UNOESC

Prof. Mauricio Roberto Gonzatto

Reutilização por meio da Herança

Reutilização de código

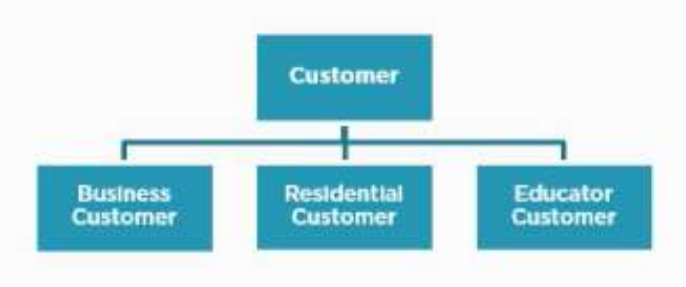
- É a quarta tarefa chave do Paradigma Orientado a Objeto;
  - 1) Identificar as Classes
  - 2) Separar as Responsabilidades
  - 3) Estabelecer Relacionamentos
  - 4) Aproveitar da Reutilização

Reutilização de Código

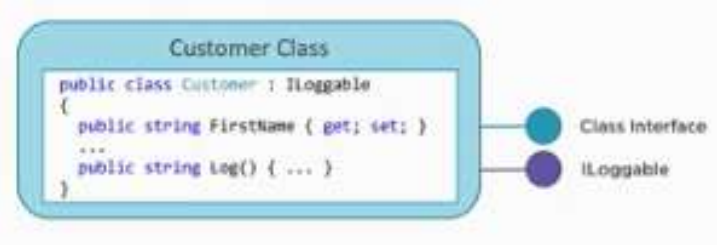
- Envolve o processo de extrair as especificações em comum entre as classes.
- Processo de construção de classes reutilizáveis e componentes.
- Definir Interfaces.

1

2



4



3

5

Técnicas de Reaproveitamento

- 1) Colaboração / Composição
- 2)Herança
- 3) Componentes
- 4) Interfaces
- 5) Copiar/Colar

Segredos da Reusabilidade de Software

- Construa uma única vez;
- Teste;
- Reutilize a classe, componente ou interface;
- Atualize em apenas um local;

Vantagens da Reusabilidade

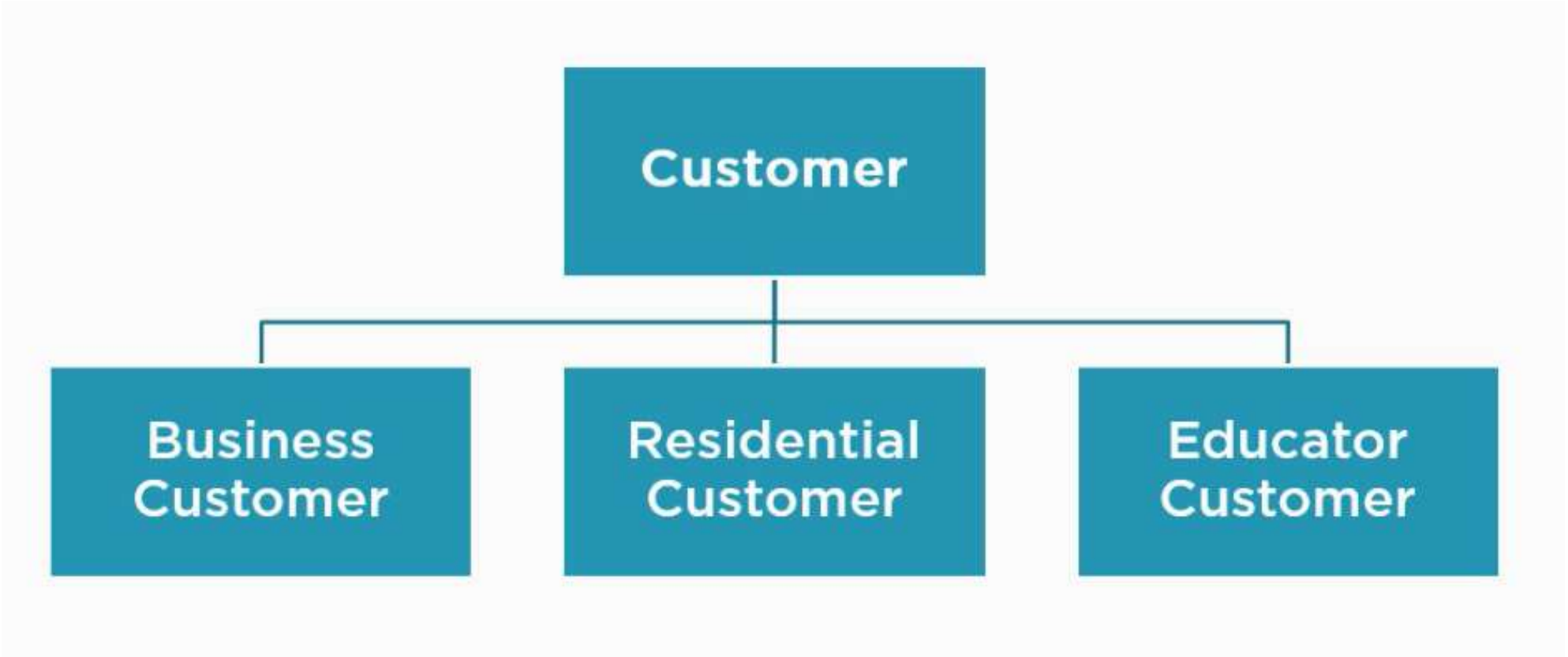
Reduz a quantidade de código;

Reduz o tempo de desenvolvimento;

Reduz custos;

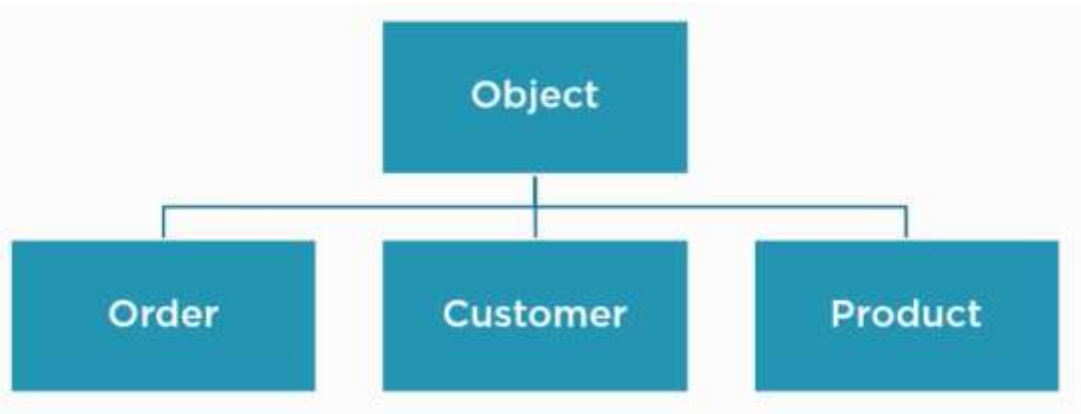
Reduz bugs;

Herança


Customer
Business Customer
Residential Customer
Educator Customer

.NET Object Class

- O conceito de herança é utilizado implicitamente dentro da plataforma .NET.
- Todas as classes herdam de uma super-classe, chamada Object.
- Ou seja, toda as especificações da super-classe Objetct estão disponíveis em quaisquer classe-derivada.



Herança implícita de Object

Acesse a classe ProductRepository.cs e faça o teste indicado no método Retrieve();

Observe que o método ToString() é um comportamento de Object, que é herdado automaticamente por Product.

```
namespace ACM.BL
{
    3 referências
    public class ProductRepository
    {
        /// <summary> Retorna um produto
        1 referência
        public Product Retrieve(int productId)
        {
            Product product = new Product(productId);

            if (productId == 2)
            {
                product.ProductName = "Vodka";
                product.ProductDescription = "Vodka Smirnoff 1L";
                product.CurrentPrice = 15.96M;
            }

            //Herança: Observe a herança de Object->Product->ToString()
            Object myObject = new Object();
            Console.WriteLine($"Object: {myObject.ToString()}");
            Console.WriteLine($"Product: {product.ToString()}");


            return product;
        }
```

Sobrescrevendo Funcionalidades de Classes Base

Acesse a classe Product.cs e implemente uma sobrecarga para o método ToString();

```
1 referência
public override string ToString() => ProductName;

/// <summary> Valida propriedades do produto
2 referências
public override bool Validate()
{
    var isValid = true;

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

    return isValid;
}
```

Sobrescrevendo Funcionalidades de Classes Base

Faça o mesmo na Classe Order.cs

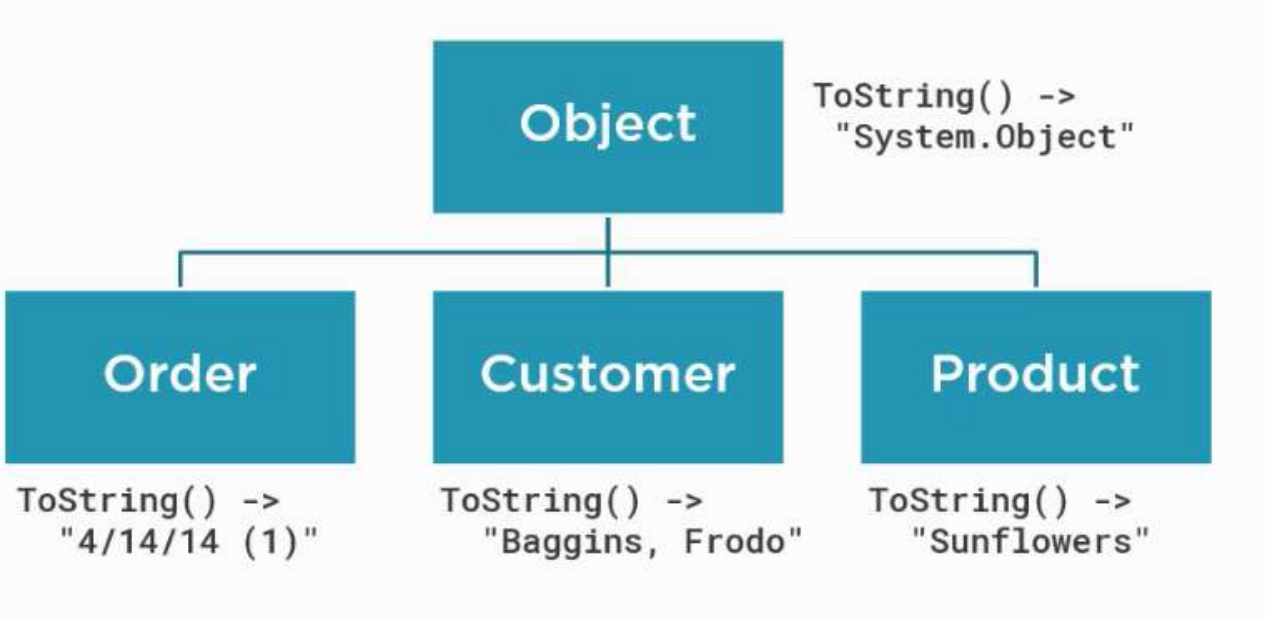
```
// Herança -> Sobrescrita de métodos
// Aqui também vemos a atuação do polimorfismo, já que o objeto herda de Object, cada um dos objetos
// que implementa este método terá um comportamento específico para sua instância.
0 referências
public override string ToString()
{
    return $"{OrderDate.Value.Date} ({OrderId})";
}

0 referências
public bool Validate()
{
    var isValid = true;

    if (OrderDate == null) isValid = false;

    return isValid;
}
```

Sobrescrevendo Funcionalidades de Classes Base

O mesmo em Customer.cs

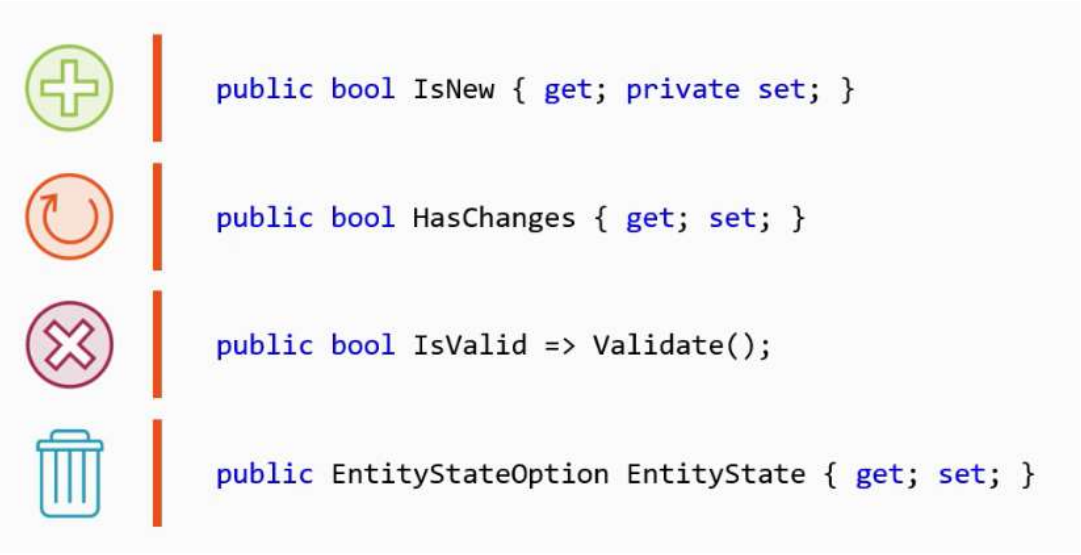
```
0 referências
public override string ToString() => FullName;

/// <summary>
/// Validação dos dados do cliente
/// </summary>
/// <returns>Válido?</returns>
1 referência | 1/1 passando
public bool Validate()
{
    var isValid = true;

    if( string.IsNullOrWhiteSpace(LastName) ) isValid = false;
    if( string.IsNullOrWhiteSpace(EmailAddress) ) isValid = false;

    return isValid;
}
```

Polimorfismo



- Basicamente, polimorfismo quer dizer "muitas camadas";
- É o conceito de que um método pode ter um comportamento diferente dependendo do objeto que o invocou;
- Observe a implementação que acabamos de fazer em Order, Customer e Product. O método ToString() terá um comportamento diferente para cada invocação específica;

Classe Base

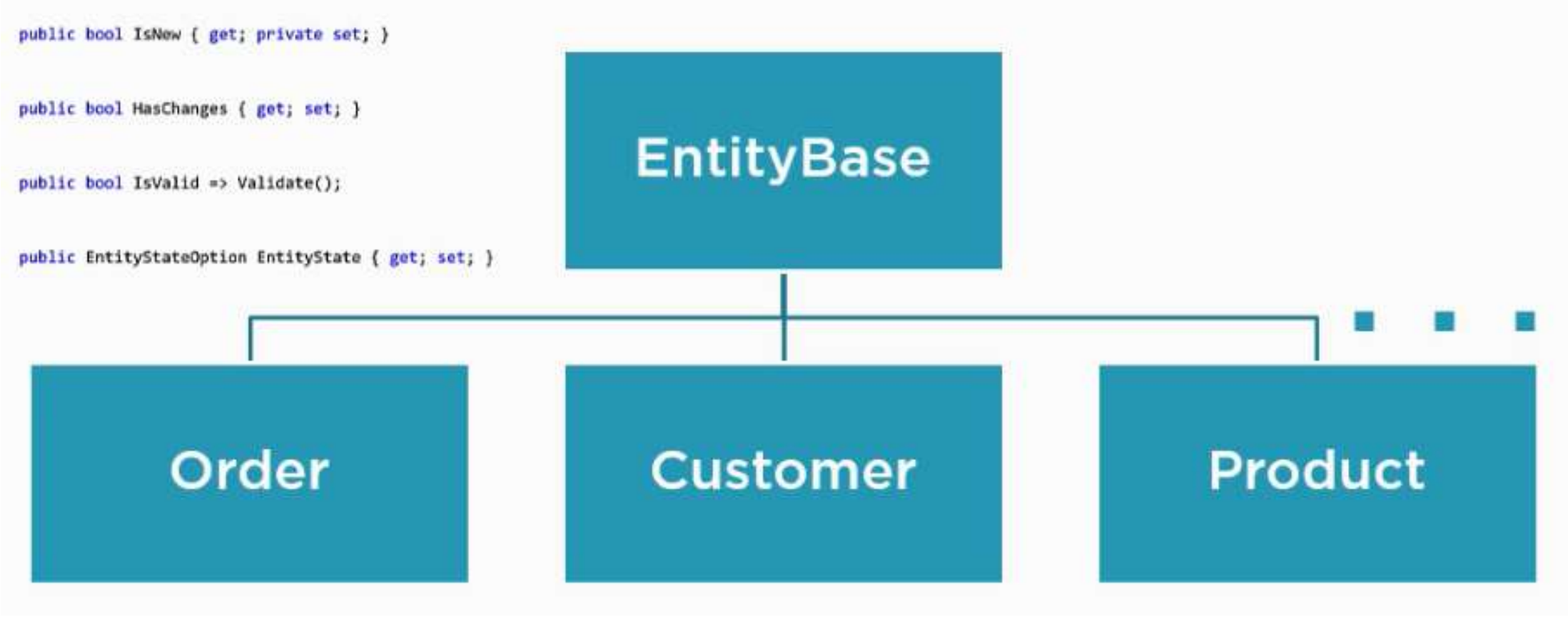


- Imagine que precisamos incluir um conjunto de atributos a todas as nossas classes com o objetivo de criar uma funcionalidade de Salvar Dados.
- Seria uma boa prática repetir estas propriedades em todas Entidades do projeto?
  - Acho que não!
- Para isto definimos uma EntidadeBase como Classe Base para as demais.

Classe Base

Desta forma, todas as entidades hedariam as novas propriedades.

```
public bool IsNew { get; private set; }

public bool HasChanges { get; set; }

public bool IsValid => Validate();

public EntityStateOption EntityState { get; set; }
```



Tipos de Classe Base

- Abstratas
  - Classe incompleta com ao menos uma propriedade ou método sem implementação;
  - Não pode ser instanciada. Não utiliza o operador new;
  - Propósito de uso como classe base;
  - Especificada com a palavra reservada abstract;
- Concretas
  - Classe "normal";
  - Pode ser instanciada. Utiliza o operador new;
  - Pode ser utilizada como classe base;
  - Não utiliza a palavra reservada abstract em sua especificação;
- Seladas
  - Não pode ser extendida através de herança;
  - Utiliza a palavra reservada sealed em sua especificação;

Criando uma Classe Base

- Abra o Gerenciador de Soluções
- Clique com o botão direito na Biblioteca de Classes BL
- Adicione uma nova classe chamada EntityBase
- Defina a classe como public e abstract
- Implemente as especificações da classe definindo o método Validate() como abstract

```
namespace ACM.BL
{
    1 referência
    public enum EntityStateOption
    {
        Active,
        Deleted
    }

    1 referência
    public abstract class EntityBase
    {
        0 referências
        public EntityStateOption EntityState { get; set; }
        3 referências | 2/2 passando
        public bool HasChanges { get; set; }
        1 referência
        public bool IsNew { get; private set; }
        1 referência
        public bool IsValid => Validate();

        2 referências
        public abstract bool Validate();
    }
}
```

Herdando de EntityBase

- Altere a classe Product.cs e defina a herança de EntityBase
- Sobrescreva o método Validate*(

```
using System.ComponentModel.DataAnnotations.Schema;

namespace ACM.BL
{
    16 referências
    public class Product : EntityBase
    {
        1 referência
        public Product()...

        4 referências | 2/3 passando
        public Product(int produtctId)...

        //----------

        [Column(TypeName = "decimal(8,2)")]
        9 referências | 2/3 passando
        public decimal? CurrentPrice { get; set; } //? = nullable type possibilita valor nulo
        8 referências | 2/3 passando
        public string ProductDescription { get; set; }
        1 referência
        public int ProductId { get; private set; }

        private string _productName;
```

```
        10 referências | 2/3 passando
        public string ProductName
        {
            get { return _productName; }
            set { _productName = value; }
        }

        //----------

        1 referência
        public override string ToString() => ProductName;

        /// <summary> Valida propriedades do produto
        2 referências
        public override bool Validate()
        {
            var isValid = true;

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

            return isValid;
        }
    }
}
```

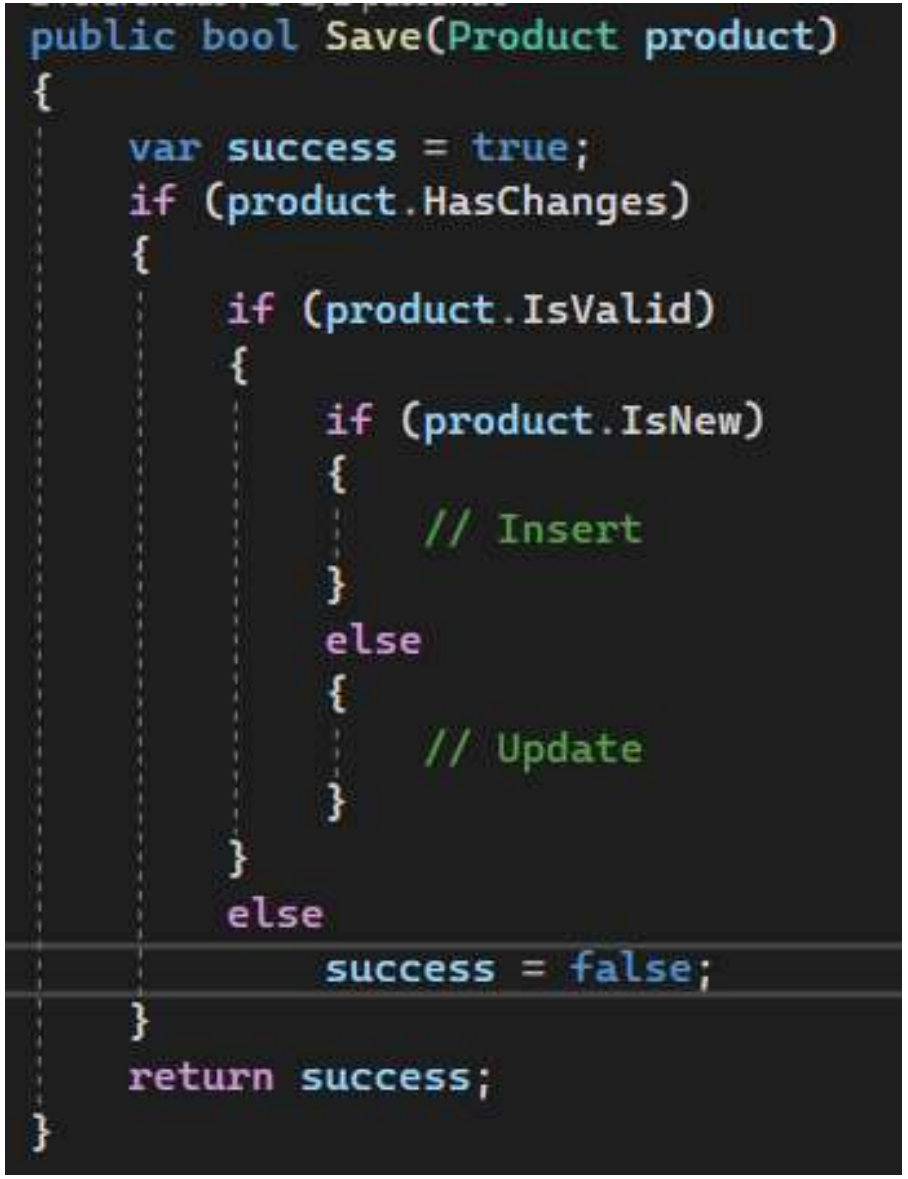
```
//Herança: Observe a herança de Object->Product->ToString()
Object myObject = new Object();
Console.WriteLine($"Object: {myObject.ToString()}");
Console.WriteLine($"Product: {product.ToString()}");

Console.WriteLine($"Product: {product.EntityState.ToString()}");
Console.WriteLine($"Product: {product.HasChanges}");
Console.WriteLine($"Product: {product.IsValid}");
```

Herdando de EntityBase

- Agora, em ProductRepository.cs você consegue verificar que o objeto product consegue acessar suas novas propriedades!

Preparando Membros Sobrescrevíveis em Classes Base

- Podemos fazer de 2 formas:
- **Abstract**
  - Assinatura do método apenas como definição, sem implementação;
  - Utilizado apenas em classes abstratas;
  - DEVE ser sobrescrito/implementado por uma classe derivada;
  - Utiliza a palavra chave abstract;
- **Virtual**
  - Método com uma implementação padrão;
  - Utilizado em classes concretas ou abstratas;
  - PODE ser sobrescrito/re-implementado por uma classe derivada;
  - Utiliza a palavra chave virtual;



```
public bool Save(Product product)
{
    var success = true;
    if (product.HasChanges)
    {
        if (product.IsValid)
        {
            if (product.IsNew)
            {
                // Insert
            }
            else
            {
                // Update
            }
        }
        else
            success = false;
    }
    return success;
}
```

Implemente o método Save() em ProductRepository.cs

- Acesse o arquivo ProductRepository.cs e implemente o método Save()

Implemente os Testes para ProductRepository

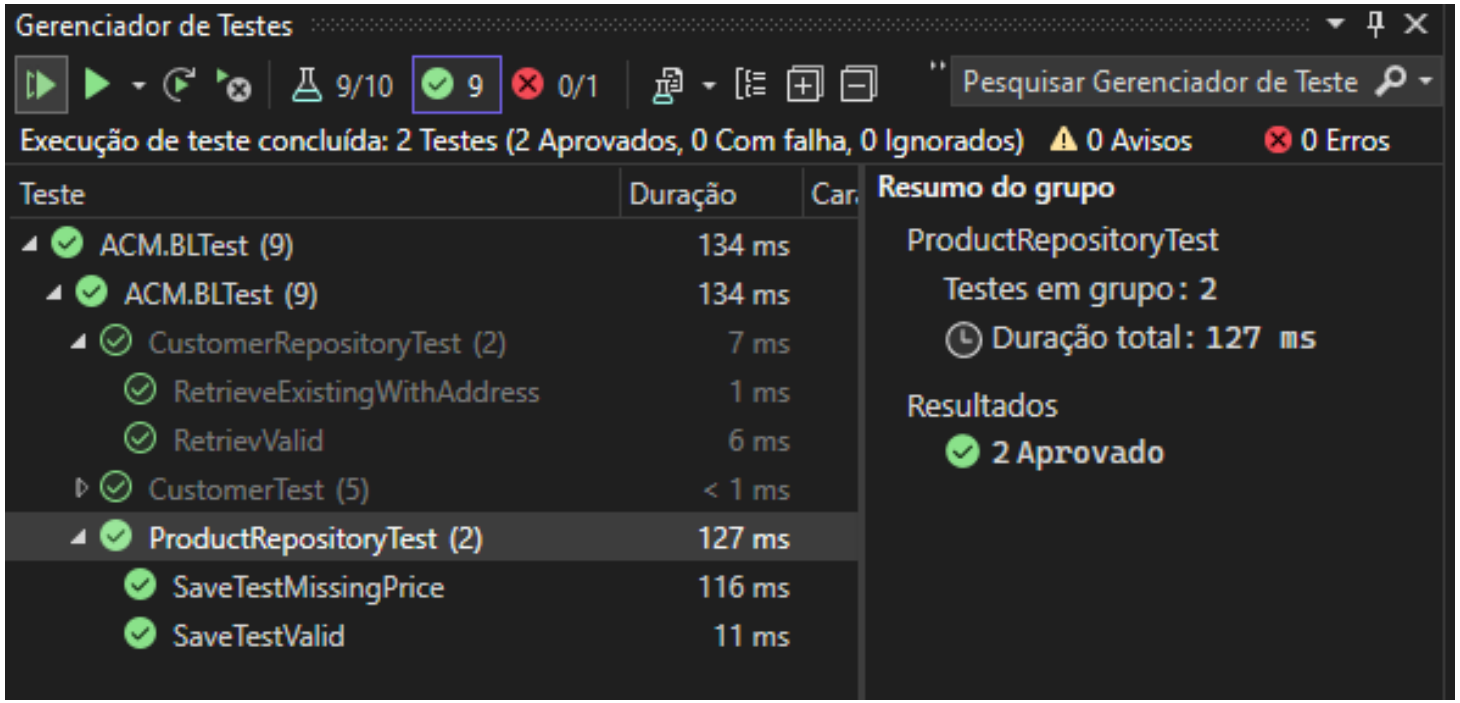
- Implemente os métodos SaveTestValid() e SaveTestMissingPrice() em ProductRepository.cs

```
[TestMethod()]
| 0 referências
public void SaveTestMissingPrice()
{
    //-- Arrange
    var productRepository = new ProductRepository();
    var updatedProduct = new Product(2)
    {
        CurrentPrice = null,
        ProductDescription = "Pequeno conjunto com 4 flores de girassol amarelas",
        ProductName = "Flor de Girassol",
        HasChanges = true
    };

    //-- Act
    var actual = productRepository.Save(updatedProduct);

    //-- Assert
    Assert.AreEqual(false, actual);
}
```

```
[TestMethod()]
| 0 referências
public void SaveTestValid()
{
    //-- Arrange
    var productRepository = new ProductRepository();
    var updatedProduct = new Product(2) {
        CurrentPrice = 18M,
        ProductDescription = "Pequeno conjunto com 4 flores de girassol amarelas",
        ProductName = "Flor de Girassol",
        HasChanges = true
    };

    //-- Act
    var actual = productRepository.Save(updatedProduct);

    //-- Assert
    Assert.AreEqual(true, actual);
}
```

Teste!

- Abra o Gerenciador de Testes
- Clique com o botão direito em ProductRepositoryTest e execute
- Verifique o resultado



Pilares da Programação OO