## CETEJ35 - Java Web - JAVA\_XXX (2024\_01)

Meus cursos / CETEJ35 - Web (2024 01) / Semana 06: 07/10 a 13/10 / Integração

## Integração

Feito: Ver ( Feito: Gastar pelo menos 20 minutos na atividade )

A fazer: Passar pela atividade até o fim

Fecha: segunda-feira, 2 dez. 2024, 00:00

Normalmente, um aplicativo real vai precisar mais do que salvar dados em memória - ele vai precisar de persistência em banco! É aí que entra o conteúdo dessa aula.

## INJEÇÃO/INVERSÃO DE DEPENDÊNCIA

Dependência costuma ser algo ruim. Por exemplo, depender dos seus pais para ter dinheiro, do seu professor para passar de ano, da Netflix para distrair seus finais de semana ...

Por isso, tentamos evitar dependência. No código não é diferente. No código, dependência gera acoplamento. Normalmente, muito acoplamento é ruim porque dificulta o escalonamento do código. Um código fortemente acoplado tende a ser difícil de manter e estender.

A inversão de dependência costuma ser uma estratégia para reduzir a dependência no código. Em Java, uma dependência é criada toda vez que você instancia uma classe. Por exemplo, inserir o código new CidadeService() na classe CidadeController cria uma dependência entre as duas classes. Agora, elas estão acopladas e, portanto, mudanças na classe CidadeService podem impactar a classe CidadeController.

Os termos "inversão" e "injeção" de dependência costumam ser usados de maneira intercambiável. Tecnicamente, "inversão" costuma ser usado quando se fala de design da aplicação, enquanto "injeção" costuma ser usado quando se fala de frameworks.

Nesse ponto talvez você esteja pensando: "Como vou usar uma classe sem instanciar?" Os mais astutos talvez tenham considerado usar métodos estáticos para tudo. Na verdade, essa não é a saída.

Existem vários padrões que favorecem a inversão de dependência. Uma saída comum é o uso de fábricas.

AbstractFactory e FactoryMethod são dois padrões de projeto frequentemente referenciados como fábricas. Eles funcionam como fábricas, de fato. Contudo, o padrão que vou mostrar aqui não é nenhum desses dois, mas é frequentemente usado também.

Considere o código abaixo. Esse código usa a instanciação direta. Agora, considere que você precisa usar diferentes versões da CidadeService. Por exemplo, uma CidadeServiceEUA para ser usada quando a cidade está nos Estados Unidos, e CidadeServiceBrasil, para ser usada quando a cidade está no Brasil. Nesse caso, você não consegue fazer uma mudança direta no código sem depender de estruturas condicionais mais complexas.

```
public class CidadeController {
    ...
    var cidadeService = new CidadeService();
        System.out.println(cidadeService.calculaImposto());
    ...
}
```

Agora, considere o uso de uma fábrica. Para ter uma fábrica, primeiro precisamos de uma interface que defina operações comuns, como o código abaixo. A interface **ICidadeService** define operações que todas as classes de serviço de cidade devem ter.

```
public interface ICidadeService {
...
    public String getNomeCidadeInternacionalizado();
    public double calculaImposto();
...
}
```

Em seguida, todas suas classes **CidadeService** devem implementar a interface **ICidadeService**, conforme exemplificado a seguir.

```
public class CidadeServiceBrasil implements ICidadeService {
    ...
    public String getNomeCidadeInternacionalizado() {
        // Implementa método para cidades Brasileiras
    }

public double calculaImposto() {
        // Implementa método para cidades Brasileiras
    }
    ...
}
```

Por fim, você pode usar um Enum Java para "fabricar" as instâncias.



```
enum CidadeServiceFactory {
    BRASIL {
        public ICidadeService getInstance() {
            return new CidadeServiceBrasil();
        }
},
    EUA{
        public ICidadeService getInstance() {
            return new CidadeService getInstance() {
                return new CidadeServiceEUA();
        }
};

public abstract ICidadeService getInstance();
}
```

Agora, é só adicionar a fábrica no código original.

```
public class CidadeController {
...
    var cidadeService = switch (cidadeServiceFactory) {
        case BRASIL -> CidadeServiceFactory.BRASIL.getInstance();
        case EUA -> CidadeServiceFactory.EUA.getInstance();
}

System.out.println(cidadeService.calculaImposto());
...
}
```

Espera aí, eu sei exatamente o que você está pensando: "Perae, não mudou nada! A única coisa é que agora a instanciação é feita em outro lugar!" Pois é, mas é justamente isso que torna as coisas mais simples. A implementação agora está desacoplada. Na prática, você pode mudar até o nome da classe CidadeServiceBrasil para CidadeServiceAvancaBrasil e a classe CidadeController não vai perceber a diferença!

É claro, se você mudar o nome da classe, a fábrica precisa ser atualizada.

Mas por que estamos falando disso tudo? Porque a injeção de dependência é um princípio fundamental do Spring Boot. Você já se perguntou quem é instancia a classe CidadeController? Você já percebeu que quando cria um repositor para acesso aos dados do banco de dados, isso é feito usando uma interface? Já notou que você não instancia a classe CidadeRepository?

Tudo isso acontece porque o Spring Boot "injeta" a dependência no código. O próprio Spring atua como uma fábrica, gerando a instância e entregando ela pro código. Isso permite que o código seja muito menos acoplado e facilita a manutenabilidade da aplicação.

Retroceder

Avançar

■ Verificação de aprendizado - Validação de Dados

Seguir para...

API Reativa -

Você acessou como RAFAEL ROCHA DA SILVA PROENCA (Sair) CETEJ35 - Web (2024\_01)

Tema

Adaptable

Boost

Clássico

Campus

Apucarana

Campo Mourão

Cornélio Procópio

Curitiba

Dois Vizinhos

Francisco Beltrão

Guarapuava

Londrina

Medianeira

Pato Branco Ponta Grossa

Reitoria

Santa Helena

Toledo

**UTFPR** 

Ajuda

Chat UTFPR

Calendário Acadêmico

Biblioteca

e-Mail

Nuvem (OwnCloud )

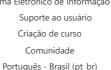
Produção Acadêmica

Secretaria Acadêmica

Sistemas Corporativos

Sistema Eletrônico de Informação - SEI

Deutsch (de)



English (en) Português - Brasil (pt\_br)

Resumo de retenção de dados

Baixar o aplicativo móvel.

Dê um feedback sobre este software

Universidade Tecnológica Federal do Paraná - UTFPR Suporte ao usuário

