Aula 25
Padrões de desenho

Iniciativa Conjunta:
Com o apoio de:

Padrões de Desenho

- Objetivo é tornar componentes reutilizáveis que facilitem a padronização
- Permitem a agilidade em soluções de problemas recorrentes
- Cada padrão:
  - Descreve um problema recorrente
  - Captura a estrutura estática e dinâmica, assim como a colaboração entre os principais atores
- Existem dois tipos de padrões: GRASP (General Responsibility Assignment Software Patterns) e GoF (Gang of Four). Os mais comuns e usados são os GoF, que estão descritos nos slides.

Padrões de Desenho

- Padrões de Desenho
  - **Abstract Factory**, Builder, Prototype, Object pool, **Factory method**, **Singleton**
- Padrões estruturais
  - Private class data, Composite, Business Delegate, Adapter, Decorator, Flyweight, Bridge, Facade, Proxy
- Padrões comportamentais
  - Chain of Responsibility, Iterator, State, **Command**, Mediator, Strategy, Interpreter, Memento, Template method, **Observer**, Visitor

Estratégia (strategy/policy)

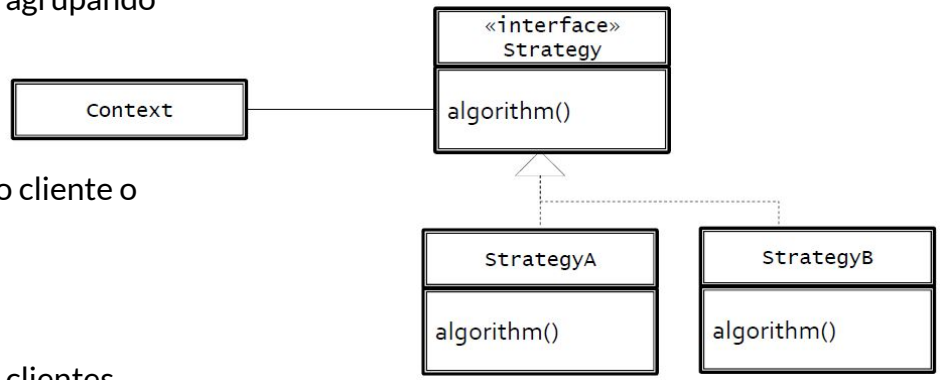
- Algoritmo concreto pode variar independentemente do cliente/contexto.
- Permite definir novas operações sem alterar as classes dos elementos sobre os quais opera.

Vantagens:

- Através de herança pode-se factorizar os vários algoritmos agrupando o que é comum. ·
- O algoritmo pode variar independentemente do contexto.
- Muito fácil a comutação de algoritmo e possível extensão.
- Eliminam as instruções condicionais implícitas no código.
- Fornece-se várias soluções para o mesmo problema sendo o cliente o seleccionador.

Desvantagens:

- Os clientes ficam inerentes à implementação.
- Têm que conhecer os algoritmos em causa
- Só se deve utilizar quando é realmente significativo para os clientes
- Podem ser criadas strategies que nunca vão ser utilizadas por parte do cliente o que aumenta o número de objectos criados

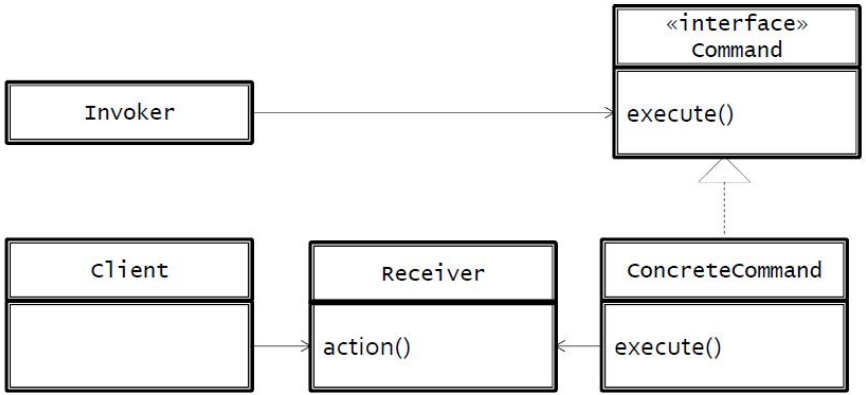


Comando (command)

- É um padrão no qual um objeto é usado para encapsular toda a informação necessária para executar uma ação ou acionar um evento em um momento posterior.
- Transformar pedidos em objectos. Estes pedidos (objectos) podem ser memorizados e enviados a outros objectos

Vantagens:

- Separa o objeto que invoca a operação do que sabe executar essa operação.
- Os comandos são objectos, que podem ser manipulados e estendidos.
- Podem ser agregados em comandos compostos.
- Facilita introdução de novos comandos.
- Os comandos podem fazer a ligação entre o receptor e as acções ou podem fazer logo tudo.
- Suportam redo e undo visto que pode haver um método que indique como é desfeito ou refeito aquele comando.

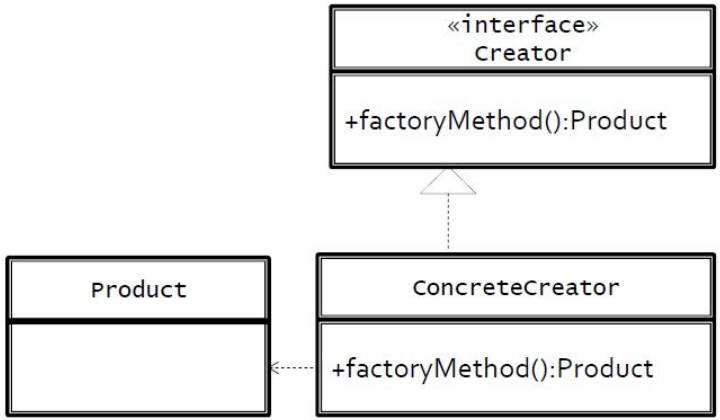


Método Fábrica (Factory method)

- Cria um objeto sem que seja especificada a classe
- O código interage somente com a interface resultante ou classe abstrata, de modo que funcionará com quaisquer classes que implementem essa interface ou que estenda essa classe abstrata.

Vantagens:

- O Factory Method Pattern permite que as subclasses escolham o tipo de objetos a serem criados.
- Ele promove o acoplamento fraco, eliminando a necessidade de vincular classes específicas do aplicativo ao código. Isso significa que

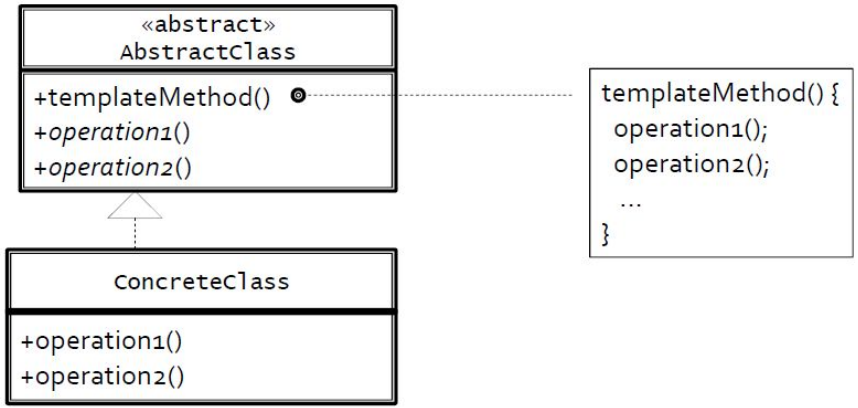


Método modelo (Template method)

- Define o esqueleto do algoritmo sem definir os detalhes.

Vantagens:

- Permite factorizar partes comuns
- Implementa um sistema de controlo, no qual a classe de base é que chama as classes derivadas



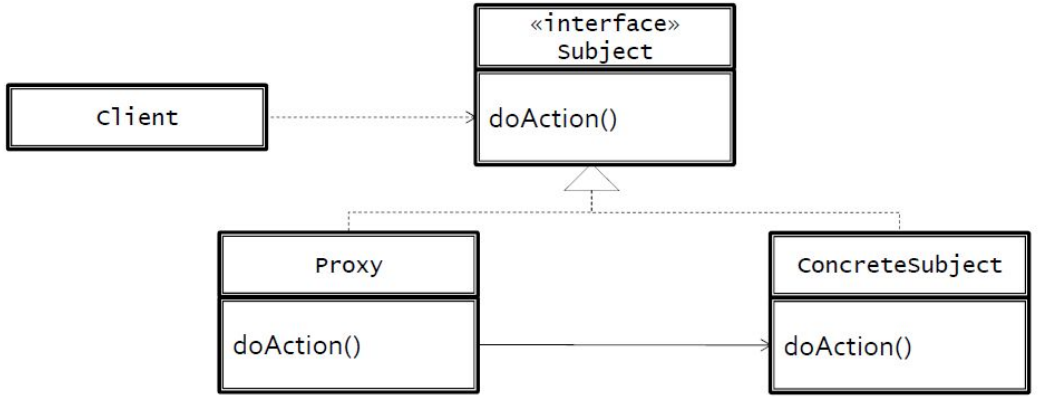
Singleton

- Este padrão garante a existência de apenas uma instância de uma classe, mantendo um ponto global de acesso ao objeto.


```
package mypackage;

public final class MySingleton {

    private static final MySingleton INSTANCE =
        new MySingleton();

    private MySingleton() {
        assert INSTANCE == null : …;
    }

    public static MySingleton getInstance() {
        return INSTANCE;
    }

    …
}
```

Proxy

- A classe funciona como representante de outra, passando os pedidos e devolvendo as respostas em seu nome, eventualmente limitando o acesso a esta.
- Principais funcionalidades:
  - Prover um substituto ou placeholder para outro objeto controlar o acesso
  - Usar um nível extra de indireção para fornecer acesso distribuído ou controlado
  - Adicionar um agregador e delegator para proteger o componente real

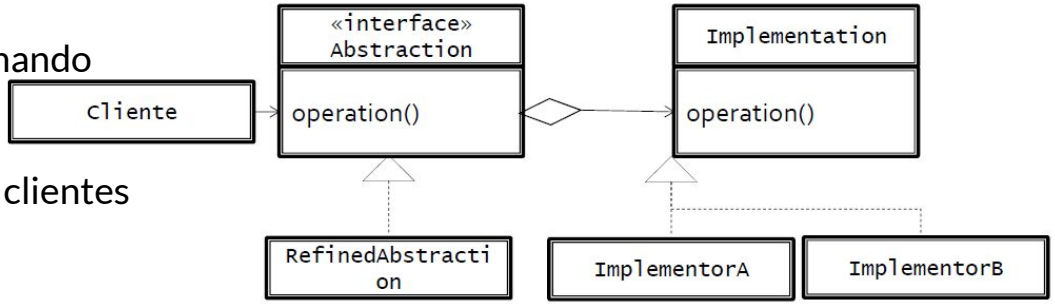


Bridge

- O objetivo é desacoplar a abstração da implementação
- Separar as hierarquias de classes (abstrações e implementações específicas). Todas as operações das subclasses são implementadas recorrendo às operações abstractas. Chama-se bridge ao relacionamento entre as abstrações e as implementações.

Vantagens:

- Separa interface e implementação, eliminando dependências de compilação.
- Melhora a extensibilidade
- Esconde detalhes da implementação aos clientes



Peso-pluma (Flyweight)

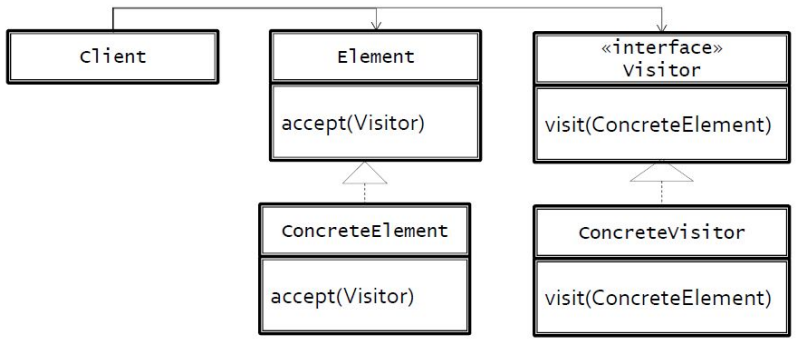
- Padrão que minimiza o consumo de memória partilhando recursos com outros objetos da mesma classe
- Propriedades ou atributos com os mesmos valores, partilham o mesmo espaço de memória.

Visitante (Visitor)

- Permite que se crie uma nova operação/operações sem que se muda a classe dos elementos sobre as quais ela opera.

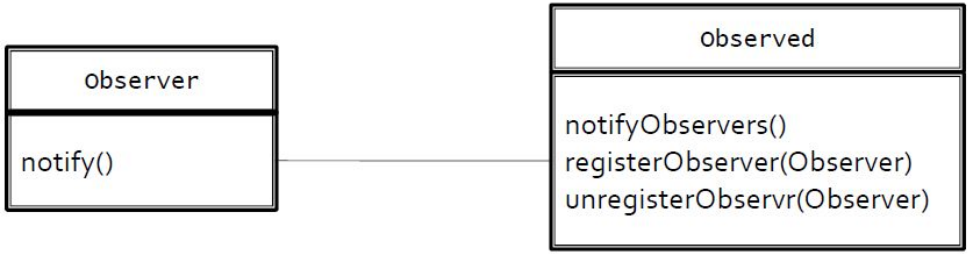
```
//concrete element
public class Book implements Visitable{
 private double price;
 private double weight;

 //accept the visitor
 public void accept(Visitor visitor) {
   visitor.visit(this);
 }
 public double getPrice() {
   return price;
 }
 public double getWeight() {
   return weight;
 }
}
```

```
//Element interface
public interface Visitable{
 public void accept(Visitor visitor);
}
```



Observador (Observer)

- O observado possui uma lista de observador que avisa quando há uma atualização



Vantagens:

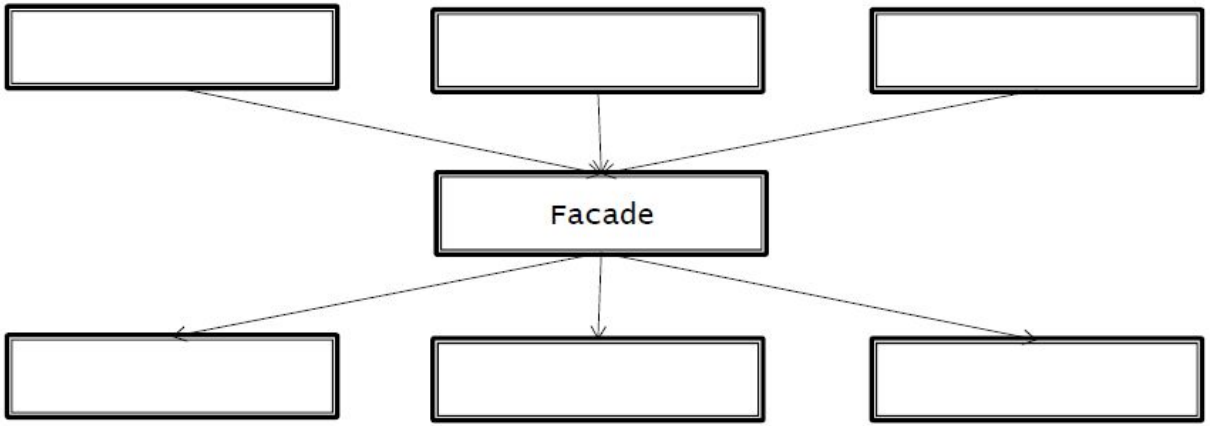
- Permite variar Observers e Subjects de modo independente
- Permite adicionar Observers sem modificar o Observable ou outros Observers
- A ligação entre elementos observáveis e observadores é abstracta.
- O observável só conhece a classe abstracta “Observer”. Podendo este variar de várias formas (polimorfismo) ·
- Permite comunicação broadcast que verifica o protocolo distribuição de mensagens por todos os observers.

Desvantagens:

- A alteração do elemento observável pode ter custos elevados. Uma operação básica pode dar em muitas alterações supérfluas.

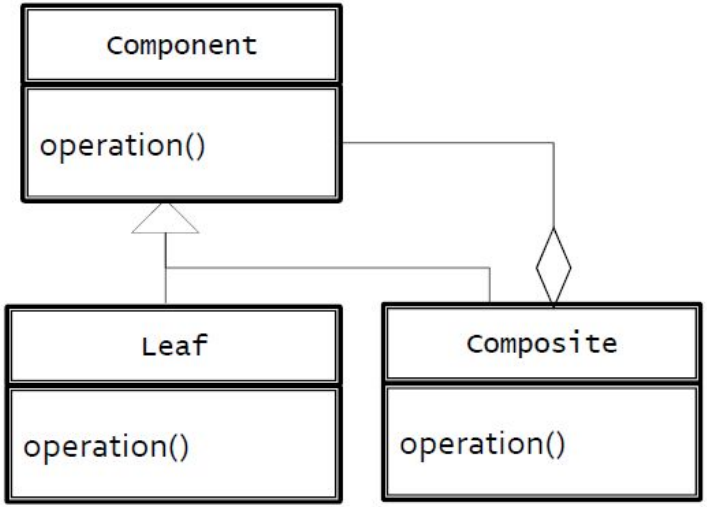
Fachada (Facade)

- Usado quando um sistema é muito complexo ou difícil de entender. Permite esconder as complexidades e disponibiliza interfaces mais simplificadas ao cliente.



Composto (Composite)

- Grupo de objetos que podem ser tratados como uma única instância de um objeto.
- A intenção é compor objetos em estruturas de árvore para representar hierarquia partes-todo.



Vantagens:

- Cliente manipula objectos que podem ser composições ou objectos primitivos. Trata-os de forma uniforme
- A composição é feita recursivamente sem influência do cliente.
- Facilidade na inserção de elementos novos visto que o cliente manipula um elemento genérico.

Desvantagens:

- Desenho pode ser demasiadamente geral o que pode trazer problemas se quiser impor restrições aos elementos das composições.

Referências

- PSDR e SFBS, LEIC, IST
- POO, DCTI, ISCTE
- Documentação Java
- https://www.javatpoint.com/factory-method-design-pattern
- https://refactoring.guru/design-patterns/flyweight/java/example
- https://www.baeldung.com/java-visitor-pattern