Creating and Using Custom Exceptions



Jason Roberts
.NET Developer

@robertsjason

dontcodetired.com

Overview



Understanding custom exceptions

- Overview
- When to use
- Implementing

Define a custom calculation exception

Define a custom calculation operation not supported exception

- Add an additional custom property

Catching custom exceptions

An alternative to custom exceptions



Understanding Custom Exceptions

Overview

Use existing predefined .NET exception types where applicable, e.g.

- InvalidOperationException if property set/method call is not appropriate for current state
- ArgumentException (or derived) for invalid parameters

Wrap inner exception if appropriate

Don't use custom exceptions for normal expected (non exceptional) logic flow



Understanding Custom Exceptions

When to use

Only create custom exception types if they need to be caught and handled differently from existing predefined .NET exceptions

 E.g. want to perform special monitoring of a specific critical exception type

If building a library/framework for use by other developers so consumers can react specifically to errors in your library

Interfacing with an external API, DLL, service



Understanding Custom Exceptions

Implementing

Naming convention: ... Exception

Implement standard 3 constructors

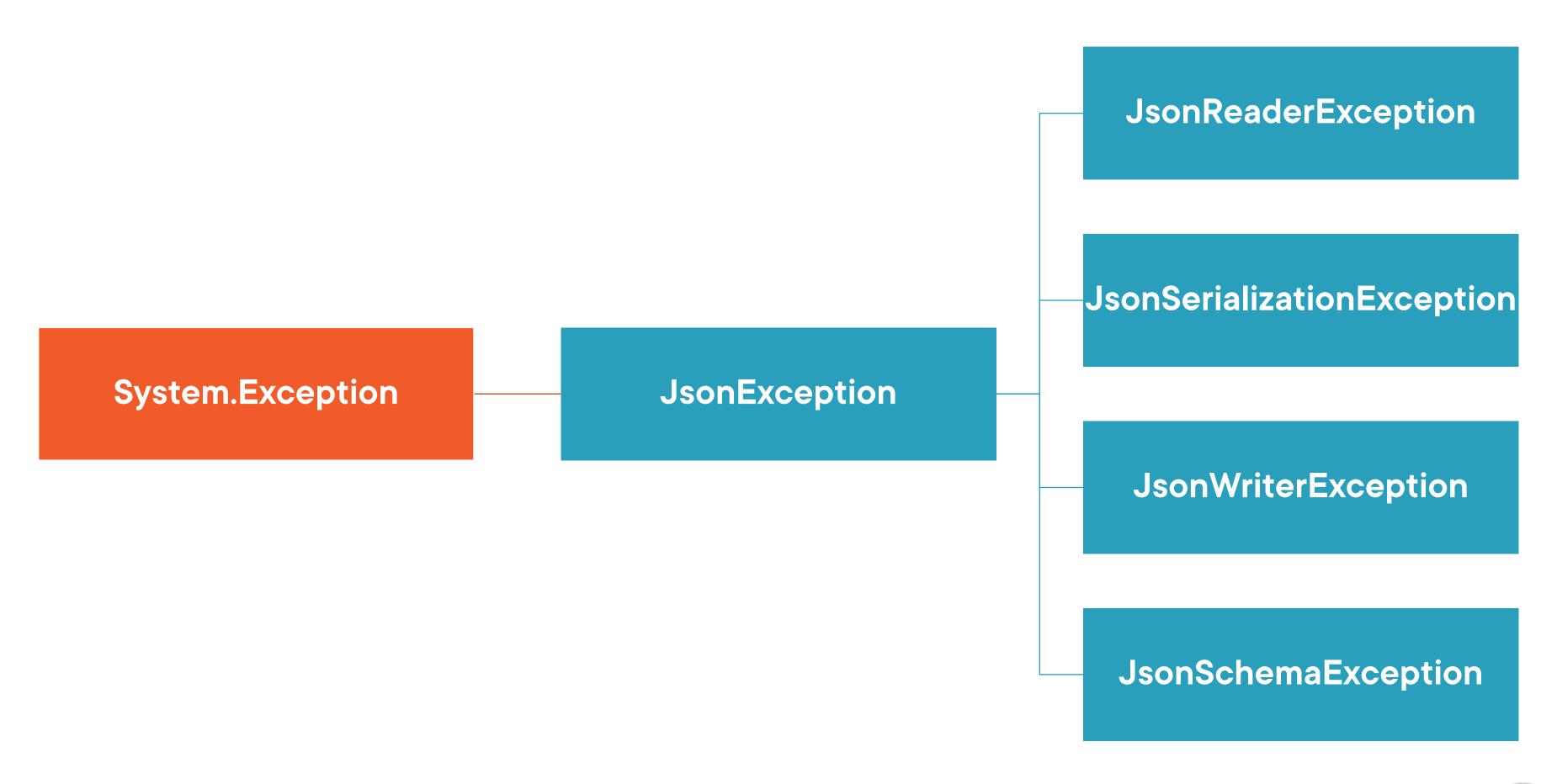
Add additional properties where needed

Never inherit from Application Exception

Inherit from Exception (or your other custom exception types)

Keep the number of custom exception types to a minimum







IDictionary

Key/value pairs

String key

Object value

Arbitrary number of items

- ListDictionaryInternal
- Optimized for smaller number of items
- "...smaller and faster than a Hashtable if the number of elements is 10 or less"

Additional/supplementary user-defined exception information

Data



```
public int Calculate(int number1, int number2, string operation)
   if (nonNullOperation == "/")
   else
      throw new ArgumentOutOfRangeException(nameof(operation),
          "The mathematical operator is not supported.");
```

```
else
{
    throw new ArgumentOutOfRangeException(nameof(operation),
        "The mathematical operator is not supported.");
}
```



```
else
{
    var ex = new ArgumentOutOfRangeException(nameof(operation),
        "The mathematical operator is not supported.");
    ex.Data.Add(key: "SUPPLIED_OPERATION", value: operation);
    throw ex;
}
```

```
catch (ArgumentOutOfRangeException ex)
   string suppliedOperation = "unknown";
   if (ex.Data.Contains("SUPPLIED_OPERATION"))
      suppliedOperation = (string) ex.Data["SUPPLIED_OPERATION"];
   WriteLine(
       $"Operation '{suppliedOperation}' is not supported. {ex}");
```



Data Considerations

Not secure:

- Any catch block can read keys/values
- Any catch block can change keys/values
- Any catch block can delete items
- Any catch block can add new items

Do not store sensitive/confidential data

Application should still operate correctly if data items missing/corrupted

Be careful of key conflicts:

- System.ArgumentException



Summary



Understanding custom exceptions

- Use existing predefined .NET exception types where applicable
- Only create custom exception types if they need to be caught
- ...Exception

CalculationException

CalculationOperationNotSupportedException

public string Operation { get; }

Catch custom exceptions

An alternative to custom exceptions



Up Next:

Writing Automated Tests for Exception Throwing Code

