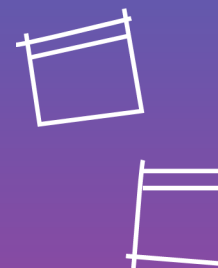




 **VEM SER**
DBC



Spring Web

Aula 3 – Injeção de Dependência

Sumário

- Injeção de Dependência
- Métodos Produtores
- Qualificando um Bean
- Escopos
- Maven e Gradle
- Properties e Profiles

Injeção de Dependência

- Injeção de dependências (ou Dependency Injection – DI) é um tipo de **inversão de controle** (ou Inversion of Control – IoC) que dá nome ao processo de prover instâncias de classes que um objeto precisa para funcionar.

```
public class ClienteServico {

    private ClienteRepositorio repositorio;

    public void salvar(Cliente cliente) {
        this.repositorio.salvar(cliente);
    }

    ...
}
```

- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Injeção de Dependência

```

1  public class ClienteServico {
2
3      @Autowired
4      private ClienteRepositorio repositorio;
5
6      ...
7  }
  
```

- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Injeção de Dependência

```

1  public class ClienteServico {
2
3      private ClienteRepositorio repositorio;
4
5      @Autowired
6      public ClienteServico(ClienteRepositorio repositorio) {
7          this.repositorio = repositorio;
8      }
9
10     ...
11 }
  
```

- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Injeção de Dependência

```

1  public class ClienteServico {
2
3      private ClienteRepositorio repositorio;
4
5      ...
6
7      // Caso você preferisse, esse método poderia se
8      // chamar também "configurarRepositorio", mas
9      // o mais comum é criar um método setter mesmo.
10     @Autowired
11     public void setRepositorio(ClienteRepositorio repositorio) {
12         this.repositorio = repositorio;
13     }
14 }

```

- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Onde podemos utilizar classes injetáveis?

- Nas classes que são Bean Spring, ou seja, anotadas com **@Component** ou:
 - **@Service**
 - **@Repository**
 - **@Controller**
- O Spring chama essas quatro anotações de estereótipos, sendo que as três últimas são como anotações “filhas” da anotação **@Component**.
- A anotação **@Component** é a mais geral e as outras, que são **@Repository**, **@Service** e **@Controller**, são para usos mais específicos em componentes de persistência, serviço e controlador, respectivamente.
- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Métodos Produtores

- Utilizado quando precisamos de beans cujas classes nós não podemos anotar.

```

1  import org.springframework.context.annotation.Bean;
2  import org.springframework.context.annotation.Configuration;
3
4  @Configuration
5  public class AplicacaoConfig {
6
7      @Bean
8      public ArquivoNuvem arquivoNuvem() {
9          ArquivoNuvem nuvem = new ArquivoNuvem();
10         return nuvem;
11     }
12 }

```

```

1  public class ClienteServico {
2
3      @Autowired
4      private ArquivoNuvem nuvem;
5
6      @Autowired
7      private ClienteRepositorio repositorio;
8  }

```

- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Qualificando um Bean

- Quando temos duas implementações da mesma interface

```

1 public interface Notificador {
2
3     void notificar(Mensagem mensagem);
4
5 }
6
7 @Component
8 @Qualifier("importante")
9 public class Email implements Notificador {
10
11     @Override
12     public void notificar(Mensagem mensagem) {
13         ...
14     }
15 }
16
17 @Component
18 @Qualifier("urgente")
19 public class Sms implements Notificador {
20
21     @Override
22     public void notificar(Mensagem mensagem) {
23         ...
24     }
25 }

```

```

1 import org.springframework.beans.factory.annotation.Autowired;
2 import org.springframework.beans.factory.annotation.Qualifier;
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class ClienteServico {
7
8     @Qualifier("importante")
9     @Autowired
10     private Notificador importante;
11
12     @Qualifier("urgente")
13     @Autowired
14     private Notificador urgente;
15 }

```

- <https://blog.algaworks.com/injecao-de-dependencias-com-spring>

Escopos de um Bean Spring

- **Singleton:** Ele retorna uma única instância de bean por contêiner IoC do Spring. Esta instância única é armazenada em um cache de tais beans singleton, e todos os pedidos subsequentes e referências para esse bean nomeado retornam o objeto em cache. Se nenhum escopo de bean for especificado no arquivo de configuração do bean, default para singleton.
- **Prototype:** retorna uma nova instância de bean toda vez que solicitado. Não armazena qualquer versão de cache como singleton.
- **Request:** retorna uma instância de bean único por solicitação HTTP.
- **Session:** retorna uma única instância de bean por sessão HTTP (sessão de nível de usuário).

- <https://www.ti-enxame.com/pt/java/scopes-spring-bean/1041315836>

@Scope

```

1  @Scope("prototype")
2  @Service
3  public class ClienteServico {
4
5      ...
6
7  }
```

- <https://www.ti-enxame.com/pt/java/scopes-spring-bean/1041315836>



Exercício #1

- Utilizando injeção de dependência:
 - Anotar todas as classes de Repository para **@Repository**
 - Remover os new das classes que usam o repository e substituir por **Injeção de dependência** (@Autowired o antigo, via construtor é o atual);
 - Anotar todas as classes de Service para **@Service**
 - Remover os **new** das classes que usam o service e substituir por **Injeção de dependência**
- Testar a app para ver se está tudo certo.

Maven e Gradle

- Basicamente servem para construir o seu projeto (build).
- Fazem tarefas como:
 - **Validação do código**
 - **Métricas**
 - **Testes unitários**
 - **Testes de integração**
 - **Compilação**
 - **Empacotamento**

Maven – Adicionando Dependências

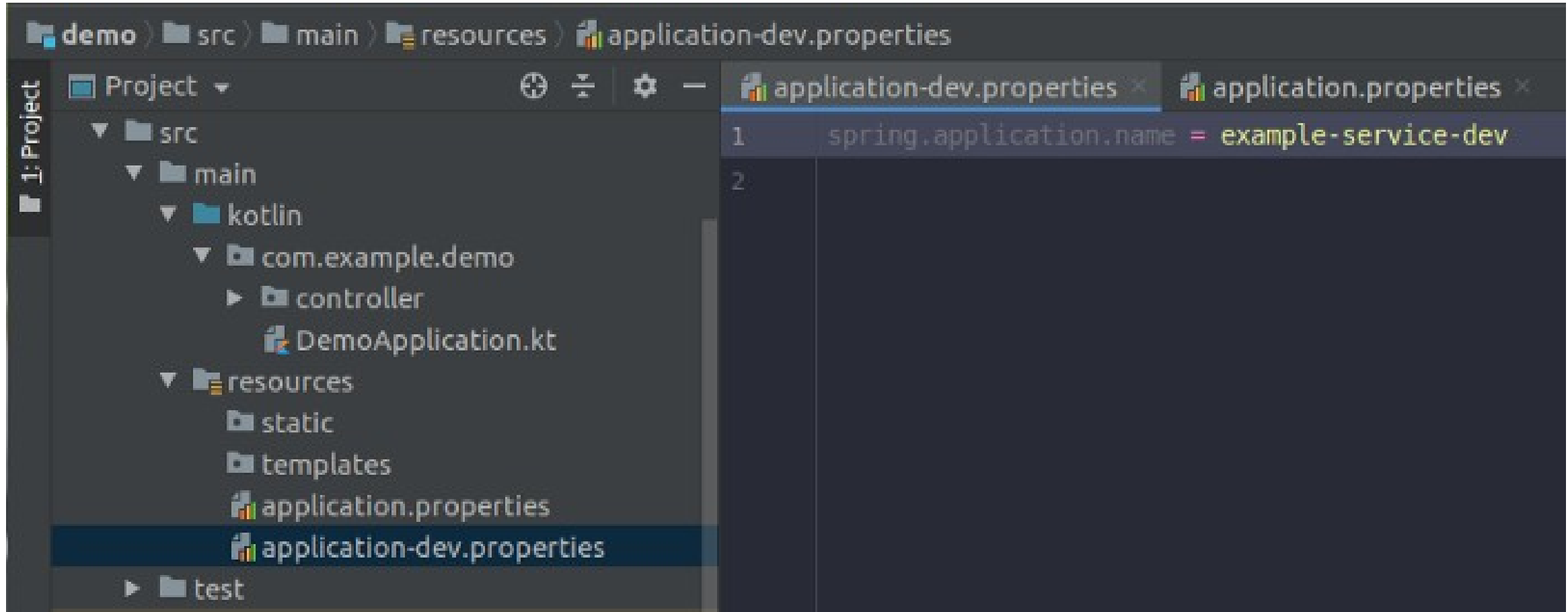
- <https://mvnrepository.com/artifact/org.apache.commons/commons-lang3>

```
<dependency>
  <groupId>org.apache.commons</groupId>
  <artifactId>commons-lang3</artifactId>
  <version>3.12.0</version>
</dependency>
```

Exercício #2

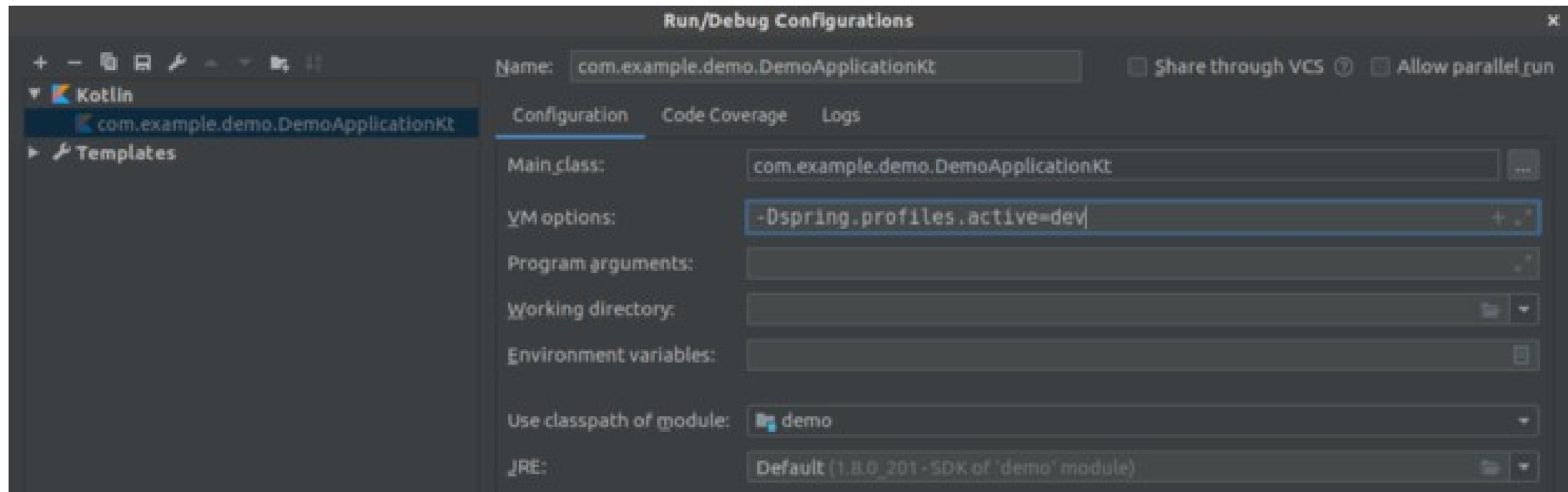
- Adicionar a biblioteca apache **commons-lang3**
- Fazer regras com as funções da biblioteca, para inserção de pessoas:
 - Criar regra para nome: `StringUtils.isBlank` (nome não pode estar em branco)
 - Criar regra para data de nascimento: `ObjectUtils.isEmpty` (não pode inserir pessoa sem data de nascimento)
 - Criar regra para cpf (também contar se possui 11 caracteres): `StringUtils.isBlank`

Properties e Profiles



Properties e Profiles

•



Properties Spring

- <https://docs.spring.io/spring-boot/docs/current/reference/html/application-properties.html>

Exercício #3

- Crie um novo arquivo de propertie “**application-prd.properties**”:
 - No arquivo -prd, altere a porta do tomcat para **8090**
 - Nos dois arquivos defina uma nova propertie chamada “**ambiente**”
 - Para o application.properties defina: “**local**”
 - Para o application-prd.properties defina: “**producao**”
- Crie uma classe chamada **PropertieReader** e anote com **@Component** e leia a propriedade “**ambiente**”. Crie o respectivo get dessa propertie (com o **@Value**).
- Disponibilizar um endpoint (método) na classe **PessoaController** “**/ambiente**” e retorne o valor da propriedade (Utilize a **Injeção de dependência** para injetar o PropertieReader)
- Testar sem profile, na sequencia mude o profile e teste com o profile de **PRD** com o
- **VM args (-Dspring.profiles.active=prd)**

Task #1

- Criar um novo **controller** para adicionar endereço à pessoa
- Utilizar a classe **Endereço** disponibilizada no projeto **pessoa-api**
- Utilizar injeção de dependência
- **Métodos:**
 - GET “/endereco” : recupera todos os endereços.
 - GET “/endereco/{idEndereco}”: recupera o endereço específico.
 - GET “/endereco/{idPessoa}/pessoa”: recupera os endereços por pessoa.
 - POST “/endereco/{idPessoa}”: recebe a pessoa, o endereço e cria o endereço com id da pessoa.
 - PUT “/endereco/{idEndereco}”: altera os dados do endereço.
 - DELETE “/endereco/{idEndereco}”: remove o endereço pelo id.

Task #2 Grupo (opcional)

- O projeto deverá conter a implementação das APIs do tema escolhido pela equipe;
- Ter no mínimo 4 serviços (controller, etc) com as operações GET, POST, PUT e DELETE
- Utilizar Injeção de Dependência de forma correta;
- Seguir o padrão de projetos MVC;
- O repositório deve ser no modelo atual (com JDBC acessando o BD da DBC);

Obrigado!

DBC

DIGITAL BUSINESS COMPANY®



 /dbc.company

 /dbccompany

 /dbccompany.com.br

 /company/dbc-company