SOLID
Princípios

Exemplos

por Felipe Arantes

Single Responsibility Principle

```
public void AddUser(string username)
{
    //Validate
    if ( username == "Admin" )
        throw new InvalidOperationException();

    //Create a User
    var connection = new SqlConnection();
    connection.Open();
    var command = new SqlCommand("INSERT INTO...");

    //Send Email
    var client = new SmtpClient("host");
    client.Send(new MailMessage());
}
```

```
public void AddUser (string username)
{
    var user = UserDomain.Create(username);
    if ( user.IsNotValid )
        throw new InvalidOperationException();

    _userRepository.Insert(user);
    _emailService.Send(user);
}
```

Open-closed Principle

```
private double Area(object[] shapes)
{
    double area = 0;

    foreach ( var shape in shapes )
    {
        if ( shape is Rectangle )
        {
            var rectangle = (Rectangle)shape;
            area += rectangle.Width * rectangle.Height;
        }

        if ( shape is Circle )
        {
            var circle = (Circle)shape;
            area += circle.Radius * circle.Radius * Math.PI;
        }
    }

    return area;
}
```

```
public double GetRectangleArea(Rectangle[] shapes)
{
    double area = 0;

    foreach ( var shape in shapes )
    {
        area += shape.Width * shape.Height;
    }

    return area;
}
```

Open-closed Principle

```
public abstract class Shape
{
    public abstract double Area();
}
```

```
public class RectangleObject : Shape
{
    public double Width { get; }
    public double Height { get; }

    private RectangleObject(double width, double height)
    {
        Width = width;
        Height = height;
    }

    public static RectangleObject Create(double width, double height)
    {
        return new(width, height);
    }

    public override double Area()
    {
        return Width * Height;
    }
}
```

```
public class CircleObject : Shape
{
    public double Radius { get; }

    private CircleObject(double radius)
    {
        Radius = radius;
    }

    public static CircleObject Create(double radius)
    {
        return new(radius);
    }

    public override double Area()
    {
        return Math.Pow(Radius, 2) * Math.PI;
    }
}
```

```
private double Area(Shape[] shapes)
{
    double area = 0;

    foreach ( var shape in shapes )
    {
        area += shape.Area();
    }

    return area;
}
```

Liskov substitution Principle

```
public class Apple
{
    public virtual string GetColor()
    {
        return "Red";
    }
}
```

```
public class Pineapple : Apple
{
    public override string GetColor()
    {
        return "Yellow";
    }
}
```

```
public string Color()
{
    Apple apple = new Pineapple();
    return apple.GetColor();
}
```

Liskov Substitution Principle

```
public abstract class Fruit
{
    public abstract string GetColor();
}
```

```
public class PineappleObject : Fruit
{
    public override string GetColor()
    {
        return "Yellow";
    }
}
```

```
public class AppleObject : Fruit
{
    public override string GetColor()
    {
        return "Red";
    }
}
```

```
public void Color()
{
    var fruits = new Fruit[] { new AppleObject(), new PineappleObject() };

    foreach ( var fruit in fruits )
    {
        Console.WriteLine(fruit.GetColor());
    }
}
```

Interface Segregation Principle

```
public interface IWorker
{
    string Id { get; set; }
    string Name { get; set; }
    string Email { get; set; }
    decimal MonthlySalary { get; set; }
    decimal OtherBenefits { get; set; }
    decimal HourlyRate { get; set; }
    decimal HoursInMonth { get; set; }
    decimal CalculateNetSalary();
    decimal CalculateWorkedSalary();
}
```

```
public class ContractEmployee : IWorker
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public decimal MonthlySalary { get; set; }
    public decimal OtherBenefits { get; set; }
    public decimal HourlyRate { get; set; }
    public decimal HoursInMonth { get; set; }
    public decimal CalculateNetSalary() => throw new NotImplementedException();
    public decimal CalculateWorkedSalary() => HourlyRate * HoursInMonth;
}
```

```
public class FullTimeEmployee : IWorker
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public decimal MonthlySalary { get; set; }
    public decimal OtherBenefits { get; set; }
    public decimal HourlyRate { get; set; }
    public decimal HoursInMonth { get; set; }
    public decimal CalculateNetSalary() => MonthlySalary + OtherBenefits;
    public decimal CalculateWorkedSalary() => throw new NotImplementedException();
}
```

Interface Segregation Principle

```
public interface IBaseWorker
{
    string Id { get; set; }
    string Name { get; set; }
    string Email { get; set; }
}
```

```
public interface IContractWorkerSalary : IBaseWorker
{
    decimal HourlyRate { get; set; }
    decimal HoursInMonth { get; set; }
    decimal CalculateWorkedSalary();
}
```

```
public interface IFullTimeWorkerSalary : IBaseWorker
{
    decimal MonthlySalary { get; set; }
    decimal OtherBenefits { get; set; }
    decimal CalculateNetSalary();
}
```

```
public class ContractEmployedDomain : IContractWorkerSalary
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public decimal HourlyRate { get; set; }
    public decimal HoursInMonth { get; set; }
    public decimal CalculateWorkedSalary() => HourlyRate * HoursInMonth;
}
```

```
public class FullTimeEmployedDomain : IFullTimeWorkerSalary
{
    public string Id { get; set; }
    public string Name { get; set; }
    public string Email { get; set; }
    public decimal MonthlySalary { get; set; }
    public decimal OtherBenefits { get; set; }
    public decimal CalculateNetSalary() => MonthlySalary + OtherBenefits;
}
```

Dependency Inversion Principle

```
public interface ICustomerDataAccess
{
    string GetCustomerName(int id);
}
```

```
public class CustomerController
{
    ICustomerDataAccess _customerDataAccess;

    public CustomerController()
    {
        _customerDataAccess = DataAccessFactory.GetCustomerDataAcessObj();
    }

    public string GetCustomerName(int id)
    {
        return _customerDataAccess.GetCustomerName(id);
    }
}
```

```
public class DataAccessFactory
{
    public static ICustomerDataAccess
        GetCustomerDataAcessObj() =>
            new CustomerDataAccess();
}
```

```
public class CustomerDataAccess : ICustomerDataAccess
{
    CustomerRepository _customerRepository { get; } = new();

    public string GetCustomerName(int id)
    {
        var name = _customerRepository.GetCustomerNameById(id);
        return name;
    }
}
```