



design PATTERNS

Lucas Shinji Yano
Vinícius Araújo Beltrami
Vitor Emanuel da S. Rozeno

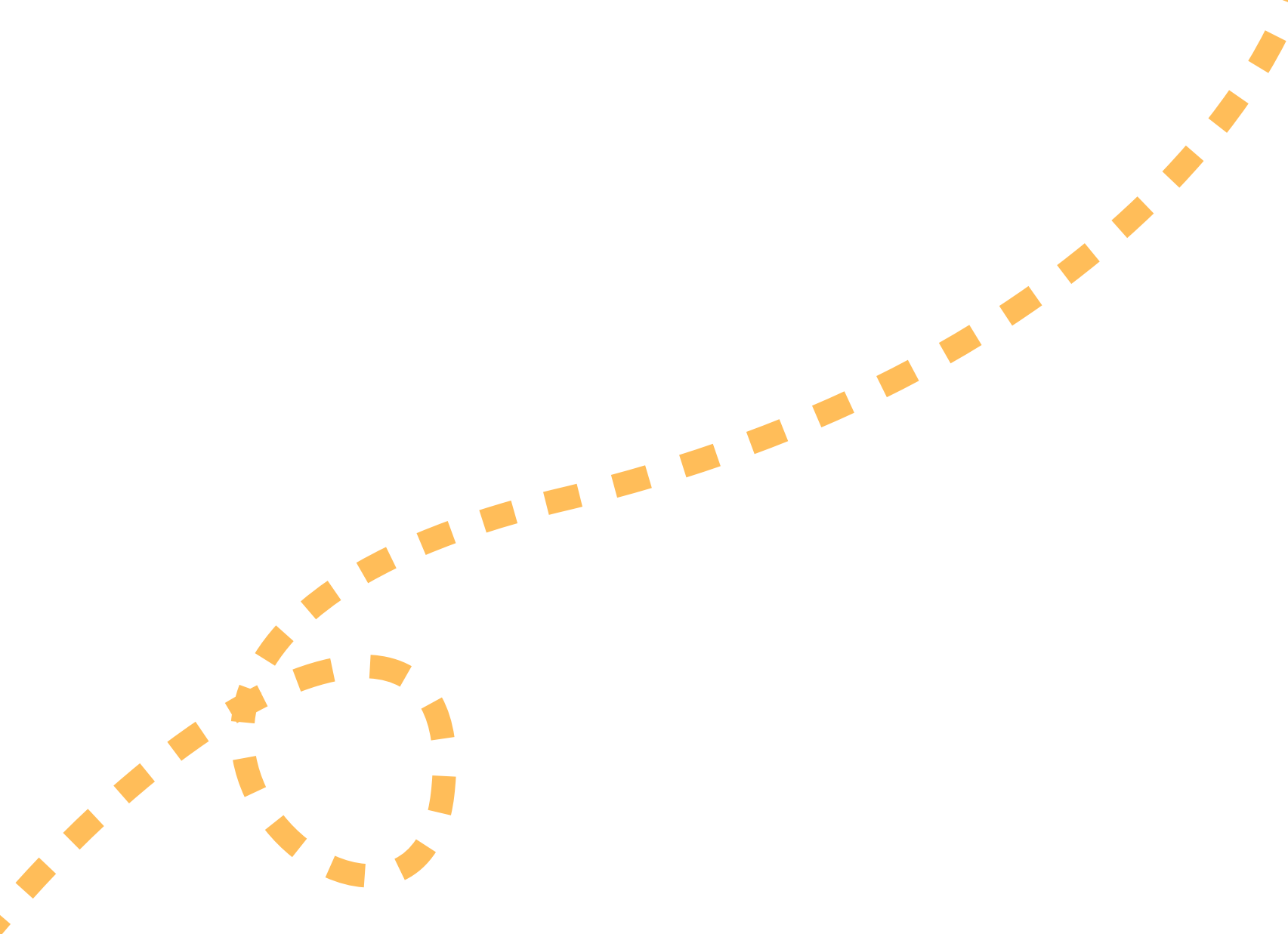
Engenharia de Software
Unesp - São José do Rio Preto

TÓPICOS DE DISCUSSÃO

- | | | | |
|----|-----------------------------|----|---------------------------------|
| 01 | O QUE SÃO DESIGN PATTERNS? | 06 | DESIGN PATTERNS COMPORTAMENTAIS |
| 02 | BENEFÍCIOS E DESAFIOS | 07 | IMPLEMENTAÇÃO: EXEMPLOS |
| 03 | CATEGORIAS | 08 | CASOS DE USO |
| 04 | DESIGN PATTERNS CRIACIONAIS | 09 | DICAS - PADRÃO ADEQUADO |
| 05 | DESIGN PATTERNS ESTRUTURAIS | 10 | CONCLUSÃO |

o que são
DESIGN PATTERNS?

**CAMINHO ATÉ OS
DESIGN PATTERNS**



o que são **DESIGN PATTERNS?**

CAMINHO ATÉ OS DESIGN PATTERNS

1. Programar é difícil.

o que são **DESIGN PATTERNS?**

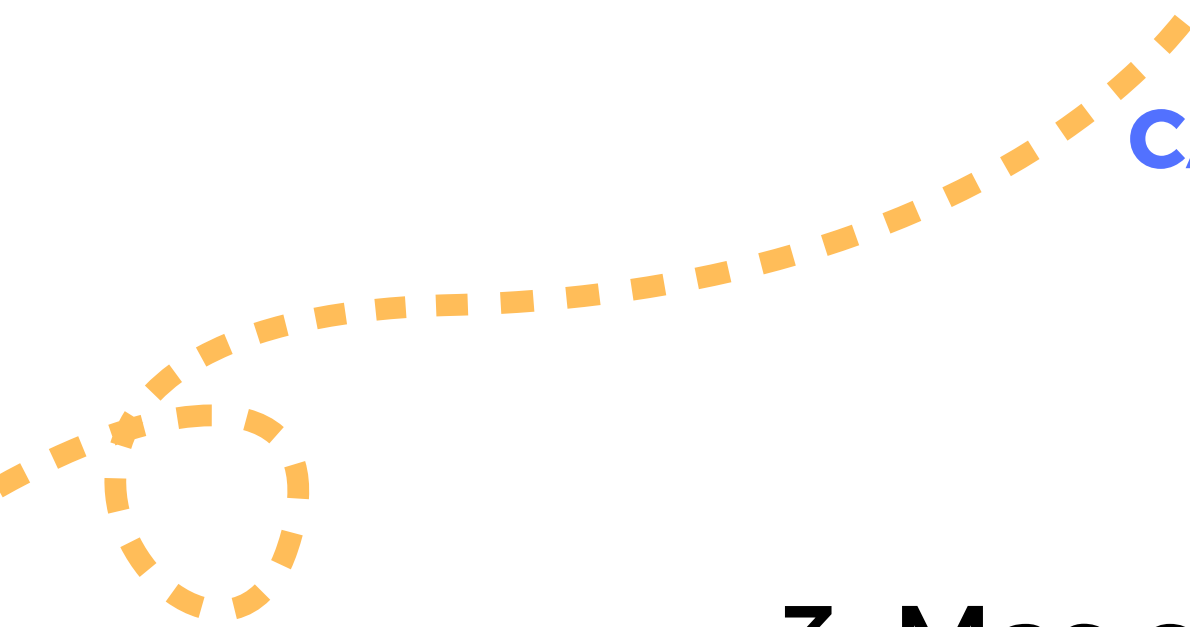
CAMINHO ATÉ OS DESIGN PATTERNS



2. Modelagem de sistemas é mais fácil, mas
poucos têm interesse

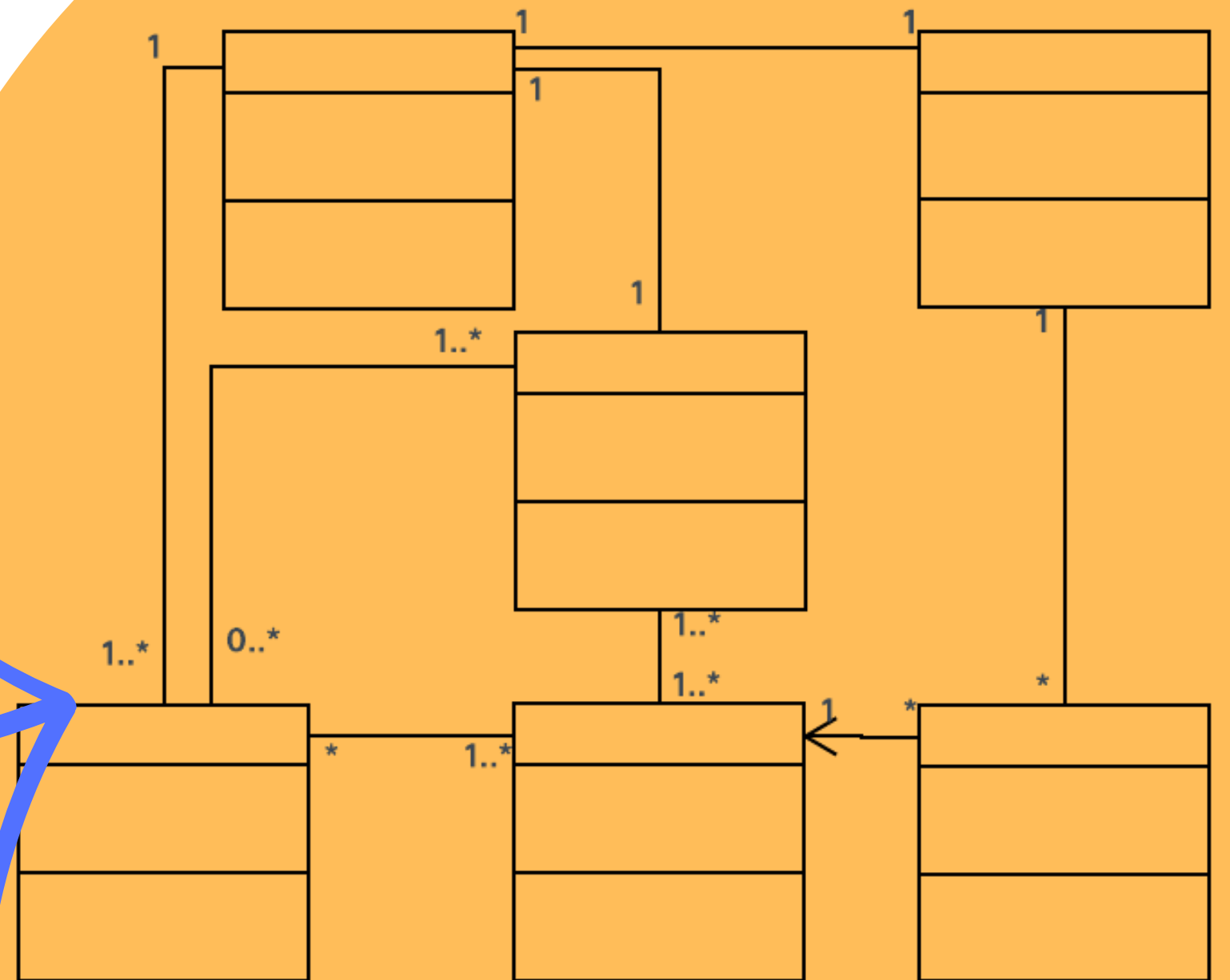
o que são **DESIGN PATTERNS?**

CAMINHO ATÉ OS DESIGN PATTERNS

- 
3. Mas caso você goste, mesmo assim seus diagramas UML podem se mostrar ineficientes na implementação

DESIGN PATTERNS?

diagrama UML



o que são **DESIGN PATTERNS?**

CAMINHO ATÉ OS DESIGN PATTERNS

4. Se a implementação dá errado,
mudanças “em voo” terão que ser feitas

(o que pode dar muita dor de cabeça)

o que são **DESIGN PATTERNS?**

CAMINHO ATÉ OS DESIGN PATTERNS

mas por que é tão difícil modelar?

- Existem inúmeras possibilidades de modelagem
- Não existe “certa” ou “errada” **(mas sim, adequada aos requisitos)**
- Requer tempo e esforço
- Remodelagem às vezes é necessária

**Uma modelagem inadequada pode confundir
sua equipe e atrasar o projeto!**

o que são **DESIGN PATTERNS?**

CAMINHO ATÉ OS DESIGN PATTERNS

com esses problemas...

Surgem os PADRÕES de Projeto

(ou Design Patterns)

o que são **DESIGN PATTERNS?**

CAMINHO ATÉ OS DESIGN PATTERNS

com esses problemas...

Surgem os PADRÕES de Projeto

(ou Design Patterns)

(não tem a ver com Design Gráfico ou Interface Gráfica)

o que são DESIGN PATTERNS?

Design Patterns, também conhecidos como **padrões de projeto**, são **abordagens genéricas para resolver desafios frequentes** que surgem durante o processo de desenvolvimento de software.

Eles não consistem em **código pronto ou frameworks específicos**, mas sim em **diretrizes de alto nível** que descrevem como resolver problemas comuns de forma eficaz e eficiente.

o que são **DESIGN PATTERNS?**

Estão relacionados com a
Arquitetura e Engenharia
de Software;

E fortemente ligados ao
paradigma de programação
Orientado à Objetos

o que são **DESIGN PATTERNS?**

Estão relacionados com a
Arquitetura e Engenharia
de Software;

- **Arquitetura:**
estrutura e
organização geral
do sistema
- **Engenharia:**
implementação
detalhada e
construção real do
software.

o que são **DESIGN PATTERNS?**

Estão relacionados com a
Arquitetura e Engenharia
de Software;

E fortemente ligados ao
paradigma de programação
Orientado à Objetos

o que são **DESIGN PATTERNS?**

POO:

- Classes
 - Objetos
 - Atributos
 - Métodos
- Abstração, herança, encapsulamento e polimorfismo

E fortemente ligados ao
paradigma de programação
Orientado à Objetos

o que são DESIGN PATTERNS?

São como **receitas de bolo**
para modelar **sistemas**
Orientados a Objetos



o que são DESIGN PATTERNS?

HISTÓRICO

1978: Livro “Uma Linguagem de Padrões”

- Christopher Alexander, Sara Ishikawa e Murray Silverstein
- Catalogaram 253 tipos de problemas (ou desafios de projeto)
- Proporam uma solução padrão e provada

1987: Palestra “Utilizando a linguagem dos padrões para programação orientada a objetos”

- Kent Back (Engenheiro de Software, posteriormente foi um dos criadores das metodologias Extreme Programming (XP) e Test Driven Development (TDD))
- Ward Cunningham
- Propuseram cinco padrões de projetos no campo da ciência da computação.

o que são DESIGN PATTERNS?

HISTÓRICO


1994: Livro “Padrões de Projeto – Soluções Reutilizáveis de Software Orientado a Objetos”

- Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides
- Os autores catalogaram 23 padrões que utilizaram ao longo de suas carreiras.
- Mais de 500.000 exemplares vendidos e foi publicado em 13 idiomas.
- Os autores do livro ficaram conhecidos como Gang of Four - GoF

Depois disso muitos outros livros surgiram, alguns criticando alguns desses padrões, e outros divulgando novos padrões.

o que são DESIGN PATTERNS?

(GoF) 4 Elementos Essencias de um Padrão:

- **Nome do Padrão**
 - Descrever os outros 3 elementos em uma palavra ou duas
 - Comunicação - Vocabulário de padrões
 - **O Problema**
 - Quando aplicar
 - Contexto
 - **A Solução**
 - Elementos que formam o padrão
 - Modelagem
 - Generalizada
 - **As Consequências**
 - Custos e benefícios de se aplicar um padrão
- 

benefícios dos **DESIGN PATTERNS**

01

Ganho de produtividade

Modelos já foram utilizados e testados anteriormente

02

Organização e Manutenção

Os padrões se baseiam em baixo acoplamento entre as classes e padronização do código.

03

Facilitação da discussão técnica

É mais fácil falar o nome de um design pattern em vez de ter que explicar todo o seu comportamento.

01 **Prática**
Identificar a aplicação e
implementar requer
prática

02 **Analisar seu requisitos**
Uma escolha de padrão errada
pode complicar seu projeto

03 **Adaptar o padrão**
Às vezes, modificações
serão necessárias

desafios dos
**DESIGN
PATTERNS**



categorias de **DESIGN PATTERNS**

CRIACIONAIS

ESTRUTURAIS

COMPORTAMENTAIS





categorias de **DESIGN PATTERNS**

CRIACIONAIS

Responsáveis por
instanciar objetos

(desacoplando o cliente e reduzindo
dependência entre dois componentes)



categorias de DESIGN PATTERNS

CRIACIONAIS

Responsáveis por
instanciar objetos

(desacoplando o cliente e reduzindo
dependência entre dois componentes)

Classe que utiliza
outra classe



categorias de **DESIGN PATTERNS**

CRIACIONAIS

ESTRUTURAIS

COMPORTAMENTAIS



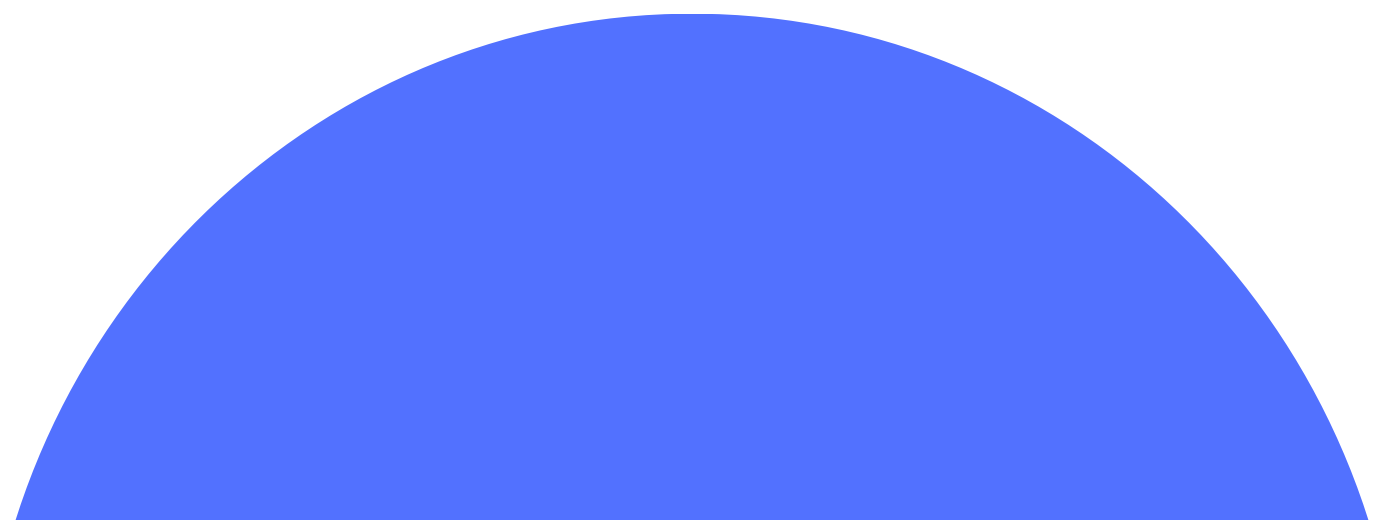


categorias de **DESIGN PATTERNS**

ESTRUTURAIS

Definem estruturas que
facilitam a utilização de
certas classes

(Conjunto de classes representa
uma estrutura)





categorias de **DESIGN PATTERNS**

CRIACIONAIS

ESTRUTURAIS

COMPORTAMENTAIS





categorias de **DESIGN PATTERNS**

Definem como classes
interagem e distribuem
responsabilidades

COMPORTAMENTAIS

(comportamentos são métodos)



categorias de **DESIGN PATTERNS**

COMPORTAMENTAIS

Definem como classes
interagem e distribuem
responsabilidades

Aderem ao SRP (Single
Responsability Principle)

“A classe ou
método deve ter
uma única
responsabilidade”



design patterns

CRIACIONAIS

Factory Method

Definição de uma **interface** que **decidirá** qual classe instanciar, por meio de parâmetros.

Abstract Factory

Organiza a criação de objetos em **grupos relacionados**, como **vários Factory Method juntos**.
ex. Botões e Janelas IOS e Android



design patterns

CRIACIONAIS

Builder

Criação do objeto **passo a passo** utilizando **métodos**, ao invés de utilizar o construtor

Prototype

Criação de objetos “**clones**”. Ao invés de usar herança de classes, cria-se um objeto com base em outro objeto protótipo.

Singleton

Um tipo de objeto que é instanciado **apenas uma vez**.
ex. “Global Settings”



design patterns

ESTRUTURAIS

Adapter

Criação de uma nova classe que permite **duas classes incompatíveis conversarem**.
(conversão de interface)

Bridge

Separa uma classe abstrata da implementação concreta.
(para variarem independentemente)
ex. formas e cores

Composite

Permite criar objetos compostos numa **estrutura de árvore**, que possuem **métodos iguais** aos **objetos das folhas**

design patterns

ESTRUTURAIS

Decorator

Permite adicionar funcionalidades aos objetos dinamicamente

Facade

Uma **API simplificada** para esconder detalhes do seu código ao criar classes de fachada, contendo os detalhes do sistema, mas facilitando a operação. Examples: JQuery.

Flyweight

Otimizar uso da RAM: vários objetos compartilham **valores imutáveis**

Proxy

Utilização de objetos “**substitutos**”, que tomam o lugar do objeto alvo para modificações, controlando o acesso.

design patterns **COMPORTAMENTAIS**

Chain of Responsibility

Transição de uma requisição ao longo de uma série de objetos que possuem métodos específicos para resolverem o problema

Command

Transforma uma requisição em um objeto que contém todas as informações e poderá ser reutilizado.

Interpreter

Mapeamento de um domínio de problemas para uma linguagem simples, definindo uma gramática.

design patterns

COMPORTAMENTAIS

Iterator

Uma maneira de acessar objetos sequencialmente, sem exposição, utilizando **loop**.

Mediator

Criação de objetos “**mediadores**”, evitando referência direta entre objetos.

Memento

Permite **salvar e restaurar o estado** de um objeto sem revelar detalhes de sua implementação a outros objetos

Observer

Dependência “**um para muitos**”.
Objetos são notificados e atualizados quando um certo objeto muda de estado.

design patterns

COMPORTAMENTAIS

State

Objetos mudam seu comportamento baseado em um número finito de **estados**.

Um mesmo método, resultados diferentes (com base no estado informado)

Template Method

Define um protótipo de algoritmo na superclasse que permite subclasses implementarem sem modificar a estrutura.

Strategy

Algoritmos separados em classes que possuem objetos intercambiáveis. Assim, objetos pode mudar de estratégia dinamicamente.

Visitor

Adição de novos métodos a classes existentes sem modificar a classe

exemplos de **IMPLEMENTAÇÃO** **FACTORY METHOD** **(CRIACIONAL)**

Objetivo: Criar botões baseados no sistema operacional.
Um botão se for IOS, outro se for Android.



exemplos de
IMPLEMENTAÇÃO
{ JavaScript }

O que normalmente seria feito:


```
class BotaoIOS( ){ }  
class BotaoAndroid( ){ }
```

```
const botao1 = so === 'ios' ? new BotaoIOS() : new BotaoAndroid();  
const botao2 = so === 'ios' ? new BotaoIOS() : new BotaoAndroid();
```

exemplos de
IMPLEMENTAÇÃO
FACTORY METHOD
(CRIACIONAL)
{ JavaScript }

Aplicando o Factory Method:

```
class FabricaBotao{  
  criaBotao(os:string):BotaoIOS|BotaoAndroid{  
    if(so === 'ios'){  
      return new BotaoIOS();  
    }  
    else{  
      return new BotaoAndroid();  
    }  
  }  
}
```



exemplos de
IMPLEMENTAÇÃO
FACTORY METHOD
(CRIACIONAL)
{ JavaScript }

Aplicando o Factory Method:

```
const fabrica= new FabricaBotao();  
const btn1 = fabrica.criaBotao(so);  
const btn2 = fabrica.criaBotao(so);
```

exemplos de **IMPLEMENTAÇÃO** **FACADE** **(ESTRUTURAL)**


Objetivo: criar uma “API” simples para acionar diferentes métodos de uma classe, **subjugando a complexidade** do usuário



exemplos de
IMPLEMENTAÇÃO
{ JavaScript }

O que normalmente seria feito:

```
class Eletricidade {  
    liga() { }  
    desliga() { }  
    setVoltagem() { }  
}  
  
class Casa {  
    constructor() {  
        this.eletricidade = new Eletricidade();  
    }  
}
```



exemplos de
IMPLEMENTAÇÃO
{ JavaScript }

O que normalmente seria feito:

```
const user = new Casa();  
user.eletricidade.setVoltagem(110);  
user.eletricidade.liga();
```

exemplos de
IMPLEMENTAÇÃO
{ JavaScript }

O que normalmente seria feito:

```
const user = new Casa();  
user.eletricidade.setVoltagem(110);  
user.eletricidade.liga();
```

Detalhes do sistema
(que não precisam ser aparentes)

exemplos de
IMPLEMENTAÇÃO
FACADE
(ESTRUTURAL)
{ JavaScript }


Aplicando o Facade:

```
class Eletricidade {  
    liga() { }  
    desliga() { }  
    setVoltagem() { }  
}
```

exemplos de
IMPLEMENTAÇÃO
FACADE
(ESTRUTURAL)
{ JavaScript }

Aplicando o Facade:

```
class Casa {  
  constructor() {  
    this.eletricidade = new Eletricidade();  
  }  
  
  ligaSistema() {  
    this.eletricidade.setVoltagem(110);  
    this.eletricidade.liga();  
  }  
}
```



exemplos de
IMPLEMENTAÇÃO
FACADE
(ESTRUTURAL)
{ JavaScript }

Aplicando o Facade:

```
const user = new Casa();  
user.ligaSistemas();  
user.desligaSistemas();
```


exemplos de IMPLEMENTAÇÃO MEDIATOR (COMPORTAMENTAL)


Objetivo: promover a comunicação entre objetos por meio do **objeto mediador**.



exemplos de
IMPLEMENTAÇÃO
MEDIATOR
(COMPORTAMENTAL)
{ JavaScript }

Aplicando o Mediator:

```
class Mediator {  
  notifica() { }  
}  
  
class Colegas {  
  constructor(mediator) {  
    this.mediator = mediator;  
  }  
}
```



exemplos de
IMPLEMENTAÇÃO
MEDIATOR
(COMPORTAMENTAL)
{ JavaScript }

Aplicando o Mediator:

```
const mediator = new Mediator();  
const colega1 = new Colegas(mediator);  
const colega2 = new Colegas(mediator);
```

```
colega1.mediator.notifica();  
colega2.mediator.notifica();
```






casos de uso de **DESIGN PATTERNS**

e-commerce:

clientes receberão atualizações em tempo real:

- atualização de preços
 - disponibilidade de produtos
- 

casos de uso de **DESIGN PATTERNS**

e-commerce:

clientes receberão atualizações em tempo real:

- atualização de preços
- disponibilidade de produtos

**ao atualizar um,
atualiza todos**

dependência “um para muitos”

casos de uso de **DESIGN PATTERNS**

e-commerce:

Observer





casos de uso de **DESIGN PATTERNS**

sistema bancário:

apenas uma instância da classe que gerencia conexões de banco de dados será criada.

- evitar problemas de concorrência
 - garantir a integridade dos dados dos clientes
- 

casos de uso de DESIGN PATTERNS

sistema bancário:

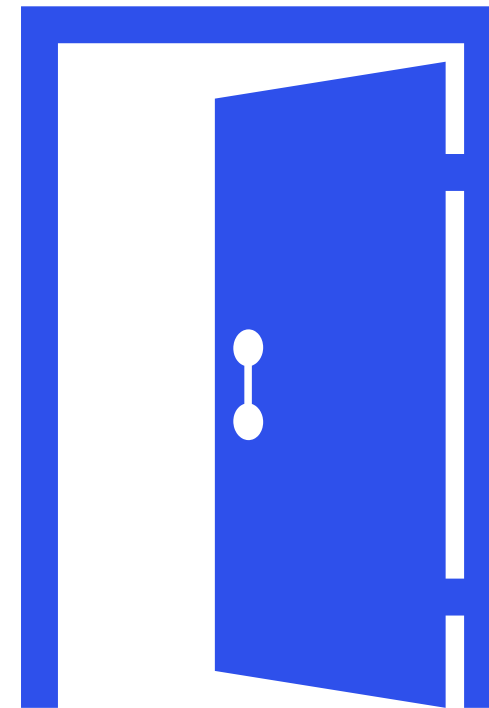
apenas uma instância da classe que gerencia conexões de banco de dados será criada.

- evitar problemas de concorrência
- garantir a integridade dos dados dos clientes

casos de uso de **DESIGN PATTERNS**

sistema bancário:

Singleton



casos de uso de **DESIGN PATTERNS**

sistema de cache:

controlar acesso a objetos “caros” de recursos, como bancos de dados ou serviços web:

- criar um objeto intermediário
- controlar quando e como os dados são buscados no objeto real
- evitar consultas custosas e frequentes de dados no real, implementando cache

casos de uso de DESIGN PATTERNS

sistema de cache:

controlar acesso a objetos “caros” de recursos, como bancos de dados ou serviços web:

- criar um objeto intermediário
- controlar quando e como os dados são buscados no objeto real
- evitar consultas custosas e frequentes de dados no real, implementando cache

casos de uso de **DESIGN PATTERNS**

sistema de cache:


Proxy



dicas: QUAL PADRÃO USAR?

- entenda os **requisitos**
- analise o **contexto**
- identifique problemas **recorrentes**
- **estude** os padrões disponíveis
- considere a **manutenibilidade**
- avalie o nível de **complexidade**
- **discuta** com sua equipe

CONCLUSÃO

- Padrões de Projeto existem para solucionar **problemas comuns**
 - Facilitar a modelagem e evitar mudanças durante o projeto
 - É necessária muita **prática e estudo** para dominá-los e aplicá-los corretamente
 - Avaliar os prós e contras
 - Não é só aplicar todos padrões possíveis
 - Em conclusão, aplicando corretamente padrões de projeto, certamente seus processos serão mais **produtivos** e **organizados**.
- 



OBRIGADO!
DÚVIDAS?



REFERÊNCIAS

- 1.(Conceitos) Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1995). **Design Patterns. Elements of Reusable Object-Oriented Software** 1 ed. Estados Unidos: Addison-Wesley.
- 2.(Vídeo - Introdução) Prof. Me. Manoel Campos da Silva Filho - **DESIGN PATTERNS: o que são PADRÕES de PROJETOS e porque precisa deles como Engenheiro de Software**
- 3.(Implementação) Alexander Shvets (2021). **Dive into Design Patterns**