Instituto Federal da Paraíba
Unidade Acadêmica de Informática
Curso Superior de Tecnologia em Sistemas para Internet
Disciplina: Programação para a Web II
Professor: Frederico Costa Guedes Pereira

Prática01 - Spring IOC

por Frederico C. G. Pereira

Objetivos

- Utilizar o framework Spring e a Inversão de Controle
- Utilizar a injeção de dependência

ATENÇÃO: Não copie e cole os trechos de código, DIGITE-OS VOCÊ MESMO! Isto faz uma gigantesca diferença para o aprendizado e é uma recomendação sempre encontrada em livros de programação. Sempre após digitar, certifique-se de que entendeu o que você fez, do contrário a prática é apenas um exercício de Ctrl C + Ctrl V. Siga corretamente os passos neste roteiro. Obedeça às versões indicadas do que deve ser baixado e não baixe alternativas mais recente porque pode quebrar seu código. Faça estas experiências depois de completar a prática.

1) Crie um projeto Java/Maven chamado **spring-ioc.**
   a) No VSCode, Ctrl+Shift+P
      i) Digite **Create Java Project**
      ii) Selecione Spring Boot
      iii) Selecione Maven Project > 2.6.3 > Java
      iv) Digite o GroupId **br.edu.ifpb.pweb2**
      v) Digite o ArtifactId: **spring-ioc**
      vi) Selecione **Jar** > **17**
      vii) Tecle Enter nos Starters do Spring Boot, não selecionaremos nenhum para este projeto.
      viii) Selecione a pasta (workspacespring, se você fez a prática 02 de JEE) onde o seu projeto será criado.
      ix) A partir da próxima prática não daremos mais detalhes do passo-a-passo para criação de um projeto Spring Boot no VSCode, apenas os parâmetros.
   b) Alternativamente a criar o projeto pelo VSCode, você pode ir ao site https://start.spring.io/ e criar o projeto por lá para depois abri-lo na IDE de sua preferência. Para o passo vii pelo site, não selecione nenhuma dependência.
2) Depois de adicionar o seu projeto, por ser necessário reiniciar o workspace de projetos Java do VSCode (no ItelliJ pode ser necessário fechar e abrir a IDE), para isso, tecle Ctrl+Shift+P, digite **clean java workspace** e selecione a opção única, depois confirme no botão **Restart and Delete** que aparece no canto inferior direito do editor.
3) Exclua a classe **SpringiocApplication.java** do pacote br.edu.ifpb.pweb2.springioc na pasta src/main/java, não precisaremos dela porque este projeto ainda não será uma aplicação Spring Web.
4) Clique com o botão direito no último nome do pacote: springioc e selecione New File
   a) Digite **configuration/DocumentGenerationConf.java**. Um subpacote configuration e uma nova classe Java são criados. Nesta classe configuraremos o Spring para encontrar componentes que criaremos.
   b) Digite o seguinte código na sua classe:

```java
package br.edu.ifpb.pweb2.springioc.configuration;

import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.Configuration;

@Configuration
@ComponentScan("br.edu.ifpb.pweb2.springioc")
public class DocumentGeneratorConf {

}
```

c) A classe não tem nada no seu corpo, apenas duas anotações que informam para o Spring que ela é a classe de configuração e que o Spring encontrará por componentes (ou beans) a partir do pacote "br.edu.ifpb.pweb2.springioc" e subpacotes.

d) Esta aplicação ainda não será uma aplicação web com o Spring Boot, mas uma aplicação para aprendermos o princípio da Inversão de Controle (IoC) através da Injeção de Dependência (DI).

Vamos agora criar uma classe Geradora de documentos que depende de tipos específicos de geradores de parágrafos. Por exemplo, se quisermos gerar arquivos PDF, usaremos um gerador de parágrafos PDF, se quisermos gerar LibreOffice usaremos um gerador de parágrafos LibreOffice e assim por diante. Nossa classe de gerador depende de geradores específicos e queremos trocá-los facilmente na classe geradora, reduzindo, assim, o acoplamento entre a aplicação geradora de documentos e o gerador de um tipo específico.

5) Crie inicialmente a interface **GeradorDocumentoIntf**, num novo pacote **gerador**, que defina o que deve ter um gerador de documento (lembre-se, clique com o botão direito no pacote springioc e selecione New File):

```java
package br.edu.ifpb.pweb2.springioc.gerador;

public interface GeradorDocumentoIntf {

    public void addTexto(String texto);

}
```

6) Agora implemente a classe **GeradorDocumento** no pacote **gerador**. Eis o código da classe:

```java
package br.edu.ifpb.pweb2.springioc.gerador;

import org.springframework.stereotype.Component;

@Component
public class GeradorDocumento implements GeradorDocumentoIntf {

    @Override
    public void addTexto(String texto) {
        // TODO Auto-generated method stub

    }
}
```

Por enquanto não conseguiremos terminar de implementar o **GeradorDocumento** pois não temos nenhum gerador de parágrafos específico implementado.

7) Implemente a interface GeradorParagrafoIntf num novo pacote paragrafo:

```
package br.edu.ifpb.pweb2.springioc.paragrafo;

public interface GeradorParagrafoIntf {

    public void addParagrafo(String texto);
}
```

8) Agora programemos duas classes que implementam a interface GeradorParagrafoIntf, no mesmo pacote paragrafo:

```
package br.edu.ifpb.pweb2.springioc.paragrafo;

import org.springframework.stereotype.Component;

@Component
public class GeradorParagrafoPDF implements GeradorParagrafoIntf {

    @Override
    public void addParagrafo(String texto) {
        System.out.println("{PDF}" + texto + "{/PDF}");
    }

}
```

```
package br.edu.ifpb.pweb2.springioc.paragrafo;

import org.springframework.stereotype.Component;

@Component
public class GeradorParagrafoLibreOffice implements GeradorParagrafoIntf {

    @Override
    public void addParagrafo(String texto) {
        System.out.println("{LibreOffice}" + texto + "{/LibreOffice}");
    }

}
```

9) Agora podemos modificar a classe **GeradorDocumento** para usar a injeção de dependência para uma das classes acima:

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

import br.edu.ifpb.pweb2.springioc.paragrafo.GeradorParagrafoIntf;

@Component
public class GeradorDocumento implements GeradorDocumentoIntf {
    @Autowired
    GeradorParagrafoIntf genParagrafo;

    @Override
    public void addTexto(String texto) {
        genParagrafo.addParagrafo(texto);

    }
}
```

A anotação @Autowired é a sintaxe do Spring para usar a injeção de dependência. A classe GeradorDocumento depende de um GeradorParagrafo. O programador não precisa instanciar esta classe dependente, pois o Spring providenciará uma instancia devidamente inicializada e a "injetará" (atribuirá) na variável **genParagrafo**.

10) Pronto, agora podemos construir uma classe de aplicação que instancie um Gerador de Documentos e gere um texto em determinado formato. Esta classe será a GeradorTextoApp e ficará no pacote app:

```java
package br.edu.ifpb.pweb2.springioc.app;

import org.springframework.context.annotation.AnnotationConfigApplicationContext;

import br.edu.ifpb.pweb2.springioc.configuration.DocumentGeneratorConf;
import br.edu.ifpb.pweb2.springioc.gerador.GeradorDocumentoIntf;

public class GeradorTextosApp {

    Run | Debug
    public static void main(String[] args) {
        AnnotationConfigApplicationContext context = new AnnotationConfigApplicationContext(
                DocumentGeneratorConf.class);
        GeradorDocumentoIntf gerador = context.getBean(GeradorDocumentoIntf.class);

        gerador.addTexto("As fronteiras da minha linguagem são as fronteiras do meu mundo.");
        gerador.addTexto("Sobre o que não se pode falar, deve-se calar.");
        gerador.addTexto("-- Ludwig Wittgenstein");

        context.close();
    }

}
```

11) Agora podemos executar a classe de aplicação, que inicializa o Spring (AnnotationConfigApplicationContext), gera uma instância de um *bean spring* (classe anotada com **@Component**) **GeradorDocumentoIntf**. Na prática, apenas uma classe implementa esta interface, então será uma instância de **GeradorDocumento**.

a) Clique em Run, sobre o método main() e veja o resultado.

b) Se você se deparou com um *stack trace* de erros de Java, era isso mesmo que esperávamos:

```
Exception in thread "main" org.springframework.beans.factory.UnsatisfiedDependencyException: Error creating bean with name
 'geradorDocumento': Unsatisfied dependency expressed through field 'genParagrafo'; nested exception is org.springframewor
k.beans.factory.NoUniqueBeanDefinitionException: No qualifying bean of type 'br.edu.ifpb.pweb2.springioc.paragrafo.Gerador
ParagrafoIntf' available: expected single matching bean but found 2: geradorParagrafoLibreOffice,geradorParagrafoPDF
        at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.resolve
FieldValue(AutowiredAnnotationBeanPostProcessor.java:659)
        at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor$AutowiredFieldElement.inject(
AutowiredAnnotationBeanPostProcessor.java:639)
        at org.springframework.beans.factory.annotation.InjectionMetadata.inject(InjectionMetadata.java:119)
        at org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.postProcessProperties(Autowir
edAnnotationBeanPostProcessor.java:399)
        at org.springframework.beans.factory.support.AbstractAutowireCapableBeanFactory.populateBean(AbstractAutowireCapab
```

c) Este erro, NonUniqueBeanDefinitionException, ocorreu porque nós temos duas classes que implementam a interface GeradorParagrafoIntf: GeradorParagrafoPDF e GeradorParagrafoLibreOffice. O Spring não sabe quais dos dois injetar na classe GeradorDocumento (no @Autowired).

d) Para solucionar este problema, precisamos passar mais informações ao Spring, através de anotação, de qual classe dependente queremos a injeção. Fazemos isto com a anotação **@Qualifier**, na classe **GeradorParagrafo**, logo abaixo do **@Autowired**:

```
@Autowired
@Qualifier("geradorParagrafoPDF")
GeradorParagrafoIntf genParagrafo;
```

e) O parâmetro desta anotação é o nome da classe concreta cujo objeto será injetado, iniciando com minúsculo. Se tivéssemos dado um nome ao nosso componente com @Component("genPDF") deveríamos usar este nome "genPDF" em vez do nome da classe. O programador é livre para escolher. Não esqueça de importar a anotação @Qualifier teclando Shift+Alt+O no VSCode.

f) Na classe **GeradorTextoApp**, clique no **Run** sobre o método main. Desta vez, você deverá ver o texto impresso na console com o seguinte conteúdo:

```
{PDF}As fronteiras da minha linguagem são as fronteiras do meu mundo.{/PDF}
{PDF}Sobre o que não se pode falar, deve-se calar.{/PDF}
{PDF}-- Ludwig Wittgenstein{/PDF}
```

g) Se mudarmos o **qualificador** para "geradorParagrafoLibreOffice", teremos (experimente!):

```
{LibreOffice}As fronteiras da minha linguagem são as fronteiras do meu mundo.{/LibreOffice}
{LibreOffice}Sobre o que não se pode falar, deve-se calar.{/LibreOffice}
{LibreOffice}-- Ludwig Wittgenstein{/LibreOffice}
```

12) Implemente um gerador de parágrafo que gere texto em HTML e faça a aplicação exibir parágrafos com a *tag* <p> do HTML para cada parágrafo adicionado.

13) Pelos slides da matéria, é possível adicionar um gerador de parágrafos dentro da classe de configuração (DocumentGeneratorConf.java). Experimente implementar um gerador de parágrafo de texto puro (**GeradorParagrafoTexto**) desta forma.

14) Crie uma classe no pacote **gerador** chamada **GeradorRelatorio.java**. Esta classe deve ser instanciada de forma programática dentro da classe de configuração (assim como foi feito no exercício anterior). Crie uma classe de aplicação, **GeradorRelatorioApp**, que instancie, através do Spring, esta classe geradora e a execute. Consulte os slides do assunto. Esta classe deve usar um GeradorParagrafoTexto do exercício anterior como gerador de parágrafos.