1. Explique em detalhes o que é o padrão de projeto Template Method e como ele se diferencia de outros padrões de projeto comportamentais.

O padrão Template Method é uma poderosa técnica para definir algoritmos com partes variáveis, promovendo reuso de código e flexibilidade, enquanto mantém uma estrutura fixa para o algoritmo geral. Ele se diferencia de outros padrões comportamentais em seu foco na personalização de passos de um algoritmo dentro de uma estrutura rígida predefinida.

2) Descreva a estrutura do Template Method. Quais são os componentes principais e como eles interagem para implementar este padrão?

Um método definido na classe abstrata que estabelece a sequência de passos do algoritmo.

Este método chama outros métodos (alguns implementados na classe abstrata e outros a serem implementados pelas subclasses) para executar as etapas do algoritmo.

Esta interação entre a classe abstrata e suas subclasses permite que a estrutura geral do algoritmo permaneça constante, enquanto os detalhes de certos passos possam ser alterados por subclasses conforme necessário.

3) Quais são as vantagens do uso do padrão Template Method em projetos de software? Forneça exemplos práticos onde essas vantagens são evidentes.

O padrão Template Method oferece uma maneira estruturada de definir algoritmos com partes variáveis, promovendo reutilização de código, facilidade de manutenção, e flexibilidade. Ele é particularmente útil em cenários onde a estrutura do algoritmo é fixa, mas partes específicas do algoritmo variam entre diferentes implementações.

4) Quais são as desvantagens do padrão Template Method e como elas podem impactar a flexibilidade e manutenção de um sistema? Dê exemplos de situações em que essas desvantagens podem surgir.

Desvantagens do Padrão Template Method

Relação Forte entre Classe Abstrata e Subclasses:

O padrão Template Method cria uma dependência estreita entre a classe abstrata e suas subclasses. Mudanças na classe abstrata podem exigir mudanças em todas as subclasses, dificultando a manutenção.

Complexidade na Hierarquia de Classes:

Pode introduzir complexidade adicional na hierarquia de classes, especialmente quando há muitas subclasses. Isso pode dificultar a compreensão do código e a navegação entre as classes.

Dificuldade na Modificação da Estrutura do Algoritmo:

A estrutura do algoritmo é definida na classe abstrata e é fixa. Se for necessário modificar essa estrutura, todas as subclasses podem precisar ser revisadas e alteradas.

Rígido e Menos Flexível:

O padrão pode ser menos flexível em cenários onde é necessário alterar dinamicamente o comportamento do algoritmo em tempo de execução. A variação é restrita aos métodos específicos que as subclasses podem sobrescrever.

Propagação de Mudanças:

Pequenas mudanças na estrutura do método template podem ter um impacto significativo em todas as subclasses, levando a uma maior propagação de mudanças e potencial introdução de erros.

### Exemplos de Situações Onde Essas Desvantagens Podem Surgir

### Mudança na estrutura do algoritmo, impactos nas classes, complexidade na hierarquia de classes, rigidez e menos flexibilidade, propagação de mudanças.

5) Desenvolva um exemplo em Java utilizando o padrão Template Method para um cenário diferente do apresentado (processamento de documentos). Explique como o padrão foi aplicado e como ele melhora a estrutura do código.

// Classe Abstrata

abstract class Beverage {

// Método Template

public final void prepareRecipe() {

boilWater();

brew();

pourInCup();

addCondiments();

}

// Passos comuns, implementados na classe abstrata

private void boilWater() {

System.out.println("Boiling water");

}

private void pourInCup() {

System.out.println("Pouring into cup");

}

// Passos específicos, declarados como abstratos

protected abstract void brew();

protected abstract void addCondiments();

}

// Subclasse Concreta para Chá

class Tea extends Beverage {

@Override

protected void brew() {

System.out.println("Steeping the tea");

}

@Override

protected void addCondiments() {

System.out.println("Adding lemon");

}

}

// Subclasse Concreta para Café

class Coffee extends Beverage {

@Override

protected void brew() {

System.out.println("Dripping coffee through filter");

}

@Override

protected void addCondiments() {

System.out.println("Adding sugar and milk");

}

}

// Classe Principal para Demonstrar o Uso

public class BeverageTest {

public static void main(String[] args) {

Beverage tea = new Tea();

tea.prepareRecipe();

System.out.println();

Beverage coffee = new Coffee();

coffee.prepareRecipe();

}

}

Explicação de Como o Padrão Foi Aplicado

Classe AbstrataBeverage:

* + A classe Beverage define o método template prepareRecipe(), que descreve a sequência de passos necessários para preparar uma bebida: ferver a água, preparar a bebida, colocar na xícara e adicionar condimentos.
  + Os métodos boilWater() e pourInCup() são implementados na classe abstrata, pois são comuns a todas as bebidas.
  + Os métodos brew() e addCondiments() são declarados como abstratos, permitindo que as subclasses forneçam a implementação específica desses passos.

SubclassesTea e Coffee:

* + As subclasses Tea e Coffee estendem a classe Beverage e implementam os métodos brew() e addCondiments(), fornecendo a lógica específica para preparar chá e café, respectivamente.

Como o Padrão Melhora a Estrutura do Código

Reutilização de Código:

* + A lógica comum de preparar bebidas (ferver a água e colocar na xícara) é centralizada na classe abstrata Beverage, evitando duplicação de código nas subclasses.

Clareza e Manutenção:

* + A sequência de passos do algoritmo está claramente definida no método template prepareRecipe(), facilitando a compreensão da lógica de preparação de bebidas.
  + Qualquer mudança na sequência de passos pode ser feita na classe abstrata, impactando todas as subclasses de forma consistente.

Flexibilidade:

* + As subclasses podem facilmente personalizar passos específicos da preparação sem alterar a estrutura geral do algoritmo, permitindo a criação de novas subclasses para outros tipos de bebidas (por exemplo, HotChocolate) de forma simples e consistente.

Encapsulamento:

* + Os detalhes específicos da preparação de cada tipo de bebida são encapsulados nas subclasses, enquanto a estrutura geral do algoritmo é mantida na classe abstrata.