Creating Good Methods

Deborah Kurata

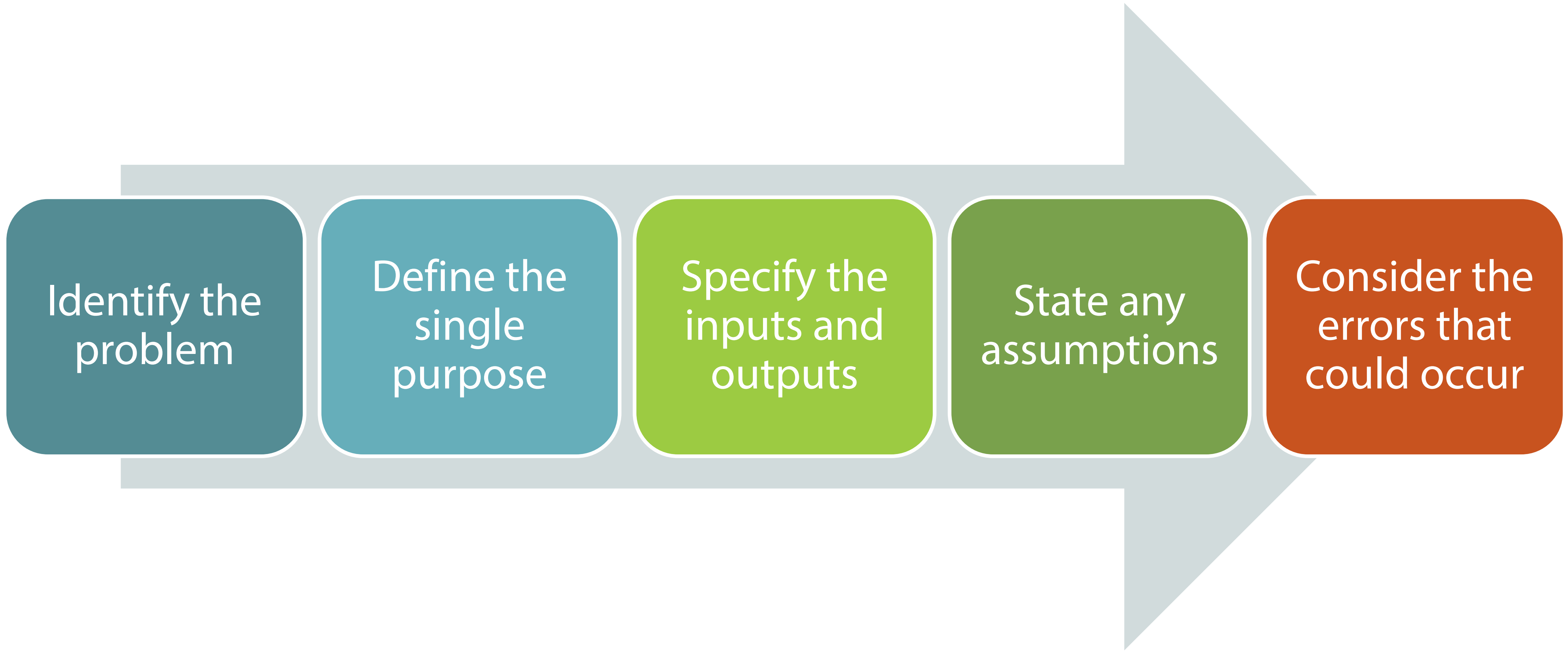
@deborahkurata | blogs.msmvps.com/deborahk/

Method

```
public bool PlaceOrder(Product product, int quantity)
{
    // Code here
}
```

- A code block
- Contains a set of programming statements
- Also called a function
- Purpose: To implement the logic required for specific behavior or functionality in a class

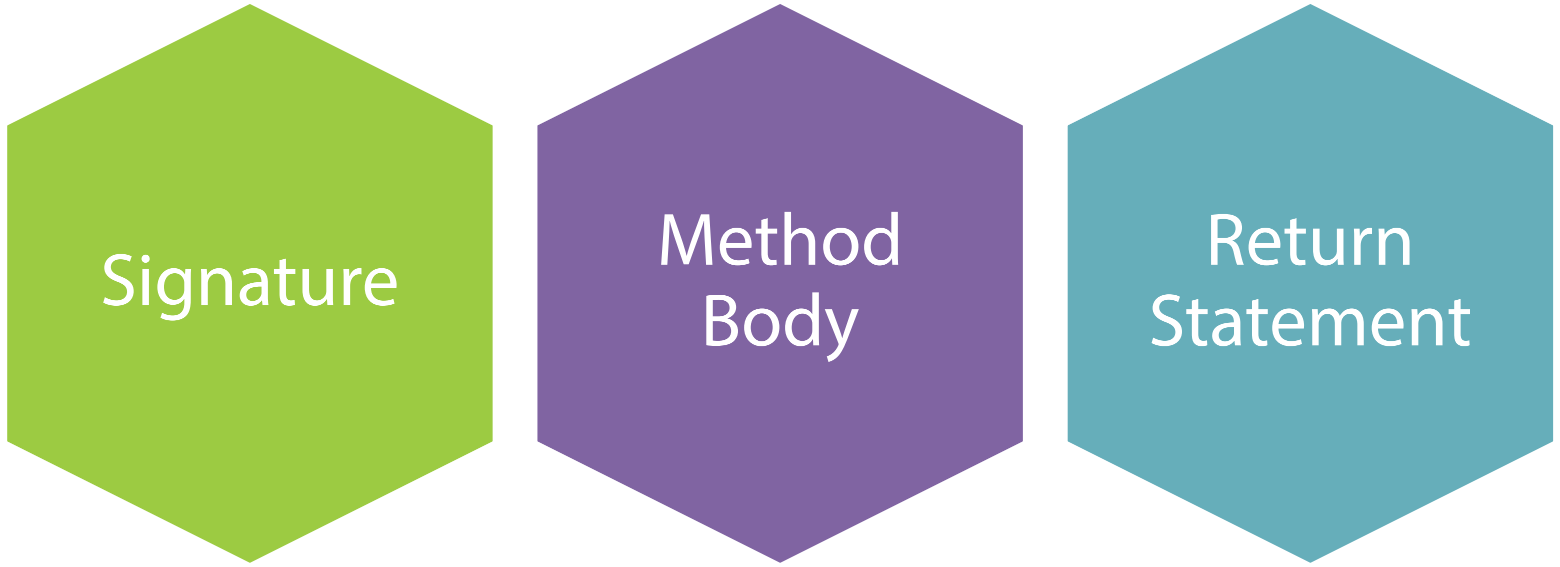
Defining a Method


Identify the problem
Define the single purpose
Specify the inputs and outputs
State any assumptions
Consider the errors that could occur

Module Overview

- Building a Method
- Property or Method?
- Method Overloading
- Method Chaining
- Method Overriding
- Expression-Bodied Methods
- FAQ

Building a Method



Method Signature

```
public bool PlaceOrder(Product product, int quantity)
```

- Optional accessibility modifier
  - Default is private

Method Signature

```
public bool PlaceOrder(Product product, int quantity)
```

- Optional accessibility modifier
  - Default is private
- Return type
  - `void` if no return value
- Method name
- Parameter list
  - Empty parenthesis if no parameters

XML Document Comment

```
/// <summary>
/// Sends a product order to the vendor.
/// </summary>
/// <param name="product">Product to order.</param>
/// <param name="quantity">Quantity of the product to order.</param>
/// <returns></returns>
public bool PlaceOrder(Product product, int quantity)
```

- summary for method purpose
- param for a description of each parameter

Method Signature Best Practices

Do:

Naming

- Define a meaningful name
- Use a verb
- Use PascalCasing

Define the most restrictive accessibility possible

Keep the number of parameters to a minimum

Define an XML document comment

Avoid:

Naming

- Vague terms
- Abbreviations
- Conjunctions
- Inconsistent naming

Method Body

```
public bool PlaceOrder(Product product, int quantity)
{
   if (product == null) throw new ArgumentNullException(nameof(product));
   if (quantity <= 0) throw new ArgumentOutOfRangeException(nameof(quantity));

   var success = false;

   var orderText = "Order from Acme, Inc" + System.Environment.NewLine +
                   "Product: " + product.ProductCode + System.Environment.NewLine +
                   "Quantity: " + quantity;

   var emailService = new EmailService();
   var confirmation = emailService.SendMessage("New Order", orderText, this.Email);
   if (confirmation.StartsWith("Message sent:"))
   {
      success = true;
   }
   return success;
}
```

Method Return Value

```
public bool PlaceOrder(Product product, int quantity)
{
   var success = false;
   ...
   return success;
}
```

- Signature defines the type of the return value
  - Return statement must return that type
- Use a type of **`void`** if the method does not return a value
  - The return statement can be omitted.

Return Multiple Values

```
public bool PlaceOrder(Product product, int quantity)
{
   if (product == null) throw new ArgumentNullException(nameof(product));
   if (quantity <= 0) throw new ArgumentOutOfRangeException(nameof(quantity));

   var success = false;

   var orderText = "Order from Acme, Inc" + System.Environment.NewLine +
                   "Product: " + product.ProductCode + System.Environment.NewLine +
                   "Quantity: " + quantity;

   var emailService = new EmailService();
   var confirmation = emailService.SendMessage("New Order", orderText, this.Email);
   if (confirmation.StartsWith("Message sent:"))
   {
      success = true;
   }
   return success;
}
```

```
public class OperationResult
{
   public OperationResult()
   {}

   public OperationResult(bool success, string message) : this()
   {
      this.Success = success;
      this.Message = message;
   }

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

```
public OperationResult PlaceOrder(Product product, int quantity)
{
   ...
   var operationResult = new OperationResult(success, orderText);
   return operationResult;
}
```

Method Body Best Practices

Do:

Keep methods short

Avoid:

The first rule of functions is that they should be small. The second rule of functions is that they should be smaller than that. Functions should not be 100 lines long. Functions should hardly ever be 20 lines long.

— Robert Martin

"Clean Code: A Handbook of Agile Software Craftsmanship"

Method Body Best Practices

Do:

Keep methods short

Use white space

Use guard clauses

Return an expected result
Use an object to return multiple values

Implement exception handling

Avoid:

`void` methods

Property or Method

ProductName
Description

Property

- ProductName
- Description

Method

Property or Method

Place Order

Property

- ProductName
- Description

Method

- PlaceOrder()

Property or Method

Inventory Count

Property

- ProductName
- Description
- InventoryCount?

Method

- PlaceOrder()
- CalculateInventoryCount()?

Property or Method

Suggested Price

Property

- ProductName
- Description
- InventoryCount?
- SuggestedPrice?

Method

- PlaceOrder()
- CalculateInventoryCount()?
- CalculateSuggestedPrice()?

Property or Method

Property

- Does it describe data?
- Does it execute quickly?

Method

- Does it describe processing?
- Does it produce side effects?
- Does it require parameters?

Method Overloading

```
public bool PlaceOrder(Product product, int quantity)

public bool PlaceOrder(Product product, int quantity,
                           DateTimeOffset deliveryBy)

public bool PlaceOrder(Product product, int quantity,
       DateTimeOffset deliveryBy, string instructions)

public void PlaceOrder(Product product, int quantity)
```

Method Overloading Best Practices

Do:

Keep the number of parameters to a minimum

Keep the order of the parameters consistent

Define an XML document comment for each overload

Consider optional parameters

Avoid:

Confusing overloads

Overloads that differ in purpose

Example: OrderItems

One overload gets the ordered items

Second overload orders a set of items

Duplicating code

Method Chaining

```
public OperationResult PlaceOrder(Product product, int quantity)
{
   return PlaceOrder(product, quantity, null, null);
}

public OperationResult PlaceOrder(Product product, int quantity,
                                  DateTimeOffset? deliverBy)
{
   return PlaceOrder(product, quantity, deliverBy, null);
}

public OperationResult PlaceOrder(Product product, int quantity,
                                   DateTimeOffset? deliverBy, string instructions)
{
 // All of the code here
}
```

Method Chaining Best Practices

Do:

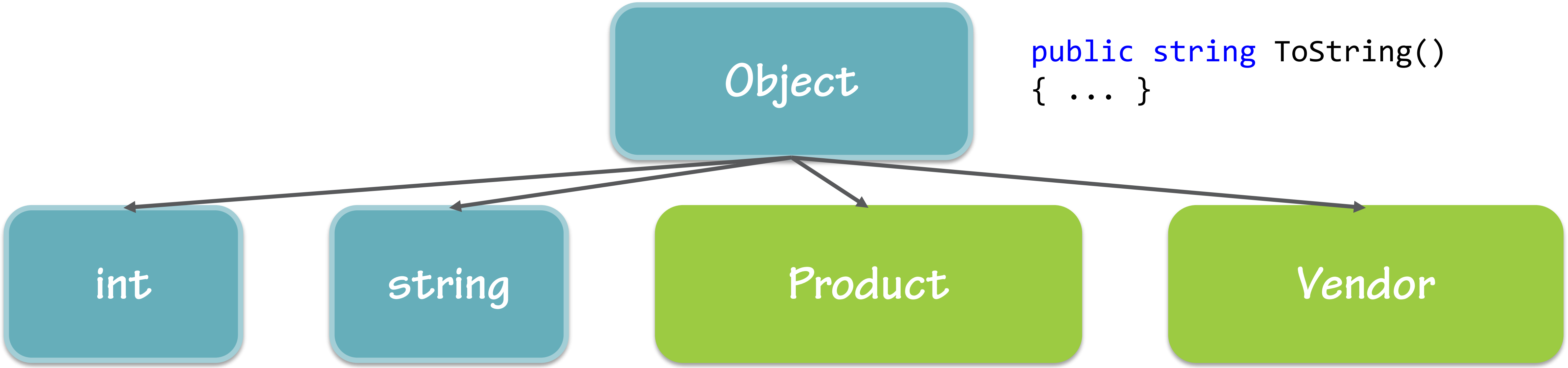
Use to minimize repeated code in method overloads

Consider optional parameters

Avoid:

If it adds complexity

Method Overriding



```
public string ToString()
{ ... }
```

```
public override string ToString()
{
    return this.ProductName + " (" +
           this.productId + ")";
}
```

Method Overriding Best Practices

Do:

Override ToString for each entity class

Avoid:

Expression-Bodied Methods

```
public decimal CalculateSuggestedPrice(decimal markupPercent)
{
   return this.Cost + (this.Cost * markupPercent / 100);
}
```

```
public decimal CalculateSuggestedPrice(decimal markupPercent) =>
                 this.Cost + (this.Cost * markupPercent / 100);
```

- Syntax Shortcut
- Single statement methods
- That return a value
- No curly braces
- No return statement
- Just a =>

Expression-Bodied Method Best Practices

Do:

Use it for very simple methods

Avoid:

Using it when there should be guard clauses

Using it when there should be exception handling

Frequently Asked Questions

- What is the primary purpose of a method?
  - To implement the logic required for specific behavior or functionality in a class
- What is the difference between a parameter and an argument?
  - A parameter is part of the method signature
  - An argument is part of the method call
- What is method overloading?
  - Methods with the same name and purpose but different signatures
- What is method chaining?
  - One method overload calls another overload to prevent repeated code

Frequently Asked Questions (cont)

- When is it best to use method overloading vs. method overriding?
  - Use overloading when one method requires multiple signatures
    - Such as a GetCustomer(id) to get a customer by Id and GetCustomer(name) to get the customer by name
  - Use overriding when replacing a method defined higher up the object hierarchy.
    - Such as replacing the ToString() method
- What is an expression-bodied method?
  - A syntax shortcut for single statement methods that return a value

This Module Covered

Building a Method

Property or Method?

Method Overloading

Method Chaining

Method Overriding

Expression-Bodied Methods