

Injeção de Dependências

Profa. Fabíola S. F. Pereira

fabiola.pereira@ufu.br

Injeção de Dependências

- É um padrão de projeto
- Não faz parte dos padrões de projeto do livro GoF. Porém, a solução é frequentemente listada como um padrão que poderia ser incluído em uma possível segunda edição do livro.
- A ideia de Injeção de Dependência é bastante simples e quase que uma aplicação literal do seu nome.

Injeção de Dependências

- Suponha que uma classe A dependa de uma outra classe B

```
class A {  
    B b; // A depende de B  
}
```

Injeção de Dependências

- No entanto, para seguir a ideia do padrão, a classe A não deve instanciar diretamente – isto é, no seu código — objetos do tipo B, como em:

```
class A {  
    B b = new B(); // instanciacao incompativel com injeção de dependência  
}
```

Tal solução **não** usa injeção de dependência

```
class A {  
    B b = new B(); // instânciação incompatível com injeção de dependência  
}
```

Dependência está “hard-coded”

Injeção de Dependências

- Em vez disso, a classe A deve receber essa dependência por meio de um construtor:

```
class A {  
    B b;  
    A(B b) { // injeção de dependência via construtor  
        this.b = b;  
    }  
}
```

Injeção de Dependências

- ou então receber a dependência por meio de um método set:

```
class A {  
    B b;  
    void setB(B b) { // injeção de dependência via setter  
        this.b = b;  
    }  
}
```

Injeção de Dependências

- Logo, agora fica fácil entender o nome do padrão: as dependências de uma classe são injetadas nela, seja
 - por meio de chamadas do seu construtor ou
 - por meio de chamadas de um setter.

Injeção de Dependências

- Na verdade, o mais recomendado é que o código de A use uma interface (isto é, uma abstração) para a classe concreta B. Ou seja, em vez de usar B (uma classe concreta), deve-se usar IB (uma interface):

```
class A {  
    IB b; // dependência para uma interface IB  
    A(IB b) {  
        this.b = b;  
    }  
}
```

ou

```
class A {  
    IB b; // dependência para uma interface IB  
    void setterB(IB b) {  
        this.b = b;  
    }  
}
```

Injeção de Dependências

- Quando usamos Injeção de Dependência devemos fazer uso do princípio de projeto **Prefira Interfaces a Classes Concretas**

Injeção de Dependências: vantagens (1)

- Torna mais fácil mudar a dependência concreta (B) usada por uma classe A.
 - Por exemplo, A pode ser uma classe que precisa enviar mails. Para isso, ela faz uso de uma classe B1. Amanhã, no entanto, podemos decidir que os mails serão enviados por uma classe B2. Para isso, basta que B1 e B2 implementem a interface IB.

Injeção de Dependências: vantagens (2)

- Torna mais fácil o teste da classe A, pois podemos ***mockar*** mais facilmente a dependência para B.
 - Por exemplo, em vez de um serviço de mail real (B1 ou B2), podemos usar um serviço de mail **fictício**, que apenas emule o envio de uma mensagem simples. Para isso, basta que esse serviço fictício implemente a interface IB.

Injeção de Dependência: Resumo

- Injetar dependências: “parametrizar” uma classe de forma que seja possível alterar suas dependências.
- Objetivo: permitir mudar as dependências de uma classe
- Por que isso é importante?

Frameworks de Injeção de Dependência

- Assumem a responsabilidade de:
 - Criar uma classe A
 - Criar e injetar as dependências de A
- Exemplos:
 - Java: Spring, Guice, Dagger2
 - TypeScript: InversifyJS
 - Go: wire


Exemplo: Classe na qual a dependência será injetada

```
class A {  
    IB b;  
  
    @Inject // anotação disponibilizada pelo framework  
    A(IB b) {  
        this.b = b;  
    }  
}
```

Exemplo: Classe que cria a classe da dependência

```
class Cliente {  
    void foo() {  
        A a = DIF.getInstance(A.class);  
        ...  
    }  
}
```

Nome hipotético de um
framework de injeção de
dependência



Pergunta: como o framework vai inferir a dependência que precisa injetar na classe A?

Pergunta: como o framework vai inferir a dependência que precisa injetar na classe A?

- Esta informação é declarada em um arquivo externo
- Por exemplo, um arquivo XML ou JSON
- Neste arquivo, define-se o seguinte:
 - Toda vez que for solicitada uma dependência de um determinado tipo, digamos IB
 - Deve ser criado e injetado no lugar um objeto de uma classe B (tipo concreto)

Frameworks de Injeção de Dependência: passo-a-passo

- ***getInstance*** é um método que instancia classes que usam Injeção de Dependência.

```
class Cliente {  
    void foo() {  
        A a = DIF.getInstance(A.class);  
        ...  
    }  
}
```

Nome hipotético de
um framework de
injeção de
dependência

Frameworks de Injeção de Dependência: passo-a-passo

1. A classe A faz uso de injeção de dependência, pois seu construtor foi anotado com `@Inject`.
2. Antes de instanciar um objeto da classe A, o framework deve criar os objetos (dependências) usados pelo construtor dessa classe. No caso, esses objetos são de classes que implementam a interface IB.

Frameworks de Injeção de Dependência: passo-a-passo

3. Mas qual classe que implementa IB deve ser instanciada? Para isso, o framework consulta o arquivo de configuração e descobre que IB está mapeada para B1, conforme explicamos anteriormente.

4. Então o framework instancia um objeto do tipo B1 e um objeto do tipo A . Ao instanciar esse último objeto ele também passa o objeto do tipo B1 como parâmetro de seu construtor.

Pergunta

Injeção de Dependência, muitas vezes, é comparada com padrão de projeto **Fábrica**. Qual a **desvantagem** de injetar dependências por meio de fábricas?

```
class A {  
    IB b;  
    A(IB b) {  
        this.b = b; // injeção de dependência  
    }  
}
```

```
class A {  
    IB b;  
    A() {  
        this.b = IB_Factory.getInstance(); // fábrica  
    }  
}
```

Créditos

- CC-BY: Slides adaptados de Marco Tulio Valente, ESM

Fim