Módulo 3

No módulo anterior experimentamos injeção em casos de testes graças ao componente ***demoiselle-junit***. Praticamos injeções de *Logger* e *ResourceBundle* providas pelo Framework, pois tais elementos não suportam nativamente o CDI. Neste módulo conheceremos mais funcionalidades do Demoiselle, tais como tratamento de exceções e configuração da nossa aplicação.

Antes de prosseguir, execute os testes automatizados e garanta que está tudo verde, como quando terminamos o módulo anterior.

Tratamento de exceções

Dando continuidade à nossa aplicação, interceptaremos as exceções lançadas pelo método *matricular* da classe *Turma*. Para cada exceção, um alerta no *log*. Poderíamos fazer isto da seguinte forma:

```
public void matricular(String aluno) {
        try {
                . . .
        } catch (RuntimeException e) {
                logger.warn(...);
        }
}
```

Supondo que a classe *Turma* necessitasse deste tratamento em outros métodos, como você faria? Repetiria o código em todos eles? Como esta necessidade é muito comum em aplicações, o Framework oferece um mecanismo de tratamento de exceções.

1. Na classe *Turma*, crie o método ***public void tratar(RuntimeException e)***.
2. Anote o método com ***@ExceptionHandler***.
3. Anote a classe *Turma* com ***@Controller***.
4. No método *tratar, adicione o comando* ***logger.warn(bundle.getString("matricula.erro"))***.
5. No arquivo *messages.properties*, inclua a linha ***matricula.erro=Aluno matriculado ou turma cheia***.

O método *tratar* será invocado pelo Demoiselle sempre que uma *java.lang.RuntimeException* não tratada ocorrer. O nome do método anotado com *@ExceptionHandler* não segue nenhum padrão pré-definido, sugerimos, apenas, que seja intuitivo.

A classe *Turma* vai ficar assim:

```
@Controller
public class Turma {

	...

	@ExceptionHandler
	public void tratar(RuntimeException e){
		logger.warn(bundle.getString("matricula.erro"));
	}

	...
}
```

Rode o teste, vai ficar vermelho. O que aconteceu? O método *tratar* registrou o alerta e abafou a exceção, não atendendo às regras implementadas nos testes. Faça o seguinte experimento:

1. Inclua a linha ***throw e*** no método *tratar logo abaixo da linha de logger*.
2. Observe o comportamento dos testes. O que aconteceu? Por que aconteceu isso?

O método vai ficar assim:

```
@ExceptionHandler
public void tratar(RuntimeException e){
	...
	throw e;
}
```

O tratador registrou o alerta e passou a exceção lançada adiante, atendendo as especificações dos testes que estavam esperando exceção. Agora sim ficou verde!

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-3-video-1

Exceção específica

De acordo com a cartilha de boas práticas não é recomendado lançar exceções nativas na aplicação. Vamos refatorar nosso projeto para lançar a nossa própria exceção:

1. Crie a classe TurmaException na pasta *src/**main**/java* no pacote *br.gov.serpro.inscricao**.exception*** herdando de *RuntimeException*.
2. Modifique os testes que esperavam *RuntimeException* e que agora devem esperar *TurmaException.*

A classe *TurmaException* ficará assim:

```
public class TurmaException extends RuntimeException {
}
```

As anotações nos testes ficarão assim:

```
@Test(expected = TurmaException.class)
public void falhaAoTentarMatricularAlunoDuplicado() {
	...
}

@Test(expected = TurmaException.class)
public void falhaAoTentarMatricularAlunoNaTurmaCheia() {
	...
}
```

Rode os testes, vai falhar! Modifique a implementação da classe *Turma* para lançar *TurmaException* ao invés de *RuntimeException*. Atualize também o parâmetro do método *tratar:*

```
public void matricular(String aluno) {
	if (...) {
		throw new TurmaException();
	}
	...
}

@ExceptionHandler
public void tratar(TurmaException e){
	...
}
```

Agora rode os testes, vai ficar verde!

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-3-video-2

Configuração da aplicação

A validação no método *matricular* da classe *Turma* não está elegante, pois a capacidade da turma está fixa sem a possibilidade de parametrização:

alunosMatriculados.size() == 5

Vamos utilizar mais uma funcionalidade do Framework para incrementar o nosso projeto:

1. Crie o arquivo *inscricao.properties* na pasta *src/main/**resources***.
2. Acrescente no arquivo a linha ***capacidade.turma=5***.
3. Crie a classe *InscricaoConfig* na pasta src/main/**java** no pacote *br.gov.serpro.inscricao**.config***.
4. Anote-a com @Configuration(resource = "inscricao").
5. Acrescente o atributo *private int capacidadeTurma.*
6. Gere o método *public int getCapacidadeTurma()* que retorna o respectivo atributo.
7. Abra a classe *Turma*.
8. Acrescente o atributo *@Inject private InscricaoConfig config*.
9. Modifique a validação do método *matricular* para utilizar a configuração.

A classe *IncricaoConfig* ficará assim:

```
@Configuration(resource = "inscricao")
public class InscricaoConfig {

	private int capacidadeTurma;

	public int getCapacidadeTurma(){
		return capacidadeTurma;
	}
}
```

O parâmetro *resource = "inscricao"* da anotação *@Configuration* indica que o arquivo de configuração será *inscricao.properties*. Por convenção, o Demoiselle deduz que o atributo *capacidadeTurma* será preenchido com o parâmetro *capacidade.turma* do arquivo de configuração. É possível mudar o nome desse parâmetro através da anotação ***@Name.*** Para isto, basta anotar o atributo com **@Name("nome.do.atributo")** e o nome do parâmetro do arquivo de configuração.

As modificações na classe *Turma* serão:

```
@Inject
private InscricaoConfig config;

public void matricular(String aluno) {
	if (... alunosMatriculados.size() == config.getCapacidadeTurma()) {
		...
	}
	...
}
```

Rode os testes, vai ficar verde!

Esta funcionalidade do Demoiselle oferece muita flexibilidade e detalhes na configuração. Além de arquivos de propriedades, você pode utilizar XML ou variáveis de ambiente. Se quiser explorar estes e outros recursos, consulte o guia de referência do Framework.

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-3-video-3

A classe Aluno

Que tal tornar nosso projeto mais elegante? Ao invés de utilizar apenas uma String representando um aluno, vamos criar a classe Aluno!

1. Crie a classe *Aluno* na pasta *src/main/**java*** no pacote *br.gov.serpro.inscricao.entity*.
2. Acrescente o atributo *private String nome*.
3. Gere os métodos *get* e *set* para o atributo *nome*.
4. Crie um construtor vazio. (A especificação JPA exige que toda entidade tenha uma construtor vazio)
5. Crie um construtor com o nome do aluno atribuindo ao respectivo atributo.
6. Crie o método *equals* conforme a classe abaixo.

A entidade Aluno ficará assim:

```
public class Aluno {

	private String nome;

	public Aluno() {
	}

	public Aluno(String nome) {
		this.nome = nome;
	}

	@Override
	public boolean equals(Object outro) {
		return ((Aluno)outro).nome.equals(this.nome);
	}

	public String getNome() {
		return nome;
	}

	public void setNome(String nome) {
		this.nome = nome;
	}
}
```

Modifique os testes, substituindo a *String* pela classe *Aluno*:

```java
@Test
public void matricularAlunoComSucesso(){
	Aluno aluno = new Aluno("Santos Dumont");

	turma.matricular(aluno);
	Assert.assertTrue(turma.estaMatriculado(aluno));
}

@Test(expected = TurmaException.class)
public void falhaAoTentarMatricularAlunoDuplicado() {
	turma.matricular(new Aluno("Orville Wright"));
	turma.matricular(new Aluno("Orville Wright"));
}

@Test(expected = TurmaException.class)
public void falhaAoTentarMatricularAlunoNaTurmaCheia() {
	for (int i = 1; i <= 5; i++) {
		turma.matricular(new Aluno("Aluno " + i));
	}

	turma.matricular(new Aluno("Aluno 6"));
}
```

Modifique também o atributo e os métodos da classe *Turma*:

```java
private List<Aluno> alunosMatriculados = new ArrayList<Aluno>();

...

public void matricular(Aluno aluno) {
	...
	logger.info(bundle.getString("matricula.sucesso", aluno.getNome()));
}

public boolean estaMatriculado(Aluno aluno) {
	...
}
```

Para finalizar, rode os testes. Tem que passar!

Confira o vídeo de demonstração clicando no link abaixo:

http://www.frameworkdemoiselle.gov.br/documentacaodoprojeto/manuais-e-tutoriais/tutorial-da-versao-2-2-3-0/videos/modulo-3-video-4

Retrospectiva

Aprendemos as facilidades que o Demoiselle oferece para tratamento de exceções. Seguindo boas práticas, criamos uma exceção específica do projeto. Parametrizamos a aplicação com arquivo de configuração e ajustamos a classe *Turma* para referenciar estruturas complexas ao invés de *String*.

No próximo módulo, persistiremos as informações em um banco de dados. Utilizaremos a extensão *demoiselle-jpa* para facilitar a nossa vida. Além disso, experimentaremos o controle transacional provido pelo Framework. Até lá!