**GRUPO 8**

Template Method

Visitor

Interceptor

**Participantes**

Ângelo Rodrigues

Lívia Negrini

Luca Silvestri

Lucas Araújo

Ricardo Tamahe

**Interceptor**

**Definição –** O que é?

O padrão de design do interceptor é usado para interceptar uma solicitação. O uso principal desse padrão é na implementação da política de segurança.

Podemos usar esse padrão para interceptar as solicitações de um cliente para um recurso. Na interceptação podemos verificar a autenticação e autorização do cliente para o recurso que está sendo acessado.

É utilizado para aumentar o ciclo do método, aumentando as etapas e inserindo mais verificações de segurança durante o ciclo. Os principais aspectos do padrão são que a mudança é transparente e usada automaticamente. Em essência, o resto do sistema não precisa saber que algo foi adicionado ou alterado e pode continuar funcionando como antes.

**Histórico –** Breve contexto histórico.

Surgiu da necessidade de se aplicar um desenvolvimento de sistemas que pode ser estendido de maneira transparente, focando na implementação de medidas de segurança através das interceptações do ciclo.

**Formas de utilização/Exemplos –** Como se utiliza? Exemplos.

O uso principal desse padrão é na implementação da política de segurança. Podemos usar esse padrão para interceptar as solicitações de um cliente para um recurso. Na interceptação podemos verificar a autenticação e autorização do cliente para o recurso que está sendo acessado.

Um interceptor é usado quando sistemas de software ou frameworks desejam oferecer uma maneira de alterar seu ciclo de processamento usual. Por exemplo, uma sequência de processamento típica para um servidor web é receber um URI do navegador, mapeá-lo para um arquivo no disco, abrir o arquivo e enviar seu conteúdo para o navegador. Qualquer um destes passos pode ser substituído ou alterado, substituindo a maneira como os URIs são mapeados para nomes de arquivos ou inserindo uma nova etapa que processa o conteúdo dos arquivos.

**Visitor**

**Definição –** O que é?

O Visitor pode ser compreendido como um padrão de projeto comportamental, isto é, são destinados ao gerenciamento de algoritmos e a sua consequente distribuição entre classes e objetos. Como descrito na própria nomenclatura, haverá dentro do sistema uma espécie de visitante, que possibilita que novas ações sejam executadas sobre um respectivo objeto sem modificar sua classe, mantendo separados.

Para implementá-lo, é necessário passar o objeto que irá executar as operações como um argumento do método do Visitor, que conseguirá obter todas as informações relevantes relacionadas ao objeto.

A estrutura deverá ser composta da seguinte maneira:

**Interface visitante:** Essa parte do sistema se responsabilizará por declarar um grupo de métodos ou operações visitantes que irão receber como argumento os objetos das classes concretas que constituem um respectivo sistema. Além de ser necessário que os parâmetros sejam diferentes, é igualmente relevante determinar cada um dos métodos supracitados a uma classe correspondente.

**Interface do elemento:** Em seguida, é necessário que a interface do elemento declare um método Accept() com o parâmetro com o tipo do visitante, que irá recebê-lo como um argumento e aceitá-lo.

**Visitante concreto:** Nesta etapa, serão implementados todos os métodos definidos em sua interface, estabelecendo fragmentos do algoritmo para a classe correspondente. Este poderá implementar repetidamente o mesmo método, sendo cada um deles adaptado para receber elementos concretos diversos.

**Elemento concreto:** É imprescindível que cada elemento concreto implemente o método de aceitação já declarado anteriormente em sua interface. O intuito de sua utilização é o redirecionamento da chamada para o método visitante mais adequado. Além do mais, vale ressaltar que, independentemente do fato de já haver sido declarada no elemento base, persiste a necessidade de sobrescrevê-lo nas classes herdeiras, que também deverão chamar o método apropriado do objeto visitante.

Cliente: Por sua vez, o cliente ficará responsável por instanciar novos objetos visitantes e informá-los aos elementos concretos por via dos métodos de aceitação.

**Benefícios**

**Código limpo:** A utilização deste padrão de projeto permite que as regras de negócio que regem determinada aplicação sejam desobstruídas, evitando que estruturas complexas excedentes impeçam o cumprimento de seu objetivo principal.

**Single Responsibility Principle (SRC):** Caracterizando-se como o primeiro princípio do padrão S.O.L.I.D. ele define que uma classe deve ser alterada por um único motivo, caso contrário, já não estaria sendo coerente com as definições da Programação Orientada a Objetos (POO, portanto, uma objeto deve fazer apenas aquilo que se propôs a fazer, portanto, o Visitor permite que as responsabilidades sejam melhor distribuídas, separando os algoritmos complexos em objetos auxiliares.

**Open/Closed Principle (OCR):** Esse princípio é correspondido a partir do momento em que o software desenvolvido se tor extensível, isto é, possibilita que novos métodos sejam adicionados sem que os anteriores ou o próprio código originais das classes sejam alterados, conceito compatível com a proposta do Visitor.

**Malefícios**

**Atualizações:** Sempre que houver alterações nas classes pertencentes a hierarquia de elementos, faz-se necessário a atualização de todos os visitantes.

**Encapsulamento:** Os Visitantes podem acabar sendo barrados por métodos privados, ou seja, protegidos de intervenções externas.

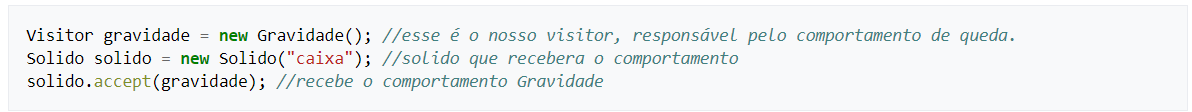
**Para que serve –** Utilização, onde se utiliza?

O padrão Visitor permite que você execute uma operação sobre um conjunto de objetos com diferentes classes ao ter o objeto visitante implementando diversas variantes da mesma operação, que correspondem a todas as classes alvo. O Visitor permite que se crie uma nova operação sem que se mude a classe dos elementos sobre os quais ela opera.

**Formas de utilização/Exemplos –** Como se utiliza? Exemplos.

O padrão utiliza dois métodos, um método Visit() que define a função que será implementada; e um método Accept() que será utilizado para definir as funcionalidades do Visit() para outras classes.

Exemplo:



**Breve resumo sobre padrões similares, caso exista –** Quais são os padrões similares? O que existe no mercado atual?

O Visitor pode ser tratado como uma poderosa versão do padrão Command. Seus objetos permitem executar operações sobre vários objetos de diferentes classes.

**Template Method**

**Definição –** O que é?

O Padrão de Projeto Template Method define os passos de um algoritmo e permite que a implementação de um ou mais desses passos seja fornecida por subclasses. Assim, o Template Method protege o algoritmo e fornece métodos abstratos para que as subclasses possam implementá-los.

Sua definição oficial é: “O Padrão Template Method define o esqueleto de um algoritmo dentro de um método, transferindo alguns de seus passos para as subclasses. O Template Method permite que as subclasses redefinam certos passos de um algoritmo sem alterar a estrutura do próprio algoritmo”.

Portanto, ele basicamente oferece um método que define um algoritmo (uma sequência de passos) que pode, por sua vez, ser definido como abstrato para posteriormente ser implementado por uma subclasse. Pode-se notar que a estrutura do algoritmo fica inalterada mesmo com as subclasses fazendo parte da implementação.

**Histórico –** Breve contexto histórico.

A ideia inicial de [padrão de projeto](http://www.devmedia.com.br/curso/padroes-de-projeto-em-java/25) surgiu em 1977 com Christopher Alexander na área de Arquitetura (prédios e cidades). Seus livros e suas ideias foram usados de inspiração para os desenvolvedores de software. Além da arquitetura e do desenvolvimento de software, outras áreas como a Química e as áreas da Engenharia também possuem catálogos de soluções para problemas recorrentes.

Na área do desenvolvimento de software existe um catálogo de soluções desde 1995 com o livro "Design Patterns - Elements of Reusable Object-Oriented Software". Seus idealizadores foram Gamma, Helm, Johnson e Vlissides. Nesse livro são descritos diversos Padrões de Projetos. Com o catálogo de Padrões de Projetos passamos a ter um vocabulário em comum, soluções descritas e com nomes descritivos, entre diversas outras vantagens.

**Para que serve –** Utilização, onde se utiliza?

* Utilizamos o padrão Template Method quando queremos deixar os clientes estender apenas etapas particulares de um algoritmo, mas não todo o algoritmo e sua estrutura.
* Utilizamos o padrão quando temos várias classes que contém algoritmos quase idênticos com algumas diferenças menores. Como resultado, você pode querer modificar todas as classes quando o algoritmo muda.
* Também pode ser usado para implementar as partes invariantes de um algoritmo uma só vez e deixar para as subclasses a implementação do comportamento que pode variar. Ou quando o comportamento comum entre subclasses deve ser fatorado e concentrado numa classe comum para evitar duplicação de código.

**Formas de utilização/Exemplos –** Como se utiliza? Exemplos.

Template Methods são frequentemente usados ​​em estruturas ou bibliotecas de uso geral que serão usadas por outro desenvolvedor. Um exemplo é um objeto que dispara uma sequência de eventos em resposta a uma ação, por exemplo, uma solicitação de processo. O objeto gera um evento 'pré-processo', um evento 'processo' e um evento 'pós-processo'. O desenvolvedor tem a opção de ajustar a resposta imediatamente antes do processamento, durante o processamento e imediatamente após o processamento.

O Template Method é usado em frameworks, onde cada um implementa as partes invariáveis ​​da arquitetura de um domínio, deixando “espaços reservados” para opções de customização.

**Breve resumo sobre padrões similares, caso exista –** Quais são os padrões similares? O que existe no mercado atual?

Alguns padrões similares ao Template Method são o Factory Method, que é sua especialização, e o Strategy, que, diferente do Template Method, que é baseado em herança, o Strategy é baseado em composição. Enquanto o primeiro permite que você altere partes de um algoritmo ao estender essas partes em subclasses, o segundo permite que você altere partes do comportamento de um objeto ao suprir ele como diferentes estratégias que correspondem a aquele comportamento. O Template Method funciona a nível de classe, o Strategy trabalha a nível de objeto, permitindo que você troque os comportamentos durante a execução.