upGrad ENTERPRISE

Exception Handling in C#

By
Narasimha Rao T
Microsoft.Net FSD Trainer
Professional Development Trainer
tnrao.trainer@gmail.com

1. Introduction to Exception Handling

Exception Handling is a mechanism in C# to handle runtime errors gracefully, ensuring program flow continues or fails safely.

Goal: Prevent application crashes by managing exceptions in a structured way.

```
try
{
    // Risky code
}
catch (Exception ex)
{
    // Handle the exception
}
finally
{
    // Cleanup code
}
```

2. Error vs Exception

Aspect	Error	Exception
Type	Unrecoverable	Recoverable
Occurs When	System fails (e.g., hardware crash)	Logical/programming issues
Handling	Cannot be caught in code	Can be caught using try-catch
Example	StackOverflow, OutOfMemory	NullReferenceException, DivideByZeroException

3. try, catch, finally Blocks

`try` Block

- Contains code that may throw an exception.

`catch` Block

- Handles specific exceptions.

`finally` Block

- Executes regardless of an exception occurring or not. Used for cleanup.

```
try
{
    int x = 10, y = 0;
    int result = x / y;
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Cannot divide by zero.");
}
finally
{
    Console.WriteLine("Cleanup done.");
}
```

4. Common Exceptions

`DivideByZeroException`

Occurs when dividing by zero.

```
int x = 10;
int y = 0;
int result = x / y; // Throws DivideByZeroException
```

`NullReferenceException`

Occurs when accessing members of a null object.

```
string str = null;
Console.WriteLine(str.Length); // Throws NullReferenceException
```

5. Catching Multiple Exception Types

```
try
{
    // Some code
}
catch (DivideByZeroException ex)
{
    Console.WriteLine("Math error: " + ex.Message);
}
catch (NullReferenceException ex)
{
    Console.WriteLine("Null reference: " + ex.Message);
}
catch (Exception ex)
{
    Console.WriteLine("General exception: " + ex.Message);
}
```

6. Exception Hierarchy and Flow

All exceptions derive from the base class `System.Exception`.

```
System.Object
   ↳ System.Exception
        ↳ System.SystemException
            ↳ System.DivideByZeroException
            ↳ System.NullReferenceException
```

Execution flow:

- If an exception is thrown, control moves to the first matching `catch` block.
- If no match, it propagates up the call stack.

7. throw vs throw ex

`throw ex`

- Resets the stack trace, losing the original error location.

`throw`

- Preserves original stack trace. Best practice.

```
try
{
    throw new Exception("Error");
}
catch (Exception ex)
{
    // throw ex; ❌ Stack trace reset
    throw;   //  Stack trace preserved
}
```

8. Re-throwing and Preserving Stack Trace

Use `throw` to re-throw an exception while preserving the original call stack.

```
try
{
    SomeMethod();
}
catch (Exception ex)
{
    // Log exception
    throw; // Keeps original trace
}
```

9. Nested try-catch Blocks

Useful for handling exceptions at different levels.

```
try
{
    try
    {
        int[] arr = new int[3];
        Console.WriteLine(arr[5]);
    }
    catch (IndexOutOfRangeException ex)
    {
        Console.WriteLine("Inner catch: " + ex.Message);
    }
}
catch (Exception ex)
{
    Console.WriteLine("Outer catch: " + ex.Message);
}
```

10. Creating Custom Exceptions

Define your own exceptions by extending `System.Exception`.

```
public class MyCustomException : Exception
{
    public MyCustomException(string message) : base(message) { }
}
```

Usage:

```
throw new MyCustomException("This is a custom error.");
```

11. Best Practices in Exception Structure

Best Practices:

- Use specific exceptions first in catch blocks.
- Avoid empty catch blocks.
- Always clean up with `finally` or `using`.
- Do not use exceptions for flow control.
- Log meaningful exception messages.
- Prefer `throw` over `throw ex`.
- Catch only what you can handle.

12. Quiz Questions

1. What is the difference between `throw` and `throw ex`?
2. Which block is always executed in a try-catch-finally structure?
3. What exception is thrown when trying to access a method on a null object?
4. Can we have multiple `catch` blocks for one `try` block?
5. What does the `finally` block usually contain?
6. What is the base class for all exceptions in C#?
7. When should you create a custom exception?
8. What happens if an exception is not caught?
9. Why should you avoid empty `catch` blocks?
10. Which keyword is used to manually raise an exception?

upGrad ENTERPRISE

Q & A

Narasimha Rao T
tnrao.trainer@gmail.com