Programação para Web II

Programação para Web II

Vanessa Cadan Scheffer
Kleber Ricardi Rovai
Hudson Cadan Scheffer
Paulo Vitor de Campos Souza

Dados Internacionais de Catalogação na Publicação (CIP)

Scheffer, Vanessa Cadan
S316p Programação para web II / Vanessa Cadan Scheffer... [et al.]. – Londrina : Editora e Distribuidora Educacional S.A., 2018.
240 p.

ISBN 978-85-522-1167-9

1. Programação web. 2. Back end. 3. Servidor. I. Scheffer, Vanessa Cadan. II. Título.

CDD 616

Thamiris Mantovani CRB-8/9491

2018
Editora e Distribuidora Educacional S.A.
Avenida Paris, 675 – Parque Residencial João Piza
CEP: 86041-100 — Londrina — PR
e-mail: editora.educacional@kroton.com.br
Homepage: http://www.kroton.com.br/

Sumário

Palavras do autor

Olá, aluno! Vamos juntos aprimorar suas práticas de programação para WEB. Esta disciplina é importante para sua formação, pois a WEB domina grande parte dos negócios das empresas e das demandas de softwares para os profissionais de informática. Para atender às necessidades do mercado, você precisa estar preparado para conhecer conceitos de qualidade, organização e tecnologias de *softwares* voltados para a WEB.

No material de Programação para WEB II, você relembrará conceitos fundamentais sobre a programação orientada a objetos, utilizará a linguagem de programação Java e verá como esses recursos estão presentes na qualidade dos sistemas para WEB. Além da programação orientada a objetos, você utilizará e entenderá a importância do padrão de projeto MVC (Modelo, Visão e Controle) para a organização e comunicação entre classes e camadas do sistema. Desse contexto, você sairá apto a entender e organizar seus arquivos em camadas, preconizando a comunicação eficiente e correta entre elas.

Para facilitar a construção de sistemas WEB em camadas utilizando o MVC, você aprenderá como ferramentas contemporâneas podem tornar a produção e a aceitação de um *software* mais tranquila junto ao mercado. Com o padrão *Java Server Faces* (JSF) acoplado aos sistemas WEB, você estará apto a dinamizar cada vez mais a comunicação entre a camada de visão e a camada de controle de seu *software*. Neste curso, você verá práticas de como utilizar esse recurso e como ele poderá facilitar sua vida de desenvolvedor WEB.

Também abordaremos aspectos referentes à camada de modelo, fator fundamental para a abstração do mundo real para o mundo computacional. Para aprimorar as relações entre as classes em Java e o banco de dados de uma aplicação, você aprenderá a utilizar a *Java Persistence API* (JPA), para fazer a modelagem dessas classes, e o *Hibernate*, para facilitar ações envolvidas na persistência de dados.

Para complementar seus estudos sobre a eficiência e modernidade aplicadas a sistemas WEB, serão apresentadas a você formas práticas de utilizar recursos para dinamizar a construção das interfaces do sistema por meio de plug-ins.

O conteúdo desse livro é fundamental para o mercado de desenvolvimento de softwares e indispensável para sua futura profissão. Realize os estudos práticos, leia o material e esclareça dúvidas, pois esse conteúdo, além de importante, é atual e adequado às tendências de um público cada vez mais exigente.

Desejamos a você bons estudos e que tenha um grande aprendizado sobre as tecnologias para o desenvolvimento WEB.

Programação em JSF - *Java Server Faces*

Convite ao estudo

Olá, estudante! Bem-vindo à primeira unidade do livro de Programação para WEB II. Nesta unidade, você estudará conceitos iniciais sobre a programação WEB usando o framework Java Server Faces (JSF). Além de relembrar conceitos fundamentais para o desenvolvimento orientado a objetos, você aprenderá desde os aspectos da instalação e configuração até o desenvolvimento de sistemas usando tecnologias aderentes ao mercado. Conhecer e compreender a utilização do Java Server Faces é imprescindível para um desenvolvedor web e, certamente, um diferencial no mercado de trabalho.

Como funcionário de uma empresa de desenvolvimento de software, você enfrentará diferentes desafios em cada projeto. Recentemente, seu chefe fechou um projeto com uma concessionária de automóveis e você foi designado para compor essa equipe. A concessionária solicitou o desenvolvimento de um software para o setor de revisão mecânica. Atualmente, todo o processo de agendamento, compra de peças, cadastro de clientes, entre outros, é feito manualmente, o que gera várias inconsistências para o setor. Como premissas de projeto, foram definidas a utilização do contexto WEB de desenvolvimento, a aplicação de frameworks comumente utilizados em outras soluções informatizadas e a aplicação de boas técnicas de programação utilizando a linguagem Java.

Agora que você conhece seu desafio, como você poderá organizar o sistema? Quais ferramentas poderão ser utilizadas para o desenvolvimento do sistema? Existem tecnologias específicas para trabalhar o layout?

O estudo do conteúdo desta unidade lhe permitirá dar o primeiro passo no desenvolvimento do projeto que lhe foi designado. Na primeira seção, você aprenderá como instalar e configurar o ambiente para trabalhar com o JSF, além de explorar o padrão de desenvolvimento em camadas MVC. Na segunda seção, veremos aspectos relacionados aos componentes visuais, aprendendo como inserir folhas de estilos no projeto. Por fim, na terceira seção, veremos componentes estáticos e dinâmicos que colaboram com a organização.

Por ser uma unidade bem prática é necessário que você esteja atento às práticas e às atividades propostas. A maneira mais fácil de aprender a programação para WEB é botando a mão na massa. Então, vamos lá?

Seção 1.1

Frameworks Java Server Pages (JSF)

Diálogo aberto

Caro estudante, bem-vindo à primeira seção de estudos sobre o desenvolvimento web. O desenvolvedor de sistemas pode se deparar com *softwares* de diversas naturezas. Uma das grandes habilidades de um profissional de TI é a de se adaptar a diversos contextos, mesmo aqueles que não estão ligados diretamente a sua área de formação. O sucesso no desenvolvimento de qualquer sistema depende do planejamento e faz parte desse processo escolher o padrão de arquitetura de *software* que será usado, pois assim é possível prever como as classes do sistema estarão conectadas. O desenvolvedor deve ter em mente que a divisão de um problema complexo facilita sua resolução, pois os problemas são direcionados e podem ser resolvidos por partes.

A empresa de desenvolvimento de *software* em que você trabalha recebeu, recentemente, um cliente potencial que solicitou um sistema para o setor mecânico de sua concessionária. Esse setor tem diversos problemas operacionais e de controle que podem ser solucionados com a implantação de um sistema WEB, principalmente na divulgação e disseminação de seus serviços.

Você, enquanto desenvolvedor, optará por começar o *software* utilizando algum padrão de arquitetura dos sistemas? Qual é a importância de se configurar um ambiente de desenvolvimento de forma correta no início das atividades? Por que a qualidade do *software* deve ser alvo do desenvolvedor antes mesmo de programar uma linha de código?

Os analistas de sistemas voltados para a área de requisitos já fizeram a coleta das informações e a partir de agora tem início o seu desafio no projeto. Apresente, em um relatório, um esboço inicial das classes que farão parte do sistema. Nesse documento, também deverá estar especificado o padrão de arquitetura de *software* que será utilizado. Justifique sua escolha apresentando a definição do padrão e as vantagens. Nesse contexto, é importante

utilizar uma ferramenta que possua recursos que facilitem a dinâmica da construção do software por parte do desenvolvedor e, ao mesmo tempo, facilite a experiência e utilização do usuário ao operacionalizar a solução WEB. Portanto, você também deverá especificar qual ferramenta usará e os requisitos e passos necessários para a preparação do ambiente.

Para completar a primeira etapa, você verá, nesta seção, a arquitetura de software MVC, além de conhecer o *framework Java Server Pages* (JSF), aprendendo a instalar e configurar o ambiente para utilização de tal recurso.

Não deixe a sua empresa na mão, vamos juntos resolver esse problema?

Não pode faltar

A programação WEB está em alta no desenvolvimento de *softwares*. Cada dia mais empresas estão investindo em contratação de profissionais para trabalharem na construção ou manutenção de sistemas informatizados. Esse fator se deve à imensa quantidade de pessoas que acessam a internet e a utilizam para diversão e negócios.

Para atender a esse mercado, você deve estar atento aos principais elementos presentes dentro da criação de *softwares* WEB, sendo a linguagem de programação Java uma das grandes referências entre os profissionais da área. Para desenvolver sistemas eficientes, você deve se recordar dos principais conceitos relacionados à construção moderna de *softwares*, baseada em interpretações do mundo real. Nesse caso, estamos falando da orientação a objetos e seus principais pilares.

Classe: conceito referente à modelagem de elementos do mundo real para o mundo computacional.

Objeto: instância de uma classe.

Abstração: capacidade de um sistema representar o conceito do mundo real que ele deve atender.

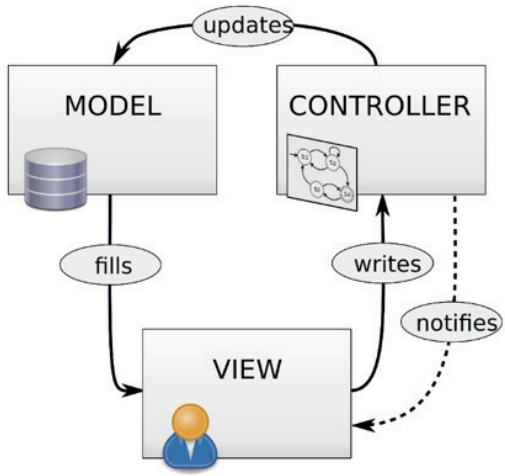
Encapsulamento: proteção e organização da visibilidade dos atributos dentro de uma classe.

Herança: capacidade de transmissão e organização de características em comum para outras classes.

Polimorfismo: ao se utilizar a herança de maneira correta, existe a possibilidade de se criar objetos que possam assumir forma de outros objetos dentro de sua cadeia de herança.

Veja que todos esses conceitos já fazem parte da vida do estudante engajado em soluções tecnológicas por meio de softwares, mas como devemos organizá-los? Nesse caso, para auxiliar nossos projetos, existem os padrões voltados para a arquitetura do *software*, que nos dizem como as classes criadas serão organizadas e como elas deverão se comunicar. Um dos padrões mais utilizado dentro da rotina de desenvolvimento de *softwares* é o MVC, que pode ser visto como um grande "guarda-roupas", em que cada uma das camadas tem uma função bastante específica. A Figura 1.1 apresenta de forma gráfica a organização do MVC dentro do desenvolvimento de *softwares*.

Figura 1.1 | Esquema de funcionamento da arquitetura MVC



Fonte: adaptada de <https://goo.gl/mpkGLV>. Acesso em: 25 jun. 2018.

Na Figura 1.1, podemos verificar que a camada de modelo (*model*) se comunica com a camada de visão, passando informações que normalmente estão armazenadas em um ou vários bancos de dados. Já a camada de visão é responsável por prover uma interface de comunicação entre o usuário e o sistema, pois é nela que existem elementos que possibilitam que o usuário use os serviços do sistema. Ao solicitar um serviço, como realizar o cadastro de um novo usuário, os dados serão enviados para as

classes da camada de controle para tratamento e validação. Essas classes, além de notificarem possíveis respostas aos usuários por intermédio da camada de visão, realizam atividades que geram atualizações na camada de modelo e, por consequência, no banco de dados (GALOTTI, 2016). Uma arquitetura como essa preconiza a organização, a divisão de tarefas e a comunicação entre camadas, todos os conceitos esperados de um sistema orientado a objetos.

A seguir vamos destacar algumas das principais características de cada uma das camadas do MVC. Segundo Luchow et al (2010), são elas:

- **Visão**: requisita atualização da camada de modelo, informa ao usuário respostas efetuadas pela camada de controle, facilita a integração com a camada de modelo, apresentando os principais objetos existentes na solução de forma gráfica, além de permitir às classes de controle selecionar as melhores opções para apresentarem ao usuário. Como principais elementos, estão presentes as tecnologias de criação de páginas (html, xhtml, JSP, JSF), elementos de estilo (CSS) e de eventos (javaScript, jQuery, ajax).
- **Controle**: camada responsável por mapear usuários que utilizam a camada de visão, definir os comportamentos e formas de agir do software, fornecer respostas às requisições da camada de visão, além de mudar o estado da camada de modelo. Podemos destacar como classe presente dentro desse contexto o controle de fluxo do sistema (de qual tela para qual tela, qual funcionalidade ou classe é o alvo da requisição). Nela estão as principais regras de negócio do sistema, fatores esses que determinam o funcionamento de todas as classes envolvidas no contexto do problema.
- **Modelo**: camada responsável por qualquer atualização nas classes pertencentes ao problema. Ela detém e protege o estado de toda a aplicação, expõe as funcionalidades do sistema e avisa a camada de visão sobre quaisquer mudanças ocorridas dentro das atividades do sistema (se, por acaso, um nome de um cliente for alterado no banco de dados, essa nova informação aparecerá na tela do usuário). Temos como principais exemplos de elementos nessa camada as classes de domínio, com seus métodos, atributos e relacionamentos.

Reflita

Regras de negócio estão somente em uma camada do MVC? Entender como as necessidades do cliente podem interferir nas funcionalidades de diversas camadas do sistema pode te auxiliar a compreender esse questionamento.

Como implementar esse tipo de padrão em um projeto Web usando Java? A principal dica é conhecer o contexto com que você vai trabalhar e colocar as classes em seus devidos lugares. Ter organização e lógica na construção de conteúdos para um sistema permitirá que ele tenha dois aspectos extremamente relevantes para o contexto de *software*: capacidade de reuso e refatoração. O reuso diz respeito à reutilização de recursos em outro *software*. Já a refatoração é a forma de melhorar o código e sua organização para otimização de desempenho e organização do projeto.

Mas, e no início do projeto? Como proceder para realizar as atividades iniciais que devem ocorrer?

O primeiro passo é criar um projeto em um ambiente integrado de desenvolvimento (IDE – do inglês, *Integrated Development Environment*) que suporte o Java EE, porém, antes de criar um projeto, é necessário configurar o ambiente de desenvolvimento.

Assimile

Você conhece a diferença entre o Java EE, o Java SE e o Java ME? Todos utilizam a linguagem Java para construir sistemas. Porém, existem sistemas focados em aplicações desktop e, nesse caso, estamos falando do Java SE. Para aplicações embarcadas e móveis, no contexto da "internet das coisas", usa-se o Java ME, e para os fins de construção de sites WEB deve ser usada a plataforma Java EE.

Introdução ao *framework Java Server Pages* (JSF)

O *framework* JSF é um recurso, aplicado na camada de visão, que auxilia a comunicação entre páginas WEB e os recursos existentes nas classes em Java. Segundo Faria, o "*JavaServer Faces*, também conhecido como JSF, é uma tecnologia para desenvolvimento web que utiliza um modelo de interfaces gráficas baseado em eventos" (FARIA, 2015, p. 51). Esse recurso facilita a ligação entre os objetos

java e as ações executadas pelos usuários em suas interações com uma página WEB. Ele facilita a redução das linhas de código para captação de informações presentes em formulários e dinamiza a criação de classes de controle que sejam capazes de se comunicar com uma página que utiliza linguagem de marcação (html ou xhtml). Esse *framework* é baseado no padrão MVC, pois atua diretamente nos elementos da camada de visão. Essa tecnologia proporciona aos desenvolvedores um ganho de produtividade na criação das interfaces, por meio do uso de componentes nos formulários web, tais como botões, campos de digitação, campos de seleção, entre muitos outros. Esses componentes podem ser facilmente acessados pelas classes do sistema por meio de programação, permitindo controle e manipulação dos valores fornecidos pelos usuários.

O controle de um sistema JSF é feito por uma *Servlet* chamada *Faces Servlet*. Nesse contexto, a *Faces Servlet* tem por principal função receber as requisições da camada de visão e transmiti-las à camada de modelo, em especial para as classes de objetos de negócio que realizam a ponte entre a requisição da *Servlet* e da consulta e o banco de dados. Esse conjunto de classes é chamado de *Business Object*. (GEARY; HORSTMANN, 2012). Geralmente, em programas Java, esse tipo de controle está no arquivo faces.config.xml, o qual ainda veremos com mais detalhes.

Quando não se utiliza o contexto do JSF, é necessário utilizar os comandos da linguagem de marcação html para estruturar a página. Além do html, também é necessária alguma linguagem de programação para capturar os valores inseridos pelo usuário que, em muitos casos, precisam de conversão de tipos (um formulário que solicita um número para representar a idade pode gerar problemas se o usuário não digitar um número no momento de se fazer a conversão de tipos). No JSF isso acontece de maneira automática, preenchendo o objeto que será alvo de manipulações pela camada de controle e, posteriormente, pela camada de modelo. Como desvantagem da utilização do JSF, podemos destacar a sua elevada dependência de componentes e a sua curva de aprendizagem, pois, devido ao grande número de recursos, você precisará treinar bastante para entender e compreender o papel de cada um em seu sistema.

Instalação e configuração do ambiente de desenvolvimento

Se você ainda não tem uma IDE de desenvolvimento Java, aconselhamos instalar a *Eclipse IDE for Java EE Developers*, que contém a maioria dos pacotes necessários para a execução das atividades do curso. Faça o *download* da IDE Eclipse acessando o endereço <https://www.eclipse.org/downloads/packages/eclipse-ide-java-ee-developers/oxygen2>. Nele você terá acesso às informações e às diversas versões da IDE. Escolha a opção de download de acordo com seu sistema operacional.

Lembre-se que, para rodar o Eclipse, também são necessários recursos ligados ao Java, como por exemplo um JDK (*Java Development Kit*). Para instalar o JDK acesse <http://www.oracle.com/technetwork/java/javase/downloads/jdk9-downloads-3848520.html>. Faça o *download* do arquivo compatível com a sua versão de sistema operacional.

Para que você possa publicar seus programas WEB, você precisará de um servidor de aplicação. Algumas opções conhecidas são o *JBoss* e o *Tomcat*, porém, nesse curso, utilizaremos o *WildFly*, que tem mantido atualizações constantes. Para fazer o donwload de sua última versão, acesse <http://wildfly.org/downloads/>. Busque a última versão, atentando para se o programa tem suporte para a versão do Eclipse instalado e para o seu sistema operacional. Ao terminar de baixar o arquivo, descompacte-o em uma pasta de sua preferência para que você possa referenciá-la posteriormente na configuração de seu ambiente no Eclipse.

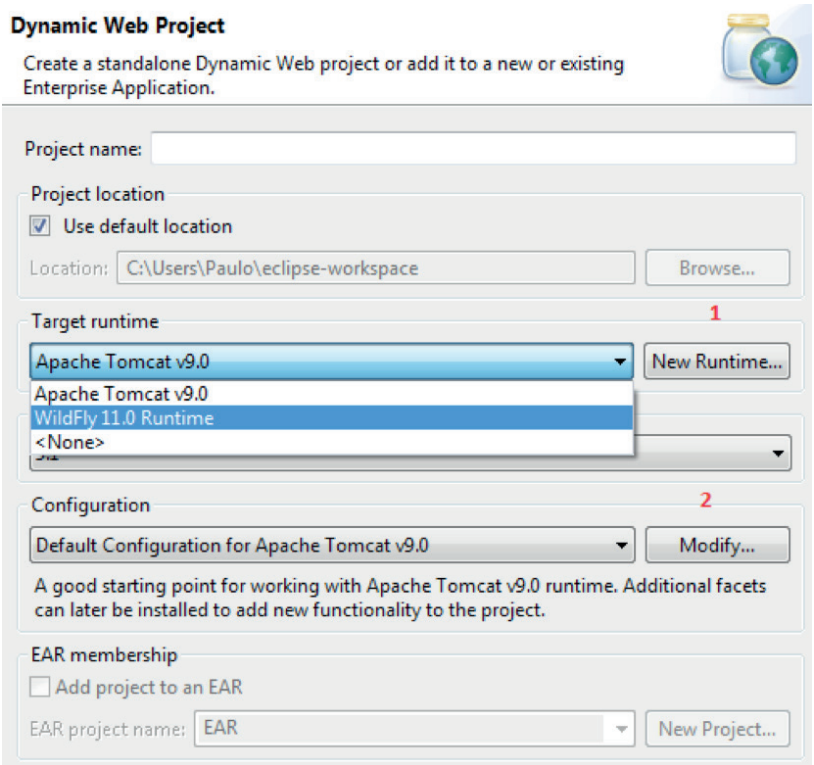
Por fim, é necessário baixar o conector do banco de dados que será utilizado em seu projeto. Existe um conector para cada banco de dados possível. No nosso caso, utilizaremos o conector do MySQL. Para baixá-lo para o Java, acesse: <https://dev.mysql.com/downloads/connector/j/>.

Criação de um projeto web com JSF

Agora que você instalou todos os recursos necessários, abra a IDE Eclipse, escolha o seu *workspace*. A tela inicial da IDE surgirá e, para começar as configurações pertinentes a inclusão do servidor de aplicação Wildfly, acesse o menu File > New > Other. Uma janela de opções se abrirá, procure pela pasta "web" e expanda-a. Escolha a opção *Dynamic Web Projetc*. Após selecionar a opção para criar

um projeto web dinâmico, na tela seguinte, você deverá especificar o nome do projeto, bem como escolher o servidor de execução no campo "*Target runtime*", conforme apresentado na Figura 1.2.

Figura 1.2 | Configurações de um projeto web dinâmico



Fonte: captura de tela da *Eclipse IDE for Java EE Developers*.

Para configurar seu servidor de aplicação, caso não tenha nenhuma opção instalada ou configurada você deverá:

1. Acesse o botão indicado na Figura 1.2 pelo número 1.
2. Escolha a opção que você deseja configurar. Clique em "*Next*".
3. Coloque o caminho dos arquivos que você descompactou no campo "*Home Directory*". Isso permitirá que o Eclipse tenha os arquivos necessários para fazer as configurações adequadas. Por fim, clique para finalizar (*Finish*).

Assimile

Como saber se o meu projeto está habilitado ou não para utilizar o JSF? Nesse caso, você tem duas opções.

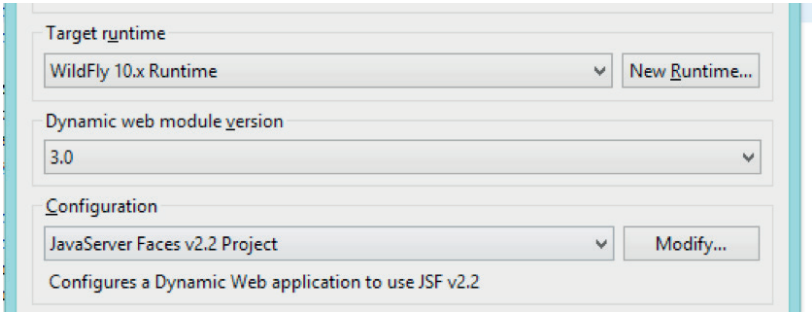
1. **Configurando no momento de criação do projeto:** ao criar um projeto, você pode optar, após escolher o servidor de aplicação para o projeto (*target runtime*), por qual versão do módulo *dynamic web* você utilizará. Nesse projeto, utilizaremos o 3.0. Em seguida, no campo *configuration*, utilize o botão representado pelo número 2 na Figura 1.2 para escolher uma versão do JSF (optamos pela versão 2.2), conforme ilustra a Figura 1.3 (a) abaixo.

2. **Configurando após a criação do projeto:** após a criação do projeto no seu ambiente, você deverá clicar com o botão direito em seu projeto e navegar até propriedades. Busque a aba *Project Facets*. Ao clicar nesse item, aparecerão os elementos presentes em seu projeto. Um desses elementos será o Java Server Faces e sua respectiva versão. Escolha a versão e marque o quadro para utilizar o contexto dentro de seu projeto.

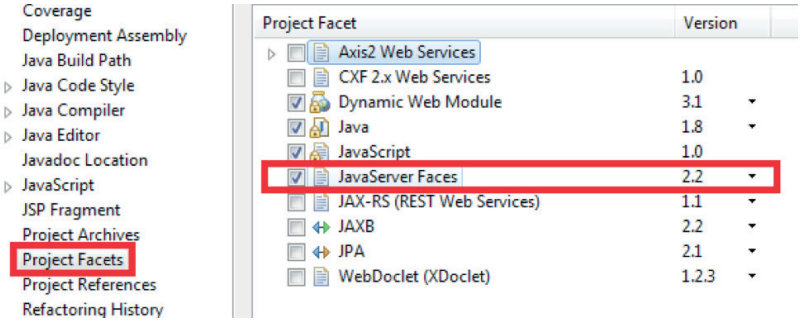
A Figura 1.3 te auxiliará na compreensão dos passos explicitados para habilitar o JSF.

Figura 1.3 | Escolha e habilitação do JSF para o projeto

a) No momento da criação do projeto



b) Após criação do projeto

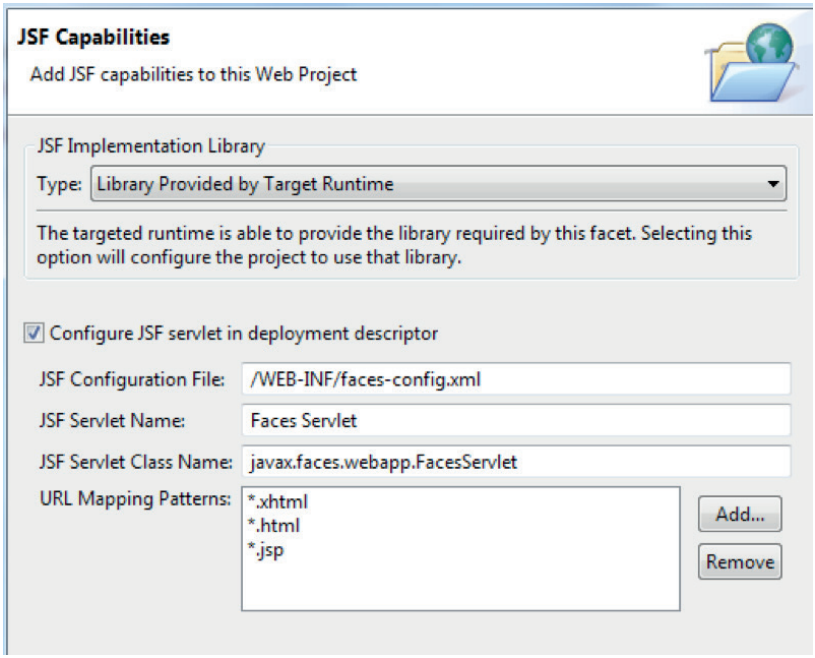


Fonte: captura de tela da *Eclipse IDE for Java EE Developers*, elaborada pelo autor.

Ao escolher essa opção, aparecerá uma mensagem de alerta na parte inferior de seu projeto, solicitando que você faça configurações sobre a atuação do JSF em seu projeto. Nas configurações estão presentes os principais elementos para você operacionalizar o JSF dentro de seu projeto, com destaque para o arquivo *faces.config.xml*.

A Figura 1.4 apresenta a interface para as configurações necessárias do JSF. Nesse contexto, a única configuração que aconselhamos que seja alterada é no elemento "*URL Mapping Pattern*". Ele vem com um padrão que, em algumas versões de Eclipse, não é identificado de maneira adequada o JSF. Para ter certeza de que suas páginas terão o padrão JSF presente, exclua a opção original com o botão "*remove*", apresentado na Figura 1.4, e acrescente, com o botão "*add*", os seguintes elementos: *.xhtml, *.html. *.jsp. Desta forma, você está dizendo ao seu projeto que todas as páginas com extensões .xhtml, .html e .jsp terão o JSF como *framework* atuante em sua estrutura.

Figura 1.4 | Configurações iniciais do JSF em seu projeto



Fonte: captura de tela da *Eclipse IDE for Java EE Developers*, elaborada pelo autor.

Ao criar um projeto seguindo essas premissas, o Eclipse cria um conjunto de arquivos e pastas padrão para atender aos requisitos de um projeto WEB. Para atender ao MVC, basta que você crie pacotes na pasta **src** para receber inicialmente classes de controle e de modelo. As classes da camada de visão ficam na pasta ***WebContent***.

Um arquivo importante pré-criado pelo eclipse para esse tipo de projeto é o *faces-config.xml*, localizado na pasta responsável pela gestão da camada de visão (*WebContent*). Esse arquivo possui todas as configurações particulares ao JSF e seu funcionamento é análogo ao arquivo web.xml, que tem configurações de todo o projeto. Nesses tipos de arquivos, utilizamos *tags* para inserir contextos importantes ao nosso projeto.

Exemplificando

O código, a seguir, demonstra como as configurações são feitas em arquivos *xml* por meio de *tags*.

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config
 xmlns="http://xmlns.jcp.org/xml/ns/javaee"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
javaee
http://xmlns.jcp.org/xml/ns/javaee/web-
facesconfig_2_2.xsd"
    version="2.2">
</faces-config>
```

Já os arquivos *Faces Servlets* recebem respostas de acordo com os eventos ou ações executadas nos arquivos JSF. Eles funcionam como uma classe intermediária, que recebe da JSF as demandas e as encaminha para as devidas classes dentro do sistema.

Se ao criar o projeto você marcou a opção *Generate web.xml deployment descriptor*, então o arquivo web.xml também pertence ao *WebContent*. Esse arquivo possui configurações que permitem que as páginas de extensão xhtml, html e jsp sejam controladas por *Servlets*. Você pode escolher quaisquer formatos válidos de páginas WEB.

Exemplificando

O código, a seguir, demonstra como as configurações de mapeamento de padrão são feitas no arquivo web.xml

```
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
    <url-pattern>*.html</url-pattern>
    <url-pattern>*.jsp</url-pattern>
  </servlet-mapping>
```

Para testar se as configurações foram aplicadas com sucesso ao projeto, crie na pasta *WebContent* uma página html qualquer. Na tag html da sua página, digite:

```
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html">
```

Se ao digitar *xmlns:h=* aparecer uma lista de opções, significa que o JSF está habilitado em sua página. Caso essa lista não apareça, volte nas configurações e verifique se todos os passos foram seguidos corretamente.

Pesquise mais

Você conhece o xhtml? Ele facilita bastante a construção de interfaces. Leia mais sobre o tema em:

SILVA, M. S. **Tutorial – XHTML**. 23 nov. 2011. [s.d]. Disponível em: <http://www.maujor.com/tutorial/xhtml.php>. Acesso em: 2 maio 2018.

Caro estudante, nesta seção você viu todos os passos necessários para instalar, configurar e habilitar o JSF em um projeto WEB. Além disso, vimos a importância do padrão de *software* MVC e como as classes devem ser organizadas pelas premissas do padrão. Agora estamos prontos para começar a produzir!

Sem medo de errar

Agora que você já conhece algumas ferramentas necessárias para o desenvolvimento web, você pode iniciar seu trabalho na empresa. Lembrando que você foi alocado para o projeto de produção de um sistema para o setor de revisão mecânica de uma concessionária. Sua primeira demanda consiste em elaborar um relatório para o cliente, apontando qual arquitetura de *software* será utilizada e por que esse padrão será adotado. Além disso, também é necessário incluir no documento um esboço inicial das classes que farão parte do sistema. Essa etapa auxilia no desenvolvimento dos objetos na camada de visão.

Agora que os estudos foram realizados, podemos verificar que o padrão arquitetural MVC auxilia na organização de diversos elementos dentro de um projeto. Esse tipo de situação é definido

pela organização de conteúdos de acordo com sua natureza e importância dentro do sistema. Um sistema com os devidos *plugins* e recursos instalados facilita a organização e a programação de elementos fundamentais dentro da construção do *software*.

Para resolver o problema do software da concessionária, podemos utilizar o Eclipse com as devidas configurações para receber projetos *web* e o *framework* JSF. Ele é uma ferramenta gratuita, de grande utilização no mercado e compatível com servidores de aplicação modernos, tais como o WildFly, e com a utilização do *framework* JSF. Os recursos e ferramentas adequadas para a produção dos *softwares* WEB facilitam a construção dos recursos e te auxiliam a organizar os arquivos e facilitar futuras pesquisas.

As classes a serem criadas precisam estar em pacotes determinados para atender a arquitetura MVC. No pacote de modelo, podem estar presentes classes candidatas, como Pedido.java, ServicoMecanico.java, Mecanico.java, Cliente.java e Carro.java. Essas classes vão desenhar os elementos presentes no mundo real que são necessários para atuarem dentro do mundo computacional.

Já em elementos que fazem a apresentação do sistema para o usuário, podemos destacar páginas web como a de CadastroDeServicoMecanico.html, CadastroMecanicos.html, dentre outros. Essas páginas ficam em um pacote *WebContent* que tem a mesma função de um pacote de visão.

E, para fazer a conexão das informações, utilizamos classes do tipo MBMecanico.java, ServletCliente.java, CarroManagedBean.java, geralmente em pacotes de controle.

Esses fatores são fundamentais para serem trabalhados no início do projeto, pois facilitam a condução das ações por parte da equipe de desenvolvedores. Um projeto bem arrumado e organizado dentro de seus propósitos facilita a dinâmica da construção do sistema, auxiliando na percepção de erros, na facilidade de utilização do software pelo reuso e diminuindo as chances de não aceitação do software por parte do cliente.

Pensar em classes adequadas, na utilização de orientação a objetos e na modularização de um projeto facilita ações futuras junto a esse sistema. Um sistema que começa certo tem tudo para continuar certo, enquanto um sistema que começa errado dificilmente conseguirá ter êxito.

A qualidade em construção de softwares deve ser levada a sério. Para isso, configurar as especificidades do sistema diminuem o retrabalho e a probabilidade de perda de produtividade.

Agora complete o relatório com um esboço inicial das classes que serão utilizadas, para que você possa, após a aprovação, implementá-las.

Avançando na prática

Aplicativo web para livraria

Descrição da situação-problema

Você foi procurado por uma pequena livraria que almeja expandir seus negócios com as vendas *online*. O proprietário não tem a menor ideia do valor que será necessário investir para a solução tecnológica, nem do tempo necessário para a implementação. Após uma primeira reunião com seu cliente, você foi solicitado a apresentar um documento contendo: (i) as ferramentas que você utilizará para desenvolver o projeto e (ii) os principais artefatos entregues para o cliente.

Resolução da situação-problema

O desenvolvimento do projeto será feito usando ferramentas gratuitas, assim o custo se torna mais viável. A IDE de desenvolvimento será o Eclipse, uma das ferramentas de desenvolvimento de software mais utilizadas no mundo. Ela atuará com o banco de dados MySQL, que é gratuito e bastante difundido dentro da programação de sistemas WEB, o que também impacta no custo. Esse sistema utilizará conceitos de MVC e de programação WEB em Java.

Os principais artefatos a serem entregues por você são aqueles que agregarão valor ao cliente. Nesse caso, destacam-se como artefatos de um sistema: um banco de dados modelado e com dados dos clientes da livraria; o cadastro de livros e outros produtos que sejam comercializados na livraria; um cadastro com dados básicos dos clientes; e um sistema WEB que pode ser acessado via navegador para realizar consultas, cadastros, exclusões e atualização de dados de livros, clientes e outros produtos presentes na loja.

Por fim, apesar de não ser um elemento físico, a empresa se compromete a gerenciar os servidores de aplicação e de banco de dados para que a livraria se preocupe somente com o negócio. A parte técnica ficará por sua conta.

Faça valer a pena

1. O desenvolvimento para WEB permite que sistemas produzidos sejam feitos de forma organizada e precisa, priorizando aspectos de qualidade. O padrão MVC de arquitetura de software permite que elementos sejam organizados de acordo com os padrões de qualidade exigidos de um desenvolvedor.

Sobre o padrão MVC (Modelo, Visão, Controle), marque a alternativa correta.

a) O padrão MVC permite que classes que devem ser inseridas no banco de dados sejam organizadas no pacote de modelos, permitindo assim que exista comunicação com o banco de dados e a camada de controle.
b) O padrão MVC determina a separação de classes que tenham contextos iguais dentro de um mesmo conjunto de funcionalidades. Nesse caso, classes que sejam relacionadas à comunicação com o cliente devem ficar separadas entre si, pois cada interface ou página refletirá uma classe do sistema.
c) O padrão MVC permite a separação de classes que tenham contextos diferentes, porém não determina quais elementos devem pertencer a cada um dos agrupamentos de classes. Por exemplo, uma página web pode pertencer tanto à camada de controle quanto à camada de visão.
d) O padrão MVC facilita o agrupamento de classes que tenham características em comum. No caso de classes que atuam juntas, captando as informações de uma página WEB, tais como Servlets e classes de *Business Objects*, elas pertencem ao mesmo contexto e ao mesmo grupo de classes.
e) O padrão MVC determina que a camada de visão seja a responsável por captar as informações do usuário, geralmente por forma de digitação de dados, e as transmita até que as classes de modelo realizem a persistência desse grupo de dados no banco de dados da aplicação.

2. A camada de modelo tem características peculiares em relação à sua atuação e ao controle de informações dentro dos sistemas informatizados, baseados no MVC. A seguir, são apresentadas alternativas que apresentam

características de elementos presentes na camada de modelo de uma aplicação.

I. A classe que possui todos os atributos do problema geralmente é composta por um nome significativo ao contexto, aos atributos privados, aos métodos públicos e aos construtores que dinamizam sua atuação junto ao sistema.
II. As classes de *Business Object* têm por principal característica intermediar o contato das classes que pertencem à camada de modelo e às informações manipuladas pela camada de controle. Essa é uma forma de comunicação entre classes que expressa notoriamente o sentido de orientação a objetos: comunicação de classes em contextos distintos.
III. As classes de *Data Access Object* (DAO) são responsáveis por captar as informações vindas da camada de controle por intermédio das classes de *Business Object* e por guardá-las no banco de dados da aplicação.

Sobre as afirmativas acima, marque a opção que apresenta a(s) alternativa(s) correta(s):

a) I somente.
b) I e II somente.
c) I e III somente.
d) II e III somente.
e) I, II e III.

3. Na aula de Programação para WEB II, dois alunos apresentavam seus argumentos sobre a utilização do padrão MVC nos trabalhos propostos pelo professor. O que foi discutido entre eles é apresentado nas assertivas a seguir:

I. Aluno 1: a utilização correta do padrão MVC está condicionada a uma alocação correta de classes dentro de seus respectivos pacotes, permitindo assim que o projeto seja organizado e permita que processos como evolução e manutenção de códigos sejam efetuados sem maiores problemas.
II. Aluno 2: o MVC é um padrão muito sólido de desenvolvimento, que requer conhecimento sobre o problema e a arquitetura do sistema. Por ser um contexto tão complexo, ele deve ser empregado em projetos que tenham natureza de complexidade alta, pois se o projeto for muito simples ou tiver complexidade baixa, não será viável gastar recursos e conhecimentos do MVC nesse tipo de software.

Sobre as assertivas acima, marque a alternativa correta.

a) As duas assertivas estão corretas e a segunda assertiva complementa a primeira.
b) As duas assertivas estão corretas e a segunda assertiva não complementa a primeira.
c) Somente a primeira assertiva está correta.
d) Somente a segunda assertiva está correta.
e) Nenhuma assertiva está correta.

Seção 1.2

Componentes visuais do JSF

Diálogo aberto

Caro estudante, bem-vindo a mais uma seção de estudo sobre o desenvolvimento web. Quantas páginas/aplicações web você já acessou hoje? O que achou do aspecto visual delas? Foi fácil encontrar o que você buscava? Você acha que os elementos usados (texto, botões, figuras, etc) foram elaborados com cuidado, com planejamento? Você faz ideia de como inserir e configurar esses elementos? Todas essas questões envolvem os componentes visuais do JSF, tema central do nosso estudo nessa seção.

Após o levantamento de requisitos e a escolha de uma arquitetura coesa para o desenvolvimento do *software*, inicia-se a implementação da solução. Nessa fase, a especificação dos recursos visuais e dos eventos que vão possibilitar uma comunicação eficiente entre usuários e sistema deve ser levada em conta, pois eles têm implicações na escolha das tecnologias empregadas na camada de visão. O desenvolvedor deve utilizar recursos que sejam facilitadores para a inserção de novas tecnologias e, ao mesmo, tempo pensar na relação desempenho versus usabilidade, principalmente, focando no público alvo do seu projeto. Os funcionários do setor de revisão mecânica podem não estar familiarizados com a utilização de sistemas computacionais, portanto, escolher um *framework* que dinamize as operações de inserção de elementos visuais, gráficos e de eventos é indispensável para o sucesso da solução informatizada. Nesse contexto, como os códigos dinâmicos e de eventos podem ser inseridos em elementos da camada de visão sem prejudicar sua estrutura? Nessa fase do projeto, você deverá criar uma página para cadastro de clientes, utilizando componentes visuais do framework JSF. Os dados a serem cadastrados são: nome do cliente, idade, CPF, endereço, telefone e e-mail. Além disso, a página deverá receber uma folha de estilo com as seguintes características: fonte da página predominantemente azul, arial, tamanho 12. Outros arquivos de CSS podem ser incluídos para melhorar ainda mais o aspecto produzido

na página. Lembrando que eles devem estar presentes pela atuação em conjunto com o JSF.

Além disso, você deverá inserir códigos javascript para atuar em conjunto com o JSF para dinamizar o feedback ao usuário. Nesse contexto, você pode inserir eventos e mensagens que auxiliem o usuário a interagir com o sistema.

Tome cuidado para não misturar aspectos de cores que tornem a navegação cansativa e eventos de script que tornem morosa a navegação. O sucesso da interface está altamente relacionado com a aceitação de uma solução WEB.

Para cumprir essa missão, você conhecerá componentes visuais no JSF e como inserir folhas de estilo e códigos javascript para tornar as páginas mais dinâmicas.

Vamos lá?!

Não pode faltar

Em um projeto de software você necessita criar elementos que facilitem a dinâmica de seu trabalho. Para tanto, a configuração dos *plugins*, *frameworks* e recursos a serem utilizados são fundamentais para melhorar a produtividade do trabalho. Um dos principais elementos utilizados para melhorar a criação de elementos na camada de visão é o *Java Server Faces* ou, como estamos utilizando em nosso curso, o JSF.

Para a criação de páginas web, é necessária a utilização da linguagem de marcação HTML para inserir elementos como rótulos (*label*), caixas de texto, imagens, títulos, etc. Esses recursos formam a estrutura da página, assim, qualquer janela de navegação (*browser*) é capaz de interpretar e apresentar as informações na tela para o usuário.

Ao optar por utilizar o *framework web* JSF, o código HTML passa a ser gerado no servidor por meio de *tags* especiais. Segundo Horstmann e Geary (2012), podemos fazer uma analogia entre o pacote Java *Swing*, que é usado para criar interfaces gráficas para aplicações *desktop*, com o JSF, que é usado para criar a interface gráfica em aplicações web.

Os componentes JSF podem ser inseridos em meio aos códigos *html*, em arquivos *xhtml* e *jsp*, portanto você reconhecerá muitos

comandos. Os elementos JSF serão inseridos dentro de uma estrutura *html*, conforme o esquema ilustrado na Figura 1.5.

Figura 1.5 | Esquema de inserção de comandos JSF

Fonte: elaborada pelo autor.

No JSF, existem diversos recursos que podem ser empregados em formulários (MANN, 2005), mas o primeiro passo é incorporar os *frameworks* de componentes visuais na página em que se pretende utilizar o JSF. Para isso, é preciso inserir os seguintes atributos na tag <html>:

```
1.  <html
2.        xmlns="http://www.w3.org/1999/xhtml"
3.        xmlns:h="http://xmlns.jcp.org/jsf/html"
4.  >
```

Na linha 2, o atributo xmlns (*xml namespace*) especifica a diretriz padrão para o documento *xml*. Veja que, na linha 3, o atributo *xmlns* é atribuído a um identificador (***:h***), o que significa que todo elemento que possuir o prefixo ***h*** pertencerá a esse *namespace*.

Pesquise mais

Namespace é uma forma de organizar os arquivos dentro de um projeto. Essa nomenclatura é utilizada por vários softwares e sua definição é análoga a um pacote em Java. Veja os 5 primeiros minutos do vídeo a seguir:

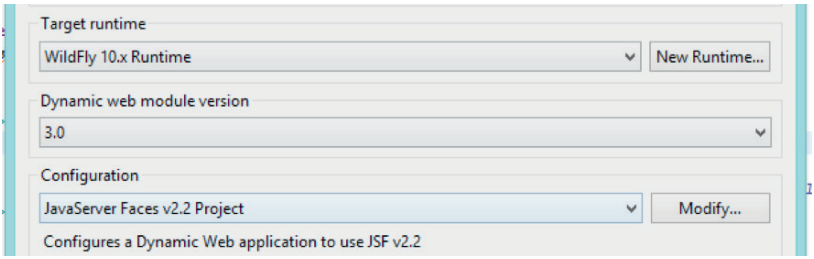
BÓSON TREINAMENTOS. **Programação em C# - Namespaces, Assemblies e a Diretiva using**. 5 mar. 2014. Disponível em <https://www.youtube.com/watch?v=P0_DZjaC2cY>. Acesso em: 12 mar. 2018.

Os atributos inseridos vão permitir acessar os elementos presentes no JSF, que são aplicados aos contextos de um formulário em uma página WEB. Existem outros *frameworks* de camada de visão, como *Richfaces*, *Primefaces* e tantos outros, que têm elementos pontuais para construção das interfaces. Eles serão abordados de forma mais detalhada posteriormente.

Após realizar todas as premissas para que seu ambiente esteja preparado para as configurações de conexão ao *framework* JSF, vamos criar um pequeno projeto para uma biblioteca para que você visualize recursos importantes do JSF e do padrão MVC.

Vamos iniciar criando um novo projeto do tipo *Dinamyc Web Project*, com o nome de "Biblioteca2". No momento da criação, além do nome, escolha as opções conforme a Figura 1.6.

Figura 1.6 | Configurações para criação do projeto



Fonte: captura de tela do Eclipse, elaborada pelo autor.

Para essa primeira etapa do sistema, criaremos uma interface gráfica para o cadastro de livros. Foi definido que um livro precisa de três atributos: nome do livro (*String*), autor (*String*) e número de Páginas (*int*). A página elaborada, portanto, tem um formulário que capta esses três elementos. Para criação do formulário, clique com o botão direito sobre o projeto e escolha *New* >> *HTML File*. Nomeio o arquivo como *cadastroLivro.xhtml* e clique em *finish*. Será criado um arquivo com a estrutura básica de um arquivo HTML 5.

Com a base do *html* criada, agora podemos inserir os comandos JSF, conforme o código exibido no Quadro 1.1. Nas linhas 3 e 4, são inseridos os *namespaces xml*. Repare que, na linha 6, a tag *<head>* ganha um prefixo *h*, pois agora ela será executada pelo JSF no servidor, o que gerará o *html* correspondente. O mesmo acontece para os campos de entrada de texto, nas linhas 13, 16 e 19, e para o botão, na linha 22. O atributo *value* dos campos de

entrada tem o valor #{xxx}, que será alterado conforme avançamos. Na linha 22, o atributo *action* do componente botão indica que o resultado chamará a mesma página. Na linha 25, deixamos um parágrafo que exibirá o resultado quando tudo estiver devidamente implementado e configurado.

Quadro 1.1 | Código para geração da página para cadastro de livros

```
1.  <!DOCTYPE html>
2.  <html
3.          xmlns="http://www.w3.org/1999/xhtml"
4.          xmlns:h="http://xmlns.jcp.org/jsf/html"
5.  >
6.  <h:head>
7.          <meta charset="UTF-8" />
8.          <title>Cadastro de livros</title>
9.  </h:head>
10. <h:body>
11.     <h:form>
12.         <p class="rotulo">Nome do livro:
13.             <h:inputText id="txtNomeLivro"
value="#{xxx}" class="entrada" />
14.         </p>
15.         <p class="rotulo">Autor:
16.             <h:inputText id="txtAutorLivro"
value="#{xxx}" class="entrada" />
17.         </p>
18.         <p class="rotulo">Número de páginas:
19.           <h:inputText id="txtNumPaginas"
value="#{xxx}" class="entrada" />
20.         </p>
21.         <p class="botao">
22.             <h:commandButton value="Submit"
                        action=" cadastroLivro" />
23.         </p>
24.     </h:form>
25.   <p>Livro confirmado: "#{xxx}" "#{xxx}" "#{xxx}"</p>
26. </h:body>
27. </html>
```

Fonte: elaborado pelo autor.

Olhe para a estrutura de pastas do projeto no Eclipse. O arquivo *cadastroLivro.xhtml* Foi criado dentro de *WebContent*, pois, ao escolhermos o tipo de projeto *Dynamic Web*, os arquivos pertencentes à camada de visão são destinados a essa pasta.

Nosso próximo passo no projeto do site para biblioteca é criar a classe "Livro". Essa classe será o modelo do objeto livro e deverá conter os campos que foram especificados. Para atender ao padrão MVC, iremos criar um novo pacote chamado "modelo", que será destinado a todas as classes dessa categoria. Clique com o botão direito no projeto e escolha *New* → *Package* e dê o nome escolhido. Novamente, clique com o botão direito no projeto e escolha *New* → *Class*. Verifique se o campo Package está como "modelo" e dê o nome de "Livro", em seguida, clique em *Finish*. A criação dessa classe em nada difere do modo utilizado para o padrão de desenvolvimento desktop, portanto o código ficará como no Quadro 1.2.

Quadro 1.2 | Classe Livro

```
1.  package modelo;
2.  public class Livro {
        //atributos
3.      private String _nome;
4.      private String _autor;
5.      private int _paginas;

        //construtores: personalizado e padrão
6.      public Livro(String nome, String autor, int
7.  paginas) {
8.          super();
9.          this._nome = nome;
10.         this._autor = autor;
11.         this._paginas = paginas;
12.     }
13.     public Livro() {
            //construtor padrão
14.     }

        //métodos
15.     public String getNome() {
16.         return _nome;
```

```
17.     }
18.     public void setNome(String nome) {
19.         this._nome = nome;
20.     }
21.     public String getAutor() {
22.         return _autor;
23.     }
24.     public void setAutor(String autor) {
25.         this._autor = autor;
26.     }
27.     public int getPaginas() {
28.         return _paginas;
29.     }
30.     public void setPaginas(int paginas) {
31.         this._paginas = paginas;
32.     }
}
```

Fonte: elaborado pelo autor.

Até o momento, temos uma parte do projeto na camada de visão e outra na camada de modelo. Para que essas partes se comuniquem, é preciso criar uma ***Managed Bean***. Esse novo elemento nada mais é do que uma classe em java, que funciona como um canal entre a interface gráfica e o *back-end* da aplicação, ou seja, classes modelo, regras de negócio e acesso ao banco de dados (FARIA, 2015).

Em uma *Managed Bean*, pode haver regras da lógica do negócio, portanto ela será criada em um novo pacote chamado "*controle*". Dentro do pacote de controle criaremos uma nova classe em java. Para isso, clique com o botão direito em cima do pacote e escolha *New* → *Class*, dando o nome de "*LivroMB*".

Reflita

Os dados do formulário poderiam ser capturados por uma *Managed Bean* sem ter uma classe modelo intermediária. Se isso é possível, existe vantagem em se criar classes modelos, como por exemplo, a classe Livro.java?

Usando a *Managed Bean*, os dados digitados pelo usuário serão capturados por métodos SET e guardados dentro de um objeto do tipo livro. O resgate dos dados será feito por métodos GET. Veja no Quadro 1.3 o código da classe *LivroMB*. Na linha 2, importamos a classe "Livro", que está no pacote "modelo", para instanciarmos um objeto desse tipo na linha 6. Na linha 3 é feita a referência à biblioteca *javax.faces.bean.ManagedBean* para usarmos esse recurso, com isso podemos usar a notação *@ManagedBean* para identificar o tipo da classe (linha 4). Na linha 7, deixamos explícito o construtor padrão da classe *LivroMB*, item obrigatório. Das linhas 9 a 26, utilizamos métodos *setXXXX*(), para guardar o valor de cada atributo que o usuário digitar na propriedade específica do objeto *n_livro*, e métodos *getXXXX*(), para ler os dados guardados dentro dos atributos do mesmo objeto.

Quadro 1.3 | Maneged Bean LivroMB

```
1.  package controle;
2.  import modelo.Livro;
3.  import javax.faces.bean.ManagedBean;
4.  @ManagedBean
5.  public class LivroMB {
6.        private Livro n_livro = new Livro();
7.        public LivroMB(){
          //construtor da classe LivroMB - item obrigatório
8.        }
9.        public String getNome() {
10.             return n_livro.getNome();
11.       }
12.       public void setNome(String nome) {
13.             n_livro.setNome(nome);
14.       }
15.       public String getAutor() {
16.             return n_livro.getAutor();
17.       }
18.       public void setAutor(String autor) {
19.             n_livro.setAutor(autor);
20.       }
21.       public String getPaginas() {
22.             return String.valueOf(n_livro.getPaginas());
```

```
23.         }
24.         public void setPaginas(String paginas) {
25.
26.         n _ livro.setPaginas(Integer.parseInt(paginas));
27.         }
```

Fonte: elaborado pelo autor.

Para que uma *Managed Bean* funcione, é preciso incluí-la no arquivo *faces-config.xml*, na pasta WEB-INF. É necessário acrescentar as *tags* desse elemento, conforme mostra o código do Quadro 1.4. Veja que das linhas 7 à 11 foram incluídas três *tags* dentro do corpo da <*managed-bean*>. Na linha 8, a primeira *tag*, <*managed-bean-name*>, será o nome usado para acessar os métodos dessa *managed*. Na linha 9 é especificada a pasta e o nome da classe. Por fim, na linha 10 é definido o escopo da *managed bean*, nesse caso, de requisição (*request*).

Quadro 1.4 | Inclusão da Managed Bean no arquivo faces-config.xml

```
1.  <?xml version="1.0" encoding="UTF-8"?>
2.  <faces-config
3.        xmlns="http://xmlns.jcp.org/xml/ns/javaee"
4.        xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
5.  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-
    facesconfig _ 2 _ 2.xsd"
6.        version="2.2">
7.        <managed-bean>
8.           <managed-bean-name>livroMB</managed-bean-name>
9.  <managed-bean-class>controle.LivroMB</managed-bean-
10. class>
11.          <managed-bean-scope>request</managed-bean-
    scope>
12.       </managed-bean>
    </faces-config>
```

Fonte: elaborado pelo autor.

Feitas todas as configurações, agora podemos usar a *Managed Bean* criada. Para isso, iremos altear o arquivo *cadastroLivro.xhtml*. Em cada componente que possui o campo *value* precisamos indicar o uso da classe "*LivroMB*", conforme mostra o código no Quadro 1.5.

Quadro 1.5 | Usando uma *Managed Bean* no arquivo *cadastroLivros.xhtml*

```
1.  <h:inputText id="txtNomeLivro" value="#{livroMB.nome}"/>
2.  <h:inputText id="txtAutorLivro" value="#{livroMB.autor}" />
3.  <h:inputText id="txtPaginas" value="#{livroMB.paginas}"
4.  />
    <p>Livro confirmado: "#{livroMB.nome}" "#{livroMB.autor}"
    "#{livroMB.paginas}"</p>
```

Fonte: elaborado pelo autor.

Na linha 1 do Quadro 1.5, a estrutura "#{livroMB.nome}" dá o comando para acessar o método *setNome()* da *Managed bean* chamada *livroMB*. Veja que não foi preciso colocar todo o comando, usamos apenas "*nome*". O mesmo para *setAutor* e *setPaginas()*, nas linhas 2 e 3. Na linha 4, os métodos que serão acessados são o *getNome()*, *getAutor()* e *getPaginas()*, esse controle de acesso aos métodos é feito pela *Managed Bean*.

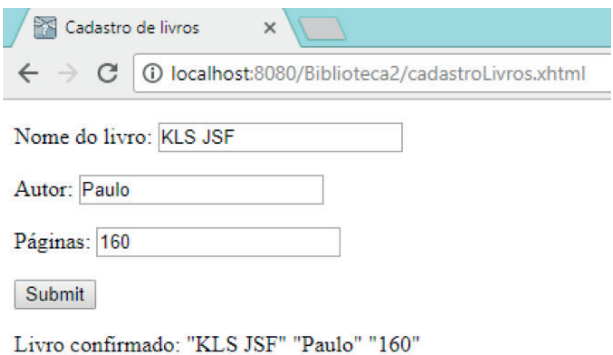
Assimile

Managed Beans são usadas para a comunicação entre as classes de visão e de modelo. Para isso são usados métodos SET e GET.

A notação **#{nomeBean.propriedade}** é usada para acessar os métodos (SET e GET) e realizar a comunicação entre as camadas.

Para testar tudo o que fez até o momento, clique com o botão direito no arquivo *cadastroLivros.xhtml* e selecione *Run as* → *Run on Server*. Preencha os campos e, clicando no botão *Submit*, você deverá ter um resultado similar ao da Figura 1.7.

Figura 1.7 | Resultado esperado da criação do projeto para biblioteca



Fonte: elaborada pelo autor.

Temos um projeto JSF funcionando corretamente, mas o que achou da aparência? Será que um usuário final gostaria do resultado visual da interface? Certamente que não.

Um aplicativo web precisa ser funcional, mas também precisa ter uma boa aparência. Se estamos falando de estilo, então temos que criar uma folha de estilo (CSS).

Exemplificando

Para exemplificar o uso de folhas de estilo, dentro da pasta *WebContent*, vamos criar a pasta *resources* e, dentro dessa, mais uma chamada *css*. Dentro da pasta *css*, vamos criar um arquivo chamado *estilo1.css*. Nesse arquivo, podemos fazer configurações, por exemplo, para o rótulo e para as caixas de texto com o seguinte código:

```
.rotulo{
	font-family: Arial, Sans-serif;
	text-transform: uppercase;
	font-size: 12pt;
	font-weight: bold;
}
.entrada{
	border: 1px inset;
	background-color: #D3D3D3;
	color: #000000;
	display:block;
}
```

Agora vamos abrir o arquivo *cadastroLivros.xhtml* e, dentro da *tag* <*h:head*>, inserir a chamada ao arquivo *css*. Portanto, o código ficará:

```
<h:head>
  <meta charset="UTF-8" />
  <title>Cadastro de livros</title>
  <h:outputStylesheet library="css"
name="estilo1.css" />
</h:head>
```

O arquivo CSS também poderia ser adicionado usando a *tag* <*link*> como se fosse uma página *html* sem JSF.

Além de especificarmos estilos com o CSS, também podemos adicionar Java Script (JS) para deixar a página mais dinâmica. Essa linguagem de scripts nos possibilita inserir e remover elementos na tela, sem a necessidade de fazer requisições ao servidor.

Exemplificando

Para exemplificar o uso de JS, dentro da pasta *resources* vamos criar mais uma chamada "js". Dentro dessa nova pasta, crie um arquivo com o nome de "*meuScript1.js*". Nesse arquivo, podemos usar qualquer comando javaScript, por exemplo, mostrar a data e hora assim que a página for carregada com o seguinte código:

```
window.onload = function(){
       alert("Hoje é " + Date());
}
```

Agora vamos abrir o arquivo *cadastroLivros.xhtml* e, dentro da *tag* <*h:head*>, inserir a chamada ao arquivo js. Portanto, o código ficará:

```
<h:head>
  <meta charset="UTF-8" />
  <title>Cadastro de livros</title>
  <h:outputStylesheet library="css"
name="estilo1.css" />
  <h:outputScript library="js" name="meuScript1.js"
/>
</h:head>
```

Nesta seção, vimos passo a passo como criar uma página utilizando o JSF e como criar uma classe modelo e uma classe de controle respeitando os critérios da arquitetura MVC. Vimos que existe uma classe especial usada para comunicação entre camadas no desenvolvimento usando JSF, a *Managed Bean*. E, para finalizar, vimos como inserir folhas de estilo e *javaScript* nas páginas JSF.

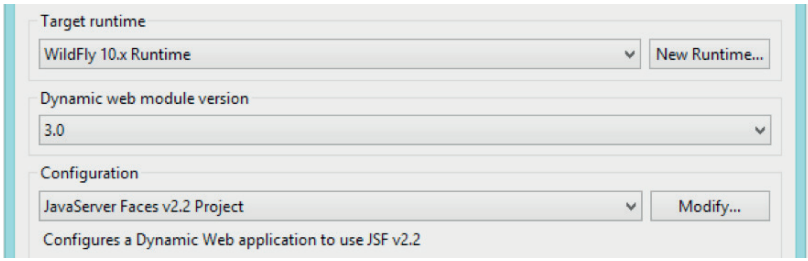
Sem medo de errar

Agora que você aprendeu a utilizar alguns componentes do JSF em páginas xhtml, podemos dar continuidade ao desenvolvimento do sistema para o setor de revisão mecânica. Foi pedido a você

a implementação da página que fará o cadastro dos clientes. Para implementar essa primeira etapa, siga os seguintes passos:

1. Crie um novo projeto web dinâmico com o nome "Concessionaria" e com as configurações, conforme Figura 1.8.

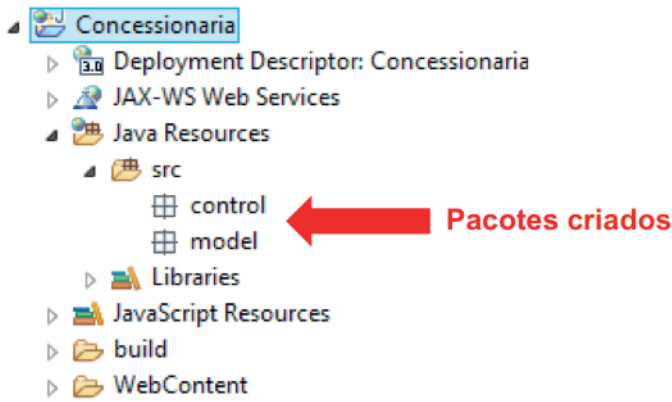
Figura 1.8 | Configurações iniciais



Fonte: captura de tela do Eclipse, elaborada pelo autor.

2. Crie nesse projeto dois pacotes, um chamado "model" e outro chamado "control". Veja a estrutura na Figura 1.9.

Figura 1.9 | Estrutura de pastas



Fonte: elaborada pelo autor.

3. Agora crie um novo arquivo chamado cadastroCliente.xhtml, usando como base um HTML 5, e faça as implementações JSF necessárias para construir o formulário. Lembrando que os campos deverão ser: nome do cliente, idade, CPF, endereço, telefone e e-mail.

4. O próximo passo é criar a classe modelo para o cliente. Lembre-se de que ela deverá estar dentro do pacote model.

5. Para que a camada de visão e modelo se comuniquem, é preciso criar uma Managed Bean, então crie a classe ClienteMB e faça devida implementação.

6. O próximo passo é configurar as tags, conforme os nomes com que você criou os demais arquivos.

```
<managed-bean>
   <managed-bean-name>classeMB</managed-bean-name>
   <managed-bean-class>pacote.ClasseMB</managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

7. Agora é o momento de criar a parte da folha de estilo para melhorar a aparência da interface gráfica. Para isso, dentro da pasta WebContent, crie uma pasta chamada resources e, dentro desta, crie uma nova pasta chamada css. Agora crie um novo arquivo para a folha de estilo dentro dessa pasta e faça as configurações solicitadas: letra da página predominantemente azul, arial, tamanho 12, além de outras que julgar necessário.

8. Agora utilize a tag <h:outputStylesheet /> para fazer a ligação do CSS com o arquivo xhtml.

9. Também dentro da pasta resources, crie uma nova pasta chamada js e, dentro dessa pasta, crie um arquivo para o código javaScript e coloque mensagens para auxiliar o usuário do sistema.

10. Por fim, utilize a tag <h:outputScript /> para fazer a ligação do JS com o arquivo xhtml.

Com esses 10 passos, você criará a primeira parte do sistema, responsável pela interação do usuário com as funcionalidades. Ainda neste livro, você aprenderá como persistir esses dados em um banco de dados.

Avançando na prática

Consultoria para desenvolvimento web

Descrição da situação-problema

Após encontrar seu currículo na internet, uma empresa que está tentando migrar seu sistema *desktop* para um aplicativo web

o contratou para uma consultoria. Em sua primeira visita, um dos programadores mostrou o resultado da página que estava criando. Ele havia feito todas as configurações necessárias para o funcionamento do JSF, mas ainda não obtinha êxito na execução do projeto. Diante do problema, o que deve ser verificado no sistema? Quais arquivos podem conter erros que o programador não percebeu?

Resolução da situação-problema

Certamente, o uso de *frameworks* facilita o trabalho de desenvolvimento de soluções de *softwares*. Entretanto, é preciso conhecer os recursos e cada componente do sistema, pois muitos códigos são gerados de forma automatizada pelos sistemas. Para encontrar um erro no sistema, você pode seguir uma sequência de revisões:

- Comece analisando o arquivo de interface gráfica. Veja se o nome e a propriedade da Managed Bean estão corretos.

- Abra o arquivo *faces-config.xml* e verifique se os valores das *tags* correspondem, de fato, ao nome da classe e ao atributo *value* usado no arquivo xhtml:

```
<managed-bean>
   <managed-bean-name>classeMB</managed-bean-name>
   <managed-bean-class>pasta.ClasseMB</managed-bean-class>
   <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

- Outro arquivo que é muito importante verificar é o web.xml, dentro da pasta WEB-INF. Nesse arquivo é feito o mapeamento do JSF por meio da tag <servlet-mapping>. O valor dessa tag implicará na maneira de acessar a url do projeto. Caso o arquivo esteja da forma a seguir, o endereço de acesso será: http://localhost:8080/Projeto/faces/arquivo.xhtml:

```
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

- Outro ponto a se verificar é o nome dos campos na classe modelo, se eles são públicos ou privados e se os métodos get e set estão devidamente escritos.

- Verifique também os nomes nas demais classes, como, por exemplo, na *Managed Bean*.

Não se esqueça de sempre ler os erros, e tentar identificá-los. Muitos erros são possíveis de entender e resolver apenas lendo a descrição.

Faça valer a pena

1. A criação de elementos em uma interface envolve etapas importantes, que devem seguir passos, desde a modelagem até a entrega do *software*. No desenvolvimento de *softwares* orientados a objetos com a utilização do MVC, alguns passos são fundamentais e devem ser executados. Sobre esses passos, foram feitas as seguintes afirmativas:

I. Organização de pacotes, conforme a natureza de atuação dentro das classes.
II. Definição da tecnologia a ser utilizada na camada de visão, como o JSF.
III. Definição de *framework* para atuar em páginas WEB, desconsiderando a etapa de conferência da compatibilidade com o servidor de aplicação.

Sobre as afirmativas acima, marque a opção que apresenta somente afirmativas corretas.

a) I, II e III.
b) I e III somente.
c) I e II somente.
d) III somente.
e) II e III somente .

2. A camada modelo envolve a programação de classes que são importantes para a comunicação do sistema com o banco de dados da aplicação, além de conter elementos que aproximam o software produzido do mundo real. Sobre a atuação do *framework* JSF e as classes de modelo, foram feitas as seguintes assertivas.

I. O JSF tem atuação destacada dentro das classes de modelo, pois ele propicia a inserção de elementos e recursos da WEB para interagir com o usuário e facilitar a aceitação do sistema por parte dos clientes.
PORQUE
II. O JSF permite, por meio da criação de classes, que elementos presentes na camada de modelo sejam expressos em forma de tags html, permitindo que a camada de controle tenha uma atuação precisa na comunicação entre camadas.

Sobre as assertivas acima, marque a alternativa correta.

a) As duas assertivas estão corretas e a segunda explica o que foi dito na segunda assertiva.
b) As duas assertivas estão corretas e a segunda não explica a primeira.
c) Somente a primeira assertiva está correta.
d) Somente a segunda assertiva está correta.
e) As duas assertivas estão incorretas.

3. A inserção de recursos como o JSF pode mudar toda a estrutura já construída de um sistema. Essas mudanças são necessárias para que sistemas já desenvolvidos passem a utilizar recursos informatizados mais adequados para a realização da comunicação entre camadas em uma estrutura de software desenvolvida nas bases do MVC, além de recursos gráficos na camada de visão, como o css e o javascript. A inserção de elementos como as ManagedBeans em substituição a arquivos do tipo Servlet pode gerar impactos profundos no código fonte e na camada de visão de uma aplicação.

Sobre a substituição das classes Servlet por classes ManagedBeans ao inserirmos o JSF, podemos afirmar que

a) A inserção de metodologias baseadas no JSF altera a modelagem de classes realizadas na camada de modelo e, consequentemente, em todas as classes ligadas à inserção de informações no banco de dados devido a mudança de tecnologia na camada de controle.

b) A inserção de metodologias como o JSF permite que as classes da camada de modelo e as páginas criadas na camada de visão sejam aproveitadas em sua integralidade, gerando impactos somente na forma de se conduzir as tarefas na camada de controle.
c) A inserção de tecnologias como o JSF causa mudanças mais drásticas na camada de visão, alterando a forma de se capturar os dados na interface e na camada de controle por meio da simplificação de captações vindas da camada de visão pela utilização de objetos.
d) A inserção de metodologias como o JSF se torna muito complexa para ser realizada em sistemas baseados no MVC devido à necessidade de alteração em todas as camadas envolvidas na organização do projeto.
e) A inserção de metodologias como o JSF facilita a gestão dos dados inseridos no banco de dados por meio das mudanças realizadas na camada de visão, que passam a utilizar recursos modernos do html, porém sem a utilização direta de objetos, fator realizado pela camada de controle.

Seção 1.3

Componentes de organização

Diálogo aberto

Você já começou a preencher um formulário e conforme preenchia alguns campos o valor de outros se alterava? Por exemplo, ao escolher um estado o conjunto de cidades disponíveis se altera. Isso ocorre porque alguns campos permitem ter seu preenchimento de forma dinâmica, o que melhora a experiência do usuário e valoriza um software. Até o presente momento, você já sabe como criar uma página para interação do usuário usando a extensão xhtml, e como fazer a interação dessa classe com o sistema por meio de uma Manage Bean. Agora chegou a hora de você conhecer novos recursos para incrementar as interfaces gráficas.

Dando continuidade no desenvolvimento para o sistema do setor de revisão mecânica, você deverá criar a interface gráfica que fará o cadastro do veículo a ser atendido. Devem ser cadastros o CPF do cliente, a marca do veículo, seu modelo e ano. Nessa tela é muito importante que o usuário do sistema tenha a opção de selecionar quais os serviços que serão realizados no veículo, sendo que a oficina oferece os seguintes serviços: troca de pneus, troca de óleo, troca de pastilhas de freio, injeção eletrônica, reparos em motores, sistema elétrico e suspensão. Como você implementará esses recursos em seu software? Será preciso criar uma ou mais Manage Bean?

Nessa seção, você aprofundará seu conhecimento a respeito do funcionamento do *framework* JSF, além de aprender novos recursos para a interface gráfica, alguns que só trabalham de maneira estática, outros de forma dinâmica. Bons estudos!

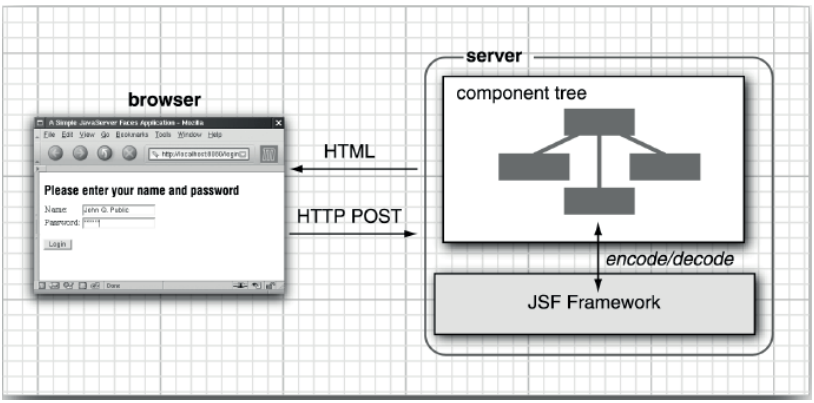
Não pode faltar

Você já aprendeu que para criar uma página JSF é preciso usar *tags* específicas dessa tecnologia, por exemplo, *<h:inputText>*. Um dos grandes diferenciais do JSF é que seu mecanismo de

funcionamento é baseado em componentes e não em eventos, como é o caso dos *frameworks* SpringMVC e VRaptor (CORDEIRO, 2014). Ao utilizar o JSF, a criação da tela é responsabilidade desse *framework*, que irá criar uma árvore de componentes para ser renderizada (HORSTMANN, 2012).

A primeira vez que a página é enviada para o servidor, o JSF irá criar a árvore de componentes, já nas demais requisições esta será recuperada, pois o JSF mantém um cache de árvores (FARIA, 2015). A vantagem é que a árvore é mantida entre a requisição e a resposta, o que possibilita a validação dos dados enviados com os dados recebidos. A Figura 1.9 ilustra um formulário que é submetido a um servidor, que por sua vez, por meio do JSF, cria a árvore de componentes, renderiza e devolve para o cliente o código HTML.

Figura 1.9 | JSF e sua árvore de componentes



Fonte: Horstmann (2012, p. 26).

Assimile

Cada componente da árvore possui um renderizador que é responsável por gerar o código HTML correspondente, esse processo é chamado de *encoding*.

Após a resposta do servidor, por meio de um navegador, o usuário irá interagir com os elementos, por exemplo, preenchendo os campos de um formulário e irá submeter os dados. Quando os dados chegarem no servidor, o *framework* JSF fará o processo

de decodificação, no qual cada objeto interpretará os dados recebidos de forma independente consultando uma tabela (HORSTMANN, 2012).

Exemplificando

Para inserirmos uma caixa de texto em uma página JSF usamos a tag *<h:inputText>*. O renderizador desse componente irá gerar o seguinte código HTML: *<input type="text" name="nome" value="valor"/>*.

No desenvolvimento de interfaces gráficas é importante manter uma organização lógica dos componentes, que deve ser representada graficamente para que o usuário tenha a melhor experiência. Tanto a organização, quanto os componentes gráficos, podem ser estáticos ou dinâmicos, dependendo da necessidade da aplicação.

Na seção anterior, vimos alguns componentes gráficos para a construção de interfaces gráficas com JSF e agora vamos aumentar nosso repertório para que possamos implementar interfaces mais completas.

O componente *<h:outputText>* pode ser usado para exibir textos na tela, de forma estática ou dinâmica. Será estático quando o programador especificar o conteúdo e este não se altera em nenhum momento, e será dinâmico quando o conteúdo depende de decisões na classe de controle. Por exemplo, por meio dos códigos:

Estático: `<h:outputText value="Programação para web "/>`

Dinâmico: `<h:outputText value="#{minhaBean.descricao/>`

Uma observação interessante é o sobre o HTML gerado pelos renderizadores dos comandos *h:outputText* ilustrados. Para o estático será gerado um texto simples, já para o texto dinâmico será criado dentro de uma tag span, similar ao resultado:

```
Bem-vindo <span style="font-weight: bold">Texto</span>
```

Outro componente interessante para saída de dados é o <h:outputFormat> que permite exibir na tela uma mistura de texto e variáveis por meio de formatadores entre chaves. Esse componente pertence a outra biblioteca JSF, portanto, para utilizarmos precisamos

adicionar a diretiva `xmlns:f=http://java.sun.com/jsf/core` dentro da tag <html>. O Quadro 1.6 ilustra a utilização desse componente. Veja na linha 12 que o atributo *value* recebe um texto mesclado com valores entre chaves. Em seguida nas linhas 13 e 14 a tag <f:param> permite determinar o conteúdo que será exibido no lugar das chaves, por ordem de especificação, ou seja, a linha 13 substituirá o parâmetro zero {0} e a linha 14 o parâmetro um {1}.

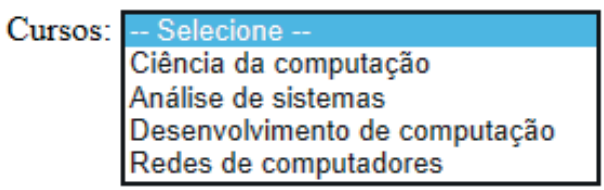
Quadro 1.6 | Usando o componente *Output Format*

```
1.  <!DOCTYPE html>
2.  <html
3.     xmlns="http://www.w3.org/1999/xhtml"
4.     xmlns:f="http://xmlns.jcp.org/jsf/core"
5.     xmlns:h="http://xmlns.jcp.org/jsf/html">
6.     <h:head>
7.       <title>OutputFormat</title>
8.     </h:head>
9.     <h:body>
10.      <h:form>
11.         <h:outputFormat
12.         value="Olá {0}! Curso de {1}.">
13.         <f:param value="#{minhaBean.nome}" />
14.         <f:param value="#{minhaBean.descricao}" />
15.         </h:outputFormat>
16.      </h:form>
17.    </h:body>
18. </html>
```

Fonte: elaborado pelo autor.

É comum ao preencher um formulário, um usuário se deparar com um campo de seleção como o da Figura 1.10.

Figura 1.10 | Componente de seleção



Fonte: elaborado pelo autor.

Esse componente pode ser criado pela tag <h:selectOneMenu> e as opções da lista podem ser compostas por itens estáticos e dinâmicos. Para criar itens estáticos usa-se a tag <f:selectItem> e parar criar as dinâmicas usa-se <f:selectItems>, conforme ilustra o código:

```
1. <h:selectOneMenu value="#{minhaBean.descricao}">
2.   <f:selectItem itemLabel="-- Selecione --"
   noSelectionOption="true"/>
3.   <f:selectItems value="#{cursosBean.cursos}"
   var="curso" itemValue="#{curso}"/>
4. </h:selectOneMenu>
```

A linha 1 indica que o valor escolhido será enviado a um método chamado setDescricao() da Manage Bean chamada minha Bean. Na linha 2, um item estático é adicionado à lista, o qual exibe uma informação para o usuário. O comando na linha 3 é responsável por popular a lista de forma dinâmica, com os dados vindo de uma Manage Bean chamada CursosBean. Foi atribuído o nome curso a uma variável para facilitar a utilização de outros atributos. O atributo *itemValue*, da seleção de itens, receberá o valor e passará para o atributo *value* do componente.

O componente <h:selectOneRadio> também pode ser usado para selecionar um item dentre vários, conforme exemplo:

```
<h:selectOneRadio value="#{minhaBean.descricao}">
    <f:selectItems  value="#{cursosBean.cursos}"
var="curso"
    itemValue="#{curso}" />
</h:selectOneRadio>
```

Outro componente para ser usado com opções é o <h:selectManyCheckbox> que permite selecionar várias opções e pode ser construído conforme o exemplo:

```
<h:selectManyCheckbox value="#{minhaBean.descricao}">
    <f:selectItems  value="#{cursosBean.cursos}"
var="curso"
    itemValue="#{curso}" />
</h:selectManyCheckbox>
```

Um componente muito interessante é o <h:selectBooleanCheckbox> que define atributos booleanos. Para criar esse tipo de componente a Manage Bean deve possuir

um campo booleano para receber o resultado. O primeiro passo consiste em criar o componente na interface gráfica, ou seja, no arquivo xhtml conforme código:

```
<h:selectBooleanCheckbox id="aceite"
value="#{minhaBean.termoAceito}"/>
<h:outputLabel value="Li e aceito os termos e
condições" for="aceite"/>
```

E na classe de controle implementar um atributo que o receba:

```
@Manage Bean
Public class MinhaBean{
    private boolean termoAceito;

    public boolean getTermoAceito() {
          return termoAceito;
    }
    public void setTermoAceito(boolean termoAceito) {
          this.termoAceito = termoAceito;
    }
}
```

Reflita

A utilização de diversos componentes pode enriquecer a criação de uma interface gráfica e melhorar a experiência do usuário, hoje conhecida por UX (*User Experience*). Como a utilização de componentes de forma estática ou dinâmica pode influenciar a experiência do usuário? Se usarmos somente componentes dinâmicos estaremos garantindo uma melhor experiência para ele?

Alguns componentes aceitam atributos que permitem trabalhar com scripts deixando a página ainda mais dinâmica. Veja o código:

```
<h:inputText onclick="this.value = '';"
    onchange="this.value=this.value.toUpperCase();"
    onmouseover="this.style.backgroundColor='yellow';"
    onmouseout="this.style.backgroundColor='white';"/>
</h:form>
```

Considerando o código apresentado, ao clicar sobre a caixa de texto em questão ela ficará em branco, ao sofrer uma mudança seu texto será colocado em letras maiúsculas, pelo método toUpperCase(), ao passar o mouse sobre, sua cor de fundo será alterada para amarelo e, ao tirar o mouse de cima, seu fundo ficará branco.

Além de inserirmos componentes, podemos organizar o *layout* de um formulário, de uma maneira simples, usando um componente do JSF chamado *panel grid*. Ele funcionará como uma espécie de tabela, na qual definiremos somente a quantidade de colunas, pois a quantidade de linhas será criada automaticamente conforme necessário. Observe o código no Quadro 1.7. Os campos do formulário foram criados dentro de um elemento <h:panelGrid> com duas colunas, isso significa que os rótulos (nome, idade, etc.) ficarão na coluna da esquerda e as caixas de texto, para digitação do usuário, ficarão na coluna da direita. Nesse caso serão criadas cinco linhas, pois ao completar uma coluna uma nova linha é aberta.

Quadro 1.7 | Usando *panel grid*

```
1.  <h:body>
2.    <h:form>
3.       <h:panelGrid columns="2">
4.          Nome: <h:inputText />
5.          Idade: <h:inputText />
6.          E-mail: <h:inputText />
7.          Telefone: <h:inputText />
8.          Observações: <h:inputTextarea />
9.       </h:panelGrid>
10.   </h:form>
11. </h:body>
```

Fonte: elaborado pelo autor.

O *panel grid* deve ser utilizado para fins de organização de *layout*, já para organizar dados em tabelas o objeto a ser usado é o *data table*. O conteúdo de cada coluna deve ser especificado manualmente, conforme o código do Quadro 1.8, ou populando a tabela, de forma dinâmica, com dados armazenados em um banco de dados conforme veremos mais adiante. Para adicionar uma identificação para cada coluna precisamos utilizar um novo

componente chamado de *facet*. Esse componente é diferente dos que vimos até o momento, pois após a requisição e a resposta do servidor não é gerado código HTML para ele. Sua função é modificar um componente e não criar, por esse motivo ele é alocado em outra *namespace*, e faz parte da biblioteca core do JSF. Para utilizarmos precisamos adicionar a diretiva `xmlns:f=`http://java.sun.com/jsf/core dentro da tag <html>.

Quadro 1.8 | Usando o *data table*

```
1.  <h:dataTable var="automovel" border="1">
2.     <h:column>
3.       <f:facet name="header">
4.         Cabeçalho coluna 1
5.       </f:facet>
6.         Conteúdo da coluna 1
7.     </h:column>
8.
9.     <h:column>
10.      <f:facet name="header">
11.        Cabeçalho coluna 2
12.      </f:facet>
13.        Conteúdo da coluna 2
14.    </h:column>
15.    <!-- outras colunas -->
16. </h:dataTable>
```

Fonte: elaborado pelo autor.

Outra forma de utilizar elementos dinâmicos dentro do contexto do JSF é a utilização do AJAX. Essa tecnologia permite atualizar partes da página, ao invés da página toda, diminuindo o tráfego de comunicação entre cliente e servidor, o que aumenta a performance do sistema. A partir da versão 2.0 do JSF já pode-se utilizar tags do estilo <f:ajax>. Ela permite a habilitação da tecnologia ajax em elementos do html, como um botão. Esses elementos do ajax são muito utilizados quando a atualização do valor de um elemento tem que refletir instantaneamente em outro elemento da interface. Esses elementos estão ligados à renderização de um elemento presente na interface, baseado nas respostas de um evento executado por elementos na página. Veja o exemplo:

```
1. <h:panelGroup id="painelAjax">
2. ...
3. </h:panelGroup>
4. <h:outputText id="nomeCliente" value="..."/>
5. <h:commandButton action="...">
6. <f:ajax execute="@form" render="painelAjax
   nomeCliente"/>
7. </h:commandButton>
```

Nesse exemplo, o comando ajax foi colocado dentro de um botão, ou seja, ao clicar nesse botão, serão atualizadas duas partes da página, o <h:panelGroup> e o <h:outputText>. Os elementos que serão atualizados são especificados, obrigatoriamente, pelo seu *id* no parâmetro *render* do ajax.

Pesquise mais

Um recurso muito interessante para o desenvolvimento de aplicações web é a possibilidade de renderizar somente partes da página. Esse recurso é chamado de AJAX. Leia o artigo a seguir e aprenda um pouco mais sobre recursos dinâmicos na web. COIMBRA, Everton. **Utilizando AJAX com Java Server Faces (JSF)**. Disponível em: <https://www.devmedia.com.br/utilizando-ajax-com-java-server-faces-jsf/24832>. Acesso em: 21 maio 2018.

Com isso fechamos a primeira parte do desenvolvimento usando o JSF. Agora que já sabemos criar as interfaces gráficas, vamos aprender a trabalhar com o banco de dados?

Continue seu estudo!

Sem medo de errar

Agora que você já conhece novos componentes, é hora continuar a implementar a solução para o setor de revisão mecânica. Foi solicitado a você criar o formulário de cadastro de serviço, no qual o usuário do sistema poderá escolher quais serviços serão realizados.

Abra o projeto criado na primeira etapa e crie uma nova classe java, dentro do pacote "controle", chamada *AutomovelBean*. Essa classe deverá conter dois vetores de strings, um contendo os tipos de veículos que a empresa aceita, nesse caso, somente automóvel

e moto e, outro contendo os serviços que são aceitos, nesse caso troca de pneus, troca de óleo, troca de pastilhas de freio, injeção eletrônica, reparos em motores, sistema elétrico e suspensão. Veja no Quadro 1.9 parte do código necessário.

Quadro 1.9 | Parte da classe *AutomovelBean*

```
1.  import java.util.ArrayList;
2.  import javax.faces.bean.ManagedBean;
3.  @ManagedBean
4.  public class AutomovelBean {

5.  private ArrayList<String> tipos;
6.  private ArrayList<String> servicos;
7.  private String tipo;
8.  private String servico;

9.       public AutomovelBean() {
10.           tipos = new ArrayList<String>();
11.           tipos.add("Automóvel");
12.           tipos.add("Moto");

13.           servicos = new ArrayList<String>();
14.           servicos.add("Troca de pneus");
15.           servicos.add("Troca de óleo");
16.           servicos.add("Troca de pastilhas de freio");
17.           servicos.add("Injeção eletrônica");
18.           servicos.add("Reparos em motores");
19.           servicos.add("Sistema elétrico");
20.           servicos.add("Suspensão");

21.      }
22.      public ArrayList<String> getTipos(){
23.           return tipos;
24.      }

25.      public ArrayList<String> getServicos(){
26.           return servicos;
27.      }
28. // implementar os get e set restantes
29. }
```

Fonte: elaborado pelo autor.

O próximo passo é inserir a referência do AutomovelBean dentro do arquivo faces-config.xml, conforme código:

```
<managed-bean>
    <managed-bean-name>automovelBean</managed-bean-name>
    <managed-bean-class>controle.AutomovelBean</managed-bean-class>
    <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

Por fim, crie o arquivo "cadastrarAutomovel.xhtml" para fazer a interface com o usuário. Nessa tela, o usuário deverá ter a opção de selecionar um dos dois tipos de automóvel, para isso use um <h:selectOneMenu>. Já os serviços devem ser exibidos de modo que o usuário possa selecionar mais de um, para isso use um componente <h:selectManyCheckbox>. Ambos componentes devem carregar, de forma dinâmica, os dados da *Manage Bean Automovel*. O Quadro 1.10 apresenta parte do código necessário para a criação da interface, complete com os demais.

Quadro 1.10 | Parte da classe *AutomovelBean*

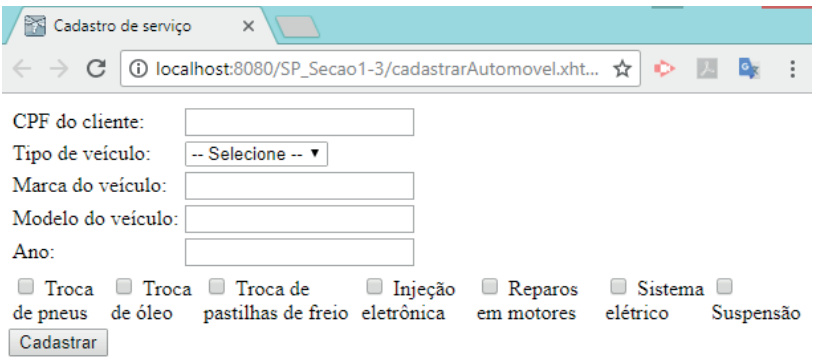
```
    <!— Completar o código -->
1.  <h:form>
2.     <h:panelGrid columns="2">
3.        CPF do cliente: <h:inputText />

4.        Tipo de veículo:
5.        <h:selectOneMenu value="#{automovelBean.tipo}">
6.        <f:selectItem itemLabel="-- Selecione --"
               noSelectionOption="true"/>
7.        <f:selectItems value="#{automovelBean.tipos}"
               var="tipo" itemValue="#{tipo}"/>
8.        </h:selectOneMenu>
9.        Marca do veículo: <h:inputText />

10.       Modelo do veículo: <h:inputText />
11.       Ano: <h:inputText />
12.    </h:panelGrid>
```

```
13.     <h:selectManyCheckbox
14.     value="#{automovelBean.servico}">
            <f:selectItems value="#{automovelBean.servicos}"
15.         var="servico" itemValue="#{servico}" class="opcao"
            />
16.     </h:selectManyCheckbox>
17.     <h:commandButton value="Cadastrar"
        action="cadastroLivros"  />
18. </h:form>
    <!— Completar o código -->
```

Fonte: elaborado pelo autor.

Veja se o resultado ficou similar ao da Figura 1.11. Aproveite a oportunidade e crie uma folha de estilos para melhorar a aparência da tela.

Figura 1.11 | Interface gráfica inicial para o cadastro de serviços



Fonte: elaborada pelo autor.

Avançando na prática

Seleção de curso para matrícula

Descrição da situação-problema

Você foi contratado para melhorar a experiência do usuário (UX) de um sistema de matrícula escolar. A atual versão do sistema permite que um aluno seja matriculado em mais de um curso, o que gera várias inconsistências. Seu trabalho é criar um componente na

interface gráfica que carregue todos os cursos disponíveis, mas que dê ao usuário do sistema a possibilidade de selecionar apenas uma opção. Qual recurso poderá ser usado nesse caso? É preciso criar uma Manage Bean? Implemente a solução conforme julgar mais adequado.

Resolução da situação-problema

Para resolver a questão de limitar a quantidade de opções selecionadas, o melhor recurso é utilizar um *<h:selectOneRadio>*. Se tratando de um objeto que terá seu conteúdo preenchido de forma dinâmica, você precisa criar uma Manage Bean para carregá-la. Os Quadros 1.11 e 1.12 trazem, respectivamente, os códigos necessários para incluir o objeto da Manage Bean e da interface gráfica (arquivo xhtml).

Quadro 1.11 | Código da Manage Bean "CursoBean"

```
1.  package controle;

2.  import java.util.ArrayList;
3.  import javax.faces.bean.ManagedBean;

4.  @ManagedBean
5.  public class CursoBean {
6.        private ArrayList<String> cursos = new
    ArrayList<String>();

7.        public CursoBean() {
8.            cursos.add("Ciência da computação");
9.            cursos.add("Análise e desenvolvimento de
    sistemas");
10.           cursos.add("Sistemas de informação");
11.           cursos.add("Segurança em rede");
12.           cursos.add("Engenharia da computação");
13.           cursos.add("Outro");
14.       }

15.       public ArrayList<String> getCursos() {
16.           return cursos;
17.       }
18.       public void setCursos(ArrayList<String> cursos) {
19.           this.cursos = cursos;
20.       }
21. }
```

Fonte: elaborado pelo autor.

Quadro 1.12 | Código do arquivo "matricula.xhtml"

```
1.  <!DOCTYPE html>
2.  <html
3.        xmlns="http://www.w3.org/1999/xhtml"
4.        xmlns:h="http://xmlns.jcp.org/jsf/html"
5.        xmlns:f="http://java.sun.com/jsf/core"
6.  >
7.  <h:head>
8.        <meta charset="UTF-8" />
9.        <title>Matrícula</title>
10. </h:head>
11. <h:body>
12.       <h:form>
13.           Escolha um curso para se matricular:
14.           <h:selectOneRadio
    value="#{minhaBean.descricao}">
15.                 <f:selectItems
    value="#{cursoBean.cursos}" var="curso"
    itemValue="#{curso}" />
16.           </h:selectOneRadio>
17.       </h:form>
18.       <h:commandButton
    value="Selecionar"></h:commandButton>
19. </h:body>
20. </html>
```

Fonte: elaborado pelo autor.

Faça valer a pena

1. Um desenvolvedor *front end* (aquele responsável por atuar em elementos da camada de visão) descreveu um problema sobre a falta de recursos tecnológicos para resolver problemas de atualização na tela de valores ligados a lista de beneficiários de um seguro. O sistema faz o cadastro de uma pessoa e liga a ela seus beneficiários em caso de morte. A tela de fornecimento desse benefício que está com problemas, pois ele não consegue recuperar de forma dinâmica, em um campo de seleção, os beneficiários ligados a esse contexto. Ao pedir ajuda a três colegas, também desenvolvedores, ele obteve as seguintes respostas:

Desenvolvedor I - Para resolver esse problema, basta você criar um elemento do tipo BeneficiarioManagedBean para realizar a captura dessas informações e mandar para a tela.

Desenvolvedor II - Para resolver o seu problema basta você utilizar recursos do javascript em conjunto com o JSF para recuperar informações no banco de dados, subsidiada por classe do tipo ManagedBeans que seu problema estará resolvido.
Desenvolvedor III - Basta utilizar recursos do JSF, ou seja, componentes dinâmicos junto com Manage Bean para capturar as informações vindas de camadas de controle e modelo. Nesse caso, seu problema será resolvido.

Sobre as respostas dos desenvolvedores e seus conhecimentos sobre o assunto, marque a alternativa que apresenta somente desenvolvedores que apresentaram uma resposta adequada para a resolução do problema.

a) Desenvolvedor I somente.
b) Desenvolvedor II somente.
c) Desenvolvedores I e III somente.
d) Desenvolvedor III somente.
e) Desenvolvedores II e III somente.

2. A utilização do JSF traz benefícios de forma a melhorar o desenvolvimento de um software e apresentar recursos modernos na camada de visão A utilização de frameworks oriundos do JSF facilita a forma com que você pode trabalhar com recursos do html. Considere as seguintes tags apresentadas a seguir:

I. <h:selectOneMenu>
II. <h:selectOneRadio>
III. <h:selectBooleanCheckbox>

Quais das tags apresentadas permite a seleção de apenas uma opção?

a) Somente a tag I.
b) Somente as tags I e II.
c) Somente as tags II e III.
d) As tags I, II e III permitem a seleção de somente um item.
e) Apenas a tag II.

3. Uma interface gráfica de solução WEB utiliza recursos de forma a organizar e facilitar a visualização de uma página de maneira adequada para os seus usuários. A utilização desse tipo de recurso requer conhecimento prévio do desenvolvedor, pesquisas sofre formas de atuação e de implementação desses recursos e seus impactos na interface produzida.

Sobre a utilização de elementos estáticos e dinâmicos no JSF para inserção em uma página da WEB, marque a alternativa correta.

a) Os elementos estáticos têm o mesmo comportamento que os elementos dinâmicos na interface. Sua diferenciação é de qual biblioteca do JSF que eles são originados, pois elas interferem na forma de atuação do elemento.
b) Os elementos estáticos e dinâmicos são usados para estruturar e organizar o conteúdo de uma página WEB. Os elementos de cunho dinâmico podem ser usados para diminuir a probabilidade de erros no sistema, o que melhora a experiência do usuário na utilização do sistema.
c) Os elementos estáticos do JSF em uma página representam situações contornadas e renderizadas por bibliotecas do ajax, da mesma forma que elementos dinâmicos apresentam características ligadas a forma, cor e contextualização dentro da interface.
d) As interfaces podem utilizar elementos de natureza estática somente quando os elementos de natureza dinâmica não estão presentes, pois a medida que os elementos estáticos cuidam de conteúdo, organização e apresentação de informações, os elementos de natureza dinâmica como o ajax e o javascript atuam de forma a melhorar as respostas dos eventos da página.
e) Os elementos estáticos estão representados por elementos de texto, botões, validadores e renderização de campos digitáveis. Já os elementos de natureza dinâmica apresentam comportamento de grande mudança na interface, principalmente por tratar de forma assíncrona (não no mesmo tempo) os eventos realizados pelo usuário na interface.

Referências

CORDEIRO, G. **Aplicações Java para web com JSF e JPA**. São Paulo: Casa do Código, 2014.

HORSTMANN, C.; GEARY, D. **Core Javaserver Faces**. São Paulo: Alta Books, 2012.

FARIA, T. **Java EE 7 com JSF, PrimeFaces e CDI**. São Paulo. AlgaWorks Softwares, 2015.

Desenvolvimento de aplicações em JBF com persistência

Convite ao estudo

Prezado aluno, daremos continuidade aos nossos estudos em Programação para Web II. Relembrando: você aprendeu na unidade anterior sobre Programação em *Java Server Faces* (JSF), o padrão *Model-View-Controller* (MVC) no desenvolvimento web e a instalação e configuração do framework para a programação de páginas web em Java.

Agora, para avançarmos um pouco mais nesse conhecimento, você aprenderá sobre o desenvolvimento de aplicações em JSF com persistência e verá como utilizar a *Java Persistence API* (JPA), como trabalhar com o *framework* Hibernate implementando um CRUD, permitindo assim, que seus sistemas e informações sejam mantidas após a finalização do programa. Ao final desta unidade você vai conhecer e compreender a persistência de dados usando o *framework* Hibernate e ser capaz de utilizá-lo para desenvolver um aplicativo web com persistência de dados.

Você foi designado pela empresa de TI na qual trabalha para uma consultoria em uma companhia no ramo imobiliário que está sendo aberta. O gerente desta nova empresa solicitou uma consultoria para o desenvolvimento de um novo sistema para o controle de clientes e de locações e venda de imóveis, que mantenha as informações cadastradas no sistema para que possam ser consultadas a qualquer momento e para armazenamento de histórico. Como esta nova empresa é uma cliente muito importante e seu gestor conhece sua competência profissional em desenvolver soluções para

manter a persistência de informações, ele pede sua ajuda no desenvolvimento deste sistema.

Em seu primeiro desafio, será necessário implementar algumas das classes do sistema, como a classe para cadastro de imóveis, com os campos de referência, descrição, tipo, localização e valor e também para o cadastro de clientes da imobiliária, com os campos de nome, CPF, endereço e cidade, inserindo nessas classes algumas notações JPA das informações. Continuando o desafio, você deverá criar o arquivo de persistência das informações e realizar a configuração do *framework* Hibernate no sistema imobiliário que está desenvolvendo. Para concluir o trabalho, você deverá criar as classes de CRUD no banco com base nas classes criadas e a configuração do *framework* Hibernate utilizada na aplicação.

Com o uso de um *framework* (no nosso caso, o Hibernate) como é possível ter um desempenho maior no desenvolvimento de um sistema? Quais as vantagens da utilização deste *framework*? Agora que você conhece seu desafio, que tal realizar um ótimo trabalho para a imobiliária?

Então, primeiramente, vamos alimentar nossos estudos desenvolvendo os assuntos desta unidade para sermos capazes de solucionar as situações provenientes deste desafio. Mãos à obra!

Seção 2.1

Utilização do *java persistence API* (JPA)

Diálogo aberto

Caro aluno, você consegue pensar no trabalho interno de uma empresa, com todas as vendas de produtos realizadas, notas fiscais emitidas e recebidas, cadastro de clientes, fornecedores e colaboradores etc. e, desta forma, todas as informações utilizadas no trabalho do dia a dia desta empresa? Você consegue imaginar todas as informações de um dia de trabalho sendo geradas e, no dia seguinte, tudo o que foi feito no dia anterior se perdendo e começando novamente dia após dia? Sem a persistência de informações, não haveria histórico de dados e de movimentações e, para complicar mais, todos os dias seria necessário cadastrar todos os produtos e clientes para a venda ser realizada, algo inaceitável, não acha? Por isso, nessa seção você começará a aprender sobre o processo de persistência de dados em sistemas web usando o JPA.

Trabalhando em uma empresa de TI, seu gestor o designou para uma importante tarefa em uma companhia do ramo imobiliário e você terá que ajudá-lo, criando uma solução para seu sistema imobiliário. É muito importante lembrar que você precisará realizar a persistência das informações para que a empresa possa efetuar consultas aos dados cadastrados no sistema, como dados de imóveis e dados de clientes, além do histórico de locações e venda dos imóveis já realizados. Neste primeiro momento, será necessário implementar algumas das classes do sistema, como a classe para cadastro de imóveis, com os campos de referência, descrição, tipo, localização e valor, assim como o cadastro de clientes da imobiliária, com os campos de nome, CPF, endereço e cidade. Além disso, você deverá inicializar a configuração do sistema com as especificações do JPA e, então, apresentar as classes criadas à empresa imobiliária em que será realizado a persistência de informações no sistema.

Nesta seção focaremos no desenvolvimento de habilidades em programação para web utilizando a persistência de informações no desenvolvimento de aplicações em JSF. Você vai aprender sobre a

definição de mapeamento objeto-relacional, sobre a introdução ao JPA, as diferenças existentes entre o *Java Database Connectivity* (JDBC) e JPA e as notações utilizadas pelo JPA em sua estrutura. O uso de persistências solucionará problemas em que é necessária a gravação de informações que precisam posteriormente ser recuperadas para a utilização nos sistemas.

Não se esqueça de ler atentamente as dicas nos itens pedagógicos e estudar os assuntos desta seção. Faça as atividades, crie o relatório com base em seu desafio e busque se aprofundar nos conhecimentos que serão importantes para sua vida profissional.

Vamos começar?

Não pode faltar

Caro aluno, quando falamos sobre persistência em desenvolvimento de aplicações, estamos falando necessariamente em manter as informações inseridas no sistema por um período desejável ou de forma definitiva para que possa ser consultado ou utilizado posteriormente.

Segundo Faria e Normandes Jr. (2015), a persistência nas informações é a base do conceito para o desenvolvimento de aplicações em empresas. Assim, a grande maioria dos sistemas desenvolvidos para o uso dentro da empresa precisa de dados persistentes.

Podemos observar esta situação no nosso cotidiano, no uso de computadores, seja em casa, no trabalho ou na faculdade. Vamos supor que você esteja redigindo um texto no Microsoft Word. Ao abrir o programa e começar a digitar, o texto é inserido no Word, mas como o documento ainda não foi salvo, ele fica armazenado na memória do computador. De forma repentina, a energia acaba e, ao retornar, o texto que você digitou se perdeu, pois ele ainda não havia sido salvo. Desta forma, como o documento não foi salvo, não foi gerada a persistência deste texto dentro do HD do computador. Caso tivesse sido salvo antes de a energia acabar, o texto já estaria disponível para o uso ao religar o dispositivo.

Sem persistência em um sistema de informação quando ele fosse encerrado, o software não teria tanta utilidade e seu uso não seria prático. De acordo com Faria e Normandes Jr. (2015), normalmente

o uso de persistência de dados em programação requer sistemas gerenciadores de banco de dados relacionais e SQL (*Structured Query Language*, ou Linguagem de Consulta Estruturada), porém, diversas outras formas de persistência de dados, como arquivos de texto e XML, são utilizadas como alternativas.

Definição de mapeamento objeto-relacional

A linguagem de programação Java pertence ao paradigma da programação orientada a objetos. Desta forma, para realizarmos a persistência de dados entre banco de dados relacionais e linguagens de programação orientada a objetos, podemos utilizar uma técnica conhecida como Mapeamento de Objeto Relacional, também conhecido como Mapeamento Objeto-Relacional ou simplesmente ORM (*Object Relational Mapping*).

A definição sobre ORM dada por Coelho (2015) é:

> ***Object Relational Mapping*** **(ORM) quer dizer basicamente: transformar as informações de um banco de dados, que estão no modelo relacional para classes Java, no paradigma Orientado a Objetos de um modo fácil. (Coelho, 2013, p. 3)**

A metodologia ORM tem por objetivo estreitar as diferenças entre o modelo entidade-relacional (MER) e a programação orientada a objetos, criando um mapeamento entre a modelagem relacional e a modelagem orientada a objetos.

No desenvolvimento de sistemas utilizando a orientação a objetos, quando os objetos são persistidos em um banco de dados relacional é necessário criar dois modelos: o do banco e o das classes. Usa-se o termo mapeamento ao se tratar do *framework*, porque os objetos são mapeados com as tabelas, algo que, se feito de maneira displicente, pode causar uma estrutura de domínio fraca e deficiente, devido à separação dos dados em relação às regras de negócio.

Em banco de dados, as entidades são representadas por tabelas compostas por colunas que armazenam dados de tipos diversos. Por outro lado, nas linguagens de programação orientadas a objetos, as entidades são representadas por classes, cujos objetos representam

os elementos do mundo real. Assim, podemos comparar por meio do Quadro 2.1 o modelo relacional de um banco de dados com o modelo de orientação a objeto da linguagem Java.

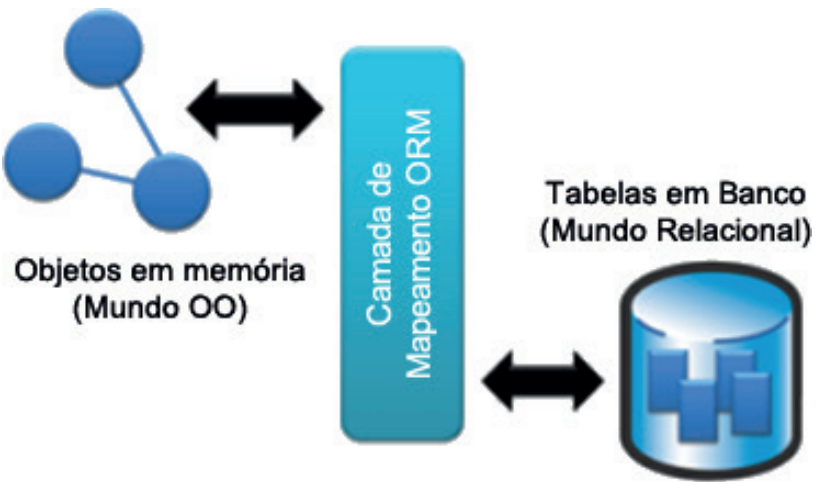
Quadro 2.1 | Comparação entre modelo relacional versus modelo orientado a objeto

Modelo Relacional	Modelo Orientado a Objeto
Tabela	Classe
Linha	Objeto
Coluna	Atributo
-	Método
Chave Estrangeira	Associação

Fonte: adaptado de Faria e Normandes Jr. (2015).

Conforme Faria e Normandes Jr. (2015), o mapeamento é realizado por uma camada intermediária que converte os objetos na memória em tabelas no banco de dados relacional, conforme ilustra a Figura 2.1. Esse mapeamento pode ser feito utilizando diversos *frameworks*, por exemplo, o Hibernate, o Entity Framework, o ObjectMapper, entre outros.

Figura 2.1 | Técnica ORM



Fonte: <https://www.devmedia.com.br/orm-object-relational-mapper/19056>. Acesso em: 5 jul. 2018.

Quando há a aplicação da técnica de ORM, o sistema se comunica com uma camada intermediária chamada API (*Application Programming Interface* ou Interface de Programação de Aplicativos). Neste momento, as classes e os códigos SQL são abstraídos e gerenciados pela API, disparando os comandos SQL a partir dos dados recebidos.

A implementação da ORM pode apresentar algumas vantagens, como, segundo Faria e Normandes Jr. (2015):

- Facilitação na criação de código de acesso ao banco de dados.
- Excelente manutenibilidade de sistemas.
- Fácil entendimento.
- Custo de esforço menor para implementação.
- Criação de abstração da sua aplicação do banco de dados, permitindo a mesma conexão do sistema para bancos diferentes.

Introdução ao *Java Persistence API* (JPA)

A metodologia ORM foi ganhando espaço e muitos *frameworks* foram surgindo no mercado, criando um novo problema: a incompatibilidade da "linguagem" falada por eles. Nesse cenário, um grupo de experts em software, conhecidos por EJB 3.0, criaram um modelo, um conjunto de regras para a persistência em java usando ORM, chamado JPA. Com essa especificação criou-se uma padronização para o uso, resolvendo diversos problemas.

Segundo Faria e Normandes Jr. (2015), o JPA é um padrão para criação de persistência em Java, permitindo uma API de mapeamento objeto-relacional e soluções de integração de persistências em aplicações escaláveis.

Na utilização de JPA, os objetos não precisam de configurações especiais para se tornarem persistentes, bastando apenas adicionar nas classes que representam as entidades do sistema, algumas anotações específicas. Conforme Coelho (2013), a JPA é um conjunto de regras, normas e interfaces que definem o comportamento, conhecido também como especificação.

Conforme Faria e Normandes Jr. (2015), JPA é uma especificação, ou seja, um conjunto de regras, e não um produto que será integrado ao sistema. Para utilizar o JPA em um projeto,

é necessário implementá-lo por meio de um arquivo XML com as regras de conexão e persistência com o banco.

Referente à utilização do JPA, Coelho (2013) define que:

> **toda especificação precisa de uma implementação para que possa ser usada num projeto. No caso da JPA, a implementação mais famosa e utilizada no mercado é o Hibernate, mas existem mais no mercado como OpenJPA, EclipseLink, Batoo e outras. Cada implementação tem suas vantagens e desvantagens na hora da utilização. (COELHO, 2013, p. 5)**

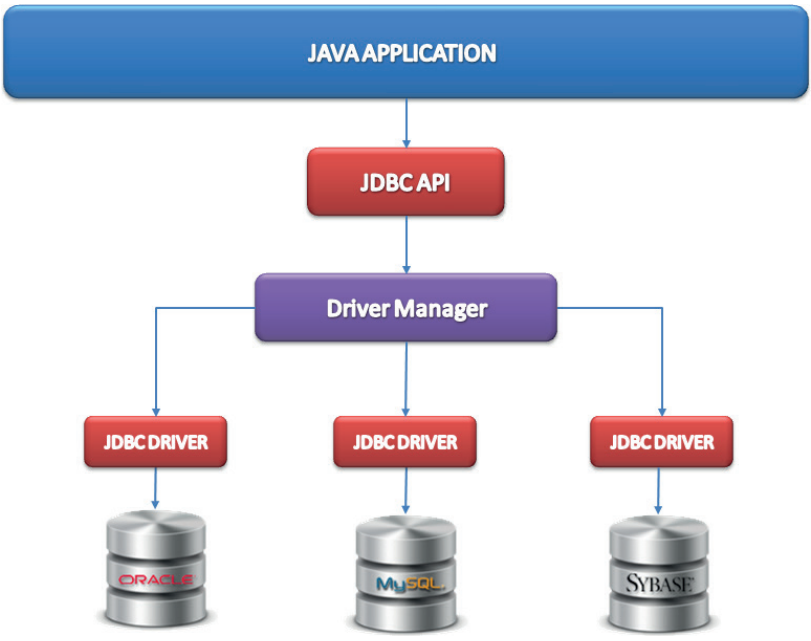
Para configurar as informações referentes do JPA no banco de dados, utilizamos a unidade de persistência. Por meio desta configuração, podemos identificar todas as classes que terão seu mapeamento realizado como entidades do banco de dados.

Assimile

A persistência é utilizada para configurar as informações referentes à implementação da especificação JPA e ao banco de dados, além de podermos identificar as classes que serão mapeadas como entidades do banco de dados.

Antes de existir a persistência de dados através do modelo ORM, usava-se o JDBC (*Java Database Connectivity*) para tal tarefa. O JDBC é uma API, ou seja, um conjunto de classes e interfaces do Java para realizar a comunicação e as transações com o banco de dados. Para cada SGBD (sistema gerenciador de banco de dados) era usado um *driver* específico para efetuar a comunicação, conforme ilustra a Figura 2.2. Através das classes do JDBC escreviam-se comandos SQL para efetuar as operações no banco.

Figura 2.2 | Esquema de utilização do JDBC



Fonte: <https://avaldes.com/connecting-to-oracle-using-jdbc/>. Acesso em: 5 jul. 2018.

Com o surgimento do ORM e posteriormente do JPA, esse processo foi simplificado: agora não é mais necessário escrever comandos SQL na implementação da solução. Basta configurar o JPA no arquivo XML e utilizar algumas notações específicas nas classes que se deseja persistir, conforme você aprenderá no decorrer dessa unidade.

Segundo Coelho (2013), outra vantagem da utilização da ORM é a possibilidade de portabilidade do sistema entre banco de dados, os tratamentos de acesso, acesso às funções, ou seja, é possível alterar ou integrar diferentes bancos de dados ao sistema.

Reflita

Você aprendeu que uma das vantagens de ter um projeto utilizando apenas notações da JPA é que automaticamente ela se torna portável para outra implementação, como a troca do Hibernate pelo EclipseLink. Sabendo-se que o JPA provê esta vantagem, podemos considerar que

seja comum precisarmos realizar a troca de implementação? Por qual motivo esta troca é uma vantagem?

Com o surgimento e a padronização do JPA, os *frameworks* passaram a ser inseridos de maneira diferente dentro do projeto. Por exemplo, para usar o *framework* Hibernate antes da especificação JPA, criava-se um arquivo chamado *hibernate.cfg.xml* para fazer as configurações. Com a especificação para qualquer *framework*, cria-se o arquivo *persistence.xml*, que deve obrigatoriamente ser criado dentro da pasta *META-INF*.

Exemplificando

Veja um exemplo do início da criação de um arquivo *persistence.xml* no Quadro 2.2.

Quadro 2.2 | Parte do arquivo persistence.xml

```
<persistence
xmlns="http://java.sun.com/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/
persistence http://java.sun.com/xml/ns/persistence/
persistence_2_0.xsd"
version="2.0">

    <persistence-unit
name="org.hibernate.tutorial.jpa" transaction-
type="RESOURCE_LOCAL">

       <description>
              Unidade de persistência
       </description>
       <provider>
              org.hibernate.ejb.
HibernatePersistence
       </provider>

       <properties>

       </properties>
```

```
</persistence-unit>
</persistence>
```

Fonte: elaborado pelo autor.

O exemplo no Quadro 2.2 exibe parte do código necessário para configurar o ORM, segundo as especificações do JPA. A configuração inicia com a tag <*persistence*> na linha 1, que possui quatro parâmetros:

1) `xmlns="http://java.sun.com/xml/ns/persistence"`. Esse atributo especifica qual *namespace* os esquemas da JPA vão compartilhar. O valor atribuído é referente à especificação 2.0 da API e, para acessar a versão 2.1, deve ser usado `xmlns="http://xmlns.jcp.org/xml/ns/persistence/"` (ORACLE, 2018).

2) `xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance`. Todo documento XML precisa estar relacionado a um esquema XML, pois esse atributo faz a instância de um determinado esquema dentro de um *namespace* que será especificado no próximo atributo (*xsi:schemaLocation*).

3) `xsi:schemaLocation="http://java.sun.com/xml/ns/persistence http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd"`. Esse atributo é composto por dois valores separados por um espaço. O primeiro valor especifica qual *namespace* será usado para o esquema do XML. O segundo são o local e o arquivo a serem usados, nesse caso, será o arquivo *persistence_2_0.xsd*. Perceba como esse atributo está relacionado ao primeiro, pois caso opte por usar a versão 2.1 aqui, deverá utilizar o arquivo *persistence_2_1.xsd*. Outro ponto interessante é que você poderá fazer o download desses esquemas no endereço disponível em: <http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html>. Acesso em: 5 jul. 2018.

4) `version="2.0"`. Esse atributo é mais simples, pois especifica a versão da API que será usada.

O próximo passo é definir uma unidade de persistência através da tag <*persistence-unit*>. Cada arquivo deve fornecer um nome único para a persistência. Isso é feito através do atributo *name* na

tag (name="*org.hibernate.tutorial.jpa*"). O valor pode ser escolhido por você, mas deverá ser um significativo, pois será usado depois em uma *Manager Factory*.

Já o atributo type="*RESOURCE_LOCAL*" especifica algumas características do mapeamento. Em um ambiente Java EE deve ser usado o valor "JTA", e em um ambiente Java SE deve ser usado o valor "RESOURCE_LOCAL".

A tag <*provider*> especifica um provedor que é um nome de classe totalmente qualificado do provedor de Persistência do EJB. Você não precisa defini-lo se não trabalhar com várias implementações do EJB3.

Dentro das tags <*properties*> </*properties*> serão especificadas as propriedades para a conexão com o banco de dados, como qual sistema gerenciador, qual o nome do banco, etc. Esses parâmetros serão vistos na próxima seção, na qual falaremos especificamente do Hibernate.

Assimile

Conforme Coelho (2015), é utilizado um arquivo XML para definir a unidade de persistência, chamado de *persistence.xml*, que deverá ser criado dentro da pasta *META-INF* de cada projeto.

As configurações feitas nesse arquivo vão refletir nas próximas implementações. Por exemplo, a unidade de persistência recebeu o nome de "org.hibernate.tutorial.jpa", então veja como será usado em um método Java no Quadro 2.3.

Quadro 2.3 | Utilização de método Java

```
protected void setUp() throws Exception {
    sessionFactory = Persistence.createEntityManagerFactory( "org.
hibernate.tutorial.jpa" );
}
```

Fonte: elaborado pelo autor.

A *sessionFactory* é usada para criar uma seção com o banco de dados. Para ajudar no entendimento, podemos fazer uma

analogia entre a seção e a conexão via JDBC. Veja que, para essa seção (conexão), será usada a unidade de persistência com o nome "org.hibernate.tutorial.jpa", pois foi o nome configurado no arquivo *persistence.xml*.

Pesquise mais

Todo programador deve acompanhar a documentação oficial das ferramentas e tecnologias. Aprenda mais sobre a especificação JPA acessando o site da Oracle: <http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html>. Acesso em: 5 jul. 2018.

O site <http://docs.jboss.org/hibernate/>, acesso em: 5 jul. 2018, oferece uma completa documentação a respeito das especificações do JPA e do Hibernate.

O site <http://tomee.apache.org/jpa-concepts.html>, acesso em: 5 jul. 2018, oferece uma descrição sobre o JPA.

Conforme Coelho (2013), a implementação da JPA precisa satisfazer todas as interfaces determinadas na especificação. Há casos em que uma implementação possa ter sido feita em 80% de todas as implementações de interfaces e, ainda assim, permitir lançar uma versão para utilização, ficando a critério de cada implementação garantir seu funcionamento correto.

Sem medo de errar

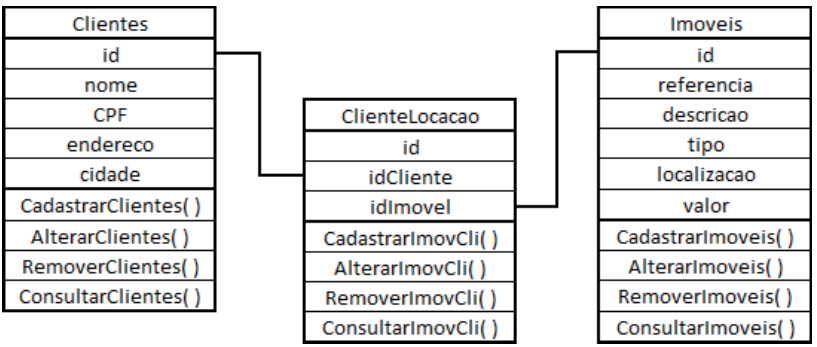
Como funcionário em uma empresa da área de TI, você foi designado pelo seu gestor para desenvolver o sistema de uma companhia do ramo imobiliário.

É muito importante lembrar que você precisará realizar a persistência das informações, para que a empresa possa efetuar consultas aos dados cadastrados no sistema, como dados de imóveis e de clientes, além de histórico de locações e vendas já realizadas dos imóveis.

Neste primeiro momento, será necessário implementar algumas das classes do sistema, como a classe para cadastro de imóveis, com os campos de referência, descrição, tipo, localização e valor e para cadastro de clientes da imobiliária, com os campos de nome, CPF, endereço e cidade. Além disso, você deverá inicializar a configuração do sistema com as especificações do JPA.

Para resolução deste desafio, temos como base fundamental o diagrama abaixo (Figura 2.3) que representa a estrutura do banco de dados do sistema imobiliário. Você deve implementar as classes na programação orientada a objetos.

Figura 2.3 | Diagrama que representa a estrutura do banco de dados do sistema imobiliário



Fonte: elaborada pelo autor.

Após a implementação das classes, você deverá criar o arquivo *persistence.xml* dentro da pasta META-INF, configurando as tags: <*persistence*> e <*persistence-unit*>, <*description*>. Não se esqueça de já deixar a tag preparada para receber as propriedades da conexão com o banco de dados.

Avançando na prática

Especificações JPA

Descrição da situação-problema

Você trabalha em uma empresa de desenvolvimento de sistemas e sua equipe está iniciando um novo projeto para um cliente que

deseja aprimorar um sistema de monitoramento de câmeras de segurança. O atual sistema já faz a persistência usando o *framework* Hibernate, por isso lhe foi entregue o arquivo *hibernate.cfg.xml* que faz a configuração com o banco, descrito no Quadro 2.4.

Quadro 2.4 | Arquivo *hibernate.cfg.xml*

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
        "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
        <!-- Database connection settings -->
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</
property>
        <property name="connection.url">jdbc:hsqldb:hsql://localhost</
property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <!-- JDBC connection pool (use the built-in) -->
        <property name="connection.pool_size">1</property>
        <!-- SQL dialect -->
        <property name="dialect">org.hibernate.dialect.HSQLDialect</
property>
        <!-- Enable Hibernate's automatic session context management -->
        <property name="current_session_context_class">thread</
property>
        <!-- Disable the second-level cache  -->
        <property name="cache.provider_class">org.hibernate.cache.
internal.NoCacheProvider</property>
        <!-- Echo all executed SQL to stdout -->
        <property name="show_sql">true</property>
        <!-- Drop and re-create the database schema on startup -->
        <property name="hbm2ddl.auto">update</property>
        <mapping resource="org/hibernate/tutorial/domain/Event.hbm.
xml"/>
    </session-factory>
</hibernate-configuration>
```

Fonte: elaborado pelo autor.

Seu chefe informou que o sistema vai usar as especificações JPA para fazer a persistência e pediu para você identificar no arquivo *hibernate.cfg.xml* o que poderá ser aproveitado. Você deve criar um arquivo com as linhas que poderão ser utilizadas na especificação JPA.

Resolução da situação-problema

Após uma pesquisa, você verificou que, embora a especificação JPA tenha determinado que a configuração deve ser feita em um arquivo chamado persistence.xml, as propriedades podem ser aproveitadas. Portanto, as linhas que poderão ser usadas no projeto estão no Quadro 2.5.

Quadro 2.5 | Linhas que serão utilizadas no projeto

```
        <property name="connection.driver_class">org.hsqldb.jdbcDriver</
property>
        <property name="connection.url">jdbc:hsqldb:hsql://localhost</
property>
        <property name="connection.username">sa</property>
        <property name="connection.password"></property>
        <property name="connection.pool_size">1</property>
        <property name="dialect">org.hibernate.dialect.HSQLDialect</
property>
        <property name="current_session_context_class">thread</
property>
        <property name="cache.provider_class">org.hibernate.cache.
internal.NoCacheProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">update</property>
```

Fonte: elaborado pelo autor.

Faça valer a pena

1. O uso de persistência de dados em programação Java requer sistemas gerenciadores de banco de dados relacionais e SQL (*Structured Query Language* ou Linguagem de Consulta Estruturada), porém, diversas outras formas de persistência de dados são utilizadas como alternativas.

Com base nesta afirmação, podemos considerar como estruturas alternativas para a persistência de dados:

a) Banco de dados não relacional apenas.
b) Arquivos texto e arquivos XML.
c) Arquivos XML apenas.
d) Banco de dados não relacional e arquivos texto.
e) Banco de dados não relacional e arquivos XML.

2. A aplicação da técnica de ORM consiste no sistema de se comunicar com uma camada intermediária chamada API (*Application Programming Interface* ou Interface de Programação de Aplicativos). Neste momento, as classes e os códigos SQL são abstraídos e gerenciados pela API, disparando os comandos SQL a partir dos dados recebidos.

Analise as afirmativas abaixo:
I- Facilitação na criação de código fonte do sistema.
II- Excelente manutenibilidade de sistemas.
III- Fácil aplicação, porém com complexo entendimento.
IV- Custo de esforço menor para implementação.
V- Criação de abstração da sua aplicação do banco de dados, permitindo a mesma conexão do sistema para bancos diferentes.

Assinale a alternativa que apresenta as sentenças corretas em relação às vantagens de implementação da ORM na programação do sistema:

a) I, II e III apenas.
b) I, III e IV apenas
c) II, III e V apenas.
d) II, IV e V apenas.
e) I, IV e V apenas.

3. Dada a afirmação a seguir:
Para configurar as _____ referentes do JPA no _____ , utilizamos a unidade de persistência e, por meio desta configuração, podemos identificar todas as _____ que terão seu mapeamento realizado como _____ do banco de dados.

Assinale a alternativa que apresenta respectivamente as palavras que completam a sentença apresentada.

a) classes, sistema, anotações e ORM.
b) informações, sistema, anotações e entidades.
c) classes, banco de dados, classes e ORM.
d) informações, banco de dados, anotações e ORM.
e) informações, banco de dados, classes e entidades.

Seção 2.2

Framework hibernate

Diálogo aberto

Caro estudante, imagine o dia a dia de um escritório contábil. Todos os dias são lançados dados de empresas clientes para geração de demonstrativos fiscais, por exemplo. Isso só é possível devido à persistência de dados de todas as informações lançadas no sistema e pelo fato de elas serem armazenadas em banco de dados. Agora, e se não houvesse a persistência de dados? Seria necessário que todos os dias fossem lançadas todas as informações dos dias anteriores novamente, pois os dados não ficariam gravados em lugar nenhum, gerando, assim, contínuas repetições de dados lançados.

Você está trabalhando em uma companhia de TI que foi contratada por um cliente que está abrindo uma empresa do ramo imobiliário. Ela necessita de um sistema que permita a realização de consultas a dados cadastrais e também a históricos de locação e venda de imóveis. Você já implementou as principais classes desse sistema, agora, como desafio desta etapa, precisará atualizar o arquivo de persistências XML e realizar a configuração do *framework* Hibernate no sistema imobiliário que está desenvolvendo juntamente ao seu cliente. Além disso, você deverá inserir nas classes já criadas as anotações para a persistência.

Nesta seção, você vai aprender e conhecer sobre o *framework* Hibernate, sua definição e arquitetura, sobre os objetos na camada de persistência e como configurar o Hibernate em sua aplicação. Com o uso deste *framework*, será possível criar sistemas que permitam a fácil organização de informações, possibilitando, desta maneira, um modelo de mapeamento relacional com o banco de dados de forma mais simples. Pronto para solucionar este desafio?

Não pode faltar

Prezado aluno, a linguagem de programação orientada a objetos pode gerar problemas para os programadores no processo de

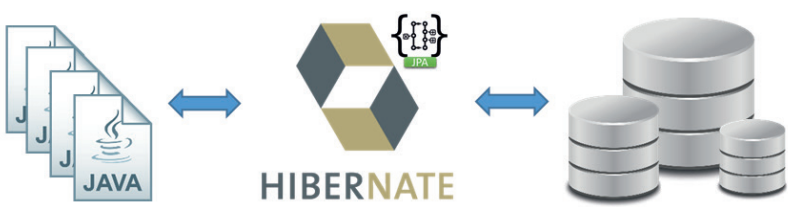
mapeamento de objetos junto ao banco de dados, devido à grande quantidade de bancos de dados existentes no mercado trabalhando com o modelo relacional. Desta forma, ao utilizarem o modelo relacional de banco com a programação orientada a objetos, os programadores precisam criar métodos para converter dados em objetos e objetos em dados, o que acaba desviando o foco principal do desenvolvimento final do sistema. Uma das soluções existentes no mercado para facilitar essa conversão é o *framework* Hibernate.

A utilização do *Java Persistence API* (JPA) em nossos projetos permite padronizar a forma como será realizada a persistência de dados no sistema. Já o Hibernate é a efetivação desta especificação, ou seja, é a implementação da especificação criada pelo JPA.

Segundo Cordeiro (2014), o Hibernate é um *framework* para realização do mapeamento objeto-relacional (ORM) para aplicações Java, ou seja, é uma ferramenta para mapear as classes Java em tabelas do banco de dados relacional e vice-versa. É o mais conhecido no mercado e sua principal função é abstrair o mapeamento, gerando economia de esforço e uma menor preocupação referentes a esta atividade. Portanto, usando essa ferramenta o programador não precisa escrever uma linha sequer de código SQL, tudo é feito pelo *framework*.

O Hibernate possui uma estrutura com suporte ao mapeamento de associações entre os objetos, coleções, composição, polimorfismo e herança. A Figura 2.4 ilustra a relação entre o sistema e o banco de dados. Veja que ele atua como uma camada intermediária para fazer a abstração.

Figura 2.4 | Representação do uso do *framework* Hibernate



Fonte: <https://www.devmedia.com.br/guia/hibernate/38312>. Acesso em: 5 jul. 2018.

Conforme Cordeiro (2014), o Hibernate não tem como função apenas realizar o mapeamento das classes Java para tabelas do banco de dados, ele também permite que consultas e retorno de dados da consulta sejam realizados com mais facilidade, podendo reduzir de maneira significativa o tempo de desenvolvimento, que antes era gasto pelas operações manuais dos dados feitas com SQL e JDBC.

Pesquise mais

Para aprofundar o conhecimento no *framework* Hibernate, assista ao vídeo *[Programação Web V2] 64 – Introdução ao Hibernate*, que apresenta a definição, os conceitos e como criar as estruturas de pastas para utilizar as bibliotecas em um projeto.

DELFINO, S. R. **[Programação Web V2] 64 – Introdução ao Hibernate**. 10 ago. 2014. Disponível em: <https://www.youtube.com/\watch?v=xirEDVhcXRc>. Acesso em: 5 jul. 2018.

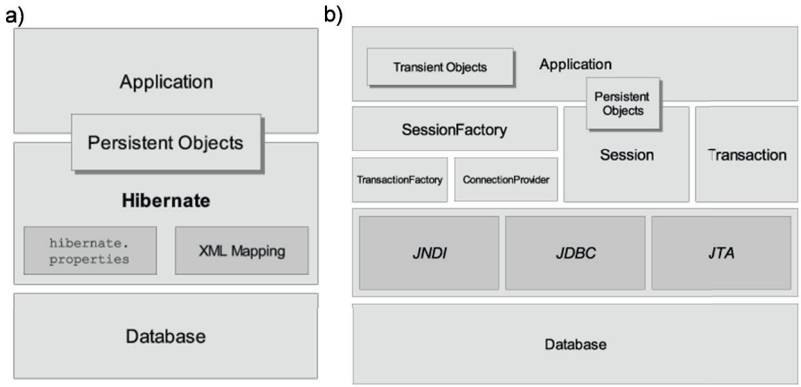
Conforme Hibernate (2004), a arquitetura do Hibernate é composta basicamente por um conjunto de interfaces, estando as mais importantes nas camadas de negócio (*Application*) e persistência (*Persistent Objects*). Na Figura 2.5 - a, temos a representação da arquitetura do Hibernate em alto nível de abstração, em que a camada de negócio permanece acima da camada de persistência, por ser um cliente da camada da persistência de dados. De acordo com o Hibernate (2004), o *framework* é muito flexível, suportando várias abordagens em Java, em que podemos observar duas situações desta arquitetura: a mínima e a compreensiva.

Na arquitetura mínima o aplicativo consegue fornecer suas próprias conexões JDBC e gerenciar suas transações, ou seja, nesse caso, é usado o mínimo das APIs do Hibernate. Já a arquitetura compreensiva (Figura 2.5 – b) abstrai a aplicação do JDBC/JTA e APIs adjacentes, ficando sob responsabilidade do Hibernate tomar conta dos detalhes. A Figura 2.5 - b apresenta o diagrama do *framework* Hibernate divido basicamente em três camadas:

- A camada de aplicação, onde encontramos a aplicação do negócio.

- A camada de persistência, onde encontramos as propriedades do Hibernate.
- A camada do banco de dados, onde os dados são armazenados.

Figura 2.5 | Arquitetura do Hibernate: a) visão geral; b) compreensiva



Fonte: <https://docs.jboss.org/hibernate/orm/3.5/reference/pt-BR/html/architecture.html>. Acesso em: 10 jul. 2018.

De acordo com Hibernate (2004), com as camadas do *framework* Hibernate identificadas na Figura 2.5 - b, podemos identificar as interfaces de cada camada, definidas como:

- *Session, Transaction e Query* – são responsáveis pela execução de operações de inserção, remoção, consulta e atualização no banco de dados.
- *Configuration* – é responsável pela configuração do Hibernate na aplicação.
- *Interceptor, Lifecycle e Validatable* – são responsáveis pela interação entre o *framework* Hibernate e a aplicação.
- *UserType, CompositeUserType, IdentifierGenerator.* – são responsáveis por permitir a extensão das funcionalidades de mapeamento do Hibernate.

Assimile

O ORM tem a função de converter dados que estão como objetos no sistema para a representação de tabelas em bancos de dados relacionais, podendo também recuperar os dados na forma de tabelas, convertendo-os em objetos.

Objetos na camada de persistência

Segundo Coelho (2013), é importante conhecer o conceito e os fundamentos das interfaces presentes na camada de persistência do Hibernate.

Desta forma, é possível definir os objetos presentes na sua camada, como:

- *Session* – considerada a principal interface utilizada por aplicativos no Hibernate e é responsável pelo gerenciamento dos estados dos objetos criados. Nesta interface, o Hibernate realiza as operações de inserção, leitura, atualização e remoção de informações, controle das transações e execução de consultas no banco de dados.
- *SessionFactory* – é responsável por fornecer ao sistema, instâncias da interface *Session*. Armazena metadados de mapeamentos utilizados pelo Hibernate quando está em execução e armazena instruções de SQL.
- *ConnectionProvider* – é responsável por fornecer inúmeras conexões JDBC. Juntamente com o sistema, abstrai os drivers disponíveis do banco de dados. O uso desta interface na estrutura é opcional.
- *Transaction* – é responsável por realizar a tarefa de abstração do código do sistema para implementar as transações do usuário. A abstração possibilita o controle dos limites de transações realizadas pelo sistema, permitindo a portabilidade dos sistemas do Hibernate, mesmo em diferentes dispositivos. No entanto, o uso dessa interface na estrutura é opcional.
- *TransactionFactory* – é responsável por fornecer instâncias da interface *Transaction* e seu uso também é opcional.

Essas interfaces podem ser implementadas manualmente ou através de uma *Entity Manager*, que veremos na próxima seção.

Reflita

A persistência com Hibernate facilita o uso de conexão e execução de comandos no banco de dados. Para se tornar um desenvolvedor nos dias atuais, você acredita que é necessário obter conhecimento em comandos SQL? Chegaremos em um momento em que aprender programação não será mais preciso?

O *framework* Hibernate apresenta três tipos de estados, referentes aos objetos instanciados, definidos como:

- Objetos transientes – são objetos com instâncias que não estão associadas a nenhum contexto de persistência. Após serem instanciados, são utilizados e depois de sua destruição não podem ser recriados automaticamente.
- Objetos persistentes – são objetos que possuem uma identidade de banco de dados com instâncias associadas a um contexto de persistência.
- Objetos desanexado – são objetos que deixaram de ter as instâncias associadas com um contexto de persistência, como fechamento ou finalização de uma sessão. Não são transientes e nem persistentes, mas são objetos em um estado intermediário.

Configuração do Hibernate em uma aplicação

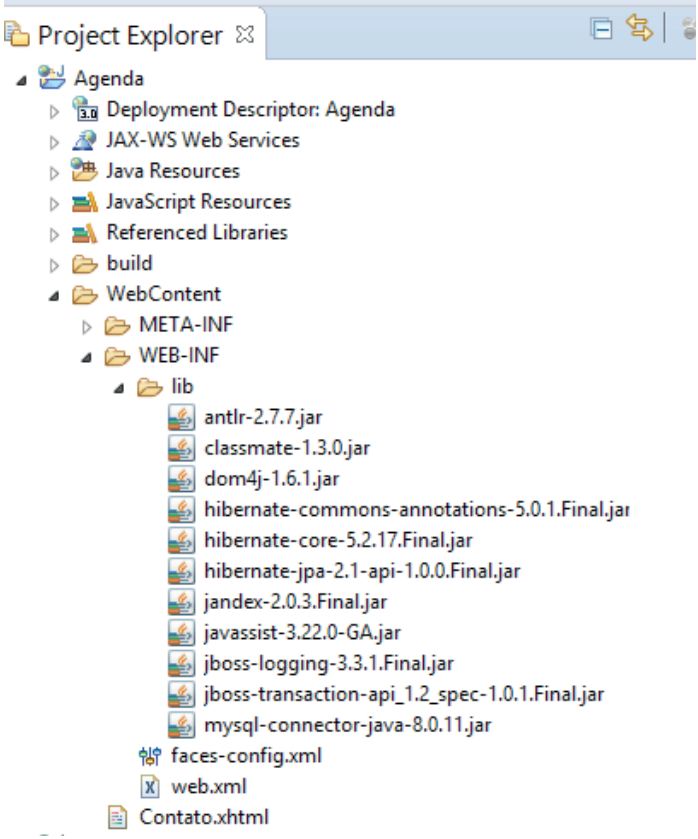
Na primeira unidade do nosso livro de estudo, você já instalou a IDE do Eclipse e o servidor WildFly. Veremos agora como configurar o Hibernate.

Caso ainda não tenha, você deve fazer o download do Oracle® MySQL Connector/J no link <https://dev.mysql.com/downloads/connector/j/>, acesso em: 10 jul. 2018. Depois de concluído, é necessário realizar o download do Hibernate acessando <http://hibernate.org/orm/releases/5.2/>, acesso em 10 jul. 2018. Ao terminar o download da biblioteca do Hibernate, descompacte o arquivo .zip. Acesse a pasta criada, depois abra a pasta *lib* e, em seguida, a pasta *required*, onde estão todos os arquivos JARs necessários. Tanto o arquivo *.jar* do conector MySQL quanto os arquivos da pasta *required* do Hibernate devem ser copiados e colados dentro da pasta *lib* do projeto no Eclipse.

Com os arquivos copiados, clique com o botão direito na pasta raiz do projeto e acesse *Properties*, ao abrir a janela, acesse *Java Build Path* ao lado direito e, ao lado esquerdo, acesse *Libraries*. Clique em *Add Jars* para adicionar as bibliotecas e, então, selecione todos os arquivos da pasta *lib*, clique em OK e depois em *Apply* para que as bibliotecas sejam referenciadas no projeto.

Pronto, suas bibliotecas já estão configuradas e prontas para uso. Veja na Figura 2.6 como fica a estrutura de pastas e arquivos no projeto.

Figura 2.6 | Estrutura de arquivos após a inclusão do MySQL Connector e do Hibernate



Fonte: captura de tela do Eclipse, elaborada pelo autor.

A pasta *lib* é apenas um repositório para os arquivos. Quando se adiciona os .jar ao *Java Build Path*, eles se tornam bibliotecas referenciadas, ou seja, passam a ser acessíveis pelo projeto. Abra a pasta *Referenced Libraries* e veja que os arquivos estarão incluídos.

A finalização da configuração necessária é feita no arquivo *persistence.xml*, que deve ser criado dentro da pasta META-INF que, no caso do projeto web, está dentro de *WebContent*. Nesse arquivo, a configuração do Hibernate é feita nas tags <*properties*>. O Quadro 2.6 apresenta as configurações necessárias.

Quadro 2.6 | Configurações das <properties> no arquivo *persistence.xml*

```
1.  <properties>
        <!-- Conexão com o banco de dados -->
2.      <property name="hibernate.connection.
    driver_class" value="com.mysql.jdbc.Driver" />
3.          <property name="hibernate.connection.url"
    value="jdbc:mysql://localhost:3306/Agenda" />
4.          <property name="hibernate.connection.
    user" value="root" />
5.          <property name="hibernate.connection.
    password" value="" />
6.          <property name="hibernate.
    dialect" value="org.hibernate.dialect.
    MySQL5InnoDBDialect" />
7.          <property name="hibernate.hbm2ddl.auto"
    value="create"/>
        <!--Configuracoes de Debug-->
8.          <property name="hibernate.show_sql"
    value="true" />
9.          <property name="hibernate.format_sql"
    value="true" />
10.         <property name="use_sql_comments"
    value="true" />
11. </properties>
```

Fonte: elaborado pelo autor.

A tag <*property*> na linha 2 define qual será o agente de conexão com o banco de dados que será utilizado. Nesse caso, vamos utilizar o MySQL (por isso instalamos o conector). O JDBC fará a "tradução" do Hibernate para o SQL (*Structured Query Language*, ou Linguagem de Consulta Estruturada).

A tag <*property*> na linha 3 é usada para informar em qual caminho se encontra o nosso banco de dados. A porta padrão do MySQL é a 3306 e o nome do banco de dados, nesse caso, é "agenda".

Na linha 4, a tag <*property javax.persistence.jdbc.user*> define o usuário de conexão do banco e, na linha 5, a tag <*property javax.persistence.jdbc.password*> define a senha do usuário para conexão com o banco (esses valores dependerão da sua conexão com o sistema gerenciador de banco de dados). Caso sua conexão não possua senha, use aspas, conforme o exemplo.

Na linha 6, o <*property*> define o dialeto que o Hibernate vai utilizar. Como podem haver pequenas diferenças entre bancos de dados diversos, definimos o dialeto para criação dos comandos SQL corretos para a linguagem.

A tag <*property*> da linha 7 especifica a ação sobre o banco de dados. No nosso caso, a opção criará o esquema toda vez que o sistema for executado, ou seja, novas tabelas sempre serão criadas. É importante você ter em mente que essa opção apaga os dados, portanto, é recomendado que se use somente na primeira execução (a não ser que queira "zerar" os dados no banco).

As tags nas linhas 8, 9 e 10 são usadas para depuração, pois permitem escolher se o SQL gerado automaticamente será ou não exibido no console e como ele será exibido.

A tag da linha 8 define se os comandos SQL serão exibidos ou não no console. O comando na linha 9 formata a exibição para acrescentar linhas a cada comando SQL e no comando na linha 10, o Hibernate colocará comentários dentro de todas as instruções SQL geradas para sugerir o que o SQL gerado está tentando fazer. Veja no Quadro 2.7 os possíveis resultados dos três comandos.

Quadro 2.7 | Comandos para depuração

Comando	Resultado
`<property name="hibernate.show_sql" value="true" />`	`Hibernate: insert into meu_banco (CHANGE, CLOSE, DATE, OPEN, STOCK_ID, VOLUME) values (?, ?, ?, ?, ?, ?)`
`<property name="hibernate.format_sql" value="true" />`	`Hibernate: insert into meu_banco (CHANGE, CLOSE, DATE, OPEN, STOCK_ID, VOLUME) values (?, ?, ?, ?, ?, ?)`

Comando	Resultado

`<property name="use_sql_comments" value="true" />`	`Hibernate:`
	`/* insert meu_banco`
	`*/ insert`
	`into`
	`meu_banco`
	`(CHANGE, CLOSE, DATE, OPEN, STOCK_ID, VOLUME)`
	`values`
	`(?, ?, ?, ?, ?, ?)`

Fonte: elaborado pelo autor.

Anotações usadas pelo Hibernate

Você já sabe que o Hibernate faz o mapeamento classes–tabelas e vice-versa, mas como isso é feito? A resposta é: usando anotações específicas.

A anotação é uma forma especial de declaração de informações sobre o comportamento de um sistema, e as classes, métodos, atributos e outros elementos de um programa são aplicáveis, além de não ter nenhum efeito sobre a execução do código onde estão inseridas (COELHO, 2013).

Uma das vantagens de possuir em um projeto somente anotações da Hibernate é que ela pode se tornar portável automaticamente, podendo realizar a troca do *framework* Hibernate pelo EclipseLink, alterando somente um arquivo de configuração no *persistence.xml*, e realizar a troca do JAR da implementação.

Segundo Faria e Normandes Jr. (2015), para a realização de um mapeamento objeto relacional, é necessário transmitir à implementação do Hibernate algumas informações de como as classes devem ser tornar persistentes, ou seja, como serão gravadas as instâncias dessa classe.

Para informar que uma classe deve ser persistida no banco como uma tabela, usa-se a anotação **@Entity** antes da declaração da classe. Caso queira especificar um nome para a tabela (diferente do nome da classe), usa-se a anotação **@Table(name = "nome_tabela")** depois do @Entity.

O mapeamento dos atributos da classe para campos na tabela é feito por meio da anotação **@Column**.

Exemplificando

Veja no Quadro 2.8 um exemplo com algumas anotações para o mapeamento da classe *Contato*.

Quadro 2.8 | Exemplo de anotações Hibernate

```
@Entity
@Table(name="contato")
public class Contato {

        @Column
        private int id;

        @Column(length = 80, nullable = false)
        private String nome;

        @Column
        private String sobrenome;

        //métodos getters e setters
}
```

Fonte: elaborado pelo autor.

Existem várias anotações para serem usadas no processo de mapeamento, veja no Quadro 2.9 algumas das mais usadas e sua respectiva função.

Quadro 2.9 | Anotações Hibernate mais utilizadas

Anotação	Descrição
@Entity	Identifica que a classe é uma entidade, representando uma tabela do banco de dados.
@Table	Define detalhes da tabela no banco de dados, como o nome.
@Id e @ GeneratedValue	Utilizadas para declarar o identificador do banco de dados, que deve ter um valor gerado no momento de inserção (autoincremento).

Anotação	Descrição
@Column	Especifica que a propriedade possui um tamanho e restrições quanto à sua utilização. Quando não informado o nome da coluna no banco, este receberá o nome da mesma propriedade.
@ManyToOne	Indica a multiplicidade do relacionamento entre tabelas.
@JoinColumn	Indica que a relação é realizada por meio da coluna especificada.
@Enumerated	Usado para mapear atributos do tipo de enumeração (enum).

Fonte: elaborado pelo autor.

Até o momento aprendemos o que é o JPA, como realizar a configuração do Hibernate no projeto, fazendo o download e a configuração dos arquivos necessários, bem como implementar o arquivo *persistence.xml* e como as classes são transformadas (mapeadas) em tabelas usando as anotações Hibernate. Agora falta pouco para concluir o processo de persistência de dados, confira na próxima seção.

Sem medo de errar

Retomando o nosso desafio, seu gestor o designou para uma importante tarefa na empresa imobiliária, para ajudá-lo na criação de uma solução para seu sistema. E como você já implementou as principais classes desse sistema, vai precisar atualizar o arquivo de persistências XML, realizar a configuração do *framework* Hibernate no sistema imobiliário que está desenvolvendo com seu cliente e adicionar as anotações nas classes criadas, para a persistência.

Comece atualizando o arquivo *persistence.xml*. Nesse momento, você deverá acrescentar a configuração do Hibernate nas tags <*properties*>. O Quadro 2.10 apresenta uma opção para essa configuração.

Quadro 2.10 | Arquivo *persistence.xml*

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
  xmlns="http://xmlns.jcp.org/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
persistence http://xmlns.jcp.org/xml/ns/persistence/
persistence_2_1.xsd">

  <persistence-unit name="imobiliaria_persist"
transaction-type = "RESOURCE_LOCAL">
      <provider>org.hibernate.ejb.
HibernatePersistence</provider>
      <properties>
          <property name="hibernate.archive.
autodetection" value="class" />

           <!-- Conexão com o banco de dados -->
          <property name="hibernate.connection.
driver_class" value="com.mysql.jdbc.Driver" />
            <property name="hibernate.connection.url"
value="jdbc:mysql://localhost:3306/imobiliaria" />
            <property name="hibernate.connection.
user" value="root" />
            <property name="hibernate.connection.
password" value="" />
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5InnoDBDialect" />
          <property name="hibernate.hbm2ddl.auto"
value="create"/>

           <!--Configuracoes de Debug-->
            <property name="hibernate.show_sql"
value="true" />
            <property name="hibernate.format_sql"
value="true" />
            <property name="use_sql_comments"
value="true" />
      </properties>
  </persistence-unit>
</persistence>
```

Fonte: elaborado pelo autor.

Você já criou as classes *Clientes, ClienteLocacao* e *Imoveis*, então volte a elas e acrescente as anotações Hibernate que julgar necessário. Veja no Quadro 2.11 algumas anotações necessárias.

Quadro 2.11 | Classe *Imoveis.java*

```
import javax.persistence.*;

@Entity
@Table(name="imoveis")
public class Imoveis {
	@Id
	@GeneratedValue(strategy=GenerationType.
IDENTITY)
	private int id;

	@Column
	private String descricao;

	@Column
	private String tipo;

	@Column
	private String localizacao;

	@Column
	private int registro;

	//métodos getters e setters

}
```

Fonte: elaborado pelo autor.

Agora é com você, atualize as demais classes do projeto da imobiliária.

Uma observação importante é que a classe *Imoveis* possui um relacionamento com a tabela *ClienteLocacao*, portanto, tente surpreender a equipe e implemente esse relacionamento tendo em mente que um cliente pode alugar vários imóveis, mas um imóvel só pode ser alugado por um cliente. Uma dica é que as anotações @ManyToOne e @JoinColumn terão que ser usadas.

Avançando na prática

Persistindo informações

Descrição da situação-problema

Uma nova startup no ramo de controle de moto táxi, recentemente liberada pela Prefeitura para trabalhar em sua cidade. Como é de

sua especialidade trabalhar com persistência de informações, você foi contrato para implementar a solução. O proprietário deseja criar um sistema que armazenaria tanto os dados dos motociclistas com permissão para trabalhar, como todos os clientes que utilizam ou venham a utilizar o serviço um dia e, assim, manter o histórico de todas as corridas realizadas pelos motociclistas e pelos clientes.

Sua tarefa é criar uma solução para a persistência dos dados, então, mãos à obra!

Resolução da situação-problema

Primeiramente, é importante conhecer e compreender como será o funcionamento deste sistema e sugerir a utilização de um sistema via web, a fim de facilitar a portabilidade e o acesso ao sistema.

Desta forma, você pode trabalhar com um banco de dados em MySQL e criar um sistema em Java, onde utilizará o *framework* Hibernate para realizar a persistência de dados inseridas pelo sistema.

Veja no Quadro 2.12 uma possível implementação para a classe Motoboy e no Quadro 2.13 para a classe Cliente.

Quadro 2.12 | Classe Motoboy

```
package model;

import javax.persistence.*;
@Entity
public class Motoboy {
	@Id
	@GeneratedValue(strategy=GenerationType.
IDENTITY)
	private int id;

	@Column
	private String nome;

	@Column
	private String sobrenome;

	@Column
	private String telefone;

	//métodos getters e setters
}
```

Fonte: elaborado pelo autor.

Quadro 2.13 | Classe Cliente

```
package model;

import javax.persistence.*;
@Entity
public class Cliente {
	@Id
	@GeneratedValue(strategy=GenerationType.
IDENTITY)
	private int id;

	@Column
	private String nome;

	@Column
	private String sobrenome;

	@Column
	private String email;

	@Column
	private String endereco;

	//métodos getters e setters
}
```

Fonte: elaborado pelo autor.

Faça valer a pena

1. O Hibernate não tem como função apenas realizar _____ das classes Java para tabelas do _____, ele também permite que _____ e retorno de dados da consulta sejam realizados com mais facilidade, podendo reduzir de maneira significativa o _____ de desenvolvimento, que antes era gasto pelas operações manuais dos dados feitas com _____ e _____ . (CORDEIRO, 2014).

Assinale a alternativa abaixo que apresenta as respectivas palavras que completam a sentença.

a) o mapeamento, sistema, importações, tempo, SQL e JSF.
b) a estrutura, banco de dados, inclusões, custo, JPA e SQL.
c) o mapeamento, programa, remoções, custo, JDBC e JPA.
d) o mapeamento, banco de dados, consultas, tempo, SQL e JDBC.
e) a estrutura, sistema, atualizações, tempo, JSF e JPA.

2. O Hibernate possui uma estrutura com suporte ao mapeamento de associações entre os objetos, coleções, composição, polimorfismo e herança. O *framework* não só realiza a função de mapeamento objeto relacional, mas também permite consulta de dados, reduzindo o tempo de desenvolvimento da aplicação.

Sobre o *framework* Hibernate, assinale a alternativa abaixo que apresenta as três camadas de arquitetura existentes.

a) Camada de Session, Configuration e Interceptor.
b) Camada de aplicação, persistência e banco de dados.
c) Camada de JNDI, JDBC e JTA.
d) Camada de SessionFactory, ConnectionProvider e TransactionFactory.
e) Camada de Objetos Transientes, Persistentes e Desanexado.

3. Para a persistência usando o Hibernate, além da configuração no arquivo *persistence.xml*, é necessário incluir anotações específicas nas classes que serão mapeadas. Por exemplo, para mapear uma tabela utiliza-se a anotação @Entity e para mapear um atributo em coluna utiliza-se @ Column. Com essas diretrizes, as interfaces do *framework* podem realizar o correto mapeamento.

Analise as sentenças abaixo e assinale V (Verdadeiro) ou F (Falso) para cada uma delas:
1- ( ) O objeto *Session* presente na camada de aplicação é considerado a principal interface utilizada por aplicativos no Hibernate.
2- ( ) A camada de aplicação é onde encontramos a aplicação do negócio do sistema desenvolvido.
3- ( ) Os objetos transientes são objetos com instâncias que não estão associadas a nenhum contexto de persistência na aplicação.
4- ( ) A função *<property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver" />* é responsável pela criação do banco de dados.
5- ( ) O Hibernate é muito flexível, suportando várias abordagens em Java, em que podemos observar duas situações desta arquitetura: a mínima e a compreensiva.

Assinale a alternativa abaixo que apresenta a sequência correta de Verdadeiro (V) e Falso (F).

a) 1 – F, 2 – F, 3 – V, 4 – V, 5 – F.
b) 1 – V, 2 – V, 3 – F, 4 – F, 5 – V.
c) 1 – V, 2 – F, 3 – F, 4 – V, 5 – V.
d) 1 – V, 2 – F, 3 – V, 4 – V, 5 – F.
e) 1 – F, 2 – V, 3 – V, 4 – F, 5 – V.

Seção 2.3

CRUD com *framework* hibernate

Diálogo aberto

Caro aluno, nos dias atuais, a quantidade de informações existentes, principalmente na internet, é enorme e, de alguma forma, elas são armazenadas em banco de dados que as disponibilizam por meio de seus sistemas específicos. Imagine em um supermercado, a quantidade de produtos que são vendidos por hora, assim como a quantidade de produtos que são comprados e entregues nas lojas todos os dias. Referente a um produto vendido nas prateleiras, primeiramente ele precisou ser cadastrado no sistema, depois foi comprado e a quantidade que entrou no supermercado foi inserida no sistema. Ao ser vendido para os consumidores finais, com a passagem pelo caixa, esse produto é listado no cupom de venda e automaticamente dá-se baixa no estoque da quantidade vendida. Quando um produto sai de linha de venda do supermercado, é removido do cadastro, para que não haja mais a compra dele em ocasiões futuras. Se não houvesse a persistência de dados para armazenamento, inclusão ou remoção dessas informações por meio de uma persistência de dados, como seria possível realizar um relatório de produtos mais vendidos no dia, na semana ou em um mês?

Assim, vamos relembrar seu desafio: um cliente de uma empresa imobiliária deseja uma solução para seu novo ramo de negócios. Devido ao seu grande conhecimento em desenvolvimento de sistemas com persistências, você foi designado pelo seu gestor para auxiliar o cliente neste desafio. Desta forma, o sistema deve gerenciar as informações de cadastro de imóveis e de clientes, levando em consideração que um cliente pode ter vários imóveis, mas um imóvel pertence somente a um cliente. Essa apresentação deverá ser realizada com a criação dos códigos-fonte para o *Create, Read, Update and Delete,* ou seja, Criar, Ler, Atualizar e Excluir (CRUD) dos dados no sistema com o *framework* Hibernate. Portanto, retome o projeto já criado e faça as novas atualizações.

Para realizar este desafio, você vai aprender e conhecer nesta seção o processo para a construção de um CRUD utilizando o *framework* Hibernate, para mapear a tabela, inserir e remover dados e também recuperar dados inseridos nas tabelas com a persistência de dados JPA, juntamente com JSF para criar as páginas em um sistema em Java.

Agora é com você. Vamos começar?

Não pode faltar

Conforme Faria e Normandes Jr. (2015), para que funcione o mapeamento objeto relacional é necessário implementar informações sobre como a classe a classe deve se tornar persistente, ou seja, como as instâncias dessas classes deverão ser gravadas e pesquisadas no banco de dados.

Parte da configuração necessária para a persistência usando o framework Hibernate você já viu nas seções anteriores, que consiste em fazer o download do conector MySQL e do Hibernate, descompactar e copiar os arquivos da pasta *lib >> required* para uma pasta chamada *lib* no projeto e adicionar esses arquivos no *classpath* no projeto no Eclipse. Também é preciso configurar o arquivo *persistence.xml* e usar anotações específicas nas classes Java para o mapeamento. Além dessas configurações, é preciso criar a interface de persistência, conhecida como *Entity Manager*, que será criada por uma *Entity Manager Factory*. Após realizar todas as configurações necessárias, é possível criar as operações no banco de dados, conhecidas por CRUD. Para entendermos todos os conceitos envolvidos no CRUD, desenvolveremos essa seção de forma prática, portanto vamos criar o projeto *Agenda* e fazer a persistência e as operações na classe *Contato*.

Primeiramente, é necessário criar o banco de dados. Você pode utilizar o programa XAMPP para usar o banco dados em MySQL, disponível em: <https://www.apachefriends.org/pt_br/download.html>, acesso em: 10 jul. 2018 (fique à vontade para usar outro software, por exemplo, o MySQL Workbench).

Para criar o banco, informe o comando:

- *create database Agenda* ou use as opções gráficas do software.

Vamos criar um novo projeto Web no Eclipse. Depois de aberto, acesse *File / New / Dynamic Web Project* e dê o nome de *Agenda*.

Vamos começar configurando o Hibernate. Para isso, copie o conector MySQL e os arquivos da pasta *lib >> required* (que estarão no local que você salvou no seu computador) para dentro da pasta *WebContent >> WEB-INF >> lib*, do projeto *Agenda*. Feito isso, adicione os arquivos no *Java Build Path*, conforme explicado na seção anterior.

Para realizar a conexão com o banco por meio do arquivo de persistência JPA com Hibernate, é preciso fazer a configuração no arquivo *persistence.xml*. Veja a configuração desse projeto no Quadro 2.14. Na linha 5 foi definido o nome para a seção de persistência *agendaPersist*, que será usado posteriormente, na criação do *Entity Manager Factory*. Na linha 7 foi definido o provedor de persistência. Na linha 9 temos uma definição importante, veja que foram especificados o pacote (*model*) e o nome da classe (*Contato*) que será mapeada usando as anotações Hibernate. Da linha 11 até 14 foram especificados os detalhes para a conexão usando o JDBC. Na linha 11, além do SGBD, da porta e do nome do banco, foi definido o *TimeZone* para que a conexão se completasse com êxito. Nas linhas 12 e 13 foram definidas as credenciais para conexão e na linha 14 o *driver* do SGBD. Da linha 15 à 18 foram especificados os detalhes do Hibernate. Os comandos nas linhas 15 e 16 configuram a exibição da saída do comando SQL, que será criado automaticamente. O comando na linha 17 determina o dialeto do banco e o da linha 18 determina o tipo de operação, nesse caso, o valor create especifica que o Hibernate vai criar as tabelas sempre que o programa for executado, portanto, elas sempre estarão vazias. Esse parâmetro é indicado quando se executa pela primeira vez. Outras opções são:

- *Create-drop*: apaga o *schema* de dados ao terminar a sessão.
- *Update*: faz atualização do *schema* de dados.
- *Validate*: valida o *schema* de dados e não faz mudanças no banco de dados.

Quadro 2.14 | Configuração do arquivo de persistência Hibernate

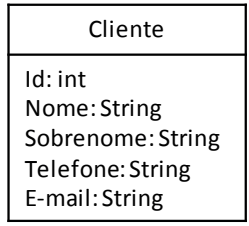
```
1.  <persistence
    xmlns="http://xmlns.jcp.org/xml/ns/persistence"
```

```
2.  xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance"
3.
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/
4.  persistence
    http://xmlns.jcp.org/xml/ns/persistence/
5.  persistence_2_1.xsd"
      version="2.1">
6.
7.    <persistence-unit name="agendaPersist"
    transaction-
8.  type="RESOURCE_LOCAL">
9.
10. <provider>
            org.hibernate.jpa.HibernatePersistence-
    Provider
11.     </provider>
        <class>model.Contato</class>
        <properties>
          <property name="javax.persistence.jdbc.
12. url" value="jdbc:mysql://localhost:3306/
    agenda?serverTimezone=UTC" />

13. <property name="javax.persistence.jdbc.user"
    value="root" />
    <property name="javax.persistence.jdbc.password"
14. value="" />
          <property name="javax.persistence.jdbc.
    driver" value="com.mysql.cj.jdbc.Driver" />
          <property name="hibernate.show_sql"
15. value="true" />
          <property name="hibernate.format_sql"
16. value="true" />
    <property name="hibernate.dialect" value="org.
17. hibernate.dialect.MySQL5InnoDBDialect" />
    <property name="hibernate.hbm2ddl.auto"
18. value="create" />
        </properties>
      </persistence-unit>
    </persistence>
```

Fonte: elaborado pelo autor.

Com a persistência configurada, agora as demais classes podem ser criadas em seus respectivos pacotes. Para seguirmos com a criação de projetos no padrão MVC, a classe *Contato* deverá ser criada dentro do pacote *model*, no diretório raiz *src*, com os campos da Figura 2.7.

Figura 2.7 | Campos da Classe *Contato*



Fonte: elaborada pelo autor.

Agora começaremos a implementar o mapeamento ORM através do Hibernate. Veja no Quadro 2.15 o código referente à classe Contato com as anotações necessárias ao mapeamento. Na linha 2 foi importado o *javax.persistence*.*, que possui as classes para a persistência. Na linha 3 a anotação @Entity é usada para informar que a classe também é uma entidade. Na linha 4 a anotação @Table foi usada para especificar o nome *contato* para a tabela que será criada no banco de dados *Agenda*. Caso a tabela não exista, o Hibernate cria uma automaticamente, com os campos de atributos que possuírem a notação @Column. A notação @Id identifica qual atributo será a chave primária da tabela.

Quadro 2.15 | Classe Contato.java com anotações para persistência

```
1.  package model;
2.  import javax.persistence.*;
3.  @Entity
4.  @Table(name="contato")
5.  public class Contato {


6.          @Id
7.          @GeneratedValue(strategy=GenerationType.
    IDENTITY)
```

```
8.          @Column
9.          private int id;

10.         @Column
11.         private String nome;

12.         @Column
13.         private String sobrenome;
14.         @Column
15.         private String telefone;

16.         @Column
17.         private String email;

18.         //criação getters e setters
19.     }
```

Fonte: elaborado pelo autor.

Assimile

Conforme Cordeiro (2014), uma classe comum somente é diferenciada de uma entidade pela presença das anotações: @Entity, @Id, @GeneratedValue, etc., devido a estas classes com anotações estarem vinculadas a uma unidade de persistência.

Seguindo as premissas da boa prática de programação, vamos criar uma classe específica que fará a conexão com o banco. Segundo Cordeiro (2014), para separar as responsabilidade e definir melhor a legibilidade do código, toda a lógica de acesso aos dados pode ser colocada em uma classe específica para isso chamada *Data Access Object* ou Objeto de Acesso a Dados (DAO).

Vamos agora criar o nosso DAO, no qual teremos a nossa fábrica de conexão do sistema. Nessa classe será criado o *Entity Manager Factory*, responsável por gerar instâncias do *Entity Manager* que são usadas para acessar o banco de dados (são como um canal de comunicação). Por conveniência, a conexão com o banco é criada usando atributos e métodos estáticos, pois assim não precisam ser instanciados. Dentro do pacote *model* crie uma nova classe chamada

DAO. Veja no Quadro 2.16 o código referente a ela. Na linha 2 foi importado o pacote para acesso às classes de persistência. Veja que na linha 4 foi criado um atributo *static final*, significando que esse atributo é imutável em qualquer classe que o acesse. Na linha 6 (dentro do comando *static*) é definido o valor desse parâmetro que, nesse caso, faz a ligação com o arquivo *persistence.xml* através de uma fábrica de conexão chamada *agendaPersist* (lembra que a criamos anteriormente no Quadro 2.14?). Como o atributo é privado, na linha 8 criou-se um método público para acessar a conexão através do comando *return emFactory.createEntityManager()*;. Por fim, na linha 11 criou-se um método para fechar a conexão, pois nunca se deve deixar uma conexão aberta após uma operação, por medidas de segurança.

Quadro 2.16 | Estrutura da Classe *DAO.java*

```
1.  package model;

2.  import javax.persistence.*;

3.  public class DAO {

4.          private static final EntityManagerFactory
    emFactory;
5.
6.          static {
                    emFactory =
7.  Persistence.
    createEntityManagerFactory("agendaPersist");
            }
8.
9.          public static EntityManager
    getEntityManager() {
10.                 return emFactory.
    createEntityManager();
11.         }
12.         public static void fecharFactory() {
13.                 emFactory.close();
14.         }
}
```

Fonte: elaborado pelo autor.

Pesquise mais

O padrão DAO é um intermediário entre o sistema e banco de dados, ele faz conexão e transações com o banco, utilizando a regra de negócio *Model*. Para conhecer mais sobre esse padrão, acesse o vídeo *Criando classes DAO e Model para recuperar do Banco de Dados*, que apresenta as classes Model e DAO para recuperar dados.

PROFESSOR Isidro. **Criando classes DAO e Model para recuperar do Banco de Dados.** 1 ago. 2016. Disponível em: <https://www.youtube.com/watch?v=OnbvhKiln4Y>. Acesso em: 10 jul. 2018.

Para uma melhor organização do projeto, vamos criar o CRUD em uma classe específica. Para isso, dentro do pacote *model*, crie uma nova classe chamada *CRUD*. Veja no Quadro 2.17 as operações de inserção (*Create*), leitura (*Read*), atualização (*Update*) e remoção (*Delete*) de dados usando o Hibernate e, logo abaixo, as respectivas explicações.

Quadro 2.17 | *CRUD* com Hibernate

```
1.  package model;

2.  import javax.persistence.*;
3.  import org.hibernate.HibernateError;
4.  import java.util.List;

5.  public class CRUD {

6.        public static void inserir(Contato c1) {
7.              try {
8.                    EntityManager entityManager
    = DAO.getEntityManager();
9.                    entityManager.
    getTransaction().begin();
10.                   entityManager.persist(c1);
11.                   entityManager.
    getTransaction().commit();
12.                   entityManager.close();
```

```
13.                         System.out.println("conectado
Salvo!");
14.                 }
15.                 catch(HibernateError ex) {
16.                         ex.printStackTrace();
17.                 }
18.         }

19.         public static List<String> ler() {
20.                 EntityManager entityManager = DAO.
getEntityManager();
21.                 entityManager.getTransaction().
begin();
22.
                    List<String> nomes = entityManager.
createQuery("SELECT nome FROM
23. contato").getResultList();
24.             entityManager.getTransaction().
commit();
25.             entityManager.close();
26.                 return nomes;
            }
27.
         public static void atualizar(int id,
String nome, String sobrenome, String telefone,
28. String email ) {
29.                 try {
                            EntityManager entityManager
30. = DAO.getEntityManager();
                            Contato emp = entityManager.
31. find(Contato.class, new Integer(id));
32.                         entityManager.
getTransaction().begin();
33.                         emp.setNome(nome);
34.                         emp.setSobrenome(sobrenome);
35.                         emp.setTelefone(telefone);
36.                         emp.setEmail(email);
37.                         entityManager.
getTransaction().commit();
```

```
38.                         entityManager.close();
                            System.out.println("Registro
40.  atualizado");
41.                 }catch(Exception ex) {
42.                         ex.printStackTrace();
43.                 }
            }
44.
45.         public static void remover(int id) {
46.                 try {
                            EntityManager entityManager
47.  = DAO.getEntityManager();
                            entityManager.
48.  getTransaction().begin();
                            Contato c = entityManager.
49.  find(Contato.class, new Integer(id));
50.                         entityManager.remove(c);
51.                         entityManager.
     getTransaction().commit();
52.                         entityManager.close();
53.                         System.out.println("registro
54.  removido!");
55.                 }catch(Exception ex) {
56.                         ex.printStackTrace();
57.                 }
            }
     }
```

Fonte: elaborado pelo autor.

Em todas as transações com o banco, cinco comandos são obrigatórios:

1. `EntityManager entityManager = DAO.getEntityManager();` - cria uma variável do tipo *EntityManager* para acessar o canal de comunicação com o banco.

2. `entityManager.getTransaction().begin();` - inicia a transação.

3. `entityManager.operação();` - o parâmetro operação especifica qual ação deve ser executada no banco, por exemplo, *persist* para salvar, *createQuery* para fazer a leitura, *find* para encontrar um valor, remove para apagar um registro, entre outros.

4. `entityManager.getTransaction().commit();` - confirma a execução da operação.

5. `entityManager.close();` - fecha a comunicação com o banco.

Os demais comandos dependem de cada caso. Por exemplo, na linha 22 o comando retorna uma lista com os nomes persistidos no banco. O comando na linha 48 resgata um registro no banco, nesse caso do tipo *Contato*, procurando pelo seu número de identificação.

Agora podemos passar a implementar as classes do pacote de controle e de visão. Dentro da pasta raiz *src*, crie um novo pacote chamado *controler* e, dentro dele, é necessário criar o *Manage Bean* para controlar a classe *Contato*, portanto, crie uma nova classe chamada *ContatoMB*. Nessa classe, vão ficar todos os *getters* e *setters* que dão acesso aos atributos da classe *Contato*, além dos métodos que fazem as transações no banco através da classe *CRUD*. Veja no Quadro 2.18 o código da classe. Observe que na linha 3 foi instanciado um objeto do tipo Contato e na linha 6 o método *inserir* da classe *CRUD* foi invocado, passando como parâmetro o objeto para ser persistido. As demais linhas do código nada diferem do que estudamos na unidade anterior.

Quadro 2.18 | Classe ContatoMB

```
1.  package controler;
    import javax.faces.bean.ManagedBean;
2.  import model.*;
```

```
@ManagedBean
public class ContatoMB {
	public ContatoMB() {

	}
	private Contato c = new Contato();

	public String getId() {
		return String.valueOf(c.getId());
	}

	public void setId(String id) {
		c.setId(Integer.parseInt(id));
	}

	public String getNome() {
		return c.getNome();
	}

	public void setNome(String nome) {
		c.setNome(nome);
	}

	//implementar demais getters e setters

	public void salvar() {
		try {
			CRUD.inserir(c);
		}catch(Exception ex) {
			ex.printStackTrace();
		}
	}
	//implementar demais operações do CRUD
}
```

Fonte: elaborado pelo autor.

Não se esqueça de configurar no arquivo *WebContent >> WEB-INF >> faces.config.xml* o *Manage Bean* criado!

```
<managed-bean>
	<managed-bean-name>contatoMB</managed-bean-name>
```

```
        <managed-bean-class>controler.ContatoMB</
managed-bean-class>
        <managed-bean-scope>request</managed-bean-
scope>
      </managed-bean>
```

Desta forma, toda a estrutura da persistência e conexão com o banco foi criada e está pronta para ser utilizada.

Exemplificando

Para usar os recursos, é preciso criar um componente na camada de visão. Para isso, você pode utilizar um arquivo XHTML dentro da pasta *WebContent*. Veja no Quadro 2.19 uma opção que fará a interface do usuário com o sistema.

Quadro 2.19 | Contato.xhtml

```
<!DOCTYPE html>
<html
       xmlns="http://www.w3.org/1999/xhtml"
       xmlns:h="http://xmlns.jcp.org/jsf/html"
>
<h:head>
       <meta charset="UTF-8" />
       <title>Cadastro de Contato</title>
</h:head>

<h:body>
       <h:form>
              <p>Nome:
                 <h:inputText value="#{contatoMB.
nome}"  /> </p>
              <p>Sobrenome:
                 <h:inputText value="#{contatoMB.
sobrenome}"  /> </p>

              <p>Telefone:
                 <h:inputText value="#{contatoMB.
telefone}"  /> </p>

              <p>E-mail:
                 <h:inputText value="#{contatoMB.
email}"  /> </p>
```

```
            <p>
                    <h:commandButton value="Salvar"
action="#{contatoMB.salvar}" />
            </p>
        </h:form>
        <p>Cadastro confirmado: <br />
           Código: #{contatoMB.id}
           Nome: #{contatoMB.nome} /p>
</h:body>
</html>
```

Fonte: elaborado pelo autor.

A utilização do Hibernate facilita o armazenamento e a recuperação de objetos por meio do Mapeamento Objeto-Relacional (ORM). O Hibernate oferece a opção de criar mapeamentos entre modelos de base de dados e modelo de objetos por meio de anotações no código dos objetos.

O Hibernate suporta as associações *One-to-one* (Um para Um), *One-to-Many* (Um para Muitos), *Many-to-one* (Muitos para Um) e *Many-to-Many* (Muitos para Muitos). No mapeamento de associações *One-to-One* embutidas, temos os atributos das entidades relacionadas que serão persistidas na mesma tabela. Por exemplo, uma classe *Contato* que tem um relacionamento *One-to-One* com a classe *Endereço*. Já no mapeamento de associações *One-to-One* convencionais, a anotação *One-to-One* é utilizada para relacionar duas entidades sem que uma seja componente da outra, diferentemente da associação embutida. Mapeando as associações *Many-to-One* ou *One-to-Many*, temos a anotação *@ OneToMany*, que pode ser aplicada para um campo ou propriedade de uma coleção ou um *array* representando o *many* da associação. Já no mapeamento de associações *Many-to-Many*, a anotação *@ ManyToMany* possui os seguintes atributos:

- *mappedBy:* campo que indica o dono do relacionamento. Este atributo só é necessário quando a associação é bidirecional.
- *targetEntity:* classe da entidade que é o destino da associação.

- *cascade:* indica o comportamento em cascata da associação, sendo que o padrão é *none* (nenhum).

- *fetch:* indica o comportamento de busca da associação, sendo que o padrão é LAZY.

Reflita

Você pode observar que a utilização de persistência JPA com Hibernate permite o mapeamento de bancos, tabelas e colunas, não sendo necessário usar comando em SQL para sua criação. De que forma a utilização de persistência influencia no aprendizado da linguagem SQL? No futuro, será necessário aprender esta linguagem de banco para os desenvolvedores de sistema?

Com essa seção finalizamos os conceitos pertinentes à utilização do JSF com o Hibernate, de acordo com as especificações JPA. Não deixe de aprofundar seus conhecimentos e implementar soluções cada vez mais sofisticadas.

Sem medo de errar

Caro estudante, trabalhando na implementação de um sistema para uma imobiliária, chegou o momento de finalizar a primeira versão deste sistema e então apresentar ao cliente. Você já implementou as classes *Clientes, Imoveis* e *ClienteLocacao* com as respectivas anotações Hibernate e iniciou a configuração do arquivo *persistence.xml*. Nesse momento, você deve avaliar se todas as classes são realmente necessárias, pois o sistema vai cadastrar imóveis e clientes com a regra de que cada cliente pode ter vários imóveis, mas um imóvel só pode pertencer a um cliente.

1. Para finalizar a primeira versão do projeto, crie um banco de dados chamado *imobiliária* em um SGBD MySQL de sua preferência.

2. Agora vamos analisar as classes já criadas. Como o relacionamento é 1 para N, a criação de uma tabela para ele não é exigida, então você pode simplesmente trabalhar com chaves

primárias e estrangeiras. Existem várias formas de implementar essa solução, veja nos Quadros 2.20 e 2.21 uma opção de implementação para as classes *Clientes* e *Imoveis*, agora com o relacionamento estabelecido.

Quadro 2.20 | Classe *Clientes.java* com relacionamentos

```
package model;

import java.util.List;

import javax.persistence.*;

@Entity
@Table(name="clientes")
public class Clientes {
	@Id
	@GeneratedValue(strategy=GenerationType.
IDENTITY)
	@Column
	private int id;
	@Column
	private String nome;
	@Column
	private String cpf;
	@Column
	private String endereco;
	@Column
	private String cidade;

	@OneToMany(mappedBy = "cliente", //determina que
essa tabela é gerenciada por outra
			targetEntity = Imoveis.class,
//informa qual a entidade estamos associando
			fetch = FetchType.LAZY, //usado
para performance, nem todos os campos são carregados
até que sejam necessários
			cascade = CascadeType.ALL)//
todos os eventos anteriores serão sempre refletidos nas
entidades relacionadas
	private List<Imoveis> imoveis;//um cliente pode
ter vários imóveis
	//demais getters e setters das propriedades
private
}
```

Fonte: elaborado pelo autor.

Quadro 2.21 | Classe *Imoveis.java* com relacionamentos

```
package model;

import javax.persistence.*;

@Entity
@Table(name="imoveis")
public class Imoveis {
	@Id
	@GeneratedValue(strategy=GenerationType.
IDENTITY)
	private int id;
	@Column
	private String referencia;
	@Column
	private String descricao;
	@Column
	private String tipo;
	@Column
	private String localizacao;
	@Column
	private double valor;

	@ManyToOne
	@JoinColumn(name="id_cliente")//nome da chave
estrangeira na tabela imóveis
	private Clientes cliente;//nome que liga o
atributo mappedBy na tabela Clientes

	public Clientes getCliente() {
		return cliente;
	}

	public void setCliente(Clientes cliente) {
		this.cliente = cliente;
	}

	//demais getters e setters
}
```

Fonte: elaborado pelo autor.

3. Com as classes devidamente preparadas, verifique a configuração do Hibernate no arquivo *persistence.xml*. A URL usada para conexão nesse projeto foi:

```
<property name="javax.persistence.jdbc.
url" value="jdbc:mysql://localhost:3306/
imobiliaria?serverTimezone=UTC" />
```

Você poderá verificar uma opção para a configuração completa do arquivo *persistence.xml* no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u2/s3/Persistence_SP.pdf> ou por meio do QR Code.

4. Para a conexão com o banco, é necessário criar a fabricação de conexões (*Entity Manager Factory*), o que pode ser feito em uma classe *DAO*.

Você poderá verificar uma opção para a criação da fábrica de conexão no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u2/s3/ClasseDAO_SP.pdf> ou por meio do QR Code.

5. Agora é hora de implementar a transação de inserção de dados no banco. Uma opção é criar uma classe chamada *CRUD* e nela realizar as transações no banco. Com essa etapa, encerra-se toda a parte de configuração e operação com o banco de dados.

Você poderá verificar uma opção para o código de inserção de dados no banco no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u2/s3/CRUD_SP.pdf> ou por meio do QR Code.

6. Para que você possa ver o projeto funcionando, é preciso implementar também o que aprendeu na primeira unidade, ou seja, o JSF. Portanto, crie um novo pacote chamado *controler* e, dentro dele, uma classe chamada *ImobiliariaMB* para gerar o *Manage Bean*.

Você poderá verificar uma opção para o código da *Manage Bean ImobiliariaMB* clicando aqui <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u2/s3/ImobiliariaMB_SP.pdf> ou por meio do QR Code.

7. Após a criação da *Manage Bean*, não se esqueça de configurá-la no arquivo *faces-config.xml*.

```
<managed-bean>
  <managed-bean-name>imobiliariaMB</managed-bean-name>
  <managed-bean-class>controler.ImobiliariaMB</managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
</managed-bean>
```

8. Por fim, é preciso criar a interface para interação do usuário com o sistema. Na pasta *WebContent* crie o arquivo *Cadastro.xhtml*.

Você poderá verificar uma opção para o código da interface gráfica clicando aqui <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u2/s3/CadastroHXML_SP.pdf> ou por meio do QR Code.

Ao completar as 8 etapas descritas, com os códigos devidamente implementados, a primeira versão do seu sistema para a imobiliária está pronta para ser apresentada ao cliente.

Avançando na prática

Relacionamento entre tabelas

Descrição da situação-problema

Uma escola de idiomas possui alunos de diversas cidades. Os donos da instituição desejam mapear os alunos de acordo com a

cidade a qual pertencem, mas, para isso, precisarão efetuar o cadastro dos estudantes e suas respectivas cidades. A empresa em que você trabalha foi contratada para implementar essa solução usando um sistema Web. Seu gerente o designou para criar as classes usando o *framework* Hibernate e já criando os relacionamentos necessários. Portanto, implemente as classes *Cliente* e *Cidade* e, pelas anotações Hibernate, crie o relacionamento entre elas.

Resolução da situação-problema

O primeiro ponto a se observar é o tipo de relacionamento. Um cliente pode morar em uma única cidade, mas uma cidade pode ter inúmeros clientes. Veja nos Quadros 2.22 e 2.23 as classes *Cliente* e *Cidade*, com as respectivas anotações necessárias para a persistência usando o Hibernate.

Quadro 2.22 | Classe *Cliente.java*

```
package model;

import javax.persistence.*;

@Entity
@Table(name="cliente")
public class Cliente {

        @Id
        @GeneratedValue(strategy=GenerationType.
IDENTITY)
        @Column
        private int id;

@Column
       private String nome;

       @ManyToOne
       @JoinColumn(name="id_cidade")
       private Cidade cidade;

       //métodos getters e setters
       }
}
```

Fonte: elaborado pelo autor.

Quadro 2.23 | Classe *Cidade.java*

```
package model;

import javax.persistence.*;

@Entity
@Table(name="cliente")
public class Cliente {

	@Id
	@GeneratedValue(strategy=GenerationType.
IDENTITY)
	@Column
	private int id;

	@Column
	private String nome;

	@ManyToOne
	@JoinColumn(name="id_cidade")
	private Cidade cidade;

	//métodos getters e setters
	}
}
```

Fonte: elaborado pelo autor.

Faça valer a pena

1. O mapeamento feito com a anotação @Entity informa que essa classe é uma _____ e a anotação @Table informa que essa classe utiliza a _____ especificada e, caso não exista, o _____ a cria automaticamente, como os campos de _____ que possuírem a anotação @Column. A anotação _____ identifica qual atributo será a chave primária da tabela.

Dada a sentença acima, assinale a alternativa que apresenta as respectivas palavras que completam a frase.

a) entidade, anotação, Hibernate, dados, @Id.
b) entidade, tabela, JPA, dados, @Column.
c) controler, tabela, JPA, atributos, @Id.

d) controler, anotação, Hibernate, atributos, @Column.
e) entidade, tabela, Hibernate, atributos, @Id.

2. Com a utilização do *framework* Hibernate, para que funcione o mapeamento objeto-relacional, é necessário implementar mais informações, bem como tornar persistente a classe, ou seja, como as instâncias dessas classes deveram ser gravadas e pesquisadas no banco de dados.

Com base na afirmação acima, assinale Verdadeiro (V) ou Falso (F) para as sentenças abaixo, referente ao uso de anotações em Hibernate.
1- ( ) A utilização de anotações nos permite um código mais intuitivo.
2- ( ) É possível mapear as tabelas, colunas e campos.
3- ( ) O Hibernate suporta as anotações JPA.
4- ( ) O arquivo persistence.xml grava as informações no banco de dados.
5- ( ) O Hibernate gera as anotações automaticamente.

Assinale a alternativa que contém a sequência correta.

a) 1 – V, 2 – V, 3 – V, 4 – F, 5 – F.
b) 1 – V, 2 – F, 3 – V, 4 – F, 5 – V.
c) 1 – F, 2 – V, 3 – F, 4 – V, 5 – V.
d) 1 – V, 2 – V, 3 – F, 4 – V, 5 – F.
e) 1 – F, 2 – F, 3 – V, 4 – F, 5 – V.

3. Em um sistema em Java, ao estabelecer conexão com o banco, o Hibernate gera toda a comunicação de scripts em SQL inclusive criando a tabela no banco e os campos, caso eles não existam, e adicionando as informações, sem precisar digitar nada em SQL.

Analise as sentenças abaixo:
I- O comando *em.getTransaction().begin()* inicia o processo de transação com o banco de dados.
II- Caso aconteça algum erro, o processo é cancelado por meio do comando *em.getTransaction().rollback()*.
III- O comando *q.executeUpdate()* atualiza informações no banco de dados.
IV- O comando *em.persist(contato)* realiza a persistência de dados.
V- A informação é gravada pela transação no banco de dados por meio do comando *em.getTransaction().commit()*.

Com relação ao CRUD no *framework* Hibernate, assinale a alternativa que corresponde às sentenças corretas.

a) I, II, III apenas.
b) II, IV e V apenas.
c) I, II, III e V apenas.
d) I, IV e V apenas.
e) III, IV e V apenas.

Referências

COELHO, H. **JPA Eficaz:** as melhores práticas de persistência de dados em Java. [S.l.]: Casa do Código, 2013, p. 155.

CORDEIRO, G. **Aplicações Java para a web com JSF e JPA.** [S.l.]: Casa do Código, 2012, p. 285.

FARIA, T.; NORMANDES Jr. **JPA e Hibernate**. 1. ed. São Paulo: AlgaWorks Softwares, 2015, p. 175.

HIBERNATE Community Documentation. **HIBERNATE - Persistência Relacional para Java Idiomático.** [S.l.], 2004. Disponível em: <https://docs.jboss.org/hibernate/orm/3.5/reference/pt-BR/html/index.html>. Acesso em: 10 jul. 2018.

JBOSS. **Hibernate Getting Started Guide.** JBoss, [s.l.], 2018. Disponível em: <http://docs.jboss.org/hibernate/orm/5.3/quickstart/html_single/#hibernate-gsg-tutorial-jpa-config-pu>. Acesso em: 5 jul. 2018.

ORACLE. **Java Persistence API: XML Schemas. Oracle, [s.l.],** 2018. Disponível em: <http://www.oracle.com/webfolder/technetwork/jsc/xml/ns/persistence/index.html>. Acesso em: 5 jul. 2018.

Desenvolvimento de aplicações em JSF e suas extensões

Convite ao estudo

Prezado aluno, bem-vindo à terceira unidade do estudo em programação para web II. Até o momento, você aprendeu a criar aplicações web usando o *Java Server Faces* (JSF) e a persistir os dados em um banco de dados relacional usando o *framework* Hibernate. Foram muitos desafios até aqui que contribuíram para seu aperfeiçoamento profissional. Continue seus estudos e vamos aprofundar esse conhecimento para que você aprenda a usar extensões para o ambiente *Java Server Faces*.

Você possui uma empresa de desenvolvimento de sistemas web e um de seus clientes mais importantes tem um sistema antigo de cadastro de clientes, fornecedores e produtos, criado por sua empresa, e gostaria de atualizá-lo com um visual mais moderno. O cliente deseja que você renove o sistema com telas de cadastro mais dinâmicas, inserindo componentes que proporcionem uma experiência de usabilidade nova aos seus usuários. No entanto, ele não quer gastar muito com esta melhoria e deseja que a implementação aconteça o mais rápido possível. Então, será necessário apresentar algumas possibilidades de utilização de biblioteca de componentes que melhor se adequem às solicitações de seu cliente.

Em seu primeiro desafio desta unidade, você vai precisar implementar o *framework* RichFaces para criar a tela de cadastro de seu cliente, inserindo componentes modernos e dinâmicos com este *framework*. Na segunda seção, você precisará implementar outro *framework* para realizar esta modernização nos campos desejados, mas utilizando o *framework* PrimeFaces. Na terceira entrega de atualização para o cliente, você deverá

atualizar a tela de cadastro de produtos. Quais as bibliotecas mais indicadas para tornar a usabilidade mais agradável? Quais recursos podem ser incorporados em cada desafio?

Para realizar a atuação do sistema, você aprenderá nesta unidade sobre alguns *frameworks* disponíveis para JSF, que são bibliotecas de componentes para extensões de recursos. Dentre elas, você conhecerá a RichFaces e a PrimeFaces, duas das mais conhecidas e utilizadas bibliotecas de componentes no mercado.

Além disso, será estudado o uso de extensões para JSF, então, você será capaz de aprimorar as telas de utilização dos usuários, tornando-as mais dinâmicas e atraentes às pessoas que acessarem a aplicação.

Seção 3.1

Framework, RichFaces, OpenFaces, ICEfaces

Diálogo aberto

Caro aluno, você já deve ter ido a algum supermercado várias vezes e, sempre que volta, pelo menos uma prateleira ou ilha (com aqueles produtos que ficam no meio do corredor) estão em locais diferentes ou com disponibilização diferente dentro do mercado. Este tipo de alteração permite que o mercado seja mais dinâmico na apresentação de produtos e que melhore a experiência do cliente. Estas mudanças no layout do mercado não alteram a funcionalidade da empresa, que é a venda de produtos, mas mudam a apresentação, o que pode favorecer a venda de certos produtos em detrimento de outros. Assim também acontece com os sistemas web: as páginas precisam ser frequentemente ajustadas, a fim de melhorar a experiência do usuário.

Um cliente o procurou para atualizar um sistema que você mesmo desenvolveu. Ele relatou que gostaria de um visual mais moderno, que cause uma boa impressão ao usuário. Como o cliente exigiu um tempo curto para finalização, você combinou três entregas, portanto, nessa primeira etapa deverá atualizar a página de cadastro de clientes. Qual recurso pode ser usado para realizar essa atualização em um curto espaço de tempo? Todos os campos precisam ser atualizados? Você precisará mexer em outros componentes do sistema, por exemplo, em alguma *Manage Bean*? A configuração do ambiente de programação precisa receber mais alguma alteração?

Para cumprir essa primeira etapa, nessa seção você conhecerá os *frameworks* RichFaces, OpenFaces e ICEfaces. Veremos alguns componentes e como realizar a configuração desses recursos no ambiente de desenvolvimento.

Bons estudos!

Não pode faltar

O *Java Server Faces* (JSF) em aplicações web facilita muito o trabalho de criação de uma página. Como ele é baseado em componentes que podem ser implementados no código da aplicação, temos um sistema mais rico em componentes visuais. Além de seus componentes padrão, é possível integrar outras bibliotecas com itens que podem enriquecer a interface visual de um sistema web. Conforme Coelho (2013), os componentes presentes nessas bibliotecas têm o foco na funcionalidade, em geral, implementando caixas de diálogos, tabela de dados, galerias, etc. Dentre as bibliotecas existentes, podemos citar RichFaces, PrimeFaces, ICEfaces, Tomahawk, OpenFaces, entre muitas outras.

Assimile

Podemos comparar o uso de uma biblioteca de componentes com o nosso dia a dia. Após um tempo com um carro que comprou, você começa a trocar o jogo de rodas para um de liga leve, trocar o som para um mais completo, enfim, você melhora a aparência do veículo. Com os frameworks a ideia é a mesma. Um site sempre com a mesma visualização passa a ser menos chamativo que novos sites que surgem e, com as bibliotecas de componentes, podemos adicionar novos atrativos de forma mais simples e fácil.

Introdução do framework RichFaces

O *framework* RichFaces é uma biblioteca gratuita de componentes JSF, ideal para aplicações que utilizam Ajax (*Asynchronous JavaScript and XML*, ou seja, *JavaScript* Assíncrono e XML).

Conforme Leathem, Fryc e Rogers (2011), a biblioteca permite criar uma interface de usuário avançada e moderna. O RichFaces fornece um recurso de *skins* (aparência) que permite definir e gerenciar diferentes esquemas de cores e outros parâmetros da aparência. É possível acessar os parâmetros de aparência do código da página durante o tempo de execução.

Segundo Leathem, Fryc e Rogers (2011), a biblioteca é composta por três arquivos *.jar*, considerados os recursos básicos: *richfaces-core*.

jar, *richfaces-a4j.jar* e *richfaces-rich.jar*. Além deles, ela trabalha com outras dependências de terceiros. Alguns dos serviços de estrutura só são ativados quando as bibliotecas opcionais estão presentes no projeto, portanto, vamos também adicioná-las ao projeto.

Para configurar o RichFaces na IDE Eclipse, faça o download do recurso em seu site oficial, pelo link <http://richfaces.jboss.org/download/stable.html>, acesso em: 31 jul. 2018. Escolha o arquivo *richfaces-distribution-4.5.17.Final* e, após o download, descompacte-o em uma pasta. Então, adicione as bibliotecas RichFaces e suas dependências obrigatórias ao projeto. Abra a pasta em que você descompactou o RichFaces e copie os arquivos *richfaces-a4j-4.5.17.Final*, *richfaces-core-4.5.17.Final* e *richfaces-rich-4.5.17.Final* para a pasta *WebContent* >> *WEB-INF* >> *lib* do seu projeto. Agora, volte para a pasta em que descompactou o RichFaces, copie os 4 arquivos *.jar* e cole-os na pasta *lib* do projeto.

Após a inserção das bibliotecas, o RichFaces está pronto para ser usado em uma página XHTML. Seu uso substitui a tag `<html>` por `<ui:composition>` e, para que funcione integrado ao JSF, devem ser adicionados os seis parâmetros, conforme as linhas 1 a 6 do Quadro 3.1.

Exemplificando

Veja no Quadro 3.1 como é simples utilizar o *framework* RichFaces. Nesse código foi criado um recurso chamado *tooltip*. Ao mover o mouse sobre um determinado campo, aparece uma dica. Observe a tag `<rich:tooltip>` na linha 16, em que o valor "`Digite seu nome`" aparecerá quando o usuário mover o mouse sobre a caixa de texto criada na linha 13.

Quadro 3.1 | Exemplo de *tooltip* com RichFaces

```
1.  <!DOCTYPE html>
2.  <ui:composition
3.  xmlns="http://www.w3.org/1999/xhtml"
4.  xmlns:h="http://java.sun.com/jsf/html"
5.  xmlns:f="http://java.sun.com/jsf/core"
6.  xmlns:ui="http://java.sun.com/jsf/
7.                                  facelets"
8.  xmlns:a4j="http://richfaces.org/a4j"
9.  xmlns:rich="http://richfaces.org/rich"
```

```
10.  <h:head>
11.     <title>ToolTip</title>
12.  </h:head>
13.
14.  <h:body>
15.    <h:form>
           <h:outputText value="Nome: " />
           <h:inputText id="nome-cliente"
16.    value="" />
          <br/><br/>
          <h:commandButton id="botao-enviar"
17.    value="Enviar"/>
18.
19.  <rich:tooltip value="Digite seu nome "
       target="nome-cliente"/>
20.  <rich:tooltip value="Clique para submeter"
       target="botao-enviar"/>
     </h:form>
     </h:body>

     </ui:composition>
```

Fonte: elaborado pelo autor.

Ainda sobre a criação do componente de dica no Quadro 3.1, a ligação do *tooltip* com um determinado campo é feita por meio do `id` e `target`. Veja que na linha 17 a dica com o texto "`Clique para submeter`" foi ligada ao botão por meio do atributo `target`, que possui como valor o `id` do botão enviar. A Figura 3.1 ilustra o resultado do Quadro 3.1, quando o usuário coloca o mouse sobre o campo de digitação do nome.

Figura 3.1 | Resultado do código do Quadro 3.1

Fonte: elaborada pelo autor.

Outro recurso interessante no *framework* RichFaces é a facilidade da criação de painéis. Se estivéssemos usando HTML, CSS (*Cascading Style Sheet*) e JS (*JavaScript*), precisaríamos criar

muitas linhas de comandos para alcançar tais resultados. Veja no Quadro 3.2 o código necessário para gerar um painel *Collapsible*, ou seja, um painel que podemos esconder ou mostrar clicando sobre ele. Na linha 12 é especificado o componente da biblioteca que possui essa função, o atributo `header` especifica o texto que irá aparecer no Painel e `switchType` informa que a ação ocorrerá no lado do cliente.

Quadro 3.2 | Exemplo de painel Collapsible com RichFaces

```
    <!DOCTYPE html>
    <ui:composition
1.    xmlns="http://www.w3.org/1999/xhtml"
2.    xmlns:h="http://java.sun.com/jsf/html"
3.    xmlns:f="http://java.sun.com/jsf/core"
4.    xmlns:ui="http://java.sun.com/jsf/facelets"
5.    xmlns:a4j="http://richfaces.org/a4j"
6.    xmlns:rich="http://richfaces.org/rich" >

7.  <h:head>
8.  <title>Painel Collapsible RichFaces</title>
9.  </h:head>

10. <h:body>
11.    <h:form>
12.         <rich:collapsiblePanel
    header="Disciplinas" switchType="client">
13.       <h4>Já concluídas:</h4>
14.          <ul>
15.              <li>Algoritmos e técnicas de
    programação</li>
                 <li>Programação Orientada a
    Objetos</li>
16.              <li>Modelagem de dados</li>
                 <li>Programação em banco de dados</li>
17.              <li>Programação para web 1</li>
18.          </ul>
19. </rich:collapsiblePanel>
20.
21.       </h:form>
         </h:body>
22.
23. </ui:composition>
24.
```

Fonte: elaborado pelo autor.

Outro recurso muito utilizado são as caixas de seleção, também chamadas de *comboBox*. Veja no Quadro 3.3 como adicionar esse elemento. Omitimos a inclusão das referências para ganhar espaço. A caixa de seleção foi criada dentro de um painel, por isso, na linha 7 definimos um `rich:panel`. Na linha 8 foi adicionado um elemento do tipo cabeçalho e na linha 9 foi especificado o texto desse cabeçalho. A caixa de seleção é, de fato, criada nas linhas 11 e 12. Os cursos serão carregados a partir de uma *Manage Bean* chamada *curso.MB.java* (não se esqueça que a *Manage Bean* tem que ser configurada no arquivo *faces-config.xml*).

Quadro 3.3 | Exemplo de caixa de seleção com RichFaces

```
1.  <!-- Incluir as referências antes -->
2.
3.  <h:head>
4.  <title>Caixa de seleção RichFaces</title>
    </h:head>
5.
6.
7.  <h:body>
8.    <h:form>
9.      <rich:panel style="width:220px;">
10.       <f:facet name="header">
11.         <h:outputText value="Selecione um curso"
    />
12.       </f:facet>
13.
14.       <rich:select enableManualInput="true"
15.                    defaultLabel="Selecione uma
    opção">
16.         <f:selectItems value="#{cursoMB.cursos}"
    />
          </rich:select>
        </rich:panel>
      </h:form>
    </h:body>
```

Fonte: elaborado pelo autor.

Para ver a caixa de seleção do Quadro 3.3 funcionando, você precisa criar uma *Manage Bean* chamada *curso.MB.java*. Acesse no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u3/s1/codigo_pwe_II.PDF> ou veja por meio do QR Code a implementação que usamos para esse caso, que também foi usado para o código no Quadro 3.5..

A biblioteca RichFaces possui uma grande quantidade de componentes que podem ser usados. Veja no Quadro 3.4 alguns deles.

Quadro 3.4 | Alguns componentes do RichFaces

Categoria	Componente	Descrição
Painel	`rich:tabPanel`	Usado para criar "abas" na página.
Painel	`rich:popupPanel`	Usado para exibir mensagens em um pop-up.
Entrada	`rich:calendar`	Usado para adicionar um calendário.
Entrada	`rich:fileUpload`	Usado para adicionar um campo para upload de arquivos.
Menu	`rich:panelMenu`	Usado para criar menus *collapsible*.
Menu	`rich:dropDownMenu`	Usado para criar menus com hierarquia.

Fonte: elaborado pelo autor.

Pesquise mais

Na documentação oficial do RichFaces é possível encontrar vários exemplos da utilização dos componentes. Acesse e explore os exemplos. Disponível em: <https://bit.ly/2KhYPXN>. Acesso em: 1 ago. 2018.

Não poderia falar do RichFaces sem trazer um exemplo de um dos recursos mais importantes da biblioteca, o Ajax. Através desses componentes é possível renderizar (atualizar) determinadas partes da página, ao invés da página toda. Por exemplo, as tags `<a4j:commandButton>` e `<a4j:commandLink>` são utilizadas para enviar uma solicitação Ajax no clique do mouse. A tag `<a4j:ajax>` permite que você adicione a funcionalidade Ajax aos componentes JSF padrão e envie a solicitação Ajax em um evento *JavaScript* escolhido, como *keyup* ou *mouseover*. Veja no Quadro 3.5 um exemplo que utiliza o Ajax. Observe que na linha 9 o componente `<a4j:ajax>` está dentro de um *inputText* e está ligado ao evento *keyup,* ou seja, toda vez que uma tecla for "solta", a ação será executada. O atributo render informa "onde" será feita a atualização. Veja que na linha 13 o componente *outputText* possui id "saida", ou seja, o mesmo valor do *render* e, portanto, ele que receberá as atualizações.

Quadro 3.5 | Exemplo de caixa de seleção com RichFaces

```
1.  <!-- Incluir as referências antes -->

2.  <h:head>
3.    <title>Ajax - RichFaces</title>
4.  </h:head>

5.  <h:body>
6.    <h:form>

7.  <h:outputText value="Curso: " />

8.      <h:inputText value="#{cursoMB.curso}">
9.        <a4j:ajax event="keyup" render="saida" />
10.     </h:inputText>

11.     <p>
12.       Saída Ajax:
13.       <h:outputText value="#{cursoMB.curso}"
    id="saida" />
14.     </p>
15.   </h:form>
16. </h:body>
```

Fonte: elaborado pelo autor.

Reflita

Com o surgimento dos *frameworks*, que facilitam a implementação de componentes com extensão de recursos como Ajax, haverá a necessidade de um conhecimento mais profundo e específico em Ajax e em linguagem *JavaScript*? Os desenvolvedores terão a necessidade de se especializar?

O RichFaces possui uma série de *skins* (aparências) pré-configuradas. A escolha o perfil a ser utilizado pode ser feita através de uma configuração no arquivo *web.xml* que fica dentro da pasta *WebContent* >> *WEB-IN*. Para isso, acrescenta-se ao arquivo:

```
<context-param>
    <param-name>org.richfaces.skin</param-name>
    <param-value>skin_name</param-value>
</context-param>
```

Onde `skin_name` pode ser: DEFAULT, emeraldTown, blueSky, wine, japanCherry, ruby, classic, deepMarine.

Pesquise mais

A biblioteca de componentes do RichFaces é muito extensa e interessante, permitindo que as páginas criadas sejam muito mais dinâmicas e com uma visualização mais harmônica. Para conhecer mais a fundo os componentes existentes e suas implementações, acesse:

JAVATPOINT. **RichFaces Tutorial**. Javatpoint. [S.l.], [s.d.]. Disponível em: <https://www.javatpoint.com/richfaces-tutorial>. Acesso em: 1 ago. 2018.

Como você pode ver, a biblioteca RichFaces possui uma vasta quantidade de componentes que podem ser usados para criar telas mais dinâmicas. Para finalizar nossa seção, vamos falar de mais duas bibliotecas que podem ser usadas, começando pela OpenFaces. Ela oferece recursos como criação de gráficos através da tag `<o:chart>`, calendários `<o:calendar>`, campos de confirmação `<o:confirmation>`, entre outros. Embora esta biblioteca seja bem interessante, no momento seu site oficial <http://www.openfaces.org> (acesso em: 1 ago. 2018) não se encontra disponível, não sendo possível fazer o download do recurso. Este pode ser um problema temporário ou ela pode ter sido removida.

O *framework* ICEfaces é uma biblioteca de componentes, desenvolvido pela ICESoft, que nesse momento está na versão 4.X. O ICEfaces é um *framework* RIA (*Rich Internet Application*) de código aberto para Java EE. Tem como diferencial funcionar em diferentes plataformas, desde desktops a smartphones. Sua configuração em um projeto demanda um pouco mais de trabalho que o RichFaces. Primeiro, acesse o endereço <https://bit.ly/2MaW9gm> (acesso em: 1 ago. 2018), faça o download do arquivo ICEfaces-4.2.0-bin.zip, (*binary bundle*) e descompacte-o em um diretório. Crie um projeto *Web Dynamic Web* e copie os arquivos .jar da pasta lib em que foi feito download para a pasta lib do projeto *WebContent >> WEB-INF >> lib*. O ICEfaces exige a criação de um template específico para seu funcionamento, por isso, dentro da pasta *WebContent >> WEB-INF* crie uma

pasta chamada *templates*. Dentro dela, crie um arquivo chamado *template.xhtml* com o código do Quadro 3.6, que será usado como um "molde" para a página. A tag `ui:insert` especifica que outra página será inserida nesse local.

Quadro 3.6 | Arquivo *template.xhtml*

```
<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html" >
    <h:head>
       <title><ui:insert
name="title">Home</ui:insert></title>
    </h:head>

    <h:body>

         <div id="content">
              <ui:insert name="content">
                <!-- o conteúdo da página será inserido
aqui -->
              </ui:insert>
         </div>

    </h:body>
</html>
```

Fonte: elaborado pelo autor.

Agora basta criar as páginas .xhtml, dentro da pasta WebContent, inserindo a referência ao framework e usando as tags ICEfaces. Veja um exemplo no Quadro 3.7, no qual foram criados três campos com suas respectivas máscaras. Observe que na linha 9 a página foi ligada ao template criado anteriormente. O parâmetro name na linha 10 precisa ter o mesmo nome usado no template.xhtml. Criamos um painel para organizar o layout, mas o principal está nas linhas 15, 17 e 19, em que usamos um componente <ace:maskedEntry> parar criar campos com máscaras.

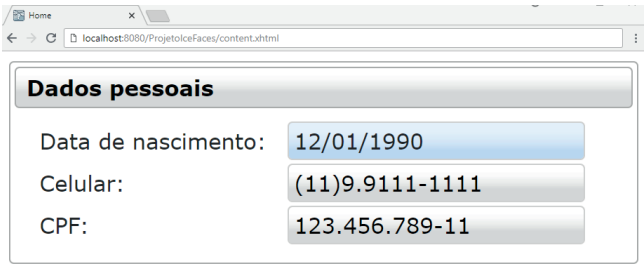
Quadro 3.7 | Exemplo de campos com máscaras com ICEfaces

```
1.  <ui:composition
2.          xmlns="http://www.w3.org/1999/xhtml"
3.          xmlns:h="http://xmlns.jcp.org/jsf/html"
4.          xmlns:f="http://xmlns.jcp.org/jsf/core"
5.          xmlns:c="http://xmlns.jcp.org/jsp/jstl/core"
6.          xmlns:ace="http://www.icefaces.org/
    icefaces/components"
            xmlns:icecore="http://www.icefaces.org/
    icefaces/core"
7.          xmlns:ui="http://xmlns.jcp.org/jsf/
    facelets"
8.          template="/WEB-INF/templates/template.
    xhtml">

9.    <ui:define name="content">
        <h:form id="form">
10.       <ace:panel id="dadosPanel" header="Dados
11. pessoais">
12.         <h:panelGrid id="inputGrid" columns="2"
    width="100%">

13.           <h:outputLabel id="dataLabel"
    for="dataInput" value="Data de nascimento:"/>
14.           <ace:maskedEntry id="dataInput"
    value="#{contatoMB.data}" mask="99/99/9999" />

15.           <h:outputLabel id="telefoneLabel"
    for="telefoneInput" value="Celular:"/>
              <ace:maskedEntry id="telefoneInput"
16. value="#{contatoMB.celular}" mask="(99)9.9999-
    9999" />

17.           <h:outputLabel id="cpfLabel"
    for="cpfInput"
    value="CPF:"/>
              <ace:maskedEntry id="cpfInput"
18. value="#{contatoMB.cpf}" mask="999.999.999-99" />
          </h:panelGrid>
19.     </ace:panel>
      </h:form>
20.   </ui:define>
21. </ui:composition>
22.
23.
24.
```

Fonte: elaborado pelo autor.

Mesmo não criando a Manage Bean contatoMB, nesse caso, é possível visualizar o resultado na Figura 3.2.

Figura 3.2 | Resultado dos Quadros 3.6 e 3.7



Fonte: elaborada pelo autor.

Pesquise mais

Na documentação oficial do ICEfaces é possível encontrar vários exemplos da utilização dos componentes. Acesse e explore os exemplos. Disponível em: <https://bit.ly/2Khnv2z>. Acesso em: 1 ago. 2018.

Agora você está apto a criar páginas com um design mais dinâmico e sofisticado para seus clientes!

Sem medo de errar

Agora que você aprendeu usar algumas bibliotecas, pode atualizar o sistema de cadastro do seu cliente. Como ele tem urgência na atualização, é possível optar pelo *framework* RichFaces, pois basta adicionar as bibliotecas na pasta *lib* do projeto, utilizar as tags que desejar e, o melhor, não é necessário atualizar as classes de modelo e de controle, pois a alteração acontecerá somente na camada de visão. Na verdade, essa é uma das grandes vantagens de utilizar o JSF, pois a divisão das camadas é feita pela própria estrutura de construção das pastas.

O atual cadastro do cliente possui os seguintes campos: código, nome, idade, telefone, e-mail, data de nascimento, CPF e endereço composto por rua, número, bairro, cidade, estado e

CEP. Uma ideia para a atualização é dividir as informações usando um *rich:collapsiblePanel* que separa os dados do endereço dos demais. Veja no Quadro 3.8 uma possível implementação para o formulário de cadastro.

Quadro 3.8 | Possível implementação da interface gráfica usando o RichFaces

```
<!DOCTYPE html>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
        xmlns:h="http://java.sun.com/jsf/html"
        xmlns:f="http://java.sun.com/jsf/core"
        xmlns:ui="http://java.sun.com/jsf/facelets"
        xmlns:a4j="http://richfaces.org/a4j"
        xmlns:rich="http://richfaces.org/rich">

<h:outputStylesheet>
  .iinput {
    max-width: 560px;
    overflow: hidden;
    text-overflow: ellipsis;
  }
         .rotulo{
          font-size: 14pt;
  }
</h:outputStylesheet>

        <h:head>
                <title>Cadastro de clientes</title>
        </h:head>

<h:body>
<h:form>
<rich:collapsiblePanel header="Dados gerais"
switchType="client">
<h:panelGrid columns="2">
        <h:outputText styleClass="rotulo" value="Código: " />
        <rich:inplaceInput defaultLabel="Clique para digitar
seu código" styleClass="iinput" />

        <h:outputText styleClass="rotulo" value="Nome: " />
        <rich:inplaceInput defaultLabel="Clique para digitar
seu nome" styleClass="iinput" />

        <h:outputText styleClass="rotulo" value="Idade: " />
        <rich:inplaceInput defaultLabel="Clique para digitar
sua idade" styleClass="iinput" />
```

```
        <h:outputText styleClass="rotulo" value="Telefone: " />
        <rich:inplaceInput defaultLabel="Clique para digitar
seu telefone" styleClass="iinput" />

        <h:outputText styleClass="rotulo" value="Email:" />
        <rich:inplaceInput defaultLabel="clique para digitar
seu email" styleClass="iinput" />

        <h:outputText styleClass="rotulo" value="Data de
nascimento: " />
        <rich:calendar style="width:200px" />

        </h:panelGrid>
</rich:collapsiblePanel>

<rich:collapsiblePanel header="Dados do endereço"
switchType="client">
        <h:panelGrid columns="2">

                <h:outputText styleClass="rotulo" value="Rua:
" />
                <rich:inplaceInput defaultLabel="Clique para
digitar a rua" styleClass="iinput" />

                <h:outputText styleClass="rotulo"
value="Número: " />
                <rich:inplaceInput defaultLabel="Clique para
digitar o número" styleClass="iinput" />

                <h:outputText styleClass="rotulo"
value="Bairro: " />
                <rich:inplaceInput defaultLabel="Clique para
digitar o bairro" styleClass="iinput" />

                <h:outputText styleClass="rotulo"
value="Cidade: " />
                <rich:inplaceInput defaultLabel="Clique para
digitar a cidade" styleClass="iinput" />

                <h:outputText styleClass="rotulo"
value="Estado:" />
                <rich:inplaceInput defaultLabel="Clique para
digitar o estado" styleClass="iinput" />

                <h:outputText styleClass="rotulo" value="CEP:
" />
                <rich:inplaceInput defaultLabel="Clique para
digitar o CEP" styleClass="iinput" />
```

```
        </h:panelGrid>
</rich:collapsiblePanel>

</h:form>
</h:body>

</ui:composition>
```

Fonte: elaborado pelo autor.

Que tal melhorar o código do Quadro 3.8 e colocar uma caixa de seleção no campo de estados? Use a sua criatividade e explore os componentes da biblioteca.

Avançando na prática

Formulário para pedidos

Descrição da situação-problema

Um restaurante de comida japonesa deseja criar um formulário de pedidos em seu sistema web para que seus clientes possam solicitar pedidos de produtos para entrega. Você foi contrato para tal tarefa e recebeu como pedido que as opções de comidas e bebidas sejam exibidas em duas abas diferentes para melhor usabilidade do usuário. Portanto, faça a implementação da tela e mostre para seus clientes.

Resolução da situação-problema

Para fazer essa implementação você pode recorrer ao componente *rich:tabPanel* do framework *RichFaces*. Veja no Quadro 3.9 como criar dois painéis usando esse componente.

Quadro 3.9 | Criando painéis com o RichFaces

```
<!DOCTYPE html>
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:a4j="http://richfaces.org/a4j"
      xmlns:rich="http://richfaces.org/rich">
```

```
        <h:head>
                <title>Cárdapio</title>
        </h:head>

<h:body>
<h:form>
        <rich:tabPanel>
                <rich:tab header="Alimentos">
                        <h1>Selecione as opções
desejadas</h1>
                        <!-- Criar menu -->
                </rich:tab>

                <rich:tab header="Bebidas">
                        <h1>Selecione as bebidas
desejadas</h1>
                        <!-- Criar menu -->
                </rich:tab>
        </rich:tabPanel>

</h:form>
</h:body>

</ui:composition>
```

Fonte: elaborado pelo autor.

Faça valer a pena

1. O *framework* RichFaces é a biblioteca de componentes JSF gratuita e uma das mais utilizadas, ideal para aplicações que utilizam Ajax (*Asynchronous Javascript and XML*, ou seja, *Javascript* Assíncrono e XML) e para manter um padrão visual nas aplicações criadas.

Em relação à arquitetura do framework RichFaces, assinale a alternativa que contém somente elementos referentes à tecnologia Ajax.

a) *a4j:ajax, a4j:commandButton, a4j:commandLink*.
b) *Ajax method, a4j:commandButton, ajax:new*.
c) *ajax:rich, a4j:ajax, a4j:commandButton*.
d) *rich:ajax, a4j:ajax, a4j:commandLink*.
e) *ajax:action, a4j:ajax, ajax:rich*.

2. As bibliotecas de componentes do ICEfaces utilizam o que há de mais recente em HTML5, bem como técnicas de design responsivo, o que permite que uma única página possa ser visualizada de forma adaptada ao dispositivo utilizado, como smartphones, tablets e computadores.

Assinale a alternativa que apresenta as opções de componentes do ICEfaces.

a) *ui:insert, ice:menu, ice:panel.*
b) *ace:panelGrid, ice:maskedEntry, ui:insert.*
c) *ace:maskedEntry, ace:menu, ui:insert.*
d) *ace:maskedEntry, ace:panel, ace:menu.*
e) *ice:maskedEntry, ice:panel, ice:menu.*

3. Com a biblioteca RichFaces é possível construir, de maneira rápida e fácil, interfaces para o usuário, permitindo um maior dinamismo em interações com o usuário, sem a necessidade de utilizar trechos de códigos escritos em JavaScript.

Com relação às vantagens para o desenvolvimento de aplicações web, utilizando a biblioteca de componentes RichFaces, analise as sentenças a seguir e assinale V para Verdadeiro ou F para Falso.
I - ( ) Permite criar visualizações complexas de aplicativos utilizando componentes prontos para uso.
II - ( ) A biblioteca RichFaces pode ser implementada juntamente ao arquivo persistence.xml em aplicações JSF.
III - ( ) Possibilidade de escrever seus próprios componentes personalizados com suporte a Ajax.
IV - ( ) As extensões dos recursos de manipulação de recursos do JSF podem gerar imagens, sons, planilhas do Microsoft Excel e muito mais durante o tempo de execução.
V - ( ) O RichFaces fornece um recurso de skins que permite definir e gerenciar diferentes esquemas de cores e outros parâmetros da aparência.

Assinale a alternativa correspondente à sequência informada.

a) I – V; II - F; III – V; IV – V; V – V.
b) I – V; II – V; III – F; IV – F; V – V.
c) I – F; II – F; III – V; IV – V; V – F.
d) I – F; II – V; III – V; IV – F; V – V.
e) I – V; II – V; III – V; IV – V; V – F.

Seção 3.2

Framework PrimeFaces, GisFaces e Gmaps4jsf e exemplos

Diálogo aberto

Prezado estudante, você já se deparou com a enorme variedade de notebooks existentes para venda? São diversos fabricantes e montadoras de equipamentos, que distribuem dezenas de modelos diferentes. Mas considerando a estrutura básica de um notebook, todos eles possuem configurações muito parecidas nos quesitos: processador, memória, placa mãe e espaço em disco. O que mais muda de um fabricante para outro é a carcaça, ou seja, cada um tem seu estilo de equipamento. O mesmo acontece com os *frameworks*: a estrutura das páginas tem a mesma forma, mas a aparência e os recursos visuais são específicos de cada biblioteca utilizada, assim, permitindo que as páginas recebam visuais mais atraentes com base no *framework* que utiliza.

Um de seus clientes possui um sistema de cadastro antigo, criado por sua empresa, e gostaria de atualizá-lo com melhorias em seu visual, deixando-o moderno. Ele deseja que a tela de cadastro seja mais dinâmica, com componentes que proporcione uma experiência de usabilidade nova aos usuários. Após atualizar a interface gráfica de cadastro de clientes, agora você deve atualizar a de cadastro de fornecedores.

Nesta seção, você vai conhecer a biblioteca de componentes PrimeFaces, uma das mais conhecidas e utilizadas do mundo, e o *framework* Gmaps4jsf, que permite integrar o Google Maps às páginas web desenvolvidas com *Java Server Faces* (JSF).

Preparados para solucionar este desafio? Bons estudos!

Não pode faltar

Caro aluno, a utilização de *frameworks* permite não só personalizar as aplicações web que criamos, como também adicionar recursos ao desenvolvimento de páginas web com JSF.

Segundo Coelho (2013), a biblioteca de componentes PrimeFaces é considerada por muitos desenvolvedores a mais avançada do mercado e estruturada como um software livre (embora exista uma versão especial não livre) utilizado em JSF, com mais de 100 componentes disponíveis, como validação no lado do cliente, mecanismo de tema e muitos outros. Podemos ressaltar duas características interessantes dessa biblioteca: ela é leve - sua integração no projeto é feita com base em um único arquivo *.jar*, sem necessidade de configuração e sem dependências - e proporciona um design responsivo (PRIMEFACES, 2018).

Durante a produção desse livro, a biblioteca disponível para a comunidade em geral está na versão 6.0. Para adicionar o PrimeFaces ao projeto, é necessário apenas baixar o arquivo *.jar*, acessando o site <http://primefaces.org/downloads.html> (acesso em: 2 ago. 2018). Procure pela seção de *Community Downloads* e faça o download da versão mais atual disponível, nesse caso, escolhermos *primefaces-6.0.jar*. Após o download, basta copiar o arquivo .jar para a pasta *WEB-INF >> lib* do projeto. Para utilizar o PrimeFaces em uma página XHTML, basta referenciar a biblioteca, conforme o Quadro 3.10.

Quadro 3.10 | Inserção do *PrimeFaces*

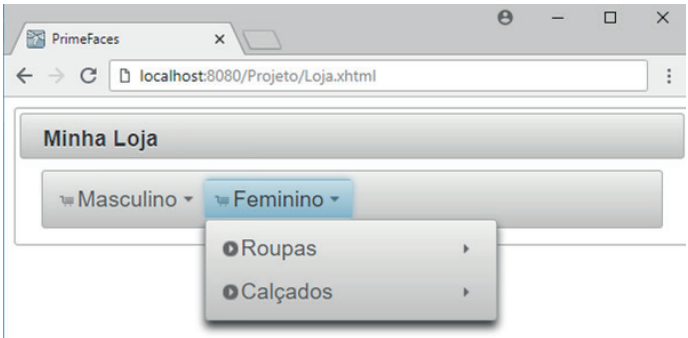
```
1. <html
2.         xmlns="http://www.w3.org/1999/xhtml"
3.         xmlns:h="http://java.sun.com/jsf/html"
4.         xmlns:p="http://primefaces.org/ui" >
5. </html>
```

Fonte: elaborado pelo autor.

Exemplificando

Vamos exemplificar o uso da biblioteca de componentes do PrimeFaces criando um menu com ícones dentro de um painel, conforme ilustra a Figura 3.3.

Figura 3.3 | Menu criado com PrimeFaces



Fonte: elaborado pelo autor.

Na criação desse tipo de componente você precisa se atentar à hierarquia dos menus e submenus, por exemplo, o submenu 1 deve ficar dentro da tag do menu 1. O submenu 1.1, deve ficar dentro da tag do submenu 1. A ideia é análoga à teoria dos subconjuntos da matemática. Veja no Quadro 3.11 o código para obter o resultado da Figura 3.3. Observe que foi criado um painel com 100% de tamanho, isso significa que o conteúdo dentro desse componente sempre ocupará 100% do tamanho da tela, independentemente de ser aberto em um desktop, smartphone ou tablet. O componente <p:menubar> na linha 10 abre a criação do menu. Da linha 11 até a 20 é criado o primeiro menu, que, por sua vez, possui dois submenus. Da linha 21 até a 31 é criado o segundo menu, também composto por dois submenus. Alguns componentes aceitam a inserção de ícones, por isso, adiciona-se esses elementos através do atributo icon. Para conhecer os ícones disponíveis, acesse <https://bit.ly/2ODC8Re>. Acesso em: 2 ago. 2018.

Quadro 3.11 | Criação de menu com PrimeFaces

```
1.  <html xmlns="http://www.w3.org/1999/xhtml"
2.        xmlns:h="http://java.sun.com/jsf/html"
3.        xmlns:f="http://java.sun.com/jsf/core"
4.        xmlns:p="http://primefaces.org/ui">
```

```
5.  <h:head>
6.        <title>PrimeFaces</title>
7.  </h:head>

8.  <h:body>
9.    <p:panel header="Minha Loja"
      style="width:100%">

10.     <p:menubar>
11.       <p:submenu label="Masculino" icon="ui-
          icon-cart">

12.         <p:submenu label="Roupas" icon="ui-icon-
            circle-triangle-e">
13.           <p:menuitem value="Blusas" url="#" />
14.           <p:menuitem value="Calças" url="#" />
15.         </p:submenu>

16.         <p:submenu label="Calçados" icon="ui-
            icon-circle-triangle-e">
17.           <p:menuitem value="Tênis" url="#" />
18.           <p:menuitem value="Sapatos" url="#" />
19.         </p:submenu>

20.       </p:submenu>

21.       <p:submenu label="Feminino" icon="ui-icon-
          cart">

22.         <p:submenu label="Roupas" icon="ui-icon-
            circle-triangle-e">
23.           <p:menuitem value="Blusas" url="#" />
24.           <p:menuitem value="Calças" url="#" />
25.         </p:submenu>
26.

27.         <p:submenu label="Calçados" icon="ui-
            icon-circle-triangle-e">
28.           <p:menuitem value="Tênis" url="#" />
29.           <p:menuitem value="Sapatos" url="#" />

30.         </p:submenu>

31.       </p:submenu>
32.     </p:menubar>
33.   </p:panel>
34. </h:body>
35. </html>
```

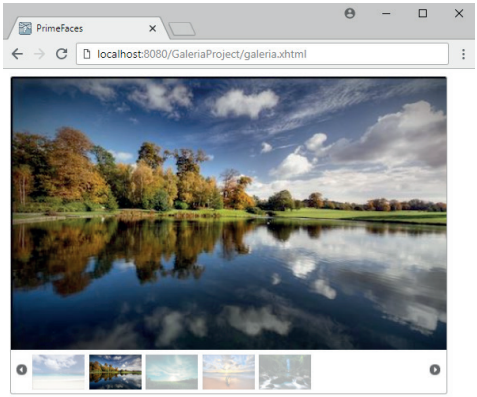
Fonte: elaborado pelo autor.

Reflita

Com a grande quantidade de frameworks no mercado, os que mais se destacam são aqueles que oferecem um pacote mais completo de componentes. Sabendo que o RichFaces e o PrimeFaces são os mais comuns e mais utilizados, qual a principal diferença entre os dois frameworks? Qual apresenta melhor estrutura?

Se você já criou uma galeria de fotos usando HTML, CSS e JavaScript sabe quanto trabalho foi necessário. O PrimeFaces oferece componentes para carregar diversas imagens de forma dinâmica. Veja na Figura 3.4 uma galeria criada com o componente `<p:galleria>`.

Figura 3.4 | Componente <p:galleria>



Fonte: elaborado pelo autor.

Assimile

Para criar a galeria de imagens, é preciso criar uma pasta chamada **resources** dentro de **WebContent** e, dentro dela, você pode criar outra com um nome qualquer para guardar as imagens.

Nesse projeto criamos uma pasta chamada *images* e copiamos 5 imagens para dentro dessa pasta, chamadas propositalmente de: natureza1.JPG, natureza2.JPG, ..., natureza5.JPG. Copie e cole

as imagens que deseja usar para dentro da pasta *WebContent* >> *resources* >> *images*, com o mesmo critério de nome, ou seja, os nomes devem ser iguais, sendo diferenciados apenas por um número (não utilize espaço e nem acento). Exemplo: "nomeX.jpg", em que X é um número inteiro sequencial.

Após copiar as imagens para o projeto, o próximo passo é criar a *Managed Bean* que carregará as imagens, via programação Java. Portanto, dentro de um pacote *control* crie uma nova classe chamada *ImagensMB* com o código do Quadro 3.12. Veja que dentro do construtor (linhas 7 a 12) foi criada uma lista que adiciona arquivos com o nome "naturezai.JPG", em que i é a variável de valor iterativo no *for*. É importante ressaltar que a extensão *.jpg* é diferente de *.JPG*. Você deverá usar conforme suas fotos estiverem salvas.

Quadro 3.12 | *Managed Bean ImagensMB.java*

```
1.  package control;
2.  import java.util.*;
3.  import javax.faces.bean.ManagedBean;
4.  @ManagedBean
5.  public class ImagensMB {
6.        private List<String> images;
7.        public ImagensMB() {
8.          images = new ArrayList<String>();
9.          for (int i = 1; i <= 5; i++) {
10.             images.add("natureza" + i + ".JPG");
11.         }
12.       }
13.   public List<String> getImages() {
14.       return images;
15.   }
16. }
```

Fonte: elaborado pelo autor.

Após a criação da *Managed Bean*, configure-a no arquivo *faces-config.xml*.

```
<managed-bean>
        <managed-bean-name>imagensMB</managed-bean-name>
```

```
            <managed-bean-class>control.
ImagensMB</managed-bean-class>
            <managed-bean-scope>request</managed-
bean-scope>
      </managed-bean>
```

Agora é hora de implementar os componentes da interface gráfica, através de um arquivo xhtml. Dentro da pasta *WebConfig*, crie um arquivo chamado *galeria.xhtml* com o código conforme o Quadro 3.13. Os comandos da linha 10 a 17 resultam na Figura 3.4. Veja que o componente `<p:galleria>` acessa a *Managed Bean* (MB) para carregar a lista de imagens e vincula cada uma delas a uma variável chamada *image* (linha 11), que é usada na linha 14 pelo componente `<p:graphicImage>` para acessar as imagens dentro da pasta *WebContent* >> *resources* >> *images*. Como essa pasta faz parte do destino padrão, no atributo *name* só é preciso informar o caminho que vem depois de *resources*, nesse caso, apenas a pasta *images*. Veja também, que a variável *image* vinculada às imagens da MB é acessada por `#{image}`.

Das linhas 18 a 22, criamos outro componente de imagens `<p:imageSwitch>` que as exibe com um "zoom" na troca entre elas. Para esse efeito também usamos `<ui:repeat>` para que as fotos fiquem repetindo enquanto a página estiver sendo exibida.

Quadro 3.13 | Arquivo *galeria.xhtml*

```
1.  <html xmlns="http://www.w3.org/1999/xhtml"
2.        xmlns:h="http://java.sun.com/jsf/html"
3.        xmlns:f="http://java.sun.com/jsf/core"
4.        xmlns:ui="http://java.sun.com/jsf/
    facelets"
5.        xmlns:p="http://primefaces.org/ui">

6.  <h:head>
7.        <title>PrimeFaces</title>
8.  </h:head>

9.  <h:body>

10. <p:galleria value="#{imagensMB.images}"
11.                 var="image"
12.                 panelWidth="500"
13.                 panelHeight="313">
```

```
15.       <p:graphicImage name="/images/#{image}"
16.                       alt="#{image}"
17.                       title="#{image}" />
          </p:galleria>
18.
19.       <p:imageSwitch effect="zoom">
              <ui:repeat value="#{imagensMB.images}"
          var="image">
                  <p:graphicImage name="/
          images/#{image}" />
20.       </ui:repeat>
21.       </p:imageSwitch>
22.
23.   </h:body>
24.   </html>
```

Fonte: elaborado pelo autor.

Pesquise mais

O *framework* PrimeFaces possui inúmeros componentes divididos em diversas categorias, por exemplo, para entrada de dados (input), botões (button), apresentação de dados (data), entre muitos outros. Acesse e explore essa rica biblioteca, disponível em: <https://bit.ly/2KjGIkd>. Acesso em: 2 ago. 2018.

Framework Gmaps4jsf

A biblioteca de Gmaps4jsf é um projeto da Google para o desenvolvimento de componentes para as aplicações JSF existentes, uma vez que esse projeto permite utilizar os recursos do Google Maps de forma simples e intuitiva (CODE, 2018).

Para adicionar a biblioteca Gmaps4jsf ao seu projeto, primeiramente você vai precisar acessar <https://code.google.com/archive/p/gmaps4jsf/downloads> (acesso em: 2 ago. 2018) e baixar o arquivo *gmaps4jsf-1.1.4.jar*. Após concluir o download, copie o arquivo baixado para o diretório *WebContent* >> *WEB-INF* >> *lib*.

Após inserir a biblioteca ao seu projeto, é preciso adicionar a *namespace* para o *framework* do Google, além de uma referência para um arquivo *JavaScript*, também do Google. Veja no Quadro 3.14, o código necessário para fazer a inserção do mapa e, após o Quadro, a explicação dos comandos e o resultado obtido (Figura 3.5).

Quadro 3.14 | Exemplo de código com Gmaps4jsf

```
1.  <html xmlns="http://www.w3.org/1999/xhtml"
2.        xmlns:f="http://java.sun.com/jsf/core"
3.        xmlns:h="http://java.sun.com/jsf/html"
4.        xmlns:ui="http://java.sun.com/jsf/
5.  facelets"
          xmlns:m="http://code.google.com/p/
6.  gmaps4jsf/">
7.
8.  <h:head>
          <title>google maps</title>
          <script src="http://maps.google.com/
9.  maps?file=api&amp;v=2&amp;"
10.             key="SUA_API_KEY"
11.             type="text/javascript">
12.       </script>
13. </h:head>

14. <h:body>
15.
16.       <h:form>
17.             <m:map id="map"
18.                    width="90%"
19.                    height="90%"
20.                    type="G_NORMAL_MAP">
                </m:map>
          </h:form>
21.
22. </h:body>
    </html>
```

Fonte: elaborado pelo autor.

Na linha 5 foi inserida a referência a *namespace* do *framework* do Google.

Na linha 8 foi adicionada a tag *JavaScript* apontando para um script do Google.

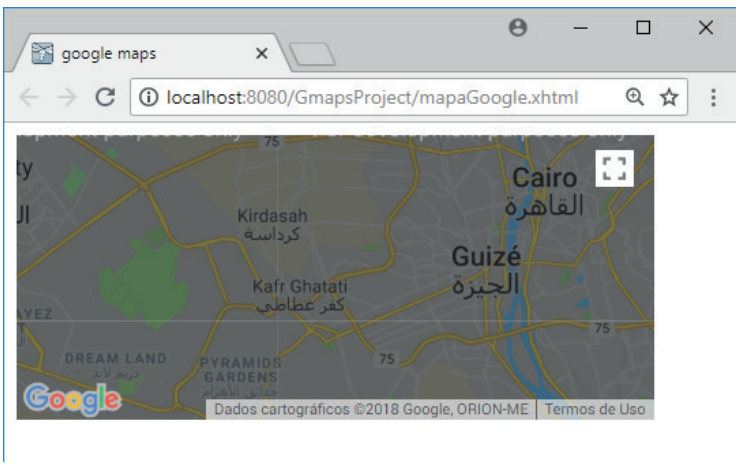
O parâmetro ***key*** na linha 9 é de extrema importância, pois para que você tenha acesso ao recurso do Google, é preciso fazer um cadastro, que gerará uma chave (*key*) para você usar nesse parâmetro. Para saber mais e criar seu registro, acesse <https://developers.google.com/maps/faq?csw=1#using-google-maps-apis> (acesso em: 2 ago. 2018) e procure pela seção *How do I get a new API key?*, onde terá um link para página de cadastro.

A inserção do mapa é feita pela tag `<m:map>` na linha 15, que especifica o tamanho do mapa na tela (veja que usamos em porcentagem para ser responsivo) e também o tipo. Além da opção usada, você pode configurar como *G_PHYSICAL_MAP* ou *G_HYBRID_MAP*.

Existem outros atributos que podem ser usados para personalizar o mapa, como:

- Zoom: o valor padrão é 11. Você pode alterá-lo conforme suas necessidades.
- Longitude e latitude: você pode apontar para coordenadas específicas. Exemplo: *<m:map latitude="-21.1979762986979" longitude="-47.8164911270142">*.
- address: o endereço para o qual você deseja apontar o mapa. Se usar esse atributo, não use os de longitude ou latitude. Exemplo: *<m:map address="São Paulo">*.

Figura 3.5 | Resultado obtido com Gmaps4jsf



Fonte: elaborado pelo autor.

Assimile

Para utilização desta biblioteca do Gmaps4jsf, é necessário possuir uma chave de API (API Key) do Google para poder integrar ao Google Maps. Para adquirir esta chave, você vai precisar acessar <http://code.google.com/apis/maps/signup.html> (acesso em: 18 jun. 2018) e gerar sua chave.

Outra opção para trabalhar com mapas é o *framework* GisFaces, criado por Chris Duncan. O GisFaces é um componente para mapeamento em JSF, utilizando uma API para JavaScript. Permite a utilização, local ou online, de mapas nas páginas web como camadas de serviços disponibilizadas publicamente. Sua documentação está disponível no endereço <https://www.gisfaces.com/> (acesso em: 2 ago. 2018). A documentação oficial recomenda sua utilização com o servidor *Glassfish* ou *Tomcat* (GISFACES, 2018).

Nessa seção você pode conhecer dois grandes *frameworks* de componentes utilizados com JSF. O PrimeFaces possui uma rica biblioteca dos mais diversos componentes, enquanto o Gmaps4jsf permite facilmente integrar mapas do Google na aplicação JSF. Aproveite o conhecimento para enriquecer suas interfaces gráficas.

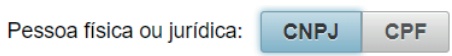
Sem medo de errar

Você está trabalhando na atualização do sistema de cadastro de um cliente e deve, nessa etapa, entregar a atualização da interface gráfica de cadastro de fornecedores em um curto espaço de tempo.

Nada melhor que recorrer a uma biblioteca de componentes para atualizar de forma rápida e eficaz. Dentre as opções disponíveis, a PrimeFaces é uma excelente ferramenta, pois possui diversos componentes.

Você pode utilizar o `<p:selectOneButton>`, que é uma alternativa para um *radio button*. Nesse componente apenas uma opção pode ser selecionada (Figura 3.6).

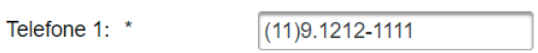
Figura 3.6 | Resultado do componente `<p:selectOneButton>`



Fonte: elaborado pelo autor.

Outro recurso interessante são os campos com máscaras (Figura 3.7). O PrimeFaces possui o componente `<p:inputMask>` para essa tarefa.

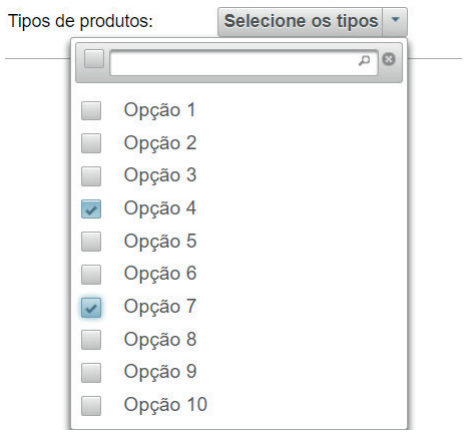
Figura 3.7 | Resultado do componente `<p:inputMask>`



Fonte: elaborado pelo autor.

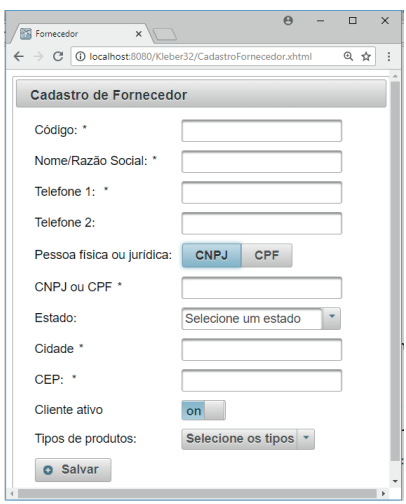
Outro componente muito interessante do PrimeFaces é o `<p:selectCheckboxMenu>`, que cria uma lista de opções do tipo checkbox, em que o usuário pode escolher diversas opções (Figura 3.8). Esse componente também já cria um campo para filtrar as opções.

Figura 3.8 | Resultado do componente `<p:selectCheckboxMenu>`



Veja na Figura 3.9 uma proposta para a tela do cliente.

Figura 3.9 | Interface gráfica para cadastro de fornecedores



Fonte: elaborado pelo autor.

Você pode conferir os códigos necessários para gerar a interface gráfica da Figura 3.9 Acesse no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u3/s2/ResolucaoSP.pdf> ou por meio do QR Code.
Lembre-se: é preciso adicionar a biblioteca! Além disso, no desenvolvimento de um software, os atributos *values="xxx"* devem ser substituídos por propriedades da *Managed Bean*.

Que tal implementar ainda mais a interface gráfica e surpreender seu cliente? Explore sua criatividade com os componentes do PrimeFaces.

Avançando na prática

Implementando gráfico no relatório

Descrição da situação-problema

Você trabalha em uma empresa de dados estatísticos e seu gestor precisa de um relatório do sistema, que é emitido em modo texto. Como será apresentado para um cliente, ele solicitou que você implemente neste relatório um gráfico de pizza para representar os dados dos estados da região Sudeste.

Como você implementará o gráfico no sistema? Qual *framework* utilizará?

Resolução da situação-problema

Para a inserção de gráficos em páginas web e relatório, o *framework* PrimeFaces é uma excelente opção, pois possui o componente `<p:chart>` destinado a esse fim.

Veja no Quadro 3.15 a inserção de um gráfico em uma página xhtml.

Quadro 3.15 | Componente `<p:chart>`

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:p="http://primefaces.org/ui">

<h:head>
      <title>Gráfico</title>
</h:head>
```

```
<h:body>
	<h:form>
		<p:chart type="pie"
			model="#{graficoMB.grafico1}"
			style="width:400px;
height:300px" />
	</h:form>
</h:body>
</html>
```

Fonte: elaborado pelo autor.

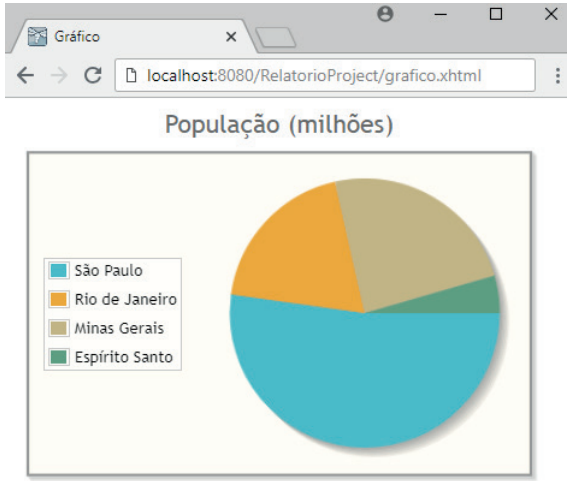
Para que o gráfico funcione, é preciso criá-lo usando comandos do próprio framework PrimeFaces em uma *Managed Bean*. Veja no Quadro 3.16 o código necessário para que o gráfico funcione, e na Figura 3.10, o resultado obtido.

Quadro 3.16 | Managed Bean GraficoMB.java

```
package control;

import javax.faces.bean.ManagedBean;
import org.primefaces.model.chart.PieChartModel;

@ManagedBean
public class GraficoMB {
	public GraficoMB() {
		createGrafico1();
	}

	private PieChartModel grafico1;

	private void createGrafico1() {
		grafico1 = new PieChartModel();
		grafico1.set("São Paulo", 45.34);
		grafico1.set("Rio de Janeiro", 16.72);
		grafico1.set("Minas Gerais", 20.87);
		grafico1.set("Espírito Santo", 3.9);
		grafico1.setTitle("População (milhões)");
		grafico1.setLegendPosition("w");
	}
	//propriedade acessada pela interface gráfica
	public PieChartModel getGrafico1() {
		return grafico1;
	}
}
```

Fonte: elaborado pelo autor.

Figura 3.10 | Gráfico gerado



Fonte: elaborado pelo autor.

Faça valer a pena

1. A biblioteca de componentes PrimeFaces é considerada por muitos a mais avançada do mercado e estruturada como um software livre popular, utilizado em JSF (Java Server Faces), com mais de 100 componentes disponíveis, como validação no lado do cliente, mecanismo de tema e muitos outros componentes.

Com referência ao PrimeFaces, assinale a alternativa correta.

a) A composição de design do PrimeFaces tem base em princípios pouco definidos.
b) Os componentes são criados utilizando o princípio de design, e a complexidade fica para a instalação do usuário.
c) Uma das vantagens de utilizar o framework PrimeFaces é a questão dos temas, em que cada desenvolvedor somente pode criar o seu.
d) Possui uma extensa participação da comunidade de desenvolvedores que fornece novas ideias, relatórios de falhas, atualizações e feedback dos usuários.
e) O PrimeFaces, assim como o RichFaces, utiliza somente o JavaScript em sua estrutura de componentes.

2. O *framework* PrimeFaces possui uma grande facilidade de configuração, pois basta fazer o download do arquivo *.jar*, copiá-lo para dentro da pasta *WebContent* >> *WEB-INF* >> *lib*, e adicionar a referência a *namespace* dentro do arquivo xhtml. Diferente de outras bibliotecas, que utilizam várias outras dependências e por isso precisam usar vários arquivos *.jar*.
A respeito dos componentes do *framework* PrimeFaces, avalie como verdadeira ou falsa cada uma das afirmações.
I – ( ) Para desenvolver um menu, basta utilizar as tags **<p:menubar>** e **<p:submenu>** para criar os itens do menu em diversos níveis.
II – ( ) Através do componente **<p:galleria>**, é possível criar uma galeria de imagens.
III – ( ) O componente **<p:selectOneButton>** possibilita ao desenvolvedor criar um conjunto de opções através de botões, dentre os quais o usuário só poderá escolher uma opção.

a) I – V; II – V; III – V.
b) I – F; II – V; III – V.
c) I – V; II – V; III – F.
d) I – F; II – V; III – F.
e) I – F; II – F; III – V.

3. A Google é um player de destaque no ramo de TI, atuando em áreas como mecanismo de busca e banco de dados na nuvem e oferecendo diversas API. A biblioteca de Gmaps4jsf é um dos projetos dessa empresa e oferece componentes que permitem utilizar os recursos do Google Maps de forma simples e intuitiva (CODE, 2018).
A respeito do framework Gmaps4jsf, analise as afirmações e escolha a opção correta.
I – A utilização da tag **<script>** é opcional para inserção de mapas através do Gmaps4jsf.
II – O atributo *key* dentro da tag **<script>** deve ser preenchido com um valor fornecido pelo próprio Google.
III – Um mapa pode ser inserido em uma página através da tag **<m:map>**.
IV – O atributo *address* é usado para especificar um local a ser exibido quando o mapa for carregado e deve estar dentro da tag **<script>**.

a) Somente as afirmações III e IV são verdadeiras.
b) Somente as afirmações II, III e IV são verdadeiras.
c) Somente as afirmações II e III são verdadeiras.
d) Somente a afirmação II é verdadeira.
e) Somente a afirmação III é verdadeira.

Seção 3.3

Framework Tomahawk, HighFaces e BooFaces

Diálogo aberto

Prezado aluno, você já deve ter se deparado com diversos modelos de carros pertencentes a diversos fabricantes. Dentre os modelos há também versões, por exemplo, um carro zero pode possuir a opção básica, a intermediária e uma mais completa. O que difere entre as versões são os adicionais presentes em cada carro. Da mesma maneira, em tecnologia da informação, temos diferentes tipos de *frameworks* disponíveis no mercado, que aumentam o desempenho no desenvolvimento, além de enriquecer os recursos.

Lembre-se de que você possui uma empresa de desenvolvimento de sistemas web que foi procurada por um antigo cliente que deseja atualizar seu sistema. Este sistema tem cadastro de clientes, fornecedores e produtos antigos. Após implementar o *framework* RichFaces, atualizar a tela de cadastros clientes e modernizar os campos do cadastro de funcionários com o *framework* PrimeFaces, chegou a hora de atualizar a tela de cadastro de produtos, deixando o site mais dinâmico. Agora você precisa apresentar a última opção de personalização ao seu cliente, utilizando um *framework*, com a customização dos campos de textos e calendário para data de nascimento.

Nesta seção você terá uma introdução ao *framework* Tomahawk, que permitirá a customização de componentes JSF, ao *framework* HighFaces, que permitirá adicionar recursos de gráficos, e ao *framework* BooFaces, que permitirá adicionar recursos para criação de aplicativos de forma rápida e simples. Com este conteúdo, você conseguirá resolver o problema acima.

Pronto para começar?

Não pode faltar

O BootsFaces é um poderoso e leve *framework* JSF baseado em Bootstrap 3 e jQuery UI (*User Interface*, ou seja, *interface* com o usuário), que permite desenvolver aplicativos

corporativos *front-end* com rapidez e facilidade (BOOTSFACES, 2018). Conforme relatado na documentação oficial, os recursos responsivos do BootsFaces permitem projetar apenas uma versão de um projeto de site, que se ajustará a smartphones, tablets pequenos e grandes e telas de área de trabalho, assim, as páginas são adaptadas automaticamente em todos os dispositivos, desde móveis a computadores (THECODER4, 2018).

A instalação do framework BootsFaces é bem simples para ser utilizado com o JSF. Ao acessar o site <https://bit.ly/2vxOFwU> (acesso em: 3 ago. 2018), você conseguirá realizar o download, podendo ser da versão com temas, com tema padrão do Bootstrap ou a versão do GitHub. Nesse projeto optamos por baixar a versão com temas (*Download Bootsfaces (with Themes)*). Após baixado, basta adicionar o arquivo à pasta lib de seu projeto.

Para incluir a biblioteca do BootsFaces em um projeto, adicionamos a referência do *framework* no início da página, como no Quadro 3.17.

Quadro 3.17 | Referenciando o BootsFaces em uma página

```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml"
3.       xmlns:h="http://java.sun.com/jsf/html"
4.       xmlns:f="http://java.sun.com/jsf/core"
5.       xmlns:b="http://bootsfaces.net/ui" >
```

Fonte: <https://bit.ly/2LV8tEP>. Acesso em: 3 ago. 2018.

Como o BootsFaces faz uso de elementos HTML e propriedades CSS, exigidos pela documentação do HTML5 e pela referência citada *xmlns:b="http://bootsfaces.net/ui"*, é necessário utilizar a declaração <!DOCTYPE html>, permitindo o uso de tags do BootsFaces nas páginas JSF, por isso, na linha 1 do Quadro 3.17 foi adicionada essa tag.

Outro diferencial desse *framework* é a quantidade de temas disponíveis para serem aplicados, bastando adicionar a referência no arquivo web.xml.

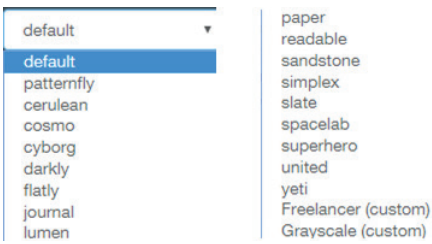
Assimile

O framework BootsFaces possui vários temas pré-definidos. Para utilizá-los, é preciso adicionar o seguinte código no arquivo web.xml:

```
<context-param>
   <param-name>BootsFaces_THEME</param-name>
    <param-value>tema</param-value>
</context-param>
```

Na tag `<param-value>` deve ser usada uma das opções de tema disponíveis na Figura 3.11.

Figura 3.11 | Opções para configuração de tema



Fonte: <https://bit.ly/2LU6zoQ>. Acesso em: 3 ago. 2018.

Para que os componentes funcionem, é indicado criar a página dentro de um *container*, para isso usa-se a tag `<b:container>`. Este container deve vir dentro de um formulário `<h:form>`, pois ela é obrigatória para submissão de dados. Veja um exemplo no Quadro 3.18. Na linha 11 o container foi aberto e na linha 21, fechado. Foi usado o atributo *fluid="true"* para que o conteúdo ocupe toda a extensão da tela. Na linha 12, foi aberto um componente `<b:jumbotron>`, que cria um quadro com bordas arredondadas. Na linha 16 foi criado um link e aplicado um estilo de botão do Bootstrap. Nesse comando, os atributos indicam que: *btn* = informa que é um botão, *btn-lg* = especifica o tamanho e *btn-info* = especifica o estilo a ser usado.

Quadro 3.18 | Exemplo do uso da tag `<b:container>`

```
1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml"
```

```
3.          xmlns:h="http://java.sun.com/jsf/html"
4.          xmlns:f="http://java.sun.com/jsf/core"
5.          xmlns:b="http://bootsfaces.net/ui">

6.  <h:head>
7.    <title>BootsFaces</title>
8.  </h:head>

9.  <h:body>
10.   <h:form>
11.     <b:container fluid="true">
12.       <b:jumbotron>
13.         <h2>KLS 2.0</h2>
14.         <p>Programação para web 2 - JSF</p>
15.         <p>
16.           <a href="link" class="btn btn-lg btn-
info">
17.             Veja mais. »
18.           </a>
19.         </p>
20.       </b:jumbotron>
21.     </b:container>
22.   </h:form>
23. </h:body>
24. </html>
```

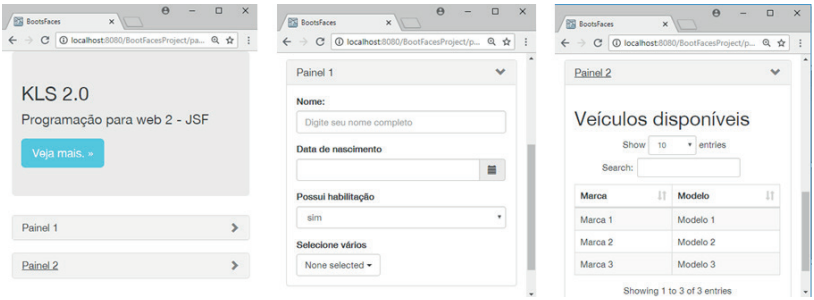
Fonte: elaborado pelo autor.

Pesquise mais

Para conhecer mais sobre layouts e estruturas responsivas utilizando o Bootstrap, acesse o site <https://bit.ly/2wyh6P9> (acesso em: 3 ago. 2018). Assim, você terá mais informações para criar páginas web e saberá as vantagens de sua utilização nos dias de hoje. Você também pode conferir exemplos de todos os componentes do *framework* BootsFace em: <https://bit.ly/2n9kXL3>. Acesso em: 19 ago. 2018.

Vamos explorar mais um exemplo com esse importante framework. Veja a Figura 3.12, que foi gerada pelo código no Quadro 3.19.

Figura 3.12 | Interface gráfica criada com o *framework* BootsFaces



Fonte: captura de tela do software Eclipse, elaborada pelo autor.

Para alcançar o resultado da Figura 3.12 foram usados dois painéis, um para os dados pessoais e outro para carregar dados de uma *Managed Bean*. O primeiro painel foi criado entre as linhas 20 e 34 (Quadro 3.19). Nele foram usados componentes de entrada de texto (b:inputText), de seleção de data (b:datepicker), caixa de opções para selecionar uma opção (b:selectOneMenu) e caixa para selecionar várias opções (b:selectMultiMenu). Todos esses itens são familiares a você de outros *frameworks* que já foram apresentados.

O painel 2 foi criado entre as linhas 35 e 41 e dentro dele foi usado um componente muito interessante para trabalhar com exibição de dados armazenados em banco de dados, o `<b:dataTable>`. Dentro desse componente, cada coluna da tabela é especificada no componente `<b:dataTableColumn>`. O grande desafio desses dois componentes é compreender o uso do atributo *value*, que permitirá carregar os dados. No `dataTable` o *value* corresponde a propriedades de uma Managed Bean (que, no nosso caso, chama-se CarrosMB.java), enquanto que no `dataTableColumn` o *value* corresponde a propriedades diretas da classe modelo, que, no nosso caso, chama-se Carro.java, porém é acessada através da variável "car" passando pela Managed Bean. Você pode verificar os códigos das classes acessando o QR Code a seguir.

Quadro 3.19 | Exemplo *framework* BootsFaces

```
1. <!DOCTYPE html>
2. <html xmlns="http://www.w3.org/1999/xhtml"
3.       xmlns:h="http://java.sun.com/jsf/html"
4.       xmlns:f="http://java.sun.com/jsf/core"
5.       xmlns:b="http://bootsfaces.net/ui">
```

```
6.  <h:head>
7.    <title>BootsFaces</title>
8.  </h:head>

9.  <h:body>
10.   <h:form>

11.     <b:container fluid="true">
12.       <b:jumbotron>
13.         <h2>KLS 2.0</h2>
14.         <p>Programação para web 2 - JSF</p>
15.         <p>
16.           <a href="link" class="btn btn-lg btn-
    info">
17.             Veja mais. »</a>
18.         </p>
19.       </b:jumbotron>
20.     <b:panel id="p1" title="Painel 1" >
21.       <b:inputText
22.                   placeholder="Digite seu nome
    completo"
                      label="Nome:" />
23.       <b:datepicker label="Data de nascimento" />
24.       <b:selectOneMenu label="Possui
25.   habilitação" >
            <f:selectItem itemValue="sim"
26.   itemLabel="sim" />
            <f:selectItem itemValue="nao"
    itemLabel="não" />
          </b:selectOneMenu>
27.       <b:selectMultiMenu label="Selecione
    vários" >
            <f:selectItem itemValue="op1"
28.   itemLabel="Habilitação para moto" /
29.         <f:selectItem itemValue="op2"
30.   itemLabel="Habilitação para carro" />
            <f:selectItem itemValue="op2"
31.   itemLabel="Habilitação para caminhão"/>
          </b:selectMultiMenu>
32.     </b:panel>

33.     <b:panel title="Painel 2" >
34.       <h2>Veículos disponíveis</h2>

35.
36.       <b:dataTable value="#{carrosMB.carros}"
    var="car">
```

```
37.             <b:dataTableColumn value="#{car.marca}" />
                <b:dataTableColumn value="#{car.modelo}" />
            </b:dataTable>
38.     </b:panel>
39.
40.   </b:container>
41. </h:form>
    </h:body>
42. </html>
43.
44.
45.
```

Fonte: elaborado pelo autor.

Você pode verificar uma opção de implementação para a Managed Bean CarrosMB.java Acesse no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u3/s3/Carro.pdf> ou por meio do QR Code.

Você pode verificar uma opção de implementação para a classe Carro.java Acesse no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u3/s3/CarrosMB.PDF> ou por meio do QR Code.

Framework Tomahawk

A Apache Software Foundation possui um projeto chamado MyFaces, destinado a criar componentes específicos para o JSF. Dentre todos os projetos do MyFaces, o *framework* Tomahawk é um dos principais (MYFACES, 2012).

Para incluir o *framework* Tomahawk em uma aplicação, é necessário acessar <https://bit.ly/2AISwgm> (acesso em: 6 ago. 2018) e realizar o download da biblioteca. Nesse livro estamos usando o arquivo *MyFaces Tomahawk 1.1.14 for JSF 2.0 (zip)*. Após realizar o download, descompacte o arquivo baixado e copie todos os arquivos que estão dentro da pasta lib do arquivo descompactado para a pasta *WEB-INF >> lib* do seu projeto. A documentação oficial orienta copiar somente o arquivo *tomahawk.jar*, entretanto, com o

servidor WildFly tivemos que adicionar todos os arquivos, porém, não foi preciso fazer nenhuma configuração extra no arquivo *web.xml*, o que seria necessário na utilização de outros servidores.

Para usar os componentes do Tomahawk, é preciso adicionar a *namespace*, conforme código no Quadro 3.20. Veja que os componentes serão atribuídos ao prefixo "t".

Quadro 3.20 | Referenciando o Tomahawk em uma página

```
1. <html xmlns="http://www.w3.org/1999/xhtml"
2.       xmlns:h="http://java.sun.com/jsf/html"
3.       xmlns:f="http://java.sun.com/jsf/core"
4.       xmlns:t="http://myfaces.apache.org/
tomahawk" >
```

Fonte: elaborado pelo autor.

Exemplificando

Para exemplificar o uso do framework Tomahawk vamos construir a tela da Figura 3.13.

Figura 3.13 | Exemplo de utilização do Tomahawk

Fonte: captura de tela do software Eclipse, elaborada pelo autor.

Para gerar a tela da Figura 3.14 foi usado o código no Quadro 3.21, veja que criamos uma página combinando componentes do *framework* Tomahawk com o JSF padrão. Das linhas 7 a 16, criamos algumas configurações CSS para aplicar na página, com atributos que o Tomahawk permite. Usamos o componente

<*t:panelTabbedPane*> para criar abas na página (linha 20). Dentro dele configuramos quatro atributos:

(i) `selectedIndex:` especifica qual aba ficará ativa quando a página for carregada no navegador.

(ii) `activeTabStyleClass:` aplica um determinado estilo CSS na aba que estiver ativa, por isso, configuramos a classe "ativa" na linha 8.

(iii) `cellpadding:` esse atributo permite configurar um espaçamento entre o texto da aba e as bordas, a fim de melhorar o layout visual.

(iv) `styleClass:` permite especificar uma classe de estilos CSS para todas as abas.

Na linha 24 abrimos a primeira aba e na linha 44 a fechamos, isso significa que tudo que estiver entre essas linhas será exibido nessa aba. Na linha 29 usamos o componente <*t:selectManyCheckbox*>, que permite selecionar várias opções através de caixas do tipo *checkbox*. Para criar as caixas de opções, usamos o componente <*t:selectBooleanCheckbox*>.

Na linha 38 criamos outro esquema para seleção, agora usando o componente <t:selectOneRadio>, que permite selecionar uma única opção dentre várias. Veja que dentro desse componente, que é do *framework* Tomahawk, usamos tags do JSF padrão (<*f:selectItem*>).

Das linhas 45 a 50 criamos as outras abas, porém contendo somente um título de conteúdo. Você pode conferir o resultado desse código na Figura 3.13.

Quadro 3.21 | Implementação de página JSF com Tomahawk

```
1.          <html xmlns="http://www.w3.org/1999/xhtml"
2.         xmlns:h="http://java.sun.com/jsf/html"
3.         xmlns:f="http://java.sun.com/jsf/core"
4.         xmlns:t="http://myfaces.apache.org/
tomahawk">

5.  <h:head>
6.    <title>Tomahawk</title>

7.      <style>
8.        .ativa {
9.          background-color: #6A5ACD;
```

```
10.         }
11.       .painel{
12.           width:100%;
13.           font-family: Arial;
14.           background-color:#B0C4DE;
15.         }
16.       </style>

17.   </h:head>

18.   <h:body>
19.     <h:form>

20.       <t:panelTabbedPane selectedIndex="0"
21.                          activeTabStyleClass="ativa"
22.                                  cellpadding="2%"
23.                                  styleClass="painel">

24.         <t:panelTab label="Home"> <!-- Painel 1 -->

25.           <h1>Painel 1</h1>
26.             <t:outputLabel value="Selecione as
27.   opções:" />
28.           <br />
29.           <h2>Checkbox</h2>
30.           <t:selectManyCheckbox>
31.             <t:outputLabel value="Opção 1:" />
32.             <t:selectBooleanCheckbox value="1" />
33.             <t:outputLabel value="Opção 2:" />
34.             <t:selectBooleanCheckbox value="2" />
35.             <t:outputLabel value="Opção 3:" />
36.             <t:selectBooleanCheckbox value="3" />
            </t:selectManyCheckbox>
37.
38.           <h2>Radio</h2>
39.           <t:selectOneRadio id="opcoesRadio">
40.             <f:selectItem itemValue="Opção 1" />
41.             <f:selectItem itemValue="Opção 2" />
42.             <f:selectItem itemValue="Opção 3" />
43.             <f:selectItem itemValue="Opção 4" />
            </t:selectOneRadio>
44.
          </t:panelTab>
45.
46.         <t:panelTab label="Painel 2">
47.             <h1>Painel 2</h1>
```

```
48.                 </t:panelTab>
49.
50.             <t:panelTab label="Painel 3">
                        <h1>Painel 3</h1>
51.             </t:panelTab>
52.
53.         </t:panelTabbedPane>
54.     </h:form>
    </h:body>
    </html>
```

Fonte: elaborado pelo autor.

Pesquise mais

Para conhecer os demais componentes do *framework* Tomahawk, você deve acessar a documentação oficial no endereço <https://bit.ly/2AM1uto>, acesso em 6 ago. 2018.

Você pode verificar quais atributos cada um dos elementos deles suporta.

Framework HighFaces

HighFaces é uma biblioteca simples e leve com um único arquivo *.jar*. Ela adiciona componentes JSF com os quais você pode facilmente criar gráficos. Diferentemente das outras bibliotecas, para uso comercial, a HighFaces é proprietária e precisa ser adquirida uma licença para ser utilizada (LIVE, 2015).

Reflita

Você conheceu *frameworks* para adicionar recursos em seus projetos, com licença gratuita. O fato de um framework ser gratuito pode significar que tenha confiança e qualidade inferiores aos pagos? É possível ter mais suporte a estes tipos de componentes gratuitos?

Para adicionar a biblioteca de componentes HighFaces em seus projetos, acesse o site <https://sourceforge.net/projects/highfaces/> (acesso em: 6 ago. 2018). Baixe a biblioteca e adicione o arquivo *.jar* na pasta *lib* de seu projeto para utilizar

sua biblioteca. Após realizado este procedimento, inicie um novo projeto e adicione a referência da biblioteca HighFaces, conforme o Quadro 3.22.

Quadro 3.22 | Linha de referência do HighFaces

```
1.  <!DOCTYPE html>
2.  <html xmlns="http://www.w3.org/1999/xhtml"
3.        xmlns:hf="http://highfaces.org"
4.        xmlns:f="http://java.sun.com/jsf/core"
5.        xmlns:h="http://java.sun.com/jsf/html">
```

Fonte: elaborado pelo autor.

Com essa seção finalizamos a unidade que foi dedicada à apresentação de diversos *frameworks* criados especialmente para JSF. Com esses recursos, suas páginas podem ficar com uma interface gráfica atrativa e ainda atender aos critérios da responsividade, tão necessários aos dias atuais. Dentre todos os *frameworks* apresentados, o RichFaces, o PrimeFaces e o BootsFaces se destacam pela versatilidade, então, aproveite os recursos!

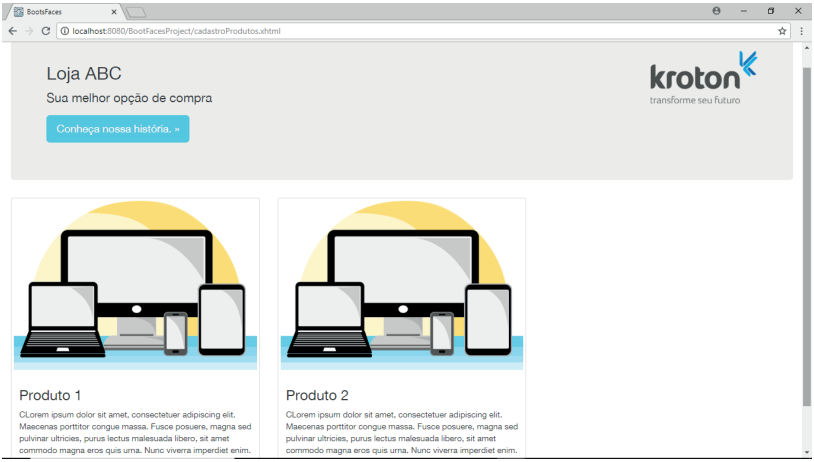
Sem medo de errar

Como etapa final de seu desafio, seu cliente mais importante possui um sistema antigo de cadastro de clientes, desenvolvido por sua empresa, e gostaria de atualizá-lo com melhorias em seu visual, para deixá-lo mais dinâmico. Desta forma, agora é hora de você apresentar a última atualização ao seu cliente.

Você pode optar pelo *framework* BootsFaces que, além de possuir diversos temas pré-definidos, tem uma série de componentes que podem enriquecer a apresentação. Um dos componentes que não pode faltar nessa interface gráfica é o `<b:thumbnail>`, que permite criar miniaturas das fotos dos produtos e, ao combinar com o componente `<f:facet>`, torna-se um recurso muito interessante para a exibição de produtos. Veja nas Figuras 3.14 e 3.15 o uso desses recursos e a responsividade do *framework*.

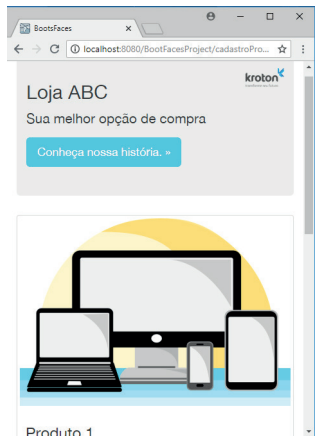
Na primeira imagem, a interface gráfica está sendo exibida em um computador, já na segunda, em um smartphone. Observe que na segunda imagem, cada produto é exibido de forma a ocupar toda a tela.

Figura 3.14 | Utilização de componentes BootsFaces: visualização a partir da tela do computador



Fonte: captura do software Eclipse, elaborada pelo autor.

Figura 3.15 | Utilização de componentes BootsFaces: visualização a partir da tela do smarthphone



Fonte: captura do software Eclipse, elaborada pelo autor.

Você pode conferir o código usado para gerar as Figuras 3.14 e 3.15 Acesse no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u3/s3/ResolucaoSP.pdf> ou por meio do QR Code.

Avançando na prática

Tela de login

Descrição da situação-problema

Um de seus clientes que atua como advogado possui uma aplicação web institucional. Esta aplicação apresenta sua empresa e os serviços online para que qualquer pessoa possa conhecê-lo ou contratá-lo. Também é utilizada apenas para acesso público, porém, seu cliente deseja colocar um painel administrativo para que ele possa alterar informações por meio de usuário e senha com acesso de administrador. Deve ser levado em conta que o acesso pode ser feito em um computador ou smartphone, ou seja, diferentes tamanhos de tela.

Seu trabalho é desenvolver a tela de acesso ao painel administrativo, com usuário e senha, possibilitando a responsividade na página.

Resolução da situação-problema

Para apresentar esta solução ao seu cliente, é importante ter em mente a utilização do framework BootsFaces, que possui uma natureza responsiva. Observe no Quadro 3.23 uma opção para a implementação.

Quadro 3.23 | *telaLogin.xhtml*

```
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:b="http://bootsfaces.net/ui">

<h:head>
  <title>Login</title>
</h:head>

<h:body>
  <h:form>
    <b:container>
```

```
      <h:panelGrid columns="2">
        <h:outputText value="Login:" />
        <b:inputText placeholder="e-mail ou CPF" />
        <h:outputText value="Senha:" />
        <b:inputSecret placeholder="Senha de 5
caracteres" />
        <b:commandButton value="login" look="primary" />
        <b:commandButton value="password forgotten"
look="danger"  />
      </h:panelGrid>
    </b:container>
  </h:form>
</h:body>
</html>
```

Fonte: elaborado pelo autor.

Faça valer a pena

1. Desenvolvido pela Apache Software Foundation, o *framework* _____ faz parte de um projeto que vai além da _____ JSF existente, e uma das principais implementações JSF do _____ é referente ao subprojeto, que fornece componentes adicionais, com suporte a _____ e também com vinculação de dados, dentre outras, para o desenvolvimento web.

Assinale a alternativa que apresenta as palavras que completam a sentença, respectivamente.

a) Tomahawk, especificação, MyFaces, Ajax.
b) HighFaces, página, Java, Web.
c) BootsFaces, implementação, Oracle, responsividade.
d) Tomahawk, implementação, Java, Ajax.
e) BootsFaces, página, MyFaces, Web.

2. Os recursos responsivos do BootsFaces permitem projetar apenas uma versão de um projeto de site, que se ajustará a smartphones, tablets pequenos e grandes e telas de área de trabalho, assim, as páginas são adaptadas automaticamente em todos os dispositivos, desde móveis a computadores.

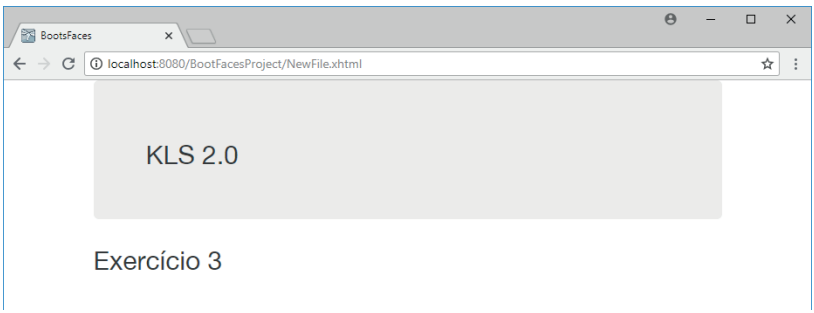
Com relação ao *framework* BootsFaces, analise as sentenças e escolha uma das opções.

I - O BootsFaces é um poderoso e leve *framework* JSF baseado em Bootstrap 3 e jQuery UI.
II - O *framework* BootsFaces permite desenvolver aplicativos pessoais *front-end* com rapidez e facilidade.
III - A biblioteca do BootsFaces oferece recursos desde a estrutura base da página até o layout final da página.
IV - O principal objetivo do BootsFaces é auxiliar os desenvolvedores a obterem sites na web bem estruturados e leves, com pouco trabalho.

a) Somente as alternativas I, II e III estão corretas.
b) Somente as alternativas I, II e IV estão corretas.
c) Todas as alternativas estão corretas.
d) Somente as alternativas I e II estão corretas.
e) Somente as alternativas I e IV estão corretas.

3. Considerando a figura a seguir, escolha a opção que contém os componentes que foram usados na construção dessa interface gráfica (Figura 3.16).

Figura 3.16 | Interface gráfica construída



Fonte: captura do software Eclipse, elaborada pelo autor.

a) `<b:container>, <b:jumbotron>, <h:head>.`
b) `<b:panel>, <b:container>, <b:selectOneMenu>.`
c) `<b:container>, <b:jumbotron>, <b:panel>.`
d) `<b:jumbotron>, <b:selectMultiMenu>, <h:head>.`
e) `<b:dataTable>, <h:form>, <b:jumbotron>.`

Referências

CODE, Google. **The Complete Integration of Google Maps with the JavaServer Faces**. 2018. Disponível em: <http://bit.ly/2Msubx3>. Acesso em: 6 ago. 2018.

COELHO, H. **JSF Eficaz**: as melhores práticas para o desenvolvedor web Java. São Paulo: Casa do Código, 2013.

APACHE MYFACES. **MyFaces Tomahawk.** 2012. Disponível em: <http://bit.ly/2ANxRYG>. Acesso em: 6 ago. 2018.

BOOTFACES. **Design your application quickly**. Disponível em: <https://www.bootsfaces.net/>. Acesso em: 19 jul. 2018.

ICESOFT Technologies Inc. **ICEfaces Overview - RIA for Java EE Web and Mobile Applications**. ICESOFT Technologies Inc. 2017. Disponível em: <https://bit.ly/2vnSFjF>. Acesso em: 1 ago. 2018.

LEATHEM, B.; FRYC, L.; ROGERS, S. Doc. **Developer Guide - Develop applications using RichFaces 4**. JBoss. [S.l.], 2011. Disponível em: <https://red.ht/2LFfwlw>. Acesso em: 1 ago. 2018.

LIVE, Bauer. **HighFaces**: Ultimate JSF2 - Chart Library based on HighCharts. [S.l.], 2015. Disponível em: <http://bit.ly/2AlDwzm>. Acesso em: 6 ago. 2018.

PIKHULYA, D. **An Introduction to OpenFaces**. JSFCentral. [S.l.], 25 out. 2010. Disponível em: <https://bit.ly/2v9F2oG>. Acesso em: 1 ago. 2018.

PRIMEFACES. **User Guide is the complete reference of PrimeFaces**. 2017. Disponível em <http://bit.ly/2M1LV6g>. Acesso em: 6 ago. 2018.

THECODER4. **BootsFaces**: the next-gen JSF Framework based on Bootstrap. 2013. Disponível em: <http://bit.ly/2KxcCKc>. Acesso em: 6 ago. 2018.

Desenvolvimento de aplicações com API

Convite ao estudo

Caro estudante, bem-vindo à última unidade no livro de Programação para Web II. Você teve a oportunidade de conhecer o desenvolvimento de aplicações web usando a especificação *Java Server Faces* para construir as interfaces do usuário, e como estruturar o projeto em camadas, seguindo as boas práticas de programação. Na primeira unidade, você viu como implementar a classe modelo, a interface gráfica usando arquivos xhtml e a *Managed Bean*, uma classe especial usada para fazer a mediação entre as camadas modelo e visão. Na segunda unidade, você conheceu o framework Hibernate, um dos mais utilizados para fazer a persistência de dados usando o mapeamento objeto-relacional. A terceira unidade foi dedicada ao estudo de frameworks que facilitam o desenvolvimento de interfaces gráficas mais modernas, com destaque para o RichFaces, o PrimeFaces e o BootsFaces. Agora, chegou o momento de se aprofundar ainda mais no universo de desenvolvimento web e se capacitar para utilizar APIs para desenvolver uma aplicação web.

Você, como proprietário de uma empresa de desenvolvimento de soluções para web, precisa estar preparado para os mais diversos desafios que surgem. Sua equipe de marketing fechou dois novos contratos. O primeiro cliente é proprietário de uma rede de lojas on-line e, para atingir o maior número de usuários possíveis, disponibiliza acesso à loja por meio de um website e de um aplicativo para dispositivo móvel. Contudo, esse cliente está enfrentando dificuldades para manter controle da comunicação entre website e dispositivo móvel. Por esse motivo, ele o contratou para desenvolver

uma solução para padronizar a comunicação entre as diferentes plataformas de acesso à loja. O cliente definido no segundo contrato é proprietário de uma agência de turismo e o contratou para desenvolver uma solução web capaz de calcular datas com diferentes fusos horários. Quais conceitos, arquiteturas e bibliotecas serão necessários para solucionar os desafios? Todos podem ser solucionados usando o *Java Server Faces* ou será preciso incluir outros recursos? E, quanto ao servidor web, você está acostumado a usar o WildFly, quais benefícios ainda poderão ser explorados?

Nesta unidade você entrará no universo das APIs. Na primeira seção, você aprenderá o que é uma RESTFul API e como implementar uma solução com esse recurso. Na segunda seção, você irá explorar o uso de API para tratamento de datas e horas, um importante recurso em qualquer tipo de solução computacional. Por fim, na terceira seção, você aprenderá a enviar mensagens e a implementar alguns recursos de segurança em suas aplicações web. Parabéns pelo seu empenho e vamos juntos em mais essa aventura.

Seção 4.1

API RESTFul

Diálogo aberto

Caro aluno, nós vivemos em um mundo mais conectado que nunca. Temos acesso a compras on-line, pedidos a restaurantes, compras de passagens áreas na ponta de nossos dedos. Podemos utilizar computadores desktops e dispositivos móveis para acessarmos informações que estão aqui ou do outro lado do mundo. Mas como toda essa conectividade acontece? Como os dados que estão aqui se comunicarão com os dados que estão do outro lado do mundo? Como diferentes dispositivos e aplicativos conseguem trocar informações? A resposta para todas estas questões é o assunto principal desta seção. Nós estudaremos uma solução capaz de padronizar a comunicação entre aplicações desenvolvidas para diferentes plataformas.

Você é o proprietário de uma empresa de desenvolvimento web e sua equipe de marketing recentemente fechou um contrato com um cliente que possui uma rede de lojas on-line. Esse cliente disponibiliza acesso à sua loja por meio de um website e de um aplicativo móvel. Com o crescimento da rede de lojas on-line, o proprietário está enfrentando dificuldades para manter a comunicação padronizada entre o website e o aplicativo para dispositivo móvel.

Você foi contratado para desenvolver uma solução para padronizar a comunicação entre diversas plataformas. O seu trabalho é desenvolver um Web Service capaz de padronizar o acesso aos Produtos e Clientes cadastrados, bem como de realizar Vendas, independentemente de se o usuário utiliza o website ou o aplicativo móvel.

Como você implementará essa solução para o cliente? Quais bibliotecas e APIs serão necessárias para desenvolver tal solução? É possível resolver utilizando o servidor WildFly?

Nesta seção, você conhecerá o que é uma RESTFul API, quais bibliotecas podem ser utilizadas para implementarmos essa arquitetura e verá como criar um Web Service. Este é só o começo de um novo universo de possibilidades.

Bom estudo!

Não pode faltar

O que é uma API?

As aplicações modernas geram um grande volume de dados. De acordo com a Google (2018), a plataforma do Google Maps recebe em média 25 milhões de atualizações diárias.

Imagine que você esteja desenvolvendo uma aplicação Web e deseja disponibilizar para os usuários informações atualizadas sobre milhões de locais. Você poderá se beneficiar de toda a estrutura já implementada pela Google e utilizar todas as atualizações diárias recebidas em sua aplicação Web. Uma **Interface de Programação de Aplicativos** (*Application Programming Interface* – **API)** estabelece um conjunto de métodos para disponibilizar dados de uma aplicação para outras aplicações.

Portanto, não devemos nos preocupar com como os dados são implementados no Google Maps. Por meio da API do Google Maps, devemos apenas solicitar as atualizações diárias e aguardar pela resposta para a disponibilizarmos em nossa aplicação Web.

Pesquise mais

A API de cada empresa irá especificar quais dados são disponibilizados, como as solicitações devem ser implementadas e em quais formatos serão retornadas. Conheça a documentação da API de grandes empresas.

FACEBOOK. **Documentação**. Disponível em: <https://developers.facebook.com/docs>. Acesso em: 20 set. 2018.

GOOGLE. **Plataforma do Google Maps Documentação**. Disponível em: <https://developers.google.com/maps/documentation/>. Acesso em: 20 set. 2018.

INSTAGRAN. **Hello Developers**. Disponível em: <https://www.instagram.com/developer/>. Acesso em: 20 set. 2018.

LINKEDIN. **Getting Started Guides**. Disponível em: <https://developer.linkedin.com/docs>. Acesso em: 20 set. 2018.

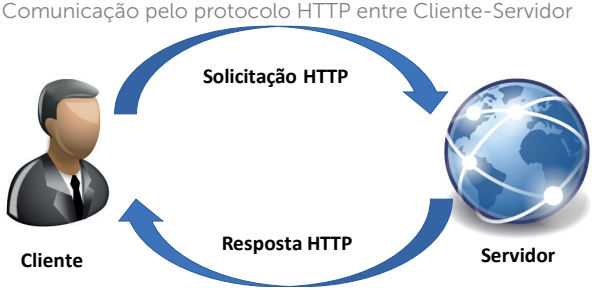
O que são Web Services?

Os Web Services são APIs que utilizam a internet para realizar trocas de dados. Um Web Service "pode ser usado para expor funcionalidades e componentes existentes para outras empresas ou para novos aplicativos." (BOND, et al., 2003, p. 793). De acordo com a Oracle (2018), os Web Services fornecem uma forma padrão de comunicação para aplicações que estão em execução em diferentes plataformas e frameworks. Ainda segundo a empresa, os Web Services utilizam o protocolo HTTP para se comunicarem.

Métodos HTTP

O **Protocolo de Transferência de Hipertexto** (*Hypertext Transfer Protocol* – **HTTP**) é o conceito inicial para a comunicação de dados na *World Wide Web* – WWW. Por meio desse protocolo, é possível que máquinas com diferentes configurações possam se comunicar. A W3Schools (2018) garante que o HTTP é projetado para trabalhar com base no protocolo de pedido-resposta entre Cliente e Servidor. A Figura 4.1 ilustra a comunicação entre um Cliente e um Servidor.

Figura 4.1 | Comunicação pelo protocolo HTTP entre Cliente-Servidor

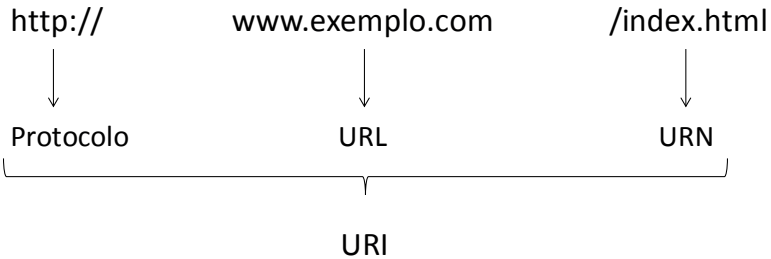


Fonte: adaptada de <https://upload.wikimedia.org/wikipedia/commons/1/19/Boss-icon.png>; <http://res.publicdomainfiles.com/pdf_view/82/13938365416588.png>. Acesso em: 18 jul. 2018.

Sempre que um Cliente envia uma solicitação para o Servidor, o Servidor retorna uma resposta. O Cliente deve enviar uma solicitação por meio de uma de uma URI. Conheça algumas definições que são utilizadas diariamente:

- O **Localizador de Recurso Universal** (*Universal Resource Locator* – **URL**): refere-se ao local que o Cliente deseja acessar.
- O **Nome de Recurso Universal** (*Universal Resource Name* – **URN**): refere-se ao nome do recurso que o Cliente deseja acessar.
- O **Identificador de Recurso Universal** (*Uniform Resource Identifier* – **URI**): refere-se ao identificador completo do recurso que o Cliente deseja acessar. Tudo o que está disponível na internet possui um caminho único para ser acessado. Esse caminho é o identificador do recurso. Em outras palavras, são os *links* disponíveis na *Web*. A Figura 4.2 apresenta a construção de uma URI.

Figura 4.2 | Exemplo de Identificador de Recurso Universal (URI)



Fonte: elaborada pelo autor.

Quando se trata de solicitações por meio do protocolo HTTP, devemos atentar a como essas solicitações são implementadas. O HTTP nos disponibiliza métodos, que são alguns verbos em inglês, para que o Servidor entenda o que o Cliente está solicitando. O Quadro 4.1 apresenta a definição de alguns métodos HTTP.

Quadro 4.1 | Métodos HTTP

Método	Descrição
GET	O método GET é utilizado para recuperar qualquer informação solicitada por meio de uma URI. Ao utilizar um navegador Web e acessar algum *website*, o usuário está fazendo uma solicitação ao Servidor por meio do método GET.

POST	Esse método é utilizado para solicitar ao Servidor que adicione o recurso enviado ao corpo da requisição e crie um ID para o mesmo.
PUT	O método PUT é utilizado para solicitar ao Servidor guardar o recurso enviado no corpo da requisição. Caso o recurso já exista, ele será atualizado; caso não exista, poderá ser criado.
DELETE	O método DELETE solicita ao Servidor deletar o recurso identificado pela solicitação URI.

Fonte: adaptado de <https://tools.ietf.org/html/rfc2616>. Acesso em: 20 set. 2018.

O que são REST e RESTFul?

Como desenvolvedores, podemos acessar diversos *Web Services* disponibilizados pelas empresas para utilizarmos em nossas aplicações. Contudo, também podemos desenvolver os nossos próprios *Web Services* para disponibilizarmos dados que estão sob nosso controle para outras empresas. Existem diversas formas de implementarmos um *Web Service*.

1. ***Representational State Transfer*** **(REST)** é um modelo de arquitetura que foi apresentado por Roy Fielding em sua tese de doutorado, no ano de 2000. O modelo REST é formado por um conjunto de regras construídas sobre o protocolo HTTP e pode ser aplicado em um *Web Service*. Um *Web Service* que atenda às restrições REST é considerado RESTFul. Abaixo estão listadas as seis restrições apresentadas na arquitetura REST que devem ser implementas em um *Web Service*.

   1.1 **Interface Uniforme** aplica o princípio que visa generalizar a resolução de um problema para que possa ser reutilizada em outros casos. Para se obter uma Interface Uniforme, é necessário adicionar quatro restrições:

   1.2 **Identificação de Recursos** é a restrição que identifica qualquer informação como um recurso, por exemplo, uma imagem ou documento.

   1.3 **Manipulação de Recursos através de Representações** indica que o Cliente não obterá um recurso do Servidor, mas sim uma Representação. A partir desta

restrição, podemos compreender o significado de REST: Transferência de Estado Representacional.

1.4 **Mensagem autodescritiva** indica que uma mensagem deve ser completa, contendo o formato de representação do recurso, a possibilidade de fazer cache, entre outras restrições exigidas por esta arquitetura.

1.5 HATEOAS - ***Hypermedia as the Engine of Application State*** é a propriedade de inserir em cada resposta todo o mapeamento possível referente ao recurso solicitado pelo Cliente. Tomemos como exemplo uma rede social, em que cada usuário poderá comentar e curtir publicações. Sempre que o Cliente solicitar ao Servidor informações sobre um determinado usuário, o Servidor deve incluir em sua resposta as URIs que darão acesso a todas publicações, bem como às curtidas.

2. ***Stateless*** define que cada requisição do Cliente deve ser completa, de modo que o Servidor consiga entender. O Servidor não deve armazenar nenhum contexto ou sessão da requisição.

3. **Cacheável** é a restrição que exige que o Servidor indique implicitamente ou explicitamente se a resposta enviada ao Cliente poderá ser armazenada para cache ou não.

4. **Cliente-Servidor** é o princípio que estabelece a separação de responsabilidades entre Cliente e Servidor. O Cliente é responsável pela apresentação dos dados para o usuário e o Servidor é responsável por armazená-los.

5. **Sistema em camadas** define uma arquitetura hierarquicamente, em camadas, de forma que cada componente só poderá interagir com aquele mais próximo na hierarquia.

6. **Código sob Demanda** não é uma restrição exigida pela arquitetura, mas poderá ser implementada.

De acordo com a Oracle (2018), no modelo REST, os dados e as funcionalidades implementadas em um *Web Service* são

considerados recursos e acessados por meio das URIs. Seguindo essa definição, para implementarmos uma URI baseada em recursos, vamos realizar a comparação entre duas URIs que acessarão um produto disponível para compra. Considere, a seguinte URI:

```
http://www.minhalojavirtual.com.br/exibirproduto?id=18
```

A URI exemplificada possui uma ação indicada por ***exibirproduto.*** Essa é uma URI válida para a construção de aplicações Web. Porém, ao implementar a arquitetura REST, não devemos utilizar ***ações*** em uma URI. A ação será indica pelo método HTTP e a URI ficará da seguinte forma:

```
http://www.minhalojavirtual.com.br/produtos/18
```

Por meio dessa URI, podemos utilizar o método GET para solicitarmos ao *Web Service* uma representação do produto 18.

Como implementar um RESTful *Web Service*?

A linguagem Java disponibiliza bibliotecas que permitem implementar diversas funcionalidades, por exemplo, para trabalharmos com coleções de objetos, datas ou manipulação de arquivos. Seguindo esse raciocínio, existem bibliotecas que nos permitem implementar RESTful *Web Services* em Java. Algumas soluções disponíveis para implementarmos RESTful *Web Services* são:

- **RESTEasy,** disponível em <https://resteasy.github.io/> (acesso em: 18 jul. 2018).
- **Restlet,** disponível em <https://restlet.com/> (acesso em 18 jul. 2018).
- **Jersey,** disponível em <https://jersey.github.io/> (acesso em 18 jul. 2018).

Assimile

JAX-RS é a API Java para RESTFul *Web Services*. Sua finalidade é simplificar e padronizar o processo de implementação. Portanto, todas as bibliotecas utilizadas para desenvolvermos RESTful *Web Services* implementam a API JAX-RS. Ela é padronizada pela *Java Specification Requests 311*, e está disponível em <https://jcp.org/en/jsr/detail?id=311> (acesso em: 18 jul. 2018).

Ao instalar o servidor **WildFly** e o **JBoss Tools** no Eclipse, também é instalada a biblioteca **RESTEasy**. Deste modo, o projeto no Eclipse já estará preparado para implementar um *Web Service*.

Caso deseje verificar se a biblioteca está instalada, siga os passos a seguir:

1) Clique com o botão direito do mouse sobre o projeto do Eclipse disponível na janela **Project Explorer**. Em seguida, selecione a opção **Properties**.

2) No menu lateral, selecione a opção **Java Build Path**.

3) Selecione a aba **Libraries**. Expanda a opção **WildFly 10.x Runtime**.

4) Para finalizar, localize a biblioteca instalada **resteasy.jaxrs-3.0.19.Final.jar**.

Para desenvolver um Web Service, devemos criar uma classe Java que representará o recurso a ser requisitado pelo Cliente. Essa classe deverá ser mapeada por anotações. O Quadro 4.2 apresenta algumas anotações utilizadas na construção de um Web Service, de acordo com a API JAX-RS.

Quadro 4.2 | Anotações referentes a API JAX-RS

Anotação	Descrição
@Path	A anotação @Path indica o caminho relativo de uma URI.
@Produces	A anotação @Produces é utilizada para especificar o tipo de retorno da representação de um recurso. Por exemplo, o Servidor poderá responder ao Cliente dados em "text/plain", "xml", "json", "html".

@Consumes	Indica quais tipos de representações de recursos o Servidor pode aceitar de um Cliente.
@PathParam	É a anotação responsável por extrair o parâmetro da URI para utilizá-lo na classe Recurso.
@GET	Essas anotações são similares aos métodos HTTP.
@POST	
@PUT	
@DELETE	

Fonte: adaptado de <https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp1v/index.html>. Acesso em: 20 set. 2018.

O código no Quadro 4.3 apresenta a construção de uma classe Java mapeada para receber solicitações do Cliente, construídas em um projeto *Dynamic Web Project* com nome ProjetoUnidadeIV. Como a arquitetura REST define que as informações de uma aplicação devem ser expostas como recursos, criamos uma classe *ProdutosResource*. Neste exemplo, implementamos um método que retornará o conteúdo em XML (*GET*) e outro método que retornará o conteúdo em HTML.

Quadro 4.3 | Código de uma classe Java com arquitetura REST

```
1  package unidadeiv;

2  import javax.ws.rs.GET;
3  import javax.ws.rs.Path;
4  import javax.ws.rs.PathParam;
5  import javax.ws.rs.Produces;
6  import javax.ws.rs.core.MediaType;

7  @Path("/produtos")
8  public class ProdutosResource {

9          @GET
10         @Produces(MediaType.TEXT_XML)
11         public String getProdutos() {
12                 StringBuilder builder = new
   StringBuilder();
```

```
13              builder.append("<?xml version = '1.0' ?>");
14              builder.append("<listagem>Exemplo de Listagem de Produtos em XML</listagem>");

15              return builder.toString();
16          }

17          @GET
18          @Path("/{descricao}")
19          @Produces(MediaType.TEXT_HTML)
20          public String getProduto(@PathParam("descricao") String descricao) {
21              String mensagem = String.format("<h1>Produto %s exibido no formato HTML</h1>", descricao);

22              return mensagem;
23          }
24  }
```

Fonte: elaborado pelo autor.

Para entendermos a implementação da classe *ProdutosResource*, acompanhe os comentários:

Linha 7: definimos o caminho que a URI deve possuir para acessar o recurso do *Web Service*. Observe que adicionamos uma barra antes da palavra "produtos", porém, não há obrigatoriedade de incluí-la.

Linha 11: declaramos um método responsável por retornar todos as representações de Produtos. Esse método será acessado pelo método HTTP GET, conforme implementado pela anotação da linha 9. Na linha 10 definimos o retorno para o Cliente no formato XML.

Linha 20: declaramos um método responsável por retornar apenas uma representação de Produto, de acordo com o parâmetro recebido na URI. Na linha 18, incluímos o parâmetro "descricao"

no caminho da URI. De volta à linha 20, recuperamos parâmetro "descricao" com a anotação @PathParam, e o utilizamos dentro do método. Esse método retornará ao Cliente uma representação em HTML, conforme definido na linha 19.

Linha 21: o método String.format() é responsável por formatar uma String. Fornecemos dois parâmetros nesse método. O primeiro refere-se ao texto que será formatado. Observe que, nesse parâmetro, incluímos o especificador de conversão %s. Tal especificador será substituído pelo segundo parâmetro informado no método, que é a *String* descrição informada na URI.

Reflita

A anotação @Produces é responsável por definir em qual formato um recurso deverá ser representado em resposta à solicitação do Cliente. A anotação @Produces poderá definir apenas um formato de representação para os recursos ou é possível incluir vários formatos?

Para finalizar a configuração do projeto no Eclipse, acesse o arquivo **web.xml** e adicione o código do Quadro 4.4. Na **linha 5** definimos os componentes da aplicação do *Web Service*. Na **linha 6** definimos o caminho inicial que será utilizado na URI para acessar o *Web Service*.

Quadro 4.4 | Configuração no arquivo web.xml

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns="http://java.sun.com/xml/ns/javaee"
                xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                                    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
                id="WebApp_ID" version="3.0">

3  <display-name>Primeiro Web Service</display-name>
```

```
4   <servlet-mapping>
5           <servlet-name>javax.ws.rs.core.Application</servlet-name>
6           <url-pattern>/rest/*</url-pattern>
7   </servlet-mapping>
8   </web-app>
```

Fonte: elaborado pelo autor.

Pesquise mais

Para exemplificar o uso de nosso Web Service, execute o projeto no Servidor. Abaixo, há duas URIs disponíveis para serem testadas. Ao testar a segunda URI, será possível alcançar o resultado conforme exibido na Figura 4.3. Observe que o nome declarado para o método não será utilizado para acessá-lo, ou seja, não utilizaremos getProdutos() ou getProduto(descricao). O acesso ao método é definido pela URI por meio da anotação @Path.

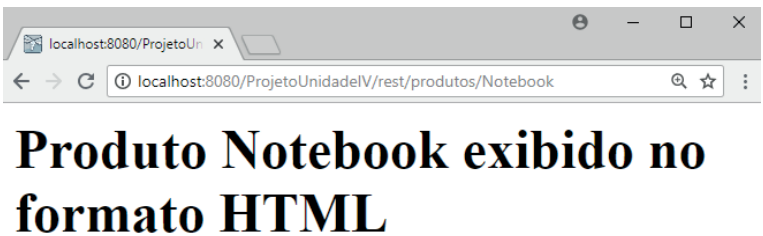
Teste essa URI:

http://localhost:8080/ProjetoUnidadeIV/rest/produtos/

Depois teste essa:

http://localhost:8080/ProjetoUnidadeIV/rest/produtos/Notebook

Figura 4.3 | Exemplo de solicitação GET ao Web Service



Fonte: captura de tela do Chrome, elaborada pelo autor.

Finalizamos esta seção com a introdução aos conceitos sobre a arquitetura REST e uma simples implementação de *Web Service*. Como desenvolvedor Web, seja sempre curioso e explore novas funcionalidades. Este conteúdo é o primeiro passo para um universo de novas possibilidades.

Sem medo de errar

A sua equipe de marketing recentemente fechou um contrato com um cliente que possui uma rede de lojas on-line. Esse cliente disponibiliza acesso à sua loja por meio de um website e de um aplicativo móvel. Com o crescimento da rede de lojas on-line, o proprietário está enfrentando complicações para manter a comunicação padronizada entre o website e o aplicativo para dispositivo móvel.

Você foi contratado para desenvolver uma solução capaz de padronizar a comunicação entre diversas plataformas. O seu trabalho é desenvolver um *Web Service* para padronizar o acesso aos Produtos e Clientes cadastrados, bem como a possibilitar Vendas, independentemente de se o usuário utiliza o website ou o aplicativo móvel.

Tal solução deverá ser implementada utilizando a biblioteca RESTEasy, disponível durante a instalação do servidor WildFly, e as ferramentas JBoss. Sobre a implementação, a equipe já definiu que, ao utilizar o *Web Service*, o Cliente que solicitar informações ao Servidor, poderá utilizar XML ou JSON. Para iniciar o desenvolvimento do Web Service, acesse o Eclipse e crie um *Dynamic Web Project*. Em seguida, acesse o arquivo web.xml e acrescente o código abaixo:

```
1  <servlet-mapping>
2        <servlet-name>javax.ws.rs.core.
   Application</servlet-name>
3        <url-pattern>/api/*</url-pattern>
4  </servlet-mapping>
```

Acompanhe as dicas abaixo para implementar o Web Service.

1) Crie uma classe Java chamada LojasResource e importe as bibliotecas necessárias.

2) Adicione a anotação @Pah("/recursos") ao declarar a classe.

3) Declare um método público, responsável por retornar um Produto solicitado por meio da URI, que atenda às seguintes características:

3.1) Inclua a anotação: @GET

3.2) Inclua a anotação: @Path("/produtos/{descricao}")

3.3) Inclua a anotação:

@Produces({ MediaType.TEXT_XML, MediaType.APPLICATION_JSON })

3.4) O método deve receber como parâmetro a anotação @PathParam("descricao") e uma *String*.

4) Declare um método público, responsável por retornar um Cliente solicitado por meioo da URI, que atenda às seguintes características:

4.1) Inclua a anotação: @GET

4.2) Inclua a anotação @Path("/clientes/{nome}")

3.3) Inclua a anotação:

@Produces({ MediaType.TEXT_XML, MediaType.APPLICATION_JSON })

3.4) O método deve receber como parâmetro a anotação @PathParam("nome") e uma String.

5) Declare um método público, responsável por adicionar uma venda por meio da solicitação da URI, que atenda às seguintes características:

4.1) Inclua a anotação: @POST

3.2) Inclua a anotação: @Path("/vendas/")

3.3) Inclua a anotação:

@Produces({ MediaType.TEXT_XML, MediaType.APPLICATION_JSON }) para retornar XML ou JSON para o Cliente que fez a solicitação.

3.4) Inclua a anotação:

@Consumes({ MediaType.TEXT_XML, MediaType.APPLICATION_JSON }) para recuperar os valores enviados com a solicitação URI.

Veja no Quadro 4.5 uma possível implementação para o *Web Service*.

Quadro 4.5 | *Web Service* REST

```
1  package api;
2
3  import javax.ws.rs.Consumes;
4  import javax.ws.rs.GET;
5  import javax.ws.rs.POST;
6  import javax.ws.rs.Path;
7  import javax.ws.rs.PathParam;
8  import javax.ws.rs.Produces;
9  import javax.ws.rs.core.MediaType;
10
11 @Path("recursos")
12 public class LojasResources {
13
14        @GET
15        @Path("produtos/{descricao}")
16        @Produces(MediaType.TEXT_XML)
17        public Produto getProduto(@PathParam("descricao") String descricao) {
18              // Aqui será implementada a funcionalidade de recuperar o Produto de acordo com o parâmetro recebido
19        }
20
21        @GET
22        @Path("clientes/{nome}")
23        @Produces(MediaType.TEXT_XML)
24        public Cliente getCliente(@PathParam("nome") String nome) {
25              // Aqui será implementada a funcionalidade de recuperar o Cliente de acordo com o parâmetro recebido
26        }
27
```

```
28          @POST
29          @Path("vendas")
30          @Produces({  MediaType.TEXT_XML,  MediaType.
    APPLICATION_JSON })
31          @Consumes({  MediaType.TEXT_XML,  MediaType.
    APPLICATION_JSON })
32          public Venda realizarVenda(Venda venda) {
33                  //    Aqui    será    implementada    a
    funcionalidade de adicionar a Venda de acordo com a
    solicitação recebida pela URI
34          }
35  }
```

Fonte: elaborado pelo autor.

Observe que estamos desenvolvendo apenas a camada que representa uma padronização de comunicação, pois essa é a finalidade de uma API. Não estamos preocupados com a implementação em si de cada funcionalidade, pois esse não é o objeto contratado pelo cliente.

Por meio da API proposta para o gerente da rede de lojas on-line, o website e o aplicativo para dispositivo móvel poderão utilizar o mesmo Web Service e conversarão de forma padronizada.

Avançando na prática

Web Service para fornecimento de estatísticas

Descrição da situação-problema

Uma grande empresa, especializada em desenvolver estatísticas sobre as preferências de usuários, deseja desenvolver uma solução capaz de disponibilizar seus dados para que outros programadores possam acessá-los em diferentes plataformas. Você foi contratado para atuar com a equipe de programadores responsável por desenvolver um *Web Service*. Esse *Web Service* deve ser capaz de receber solicitações de diversas plataformas por meio de uma URI que contenha um parâmetro. O parâmetro recebido representa uma característica da estatística, por exemplo, o *Web Service* deve

ser capaz de retornar a estatística de todos os usuários do estado de São Paulo ou, ainda, todos os usuários do sexo feminino. O *Web Service* deve fornecer o resultado nos formatos XML e JSON.

Resolução da situação-problema

Para desenvolver o *Web Service* com a equipe de programadores, acesse o Eclipse e crie um *Dynamic Web Project*. Em seguida, siga as dicas a seguir:

1) Acesse o arquivo web.xml e acrescente a configuração <servlet-mapping>.

2) Crie uma classe Java que representará a Classe Recurso.

3) Ao declará-la, adicione a anotação @Path("estatisticas").

4) Declare um método público que retorne um objeto Estatistica.

5) Esse método deve receber como parâmetro a anotação @PathParam e uma String que representará a característica da estatística.

6) Ao declarar esse método, adicione as anotações @GET, @Produces({ MediaType.TEXT_XML, MediaType.APPLICATION_JSON }), @Path("{características}").

No Quadro 4.6, temos uma possível implementação para a solução.

Quadro 4.6 | possível implementação para classe EstatisticaResource.java

```
1  package api;
2
3  import javax.ws.rs.GET;
4  import javax.ws.rs.Path;
5  import javax.ws.rs.PathParam;
6  import javax.ws.rs.Produces;
7  import javax.ws.rs.core.MediaType;
8
9  @Path("estatisticas")
10 public class EstatisticaResource {
```

```
11
12          @GET
13          @Produces({ MediaType.TEXT_XML, MediaType.
    APPLICATION_JSON })
14          @Path("{caracteristica}")
15          public      Estatistica      getEstatistica(@
    PathParam("caracteristica") String caracteristica)
    {
16          //   Aqui   deverá   ser   implementada   a
    funcionalidade de pesquisar a estatística pela
    característica recebida no parâmetro
17          }
18  }
```

Fonte: elaborado pelo autor.

Essa é a API responsável por fornecer estatísticas de acordo com um parâmetro informado na URI, pelo Cliente. Bom trabalho!

Faça valer a pena

1. Aplicações modernas têm como objetivo atingir o maior número de usuários possível, por meio de diversas plataformas. Por exemplo, o usuário tem o poder de utilizar a mesma aplicação em um desktop ou em um dispositivo móvel. Para que essa comunicação entre aplicações que estão em diferentes plataformas possa acontecer, é necessário que o programador desenvolva uma solução de comunicação padronizada.

Assinale a alternativa que apresenta a solução responsável por definir um conjunto de métodos que permitirão a comunicação entre aplicações.

a) Interface de Programação de Aplicativos
b) Localizador de Recurso Universal
c) Identificador de Recurso Universal
d) Nome de Recurso Universal
e) Transferência de Estado Representacional

2. O Protocolo de Transferência de Hipertexto, mais conhecido como HTTP, é um protocolo de comunicação utilizado na World Wide Web – WWW. Esse protocolo atua com o modelo de solicitação/resposta. Por meio dos métodos HTTP e a URI, o Cliente envia uma solicitação para o Servidor e, consequentemente, o Servidor retorna uma resposta para o Cliente.

Qual alternativa apresenta, respectivamente, os métodos HTTP responsáveis por solicitar e criar um recurso no Servidor.

a) PUT e POST
b) GET e CREATE
c) PUT e GET
d) GET e POST
e) POST e CREATE

3. JAX-RS é a API Java que fornece suporte para a implementação de *Web Services* de acordo com o padrão de arquitetura de Transferência de Estado Representacional (REST). De acordo com a Oracle (2018), programadores utilizam anotações JAX-RS para definir os recursos e as ações que serão executadas nestes recursos.

Assinale a alternativa que apresenta as anotações JAX-RS responsáveis por definir um caminho relativo da URI e extrair um parâmetro da URI.

a) @Path, @Produces
b) @Path, @GET
c) @Path, @PathParam
d) @PathParam, @GET
e) @PathParam, @Produces

Seção 4.2

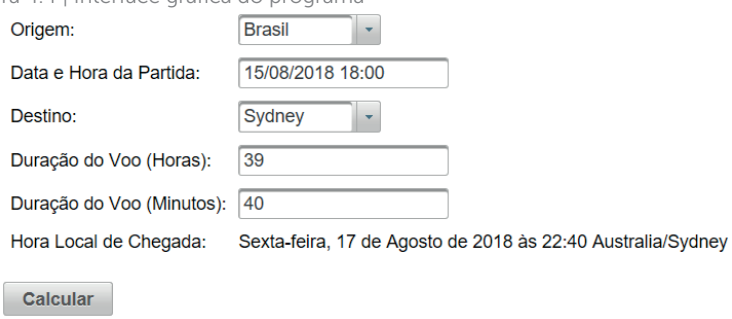
API data e hora

Diálogo aberto

Caro aluno, seja bem-vindo a mais uma seção! Atualmente, é quase impossível encontrar uma aplicação que não necessite manipular datas. Redes sociais armazenam data e horário para cada publicação, lojas virtuais registram data e horário durante a confirmação de uma venda e sistemas empresariais utilizam datas nos cadastros dos colaboradores. Nesta seção nós vamos explorar a nova API Java para manipularmos data e hora em uma aplicação web.

A equipe de marketing de sua empresa fechou um contrato com um proprietário de uma agência de turismo. Atualmente, essa agência recebe seus clientes para realizar orçamentos e consultar detalhes sobre as viagens. O atendente trabalha com uma tabela que contém a duração de voo entre origem e destino, portanto, após o cliente escolher sua viagem, o atendente precisa pesquisar na tabela a duração do voo e calcular manualmente o horário de chegada ao destino, considerando o fuso horário. A fim de agilizar o processo do cálculo do horário de chegada ao destino, você deve criar um sistema no qual o funcionário da agência informará a origem, a data e a hora de partida, o destino e o tempo de viagem, para que o sistema calcule automaticamente o horário da chegada considerando o fuso horário. A Figura 4.4 mostra como ficará a interface gráfica do programa.

Figura 4.4 | Interface gráfica do programa



Fonte: elaborada pelo autor.

Para desenvolver a solução web solicitada pelo cliente, estudaremos nesta seção a nova API Java para datas e horas, através da qual vamos manipular datas e horários, e trabalharemos com internacionalização de datas. Dedique um tempo para explorar as possibilidades de implementações através desta API. Bons estudos!

Não pode faltar

A manipulação de datas e horas é um grande desafio para programadores Java. Na primeira versão do Java, programadores manipulavam datas e horas através da classe `Date`. De acordo com Oracle (2018), esta classe representa um instante específico, com precisão de milissegundos. Ainda segundo a mesma documentação, a classe `Date` apresenta duas funcionalidades. A primeira é a possibilidade de interpretação de datas como ano, mês, dia, hora e minuto; e a segunda, a de conversão de *String* para *Date*. Contudo, esta API não apresenta uma solução segura em alguns casos, conforme exemplos:

- As classes `Date` e `SimpleDateFormatter` não são *thread-safe*.
- Não há suporte seguro para internacionalização.
- Design da API fraco e pouco intuitivo, por exemplo, na classe `Date` o ano inicia em 1900, o mês em 1 e o dia em 0.

Para contornar as limitações impostas pela classe `Date`, programadores utilizavam soluções desenvolvidas por outros programadores. A biblioteca que mais ganhou destaque foi a *Joda-Time*. Em consequência do sucesso, o seu autor Stephen Colebourne e a Oracle desenvolveram uma nova API para datas e horas, conforme a *Java Specification Request* - JSR 310 (ORACLE, 2018).

A partir do JDK 1.8 foi lançada a API para datas e horas, disponível no *package java.time*. Todas as classes implementadas nesta API são imutáveis e *thread-safe*.

Assimile

A linguagem Java utiliza o conceito de ***multithreading*** para que diferentes partes de um sistema possam ser executados simultaneamente (DEITEL; DEITEL, 2010). ***Thread-safe*** garante que, se um *thread* estiver no processo de atualização de um objeto, outro *thread* não terá acesso a este objeto até que a atualização termine. Esta técnica é utilizada para que o programador sempre tenha as informações atualizadas pelo último *thread* que acessou o objeto.

Vamos conhecer nesta seção as classes listadas abaixo. Para cada classe, haverá um exemplo de sua utilização, bem como o resultado esperado ao executar o código.

- *LocalDate*
- *LocalTime*
- *LocalDateTime*
- *Period*
- *Duration*
- *ZoneId*
- *ZonedDateTime*
- *DateTimeFormatter*
- *Instant*

LocalDate é a classe responsável por representar uma data com ano, mês e dia definidos. O seu uso é bastante simples e existem métodos que nos auxiliam ao instanciarmos uma data.

- O método `now()` nos retorna a data atual de acordo com o relógio do sistema e possui a seguinte sintaxe: `LocalDate hoje = LocalDate.now();`. Ao exibir o objeto *hoje*, será obtida a data no padrão ano-mês-dia. Em breve você aprenderá a modificar o padrão de exibição.

- O método `of()` cria uma instância de `LocalDate` de acordo com três parâmetros recebidos: ano, mês e dia. Sua sintaxe deve seguir a seguinte estrutura: `LocalDate dataDeNascimento = LocalDate.of(1990, 03, 06);` Ao invés de digitar o valor numérico do mês, este parâmetro pode ser especificado pelas constantes definidas no *enum Month*, conforme apresentado no Quadro 4.7. Para criar uma data com essa estrutura, basta utilizar a sintaxe: `LocalDate dataDeNascimento = LocalDate.of(1990, Month.MARCH, 6);`. O resultado também será numérico, porém a construção é feita de maneira mais elegante e padronizada.

Quadro 4.7 | Listagem de constantes do *Enum Month*

Mês	*Enum Month*
Janeiro	*Month.**JANUARY***
Fevereiro	*Month.**FEBRUARY***
Março	*Month.**MARCH***
Abril	*Month.**APRIL***
Maio	*Month.**MAY***
Junho	*Month.**JUNE***
Julho	*Month.**JULY***
Agosto	*Month.**AUGUST***
Setembro	*Month.**SEPTEMBER***
Outubro	*Month.**OCTOBER***
Novembro	*Month.**NOVEMBER***
Dezembro	*Month.**DECEMBER***

Fonte: adaptado de <https://docs.oracle.com/javase/8/docs/api/java/time/Month.html>. Acesso em: 20 set. 2018.

`LocalTime` é a classe responsável por representar um horário, sem data específica. Esta classe também nos fornece os métodos *`now()`* e *`of()`*.

Exemplificando

O método *`now()`* deve ser usado sempre que se deseja guardar a hora atual. Para isso, basta instanciar uma classe *LocalTime* e acessar o método guardando o valor em um objeto:

```
LocalTime agora = LocalTime.now();
//resulta no formato 20:53:43.397
```

O método `of()` permite guardar um horário especificado pelo usuário:

```
LocalTime horarioDoJogo = LocalTime.of(21, 00);
//resulta no formato 21:00a
```

`LocalDateTime` é a classe que une a data e o horário. O método `now()` também está disponível para essa classe. Para instanciar um objeto desse tipo, utilize a seguinte sintaxe:

```
LocalDateTime agora = LocalDateTime.now();
//resulta no formato 2018-07-23T20:55:54.254
```

Observe que o resultado do método `now()`, para essa classe, é construído pelo ano-mês-dia, pela letra "T" que representa *Time*, em seguida, horas-minutos-segundos-nanossegundos. Com essa classe, também podemos definir uma data e um horário através do método `of()`:

```
LocalDateTime horarioSessao = LocalDateTime.of(2018, Month.JULY, 23, 22, 0); //resulta em 2018-07-23T22:00
```

Reflita

A classe `LocalDateTime` disponibiliza o método `of()`, responsável por definir uma data e um horário composto por hora, minuto, segundo e até mesmo nanossegundos. Qual ordem dos parâmetros o programador deverá passar para o método `of()` para instanciar 27/07/2018 às 14 horas, 10 minutos e 30 segundos?

`Period` é a classe responsável por representar um período de data. Através do método `between()` você calculará o período entre duas datas. Cuidado, este método aceita apenas os objetos que contêm uma data. Veja um exemplo dessa classe no Quadro 4.8, no qual definimos um objeto com a data atual, outro com uma data

de prova e, por fim, usamos o método `between()` para calcular o tempo até a prova. Veja que o resultado é composto pela letra *P* de período no início, seguida por 7D, ou seja, nesse caso, 7 dias.

Um objeto do tipo `Period` também pode usufruir dos métodos `getDays()`, `getMonths()` e `getYears()`, os quais retornam o período de tempo em dias, meses ou anos. Veja no Quadro 4.8 que, ao imprimir o "`periodo2`" com o método `getDays()`, é exibido somente o valor 7.

Quadro 4.8 | Exemplo da classe `Period`

```
LocalDate hoje = LocalDate.now();
//resulta em 2018-07-24

LocalDate diaDaProva = LocalDate.of(2018, Month.JULY,
31);
//resulta em 2018-07-31

Period periodo = Period.between(hoje, diaDaProva);
//resulta em P7D

Period periodo2 = Period.between(hoje, diaDaProva);
System.out.println("Faltam "+ periodo2.getDays());
//resulta em 7
```

Fonte: elaborado pelo autor.

`Duration` é a classe responsável por representar um período de horário. Através do método `between()` você calculará o período entre dois horários. Este método aceita apenas objetos que tenham um horário. Veja no Quadro 4.9 o mesmo exemplo para prova, mas agora considerando as horas.

Quadro 4.9 | Exemplo da classe `Duration`

```
LocalDateTime hoje = LocalDateTime.now();
//resulta em 2018-07-24T14:48:09.559

LocalDateTime dataEHoraDaProva = LocalDateTime.of(2018,
Month.JULY, 31, 9, 0);
```

```
//resulta em 2018-07-31T09:00

Duration duracao = Duration.between(hoje, dataEHoraDaProva);
//resulta em PT162H11M50.441S
```

Fonte: elaborado pelo autor.

Você poderá converter o objeto `Duration` para um tempo específico, por exemplo, duração em dias, horas ou minutos.

```
duracao.toDays();

duracao.toHours());

duracao.toMinutes());
```

`ZoneId` é a classe responsável por identificar um fuso horário. O programador poderá utilizar o método *`of()`* para definir um fuso horário específico ou poderá chamar pelo método `ZoneId.systemDefault()` para recuperar o fuso horário do sistema.

`ZonedDateTime` representa uma data e um horário de acordo com um fuso horário. Analise os exemplos no Quadro 4.10. Na linha 1 foi criado um objeto para guardar o fuso horário do sistema. Na linha 2, foi criado um objeto para guardar a data e o horário atuais, de acordo com o fuso horário do sistema (veja que o `fusoHorarioSist` foi passado como parâmetro do método *`now()`*, na linha 2). Na linha 4, foi criado um novo objeto que agora guarda o fuso horário da América/São Paulo, usado como parâmetro no método *`now()`* na linha 5. Já na linha 7, o objeto guarda o fuso da Austrália/Sydney, que é passado como parâmetro na linha 8.

Quadro 4.10 | Exemplo da classe `ZoneId`

```
1. ZoneId fusoHorarioSist = ZoneId.systemDefault();
2. ZonedDateTime horarioSist = ZonedDateTime.now(fuso-
   HorarioSist);
3. System.out.println(horarioSist);
   // 2018-07-24T20:39:13.777-03:00[America/Sao_Paulo]
```

```
4.  ZoneId saoPaulo = ZoneId.of("America/Sao_Paulo");
5.  ZonedDateTime horarioBrasil = ZonedDateTime.
    now(saoPaulo);
6.  System.out.println(horarioBrasil);
    // 2018-07-24T20:39:13.777-03:00[America/Sao_Paulo]

7.  ZoneId sydney = ZoneId.of("Australia/Sydney");
8.  ZonedDateTime horarioSydney = ZonedDateTime.
    now(sydney);
9.  System.out.println(horarioSydney);
    // 2018-07-25T09:39:13.778+10:00[Australia/Sydney]
```

Fonte: elaborado pelo autor.

Pesquise mais

Conheça mais opções de fusos horários na documentação oficial da Oracle:

ORACLE, **Class ZoneId**. [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/ZoneId.html>. Acesso em: 21 set. 2018.

O programador também poderá utilizar o método `withZoneSameInstant()`, que recebe como parâmetro um `ZoneId`, para converter determinada data e horário para outro fuso horário. Observe no Quadro 4.11 que 06 de agosto de 2018, às 21h, no fuso horário de São Paulo representa 07 de agosto de 2018 às 02h no fuso horário de Paris. Na linha 1, declaramos um objeto `ZoneId` com o fuso horário de São Paulo. Na linha 2, declaramos um objeto `LocalDateTime` que armazena a data 06 de agosto de 2018 e o horário 21h. Na linha 3, aplicamos o fuso horário de São Paulo à data especificada na linha 2 e armazenamos em um objeto `ZonedDateTime`. Finalmente, na linha 4, utilizamos o método `withZoneSameInstant()` para descobrir qual é o horário em Paris e o armazenamos em outro objeto `ZonedDateTime`. Assim, conseguimos obter um determinado horário em diferentes países.

Quadro 4.11 | Exemplo de implementação do método `withZoneSameInstant()`

```
1    ZoneId fusoHorarioDeSaoPaulo = ZoneId.of("America/
     Sao_Paulo");

2    LocalDateTime localDateTime = LocalDateTime.
     now(2018, Month.AUGUST, 06, 21, 0); //resultado
     2018-08-06T21:00-03:00[America/Sao_Paulo]

3    ZonedDateTime saoPauloComFusoHorario = ZonedDate-
     Time.of(localDateTime, fusoHorarioDeSaoPaulo);

4    ZonedDateTime paris = saoPauloComFusoHorario.
     withZoneSameInstant(ZoneId.of("Europe/Paris"));
     //resultado 2018-08-07T02:00+02:00[Europe/Paris]
```

Fonte: elaborado pelo autor.

Assimile

A manipulação de datas e horas também está disponível na nova API. O programador poderá adicionar dias, meses, anos, horas, minutos ou até segundos para calcular novas datas e horários. O uso dos métodos é totalmente intuitivo, portanto, basta chamar por `plusDays()` para adicionar dias, `plusMonths()` para adicionar meses, `plusYears()` para adicionar anos, `plusHours()` para adicionar horas, `plusMinutes()` para adicionar minutos ou `plusSeconds()` para adicionar segundos, e passar como parâmetro o valor desejado, por exemplo: `LocalDateTime novoHorario = horarioAntigo.plusHours(1);`.

Uma classe muito utilizada na manipulação de datas é a `DateTimeFormatter`, pois é a responsável por conter métodos que criarão uma formatação da data e do horário para exibir para o usuário. Vamos conhecer uma opção disponível para formatarmos uma data e um horário. Através do método `ofPattern()`, podemos fornecer como parâmetro um padrão de letras predefinidas para criarmos uma formatação, por exemplo, "dd/MM/y" representando "dia/mês/ano". Após definirmos um padrão, devemos então chamar pelo método `format()` para exibirmos a data formatada. Veja no Quadro 4.12 como utilizar essa classe e método. Na linha 1, criamos um objeto para guardar a data atual. Na linha 2, por meio da classe `DateTimeFormatter` foi definido um objeto (`forma`) com o formato que se deseja exibir. Esse formato foi

usado como parâmetro do método `format()`, na impressão na linha 4. Os comentários exibem os resultados obtidos.

Quadro 4.12 | Exemplo de classe `DateTimeFormatter`

```
1.  LocalDate hoje = LocalDate.now();

2.  DateTimeFormatter forma = DateTimeFormatter.ofPat-
    tern("dd/MM/y");

3.  System.out.println("Sem formatação = " + hoje);
4.  System.out.println("Com formatação = " + hoje.
    format(forma));
    // Sem formatação = 2018-08-07
    // Com formatação = 07/08/2018
```

Fonte: elaborado pelo autor.

O Quadro 4.13 apresenta uma listagem de letras predefinidas que poderão ser utilizadas para formatarmos uma data e horário. Observe que há diferença entre letras maiúsculas e minúsculas, e a quantidade de letras utilizadas poderão alterar o resultado da formatação.

Quadro 4.13 | Padrões de formatação de data e hora

Símbolo	Significado	Apresentação	Exemplo
d	Dia do Mês	Número	1
E	Dia da Semana	Texto	Ter
EEEE	Dia da Semana	Texto	Terça-feira
M	Mês do Ano	Número	7
MMMM	Mês do Ano	Texto	Julho
y	Ano	Número	2018
yy	Ano	Número	18
a	AM-PM	Texto	PM
H	Hora	Número	0-23
h	Hora	Número	1-12
m	Minuto	Número	18
s	Segundo	Número	59
' '	Caractere de Escape	Texto/número	'Qualquer texto'

Fonte: adaptado de: <https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>. Acesso em: 20 set. 2018.

Exemplificando

Podemos criar combinações das letras predefinidas para exibirmos datas e horários personalizados para o usuário. O código do Quadro 4.14 exemplifica a exibição da data 01/08/2018 às 13 horas para *"01 de agosto de 2018, 01:00 PM"*.

Quadro 4.14 | Combinando diferentes formatos

```
// Define a formatação para a data e o horário
DateTimeFormatter formatador = DateTimeFormatter.
ofPattern("dd 'de' MMMM 'de' y',' hh:mm a");

// Define a data e horário para 01/08/2018 às 13 horas
LocalDateTime data = LocalDateTime.of(2018, Month.
AUGUST, 1, 13, 0);

// Aplica a formatação para a data e horário definidos
System.out.println(data.format(formatador));
```

Fonte: elaborado pelo autor.

E, para finalizar a seção, vamos analisar a seguinte situação: como migrar para a nova API de data e hora, caso você tenha um sistema já em produção e faz uso da antiga classe `Date`? Uma possível solução é através da classe `Instant`, que representa um momento específico. Através dela, poderemos converter objetos `Date` para a nova solução proposta a partir do JDK 1.8. No Quadro 4.15 são implementadas duas lógicas para convertermos um objeto `Date` para `LocalDateTime`. Na linha 1 declaramos um objeto do tipo `Date`, disponível a partir do JDK 1.0. Na linha 2, a partir do objeto data, recuperamos um objeto `Instant`, disponível a partir do JDK 1.8. A linha 3 apresenta a primeira opção disponível para converter um objeto `Date` para `LocalDateTime`. Por meio do método `LocalDateTime.ofInstant()` informamos como parâmetro o objeto `Instant` declarado na linha 2 e definimos um `ZoneId`. A segunda opção apresentada na linha 4 utiliza o objeto `Instant` e os métodos `atZone()` e `toLocalDateTime()`. Em ambas opções (linha 3 ou linha 4) é necessário definir um `ZoneId` para realizar a conversão.

Quadro 4.15 | Exemplo da classe `Instant`

```
1.  Date data = new Date();

2.  Instant instant = data.toInstant();

3.  LocalDateTime localDateTime = LocalDateTime.
    ofInstant(instant, ZoneId.systemDefault());

4.  LocalDateTime localDateTime2 = instant.
    atZone(ZoneId.systemDefault()).toLocalDateTime();
```

Fonte: elaborado pelo autor.

Chegamos ao final do conteúdo desta seção. Nós estudamos como manipular datas e horas através da API Java disponível a partir do JDK 1.8. Existem diversos métodos que merecem ser praticados, portanto, continue os seus estudos para explorar novas possibilidades.

Sem medo de errar

Atualmente, uma agência de turismo utiliza um catálogo de viagens com informações sobre a duração dos voos entre origem e destino para informar aos seus clientes o horário de chegada ao destino, considerando o fuso horário. O proprietário desta agência fechou contrato com a sua empresa para desenvolver uma solução web capaz de agilizar o processo do cálculo do horário de chegada no destino.

Você terá que desenvolver uma página que forneça as opções para os usuários selecionarem a origem da viagem, data e horário de partida, destino e a duração do voo. Após todos os requisitos serem preenchidos, o atendente poderá selecionar o botão Calcular para que a página calcule e exiba o horário de chegada de acordo com o fuso horário do destino.

Siga as dicas abaixo para desenvolver a proposta do contrato para o proprietário da agência de turismo:

1. Acesse o Eclipse e crie um Dynamic Web Project com a opção *JavaServer Faces v2.2 project*. O primeiro passo é configurar o ***ManagedBean*** no projeto.

2. Selecione a pasta *Java Resources*, em seguida a subpasta *src* e crie um pacote chamado *control*.

3. Neste pacote, crie uma classe Java chamada `VooMB`. Importe as referências necessárias e, ao declará-la, adicione as anotações `@ManagedBean(name="vooMB")` e `@RequestScoped` e declare um construtor sem parâmetros para a classe.

4. A classe `VooMB` deve possuir 5 atributos privados:

   4.1. Duas *Strings* que representarão `fusoHorarioOrigem` e `fusoHorarioDestino`.

   4.2. Um objeto do tipo *`Date`* que representará o `horarioPartidaSemFusoHorario`. Estamos utilizando um objeto `Date`, pois durante a construção da página que será disponibilizada para o usuário, usaremos o componente `calendar`, disponível na biblioteca Prime Faces, para que o usuário consiga selecionar a data e o horário de partida do voo. Este componente utiliza o objeto `Date` em sua implementação.

   4.3. Dois valores do tipo `int`, que representarão a duração do voo em horas e em minutos.

5. Crie os *Getters* e *Setters* para cada atributo declarado.

6. Declare um método chamado `getHoraLocalDoDestino()`, que não receba parâmetros e retorne um objeto `String`. Neste método deveremos implementar a lógica para calcular o horário de chegada no destino de acordo com o fuso horário.

6.1. A data e o horário que o usuário informar serão salvos em um objeto `Date`, conforme mencionado anteriormente, portanto, devemos convertê-los para o objeto `LocalDateTime`, inicialmente sem fuso horário aplicado.

6.2. O próximo passo é aplicar o fuso horário de origem e somar as horas e os minutos referentes à duração do voo.

6.3. Em seguida, devemos apenas converter o resultado encontrado no passo anterior para o fuso horário de destino.

6.4. Finalize a implementação declarando um objeto `DateTimeFormatter` para que o resultado seja apresentado formatado para o usuário.

O Quadro 4.16 apresenta uma possível implementação para o método `getHoraLocalDoDestino()`, no qual é obrigatório informar uma duração de voo, com horas e minutos diferentes de zero.

Quadro 4.16 | Sugestão de implementação para calcular duração entre dois horários

```
1  public String getHoraLocalDoDestino() {
     // Verificamos se o usuário informou a duração em horas e minutos
2    if ((duracaoHoras == 0) || (duracaoMinutos == 0)) {
3      return "A calcular";
4    }
     // Convertemos o objeto Date para LocalDateTime sem fuso horário
5            LocalDateTime semFusoHorario = horarioPartidaSemFusoHorario.toInstant()
     .atZone(ZoneId.systemDefault()).toLocalDateTime();

     // Aplicamos o fuso horário de Origem e somamos a duração do voo
6    ZonedDateTime comFusoHorario = ZonedDateTime.of(semFusoHorario, ZoneId.of(fusoHorarioOrigem))
     .plusHours(duracaoHoras)
     .plusMinutes(duracaoMinutos);

     // Aplicamos o fuso horário do Destino para descobrirmos a hora local
7            ZonedDateTime horarioDestino = comFusoHorario.withZoneSameInstant(ZoneId.of(fusoHorarioDestino));

     // Criamos uma formatação para exibir a data
```

```
8   DateTimeFormatter formatarData = DateTimeFormatter.
    ofPattern("EEEE, dd 'de' MMMM 'de' y 'às' HH:mm
    VV");

9     return horarioDestino.format(formatarData);
10  }
```

Fonte: elaborado pelo autor.

Agora devemos configurar a nossa página web que será disponibilizada para o usuário. Para construir o layout da página, utilizaremos a biblioteca Prime Faces, portanto, não se esqueça de incluí-la na pasta *WebContent/WEB-INF/lib*.

Selecione a pasta *WebContent* e crie um arquivo chamado *agencia.xhtml*, que deverá conter:

Dois componentes `<p:selectOneMenu>` para o usuário selecionar a origem e o destino do voo.

1.1. Para incluir as opções para o usuário selecionar, utilize o elemento `<c:selectItem>`.

1.2. Um componente `<p:calendar>` que representará o dia e o horário de partida do voo.

1.3. Dois componentes `<p:inputText>` que representarão a duração do voo em horas e minutos.

1.4. Um elemento `<p:commandButton>`.

O Quadro 4.17 apresenta uma proposta de implementação para o arquivo *agencia.xhtml*.

Quadro 4.17 | Sugestão de implementação para a página *agencia.xhtml*

```
1   <!DOCTYPE html>
2   <html xmlns="http://www.w3.org/1999/xhtml"
          xmlns:ui="http://xmlns.jcp.org/jsf/
    facelets"
        xmlns:h="http://xmlns.jcp.org/jsf/html"
         xmlns:c="http://java.sun.com/jsf/core"
         xmlns:p="http://primefaces.org/ui">
```

```
3   <h:head>

4   </h:head>
5   <h:body>

6     <h:form>

      <!-- Declara um Grid com duas colunas,
    desta forma os objetos estarão organizados na
    página -->
7     <h:panelGrid columns="2" style="margin-bot-
    tom:10px" cellpadding="5">

8     <p:outputLabel for="origem" value="Origem:"
    />
9     <p:selectOneMenu id="origem" value="#{vooMB.
    fusoHorarioOrigem}">
10              <c:selectItem itemLabel="Brasil"
    itemValue="America/Sao_Paulo" />
11          <c:selectItem itemLabel="Paris" item-
    Value="Europe/Paris" />
12            <c:selectItem itemLabel="New York"
    itemValue="America/New_York" />
13        <c:selectItem itemLabel="Sydney" item-
    Value="Australia/Sydney" />
14      </p:selectOneMenu>

15      <p:outputLabel for="partida" value="Data
    e Hora da Partida:" />
16      <p:calendar id="partida" value="#{vooMB.
    horarioPartidaSemFusoHorario}"  pattern="dd/
    MM/yyyy HH:mm" />

17                <p:outputLabel   for="destino"
    value="Destino:" />
18              <p:selectOneMenu   id="destino"
    value="#{vooMB.fusoHorarioDestino}">
```

```
19              <c:selectItem itemLabel="Brasil"
itemValue="America/Sao_Paulo" />
20          <c:selectItem itemLabel="Paris" item-
Value="Europe/Paris" />
21            <c:selectItem itemLabel="New York"
itemValue="America/New_York" />
22         <c:selectItem itemLabel="Sydney" item-
Value="Australia/Sydney" />
23       </p:selectOneMenu>

24               <p:outputLabel   for="textHoras"
value="Duração do Voo (Horas):" />
25     <p:inputText id="textHoras" value="#{vooMB.
duracaoHoras}" />

26             <p:outputLabel   for="textMinutos"
value="Duração do Voo (Minutos):" />
27                <p:inputText   id="textMinutos"
value="#{vooMB.duracaoMinutos}" />

28          <p:outputLabel  for="textHoraDestino"
value="Hora Local de Chegada:" />
29           <p:outputLabel  id="textHoraDestino"
value="#{vooMB.horaLocalDoDestino}" />

30     </h:panelGrid>

       <!-- Botão Calcular -->
31      <p:commandButton value="Calcular" ajax="-
false" />

32     </h:form>

33  </h:body>
34  </html>
```

Fonte: elaborado pelo autor.

Finalizamos a implementação do cálculo da duração do voo. Nesta implementação aplicamos recursos de conversão de objetos `Date` para `LocalDateTime`, em seguida de `LocalDateTime` para `ZonedDateTime`. Também utilizamos conversões de datas e horários entre diferentes fusos horários. Parabéns, você fez um ótimo trabalho!

Avançando na prática

Aplicação web para cálculo do período de hospedagem em hotel

Descrição da situação-problema

Uma rede de hotel o contratou para desenvolver uma aplicação web capaz de calcular a quantidade de diárias para os seus futuros hóspedes. A aplicação deve fornecer para os usuários as opções de data de entrada e saída da hospedagem, bem como um botão Reservar. Após o usuário clicar no botão, a quantidade de dias deverá ser calculada e exibida para ele.

Resolução da situação-problema

Para desenvolver o desafio proposto, abra o Eclipse, crie um *Dynamic Web Project* com a opção *JavaServer Faces v2.2 project* e certifique-se de seguir todos os passos abaixo:

1. O projeto deverá ter um arquivo *xhtml* que disponibilize os seguintes componentes disponíveis na biblioteca Prime Faces:

    1.2. Dois componentes `<p:calendar>` que representarão o período da hospedagem no hotel.

    1.3. Um componente `<p:commandButton>` para realizar o cálculo do período da hospedagem.

    1.4. Um componente `<p:outputLabel>` para exibir o período da hospedagem após o cálculo.

2. Inclua no projeto uma classe Java que representará o *ManagedBean* e atenda aos requisitos abaixo:

   2.1. Deve possuir a anotação `@ManagedBean` e um construtor que não receba parâmetros.

   2.2. Deve possuir dois atributos privados `Date` e seus respectivos *Getters* e *Setters*.

   2.3. Um método `getCalcularHospedagem()` que retorne um objeto `Period`.

      2.31. Este método deverá converter os objetos `Date` para `LocalDate`.

      2.32. Através do objeto `Period`, deverá ser calculado o período da hospedagem.

Finalize a configuração do *ManagedBean* no arquivo *faces-config.xml,* disponível em *WebContent/WEB-INF*.

Você pode conferir uma possível implementação da solução clicando aqui <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u4/s2/avancando-na-pratica.pdf> ou por meio do QR Code.

Finalizamos aqui uma possível implementação para o cálculo do período de hospedagem em um hotel. Você poderá ajustar o layout e o código para oferecer uma experiência mais personalizada para o usuário. Bom trabalho!

Faça valer a pena

1. Segundo Oracle (2018), a classe `DateTimeFormatter` é responsável por formatar objetos de data e hora. Ainda segundo o mesmo autor, esta classe permite que o programador construa um padrão de formatação de data através de letras. Por exemplo, é possível exibir para o usuário a data de 05/07/2018 no seguinte formato: "Quarta-feira, 05 de julho de 2018".

Assinale a alternativa que apresenta para o usuário o padrão de letras para exibir a data conforme exemplificada.

a) *DateTimeFormatter.ofPattern("E, dd 'de' MMMM 'de' yyyy");*
b) *DateTimeFormatter.ofPattern("EEEE, d 'de' MMMM 'de' yyyy");*
c) *DateTimeFormatter.ofPattern("EEEE, dd 'de' M 'de' y");*
d) *DateTimeFormatter.ofPattern("E, d 'de' M 'de' y");*
e) *DateTimeFormatter.ofPattern("EEEE, dd 'de' MMMM 'de' y");*

2. As classes *LocalDate* e *LocalTime* são responsáveis por representar, respectivamente, uma data e um horário específicos. O programador também poderá unir uma data e um horário específicos através da classe *LocalDateTime*, frequentemente representada por ano-mês-dia hora-minuto.

Assinale a alternativa que implementa um objeto *LocalDateTime* definido em 31/12/2018 às 21 horas.

a) *LocalDateTime.of(2018, Month.DECEMBER, 31, 21);*
b) *LocalDateTime.of(2018, Month.DECEMBER, 31, 21, 0);*
c) *LocalDateTime.of(2018, 31, Month.DECEMBER, 21, 0);*
d) *LocalDateTime.of(2018, 12, 31, 21);*
e) *LocalDateTime.of(2018, 31, Month.DECEMBER, 21);*

3. A nova API Java para manipulação de datas e horas, disponível a partir do JDK 1.8, inclui a classe *ZonedDateTime* que permite ao programador trabalhar com a representação de data e hora com fuso horário. Para declarar uma data e hora, o programador também deverá instanciar um objeto *ZoneId* que determinará o fuso horário.

```
ZoneId fusoHorarioBrasil = ZoneId.of("America/Sao_
Paulo");
ZonedDateTime horaAtualBrasil = ZonedDateTime.
now(fusoHorarioBrasil);
```

Qual alternativa apresenta a data e horário definidos para 07/09/2018 às 9 horas com fuso horário do Brasil?

a) *ZonedDateTime.of(2018, 9, 7, 9, 0, 0, 0, fusoHorarioBrasil);*
b) *ZonedDateTime.of(fusoHorarioBrasil, 2018, 9, 7, 9, 0, 0, 0);*
c) *ZonedDateTime.of(2018, 9, 7, 9, fusoHorarioBrasil);*
d) *ZonedDateTime.of(2018, 9, 7, 9, 0, fusoHorarioBrasil);*
e) *ZonedDateTime.of(fusoHorarioBrasil, 2018, 9, 7, 9, 0);*

Seção 4.3

APIs de uso geral

Diálogo aberto

Caro aluno, seja bem-vindo à última seção do livro!

Com qual frequência você recebe e-mails? As empresas estão utilizando o envio de e-mails como uma estratégia de *marketing* para divulgar promoções e atingir um porcentual de público específico. Recentemente, grandes instituições financeiras também estão aderindo à estratégia de envio de e-mails com anexos de faturas, evitando o atraso em suas entregas, bem como promovendo o conceito de sustentabilidade. Conheceremos, nesta seção, uma solução para que possamos implementar uma aplicação web com a funcionalidade de enviar e-mails.

Após finalizar a entrega para a agência de turismo da aplicação web capaz de calcular a data e o horário do destino da viagem, de acordo com a duração do voo e o fuso horário, foi solicitada uma nova versão da aplicação web. Você terá que ajustar o código para que a data e o horário calculados sejam enviados para o e-mail do cliente. Portanto, a página web deverá fornecer uma opção para que o usuário consiga digitar o e-mail do cliente que receberá a informação. Também será necessário implementar a funcionalidade de enviar e-mails na classe Java *Managed Bean*.

Para que possamos implementar o desafio proposto nesta seção, conheceremos a biblioteca *Commons Email*. Por meio dela, é possível enviar e-mails simples, com anexo, bem como e-mails personalizados com HTML. Dedique um tempo para implementar as possibilidades oferecidas por essa biblioteca.

Bons estudos!

Não pode faltar

A API *JavaMail* fornece classes responsáveis por modelar um sistema de e-mails (ORACLE, 2018). Todas as classes que compõem

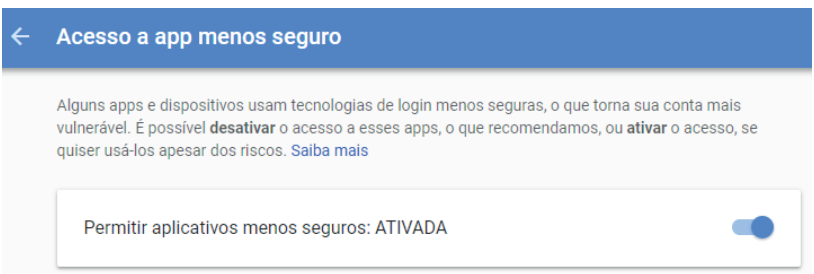
essa API estão disponíveis no pacote *javax.mail* e atendem às requisições impostas pela *Java Specification Requests* - JSR 375.

Atenção

Nesta seção, iremos utilizar o provedor Gmail para podermos enviar e-mails simples e com anexo. A empresa Google está em constante atualização para oferecer mais segurança para os seus usuários, portanto, alguns passos poderão ser modificados em relação aos apresentados nesta seção.

Antes de iniciarmos a implementação dos códigos, vamos criar uma conta no Gmail. Em seguida devemos configurá-la por meio do link <https://myaccount.google.com/lesssecureapps>, para permitir acesso de outros aplicativos, conforme exibido na Figura 4.5.

Figura 4.5 | Permissão do Google para acesso a app menos seguro



Fonte: captura de tela do Google.

Nesta seção, é apresentada a biblioteca ***Commons Email***, que foi desenvolvida de acordo com o padrão da API JavaMail e tem o objetivo de facilitar a implementação de um sistema de e-mails. Para que possamos utilizar tal biblioteca, iremos acessar o Eclipse e criar um *Dynamic Web Project* com a opção *JavaServer Faces* v2.2 *Project*.

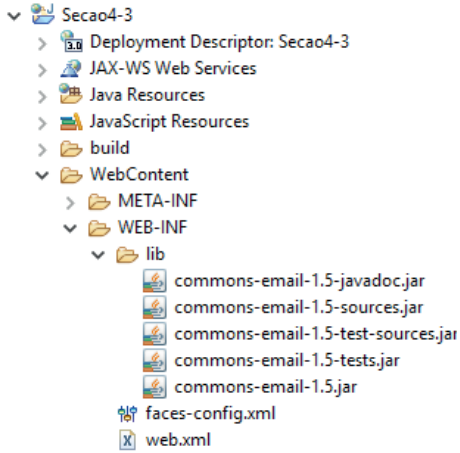
Atualmente, a biblioteca *Commons Email* está disponível na versão 1.5, portanto, os passos apresentados nesta seção poderão sofrer modificações em versões futuras. Faça o download da biblioteca *Commons Email*, por meio do site <http://commons.apache.org/proper/commons-email/download_email.cgi>. A Figura 4.6 apresenta a opção correta para download.

Figura 4.6 | Download da biblioteca *Commons Email* 1.5

Fonte: captura de tela do *website* <http://commons.apache.org>.

Após finalizar o download, descompacte o arquivo em uma pasta de sua preferência. O arquivo compactado deverá conter 5 arquivos com extensão .jar. Todos estes arquivos deverão ser incluídos no projeto do Eclipse, em WebContent/WEB-INF/lib, conforme exibido na Figura 4.7.

Figura 4.7 | Inclusão da biblioteca *Commons Email* 1.5 no Eclipse



Fonte: captura de tela do Eclipse, elaborada pelo autor.

A partir dessa ação, você já poderá utilizar a biblioteca *Commons Email* em uma aplicação web. Vamos conhecer alguns cenários possíveis de implementações.

Assimile

Desenvolver uma aplicação que ofereça a funcionalidade de enviar e-mails é uma tarefa que exige segurança. Conceitos como *Simple*

Mail Transfer Protocol (SMTP - Protocolo de Transferência de E-mail Simples) e *Secure Socket Layout* (SSL) merecem destaques. De acordo com Klensin (2008), o objetivo do SMTP é transferir e-mails de modo confiável e eficiente. Já o SSL é o padrão de segurança responsável por estabelecer uma conexão segura entre o Cliente e o Servidor.

Cenário 1: neste cenário, vamos implementar um método na classe *Managed Bean* que será responsável por enviar um e-mail simples. O Quadro 4.18 apresenta a implementação de um método para envio de um e-mail contendo uma mensagem simples. Veja, em seguida, a explicação dos principais comandos.

Quadro 4.18 | Implementação de envio de e-mail simples por meio da biblioteca *Commons Email*

```
1   package control;

2   import org.apache.commons.mail.Email;
3   import org.apache.commons.mail.EmailException;
4   import org.apache.commons.mail.SimpleEmail;

    // Criação dos itens necessários para a Managed
    Bean

5   public void enviarEmail() {

6   try {
7     Email email = new SimpleEmail();

8     email.setHostName("smtp.googlemail.com");
9     email.setSmtpPort(465);

10          email.setAuthentication("email@gmail.com",
    "senha");

11    email.setSSLOnConnect(true);

12    email.setFrom("remetente@gmail.com");
```

```
13      email.setSubject("Assunto do E-mail");
14      email.setMsg("Está é uma mensagem de e-mail");
15      email.addTo("destinatario@provedor.com");

16      email.send();

17    } catch (EmailException e) {
18      e.printStackTrace();
19    }
20    }
```

Fonte: adaptado de <http://commons.apache.org/proper/commons-email/userguide.html> (2018, [s.p.]).

Linhas 2 a 4: para utilizar as classes e métodos, é preciso importar a referência ao pacote de e-mail da Apache. As três importações podem ser substituídas por: `org.apache.commons.mail.*;`

Linha 5: declara um método público e sem retorno, chamado **`enviarEmail()`**. Esse método deverá ser declarado dentro de uma classe ManagedBean.

Linha 6 e 17: é necessário definir a implementação do método dentro de um bloco *try/catch* para caso ocorra alguma exceção durante a execução.

Linha 7: declara um objeto *SimpleEmail*, que será configurado nas próximas linhas para enviar o e-mail.

Linha 8: configura o servidor que enviará o e-mail. Esse servidor deverá ser consultado de acordo com o provedor de cada empresa de e-mail.

Linha 9: define a porta de envio. 465 é a porta padrão para a conexão SSL.

Linha 10: por meio do método **`setAuthentication()`**, definimos o usuário e a senha a serem utilizados quando o servidor solicitar autenticação.

Linha 11: ativa o protocolo de segurança.

Linha 12: define o remetente do e-mail.

Linha 13: define o Assunto do e-mail.

Linha 14: define a mensagem no corpo do e-mail.

Linha 15: define o destinatário.

Linha 16: envia o e-mail.

Exemplificando

Para exemplificarmos o uso do método `enviarEmail()`, disponível em uma classe *Managed Bean*, vamos considerar uma página chamada cliente.xhtml, que contenha o componente formulário apresentado no Quadro 4.19:

Quadro 4.19 | Formulário para envio de e-mail

```
1  <h:form>

     <!-- Declara um Grid com duas colunas, desta forma os componentes estarão organizados na página -->
2    <h:panelGrid columns="2" style="margin-bottom:10px" cellpadding="5">

3      <h:outputLabel for="para" value="Para:" />
4      <h:inputText id="para" value="#{emailMB.enviarPara}" />

5    <h:outputLabel for="assunto" value="Assunto:" />
6      <h:inputText id="assunto" value="#{emailMB.assunto}" />

7                <h:outputLabel for="mensagem" value="Mensagem:" />
8                <h:inputTextarea id="mensagem" value="#{emailMB.mensagem}" />

9    </h:panelGrid>
```

```
10          <h:commandButton value="Enviar"
action="#{emailMB.enviarEmail}"  ajax="false"
/>


11  </h:form>
```

Fonte: elaborado pelo autor.

Cada componente do formulário faz referência a um atributo da classe *Managed Bean*, portanto, cada atributo deverá ter os *Getters* e *Setters* declarados na *Managed Bean*.

Observeque,nalinha10,ocomponente`<h:commandButton>` possui o atributo *action* e é responsável por chamar o método declarado no *Managed Bean*. Esse componente também deve conter o atributo *ajax* definido para *false*.

Veja, no link <https://cm-kls-content.s3.amazonaws.com/ebook/embed/qr-code/2018-2/programacao-para-web-II/u4/s3/EmailMB_1.pdf> ou no QR Code, uma possível implementação completa para a classe Managed Bean, necessária para o funcionamento da interface gráfica criada no Quadro 4.19.

Cenário 2: nesse cenário, vamos implementar um método responsável por enviar um e-mail com anexo. O primeiro passo é construir um formulário que permita ao usuário selecionar o arquivo que deseja anexar ao e-mail. Portanto, conheceremos mais um componente, disponível na biblioteca *Prime Faces*. O Quadro 4.20 apresenta uma implementação de formulário com o componente `<p:fileUpload>`.

Quadro 4.20 | Implementação de formulário para envio de e-mail com anexo

```
1   <h:form enctype="multipart/form-data">


      <!-- Declara um Grid com duas colunas, desta
forma os componentes estarão organizados na página
-->
2       <h:panelGrid columns="2" style="margin-bot-
tom:10px" cellpadding="5">
```

```
3       <p:outputLabel for="para" value="Para:" />
4        <p:inputText id="para" value="#{emailMB.enviarPara}" />
5       <p:outputLabel for="assunto" value="Assunto:" />
6        <p:inputText id="assunto" value="#{emailMB.assunto}" />
7      <p:outputLabel for="mensagem" value="Mensagem:" />
8      <p:inputTextarea id="mensagem" value="#{emailMB.mensagem}" />
9       <p:outputLabel for="anexo" value="Anexo:" />
10       <p:fileUpload id="anexo" value="#{emailMB.arquivo}" mode="simple" skinSimple="true" label="Selecionar anexo" />
11     </h:panelGrid>
12      <p:commandButton value="Enviar" action="#{emailMB.enviarEmailAnexo}" ajax="false" />
13  </h:form>
```

Fonte: elaborado pelo autor.

Linha 1: o formulário deve obrigatoriamente conter o atributo `enctype="multipart/form-data"`. De acordo com a W3schools (2018), esse atributo é exigido quando estamos trabalhando com o método POST e com envios de arquivo em um formulário.

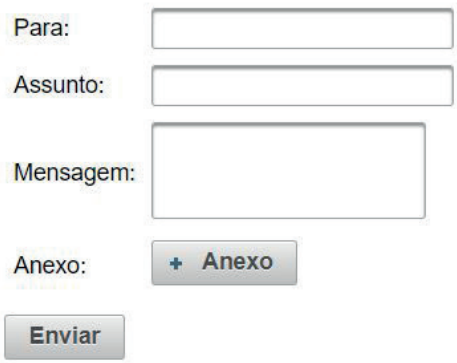
Linha 10: incluímos o componente `<p:fileUpload>` com diversos atributos. Vamos analisá-los, conforme listagem abaixo.

- O atributo **`value`** faz referência à classe *Managed Bean*, portanto, será necessário declarar um objeto do tipo **`UploadedFile`**, chamado **`arquivo`** e criar os respectivos *Getter* e *Setter*.

- O atributo **mode** indica o modo do selecionador de arquivos. As opções disponíveis são *simple* ou *advanced*.
- O atributo **skinSimple** define o *layout* de exibição para o usuário.
- O atributo **label** define o texto que será exibido para o usuário.

Linha 11: define o componente `<p:commandButton>` e, por meio do atributo `action`, chamamos o método declarado na classe *Managed Bean*. Esse componente também deve possuir o atributo `ajax` definido para *false*, pois estamos utilizando o modo simples do `<p:fileUpload>`. A Figura 4.8 ilustra o resultado do componente `<p:fileUpload>`, de acordo com as configurações apresentadas no Quadro 4.20.

Figura 4.8 | Formulário para envio de e-mail com anexo



Fonte: Captura de tela do Google Chrome, elaborada pelo autor.

Reflita

Existem muitas soluções disponíveis atualmente para a manipulação de arquivos em uma aplicação web. A biblioteca *Prime Faces* fornece o componente `<p:fileUpload>`, porém, quais outras soluções poderão ser utilizadas?

O Quadro 4.21 apresenta a implementação na Managed Bean para o envio de um e-mail com anexo por meio das classes `EmailAttachment` e `MultiPartEmail`. Confira, a seguir, a explicação dos principais comandos.

Quadro 4.21 | Implementação de envio de e-mail com anexo através da biblioteca *Commons Email*

```
1   import java.io.*;
2   import java.nio.file.*;
3   import org.apache.commons.mail.*;
4   import org.primefaces.model.UploadedFile;
5   import javax.faces.bean.ManagedBean;

    // Criação dos itens necessários para a Managed Bean

6   public void enviarEmailAnexo() {

7   try {

      // Criamos o arquivo do anexo
8           File arquivoTemporario = File.createTempFile("anexo", "email");
9     InputStream initialStream = arquivo.getInputstream();
10     Files.copy(initialStream, arquivoTemporario.toPath(), StandardCopyOption.REPLACE_EXISTING);

      // Declaramos o objeto que contém o anexo
11    EmailAttachment anexo = new EmailAttachment();
12      anexo.setPath(arquivoTemporario.getAbsolutePath());
13      anexo.setDisposition(EmailAttachment.ATTACHMENT);
14    anexo.setDescription("Descrição");
15    anexo.setName(arquivo.getFileName());

      // Declaramos o objeto e-mail
16    MultiPartEmail email = new MultiPartEmail();
17    email.setCharset(EmailConstants.UTF_8);
18    email.setHostName("smtp.googlemail.com");
19    email.setSmtpPort(465);
20     email.setAuthentication("seuemail@gmail.com", "sua senha");
```

```
21      email.setSSLOnConnect(true);

        // Configuramos o objeto e-mail
22      email.addTo(enviarPara);
23      email.setFrom("seuemail@gmail.com");
24      email.setSubject(assunto);
25      email.setMsg(mensagem);

        // Anexamos o objeto anexo ao e-mail
26      email.attach(anexo);

        // Enviamos o e-mail
27      email.send();

28    } catch (EmailException e) {
29      e.printStackTrace();
30    } catch (IOException e) {
31      e.printStackTrace();
32    }
33    }
```

Fonte: elaborado pelo autor.

Linhas 1 a 5: nessas linhas, são importados todos os pacotes que serão utilizados na classe *Managed Bean*. Nas linhas 1 e 2, utilizamos `import java.io.*` e `import java.nio.file.*` para podermos manipular arquivos com a linguagem Java. Por meio dessas importações, é possível acessar os arquivos e os seus atributos. Na linha 3, importamos o pacote de e-mail da biblioteca *Commons Email*. Na linha 4, importamos o componente `UploadedFile` da biblioteca *Prime Faces*. Por fim, na linha 5, importamos a anotação referente ao *Managed Bean*.

Linha 6: declara um método público e sem retorno, chamado **`enviarEmailAnexo()`.**

Linha 7, 28 e 30: é necessário definir a implementação do método dentro de um bloco *try/catch*, para o caso de ocorrer alguma exceção durante a execução. A linha 23 trata a exceção correspondente à manipulação de e-mail. A linha 25 trata a exceção

que poderá ocorrer nas linhas 3, 4 e 5, pois elas são responsáveis por manipular o arquivo que será anexado.

Linha 8: por meio do método `File.createTempFile()`, criamos um arquivo temporário. Esse método recebe dois parâmetros, que representam, respectivamente, o prefixo e o sufixo do arquivo temporário.

Linha 9: por meio do objeto `UploadedFile` chamado `arquivo`, recuperamos o conteúdo do arquivo representado por bytes e o armazenamos no objeto `InputStream`.

Linha 10: por meio do método `Files.copy()`, copiamos todo o conteúdo do arquivo selecionado pelo usuário para o arquivo temporário criado. Esse método recebe três parâmetros: o primeiro representa o conteúdo do arquivo escolhido pelo usuário, que está no objeto `InputStream`; o segundo representa o caminho que será copiado; e o terceiro é a opção indicando para substituir o arquivo, caso já exista.

Linha 11: declaramos um objeto `EmailAttachment` responsável por conter informações sobre o anexo.

Linha 12: o método **`setPath()`** recebe como parâmetro o caminho em que está armazenado o anexo. Nesse exemplo, informamos o caminho do arquivo temporário.

Linha 13: o método **`setDisposition()`** recebe como parâmetro uma *String* responsável por informar se o arquivo será anexado ou enviado no corpo do e-mail. `EmailAttachment.ATTACHMENT` é uma constante definida pela *Commons Email* responsável por informar que o arquivo deverá ser anexado ao e-mail.

Linha 14: **`setDescription()`** define uma descrição ao anexo.

Linha 15: o método **`setName()`** define o nome do anexo. No nosso exemplo, estamos utilizando o nome do arquivo original, por meio do método `arquivo.getFileName()`, disponível no objeto `UploadedFile`.

Linha 16: declaramos um objeto `MultiPartEmail`, capaz de enviar uma mensagem com um anexo.

Linha 17: definimos a codificação de caracteres para o e-mail.

Linha 18: configuramos o servidor que enviará o e-mail. Esse servidor deverá ser consultado de acordo com cada provedor de e-mail.

Linha 19: definimos a porta de envio. 465 é a porta padrão para a conexão SSL.

Linha 20: através do método **`setAuthentication()`**, definimos o usuário e a senha para serem utilizados quando o servidor solicitar autenticação.

Linha 21: O método **`setSSLOnConnect()`** ativa o protocolo de segurança.

Linha 22: por meio do método **`addTo()`**, adicionamos o destinatário.

Linha 23: por meio do método **`setFrom()`**, definimos o remetente.

Linha 24: por meio do método **`setSubject()`**, definimos o assunto do e-mail.

Linha 25: o método **`setMsg()`** define a mensagem que será enviada no corpo do e-mail.

Linha 26: o método **`attach()`** vincula o anexo ao e-mail.

Linha 27: envia o e-mail por meio do método **`send()`**.

Pesquise mais

Apache Commons é um projeto focado em componentes Java reutilizáveis. Esse projeto oferece diversas APIs e bibliotecas que merecem ser exploradas para ampliar as possibilidades de desenvolvimento. Conheça as soluções disponíveis por meio do link:

THE APACHE SOFTWARE FOUNDATION. **Apache Commons.** 19 set. 2018, v. 19. Disponível em: <http://commons.apache.org/>. Acesso em: 21 set. 2018.

Chegamos ao final da seção e finalizamos a implementação de dois métodos responsáveis por enviarem e-mails por meio da da

biblioteca *Commons Email*. Continue os seus estudos para alcançar novas possibilidades no desenvolvimento de aplicações web.

Sem medo de errar

Vamos dar continuidade ao contrato fechado com a agência de turismo, em que, na primeira etapa de desenvolvimento, nós implementamos o método capaz de calcular a data e o horário de chegada no destino da viagem, de acordo com o fuso horário e a duração do voo. Agora é hora de enviarmos essa informação para o cliente por meio de um e-mail. Portanto, essa etapa de desenvolvimento está dividida em quatro tarefas. Na primeira, você terá que criar uma conta no Gmail e habilitar a opção para acesso de outros app, por meio do link <https://myaccount.google.com/lesssecureapps>. Na segunda tarefa, você terá que incluir a biblioteca *Commons Email* no projeto do Eclipse. Na terceira tarefa, você deve adicionar na página web um campo de texto para que o usuário possa informar para qual e-mail será enviada a data e hora de chegada no destino. E por fim, na quarta tarefa o método responsável por enviar o e-mail deverá ser implementado na classe *Java Managed Bean*.

Para desenvolvermos a segunda tarefa, faça o download da biblioteca *Commons Email*, disponível em <http://commons.apache.org/proper/commons-email/download_email.cgi>, e finalize com os passos a seguir:

1) Extraia o arquivo para uma pasta de sua preferência.

2) Copie todos os arquivos .jar para a pasta *WebContent/WEB-INF/lib* do seu projeto Eclipse.

Para a terceira tarefa, certifique-se de adicionar os componentes abaixo no formulário disponível no arquivo *agencia.xhtml*.

```
<p:outputLabel for="emailCliente" value="E-mail do Cliente:" />
<p:inputText id="emailCliente" value="#{vooMB.emailCliente}" />
```

Para concluir, na quarta tarefa, acesse o arquivo VooMB.java e o implemente com os seguintes passos:

1) Declare uma *String* chamada *emailCliente* e crie os respectivos *Getter* e *Setter*.

2) Declare um método chamado `enviarEmail()`, que receba como parâmetro uma *String* com o e-mail do destinatário e outra *String* com a mensagem que será enviada no corpo do e-mail. Esse método deverá utilizar a classe `Email`, disponível na biblioteca *Commons Email*, para que seja possível enviar o e-mail com a data e o horário de chegada do destino da viagem. O Quadro 4.22 apresenta uma sugestão de implementação do método `enviarEmail()`. Ao implementar o método de acordo com o Quadro 4.22, não se esqueça de modificar os parâmetros "seuemail@gmail.com" e "suasenha", disponíveis na linha 6 e 8, para os dados de sua conta.

Quadro 4.22 | Sugestão de implementação de envio de e-mail através da biblioteca *Commons Email*

```
1   private void enviarEmail(String emailCliente,
    String mensagem) {

2   try {
3     Email email = new SimpleEmail();
4     email.setHostName("smtp.googlemail.com");
5     email.setSmtpPort(465);
6       email.setAuthentication("seu-email@gmail.com",
    "sua senha");
7     email.setSSLOnConnect(true);

8     email.setFrom("seu-email@gmail.com");
9      email.setSubject("Data e Horário de Chegada no
    Destino da Viagem");
10    email.setMsg("Seu voo chegará: " + mensagem);
11    email.addTo(emailCliente);

12    email.send();
```

```
13  } catch (EmailException e) {
14    e.printStackTrace();
15  }
16  }
```

Fonte: elaborado pelo autor.

Para finalizar a implementação da classe VooMB.java, no método `getHoraLocalDoDestino()`, será necessário chamar pelo método `enviarEmail()`. O Quadro 4.23 apresenta uma nova sugestão de implementação do método `getHoraLocalDoDestino()`.

Quadro 4.23 | Sugestão de implementação do método `getHoraLocalDoDestino()`

```
1   public String getHoraLocalDoDestino() {

    // Verificamos se o usuário informou a duração em
    horas e minutos
2   if ((duracaoHoras == 0) || (duracaoMinutos == 0))
    {
3     return "A calcular";
4   }

      // Convertemos o objeto Date para LocalDateTime
    sem fuso horário
5             LocalDateTime      semFusoHorario      =
    horarioPartidaSemFusoHorario.toInstant()
    .atZone(ZoneId.systemDefault()).
    toLocalDateTime();

      // Aplicamos o fuso horário de Origem e somamos
    a duração do voo
6     ZonedDateTime comFusoHorario = ZonedDateTime.
    of(semFusoHorario, ZoneId.of(fusoHorarioOrigem))
                              .plusHours(duracaoHoras).
    plusMinutes(duracaoMinutos);

      // Aplicamos o fuso horário do Destino para
    descobrirmos a hora local
7             ZonedDateTime      horarioDestino      =
    comFusoHorario.withZoneSameInstant(ZoneId.
    of(fusoHorarioDestino));
```

```
    // Criamos uma formatação para exibir a data e hora
8   DateTimeFormatter formatarData = DateTimeFormatter.ofPattern("EEEE, dd 'de' MMMM 'de' y 'às' HH:mm VV");

    // Verificamos se o usuário informou algum e-mail
9   if ((emailCliente != null) && (!emailCliente.isEmpty())) {
        // Chamamos pelo método responsável por enviar o e-mail para o usuário
10      enviarEmail(emailCliente, horarioDestino.format(formatarData));
11  }

12  return horarioDestino.format(formatarData);
13  }
```

Fonte: elaborado pelo autor.

Finalizamos a última etapa de desenvolvimento do projeto para a agência de turismo. Agora a aplicação web é capaz de calcular o horário local do destino da viagem, bem como enviar essa informação para o e-mail do cliente. Você poderá realizar ajustes e melhorias para oferecer mais funcionalidades para o usuário, por exemplo, acrescentar mais detalhes na mensagem que será enviada no e-mail. Seja criativo e bom trabalho!

Avançando na prática

Envio de e-mails para clientes

Descrição da situação-problema

Uma grande empresa de comércio on-line está com uma nova campanha promocional, com o objetivo de enviar cupons de descontos para os clientes selecionados. Você foi contratado para atuar com a equipe de programadores responsável por desenvolver uma aplicação web para essa. Foram-lhe delegadas as tarefas de

construir a página web com os campos do e-mail do cliente e a mensagem que será enviada, bem como, a classe Java *Managed Bean* com o método responsável por enviar o e-mail. Ao finalizar a implementação das tarefas, apresente-as para o restante da equipe de programadores.

Resolução da situação-problema

Para finalizar as tarefas que lhe foram delegadas pela equipe de programadores, crie um *Dynamic Web Project* com a opção JavaServer Faces v2.2 Project no Eclipse e certifique-se de atender aos seguintes passos:

1) Deverá ser incluído no projeto no Eclipse, a biblioteca *Commons Email* para que seja possível enviar e-mails, além da biblioteca *Prime Faces*.

2) Na página web .xhtml deverá incluir um formulário com três componentes. O primeiro componente será o `<p:inputText>`, que representará o e-mail do usuário. O segundo componente será o `<p:inputTextarea>,` que representará a mensagem com o cupom promocional que será enviado para o cliente. Por fim, o terceiro componente será o `<p:commandButton>`, responsável por chamar o método que enviará o e-mail.

3) Na classe Java *Managed Bean*, deverá declarar os atributos *String email* e mensagem com os respectivos *Getter* e *Setter*. Também será necessário declarar o método `enviarEmail()`, que estará associado ao botão do formulário.

4) Utilize a classe `Email`, disponível na biblioteca *Commons Email*, para enviar um e-mail com uma mensagem de texto.

Ao seguir as quatros dicas apresentadas acima, será possível enviar e-mails com os cupons promocionais apenas para os clientes selecionados. Recomendamos que faça sugestões para a equipe de programadores para oferecer uma experiência mais personalizada para a empresa de comércio on-line.

Faça valer a pena

1. A biblioteca *Commons Email* fornece as classes `EmailAttachment` e `MultiPartEmail` para que o programador consiga enviar um e-mail com anexo. Ao declarar ambos os objetos, o programador deve configurá-los por meio de métodos para que o e-mail possa ser enviado com sucesso.

```
EmailAttachment anexo = new EmailAttachment();
MultiPartEmail email = new MultiPartEmail();
```

Qual alternativa apresenta o método responsável por vincular o anexo ao e-mail?

a) `email.addTo(anexo);`
b) `email.setMsg(anexo);`
c) `email.setSubject(anexo);`
d) `email.attach(anexo);`
e) `email.send(anexo);`

2. De acordo com a documentação oficial da biblioteca *Commons Email* (2018), é possível enviar e-mails simples, com anexo, imagens e até mesmo e-mails formatados com HTML. Caso ocorra algum erro durante a configuração ou o envio de um e-mail, será lançada uma exceção para que o programador possa tratá-lo adequadamente.

Qual exceção é lançada caso ocorra algum erro durante o envio do e-mail?

a) `SendEmailException`
b) `EmailException`
c) `SimpleEmailException`
d) `ConfigEmailException`
e) `AttachmentException`

3. De acordo com Oracle (2018), a API JavaMail fornece meios para que o programador possa desenvolver aplicações de envio e leitura de e-mails. Com o objetivo de facilitar o desenvolvimento de um sistema de e-mails, a biblioteca Commons Email fornece classes capazes de enviar um e-mail simples, com anexo, imagem e e-mails formatados em HTML.

Quais são as classes disponibilizadas pela biblioteca *Commons Email* para enviar um e-mail simples e e-mail com anexo?

a) `Email, MultiPartEmail e EmailAttachment.`
b) `Email, HtmlEmail, EmailAttachment.`
c) `MultiPartEmail, ImageHtmlEmail e HtmlEmail.`
d) `EmailAttachment, HtmlEmail e ImageHtmlEmail.`
e) `Email, MultiPartEmail e ImageHtmlEmail.`

Referências

BERNERS-LEE, T. **Universal Resource Identifiers in WWW. Jun. 1994.** Disponível em: <https://tools.ietf.org/html/rfc1630>. Acesso em: 20 set. 2018.

BERNERS-LEE, T.; MASINTER, L.; MCCAHILL, M. **Uniform Resource Locators (URL). Dez. 1994.** Disponível em: <https://tools.ietf.org/html/rfc1738>. Acesso em: 20 set. 2018.

BOND, M.; et al. **Aprenda J2EE:** Com EJB, JSP, Servlets, JNDI, JDBC e XML. São Paulo: Pearson Education, 2003.

DEITEL, P.; DEITEL, H. **Java como programar.** São Paulo: Pearson Education do Brasil, 2010. Tradução de: Edson Furmankiewicz.

ECLIPSE. **Eclipse IDE for Java EE Developers.** [S.d.]. Disponível em: <https://www.eclipse.org/downloads/packages/release/oxygen/3a/eclipse-ide-java-ee-developers>. Acesso em: 21 set. 2018.

FIELDING, R. T. **Architectural Styles and the Design of Network-based Software Architectures.** 2000. 162 f. Tese (Doutorado) - Curso de Information and Computer Science, University of California, Irvine, 2000. Disponível em: <https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>. Acesso em: 20 set. 2018

_____.; et al. **Hypertext Transfer Protocol -- HTTP/1.1. Jun**. 1999. Disponível em: <https://tools.ietf.org/html/rfc2616>. Acesso em: 20 set. 2018.

GOOGLE. **Google Maps Platform.** [S.d.]. Disponível em: <https://cloud.google.com/maps-platform/>. Acesso em: 20 set. 2018.

_____. **Less secure app access.** [S.d.]. Disponível em: <https://myaccount.google.com/lesssecureapps>. Acesso em: 21 set. 2018.

KLENSIN, J. **Simple Mail Transfer Protocol.** Out. 2008. Disponível em: <https://tools.ietf.org/html/rfc5321>. Acesso em: 21 set. 2018.

MASINTER, L.; SOLLINS, K. **Functional Requirements for Uniform Resource Names. Dez. 1994.** Disponível em: <https://tools.ietf.org/html/rfc1737>. Acesso em: 20 set. 2018.

ORACLE. **JSRs:** Java Specifications Requests. [S.d.] Disponível em: <https://jcp.org/en/jsr/detail?id=375>. Acesso em: 21 set. 2018.

_____. **Chapter 13 Building RESTful Web Services with JAX-RS.** [S.d.]. Disponível em: <https://docs.oracle.com/cd/E19798-01/821-1841/6nmq2cp1v/index.html>. Acesso em: 20 set. 2018.

_____. **Class Date.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/util/Date.html>. Acesso em: 25 jul. 2018.

_____. **Class DateTimeFormatter.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/format/DateTimeFormatter.html>. Acesso em: 24 jul. 2018.

_____. **Class Duration.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/Duration.html>. Acesso em: 24 jul. 2018.

_____. **Class Instant.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/9/docs/api/java/time/Instant.html>. Acesso em: 30 jul. 2018.

_____. **Class LocalDate.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDate.html>. Acesso em: 23 jul. 2018.

_____. **Class LocalDateTime.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/LocalDateTime.html>. Acesso em: 23 jul. 2018.

_____ **Class LocalTime.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/LocalTime.html>. Acesso em: 23 jul. 2018.

_____. **Class Period.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/Period.html>. Acesso em: 24 jul. 2018.

_____. **Class ZonedDateTime.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/ZonedDateTime.html>. Acesso em: 24 jul. 2018.

_____. **Class ZoneId.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/ZoneId.html>. Acesso em: 24 jul. 2018.

_____. **Enum Month.** [S.d.]. Disponível em: <https://docs.oracle.com/javase/8/docs/api/java/time/Month.html>. Acesso em: 24 jul. 2018.

_____. **Java SE 8 Date and Time.** 2014. Disponível em: <https://www.oracle.com/technetwork/articles/java/jf14-date-time-2125367.html>. Acesso em: 25 jul. 2018.

_____. **What Are RESTful Web Services?** [S.d.]. Disponível em: <https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html>. Acesso em: 20 set. 2018.

_____. **What Are Web Services?** [S.d.]. Disponível em: <https://docs.oracle.com/cd/E19798-01/821-1841/gijvh/index.html>. Acesso em: 20 set. 2018.

PRIMETEK INFORMATICS. **FileUpload** - Basic. [S.d.]. Disponível em: <https://www.primefaces.org/showcase/ui/file/upload/basic.xhtml>. Acesso em: 21 set. 2018.

PROCESS, Java Community. **JSR 310: Date and Time API.** [S.d.]. Disponível em: <https://jcp.org/en/jsr/detail?id=310>. Acesso em: 25 jul. 2018.

THE APACHE SOFTWARE CORPORATION. **Apache Commons**. 19 set. 2018, V. 19. Disponível em: <http://commons.apache.org/>. Acesso em: 21 set. 2018.

W3SCHOOLS. **HTML enctype Attribute.** [S.d.]. Disponível em: <https://www.w3schools.com/tags/att_form_enctype.asp>. Acesso em: 01 ago. 2018.

ISBN 978-85-522-1167-9