Relatório da Análise de Projetos

Grupo: Daniela Faversani Barroso, Rafaela Perin Bianek , John Lenon Galli e Wellington Ruan da Silva

Factory Method(Classe)

Introdução

O padrão Factory Method, ou Método de Fábrica, é um padrão de projeto de software que permite criar objetos de diferentes tipos, mas com uma interface comum.

No código, você tem uma classe base chamada "Creator" (Criador) que define um método chamado "FactoryMethod" (Método de Fábrica). Essa classe é responsável pela criação de objetos, mas não sabe exatamente qual tipo de objeto será criado.

Em seguida, você tem subclasses da classe "Creator" que implementam o método "FactoryMethod". Cada uma dessas subclasses é responsável por criar um tipo específico de objeto, que é uma classe derivada da classe base.

Dessa forma, quando você precisa criar um objeto, em vez de instanciar diretamente a classe do objeto, você chama o método "FactoryMethod" da classe "Creator". O método "FactoryMethod" é então implementado nas subclasses, que retornam uma instância do objeto desejado.

Isso torna o código mais flexível, pois você pode adicionar novos tipos de objetos simplesmente criando uma nova subclass da classe "Creator" e implementando o método "FactoryMethod".

Prós

- A utilização desse método ajuda a evitar acoplamentos firmes entre o criador e os produtos concretos
- Facilita a manutenção do código, já que é possível mover o código de criação do produto para um único local no programa.
- É possível a introdução de novos tipos de produtos no programa sem quebrar o código cliente existente.

Contras

- O código pode se tornar mais complicado, devido a necessidade de introduzir muitas subclasses novas para implementar o padrão.

Análise do código

No código pronto é possível analisar que a classe *ChannelFactory* é a superclasse, já que centraliza a criação dos objetos que implementam a interface *Channel.*

Singleton(Objeto)

Introdução

O Singleton é um padrão de projeto criacional que permite a você garantir que uma classe tenha apenas uma instância(objeto), enquanto provê um ponto de acesso global para essa instância.

Análise do Código

Para implementar o padrão Singleton, você precisa fazer o seguinte:

1. Tornar o construtor da classe privado, para que não seja possível criar instâncias diretamente fora da classe.
```
  private Logger() {} // proíbe clientes de chamar new
Logger()
```
2. Criar uma variável estática privada que armazena a única instância da classe.
```
 private static Logger instance; // instância única
```
3. Criar um método estático público, geralmente chamado de "getInstance", que retorna a instância existente ou cria uma nova caso ela ainda não exista. Esse método é responsável por garantir que apenas uma instância seja criada e retornada.
```
  public static Logger getInstance() {
    if (instance == null) // 1a vez que chama-se
getInstance
```

Dessa forma, sempre que você precisar usar a classe Singleton, você chama o método "getInstance" para obter a instância existente. Se a instância já existir, ela é retornada. Caso contrário, uma nova instância é criada e retornada.

Prós

- Você pode ter certeza de que uma classe só terá uma única instância.
- Você ganha um ponto de acesso global para aquela instância.
- O objeto singleton é inicializado somente quando for pedido pela primeira vez.

Contras

- Viola o princípio de responsabilidade única. O padrão resolve dois problemas de uma só vez.
- O padrão Singleton pode mascarar um design ruim, por exemplo, quando os componentes do programa sabem muito sobre cada um.
- O padrão requer tratamento especial em um ambiente multithreaded para que múltiplas threads não possam criar um objeto singleton várias vezes.
- Pode ser difícil realizar testes unitários do código cliente do Singleton porque muitos frameworks de teste dependem de herança quando produzem objetos simulados.

Visitor(Objeto)

Introdução

O Visitor é um padrão de projeto comportamental que permite que você separe algoritmos dos objetos nos quais eles operam.

Esse padrão propõe a criação de uma classe separada chamada “Visitor” que contêm métodos para casa tipo de objeto na estrutura. Esses métodos representam as operações que você deseja executar nos objetos

Prós

- Você pode introduzir um novo comportamento que pode funcionar com objetos de diferentes classes sem mudar essas classes.
- Você pode mover múltiplas versões do mesmo comportamento para dentro da mesma classe.
- Um objeto visitante pode acumular algumas informações úteis enquanto trabalha com vários objetos.

Contras

- Você precisa atualizar todos os visitantes a cada vez que a classe é adicionada ou removida da hierarquia de elementos.
- Visitantes podem não ter seu acesso permitido para campos e métodos privados dos elementos que eles deveriam estar trabalhando.

Análise do código

A estrutura básica do padrão Visitor consiste em três principais componentes:

1. Element (Elemento): Define uma interface que representa o elemento sobre o qual o Visitor irá agir. Cada classe concreta que implementa essa interface aceita visitantes e, portanto, deve fornecer um método accept(Visitor).

2. ConcreteElement (ElementoConcreto): Implementa a interface Element e define o método accept(Visitor) para aceitar um objeto Visitor. Este método, por sua vez, chama o método apropriado do Visitor que corresponde a essa classe.

3. Visitor (Visitante): Define uma interface com um método para cada classe concreta de Element. Cada método recebe um objeto Element como parâmetro, permitindo que o Visitor acesse e atue sobre o Element. As classes concretas que implementam essa interface fornecem a implementação real desses métodos.

4. ConcreteVisitor (VisitanteConcreto): Implementa a interface Visitor e fornece a implementação específica para cada operação definida no Visitor.

5. ObjectStructure (Estrutura de Objetos): Mantém uma coleção de objetos Element e pode iterar sobre eles, permitindo que os visitantes acessem cada elemento na estrutura.

Adapter (Classe)

Introdução

O adapter é um padrão de projeto estrutural que permite uma classe com diferentes interfaces (o que torna elas incompatíveis) possam colaborar entre si.

Como exemplo: se você possui uma aplicação de monitoramento que utiliza uma classe no formato X, e no seu código você implementa uma nova classe com formato Y, você cria um adaptador que converte a interface de uma classe para que ou outro posso entendê-la e torne um ela visível no padrão de seu código.

Deste modo o adaptador obtém uma interface que será compatível com as classes existentes, e ao utilizar essa interface as classes existentes podem ser chamadas através do adaptador com segurança sem que elas apresentem erros no código.

Deste modo a classe Adaptador e utilizada quando você quer usar uma classe existente, mas sua interface não compatível com o resto do seu código, assim o adapter permite que você crie uma classe de meio termo que serve como um tradutor entre seu código e a classe antiga, uma classe de terceiro, ou qualquer outra classe com uma interface estranha.

No exemplo do exercício a Interface Projetor será utilizado tanto pela classe da Samsumg quanto da LG.

Decorator(Objeto)

Introdução

O Decorator é um padrão de projeto estrutural que permite que você acople novos comportamentos para objetos ao fornecê-los uma extensão de funcionalidade as subclasses.

Deste modo o padrão Decorator você utiliza quando precisa ser capaz de projetar comportamentos adicionais para objetos em tempo de execução sem quebrar o código, assim o Decorator lhe permite estruturar sua lógica, e compor objetos com várias combinações durante a execução e combinando-as para que possa tratar de todos esses objetos da mesma forma, para que todos funcionem na mesma interface comum.

No exemplo do exercício você compacta e descompacta a mensagem TCP em um formato que seja exibido na interface em um método que seja compatível com a interface.

Templated Method(Classe)

Introdução

O Template Method é um padrão de projeto comportamental que define o esqueleto de um algoritmo na superclasse, mas deixa as subclasses sobrescreveram etapas específicas do algoritmo sem modificar sua estrutura.

O padrão do Template Method sugere que você quebre um algoritmo em uma série de etapas, transforme essas etapas em métodos, e coloque uma séria de chamadas para esses métodos dentro de um único método padrão. As etapas podem ser tento abstratas, ou ter alguma implementação padrão. Para usar o algoritmo, o cliente deve fornecer sua própria subclasse, implementar todas as etapas abstratas, e sobrescrever algumas das opcionais se necessário.

Deste modo o Template Method e utilizando quando você quer deixar os clientes estender apenas etapas particulares de um algoritmo a sua estrutura, permitindo assim eu você transforme um algoritmo monolítico em uma série de etapas individuais que podem facilmente ser estendidas por subclasses enquanto ainda mantém intacta a estrutura definida em uma superclasse.

No exemplo em questão o código permite que o próprio usuário possa calcular seu salário, assim o cliente sobrescreve apenas certa parte de um algoritmo, tornando o menos afetados por mudanças que acontece por outras partes do algoritmo.