Managing Exceptions

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Exception:

- Validation issues
- Business rule violations
- System errors
- Application failures

Exception handling:

- Process by which the application catches an exception
- And responds to correct the problem or notify the user

Module Overview

Defending our code from exceptions

Throwing .NET exceptions

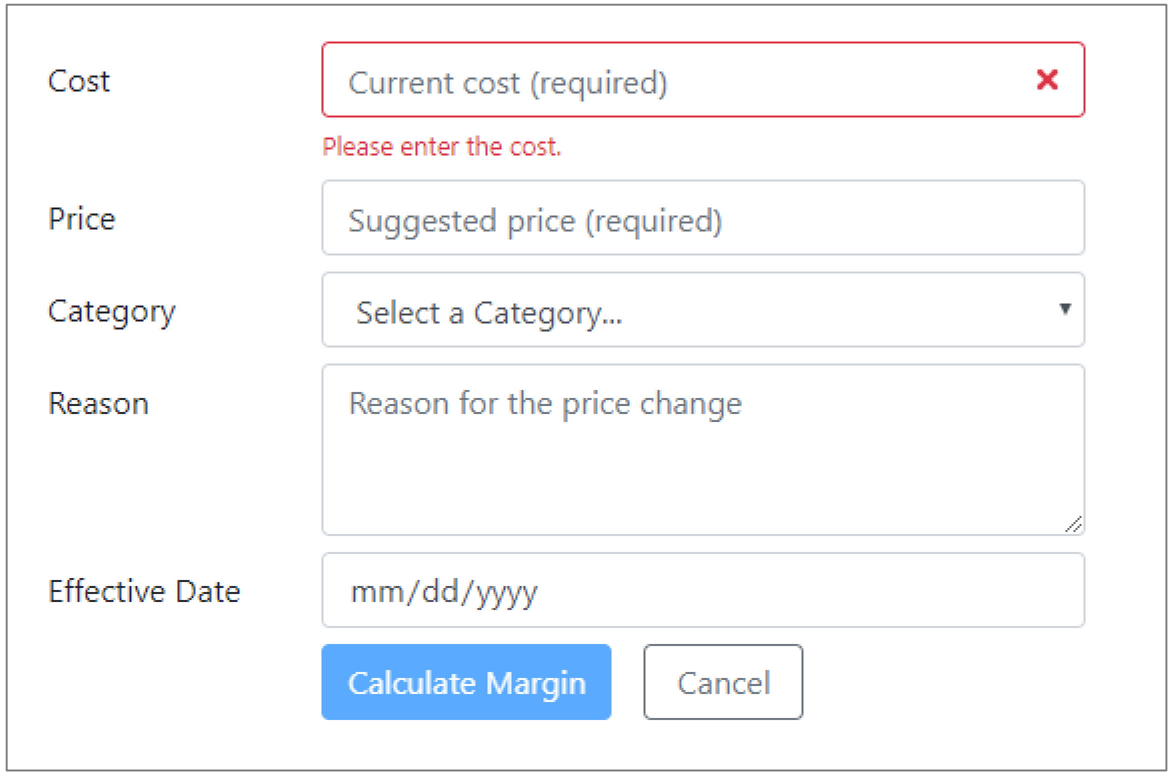
Creating and throwing custom exceptions

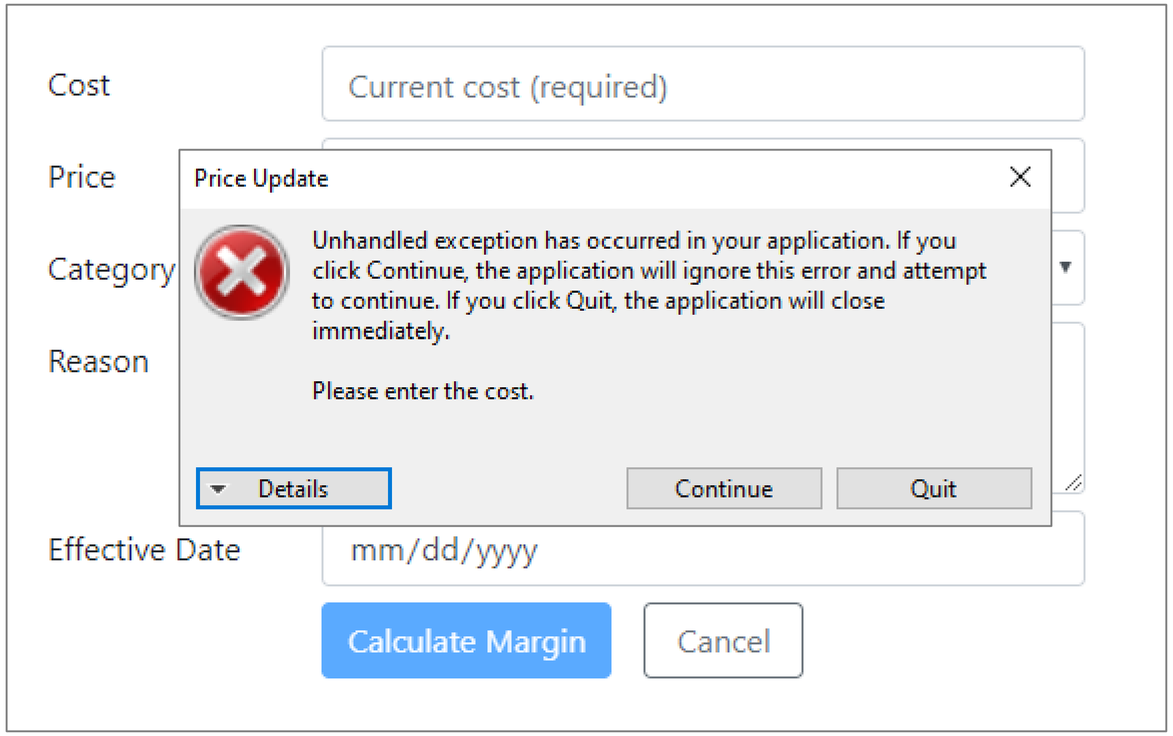
Catching what we're thrown

Defending Our Code from Exceptions

```
public decimal CalculateMargin(string costInput, string priceInput)
{
  if (string.IsNullOrWhiteSpace(costInput))
     throw new ArgumentException("Please enter the cost");
  // ...

  var success = decimal.TryParse(costInput, out decimal cost);
  if (!success || cost < 0)
     throw new ArgumentException("The cost must be >= 0");

  // ...

  return ((price - cost) / price) * 100M;
}
```

Defending Our Code from Our Exceptions





Unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately.

Please enter the cost.

Defending Our Code from System Exceptions

```
public static void LogToFile(string textToLog)
{
   string docPath = "somePath";

   using (StreamWriter w = File.AppendText(Path.Combine(docPath, "log.txt")))
   {
     w.WriteLine("");
     w.Write("Log Entry: ");
     w.WriteLine($"{DateTime.Now.ToLongTimeString()}");
     w.WriteLine($" {logText}");
   }
}
```

Anticipate Exceptions

```
public decimal CalculateMargin(string costInput,
                               string priceInput)
{
  if (string.IsNullOrWhiteSpace(costInput))
    throw new ArgumentException("Please enter the cost");
  // ...
  return ((price - cost) / price) * 100M;
}
```

Exceptions thrown from a method

```
public static void LogToFile(string textToLog)
{
   using (StreamWriter w = File.AppendText("log.txt"))
   {
     w.WriteLine($"{DateTime.Now.ToLongTimeString()}");
     w.WriteLine($"  {logText}");
   }
}
```

System or application exceptions

Exception Management Strategy

Should Our Methods Throw Exceptions?

```
public decimal CalculateMargin(string costInput,
                               string priceInput)
{
  if (string.IsNullOrWhiteSpace(costInput))
    throw new ArgumentException("Please enter the cost");
  // ...
  return ((price - cost) / price) * 100M;
}
```

```
public OperationResult ValidateDate(DateTime? effectiveDate)
{
  if (!effectiveDate.HasValue) return new OperationResult()
    { Success = false, Message = "Date has no value" };

  if (effectiveDate.Value < DateTime.Now.AddDays(7))
   return new OperationResult()
    { Success = false,
      Message = "Date must be >= 7 days from today" };

  return new OperationResult() { Success = true };
}
```

What type of exceptions?

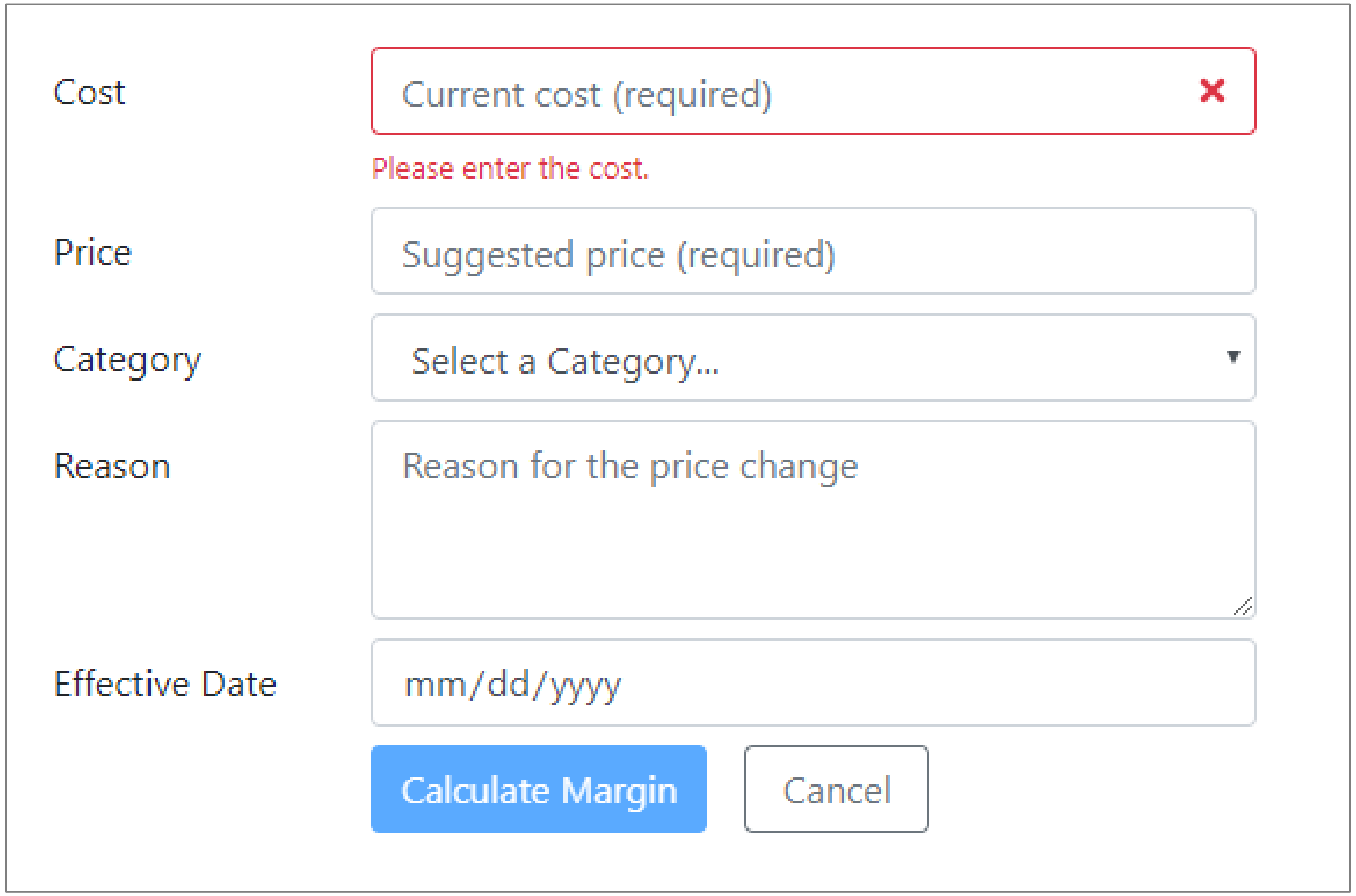
- **ArgumentException?**
- **Custom exception?**

Or use an alternate technique?

What About System and Application Exceptions?

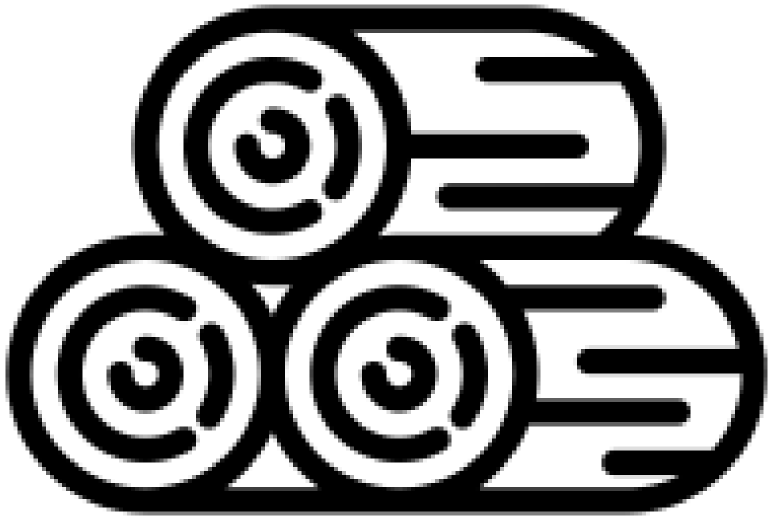
```
public static void LogToFile(string textToLog)
{
   string docPath = "somePath";

   using (StreamWriter w = File.AppendText(Path.Combine(docPath, "log.txt")))
   {
     w.WriteLine("");
     w.Write("Log Entry: ");
     w.WriteLine($"{DateTime.Now.ToLongTimeString()}");
     w.WriteLine($" {logText}");
   }
}
```

How Should the User Be Notified?



Please enter the cost.

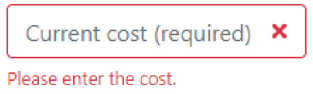
How and When Do We Log Exceptions?



Exception Management Strategy

Should our methods throw exceptions? If so, which ones?

What about system and application exceptions?


Current cost (required) ×

Please enter the cost.

How should the user be notified?

How and when do we log exceptions?

Providing Multiple Results from a Method

Throwing exceptions

`ref` or `out` parameters

A tuple

An object

Throwing .NET Exceptions

```
public Discount FindDiscount(List<Discount>? discounts, string name)
{
  if (discounts is null)
       throw new ArgumentException("No discounts found");

  var foundDiscount = discounts.Find(d => d.DiscountName == name);

  if (foundDiscount is null)
       throw new KeyNotFoundException("Discount not found");

  return foundDiscount;
}
```

```
public Discount FindDiscount(List<Discount>? discounts, string name)
{

  if (discounts is null)
        throw new ArgumentException("No discounts found");

 // ...
}
```

Argument Exception

Use if a method argument is invalid based on:

- Validation requirements
- Business rules

```
string docPath =
     Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

if (String.IsNullOrEmpty(docPath))
      throw new InvalidOperationException("Path cannot be null");
```

Invalid Operation Exception

Use if an operation is not valid based on the application state (other than invalid arguments):

- Attempting to write to a file without a valid file name

Do NOT Throw

Exception

SystemException

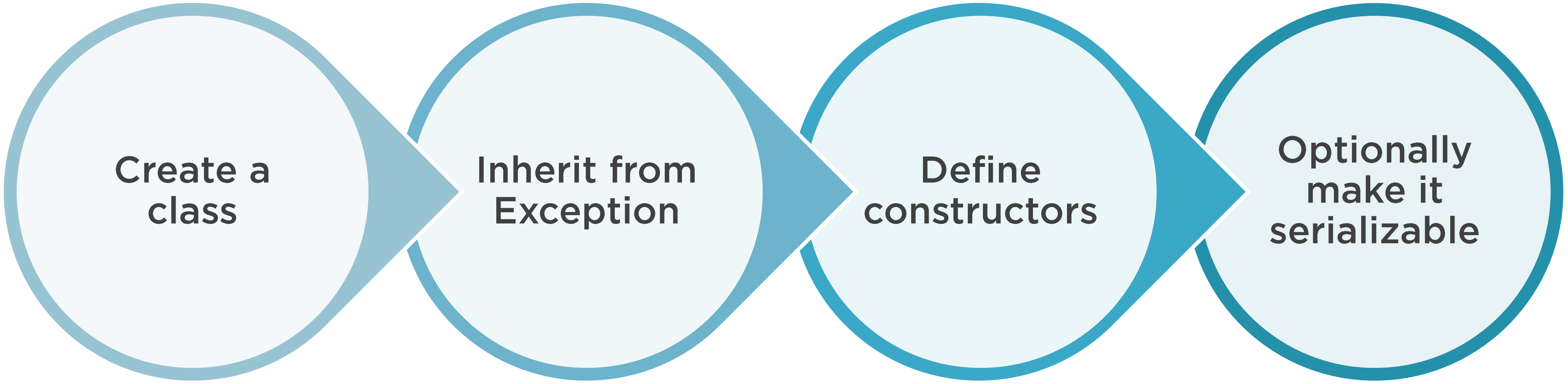
NullReferenceException

IndexOutOfRangeException

Throwing a Custom Exception

```
public Discount FindDiscount(List<Discount>? discounts, string name)
{
  if (discounts is null)
       throw new ArgumentException("No discounts found");

  var foundDiscount = discounts.Find(d => d.DiscountName == name);

  if (foundDiscount is null)
       throw new DiscountFoundException("Discount not found");

  return foundDiscount;
}
```

Creating a Custom Exception



Demo

Creating a custom exception

Catching What We're Thrown

```
public decimal CalculateMargin(string costInput,
                               string priceInput)
{
  if (string.IsNullOrWhiteSpace(costInput))
    throw new ArgumentException("Please enter the cost");
  // ...
  return ((price - cost) / price) * 100M;
}
```

Exceptions thrown from a method

```
public static void LogToFile(string textToLog)
{
   using (StreamWriter w = File.AppendText("log.txt"))
   {
     w.WriteLine($"{DateTime.Now.ToLongTimeString()}");
     w.WriteLine($"  {logText}");
   }
}
```

System or application exceptions

try-catch Statement

```
try
 {
   calculatedMargin = product.CalculateMargin(cost, price);
 }
 catch (ArgumentException ex)
 {
   // Display a nice message to the user
   Debug.WriteLine(ex.Message);
 }
```

```
 public decimal CalculateMargin(string costInput, string priceInput)
 {
   if (string.IsNullOrWhiteSpace(costInput))
       throw new ArgumentException("Please enter the cost");
   // ...
   return ((price - cost) / price) * 100M;
 }
```

Throwing Different Exceptions

```
public bool ValidateEffectiveDate(DateTime? effectiveDate)
{
   if (!effectiveDate.HasValue)
      throw new ValidationException("Please enter the effective date");

   if (effectiveDate.Value < DateTime.Now.AddDays(7))
      throw new BusinessRuleException("Date must be > 7 days from today");

   return true;
}
```

try-catch Statement

```
try
{
  calculatedMargin = product.CalculateMargin(cost, price);
}

catch (ValidationException ex)
{
  // Display a validation error message to the user
}

catch (BusinessRuleException ex)
{
  // Display a business rule message to the user
}
```

Leveraging Exception Filters

```
try
{
  calculatedMargin = product.CalculateMargin(cost, price);
}

catch (ValidationException ex) when (ex.ParamName == "cost")
{
  // Display a validation error message on the cost field
}

catch (ValidationException ex) when (ex.ParamName == "price")
{
  // Display a validation error message on the price field
}
```

Inheriting from ArgumentException

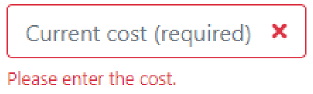
```
[Serializable()]
public class ValidationException : System.ArgumentException
{
 public ValidationException() : base() { }

 public ValidationException(string message) : base(message) { }

 public ValidationException(string message, string paramName) : base(message, paramName) { }

 public ValidationException(string message, Exception inner) : base(message, inner) { }

 protected ValidationException(System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context) : base(info, context) { }

 }
```

Guidelines and Summary

Define an Exception Management Strategy

Should our methods throw exceptions? If so, which ones?

What about system and application exceptions?



Current cost (required) ×

Please enter the cost.

How should the user be notified?

How and when do we log exceptions?

Throw the Appropriate .NET Exceptions

```
public Discount FindDiscount(List<Discount>? discounts, string name)
{
  if (discounts is null)
        throw new ArgumentException("No discounts found");
 // ...
}
```

For invalid arguments

```
string docPath =
    Environment.GetFolderPath(Environment.SpecialFolder.MyDocuments);

if (String.IsNullOrEmpty(docPath))
       throw new InvalidOperationException("Path cannot be null");
```

For most everything else

Create a Custom Exception

Create an exception class

Inherit from System.Exception

Define appropriate constructors

```
[Serializable()]
public class ValidationException : System.Exception
{

public ValidationException() : base() { }

public ValidationException(string message) : base(message) { }

public ValidationException(string message, Exception inner) : base(message, inner) { }

 protected ValidationException(System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context) : base(info, context) { }

}
```

Create a Custom Exception

Optionally make the exception serializable

```
[Serializable()]
public class ValidationException : System.Exception
{

 public ValidationException() : base() { }

 public ValidationException(string message) : base(message) { }

 public ValidationException(string message, Exception inner) : base(message, inner) { }

 protected ValidationException(System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context) : base(info, context) { }

}
```

Catch What You're Thrown

Could throw exception

```
try
{
  calculatedMargin = product.CalculateMargin(cost, price);
}

catch (ValidationException ex)
{
  // Display a validation error message
}

catch (BusinessRuleException ex)
{
  // Display a business rule message
}
```

Catch any expected exception

Leverage Exception Filters

```
try
{
  calculatedMargin = product.CalculateMargin(cost, price);
}

catch (ValidationException ex) when (ex.ParamName == "cost")
{
  // Display a validation error message on the cost field
}

catch (ValidationException ex) when (ex.ParamName == "price")
{
  // Display a validation error message on the price field
}
```

Process exceptions based on a filter

Inherit from ArgumentException if needed

Inherit from ArgumentException

Add associated constructors

```
[Serializable()]
public class ValidationException : System.ArgumentException
{
 public ValidationException() : base() { }

 public ValidationException(string message) : base(message) { }

 public ValidationException(string message, string paramName) : base(message, paramName) { }

 public ValidationException(string message, Exception inner) : base(message, inner) { }

 protected ValidationException(System.Runtime.Serialization.SerializationInfo info,
    System.Runtime.Serialization.StreamingContext context) : base(info, context) { }

 }
```

Our Application Is Defended

Incorrect entry

Invalid operations

System mishaps

Future developers