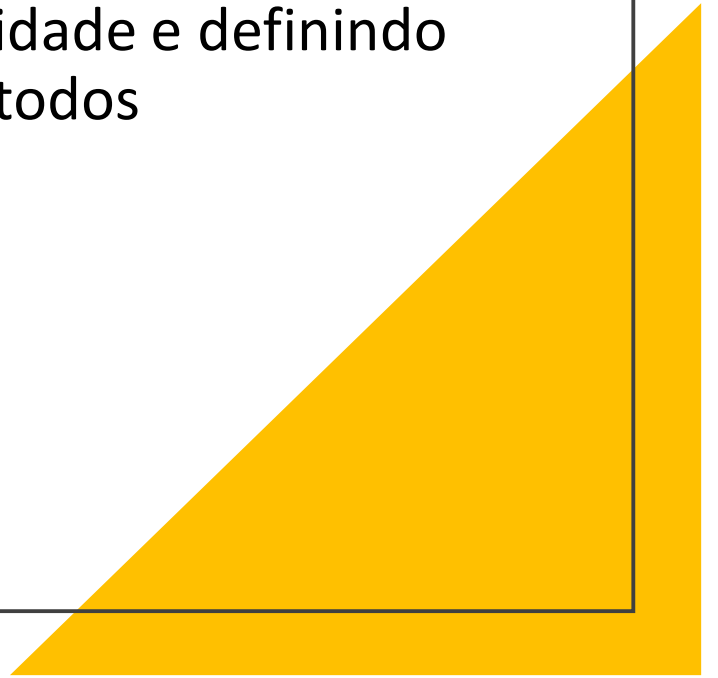


Programação OO

Construindo as Classes
Entidade e definindo
Métodos



Customer

- Name
- Email address
- Home address
- Work address
- Validate()
- Retrieve()
- Save()

Product

- Product name
- Description
- Current price
- Validate()
- Retrieve()
- Save()

Order

- Customer
- Order date
- Shipping address
- Order items
- Validate()
- Retrieve()
- Save()

Order Item

- Product
- Quantity
- Purchase price
- Validate()
- Retrieve()
- Save()

Classes
Identificadas

0 referências

```
public bool Validate()  
{  
    var isValid = true;  
    if (string.IsNullOrEmpty(LastName)) isValid = false;  
    if (string.IsNullOrEmpty(EmailAddres)) isValid = false;  
    return isValid;  
}
```

Construindo uma Classe com Métodos

- Adicione o método Validate() a classe Customer

```
public Customer Retrieve(int customerId)
{
    // Code that retrieves the defined customer

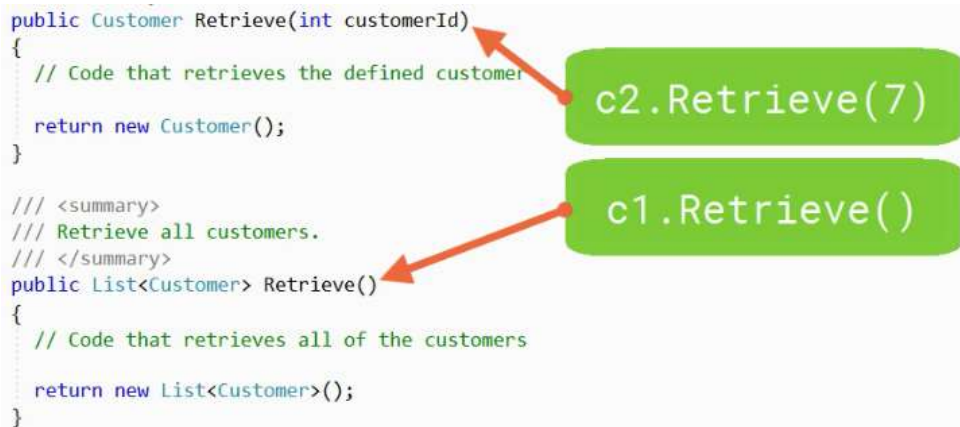
    return new Customer();
}
```

```
public Customer Retrieve(int customerId)
{
    // Code that retrieves the defined customer

    return new Customer();
}

/// <summary>
/// Retrieve all customers.
/// </summary>
public List<Customer> Retrieve()
{
    // Code that retrieves all of the customers

    return new List<Customer>();
}
```



Termos: ASSINATURA DO MÉTODO

- Composto pelo nome e conjunto de parâmetros e seus tipos;
- Não inclui o tipo de retorno;
- Utilizado pelo sistema para orientar as chamadas de funções;
- Cada assinatura de um método na classe deve ser único;
- Mas se possuímos métodos com parâmetros diferentes eles podem ter o mesmo nome;

```
public Customer Retrieve(int customerId)
{
    // Code that retrieves the defined customer

    return new Customer();
}

/// <summary>
/// Retrieve all customers.
/// </summary>
public List<Customer> Retrieve()
{
    // Code that retrieves all of the customers

    return new List<Customer>();
}
```

Termos: SOBRECARGA/OVERLOADING

- Usado para descrever métodos que possuem o mesmo nome com diferentes parâmetros;
- Métodos que são especificados com o mesmo nome, devem prover variações da mesma funcionalidade;
- No exemplo, um método retorna um Customer, enquanto o outro retorna uma coleção de consumidores List<Customer>;
- Utilize void quando não houver nada a retornar;

```
public int CustomerId { get; private set; }
public string EmailAddress { get; set; }

public string FirstName { get; set; }

public string FullName...

public static int InstanceCount { get; set; }

private string _lastName;
public string LastName...

/// <summary> Retrieve one customer.
public Customer Retrieve(int customerId)...

/// <summary> Retrieve all customers.
public List<Customer> Retrieve()...

/// <summary> Saves the current customer.
public bool Save()...

/// <summary> Validates the customer data.
public bool Validate()...
```

Termo: CONTRACT

- As especificações dos modificadores de acesso definem o contrato da classe. Quais atributos e métodos estarão visíveis, para quem e em que nível de acesso.

Construtores

```
public Customer()  
{  
    ...  
}  
public Customer(int customerId)  
{  
    customerId = customerId;  
}
```

- Um tipo especial de método com o mesmo nome da classe;
- Geralmente definidos no primeiro parágrafo da classe;
- É o método executado quando um objeto do tipo da classe é criado;
- Um método construtor sem parâmetros é definido como construtor padrão (default constructor);
- Podemos fazer a sobrecarga (overloading) de construtores;

```
public Customer Retrieve(int customerId)
```

Signature

```
public Customer Retrieve(int customerId)  
public List<Customer> Retrieve()
```

Overloading

- ⊗ Customer()
- ⊗ Customer(int)
- ✎ CustomerId : int
- ✎ EmailAddress : string
- ✎ FirstName : string
- ✎ FullName : string
- ✎ LastName : string
- ⊗ Save() : bool
- ⊗ Retrieve(int) : Customer
- ⊗ Retrieve() : List<Customer>
- ⊗ Validate() : bool

Contract

```
public Customer() { }  
public Customer(int customerId) { }
```

Constructor

Termos

Mais Classes

- Agora adicione outras classes ao seu projeto UNOESC.BL que está em sua solução UNOESC no Visual Studio 2022:
 - Product.cs
 - Order.cs
 - OrderItem.cs



Classe Product

```
/// <summary>
/// Retrieve one product.
/// </summary>
public Product Retrieve(int productId)
{
    // Code that retrieves the defined product

    return new Product();
}

/// <summary>
/// Saves the current product.
/// </summary>
/// <returns></returns>
public bool Save()
{
    // Code that saves the defined product

    return true;
}
```

```
public class Product
{
    public Product()
    {
    }
    public Product(int productId)
    {
        ProductId = productId;
    }

    public decimal? CurrentPrice { get; set; }
    public string ProductDescription { get; set; }
    public int ProductId { get; private set; }
    public string ProductName { get; set; }
}
```

```
/// <summary>
/// Validates the product data.
/// </summary>
/// <returns></returns>
public bool Validate()
{
    var isValid = true;

    if (string.IsNullOrEmpty(ProductName)) isValid = false;
    if (CurrentPrice == null) isValid = false;

    return isValid;
}
```

Classe Order

```
public class Order
{
    public Order()
    {
    }
    public Order(int orderId)
    {
        OrderId = orderId;
    }

    public DateTimeOffset? OrderDate { get; set; }
    public int OrderId { get; private set; }
```

```
    /// <summary>
    /// Retrieve one order.
    /// </summary>
    public Order Retrieve(int orderId)
    {
        // Code that retrieves the defined order

        return new Order();
    }

    /// <summary>
    /// Saves the current order.
    /// </summary>
    /// <returns></returns>
    public bool Save()
    {
        // Code that saves the defined order

        return true;
    }
```

```
    /// <summary>
    /// Validates the order data.
    /// </summary>
    /// <returns></returns>
    public bool Validate()
    {
        var isValid = true;

        if (OrderDate == null) isValid = false;

        return isValid;
    }
```

Classe OrderItem

```
public class OrderItem
{
    public OrderItem()
    {
    }
    public OrderItem(int orderId)
    {
        OrderItemId = orderId;
    }

    public int OrderItemId { get; private set; }
    public int ProductId { get; set; }
    public decimal? PurchasePrice { get; set; }
    public int Quantity { get; set; }
}
```

```
/// <summary>
/// Retrieve one order item.
/// </summary>
public OrderItem Retrieve(int orderId)
{
    // Code that retrieves the defined order item

    return new OrderItem();
}

/// <summary>
/// Saves the current order item.
/// </summary>
/// <returns></returns>
public bool Save()
{
    // Code that saves the defined order item

    return true;
}
```

```
/// <summary>
/// Validates the order item data.
/// </summary>
/// <returns></returns>
public bool Validate()
{
    var isValid = true;

    if (Quantity <= 0) isValid = false;
    if (ProductId <= 0) isValid = false;
    if (PurchasePrice == null) isValid = false;

    return isValid;
}
```

Classes Definidas

Agora que suas classes foram definidas, é hora de pensar em dividir para conquistar.

Não acha que estas classes foram especificadas com muitas responsabilidades?

- Especificar os atributos;
- Salvar
- Consultar

Agora é hora de separar as responsabilidades!