UNIVERSIDADE DO MINHO
ESCOLA DE ENGENHARIA

# Arquiteturas Aplicacionais

## Inversão de Controlo & Injeção de Dependências

Ana Marta Santos Ribeiro A82474
Jéssica Andreia Fernandes Lemos A82061
Miguel José Dias Pereira A78912

**MIEI - 4º Ano - Arquiteturas Aplicacionais**

Braga, 19 de Março de 2020

# Conteúdo

# 1 Introdução

Este relatório surge no âmbito do trabalho prático da unidade curricular de Arquiteturas Aplicacionais que se encontra inserida no perfil de Engenharia de Aplicações. Neste foi realizada uma pesquisa sobre os padrões Injeção de Dependências e Inversão de Controle.

Desta forma, será realizada uma abordagem geral a cada um dos padrões e posteriormente será apresentado um protótipo, a título de exemplo, que implemente os mesmos.

# 2 Inversão de Controlo

A Inversão de Controlo é um *pattern* que tem por base delegar a execução ou controlo dos objetos para terceiros, como *containers* ou *frameworks*.

Ao invés de no código desenvolvido fazermos chamadas a bibliotecas, o que faz com que o nosso código esteja dependente das bibliotecas utilizadas, com a Inversão de Controlo as chamadas serão feitas ao nosso código. Este conceito trás algumas vantagens como maior modularidade, facilidade em alterar as implementações, entre outros.

Um dos exemplos de utilização deste *pattern* são as *frameworks*, às quais podemos adicionar o comportamento que desejamos através da adição das nossas classes ou extensão das já existentes.

Para implementar este *pattern* existem várias alternativas, sendo que de seguida iremos abordar uma delas, a Injeção de Dependências.

# 3 Injeção de Dependências

O *pattern* de Injeção de Dependências, é bastante utilizado para implementar o *pattern* Inversão de Controlo, explicado anteriormente. Este permite desenvolver sistemas com pouca dependência entre módulos, o que facilita a manutenção do sistema. Para além disso, torna o código mais legível, bem como simplifica os testes a executar. Para tal, este admite que os objetos dependentes sejam criados fora da classe e fornece-os de diferentes modos. Desta forma, para além do método de injeção é necessário a existência de duas classes, nomeadamente a dependente e a que define o objeto.

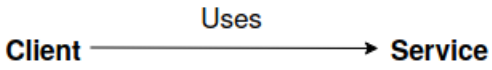


**Figura 1.** Classes

Foi possível verificar que existem três formas de injetar as dependências, nomeadamente:

- **Constructor Injection -** as dependências do objeto são injetadas diretamente no construtor. Assim, é possível atribuir à variável o objeto do qual depende.
- **Setter Injection -** método que implementa a injeção de dependências através da definição de um *Set* na classe Client.
- **Interface Injection -** a classe Client implementa uma Interface que declara o método que fornece a dependência. Esse método pode ser por exemplo um *Set*.

# 4 Exemplo Prático

Uma dependência ocorre quando temos uma classe A que depende dos métodos de uma classe B. Neste pequeno exemplo elaborado, temos a classe *Cliente* e *Pedido*, em que a primeira precisa da segunda para armazenarmos o pedido do cliente. Tipicamente iríamos implementar da seguinte forma:

```
1 public class Cliente {
2     Crepe crepe;
3
4     public Cliente(){
5         crepe = new Crepe;
6     }
7     //...
8 }
```

Contudo, a injeção de dependências defende que em vez de inicializarmos ou instanciarmos outro objeto podemos utilizar, por exemplo, uma *framework* para obtermos o objeto como parâmetro. Basicamente, o método não tem uma dependência direta numa

implementação em concreto. Em vez de termos um método que cria uma dependência no objeto, o objeto com dependências é passado como parâmetro no método. Desta forma, fazemos a inversão do controlo.

```
public class Cliente {
    Pedido pedido;

    public Cliente(Pedido pedido){
            this.pedido = pedido;
    }

    public void acompanhamento(String descricao){
        pedido.acompanhamento(descricao);
    }
}
```

Porém, temos de ter em conta que nesta creperia existem vários tipos de pedidos para além de crepes, pelo que temos de seguir os princípios da Inversão da Dependência em que utilizamos implementações abstratas em vez de concretas. Recorrendo a uma interface para os pedidos conseguimos resolver estas dependências.

```
public interface Pedido {
    void acompanhamento(String descricao);
}
```

Assim, através da injeção de dependências e inversão de controlo torna-se bastante simples acrescentarmos diferentes tipos de pedidos.

```
public class Crepe implements Pedido {
    @Override
    public void acompanhamento(String descricao){
        System.out.println("Crepe com " + descricao);
    }
}
```

# 5 Conclusão

Esta pesquisa sobre Inversão de Controlo e Injeção de Dependências permitiu-nos ficar a saber mais sobre estes padrões, que pretendem remover o controlo de certas

partes do código para terceiros, permitindo assim ter código mais modularizado e com menores dependências, bem como facilitar a execução de testes.

Através do protótipo criado, também ficamos a saber como seria a utilização prática destes mesmos conceitos.

# Bibliografia

[1] https://www.baeldung.com/inversion-control-and-dependency-injection-in-spring

[2] https://en.wikipedia.org/wiki/Inversion_of_control

[3] https://www.tutorialsteacher.com/ioc/inversion-of-control

[4] https://www.tutorialsteacher.com/ioc/dependency-injection

[5] https://stackify.com/dependency-injection/