Validating Method Arguments

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

Methods

```
public decimal CalculateMargin(string cost, string price)
{
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

```
public bool Operation(bool x, string a,
                      string b, bool y,
                      string c, bool z,
                      string d)
{
    // Perform operation
}
```

Methods

```
public void Operation(bool x, string a,
                      string b, bool y,
                      string c, bool z,
                      string d)
{ // Perform operation }
```

```
public void SendEmail(bool x, string a,
                      string b, bool y,
                      string c, bool z,
                      string d)
{ // Perform operation }
```

```
public bool SendEmail(bool saveCopy, string recipient,
                      string body, bool includeSignature,
                      dateTime sendDate, bool highPriority,
                      string subject)
{ // Perform operation }
```

Methods

```
public bool SendEmail(bool saveCopy, string recipient,
                      string body, bool includeSignature,
                      dateTime sendDate, bool highPriority,
                      string subject)
{ // Perform operation }
```

```
public bool SendEmail(string recipient, string subject,
                      string body, dateTime sendDate,
                      bool saveCopy, bool highPriority,
                      bool includeSignature)
{ // Perform operation }
```

```
public bool SendEmail(string recipient, string subject,
                      string body, dateTime sendDate,
                      bool saveCopy=false, bool highPriority=false,
                      bool includeSignature=true)
{ // Perform operation }
```

Calling a Method

```
public bool SendEmail(string recipient, string subject,
                      string body, dateTime sendDate,
                      bool saveCopy=false, bool highPriority=false,
                      bool includeSignature=true)
{ // Perform operation }
```

```
var utils = new Utility();
utils.SendEmail("Jack Harkness", "Today's Meeting",
                "Please confirm our 1PM meeting", DateTime.Now);
```

Named Arguments

```
public bool SendEmail(string recipient, string subject,
                      string body, dateTime sendDate,
                      bool saveCopy=false, bool highPriority=false,
                      bool includeSignature=true)
{ // Perform operation }
```

```
var utils = new Utility();
utils.SendEmail("Jack Harkness", "Today's Meeting",
                "Please confirm our 1PM meeting", DateTime.Now,
                true, true, false);
```

```
var utils = new Utility();
utils.SendEmail("Jack Harkness", "Today's Meeting",
                "Please confirm our 1PM meeting", DateTime.Now,
                highPriority: true);
```

Calling a Method

```
public decimal CalculateMargin(string cost, string price)
{  // Calculate profit margin }
```

```
var product = new Product();
product.CalculateMargin("100", "200");
```

```
var product = new Product();
product.CalculateMargin(cost: "100", price: "200");
```

```
var product = new Product();
product.CalculateMargin(price: "200", cost: "100");
```

```
string cost = "100";
string price = "200";
var product = new Product();
product.CalculateMargin(cost, price);
```

Module Overview

Surrounding our operations with conditionals

Failing fast with guard clauses

Unit testing for expected exceptions

Refactoring our methods

Demo

Surrounding our operations with conditionals

Guard Clauses

```
private decimal CalculateMargin(string cost, string price)
{
```

```
    // Calculate profit margin
    return ((price - cost) / price) * 100;
}
```

Benefits of Guard Clauses

```
private decimal CalculateMargin(string cost, string price)
{
   var success = decimal.TryParse(costInput, out decimal cost);
   if (!success) throw new ArgumentException("The cost must be a number");

   success = decimal.TryParse(priceInput, out decimal price);
   if (!success || price <= 0)
      throw new ArgumentException("The price must be a number greater than 0");

   // Calculate profit margin
   return ((price - cost) / price) * 100;
}
```

Benefits of Guard Clauses

```
private decimal CalculateMargin(string c
{
    var success = decimal.TryParse(costIn
    if (!success) throw new ArgumentExcep

    success = decimal.TryParse(priceInput
    if (!success || price <= 0)
        throw new ArgumentException("The p

    // Calculate profit margin
    return ((price - cost) / price) * 100
}
```

```
public decimal CalculateMargin(string costInput,
                               string priceInput)
{
 var success = decimal.TryParse(costInput,
                                out decimal cost);

 decimal margin = 0;
 if (success)
 {
  success = decimal.TryParse(priceInput,
                             out decimal price);

  if (success && price > 0)
  {
    margin = ((price - cost) / price) * 100M;
  }
 }

 return margin;
}
```

Demo

Failing fast with guard clauses

Demo

Unit testing for expected exceptions

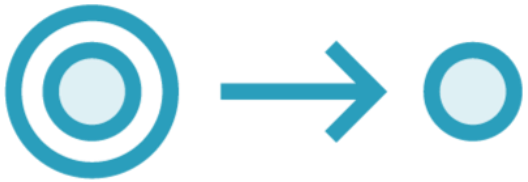
Our Method Has Grown!

```
public decimal CalculateMargin(string cost, string price)
{
    return ((price - cost) / price) * 100;
}
```

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

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

  success = decimal.TryParse(priceInput, out decimal price);
  if (!success || price <= 0) throw new ArgumentException("The price must be a number greater than 0");

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

Refactoring Our Method

Helper method



Method overloading

Guard class

Building Helper Methods

```
public decimal ValidateCost(string costInput)
{
  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 a number 0 or greater");

  return cost;
}
```

```
public decimal ValidatePrice(string priceInput)
{
  if (string.IsNullOrWhiteSpace(priceInput)) throw new ArgumentException("Please enter the price");

  var success = decimal.TryParse(priceInput, out decimal price);
  if (!success || price <= 0) throw new ArgumentException("The price must be a number greater than 0");

  return price;
}
```

Building Helper Methods

```
public decimal ValidateCost(string costInput)
{
  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 a number 0 or greater");

  return cost;
}
```

```
public decimal ValidatePrice(string priceInput)
{
  if (string.IsNullOrWhiteSpace(priceInput)) throw new ArgumentException("Please enter the price");

  var success = decimal.TryParse(priceInput, out decimal price);
  if (!success || price <= 0) throw new ArgumentException("The price must be a number greater than 0");

  return
}
```

```
public decimal CalculateMargin(string costInput, string priceInput)
{
  var cost = ValidateCost(costInput);
  var price = ValidatePrice(priceInput);

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

Method Overloading

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

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

  success = decimal.TryParse(priceInput, out decimal price);
  if (!success || price <= 0) throw new ArgumentException("The price must be a number greater than 0");

 return CalculateMargin(cost, price);
}
```

```
private decimal CalculateMargin(decimal cost, decimal price)
{
  return ((price - cost) / price) * 100M;
}
```

Building a Guard Class

```
public static class Guard
{
  public static void ThrowIfNullOrEmpty(string argumentValue, string message)
  {
    if (string.IsNullOrWhiteSpace(argumentValue)) throw new ArgumentException(message);
  }

  public static decimal ThrowIfNotPositiveDecimal(string argumentValue, string message)
  {
    var success = decimal.TryParse(argumentValue, out decimal result);
    if (!success || result < 0) throw new ArgumentException(message);

    return result;
  }
}
```

Using a Guard Class

```
public decimal CalculateMargin(string costInput, string priceInput)
{
  Guard.ThrowIfNullOrEmpty(costInput, "Please enter the cost");
  Guard.ThrowIfNullOrEmpty(priceInput, "Please enter the price");

  var cost = Guard.ThrowIfNotPositiveDecimal(costInput,
                "The cost must be a number 0 or greater");
  var price = Guard.ThrowIfNotPositiveNonZeroDecimal(priceInput,
                "The price must be a number greater than 0");

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

Demo

Building and using a Guard class

Guidelines and Summary

Define a Clear Method Signature

Define good parameter names

```
public bool SendEmail(string recipient, string subject,
                      string body, dateTime sendDate,
                      bool saveCopy = false,
                      bool highPriority = false,
                      bool includeSignature = true)
{ // Perform operation }
```

Select a good method name

Carefully consider parameter order

Surround Operations with Conditionals

```
public decimal CalculateMargin(string costInput, string priceInput)
{
 var success = decimal.TryParse(costInput, out decimal cost);

 decimal margin = 0;
 if (success)
 {
  success = decimal.TryParse(priceInput, out decimal price);

  if (success && price > 0)
  {
    margin = ((price - cost) / price) * 100M;

 return margin;
}
```

No validation information returned

Operation is obscured

Protected from invalid values

Fail Fast with Guard Clauses

```
private decimal CalculateMargin(string cost, string price)
{
    var success = decimal.TryParse(costInput, out decimal cost);
    if (!success) throw new ArgumentException("The cost must be a number");
```

Fail fast by throwing an exception

Provide validation information

```
    success = decimal.TryParse(priceInput, out decimal price);
    if (!success || price <= 0)
        throw new ArgumentException("The price must be a number greater than 0");

    // Calculate profit margin
    return ((price - cost) / price) * 100;
```

Protected from invalid values

Operation is not obscured

Refactor

```
public decimal ValidateCost(string costInput)
{  // Validate the cost argument }
```

Build helper methods

```
public decimal CalculateMargin(string costInput, string priceInput)
{ // Guard clauses here
  return CalculateMargin(cost, price);
}
```

Use method overloading

```
private decimal CalculateMargin(decimal cost, decimal price)
{
  return ((price - cost) / price) * 100M;
}
```

Build a guard class

```
public static class Guard
{
  public static void ThrowIfNullOrEmpty(string argumentValue, string message)
  {  }

  public static decimal ThrowIfNotPositiveDecimal(string argumentValue, string message)
  {  }
}
```

Validating Method Arguments

```
private decimal CalculateMargin(string cost, string price)
{
  Guard.ThrowIfNullOrEmpty(costInput, "Please enter the cost");
  Guard.ThrowIfNullOrEmpty(priceInput, "Please enter the price");

  var cost = Guard.ThrowIfNotPositiveDecimal(costInput,
                "The cost must be a number 0 or greater");
  var price = Guard.ThrowIfNotPositiveNonZeroDecimal(priceInput,
                "The price must be a number greater than 0");

   // Calculate profit margin
   return ((price - cost) / price) * 100;
}
```