Exceptions and Error Handling

Simon Robinson
Software Developer

@TechieSimon www.SimonRobinson.com

Overview

Different actions for different errors

- Use exception filters

Define custom exceptions

Handle async exceptions

Ensure cleanup code is always executed

Debugging exceptions

Exceptions vs. `Debug.Assert()`

Demo

Read data from JSON file

- Must handle any errors

Distinguishing Different Errors

Possible Errors When Reading the JSON File

```
H:\G_Work\Pluralsight\Courses\CSh Playbook\m11 exceptions\Code\ReadData - Before\bin\Debug\net6
1: DiyProducts-InvalidId.txt
2: DiyProducts-InvalidName.txt
3: DiyProducts-Locked.txt
4: DiyProducts-NotJson.txt
5: DiyProducts-OK.txt
6: DiyProducts-PermissionDenied.txt
7: README.txt
8: DiyProducts-Missing.txt
Enter the number of the file to examine>
```

Invalid product data

File locked

Doesn’t contain JSON

Permission denied

File doesn’t exist

Different Errors Merit Different Actions

Examples:

File doesn't exist

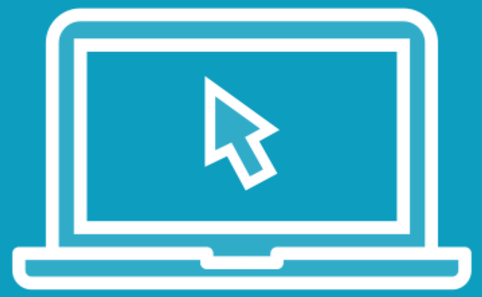
Have user select another file

Some invalid data

Just read the valid data

You need to know what the problem is to decide what action to take!

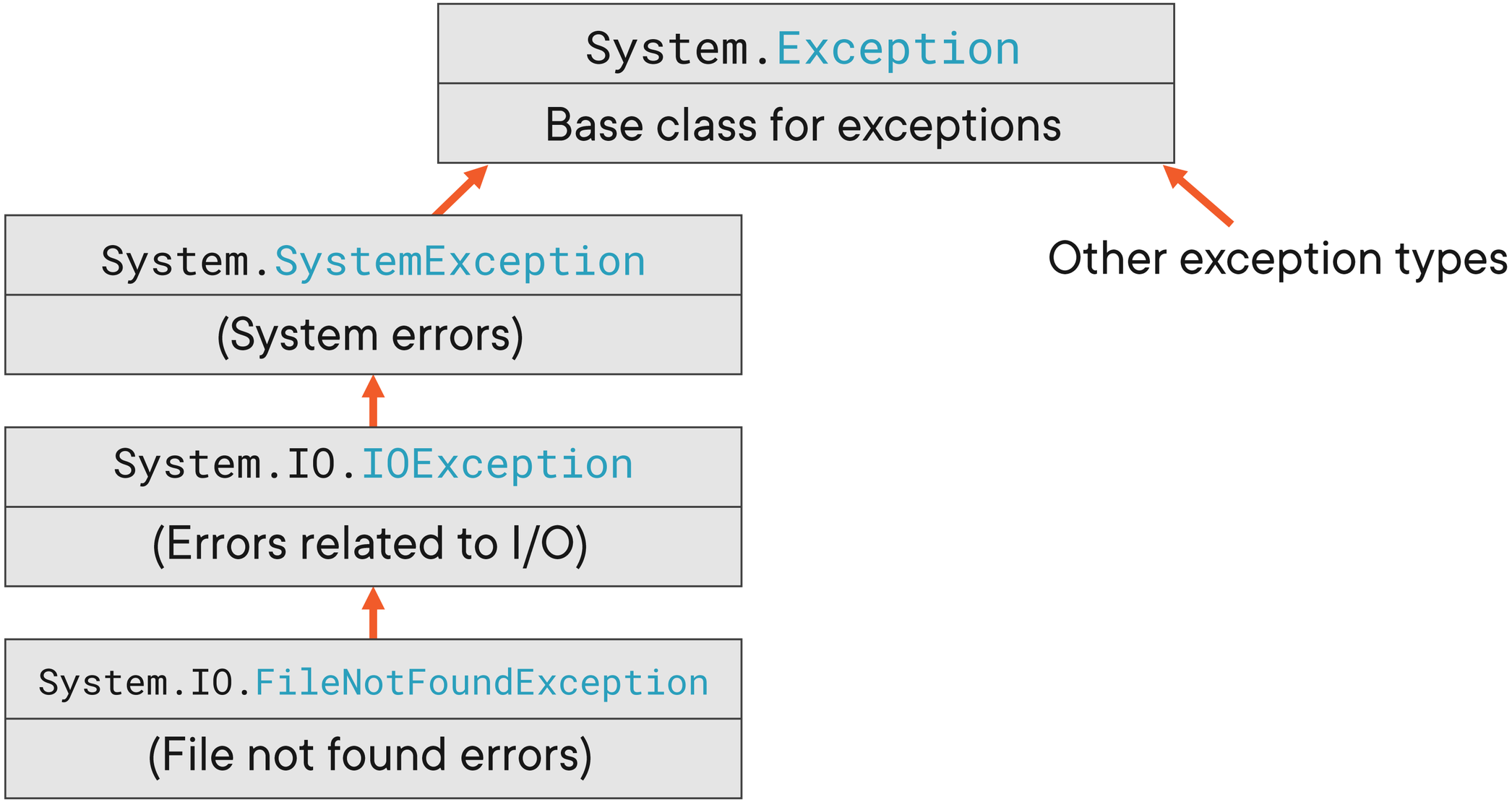
Demo



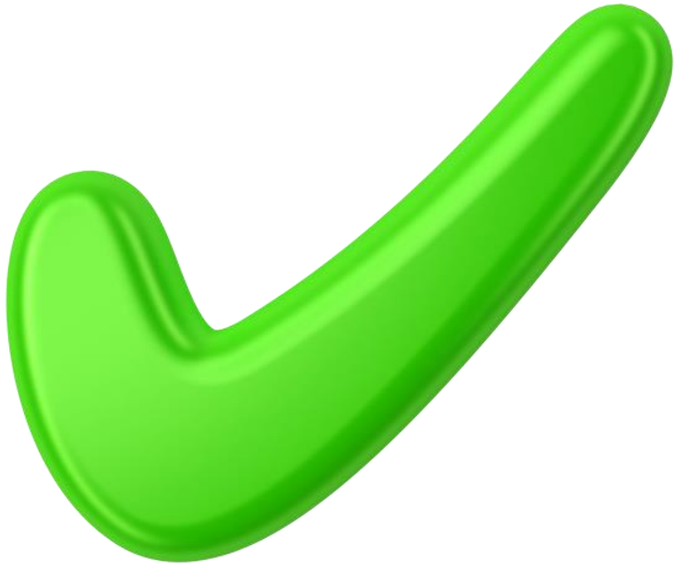
Develop the JSON file reader app

- Handle different exceptions
  - More customized information
  - Each exception handled separately

Exception Types



Multiple `catch` Blocks - Guidelines



Catch most derived exception first

Last catch block should catch every exception

- Unless you are certain the specific blocks cover every exception you need to

Only declare an exception variable if you need it

Handling Custom Errors

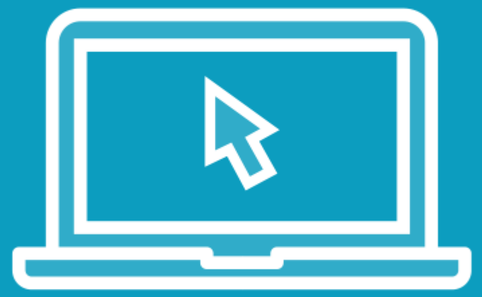
Demo

Add custom exception type

- To cover invalid product data

Using Exception Filters for Finer Error-Handling Control

Demo



New requirement:

- Display different message if the name is invalid

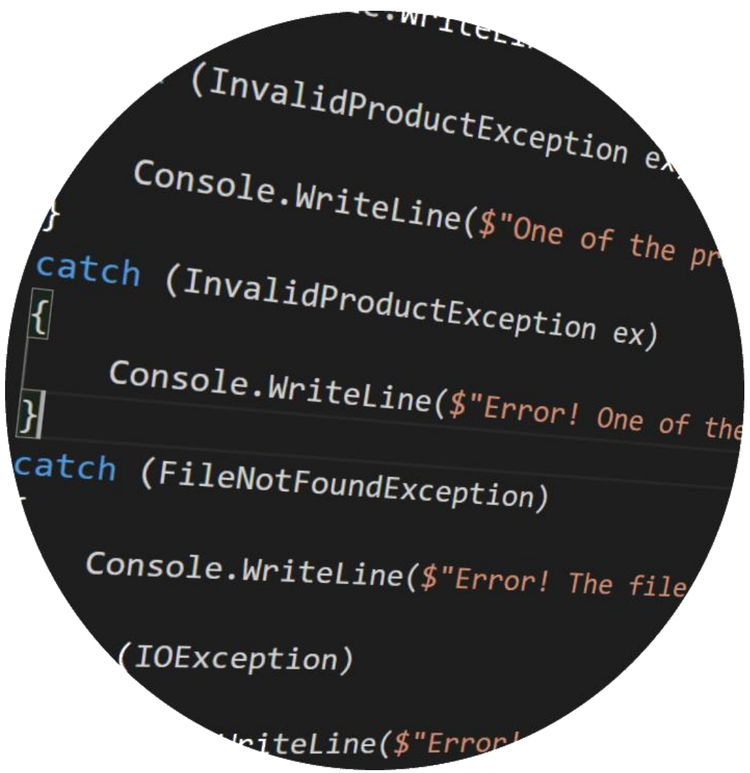
Catching Async Exceptions

Demo

Read the JSON file asynchronously

- Still catch any exceptions

Async Exceptions



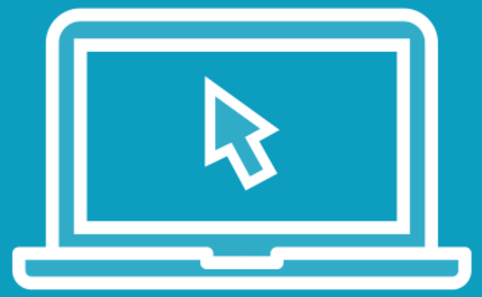
To make async exceptions work out of the box:

- Async method that throws must return Task<T> or Task
- Don't return void
  - `async void` is usually bad practice anyway!
- Async event handlers need to return `void`
  - Have them handle exceptions internally

Throwing exceptions from
async code just works
(As long as they return **Task** or
Task<T>).

Executing Cleanup Code

Demo

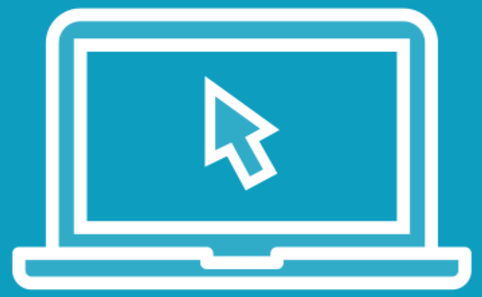


Unlocking a file

- Must always be done – irrespective of whether app ran successfully
  - `finally` block
  - `using` statement

Debugging Exceptions

Demo



Break into debugger as soon as an exception is thrown

- So you can debug why it was thrown

Exceptions vs. `Debug.Assert()`

Summary

Use multiple `catch` blocks to handle different errors differently

- Exception filters for finer control

Catch `System.Exception` last

Write your own exception types for custom errors

- Inherit from `System.Exception`

Async exceptions work just like non-async ones

Summary

Use `finally` to ensure code is always executed

- Or `using` – but only if the object implements `IDisposable` or `IAsyncDisposable`

To debug exceptions with handlers, set "break when thrown" in VS

Use `Debug.Assert()` to monitor for bugs in your code