Instituto Federal Catarinense *(Campus Blumenau)*

Professor: Ricardo de La Rocha Ladeira

Matéria: Padrões de Projeto

Nomes: Gabrielli Danker

Turma: BCC 2025.1

Data de entrega: 20 de Fevereiro de 2025

Exercícios

1. Quais são as vantagens e as desvantagens de usar padrões de projetos no desenvolvimento de software?

Vantagens:

* Aumento de produtividade.
* Ajuda a resolver problemas.
* Reutilizáveis em vários projetos.
* Reduz problemas comuns e recorrentes, melhorando a manutenção do sistema.

Desvantagens:

* Complexidade desnecessária: se um padrão for aplicado sem necessidade, pode tornar o código mais complicado e difícil de manter.
* Pode gerar sobrecarga no projeto, especialmente quando padrões são utilizados sem planejamento adequado.
* Nem sempre são a solução ideal para todos os problemas.

1. Pesquise o que são padrões GRASP e encontre três exemplos.

Padrões GRASP ajudam a especificar em que momento é atribuída a responsabilidade de decisão, quando é um comportamento ou elemento ao objeto. A formação do padrões GRASP se consiste nos seguintes padrões: Controlador (Controller), Criador (Creator), Indireção (Indirection), Especialista na informação (Information expert), Alta coesão (High Cohesion), Baixo acoplamento (Loose coupling), Polimorfismo (Polymorphism), Variações protegidas (Protected variations), e Invenção pura (Pure fabrication).

Três exemplos:

* + - 1. Controlador: Define um objeto intermediário que recebe e gerencia solicitações do usuário.

Exemplo prático: Um sistema de pedidos onde um controlador recebe a solicitação de compra do usuário e coordena a criação do pedido.

* + - 1. Especialista na informação: O objeto que possui a informação necessária deve ser responsável pela lógica de negócio.

Exemplo prático: Em um sistema bancário, a classe ContaBancaria é responsável por calcular saldo, pois contém os dados das transações.

* + - 1. Baixo acoplamento: Reduz a dependência entre os objetos para facilitar a manutenção e reutilização do código.

Exemplo prático: Um sistema de e-commerce onde a classe de pagamento não depende diretamente da classe de estoque, permitindo substituição de métodos de pagamento sem afetar outras partes do sistema.

1. Escolha um padrão GoF e implemente um pequeno exemplo em sua linguagem de programação favorita.

# Empresa de Locação de Barcos

from abc import ABC, abstractmethod

# Interface para comportamentos de movimentação

class ComportamentoMovimento(ABC):

@abstractmethod

def mover(self):

pass

# Implementações de comportamentos de movimentação

class MovimentoComMotor(ComportamentoMovimento):

def mover(self):

print("Movendo com motor")

class MovimentoRemando(ComportamentoMovimento):

def mover(self):

print("Remando")

class NavegacaoAVela(ComportamentoMovimento):

def mover(self):

print("Navegando à vela")

# Classe de barco

class Barco:

def \_\_init\_\_(self, comportamento\_movimento):

self.comportamento\_movimento = comportamento\_movimento

def definir\_comportamento\_movimento(self, comportamento\_movimento):

self.comportamento\_movimento = comportamento\_movimento

def mover(self):

self.comportamento\_movimento.mover()

# Testando a empresa de locação de barcos

bateira = Barco(MovimentoComMotor())

iate = Barco(MovimentoComMotor())

canoa = Barco(MovimentoRemando())

jangada = Barco(MovimentoRemando())

barco\_a\_vela = Barco(NavegacaoAVela())

bateira.mover()

iate.mover()

canoa.mover()

jangada.mover()

barco\_a\_vela.mover()

1. Pesquise o que são padrões GoF e encontre cinco exemplos.

Padrões GoF tem como objetivo solucionar problemas comuns de softwares que tenham algum envolvimento na orientação a objetos. São formados por três grupos, os padrões de criação, estruturais e comportamentais.

* Padrões de criação: Exigem um tratamento de como os objetos são criados para atenderem às diversas necessidades.
  + Factory Method;
  + Abstract Factory;
  + Singleton;
  + Builder;
  + Prototype;
* Padrões estruturais: Elaboração, associação e a organização entre objetos e classes/interfaces. Permitem combinar objetos em estruturas mais complexas, ou descrever como as classes são herdadas ou compostas a partir de outras.
  + Adapter;
  + Bridge;
  + Composite;
  + Decorator;
  + Facade;
  + Flyweight;
  + Proxy;
* Padrões comportamentais: Mostram o processo de como os objetos ou classes se comunicam. Em geral, buscam um baixo acoplamento entre os objetos, apesar da comunicação que existe entre eles.
  + Chain of Responsability:
  + Command;
  + Interpreter;
  + Iterator;
  + Mediator;
  + Memento;
  + Observer;
  + State;
  + Strategy;
  + Template;
  + Method;
  + Visitor.

Cinco exemplos:

* + - 1. Singleton: Garante que uma classe tenha apenas uma instância e fornece um ponto global de acesso a ela.

Exemplo prático: Um gerenciador de conexão com o banco de dados, onde apenas uma instância é necessária para evitar múltiplas conexões desnecessárias.

* + - 1. Factory Method: Define uma interface para criar um objeto, mas permite que as subclasses alterem o tipo de objeto que será criado.

Exemplo prático: Um sistema de logs que pode criar diferentes tipos de log (arquivo, console, banco de dados) dependendo da necessidade.

* + - 1. Adapter: Permite que interfaces incompatíveis trabalhem juntas.

Exemplo prático: Um adaptador que permite um software legado se comunicar com um novo sistema via API moderna.

* + - 1. Observer: Define uma dependência um-para-muitos entre objetos para que, quando um objeto mudar de estado, todos os seus dependentes sejam notificados.

Exemplo prático: Um sistema de notificações onde vários usuários são informados quando um evento ocorre (como novas mensagens em um chat).

* + - 1. Decorator: Permite adicionar responsabilidades a um objeto dinamicamente.

Exemplo prático: Um sistema de envio de e-mails onde diferentes funcionalidades (criptografia, logs, notificações extras) podem ser adicionadas sem alterar a classe principal.

1. Pesquise e descubra o nome do padrão de projeto adotado na atividade prática realizada nesta aula.

Padrão Factory Method

A classe Operacao define uma interface abstrata para as operações matemáticas.

As classes Soma, Subtracao, Multiplicacao e Divisao são implementações concretas dessa interface.

A classe OperacaoSeparada atua como uma fábrica, contendo o método estático getOperacao(operador), que retorna a instância apropriada com base no operador passado.

O código principal (main()) não instancia diretamente as classes de operação; em vez disso, ele chama getOperacao(operador), que decide qual objeto criar.