UNICURITIBA – CENTRO UNIVERSITÁRIO

HENRIQUE MAKUCH DA SILVEIRA

**VISÃO E ANÁLISE DE PROJETOS**

**DESIGN E ARQUITETURA**

CURITIBA

2023

HENRIQUE MAKUCH DA SILVEIRA

**VISÃO E ANÁLISE DE PROJETOS**

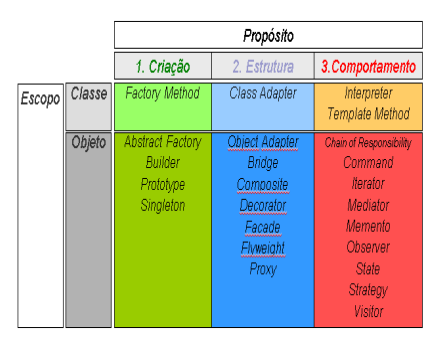
**DESIGN E ARQUITETURA**

Trabalho prático apresentado ao curso de Análise e Desenvolvimento de Software ADS como requisito parcial à obtenção de nota em Modelos Métodos e Técnicas Engenharia de Software.

Professor: Rubem Koide

CURITIBA

2023

****

**Factory Method Classe**

Interface Channel:

Define a interface para os objetos que serão fabricados, no caso, os canais de comunicação.

Classes TCPChannel e UDPChannel:

Implementam a interface Channel, representando diferentes tipos de canais de comunicação.

Classe ChannelFactory:

Implementa um método fábrica estático chamado create().

Este método é responsável por criar instâncias da interface Channel.

No exemplo, o método cria instâncias de TCPChannel, mas poderia ser modificado facilmente para criar instâncias de UDPChannel ou qualquer outra classe que implemente a interface Channel.

Classe Main:

Contém métodos f(), g() e h(), cada um criando uma instância de Channel usando o método fábrica create() da ChannelFactory.

O método main() cria uma instância da classe Main e chama esses métodos para demonstrar o uso do padrão Factory Method.

Funcionamento:

O padrão Factory Method permite que a criação de objetos seja delegada às subclasses ou a classes específicas.

A ChannelFactory centraliza a criação de objetos Channel. Neste exemplo, ela cria instâncias de TCPChannel.

Se, no futuro, for necessário mudar para UDPChannel, basta modificar a implementação do método create() na ChannelFactory, sem afetar o restante do código.

Testando o Código:

Quando você executa o método main(), são criadas instâncias de Main, e os métodos f(), g() e h() são chamados.

Cada chamada a ChannelFactory.create() cria uma instância de TCPChannel e exibe a mensagem "Neste momento, estamos trabalhando com TCPChannel" no console.

Em resumo, o padrão Factory Method permite flexibilidade na criação de objetos, tornando o código mais extensível e fácil de manter, especialmente quando se lida com hierarquias de classes e a criação de objetos pode variar.

**Builder Objeto**

Classe Livro:

Representa a classe final que desejamos construir.

Possui um construtor privado que aceita todos os atributos necessários para a criação do Livro.

Contém um método toString() para representar o Livro como uma string.

Classe Livro.Builder:

É uma classe interna, pública e estática de Livro.

Fornece métodos para configurar os diferentes atributos do Livro (setNome, setAutores, setEditora, setAno).

Cada método retorna uma instância do próprio Builder, permitindo a construção encadeada de chamadas.

Possui um método build() que utiliza os valores configurados para criar uma instância de Livro.

Classe Main:

No método main, são criadas instâncias de Livro usando o Builder.

Para cada livro, são chamados os métodos do Builder para configurar os atributos desejados.

O método build() é chamado no final para criar a instância final do Livro.

Funcionamento:

A utilização do padrão Builder permite a construção de objetos Livro de maneira flexível.

O uso de um Builder separado da classe principal Livro evita a necessidade de construtores com muitos parâmetros e torna o código mais legível.

A construção encadeada de métodos (setNome().setAutores()...) facilita a configuração seletiva dos atributos.

O método build() é responsável por criar a instância final do Livro com os atributos configurados no Builder.

Testando o Código:

No método main, são criados dois Livros, cada um usando um conjunto diferente de métodos do Builder para configurar seus atributos.

Os Livros resultantes são então impressos na saída padrão.

Benefícios do Padrão Builder:

Flexibilidade na criação de objetos com diferentes configurações.

Melhora a legibilidade do código, especialmente quando há muitos atributos opcionais.

Evita a necessidade de múltiplos construtores ou construtores com muitos parâmetros.

Em resumo, o padrão Builder é uma abordagem eficaz para criar objetos complexos, permitindo a configuração passo a passo de seus atributos, proporcionando flexibilidade e melhor legibilidade.

**Adaptadores Classe**

Classes Concretas de Projetores:

ProjetorSamsung: Representa um projetor da Samsung com um método turnOn() para ligar.

ProjetorLG: Representa um projetor da LG com um método enable(int timer) para ligar com um temporizador.

Interface Projetor:

Define uma interface comum para abstrair o comportamento de ligar um projetor.

Adaptadores:

AdaptadorProjetorSamsung e AdaptadorProjetorLG implementam a interface Projetor.

Internamente, eles contêm instâncias dos projetores concretos (ProjetorSamsung e ProjetorLG) e implementam o método liga().

SistemaControleProjetores:

SistemaControleProjetores é uma classe que não tem conhecimento dos projetores concretos.

Possui um método init(Projetor projetor) que liga qualquer projetor, sem precisar saber se é Samsung ou LG.

Main:

No método main, são criadas instâncias dos adaptadores (AdaptadorProjetorSamsung e AdaptadorProjetorLG) e do SistemaControleProjetores.

O método init do SistemaControleProjetores é chamado com instâncias dos adaptadores como argumentos.

Explicação do Funcionamento:

A classe SistemaControleProjetores aceita qualquer objeto que implemente a interface Projetor.

Os adaptadores AdaptadorProjetorSamsung e AdaptadorProjetorLG adaptam os métodos específicos dos projetores concretos para a interface comum Projetor.

Isso permite que o SistemaControleProjetores ligue qualquer projetor, independentemente da sua implementação concreta, mostrando a flexibilidade proporcionada pelo padrão Adapter na integração de diferentes sistemas ou classes com interfaces incompatíveis.

**Proxy Objeto**

Classe Auxiliar:

Book é uma classe simples que representa um livro. Ela possui um campo nome e um construtor para inicializar o nome.

Interface BookSearchInterface:

Define uma interface comum para pesquisa de livros. A única operação é getBook(String ISBN) que retorna um livro com base no ISBN.

Classe BookSearch:

Implementa a interface BookSearchInterface.

Representa o objeto base que realiza a pesquisa real de livros.

Ao chamar getBook, ele imprime uma mensagem indicando que está pesquisando no objeto base e retorna um livro se o ISBN for "2".

Classe BookSearchProxy:

Também implementa BookSearchInterface.

Representa o Proxy que envolve o objeto base BookSearch.

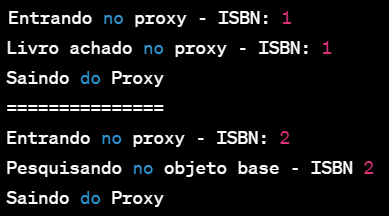
Ao chamar getBook, ele primeiro verifica se o ISBN é "1". Se for, ele retorna um livro sem consultar o objeto base. Caso contrário, repassa a chamada para o objeto base e retorna o resultado.

Main:

No método main, uma instância de BookSearch é criada como o objeto base.

Em seguida, é criada uma instância de BookSearchProxy que envolve o objeto base.

Duas chamadas ao método getBook são feitas, uma com ISBN "1" e outra com ISBN "2".



Explicação do Funcionamento:

O Proxy (BookSearchProxy) atua como um intermediário entre o cliente e o objeto base (BookSearch).

O Proxy pode realizar ações adicionais antes ou depois de chamar o objeto base.

No exemplo, o Proxy verifica se o ISBN é "1" antes de consultar o objeto base, demonstrando como o Proxy pode evitar a chamada real ao objeto base se certas condições forem atendidas.

Em resumo, o padrão Proxy é útil quando você deseja controlar ou adicionar comportamentos específicos ao acesso a um objeto, sem modificar o próprio objeto base. No exemplo fornecido, o Proxy evita a chamada ao objeto base se o ISBN for "1".

**Template Method Classe**

Classe Abstrata Funcionario:

Define um Template Method chamado calcSalarioLiquido.

Contém métodos abstratos calcDescontosPrevidencia, calcDescontosPlanoSaude e calcOutrosDescontos. Esses métodos são chamados no Template Method e devem ser implementados pelas subclasses.

Subclasse FuncionarioCLT:

Herda de Funcionario e implementa os métodos abstratos calcDescontosPrevidencia, calcDescontosPlanoSaude e calcOutrosDescontos.

O Template Method (calcSalarioLiquido) está na classe base (Funcionario), e os métodos abstratos são implementados na subclasse (FuncionarioCLT).

Main:

Cria uma instância de FuncionarioCLT com um salário de 1000.

Chama calcSalarioLiquido na instância de FuncionarioCLT para calcular o salário líquido.

Imprime o resultado.

Explicação do Funcionamento:

O padrão Template Method é utilizado aqui para definir a estrutura geral do cálculo do salário líquido na classe base (Funcionario).

O Template Method (calcSalarioLiquido) chama métodos abstratos que representam passos específicos do algoritmo de cálculo. Esses passos são implementados nas subclasses.

Na subclasse FuncionarioCLT, os métodos abstratos são implementados com lógica específica para funcionários CLT.

Isso permite que o algoritmo geral (calcSalarioLiquido) permaneça inalterado na classe base, enquanto as subclasses podem fornecer implementações específicas para cada passo do algoritmo.

O padrão Template Method é eficaz quando você tem um algoritmo com uma estrutura fixa, mas com etapas que podem variar entre as subclasses. Ele permite que as subclasses forneçam implementações específicas para as etapas variáveis, mantendo a estrutura geral no código da classe base.

**Visitor Objeto**

Classe Abstrata Veiculo:

É a raiz de uma hierarquia de classes que representa veículos.

Contém um método abstrato accept(Visitor v) que aceita visitas de objetos Visitor.

A classe é aberta para visitas, mas não sabe exatamente o que o Visitor vai fazer com seus dados.

Classes Concretas Carro e Onibus:

Estendem Veiculo e implementam o método accept(Visitor v) chamando o método apropriado do Visitor.

Permitem que diferentes tipos de veículos sejam visitados de maneira específica.

Interface Visitor:

Define métodos visit para cada tipo de veículo (concreto).

Classes que implementam essa interface podem realizar operações específicas em cada tipo de veículo sem modificar as classes dos veículos.

Classe PrintVisitor:

Implementa a interface Visitor.

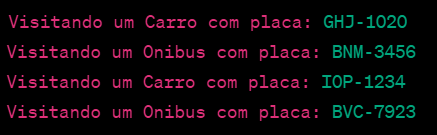
Imprime as placas dos carros e ônibus visitados.

Main:

Cria uma lista de veículos (carros e ônibus).

Cria uma instância de PrintVisitor.

Itera sobre a lista de veículos, chamando accept para cada veículo e passando o PrintVisitor como parâmetro.



Explicação do Funcionamento:

O padrão Visitor permite definir uma família de algoritmos (métodos visit na interface Visitor) sem alterar as classes que precisam desses algoritmos (classes Veiculo, Carro, Onibus).

Cada classe concreta de veículo implementa o método accept, que aceita um objeto Visitor e chama o método apropriado do Visitor.

Isso permite que diferentes tipos de veículos sejam visitados por diferentes visitantes, permitindo a extensibilidade e flexibilidade do código.

No exemplo, o PrintVisitor imprime as placas dos veículos, mas você poderia ter outros visitantes que realizam diferentes operações nos veículos sem modificar suas classes.

O padrão Visitor é útil quando você precisa adicionar novas operações a objetos sem modificar suas classes. Ele permite que você externalize operações para classes visitantes, tornando o código mais modular e facilitando a adição de novas funcionalidades sem alterar as classes existentes.