DBC
DIGITAL BUSINESS COMPANY®

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

```java
public class ClienteServico {

  private ClienteRepositorio repositorio;

  @Autowired
  public ClienteServico(ClienteRepositorio repositorio) {
    this.repositorio = repositorio;
  }

  ...
}
```

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

Injeção de Dependência

```java
public class ClienteServico {

  private ClienteRepositorio repositorio;

  ...

  // Caso você preferisse, esse método poderia se
  // chamar também "configurarRepositorio", mas
  // o mais comum é criar um método setter mesmo.
  @Autowired
  public void setRepositorio(ClienteRepositorio repositorio) {
    this.repositorio = repositorio;
  }
}
```

- 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.

```java
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;

@Configuration
public class AplicacaoConfig {

  @Bean
  public ArquivoNuvem arquivoNuvem() {
    ArquivoNuvem nuvem = new ArquivoNuvem();
    return nuvem;
  }
}
```

```java
public class ClienteServico {

  @Autowired
  private ArquivoNuvem nuvem;

  @Autowired
  private ClienteRepositorio repositorio;
}
```

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

Qualificando um Bean

- Quando temos duas implementações da mesma interface

```java
public interface Notificador {

  void notificar(Mensagem mensagem);

}

@Component
@Qualifier("importante")
public class Email implements Notificador {

  @Override
  public void notificar(Mensagem mensagem) {
    ...
  }
}

@Component
@Qualifier("urgente")
public class Sms implements Notificador {

  @Override
  public void notificar(Mensagem mensagem) {
    ...
  }
}
```

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Service;

@Service
public class ClienteServico {

  @Qualifier("importante")
  @Autowired
  private Notificador importante;

  @Qualifier("urgente")
  @Autowired
  private Notificador urgente;
}
```

- 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

Vamos praticar!

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 **@Autowired**
  - Anotar todas as classes de Service para **@Service**
  - Remover os new das classes que usam o service e substituir por **@Autowired**

- 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

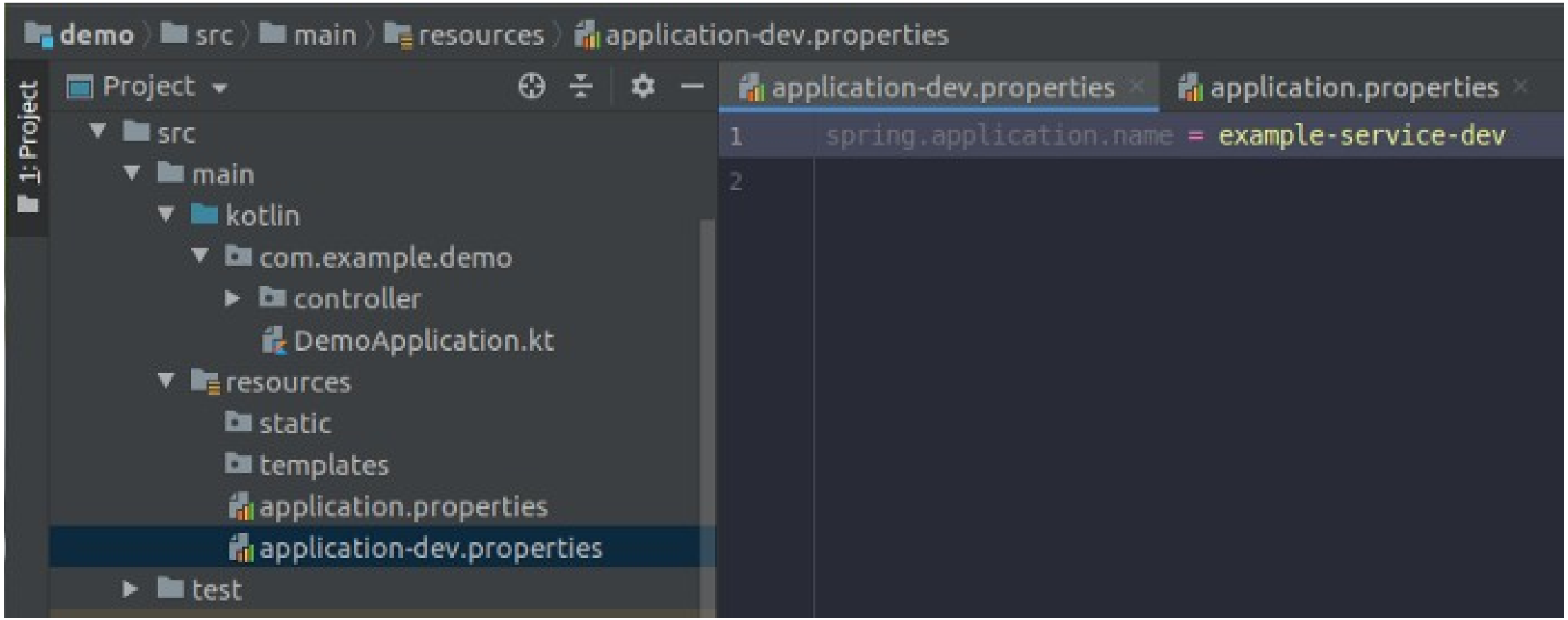
- https://mvnrepository.com/artifact/org.apache.commons/commons-lang3

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

Exercício #2

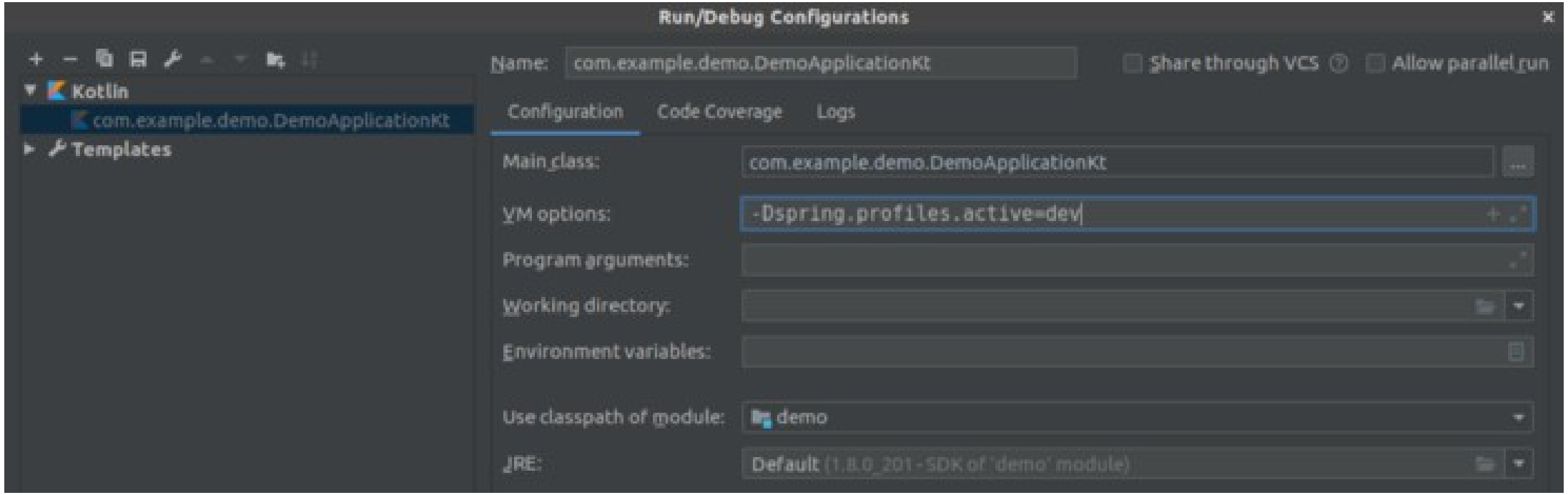
- Adicionar a biblioteca apache commons-lang3
- Fazer regras com as funções da biblioteca, para inserção de pessoas:
  - Criar regra para nome: StringUtils.isBlank (nome não pode estar em branco)
  - Criar regra para data de nascimento: ObjectUtils.isEmpty (não pode inserir pessoa sem data de nascimento)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

Properties e Profiles



Properties e Profiles

- -Dspring.profiles.active=dev



Properties Spring

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

Exercício #3

- Altere a porta do tomcat para **8090**
- Crie um novo arquivo de propertie **“application-prd.properties”:**
  - 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 o **@Autowired** 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)**

Homework

- Criar um novo controller para adicionar endereço à pessoa
- Utilizar a classe Endereco disponibilizada no projeto **pessoaapi**
- 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.

Obrigado!

DBC

DIGITAL BUSINESS COMPANY®

/dbc.company /dbccompany /dbccompany.com.br /company/dbc-company