

Matheus Antonio Oliveira Cardoso

Padrões de Projeto e o Paradigma Funcional

Brasil

2021, v-1.9.7

Matheus Antonio Oliveira Cardoso

Padrões de Projeto e o Paradigma Funcional

Modelo canônico de trabalho monográfico
acadêmico em conformidade com as normas
ABNT apresentado à comunidade de usuários
L^AT_EX.

Universidade Federal Fluminense – UFF

Instituto de Ciência e Tecnologia

Ciência da Computação

Orientador: orientador

Coorientador: Equipe abnT_EX2

Brasil

2021, v-1.9.7

Matheus Antonio Oliveira Cardoso

Padrões de Projeto

e o Paradigma Funcional/ Matheus Antonio Oliveira Cardoso. – Brasil, 2021, v-1.9.7-49p. : il. (algumas color.) ; 30 cm.

Orientador: orientador

Tese (Graduação) – Universidade Federal Fluminense – UFF

Instituto de Ciência e Tecnologia

Ciência da Computação, 2021, v-1.9.7.

1. Palavra-chave1. 2. Palavra-chave2. 2. Palavra-chave3. I. Orientador. II. Universidade xxx. III. Faculdade de xxx. IV. Título

Matheus Antonio Oliveira Cardoso

Padrões de Projeto e o Paradigma Funcional

Modelo canônico de trabalho monográfico
acadêmico em conformidade com as normas
ABNT apresentado à comunidade de usuários
L^AT_EX.

Trabalho aprovado. Brasil, 24 de novembro de 2012:

orientador
Orientador

Professor
Convidado 1

Professor
Convidado 2

Brasil
2021, v-1.9.7

*Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.*

Agradecimentos

Os agradecimentos principais são direcionados à Gerald Weber, Miguel Frasson, Leslie H. Watter, Bruno Parente Lima, Flávio de Vasconcellos Corrêa, Otavio Real Salvador, Renato Machnievscz¹ e todos aqueles que contribuíram para que a produção de trabalhos acadêmicos conforme as normas ABNT com L^AT_EX fosse possível.

Agradecimentos especiais são direcionados ao Centro de Pesquisa em Arquitetura da Informação² da Universidade de Brasília (CPAI), ao grupo de usuários *latex-br*³ e aos novos voluntários do grupo *abnT_EX2*⁴ que contribuíram e que ainda contribuirão para a evolução do abnT_EX2.

¹ Os nomes dos integrantes do primeiro projeto abnT_EX foram extraídos de <<http://codigolivres.org.br/projects/abntex/>>

² <<http://www.cpai.unb.br/>>

³ <<http://groups.google.com/group/latex-br>>

⁴ <<http://groups.google.com/group/abntex2>> e <<http://www.abntex.net.br/>>

*“Não vos amoldeis às estruturas deste mundo,
mas transformai-vos pela renovação da mente,
a fim de distinguir qual é a vontade de Deus:
o que é bom, o que Lhe é agradável, o que é perfeito.
(Bíblia Sagrada, Romanos 12, 2)*

Resumo

O presente trabalho tem como objetivo analisar o conceito de padrões de projeto no contexto do paradigma de programação funcional. Os padrões de projeto apresentam soluções comuns para problemas comuns de design de software, destacando-se os vinte e três padrões Gang of Four, que apresentam soluções comuns para problemas relacionados ao paradigma orientado a objetos. Porém, como a forma de construir um software em um paradigma funcional difere muito de um paradigma orientado a objetos, existe a dúvida de como ou se esses padrões podem ser reaproveitados ou se outros problemas comuns poderiam surgir a partir do design funcional de software, originando novos padrões. Dessa forma, o trabalho buscará analisar, do ponto de vista funcional, cada um dos 23 padrões GOF, verificando se o problema de orientação a objetos em questão também existe no contexto funcional e se é resolvido pelo padrão em questão ou porque o problema não existe nesse contexto. Também será analisado se existem problemas específicos para o paradigma funcional e se existem padrões conhecidos que podem resolvê-los. No fim, deseja-se concluir se existe alguma relação entre o tipo de problema que cada padrão resolve e a conclusão da análise do mesmo e também se os problemas relacionados ao contexto funcional encontrados podem também ter alguma relação com eles.

Palavras-chave: latex. abntex. editoração de texto.

Abstract

This is the english abstract.

Keywords: latex. abntex. text editoration.

Lista de ilustrações

Figura 1 – Estrutura do Factory Method	35
Figura 2 – Estrutura do Abstract Factory	37
Figura 3 – Estrutura do Singleton	38
Figura 4 – Estrutura do Strategy	41

Lista de códigos

Código 1 – Classe comum em Orientação a Objetos	33
Código 2 – Representação de uma classe no contexto funcional	33
Código 3 – Factory Method Orientação a Objetos	35
Código 4 – Abstract Factory Orientação a Objetos	37
Código 5 – Abstract Factory Orientação a Objetos	37
Código 6 – Singleton Orientação a Objetos	38
Código 7 – Injeção de Dependência funcional	39
Código 8 – Monad Reader	39
Código 9 – Strategy Orientação a Objetos	41
Código 10 – Strategy Funcional	42
Código 11 – Strategy Funcional: Problema	42

Lista de abreviaturas e siglas

GOF	Gang of Four
-----	--------------

Sumário

1	INTRODUÇÃO	23
I	CONCEITOS BÁSICOS	25
2	O PARADIGMA FUNCIONAL	27
2.1	Aliquam vestibulum fringilla lorem	27
3	PADRÕES DE PROJETO	29
3.1	Aliquam vestibulum fringilla lorem	29
II	DESENVOLVIMENTO	31
4	PADRÕES DE PROJETO NO CONTEXTO FUNCIONAL	33
4.1	Criacionais	34
4.1.1	Factory Method	35
4.1.2	Abstract Factory	37
4.1.3	Singleton	38
4.2	Estruturais	40
4.3	Comportamentais	40
4.3.1	Strategy	41
III	RESULTADOS	45
5	CONCLUSÃO	47
	REFERÊNCIAS	49

1 Introdução

Durante o processo de construção de um software diversos problemas de design são enfrentados, alguns mais simples, outros mais trabalhosos. Alguns desses problemas são tão comuns que achou-se necessário definir um padrão de solução para eles, reduzindo o tempo que desenvolvedores que passariam pelo mesmo problema futuramente gastariam tentando chegar até a mesma solução que outros desenvolvedores chegaram no passado. Essa ideia deu origem ao que chamamos de Padrões de Projeto, soluções ideais para problemas comuns ou difíceis de se resolver no desenvolvimento de software. Alguns desses problemas deram origem a padrões tão comuns que quatro desenvolvedores, conhecidos como Gang of Four, reuniram-se para catalogar esses padrões, dando origem aos vinte e três Padrões de Projeto GOF. Entretanto, esses padrões comuns são voltados para um paradigma de programação tão comum quanto: o Orientado a Objetos. Sendo um paradigma mais conhecido através de linguagens de programação famosas como Java, normalmente são esses os padrões aprendidos pelos estudantes ou desenvolvedores comuns. O problema é que a Orientação a Objetos não é o único paradigma

Parte I

Conceitos Básicos

2 O Paradigma Funcional

2.1 Aliquam vestibulum fringilla lorem

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

3 Padrões de Projeto

3.1 Aliquam vestibulum fringilla lorem

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Ut purus elit, vestibulum ut, placerat ac, adipiscing vitae, felis. Curabitur dictum gravida mauris. Nam arcu libero, nonummy eget, consectetur id, vulputate a, magna. Donec vehicula augue eu neque. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas. Mauris ut leo. Cras viverra metus rhoncus sem. Nulla et lectus vestibulum urna fringilla ultrices. Phasellus eu tellus sit amet tortor gravida placerat. Integer sapien est, iaculis in, pretium quis, viverra ac, nunc. Praesent eget sem vel leo ultrices bibendum. Aenean faucibus. Morbi dolor nulla, malesuada eu, pulvinar at, mollis ac, nulla. Curabitur auctor semper nulla. Donec varius orci eget risus. Duis nibh mi, congue eu, accumsan eleifend, sagittis quis, diam. Duis eget orci sit amet orci dignissim rutrum.

Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse ut massa. Cras nec ante. Pellentesque a nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Aliquam tincidunt urna. Nulla ullamcorper vestibulum turpis. Pellentesque cursus luctus mauris.

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Parte II

Desenvolvimento

4 Padrões de Projeto no Contexto Funcional

O conceito de objeto não existe no paradigma funcional. Para ater-se a não reaproveitar recursos e conceitos oriundos da Orientação a Objetos nos exemplos que utilizam os recursos e conceitos oriundos da programação funcional, será usada uma estrutura equivalente quando for necessário o uso de alguma estrutura semelhante a um objeto. Um objeto pode ser definido como uma representação do mundo real que possui características (atributos) e comportamentos (métodos). Para representar as características, será utilizado o recurso `case class` de `scala`. Já os comportamentos serão definidos por funções que recebem como entrada um valor do `case class` criado e retorna uma nova variável do mesmo tipo `case class` ou o valor de alguma de suas características. Por exemplo, a classe a seguir, construída a partir do paradigma orientado a objetos:

```
class Person(var name : String, var age : Int){  
  
    def getName() : String = this.name  
  
    def setName(name : String) : Unit = this.name = name  
  
    def getAge() : Int = this.age  
  
    def setAge(age : Int) : Unit = this.age = age  
  
}
```

Código 1 – Classe comum em Orientação a Objetos

Pode ser representada da seguinte forma no paradigma funcional:

```
case class Person(name: String, age: Int)  
  
def getName(person : Person) : String = person.name  
  
def setName(person : Person, name : String) : Person =  
    person.copy(name = name)  
  
def getAge(person : Person) : Int = person.age  
  
def setAge(person : Person, age : Int) : Person =  
    person.copy(age = age)
```

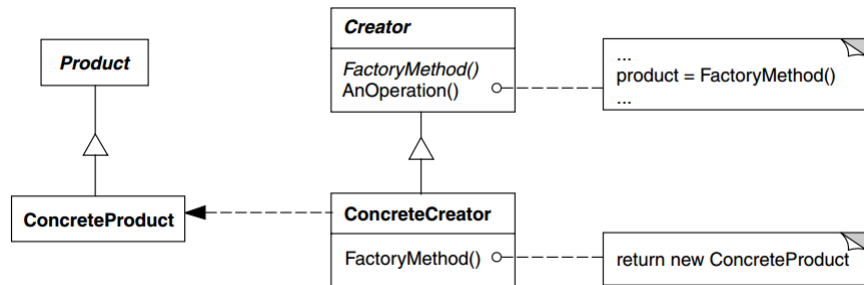
Código 2 – Representação de uma classe no contexto funcional

4.1 Criacionais

4.1.1 Factory Method

O padrão Factory Method tem como objetivo oferecer, através de uma classe Factory, uma interface para a criação de objetos. Esses objetos, porém, podem ser configurados através de classes que herdam de Factory.

Figura 1 – Estrutura do Factory Method



Exemplo Orientado a Objetos:

```
trait Product{
    def doStuff() : Unit
}

class ConcreteProduct extends Product(){

    def doStuff() : Unit = {

    }

}

abstract class Creator(){

    def someOperation() : Unit = {
        var p = createProduct()
        p.doStuff()
    }

    def createProduct() : Product
}

class ConcreteCreator() extends Creator{

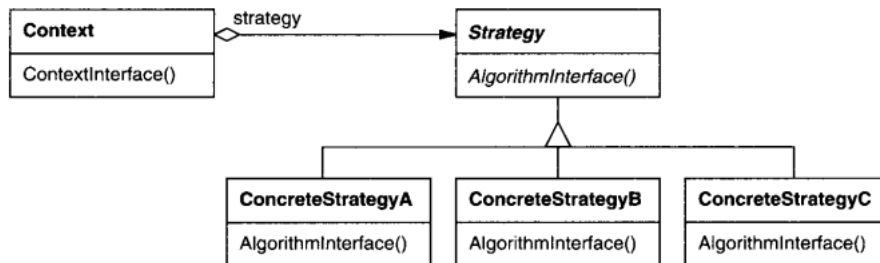
    def createProduct() : Product = {
        return new ConcreteProduct()
    }

}
```

Código 3 – Factory Method Orientação a Objetos

4.1.2 Abstract Factory

Figura 2 – Estrutura do Abstract Factory



Exemplo Orientado a Objetos:

Código 4 – Abstract Factory Orientação a Objetos

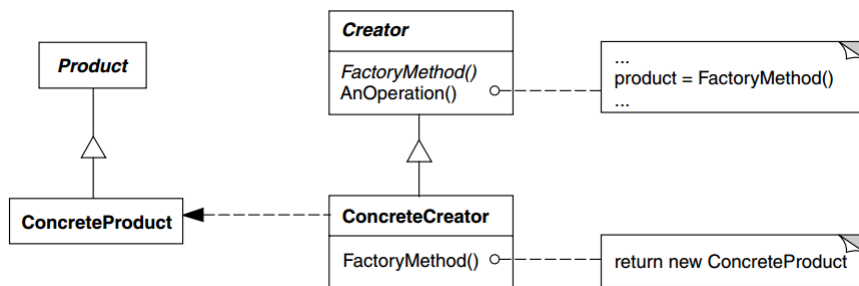
Contexto Funcional:

Código 5 – Abstract Factory Orientação a Objetos

4.1.3 Singleton

O padrão Singleton fornece um ponto de acesso global a um objeto e garante que ele possuirá apenas uma instância. Esse padrão é importante para implementar serviços e oferecer acesso a eles sem instanciar vários objetos iguais em diversos pontos diferentes do código.

Figura 3 – Estrutura do Singleton



Exemplo Orientado a Objetos:

```
class Database private () {

    def query(sql)

}

object Database {

    private val _instance = new Database()
    def instance() = _instance

}
```

Código 6 – Singleton Orientação a Objetos

Contexto Funcional:

Não existe uma forma de implementar o Singleton no contexto funcional por que ele viola o conceito de função pura, ou seja, a função não está mais dependendo apenas de seus parâmetros, mas também de um valor global que ainda pode ter seu estado modificado.

Porém, ainda existem formas de alcançar seu objetivo, ou seja, oferecer acesso a um serviço em diversos locais do código sem a necessidade de repeti-lo. A primeira forma é usando um conceito que não é exclusivamente funcional, já que até no contexto orientado a objetos é considerado um bom substituto para o Singleton. Porém, por ser uma abordagem

também utilizada por programas que seguem o paradigma funcional e consequentemente por não violar o paradigma, será mencionado como uma possível solução.

A abordagem consiste no uso da Injeção de Dependência, onde a criação de recurso utilizado por uma função ou objeto não é responsabilidade da mesma, ao invés disso, esse recurso é injetado, seja pelo construtor (no caso da orientação a objetos) ou por parâmetros de uma função (no caso do paradigma funcional).

Código 7 – Injeção de Dependência funcional

A segunda abordagem consiste na utilização de um Monad conhecido como Reader. As funções que precisam utilizar um determinado serviço são encapsuladas em um Monad. O estado desse serviço será acessável dentro dessas funções e sempre que suas execuções terminarem, o novo estado do serviço será retornado. Dessa forma, a próxima função que deseja utilizar o serviço poderá usufruir do estado atualizado.

Código 8 – Monad Reader

Essa abordagem tem algumas vantagens se comparada à injeção de dependência: Suponha que três funções são encadeadas em um programa. A primeira e a terceira precisarão utilizar o serviço que é injetado através dos parâmetros. A segunda função, mesmo sem utilizar o serviço, precisará recebê-lo em seus parâmetros para que ele seja passado para a terceira função. Isso diminui a reusabilidade dessa função, que poderia ser reaproveitada em um contexto onde o serviço não é necessário. Também há a poluição visual ao incluir, em diversas funções, parâmetros diferentes para fornecer os serviços. Em casos em que mais de um serviço é utilizado, a situação torna-se ainda mais caótica.

4.2 Estruturais

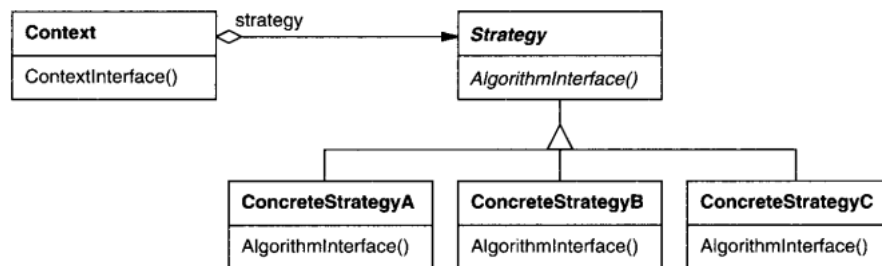
4.3 Comportamentais

4.3.1 Strategy

O padrão Strategy define grupos de algoritmos encapsulados e intercambiáveis para um determinado contexto. Esses algoritmos podem ser definidos ou trocados em tempo de execução, permitindo que os clientes que os utilizem possam alternar entre as implementações definidas livremente.

O Strategy soluciona o problema de classes relacionadas diferirem apenas em algum comportamento, permitindo que esse comportamento possa ser isolado e o resto da implementação das classes reaproveitado. Ele também evita a utilização de muitas operações condicionais. Ao invés de verificar qual deve ser o comportamento toda vez que ele precisar ser executado, o comportamento é pré-definido pelo contexto.

Figura 4 – Estrutura do Strategy



Exemplo Orientado a Objetos:

```
trait Strategy {
    def execute(a : Int, b : Int) : Int
}

class ConcreteStrategyAdd() extends Strategy {
    def execute(a : Int, b : Int) : Int = {
        a + b
    }
}

class ConcreteStrategySubtract() extends Strategy {
    def execute(a : Int, b : Int) : Int = {
        a - b
    }
}

class ConcreteStrategyMultiply() extends Strategy {
    def execute(a : Int, b : Int) : Int = {
        a * b
    }
}
```

```

class Context() {

    private var strategy : Strategy

    def setStrategy(strategy : Strategy) =
        this.strategy = strategy

    def executeStrategy(a : Int, b : Int) : Int =
        this.strategy.execute(a, b)

}

```

Código 9 – Strategy Orientação a Objetos

Contexto Funcional:

No contexto funcional, o encapsulamento de algoritmos ou de comportamentos diferentes pode ser alcançado através de funções de alta ordem (high-order functions). Nesse caso, não é necessário definir interfaces ou objetos para encapsular esses comportamentos, eles podem ser recebidos através da passagem de parâmetro como funções:

```

def executeAdd(a : Int, b : Int) : Int = {
    a + b
}

def executeSubtract(a : Int, b : Int) : Int = {
    a - b
}

def executeMultiply(a : Int, b : Int) : Int = {
    a * b
}

def executeStrategy(execute : (a : Int, b : Int) => Int) : Int =
    execute(a, b)

```

Código 10 – Strategy Funcional

Porém, existe uma desvantagem. A função [executeStrategy] acima aceita qualquer função que receba dois parâmetros inteiros e retorne um valor inteiro. Isso significa que qualquer função definida que não faça parte da solução mas que atenda a esse requisito pode ser usada como uma estratégia:

```

def executeOutOfScope(a : Int, b : Int) : Int = {
    a ** 2 + b ** 2
}

```

```
}
```

Código 11 – Strategy Funcional: Problema

No caso orientado a objetos, os comportamentos estão encapsulados em interfaces, o que torna mais segura a implementação dos comportamentos.

Parte III

Resultados

5 Conclusão

Sed consequat tellus et tortor. Ut tempor laoreet quam. Nullam id wisi a libero tristique semper. Nullam nisl massa, rutrum ut, egestas semper, mollis id, leo. Nulla ac massa eu risus blandit mattis. Mauris ut nunc. In hac habitasse platea dictumst. Aliquam eget tortor. Quisque dapibus pede in erat. Nunc enim. In dui nulla, commodo at, consectetur nec, malesuada nec, elit. Aliquam ornare tellus eu urna. Sed nec metus. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas.

Phasellus id magna. Duis malesuada interdum arcu. Integer metus. Morbi pulvinar pellentesque mi. Suspendisse sed est eu magna molestie egestas. Quisque mi lorem, pulvinar eget, egestas quis, luctus at, ante. Proin auctor vehicula purus. Fusce ac nisl aliquam ante hendrerit pellentesque. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi wisi. Etiam arcu mauris, facilisis sed, eleifend non, nonummy ut, pede. Cras ut lacus tempor metus mollis placerat. Vivamus eu tortor vel metus interdum malesuada.

Sed eleifend, eros sit amet faucibus elementum, urna sapien consectetur mauris, quis egestas leo justo non risus. Morbi non felis ac libero vulputate fringilla. Mauris libero eros, lacinia non, sodales quis, dapibus porttitor, pede. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos hymenaeos. Morbi dapibus mauris condimentum nulla. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Etiam sit amet erat. Nulla varius. Etiam tincidunt dui vitae turpis. Donec leo. Morbi vulputate convallis est. Integer aliquet. Pellentesque aliquet sodales urna.

Referências