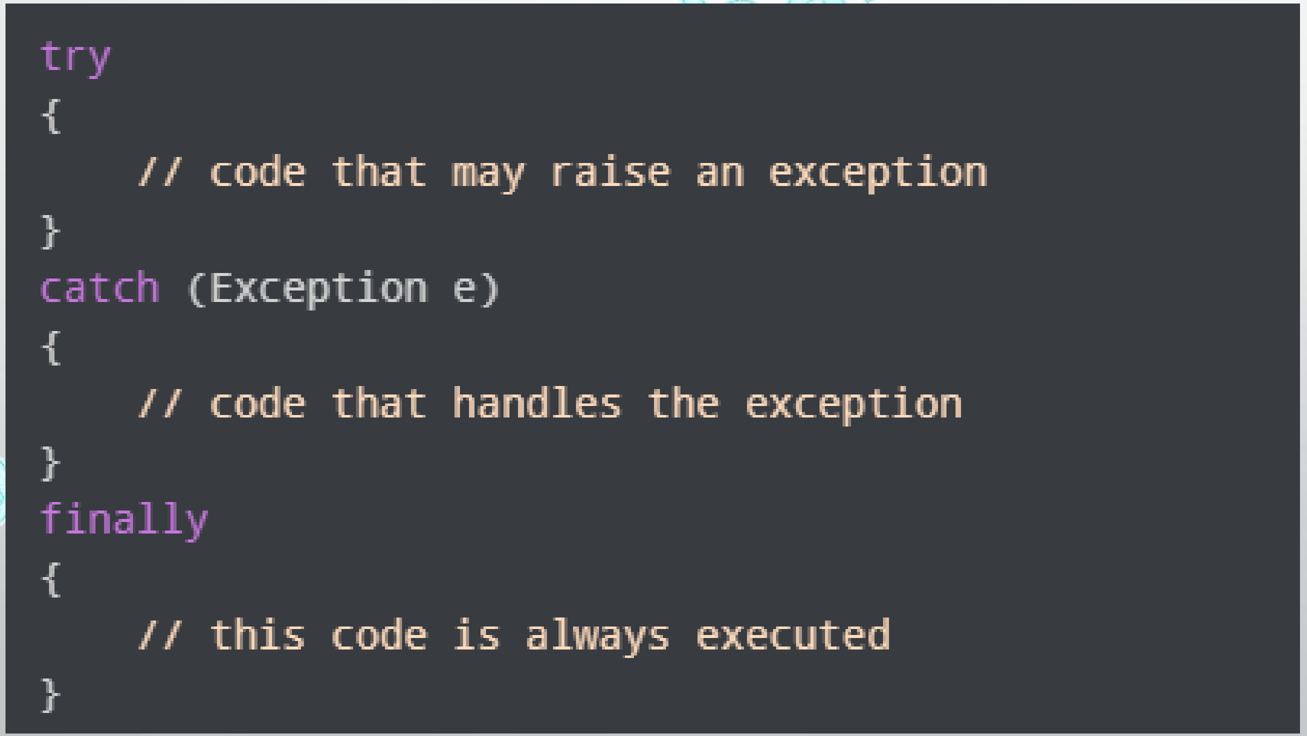
- C# provides built-in blocks to handle exceptions. They are ***try..catch*** and ***finally***



```
try
{
    // code that may raise an exception
}
catch (Exception e)
{
    // code that handles the exception
}
```

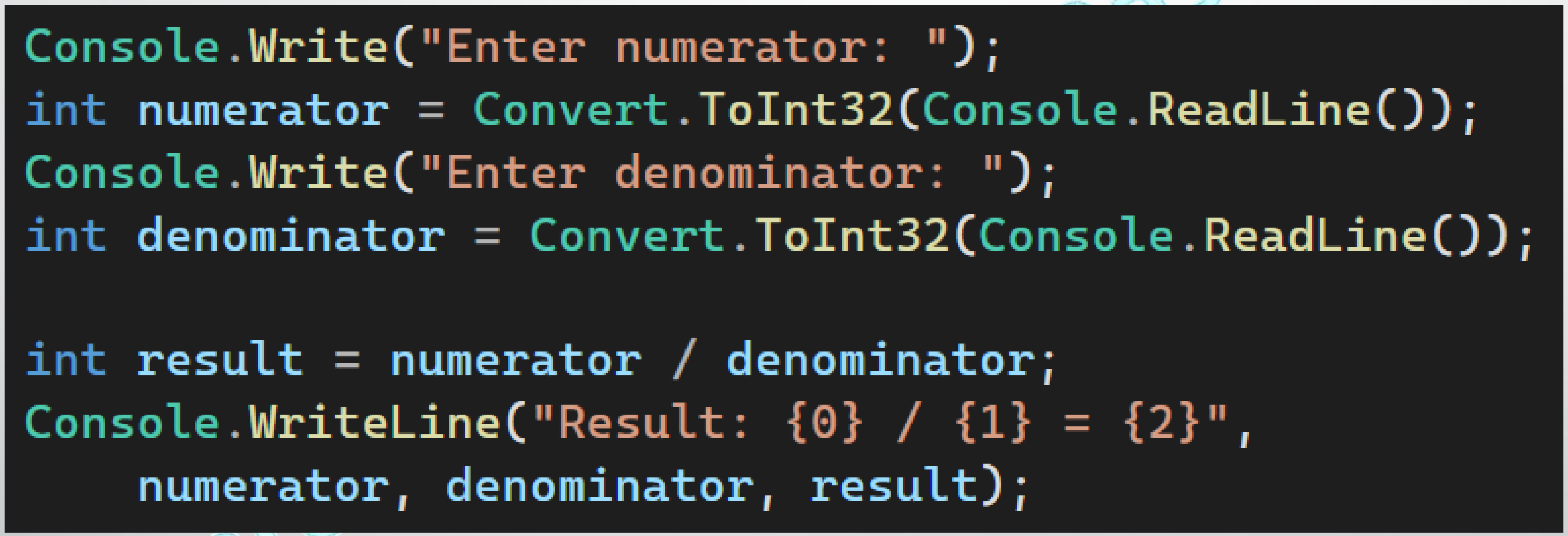
Exception Handling

- Finally
  - The finally statement lets you execute code, after try..catch, regardless of result



```
try
{
    // code that may raise an exception
}
catch (Exception e)
{
    // code that handles the exception
}
finally
{
    // this code is always executed
}
```

Example with Exception



```
Console.Write("Enter numerator: ");
int numerator = Convert.ToInt32(Console.ReadLine());
Console.Write("Enter denominator: ");
int denominator = Convert.ToInt32(Console.ReadLine());

int result = numerator / denominator;
Console.WriteLine("Result: {0} / {1} = {2}",
    numerator, denominator, result);
```