Exceptions and Exception Handling

05/14/2021 • 2 minutes to read •

In this article

Exceptions Overview

C# Language Specification

See also

The C# language's exception handling features help you deal with any unexpected or exceptional situations that occur when a program is running. Exception handling uses the `try`, `catch`, and `finally` keywords to try actions that may not succeed, to handle failures when you decide that it's reasonable to do so, and to clean up resources afterward. Exceptions can be generated by the common language runtime (CLR), by .NET or third-party libraries, or by application code. Exceptions are created by using the `throw` keyword.

In many cases, an exception may be thrown not by a method that your code has called directly, but by another method further down in the call stack. When an exception is thrown, the CLR will unwind the stack, looking for a method with a `catch` block for the specific exception type, and it will execute the first such `catch` block that if finds. If it finds no appropriate `catch` block anywhere in the call stack, it will terminate the process and display a message to the user.

In this example, a method tests for division by zero and catches the error. Without the exception handling, this program would terminate with a **DivideByZeroException was unhandled** error.

C# Copy

```
public class ExceptionTest
{
    static double SafeDivision(double x, double y)
    {
        if (y == 0)
            throw new DivideByZeroException();
        return x / y;
    }

    public static void Main()
    {
        // Input for test purposes. Change the values to see
        // exception handling behavior.
        double a = 98, b = 0;
        double result;
```

```
        try
        {
            result = SafeDivision(a, b);
            Console.WriteLine("{0} divided by {1} = {2}", a, b, result);
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("Attempted divide by zero.");
        }
    }
}
```

Exceptions Overview

Exceptions have the following properties:

- Exceptions are types that all ultimately derive from `System.Exception`.
- Use a `try` block around the statements that might throw exceptions.
- Once an exception occurs in the `try` block, the flow of control jumps to the first associated exception handler that is present anywhere in the call stack. In C#, the `catch` keyword is used to define an exception handler.
- If no exception handler for a given exception is present, the program stops executing with an error message.
- Don't catch an exception unless you can handle it and leave the application in a known state. If you catch `System.Exception`, rethrow it using the `throw` keyword at the end of the `catch` block.
- If a `catch` block defines an exception variable, you can use it to obtain more information about the type of exception that occurred.
- Exceptions can be explicitly generated by a program by using the `throw` keyword.
- Exception objects contain detailed information about the error, such as the state of the call stack and a text description of the error.
- Code in a `finally` block is executed regardless of if an exception is thrown. Use a `finally` block to release resources, for example to close any streams or files that were opened in the `try` block.
- Managed exceptions in .NET are implemented on top of the Win32 structured exception handling mechanism. For more information, see Structured Exception Handling (C/C++) and A Crash Course on the Depths of Win32 Structured Exception Handling .

C# Language Specification

For more information, see Exceptions in the C# Language Specification. The language specification is the definitive source for C# syntax and usage.

See also

- SystemException
- C# Keywords
- throw
- try-catch
- try-finally
- try-catch-finally
- Exceptions

Is this page helpful?

Yes No

Recommended content

Access Modifiers - C# Programming Guide

All types and type members in C# have an accessibility level which controls whether they can be used from other code. Review this list of access modifiers.

static modifier - C# Reference

static modifier - C# Reference

Static Classes and Static Class Members - C# Programming Guide

Static classes cannot be instantiated in C#. You access the members of a static class by using the class name itself.

Inheritance

Inheritance in C# enables you to create new classes that reuse, extend, and modify the behavior defined in other classes.

Show more