Returning Predictable Results

Deborah Kurata
CONSULTANT | SPEAKER | AUTHOR | MVP | GDE

@deborahkurata | blogs.msmvps.com/deborahk/

What Should Our Methods Return?

Value	Nullable value	Multiple values
Object	Nullable reference	Void

Exceptions

Predictable:
"Behaving in a way that is expected"

Merriam-Webster dictionary

Expected Results?

```
public bool ValidateEffectiveDate(DateTime? effectiveDate)
{
   if (!effectiveDate.HasValue) return false;

   if (effectiveDate.Value < DateTime.Now.AddDays(7)) return false;

   return  true;
}
```

Expected Results?

```
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;
}
```

Expected Results?

```
public Discount? FindDiscount(List<Discount>? discounts, string name)
{
  if (discounts is null) return null;

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

  return foundDiscount;
}
```

Expected Results?

```
public bool SavePrice(int productId, string cost, string price,
                      string category, string reason,
                      DateTime effectiveDate)
{
   // Validate arguments
   // Calls a method in the data layer to save the data...

   return result;
}
```

Module Overview

Providing multiple results from a method:

- `ref` and `out` parameters
- Tuples
- Objects

Returning predictable results from:

- Validation methods
- Simple operations
- Find and retrieve methods
- Complex operations

Providing Multiple Results from a Method

`ref` parameters

`out` parameters

A tuple

An object

ref Parameters

```
public bool ValidateEffectiveDate(DateTime? effectiveDate,
                                  ref string validationMessage)
{
   if (!effectiveDate.HasValue)
   {
     validationMessage = "Date has no value";
     return false;
   }

   if (effectiveDate.Value < DateTime.Now.AddDays(7))
   {
     validationMessage = "Date must be >= 7 days from today";
     return false;
   }
   return true;
}
```

```
var message = "";
var isValid = product.ValidateEffectiveDate(effectiveDate, ref message);
```

out Parameters

```
public bool ValidateEffectiveDate(DateTime? effectiveDate,
                                  out string validationMessage)
{
   validationMessage = "";
   if (!effectiveDate.HasValue)
   {
     validationMessage = "Date has no value";
     return false;
   }
   if (effectiveDate.Value < DateTime.Now.AddDays(7))
   {
     validationMessage = "Date must be >= 7 days from today";
     return false;
   }
   return true;
}
```

```
var isValid = product.ValidateEffectiveDate(effectiveDate,
                                            out string message);
```

Providing Multiple Results from a Method

`ref` parameters

`out` parameters

A tuple

Tuple

```
var validation = (false, "");
```

```
var validation = (IsValid: false, Message: "");
```

Tuple

```
public (bool IsValid, string Message) ValidateEffectiveDate(DateTime? effectiveDate)
{
  if (!effectiveDate.HasValue) return (IsValid: false, Message: "Date has no value");

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

  return (IsValid: true, Message: "");
}
```

```
var result = product.ValidateEffectiveDate(effectiveDate);

if (!result.IsValid) // ... display result.Message;
```

Providing Multiple Results from a Method

`ref` parameters

`out` parameters

A tuple

An object

Define a Class

```
public class OperationResult
{
  public bool Success { get; set; }
  public string Message { get; set; }

  public OperationResult()
  {
    Message = "";
    Success = false;
  }
}
```

Return an Object

```
public OperationResult ValidateEffectiveDate(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 };
}
```

```
var result = product.ValidateEffectiveDate(effectiveDate);

if (!result.Success) // ... display result.Message;
```

Returning Predictable Results

- Value
- Nullable Value
- ref/out tuple
- Object
- Nullable Reference
- Void
- Exception

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Validation Methods

```
public bool ValidateEffectiveDate(DateTime? effectiveDate)
{
    if (!effectiveDate.HasValue) throw ...;

    if (effectiveDate.Value < DateTime.Now.AddDays(7)) throw ...;

    return true;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Validation Methods

```
public OperationResult ValidateEffectiveDate(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 };
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Simple Operations Returning a Value

```
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;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Simple Operations Returning a Value

```
public (decimal? Margin, string? Message) CalculateMargin(string costInput,
                                                          string priceInput)
{
  if (string.IsNullOrWhiteSpace(costInput))
      return (Margin: null, Message: "Please enter the cost");
  // ...

  var success = decimal.TryParse(costInput, out decimal cost);
  if (!success || cost < 0)
      return (Margin: null, Message: "The cost must be >= 0");

  // ...

  var margin = ((price - cost) / price) * 100M;
  return (Margin: margin, Message: null);
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Find and Retrieve Methods

```
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;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Find and Retrieve Methods

```
public Discount? FindDiscount(List<Discount>? discounts, string name)
{
  if (discounts is null) return null;

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

  return foundDiscount;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Find and Retrieve Methods

```
public (Discount? Discount, string? Message) FindDiscount(
                                                List<Discount>? discounts,
                                                string name)
{
  if (discounts is null)
       return (Discount: null, Message: "No discounts found");

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

  if (foundDiscount is null)
       return (Discount: null, Message: "Discount not found");

  return (Discount: foundDiscount, Message: null );
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Complex Operations

```
public bool SavePrice(int productId, string cost, string price,
                      string category, string reason,
                      DateTime effectiveDate)
{
    // Validate arguments
    // Call a method in the data layer to save the data...

    return result;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Complex Operations

```
public (bool Success, string Message) SavePrice(int productId,
                                                string cost, string price,
                                                string category, string reason,
                                                DateTime effectiveDate)
{
    // Validate arguments
    // Call a method in the data layer to save the data...

    return (Success: true, Message: "Price saved successfully");
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Complex Operations

```
public OperationResult SavePrice(int productId,
                                 string cost, string price,
                                 string category, string reason,
                                 DateTime effectiveDate)
{
    // Validate arguments
    // Call a method in the data layer to save the data...

    return new OperationResult() { Success = true,
                                   Message ="Price saved successfully"};
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Complex Operations

```
public void SavePrice(int productId,
                      string cost, string price,
                      string category, string reason,
                      DateTime effectiveDate)
{
    // Validate arguments
    // Call a method in the data layer to save the data...
}
```

Command-query separation (CQS) principle states:

"Every method should either be a command that performs an action, or a query that returns data to the caller, but not both."

Wikipedia

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Complex Operations

```
public void SavePrice(int productId,
                      string cost, string price,
                      string category, string reason,
                      DateTime effectiveDate)
{
    // Validate arguments
    // Call a method in the data layer to save the data...
}
```

Returning Predictable Results

Value

Nullable Value

```
public OperationResult ValidateEffectiveDate(DateTime? effectiveDate)
```

ref/out tuple

```
public (decimal? Margin, string? Message) CalculateMargin(string costInput,
                                                          string priceInput)
```

Object

Nullable Reference

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

Void

```
public OperationResult SavePrice(int productId,
                                 string cost, string price,
                                 string category, string reason,
                                 DateTime effectiveDate)
```

Exception

Guidelines and Summary

Return a Value

- Value
- Nullable Value
- ref/out tuple
- Object
- Nullable Reference
- Void
- Exception

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

```
public bool ValidateEffectiveDate(DateTime? effectiveDate)
{
   // ...
   return true;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Return a Nullable Value

```
public decimal? CalculateMargin(string costInput, string priceInput)
{
  if (string.IsNullOrWhiteSpace(costInput)) return null;

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

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Use a **ref** Parameter

```
public bool ValidateEffectiveDate(DateTime? effectiveDate,
                                  ref string validationMessage)
{
    if (!effectiveDate.HasValue) return false;

    if (effectiveDate.Value < DateTime.Now.AddDays(7))
    {
      validationMessage = "Date must be >= 7 days from today";
      return false;
    }
    return true;
}
```

```
var message = "";
var isValid = product.ValidateEffectiveDate(effectiveDate,
                                            ref message);
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Use an **out** Parameter

```
public bool ValidateEffectiveDate(DateTime? effectiveDate,
                                  out string validationMessage)
{
    validationMessage = "";
    if (!effectiveDate.HasValue) return false;

    if (effectiveDate < DateTime.Now.AddDays(7))
    {
      validationMessage = "Date must be >= 7 days from today";
      return false;
    }
    return true;
}
```

```
var isValid = product.ValidateEffectiveDate(effectiveDate,
                                            out string message);
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Return a Tuple

```
public (bool IsValid, string Message) ValidateEffectiveDate(DateTime? effectiveDate)
{
  if (!effectiveDate.HasValue)
                return (IsValid: false, Message: "Date has no value");

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

  return (IsValid: true, Message: "");
}
```

```
var result = product.ValidateEffectiveDate(effectiveDate);

if (!result.IsValid) // ... display result.Message;
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Return an Object

```
public class OperationResult
{
  public bool Success { get; set; }
  public string Message { get; set; }
  // ...
}
```

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

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Return an Object

```
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;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Return a Nullable Reference

```
public Discount? FindDiscount(List<Discount>? discounts, string name)
{
  if (discounts is null) return null;

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

  return foundDiscount;
}
```

Value

Nullable Value

ref/out tuple

Object

Nullable Reference

Void

Exception

Don’t Return a Value

```
public void SavePrice(int productId,
                      string cost, string price,
                      string category, string reason,
                      DateTime effectiveDate)
{
    // Validate arguments
    // Call a method in the data layer to save the data...
}
```