



Embarcadero Conference

Escrevendo códigos **SOLID** com Delphi

André Luis Celestino

Embarcadero

Conference

Sobre o palestrante



André Celestino

- Embarcadero MVP desde 2017
- Dev Delphi Sênior @ DB1 Global Software
- 7 anos de experiência com programação
- *Certified Delphi Developer* e *SAFe Practitioner*
- Autor do blog www.AndreCelestino.com





Agenda

- O que é SOLID?
- SRP
- OCP
- LSP
- ISP
- DIP



Afinal, o que é SOLID?

- Conjunto de princípios para desenvolvimento OO;
- Introduzido por Robert C. Martin (Uncle Bob) na década de 2000;
- Promove uma arquitetura mais flexível e sustentável;
- Acrônimo para **S**RP, **O**CP, **L**SP, **I**SP e **D**IP;
- Fácil compreensão, desde que tenha conhecimento em OO.



SRP – *Single Responsibility Principle*

Princípio da Responsabilidade Única

“A class should have one, and only one, reason to change”

(Uma classe deve ter um, e somente um, motivo para mudar)

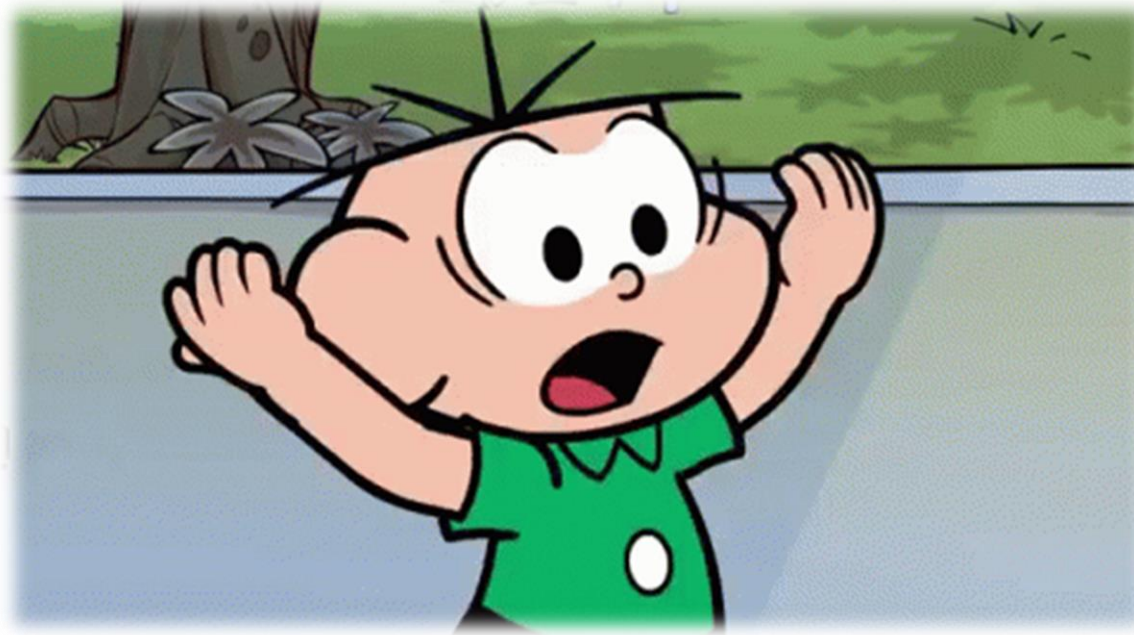
SRP – *Single Responsibility Principle*

Somente para deixar o código mais bonito?



SRP – *Single Responsibility Principle*

Evitar o POLA!



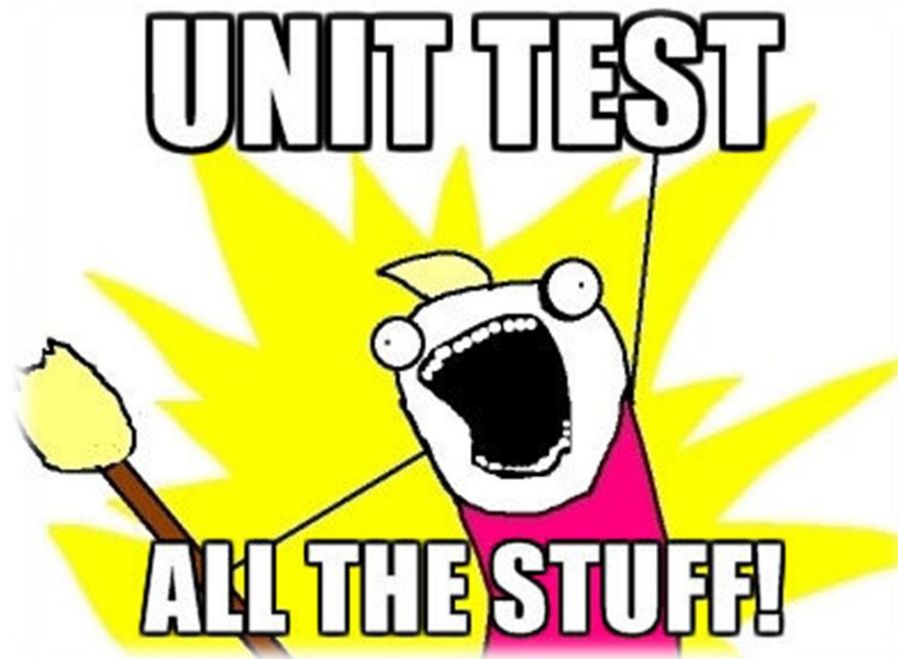
SRP – *Single Responsibility Principle*

Cuidado com a conjunção “e”



SRP – *Single Responsibility Principle*

Facilita testes unitários!



SRP – *Single Responsibility Principle*



VIEW DEMO



OCP – *Open/Closed Principle*

Princípio do Aberto/Fechado

“Software entities should be open for extension, but closed for modification”

(Entidades de software devem estar abertas para extensão, mas fechadas para modificação)

OCP – *Open/Closed Principle*



Quem nunca?

```
if A then  
    ProcessarA  
else if B then  
    ProcessarB  
else if C then  
    ProcessarC  
else if D then  
    ProcessarD  
...
```

OCP – Open/Closed Principle

Reduz a complexidade ciclomática

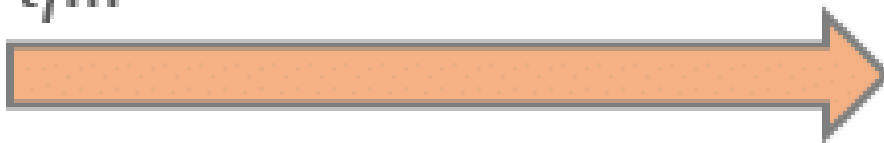
```
function register()
{
    if (!empty($_POST)) {
        $msg = '';
        if ($_POST['user_name']) {
            if ($_POST['user_password_new']) {
                if ($_POST['user_password_new'] === $_POST['user_password_repeat']) {
                    if (strlen($_POST['user_password_new']) > 5) {
                        if (strlen($_POST['user_name']) < 65 && strlen($_POST['user_name']) > 1) {
                            if (preg_match('/^[a-z\d]{2,64}$/i', $_POST['user_name'])) {
                                $user = read_user($_POST['user_name']);
                                if (!isset($user['user_name'])) {
                                    if ($_POST['user_email']) {
                                        if (strlen($_POST['user_email']) < 65) {
                                            if (filter_var($_POST['user_email'], FILTER_VALIDATE_EMAIL)) {
                                                create_user();
                                                $_SESSION['msg'] = 'You are now registered so please login';
                                                header('Location: ' . $_SERVER['PHP_SELF']);
                                                exit();
                                            } else $msg = 'You must provide a valid email address';
                                        } else $msg = 'Email must be less than 64 characters';
                                    } else $msg = 'Email cannot be empty';
                                } else $msg = 'Username already exists';
                            } else $msg = 'Username must be only a-z, A-Z, 0-9';
                        } else $msg = 'Username must be between 2 and 64 characters';
                    } else $msg = 'Password must be at least 6 characters';
                } else $msg = 'Passwords do not match';
            } else $msg = 'Empty Password';
        } else $msg = 'Empty Username';
        $_SESSION['msg'] = $msg;
    }
    return register_form();
}
```



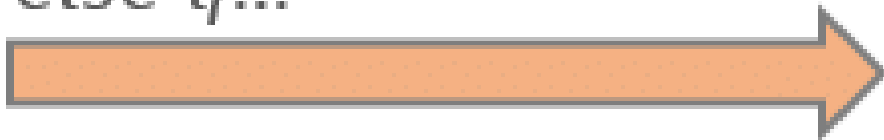
OCP – *Open/Closed Principle*

Cada *if...*

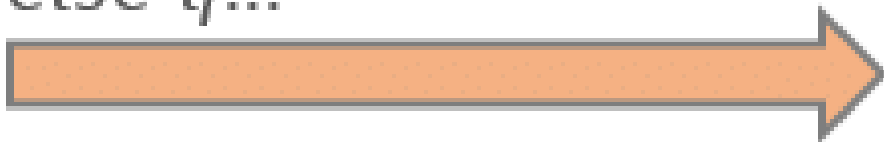
if...



else if...

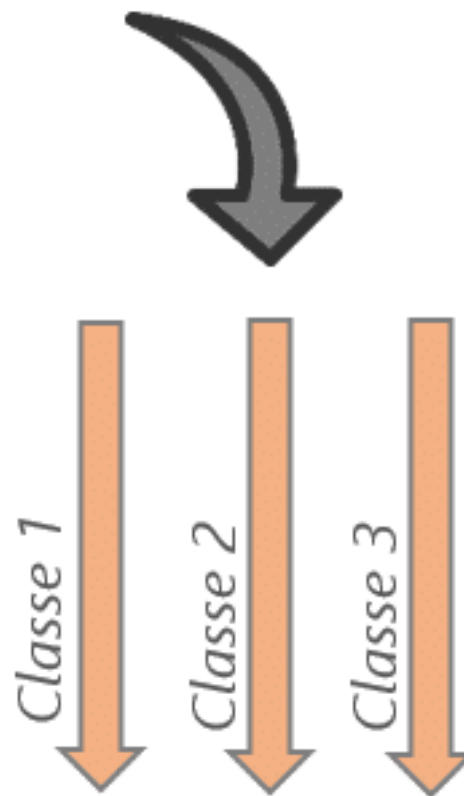


else if...



OCP – *Open/Closed Principle*

... deve se transformar em uma classe:



OCP – *Open/Closed Principle*



VIEW DEMO



LSP – *Liskov Substitution Principle*

Princípio da Substituição de Liskov

“Functions that use pointers to base classes must be able to use objects of derived classes without knowing it”

(Funções que referenciam classes base podem usar objetos de classes derivadas sem conhecimento prévio)

LSP – *Liskov Substitution Principle*

var

// declaração como classe base

Objeto: TClasseBase;

begin

// criação como classe filha

Objeto := TSubClasse.Create;

Objeto.ExecutarAcao;

...

end;



LSP – *Liskov Substitution Principle*

Princípio da Substituição de Liskov

“Objects of base classes may be replaced with objects of subclasses without altering any of the desirable properties”

(Objetos de classes base podem ser substituídas por objetos de subclasses sem alterar suas propriedades desejadas)

OCP – *Open/Closed Principle*

Cuidado com *Typecastings*



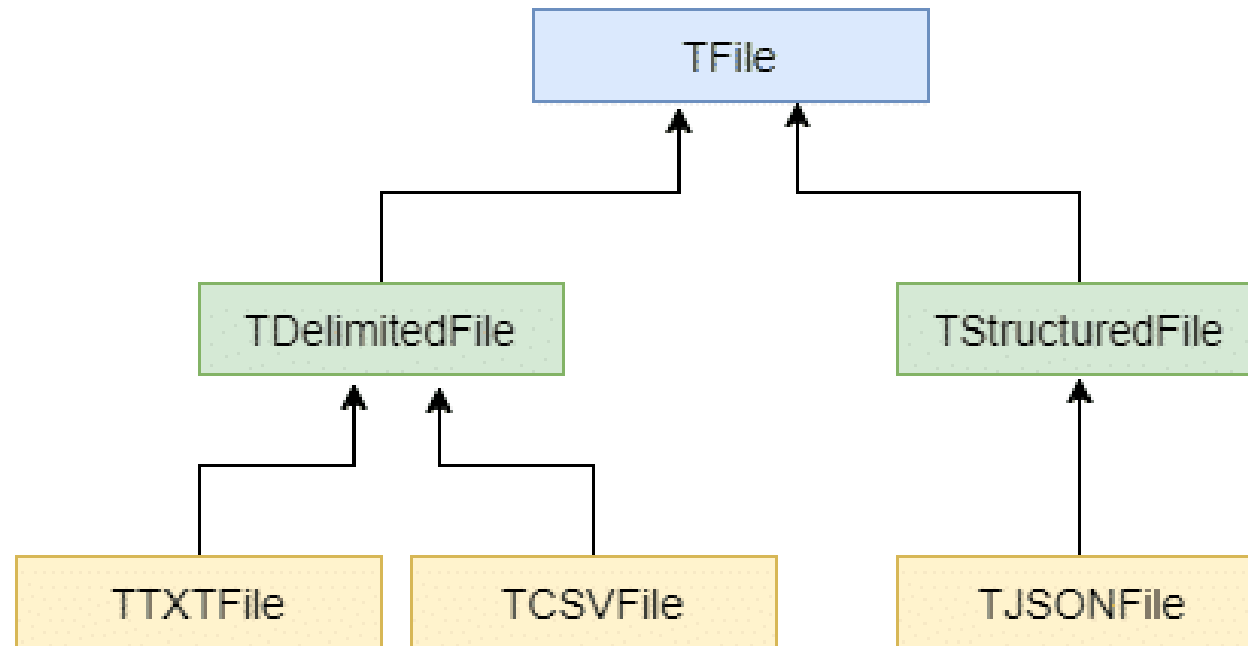
OCP – *Open/Closed Principle*



VIEW DEMO

OCP – *Open/Closed Principle*

Solução:





ISP – *Interface Segregation Principle*

Princípio da Segregação de Interface

“Clients should not be forced to depend upon interfaces that they do not use”

(Clientes não devem ser forçados a depender de Interfaces que não utilizam)

ISP – *Interface Segregation Principle*

Evitar essa palavrinha como única instrução:





ISP – *Interface Segregation Principle*

```
procedure ExecutarServico;  
begin  
    Exit;  
end;
```

ISP – *Interface Segregation Principle*



VIEW DEMO



DIP – *Dependency Inversion Principle*

Princípio da Inversão de Dependência

“High-level modules should not depend on low-level modules. Both should depend on abstractions”

(Módulos de alto nível não devem depender de módulos de baixo nível. Ambos devem depender de abstrações)



DIP – *Dependency Inversion Principle*

Princípio da Inversão de Dependência

***“Abstractions should not depend on details.
Details should depend on abstractions”***

(Abstrações não devem depender de detalhes.
Detalhes devem depender de abstrações)

DIP – *Dependency Inversion Principle*

I  T

DIP – *Dependency Inversion Principle*

type

 TLeitorArquivoZip = **class**

public

constructor Create (ArquivoZip: TArquivoZip);

end;

DIP – *Dependency Inversion Principle*

type

ILeitorArquivo = interface

 procedure LerArquivo (ArquivoZip: TArquivoZip);

end;

Abstrações não devem depender de detalhes...

DIP – *Dependency Inversion Principle*

type

ILeitorArquivo = **interface**

procedure LerArquivo (Arquivo: IArquivo);

end;

DIP – *Dependency Inversion Principle*



VIEW DEMO

OBRIGADO



andredelphi@gmail.com



www.andrecelestino.com



carreira.db1.com.br

Embarcadero

Conference



Embarcadero Conference