Arquiteturas Aplicacionais

Exercício nº2 – Design Patterns

15 de março de 2021

Grupo

Filipa Alves dos Santos (A83631)

Hugo André Coelho Cardoso (A85006)

João da Cunha e Costa (A84775)

Luís Miguel Arieira Ramos (A83930)

Válter Ferreira Picas Carvalho (A84464)





Mestrado Integrado em Engenharia Informática

Universidade do Minho

Índice de conteúdos

1. Introdução

O objetivo deste segundo exercício foi fazer uma pesquisa sobre *design patterns*, mais especificamente sobre Inversion of Control (IoC) e Dependency Injection (DI). Também foi pedido que se apresentasse um design adicional que não tivesse sido abordado na aula, ao que o grupo escolheu o Facade.

Um *design pattern* é um mecanismo de reutilização de especificação e arquitetura. É uma solução reutilizável para resolver um problema recorrente de software e, devido à sua natureza genérica, funciona como um *template* que pode ser usada em diversas situações.

Neste relatório, para cada um dos *patterns*, é explicado o seu objetivo, motivação e aplicação. Também é mostrada a sua estrutura através de um diagrama e a especificação dos participantes no mesmo. Por fim, são explicadas as consequências bem como a sua implementação.

2. Desenvolvimento

2.1. Inversion of Control

O padrão de Inversão de Controlo (IoC) é baseado na remoção de dependências do código. Assim sendo, esta é utilizada frequentemente em estruturas.

A Inversão de Controlo (IoC) é um padrão de desenho mais genérico ou um princípio de desenho. Outros padrões de design implementam o princípio de IoC, como por exemplo, a Injeção de Dependência, o Método de Modelo, a Estratégia, entre outros, como demonstra a figura 1.

2.1.1. Objetivo

- Desacoplar tarefas das suas implementações
- Focalizar o domínio de um módulo para uma tarefa
- Remover pressupostos relativos a outros sistemas
- Prevenir efeitos secundários

2.1.2. Motivação

O padrão controla o fluxo de execução e chama o código personalizado para cumprir várias tarefas. Isto é invertido do controlo regular do fluxo, onde o código personalizado chama as bibliotecas. É frequentemente referido como o "Princípio de Hollywood".

Assim, quando se utiliza uma estrutura MVC ou outra estrutura web, como por exemplo, Ruby on Rails ou Spring, está-se a utilizar uma implementação de Inversão de Controlo.

2.1.3. Estrutura

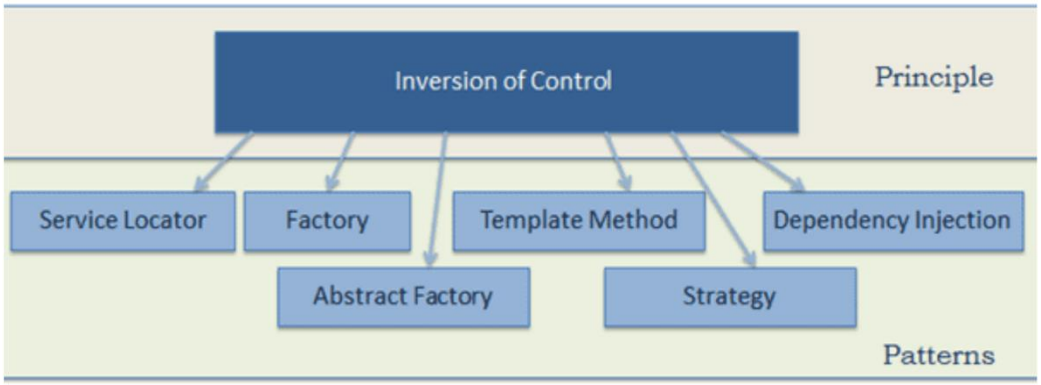

Figura 1- Estrutura do Inversion of Control

2.1.4. Participantes

O cliente em questão e o serviço e a interface utilizados pelo mesmo.

2.1.5. Consequências

O cliente que cria a classe tem controlo sobre qual a implementação das instâncias a utilizar.

2.1.6. Implementação

De seguida é apresentado um exemplo de código padrão, exemplificando um editor de texto, com uma componente de verificação ortográfica.

```
1 public class EditorTexto {
2     private VerificadorOrtografico verificador;
3     public EditorTexto () {
4         this.verificador = new VerificadorOrtografico();
5     }
6 }
```

Figura 2 - Código

Este código padrão cria uma dependência entre o editor de texto e o verificador ortográfico, pois estamos a instanciar o VerificadorOrtografico dentro da classe do EditorTexto. Em alternativa, num cenário de Inversão de Controlo, o código seria:

```
1 public class EditorTexto {
2     private VerificadorOrtografico verificador;
3     public EditorTexto(VerificadorOrtografico verificador) {
4                 this.verificador = verificador;
5     }
6 }
```

Figura 3 - Código (Parte II)

Neste caso, estamos a criar uma abstração tendo a classe de dependência VerificadorOrtografico na assinatura do construtor do EditorTexto, não inicializando a dependência na classe. Com isto, temos a possibilidade de chamar a dependência onde pretendemos e passar como argumento, da seguinte forma:

```
1 VerificadorOrtografico vo = new VerificadorOrtografico(); // dependência
2 EditorTexto et = new EditorTexto(vo);
```

Figura 4 - Código (Parte III)

Agora o cliente que cria a classe EditorTexto tem controlo sobre qual a implementação do VerificadorOrtografico a utilizar.

2.2. Dependency Injection

Dependency Injection é uma técnica de programação usada frequentemente no padrão de design estrutural de inversão de controlo, que torna uma classe independente das suas dependências, dissociando a utilização de um objeto da sua criação.

2.2.1. Objetivo

Remover a dependência de instanciação entre um serviço e o cliente que usufrui dele, abstraindo-o através de uma interface.

2.2.2. Motivação

Permitir a eventual mudança de dependências sem alterar a classe que as usa, o que é vantajoso em termos de reutilização de código, visto que diminui a frequência com que é necessário alterar uma classe.

2.2.3. Aplicação

- Guardar as configurações de um sistema em ficheiros de configuração, permitindo reconfigurar o sistema sem o recompilar;
- Desenvolvimento concorrente de classes que se usam uma à outra, precisando apenas de conhecer a interface pela qual comunicam;
- *Mocking* – testar objetos reais através de interfaces que simulam o seu comportamento.

2.2.4. Estrutura

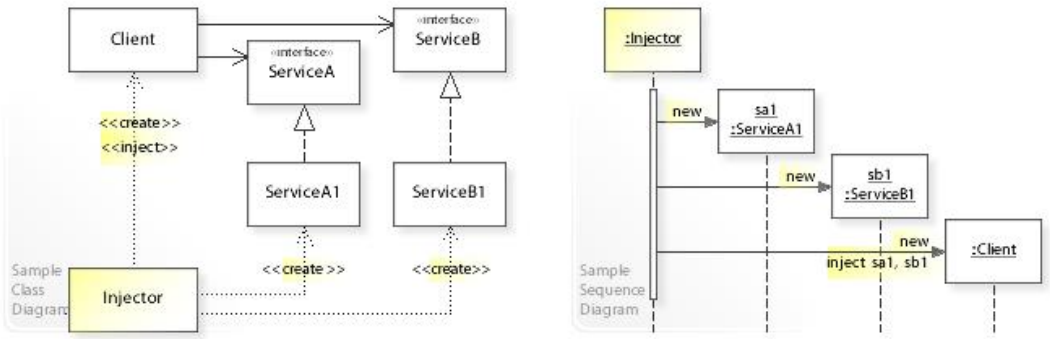

Figura 5 - Estrutura de Dependency Injection

2.2.5. Participantes

O serviço a ser usado, o cliente que o usa, a interface (instância) que é usada pelo cliente e implementada pelo serviço e, por fim, o injetor que cria uma instância do serviço e a injeta no cliente.

2.2.6. Consequências

Dá-se uma inversão do controlo, dado que o injetor passa a ditar o serviço que o cliente utiliza, passando-lhe uma instância do mesmo, em vez de ser o cliente a especificar de que serviço pretende usufruir.

2.2.7. Implementação

Criação de uma classe injetora responsável por abstrair dependências (serviços), criando interfaces que depois insere nos clientes que usufruem delas.

2.3. Facade

O Facade é um padrão de design estrutural, que demonstram como as relações dos objetos podem ser organizadas, em diversos cenários, de modo a obter facilidade de uso.

2.3.1. Objetivo

Abstrair ao cliente os vários componentes de um sistema através duma interface minimalista que expõe apenas os métodos necessários para as manipular, de modo a facilitar a utilização dos vários subsistemas.

2.3.2. Motivação

No caso da existência de um sistema complexo, o cliente não necessita de entender como este está implementado para tirar partido de alguns serviços específicos que ele dispõe. Assim, através de uma interface minimalista que exponha maneiras de os utilizar, consegue efetivamente tirar partido de funcionalidades específicas sem ter de conhecer os subsistemas.

2.3.3. Aplicação

- Se for necessária uma interface simplificada dum sistema ou subsistema complexo.
- Se for necessário estruturar um sistema ou subsistema por camadas.

2.3.4. Estrutura

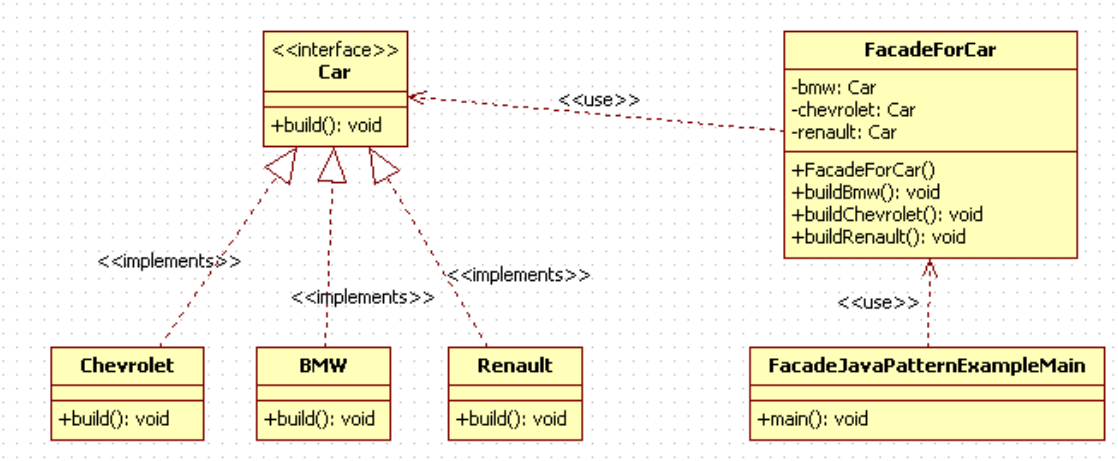

Figura 6 - Estrutura do Facade

2.3.5. Participantes

- Car – especifica uma interface abstrata para um carro
- Chevrolet, BMW, Renault – implementam a interface "Car" e definem um carro de uma marca específica

- FacadeForCar – classe "Facade" cujas variáveis de instância privadas são objetos que implementam a interface "Car" e cujos métodos públicos têm como propósito abstrair os comportamentos dos vários subsistemas (diferentes marcas de carro)
- FacadeJavaPatternExampleMain – classe exemplo que instancia uma facade

2.3.6. Consequências

- Se a "Facade" for o único ponto de acesso para o subsistema, limitará as características e a flexibilidade de partes do sistema específicas, uma vez que não se lida com eles diretamente.
- Torna mais simples a utilização dos subsistemas, devido ao encapsulamento da sua implementação.
- Protege o cliente da complexidade do sistema.

2.3.7. Implementação

Criação de uma classe injetora responsável por abstrair dependências (serviços), criando interfaces que depois insere nos clientes que usufruem delas

3. Conclusão

Concluído este segundo exercício prático, interessa mencionar que a elaboração desta pesquisa permitiu ao grupo consolidar a utilidade dos *design patterns*.

Através da execução deste exercício prático, os membros do grupo estudaram o conceito teórico de *pattern design*, aprenderam pormenores sobre *designs* específicos bem como as diferenças entre os vários.

Em suma, o desenvolvimento deste trabalho feito decorreu como planeado, satisfazendo todos os requerimentos pedidos.