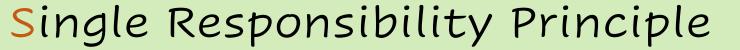
SOLID Princípios

Exemplos

por Felipe Arantes







```
public void AddUser(string username)
                                                          public void AddUser (string username)
   //Validate
                                                              var user = UserDomain.Create(username);
    if ( username == "Admin" )
                                                              if ( user.IsNotValid )
       throw new InvalidOperationException();
                                                                  throw new InvalidOperationException();
    //Create a User
                                                              _userRepository.Insert(user);
    var connection = new SqlConnection();
                                                              _emailService.Send(user);
    connection.Open();
    var command = new SqlCommand("INSERT INTO...");
    //Send Email
    var client = new SmtpClient("host");
    client.Send(new MailMessage());
```

Open-closed Principle



```
public double GetRectangleArea(Rectangle[] shapes)
private double Area(object[] shapes)
    double area = 0;
                                                                    double area = 0;
    foreach ( var shape in shapes )
                                                                    foreach ( var shape in shapes )
        if ( shape is Rectangle )
                                                                        area += shape.Width * shape.Height;
            var rectangle = (Rectangle)shape;
            area += rectangle.Width * rectangle.Height;
                                                                    return area;
        if ( shape is Circle )
            var circle = (Circle)shape;
            area += circle.Radius * circle.Radius * Math.PI;
    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 class AppleObject : Fruit
    public override string GetColor()
                                               public override string GetColor()
        return "Yellow";
                                                  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
                                         public class ContractEmployee : IWorker
    string Id { get; set; }
                                             public string Id { get; set; }
   string Name { get; set; }
                                             public string Name { get; set; }
                                             public string Email { get; set; }
   string Email { get; set; }
   decimal MonthlySalary { get; set; }
                                             public decimal MonthlySalary { get; set; }
                                             public decimal OtherBenefits { get; set; }
   decimal OtherBenefits { get; set; }
   decimal HourlyRate { get; set; }
                                             public decimal HourlyRate { get; set; }
   decimal HoursInMonth { get; set; }
                                             public decimal HoursInMonth { get; set; }
                                             public decimal CalculateNetSalary() => throw new NotImplementedException();
   decimal CalculateNetSalary();
                                             public decimal CalculateWorkedSalary() => HourlyRate * HoursInMonth;
   decimal CalculateWorkedSalary();
                                         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
                                                 public class CustomerController
    string GetCustomerName(int id);
                                                     ICustomerDataAccess _customerDataAccess;
                                                     public CustomerController()
                                                          _customerDataAccess = DataAccessFactory.GetCustomerDataAcessObj();
                                                     public string GetCustomerName(int id)
                                                         return _customerDataAccess.GetCustomerName(id);
public class DataAccessFactory
                                                 public class CustomerDataAccess : ICustomerDataAccess
    public static ICustomerDataAccess
                                                     CustomerRepository _customerRepository { get; } = new();
        GetCustomerDataAcessObj() =>
            new CustomerDataAccess();
                                                     public string GetCustomerName(int id)
                                                          var name = _customerRepository.GetCustomerNameById(id);
                                                         return name;
```