



K9

TREINAMENTOS

Desenvolvimento Web Avançado com JSF2, EJB3.1 e CDI

Desenvolvimento Web Avançado com JSF 2.2, EJB 3.2 e CDI 1.1

22 de agosto de 2015

As apostilas atualizadas estão disponíveis em www.k19.com.br

Esta apostila contém:

- 201 exercícios de fixação.
- 0 exercícios complementares.
- 0 desafios.
- 0 questões de prova.

| | |
|--|-----------|
| Sumário | i |
| Sobre a K19 | 1 |
| Seguro Treinamento | 2 |
| Termo de Uso | 3 |
| Cursos | 4 |
| 1 Enterprise JavaBeans | 1 |
| 1.1 Introdução | 1 |
| 1.2 EJB Container | 1 |
| 1.3 Exercícios de Fixação | 2 |
| 2 Stateless Session Beans | 21 |
| 2.1 Session Beans | 21 |
| 2.2 Caracterizando os SLSBs | 21 |
| 2.3 SLSB - EJB 3.0 | 22 |
| 2.4 SLSB - EJB 3.1 | 23 |
| 2.5 Cliente Java Web Local - EJB 3.0 | 24 |
| 2.6 Exercícios de Fixação | 25 |
| 2.7 Cliente Java Web Local - EJB 3.1 | 34 |
| 2.8 Exercícios de Fixação | 34 |
| 2.9 Cliente Java SE Remoto | 40 |

| | |
|---|------------|
| 2.10 Exercícios de Fixação | 41 |
| 2.11 Ciclo de Vida | 46 |
| 2.12 Escalabilidade e Pool | 47 |
| 2.13 Callbacks | 47 |
| 2.14 Exercícios de Fixação | 48 |
| 2.15 Métodos Assíncronos | 50 |
| 2.16 Exercícios de Fixação | 51 |
| 3 Stateful Session Beans | 55 |
| 3.1 Caracterizando os SFSBs | 55 |
| 3.2 SFSB - EJB 3.0 | 56 |
| 3.3 SFSB - EJB 3.1 | 58 |
| 3.4 Exercícios de Fixação | 58 |
| 3.5 Ciclo de Vida | 62 |
| 3.6 Callbacks | 64 |
| 3.7 Exercícios de Fixação | 66 |
| 4 Singleton Session Beans | 75 |
| 4.1 Caracterizando os Singleton Session Beans | 75 |
| 4.2 Implementação | 76 |
| 4.3 Exercícios de Fixação | 78 |
| 4.4 Ciclo de Vida | 83 |
| 4.5 Exercícios de Fixação | 85 |
| 4.6 Concorrência | 86 |
| 4.7 Exercícios de Fixação | 88 |
| 5 Persistência | 95 |
| 5.1 Data Sources | 95 |
| 5.2 Exercícios de Fixação | 95 |
| 5.3 persistence.xml | 102 |
| 5.4 Entity Beans | 103 |
| 5.5 Entity Classes e Mapeamento | 103 |
| 5.6 Exercícios de Fixação | 104 |
| 5.7 Entity Managers | 109 |
| 5.8 Entity Manager Factories | 110 |
| 5.9 Exercícios de Fixação | 111 |
| 6 Transações | 115 |
| 6.1 ACID | 115 |
| 6.2 Transação Local ou Distribuída | 115 |
| 6.3 JTA e JTS | 115 |
| 6.4 Container Managed Transactions - CMT | 116 |
| 6.5 Bean Managed Transactions - BMT | 118 |
| 6.6 Exercícios de Fixação | 119 |
| 7 Segurança | 127 |
| 7.1 Realms | 127 |
| 7.2 Exercícios de Fixação | 127 |
| 7.3 Autenticação - Aplicações Web | 132 |
| 7.4 Exercícios de Fixação | 132 |

| | |
|---|------------|
| 7.5 Autorização - Aplicações EJB | 144 |
| 7.6 Exercícios de Fixação | 146 |
| 8 Interceptadores | 149 |
| 8.1 Interceptor Methods | 149 |
| 8.2 Internal Interceptors | 150 |
| 8.3 External Interceptors | 150 |
| 8.4 Excluindo Interceptadores | 152 |
| 8.5 Invocation Context | 152 |
| 8.6 Ordem dos Interceptadores | 153 |
| 8.7 Exercícios de Fixação | 153 |
| 9 Scheduling | 163 |
| 9.1 Timers | 163 |
| 9.2 Métodos de Timeout | 163 |
| 9.3 Timers Automáticos | 164 |
| 9.4 Exercícios de Fixação | 165 |
| 10 Contexts and Dependency Injection - CDI | 175 |
| 10.1 Managed Beans | 175 |
| 10.2 Producer Methods and Fields | 176 |
| 10.3 EL Names | 176 |
| 10.4 beans.xml | 177 |
| 10.5 Exercícios de Fixação | 178 |
| 10.6 Escopos e Contextos | 183 |
| 10.7 Injection Points | 184 |
| 10.8 Exercícios de Fixação | 185 |
| A Projeto | 187 |
| A.1 Exercícios de Fixação | 187 |





Sobre a K19

A K19 é uma empresa especializada na capacitação de desenvolvedores de software. Sua equipe é composta por profissionais formados em Ciência da Computação pela Universidade de São Paulo (USP) e que possuem vasta experiência em treinamento de profissionais para área de TI.

O principal objetivo da K19 é oferecer treinamentos de máxima qualidade e relacionados às principais tecnologias utilizadas pelas empresas. Através desses treinamentos, seus alunos tornam-se capacitados para atuar no mercado de trabalho.

Visando a máxima qualidade, a K19 mantém as suas apostilas em constante renovação e melhoria, oferece instalações físicas apropriadas para o ensino e seus instrutores estão sempre atualizados didática e tecnicamente.



Seguro Treinamento

Na K19 o aluno faz o curso quantas vezes quiser!

Comprometida com o aprendizado e com a satisfação dos seus alunos, a K19 é a única que possui o Seguro Treinamento. Ao contratar um curso, o aluno poderá refazê-lo quantas vezes desejar mediante a disponibilidade de vagas e pagamento da franquia do Seguro Treinamento.

As vagas não preenchidas até um dia antes do início de uma turma da K19 serão destinadas ao alunos que desejam utilizar o Seguro Treinamento. O valor da franquia para utilizar o Seguro Treinamento é 10% do valor total do curso.



Termo de Uso

Termo de Uso

Todo o conteúdo desta apostila é propriedade da K19 Treinamentos. A apostila pode ser utilizada livremente para estudo pessoal . Além disso, este material didático pode ser utilizado como material de apoio em cursos de ensino superior desde que a instituição correspondente seja reconhecida pelo MEC (Ministério da Educação) e que a K19 seja citada explicitamente como proprietária do material.

É proibida qualquer utilização desse material que não se enquadre nas condições acima sem o prévio consentimento formal, por escrito, da K19 Treinamentos. O uso indevido está sujeito às medidas legais cabíveis.



Conheça os nossos cursos

-  K01 - Lógica de Programação
-  K02 - Desenvolvimento Web com HTML, CSS e JavaScript
-  K03 - SQL e Modelo Relacional
-  K11 - Orientação a Objetos em Java
-  K12 - Desenvolvimento Web com JSF2 e JPA2
-  K21 - Persistência com JPA2 e Hibernate
-  K22 - Desenvolvimento Web Avançado com JFS2, EJB3.1 e CDI
-  K23 - Integração de Sistemas com Webservices, JMS e EJB
-  K41 - Desenvolvimento Mobile com Android
-  K51 - Design Patterns em Java
-  K52 - Desenvolvimento Web com Struts
-  K31 - C# e Orientação a Objetos
-  K32 - Desenvolvimento Web com ASP.NET MVC

www.k19.com.br/cursos

ENTERPRISE JAVA BEANS



Introdução

Muitos sistemas corporativos são desenvolvidos seguindo a arquitetura definida pelo padrão Enterprise JavaBeans (EJB). Ao utilizar essa arquitetura, diversos recursos são disponibilizados a esses sistemas.

Transações: A arquitetura EJB define um suporte sofisticado para utilização de transações. Esse suporte é integrado com a Java Transaction API (JTA) e oferece inclusive a possibilidade de realizar transações distribuídas.

Segurança: Suporte para realizar autenticação e autorização de forma transparente. Os desenvolvedores das aplicações não precisam implementar a lógica de segurança pois ela faz parte da arquitetura EJB.

Remotabilidade: Aplicações EJB podem ser acessadas remotamente através de diversos protocolos de comunicação. Consequentemente, é possível desenvolver aplicações clientes de diversos tipos. Por exemplo, aplicações EJB podem ser como Web Services.

Multithreading e Concorrência: A arquitetura EJB permite que as aplicações sejam acessados por múltiplos usuários simultaneamente de maneira controlada para evitar problemas de concorrência.

Persistência: Facilidades para utilizar os serviços dos provedores de persistência que seguem a especificação JPA.

Gerenciamento de Objetos: Mecanismos de injeção de dependências e controle de ciclo de vida são oferecidos aos objetos de uma aplicação EJB. O mecanismo de controle de ciclo de vida pode garantir a escalabilidade de uma aplicação.

Integração: A arquitetura EJB é fortemente integrada com os componentes da plataforma Java EE. Podemos, por exemplo, facilmente integrar os recursos do JSF em uma aplicação EJB.



EJB Container

Toda aplicação EJB é executada e gerenciada por um EJB Container. Há diversas opções de EJB Container disponíveis. Os servidores de aplicação Java EE como o Glassfish e o JBoss possuem um EJB Container. Portanto, podemos utilizá-los para executar as nossas aplicações EJB.



Exercícios de Fixação

- 1 Copie o arquivo **glassfish-4.1.zip** da pasta **K19-Arquivos** para a sua Área de Trabalho. Depois, descompacte esse arquivo.



Importante

Você também pode obter o arquivo **glassfish-4.1.zip** através do site da K19: www.k19.com.br/arquivos.

- 2 Copie o arquivo **jboss-as-7.1.1.Final.zip** da pasta **K19-Arquivos** para a sua Área de Trabalho. Depois, descompacte esse arquivo.



Importante

Você também pode obter o arquivo **jboss-as-7.1.1.Final.zip** através do site da K19: www.k19.com.br/arquivos.

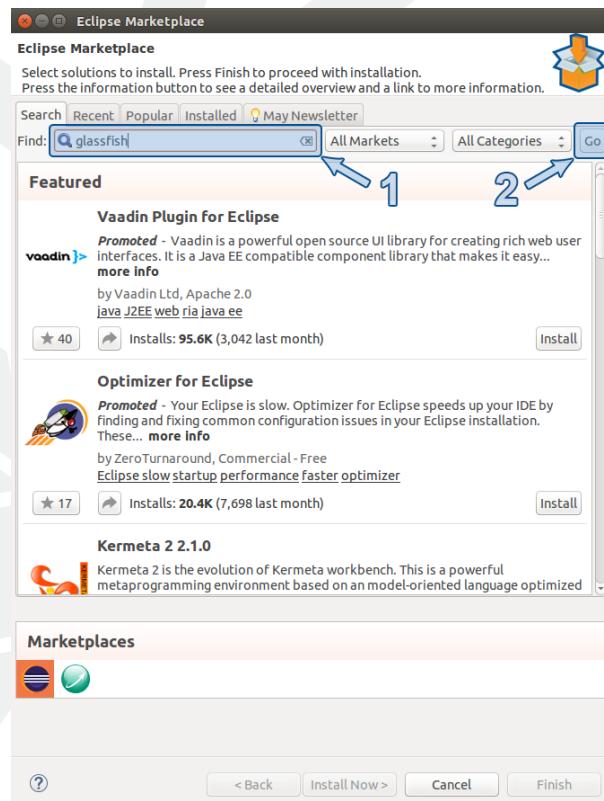
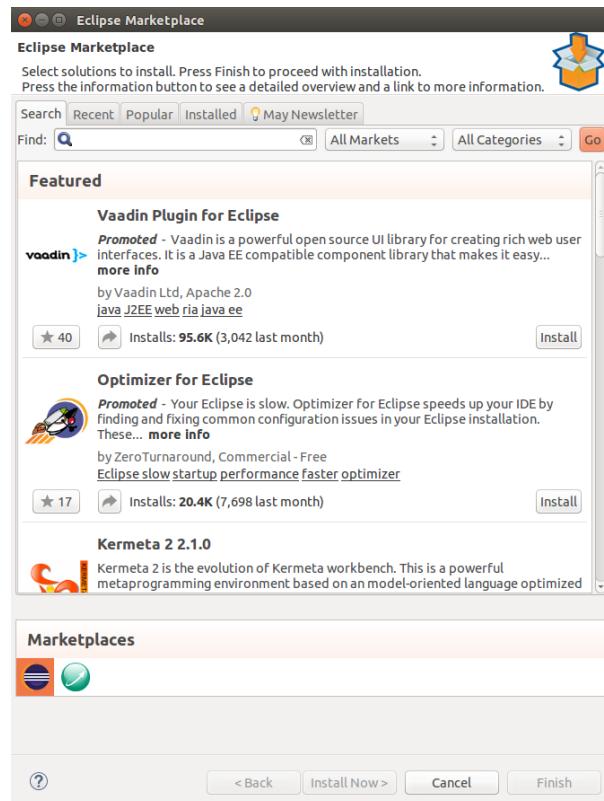
- 3 Copie o arquivo **wildfly-8.2.0.Final.zip** da pasta **K19-Arquivos** para a sua Área de Trabalho. Depois, descompacte esse arquivo.

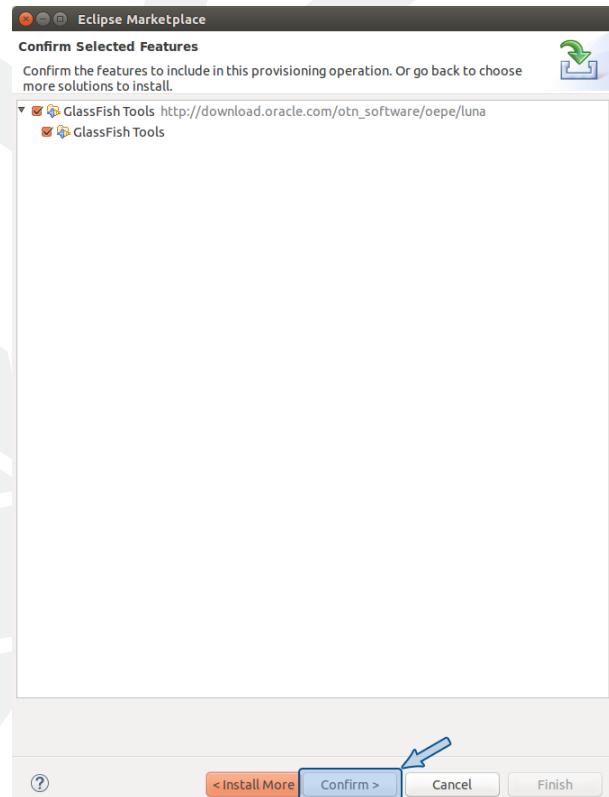
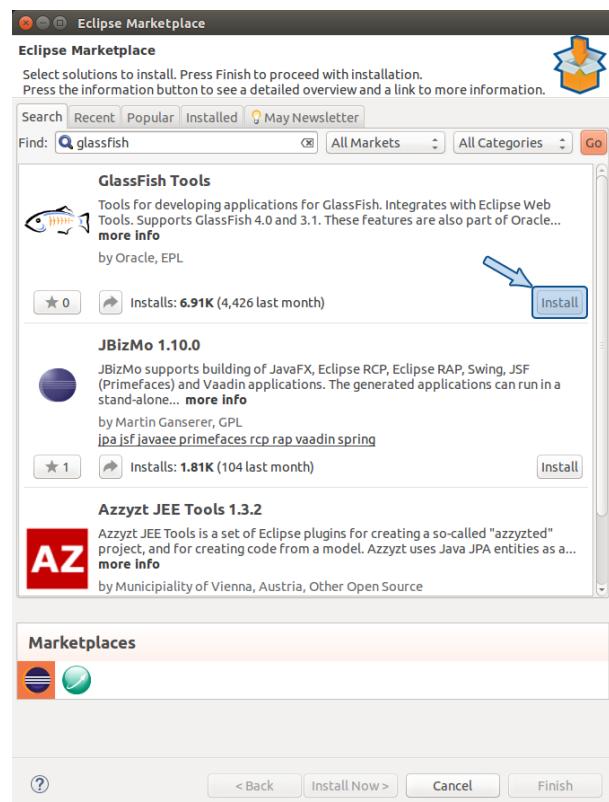


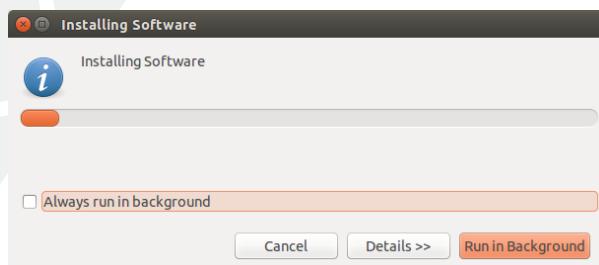
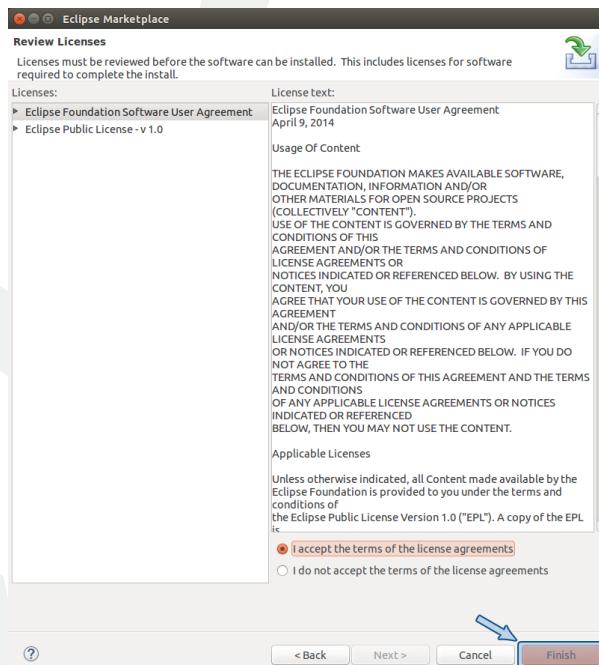
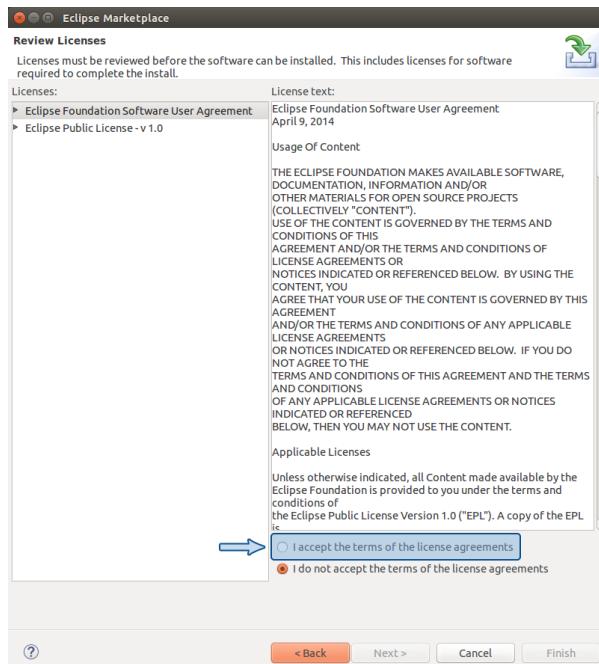
Importante

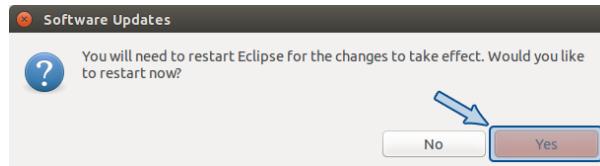
Você também pode obter o arquivo **wildfly-8.2.0.Final.zip** através do site da K19: www.k19.com.br/arquivos.

- 4 Utilizando o Eclipse Marketplace, adicione no Eclipse Luna o suporte ao Glassfish. Digite “CTRL+3” para abrir o Quick Access. Em seguida, pesquise por “Eclipse Marketplace”.

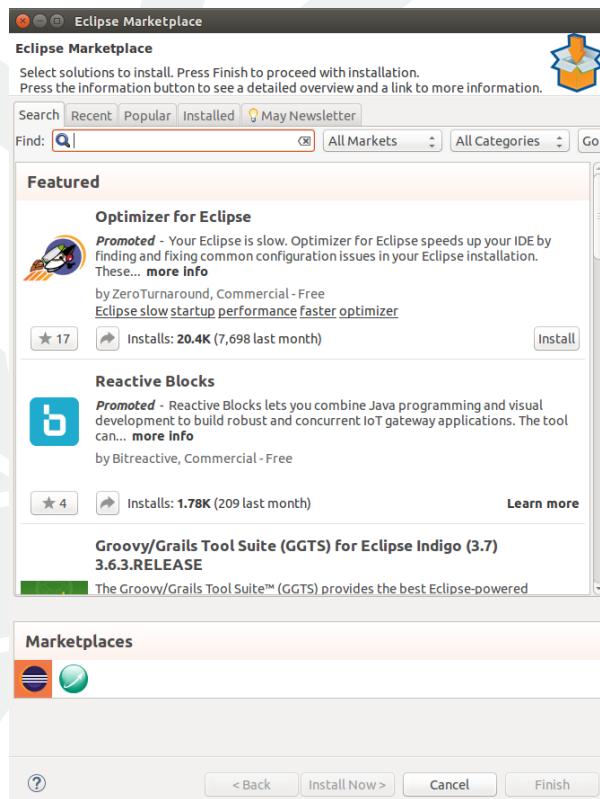


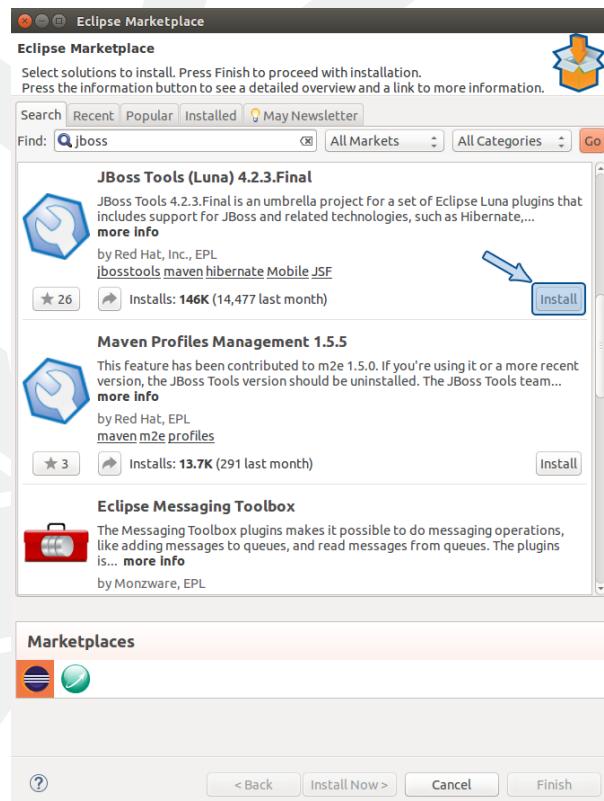
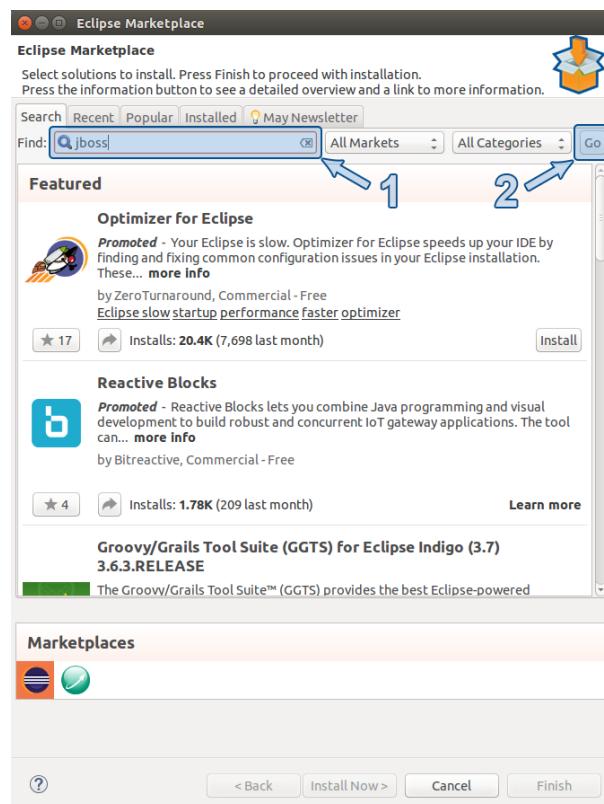


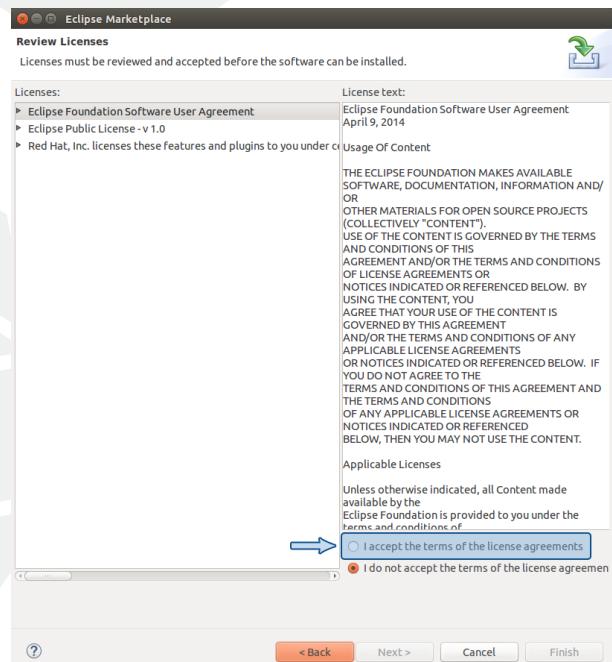
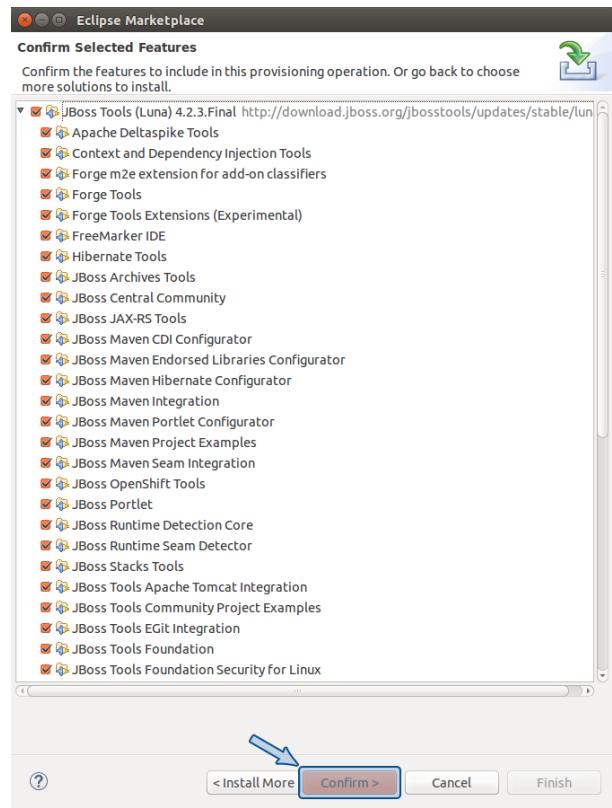


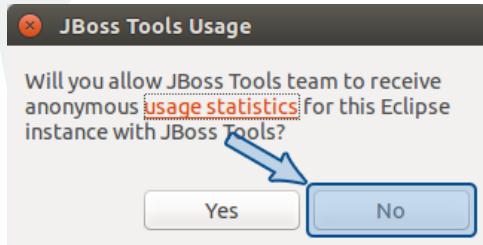
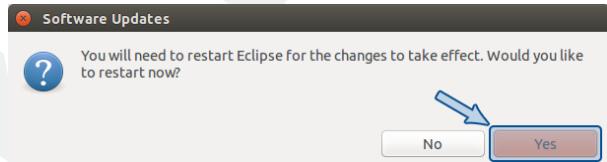
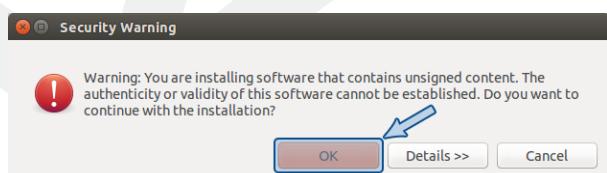
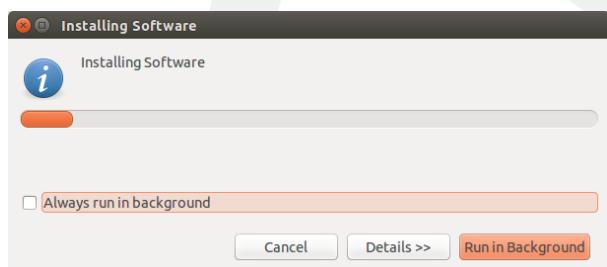
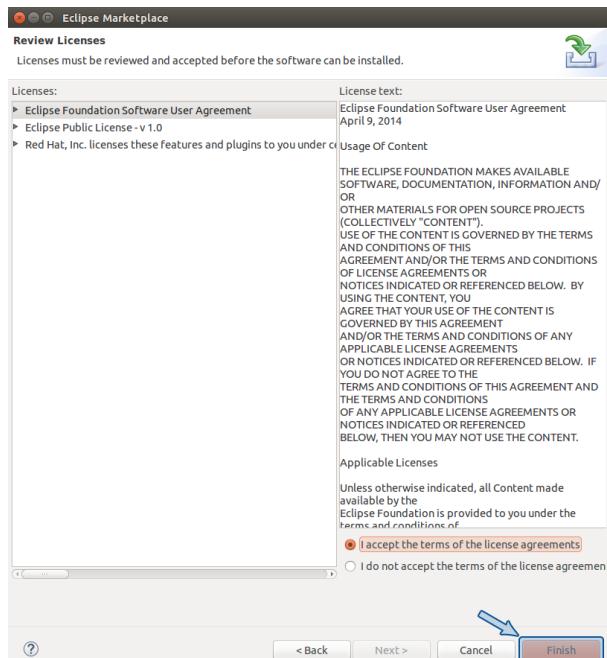


- 5 Utilizando o Eclipse Marketplace, adicione no Eclipse Luna o suporte ao JBoss e ao Wildfly. Digite “CTRL+3” para abrir o Quick Access. Em seguida, pesquise por “Eclipse Marketplace”.

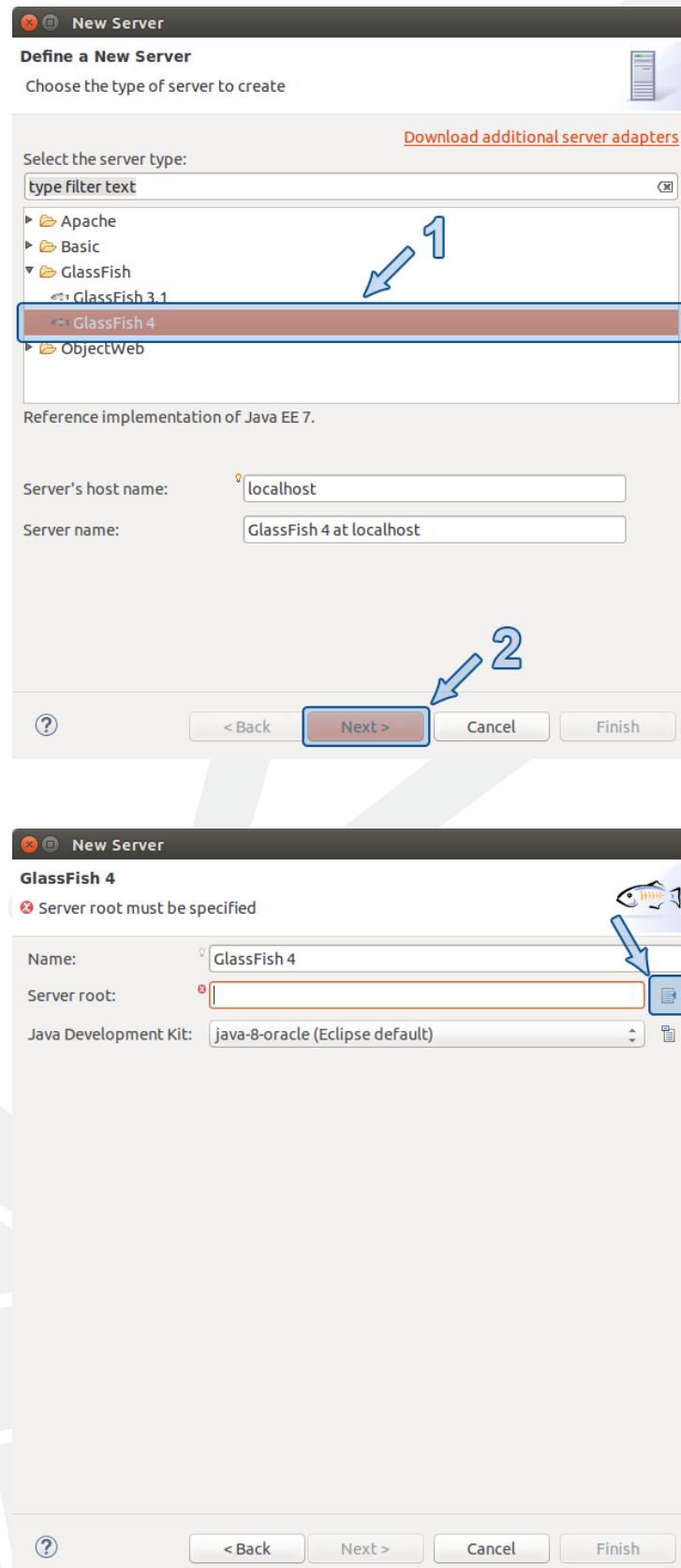


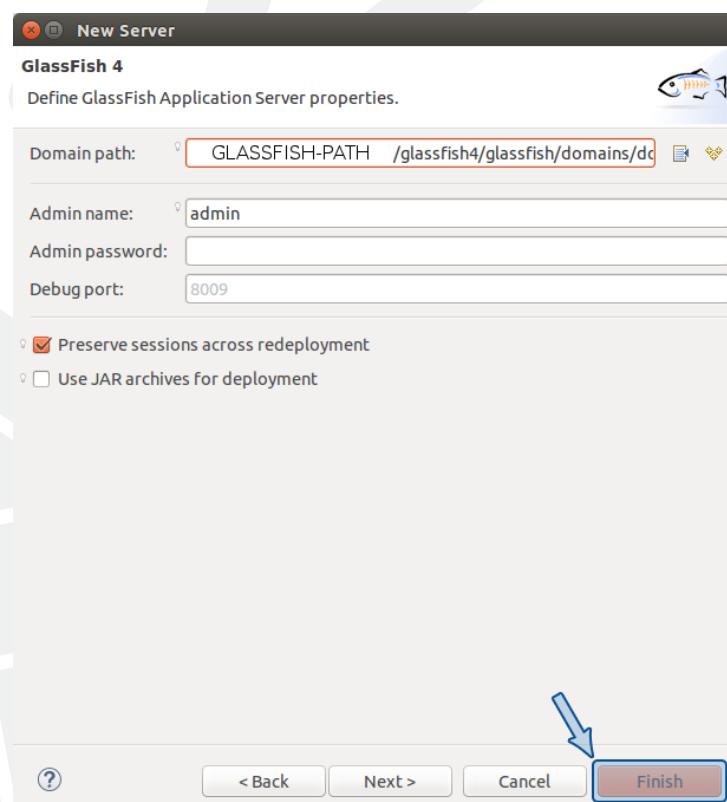
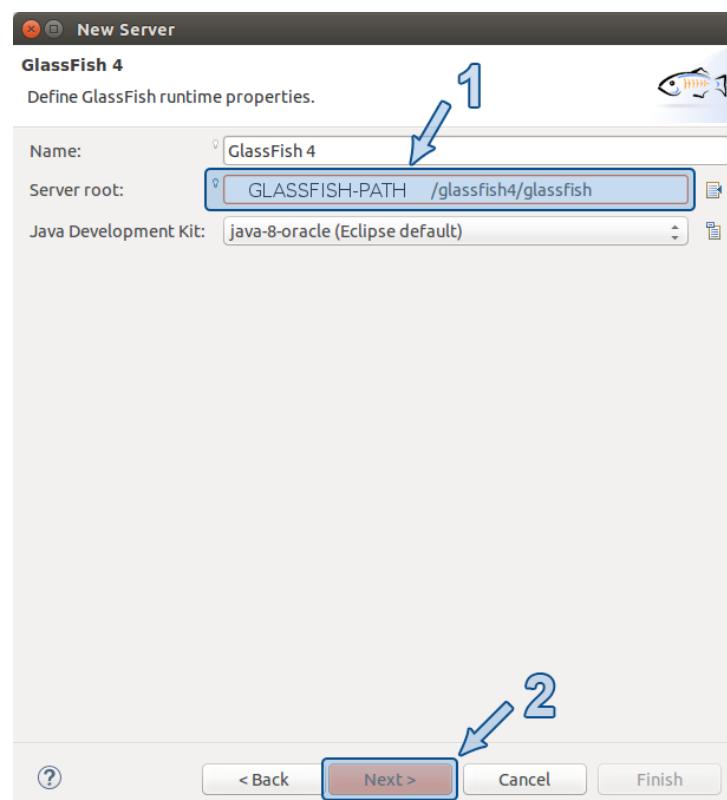


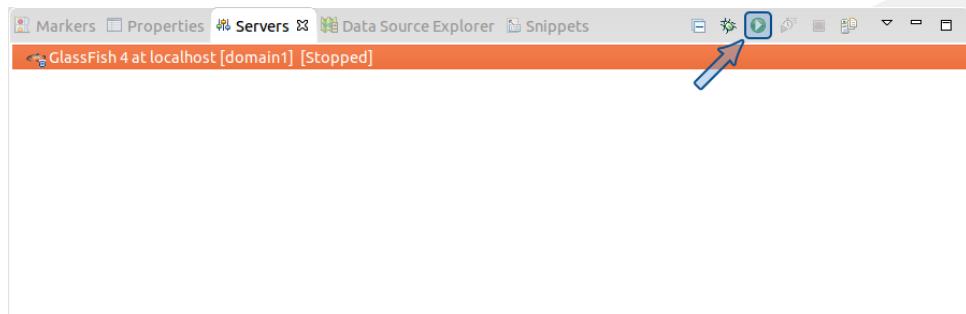




- 6 Configure o Glassfish no Eclipse Luna. Digite “CTRL+3” para abrir o Quick Access. Em seguida, pesquise por “Define a new server”.



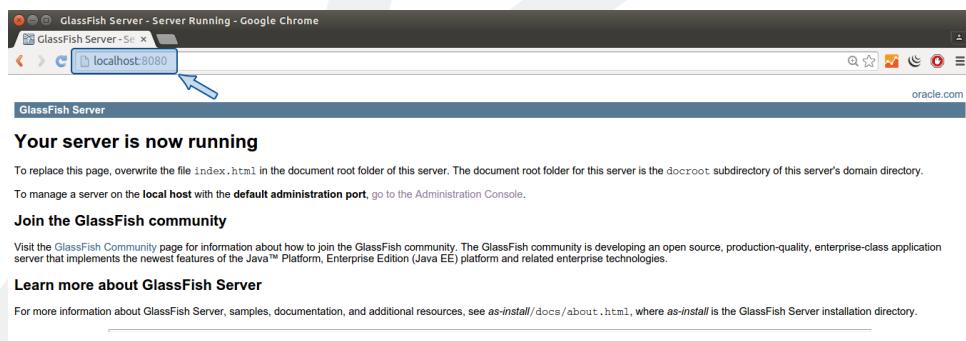




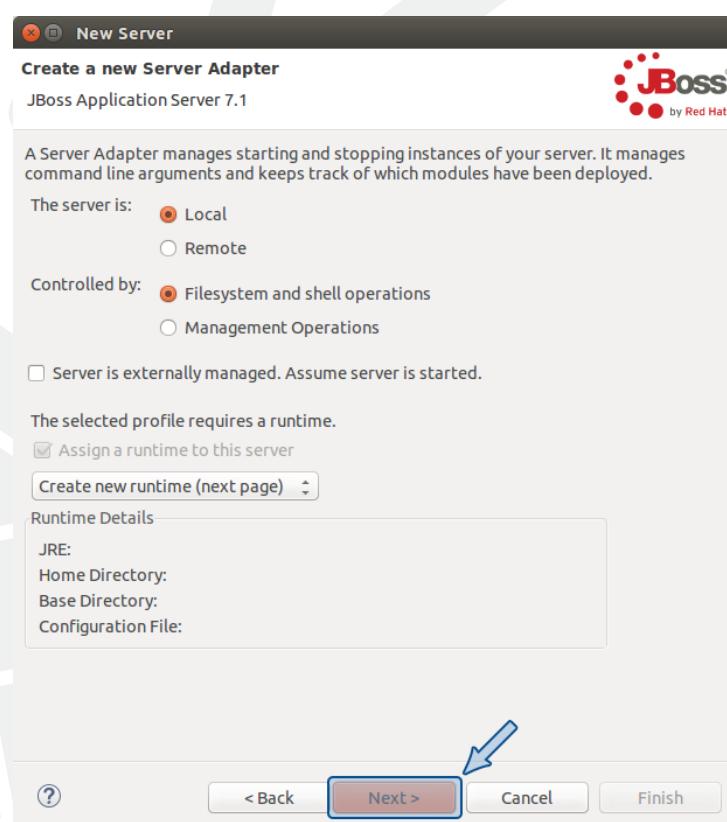
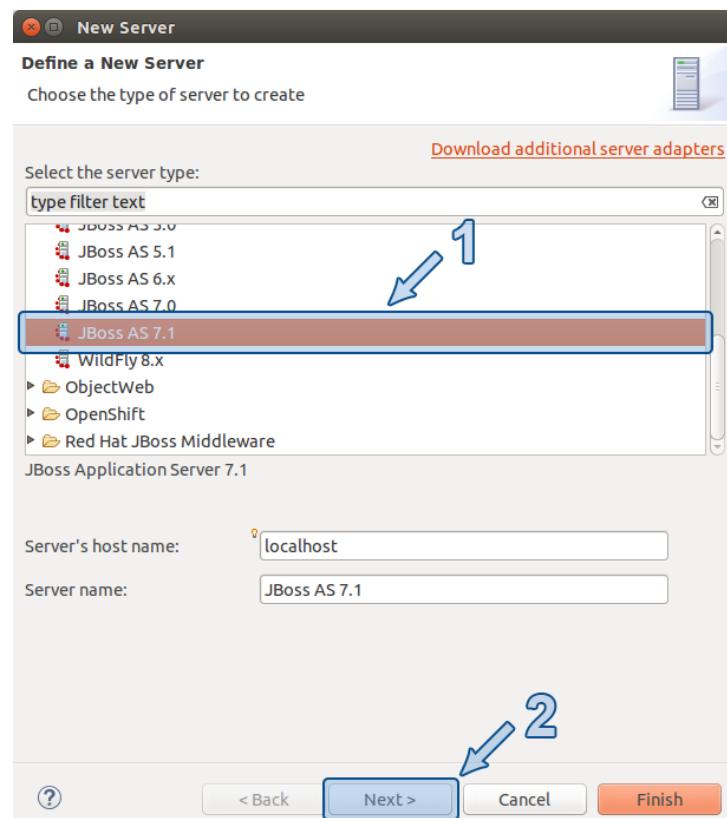
```

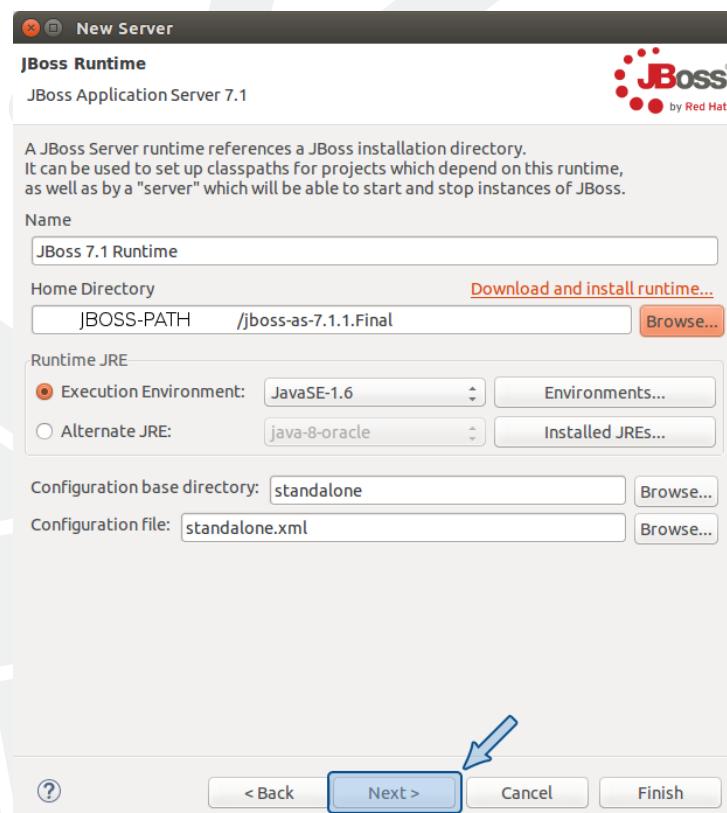
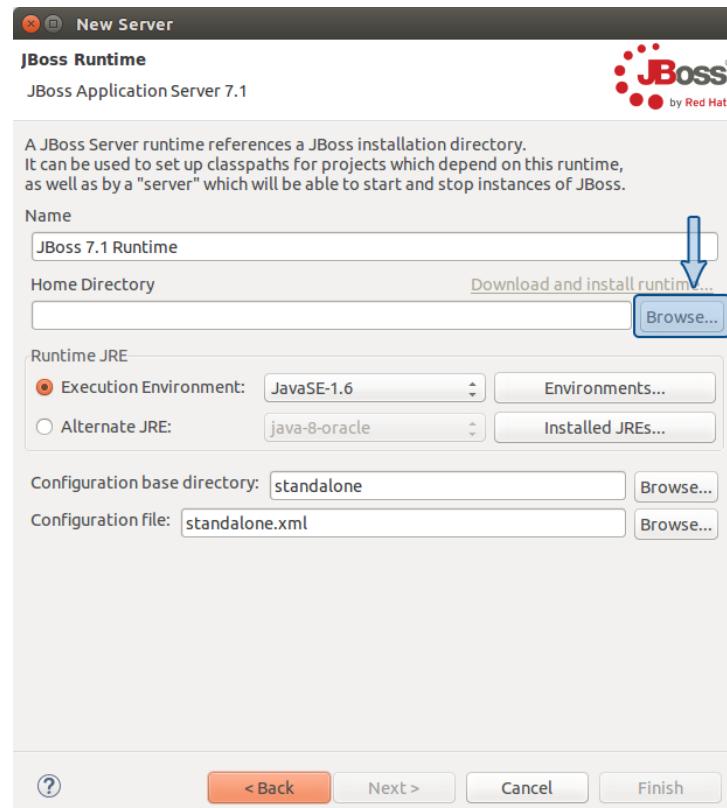
GlassFish 4 at localhost [domain1]
2015-06-17T16:17:42.843-0300|Info: Virtual server server loaded default web module
2015-06-17T16:17:43.079-0300|Info: Java security manager is disabled.
2015-06-17T16:17:43.081-0300|Info: Entering Security Startup Service.
2015-06-17T16:17:43.084-0300|Info: Loading policy provider com.sun.enterprise.security.provider.PolicyWrapper
2015-06-17T16:17:43.139-0300|Info: Security Service(s) started successfully.
2015-06-17T16:17:43.374-0300|Info: visiting unvisited references
2015-06-17T16:17:43.396-0300|Info: visiting unvisited references
2015-06-17T16:17:43.398-0300|Info: visiting unvisited references
2015-06-17T16:17:44.965-0300|Info: Initializing Mojarra 2.2.7 ( 20140610-1547 https://svn.java.net/svn/mojarra~trunk@1547 )
2015-06-17T16:17:46.072-0300|Info: Loading application [_admingui] at [/]
2015-06-17T16:17:46.074-0300|Info: Loading application _admingui done in 4,206 ms

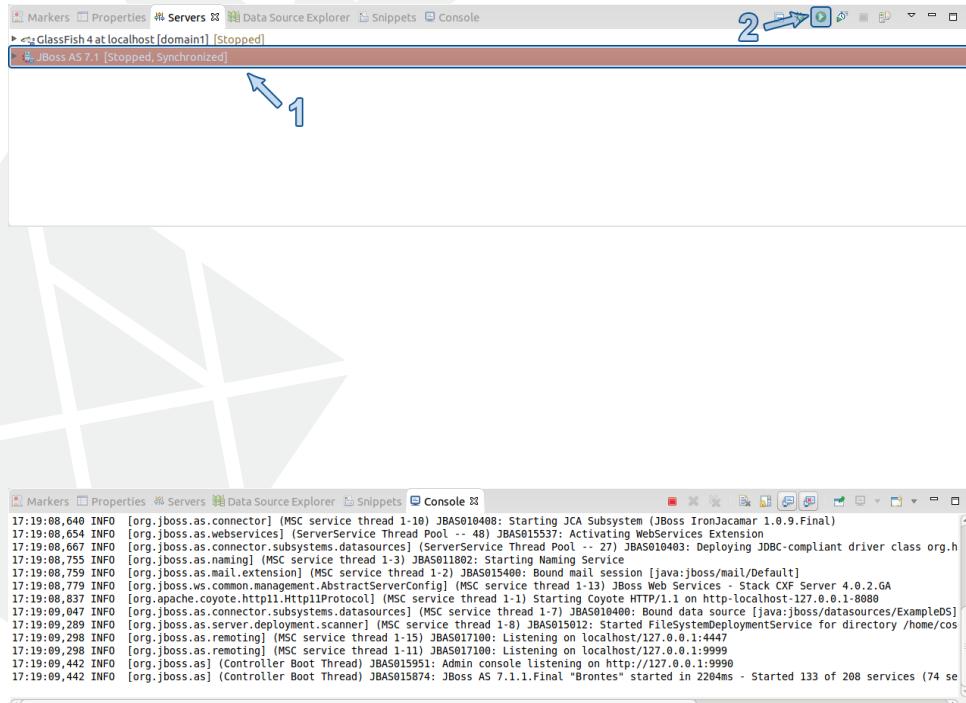
```



- 7 Pare o Glassfish. Em seguida, configure o JBoss no Eclipse Luna. Digite “CTRL+3” para abrir o Quick Access. Em seguida, pesquise por “Define a new server”.

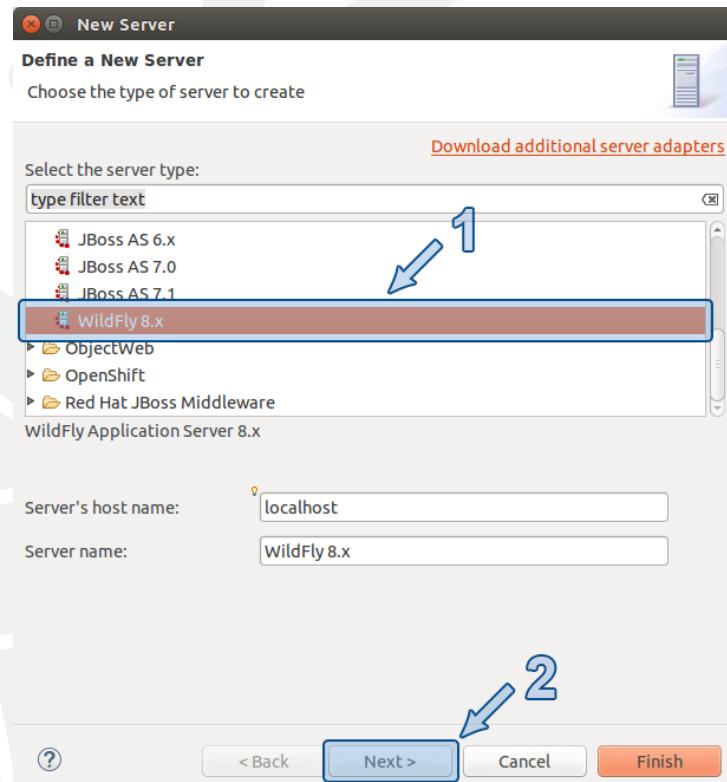


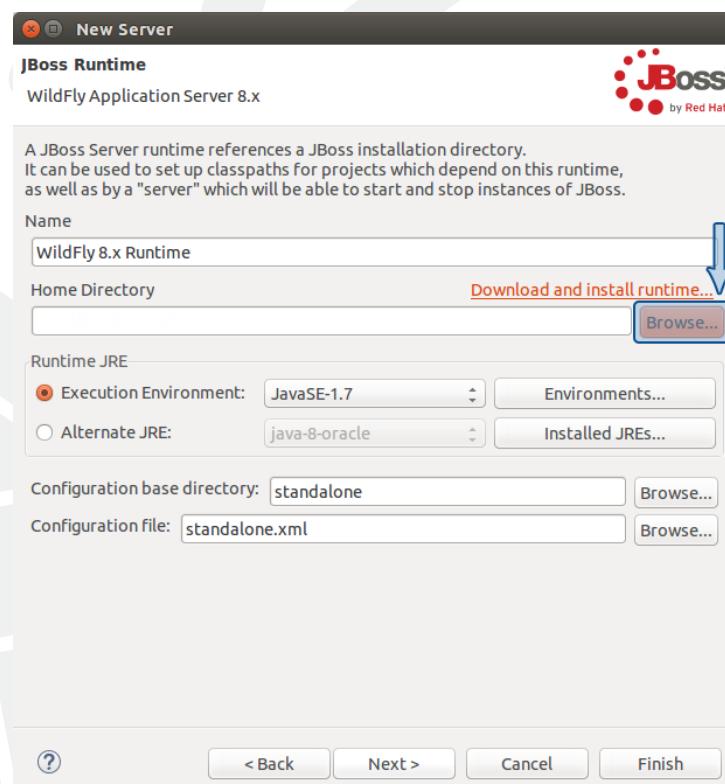
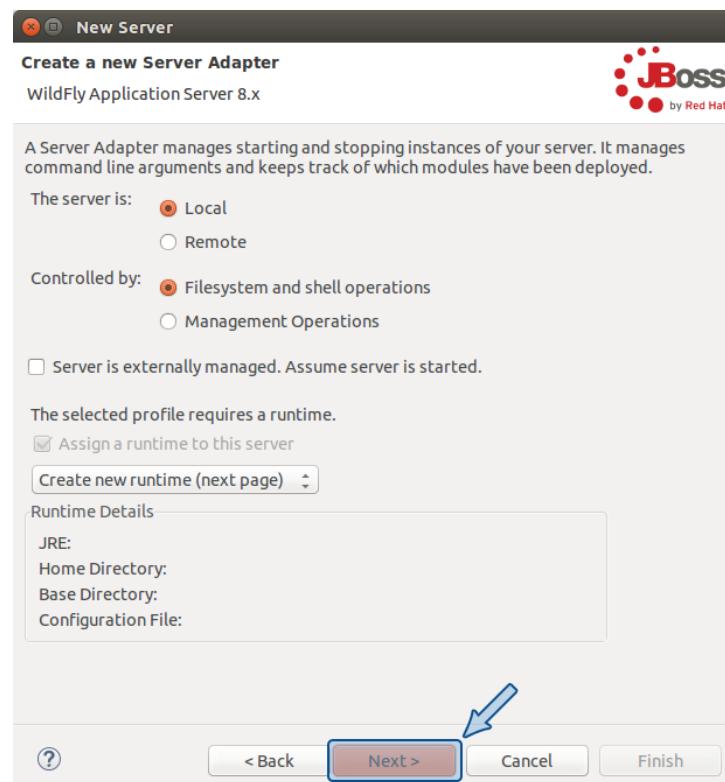






- 8 Pare o JBoss. Em seguida, configure o Wildfly no Eclipse Luna. Digite “CTRL+3” para abrir o Quick Access. Em seguida, pesquise por “Define a new server”.







Markers Properties Servers Data Source Explorer Snippets Console

WildFly 8.x [Stopped] **1**

GlassFish 4 at localhost [domain1] [Stopped]

JBoss AS 7.1 [Stopped, Synchronized]

JBoss AS 8.1 [Stopped] **2**

```

17:22:52,861 INFO [org.jboss.naming] (MSC service thread 1-13) JBAS011802: Starting Naming Service
17:22:52,862 INFO [org.jboss.as.mail.extension] (MSC service thread 1-12) JBAS015400: Bound mail session [java:jboss/mail/Default]
17:22:53,118 INFO [org.jboss.remoting] (MSC service thread 1-7) JBoss Remoting version 4.0.6.Final
17:22:53,153 INFO [org.wildfly.extension.undertow] (MSC service thread 1-1) JBAS017520: Creating file handler for path /home/cosen/servers-k22/wil
17:22:53,159 INFO [org.wildfly.extension.undertow] (MSC service thread 1-1) JBAS017525: Started server default-server
17:22:53,189 INFO [org.wildfly.extension.undertow] (MSC service thread 1-1) JBAS017531: Host default-host starting
17:22:53,242 INFO [org.wildfly.extension.undertow] (MSC service thread 1-6) JBAS017519: Undertow HTTP listener default listening on localhost/127.0.0.1:808
17:22:53,416 INFO [org.jboss.as.connector.subsystems.datasources] (MSC service thread 1-4) JBAS010400: Bound data source [java:jboss/datasources/ExampleDS]
17:22:53,486 INFO [org.jboss.as.server.deployment.scanner] (MSC service thread 1-4) JBAS015612: Started FileSystemDeploymentService for directory /home/cos
17:22:53,953 INFO [org.jboss.ws.common.management] (MSC service thread 1-13) JBWS022052: Starting JBoss Web Services - Stack CXF Server 4.3.2.Final
17:22:54,178 INFO [org.jboss.as] (Controller Boot Thread) JBAS015961: Http management interface listening on http://127.0.0.1:9990/management
17:22:54,180 INFO [org.jboss.as] (Controller Boot Thread) JBAS015951: Admin console listening on http://127.0.0.1:9990
17:22:54,181 INFO [org.jboss.as] (Controller Boot Thread) JBAS015874: Wildfly 8.2.0.Final "Week" started in 3152ms - Started 184 of 234 services (82 servi

```





STATELESS SESSION BEANS



Session Beans

Um sistema corporativo é composto por muitos processos ou tarefas. Por exemplo, um sistema bancário possui processos específicos para realizar transferências, depósitos, saques, empréstimos, cobranças, entre outros. Esses procedimentos são chamados de regras de negócio. Cada aplicação possui as suas próprias regras de negócio já que elas são consequência imediata do contexto da aplicação.

Utilizando a arquitetura EJB, as regras de negócio são implementadas em componentes específicos que são chamados de **Session Beans**. O EJB Container administra esses componentes oferecendo diversos recursos a eles.



Caracterizando os SLSBs

Stateless Session Bean é o primeiro tipo de Session Bean. Muitas vezes, utilizaremos a sigla SLSB para fazer referência a esse tipo de componente. A característica fundamental dos SLSBs é que eles não armazenam estado conversacional. Vejamos alguns exemplos a seguir.

Serviço de Câmbio

Considere um sistema de conversão monetária. Esse sistema é capaz de converter valores monetários de uma moeda para outra. Por exemplo, converter 100 reais para o valor correspondente em dólar americano. Poderíamos, implementar essa regra de negócio através do método a seguir.

```
1 public double converte(double valor, String moedaOrigem, String moedaDestino) {  
2     // lógica da conversão monetária  
3 }
```

A execução do método `converte()` não depende das suas execuções anteriores. Em outras palavras, o método `converte()` não precisa manter estado conversacional.

Dicionário

Considere a implementação de um dicionário digital de português. Dado uma palavra, o dicionário digital deve devolver a definição dela. Podemos criar um método para implementar essa regra de negócio.

```
1 public String getDefinicao(String palavra) {  
2     // lógica do dicionário  
3 }
```

As chamadas ao método `getDefinicao()` são totalmente independentes. Dessa forma, não é necessário guardar informações referentes às chamadas anteriores. Ou seja, não é necessário manter estado conversacional.

Consulta de CEP

Considere um sistema de consulta de CEP. Esse sistema é capaz de informar o CEP de uma determinada localidade. Podemos criar um método para implementar essa regra de negócio.

```
1 public String consultaCEP(String estado, String cidade, String logradouro, Integer ←
2   numero) {
3   // lógica da consulta do CEP
4 }
```

Como cada consulta de CEP independe das consultas anteriores, não é necessário manter dados entre uma consulta e outra. Em outras palavras, não é necessário manter estado conversacional.



SLSB - EJB 3.0

O primeiro passo para implementar um SLSB é definir os seus métodos de negócios através de uma interface. Por exemplo, suponha um SLSB que realiza operações matemáticas. Uma possível interface para esse SLSB seria:

```
1 public interface Calculadora {
2   double soma(double a, double b);
3   double subtrai(double a, double b);
4   double multiplica(double a, double b);
5   double divide(double a, double b);
6 }
```

Código Java 2.4: *Calculadora.java*

Os métodos contidos na interface `Calculadora` são chamados de métodos de negócios.

Depois da interface, o segundo passo seria implementar as operações do SLSB através de uma classe.

```
1 public class CalculadoraBean implements Calculadora {
2
3   public double soma(double a, double b){
4     return a + b;
5   }
6
7   public double subtrai(double a, double b) {
8     return a - b;
9   }
10
11  public double multiplica(double a, double b) {
12    return a * b;
13  }
14
15  public double divide(double a, double b) {
16    return a / b;
17  }
18 }
```

Código Java 2.5: *CalculadoraBean.java*

Observe que a classe CalculadoraBean implementa a interface Calculadora.

O terceiro passo é especificar o tipo de Session Bean que queremos utilizar. No caso da calculadora, o tipo seria SLSB. Essa definição é realizada através da anotação **@Stateless**.

```
1 @Stateless
2 public class CalculadoraBean implements Calculadora {
3     ...
4 }
```

Código Java 2.6: CalculadoraBean.java

Por fim, é necessário definir se o acesso do SLSB é local ou remoto. Quando o acesso é local, apenas quem está dentro do servidor de aplicação no qual se encontra o SLSB pode acessá-lo. Quando o acesso é remoto, tanto quem está dentro quanto quem está fora do servidor de aplicação no qual se encontra o SLSB pode acessá-lo.

A definição do tipo de acesso pode ser realizada através das anotações: **@Local** e **@Remote**.

```
1 @Stateless
2 @Remote(Calculadora.class)
3 public class CalculadoraBean implements Calculadora {
4     ...
5 }
```

Código Java 2.7: CalculadoraBean.java

```
1 @Stateless
2 @Local(Calculadora.class)
3 public class CalculadoraBean implements Calculadora {
4     ...
5 }
```

Código Java 2.8: CalculadoraBean.java

Nas anotações **@Local** e **@Remote**, devemos informar as interfaces que definem os métodos de negócio do nosso SLSB. A classe CalculadoraBean poderia implementar diversas interfaces. Contudo, apenas os métodos das interfaces declaradas nessas anotações serão considerados métodos de negócio.

SLSB - EJB 3.1

Na versão 3.1, quando o acesso a um SLSB é local, não é mais necessário definir uma interface java nem utilizar a anotação **@Local**. Então, bastaria criar uma classe java com a anotação **@Stateless**.

```
1 @Stateless
2 public class CalculadoraBean {
3
4     public double soma(double a, double b){
5         return a + b;
6     }
7
8     public double subtrai(double a, double b) {
9         return a - b;
10    }
11
12    public double multiplica(double a, double b) {
```

```

13     return a * b;
14 }
15
16 public double divide(double a, double b) {
17     return a / b;
18 }
19 }
```

Código Java 2.9: CalculadoraBean.java

Os métodos públicos dos objetos da classe CalculadoraBean serão considerados métodos de negócio.

Cliente Java Web Local - EJB 3.0

Como vimos os session beans são utilizados para implementar as regras negócios das nossas aplicações. Em particular, os stateless session beans são utilizados para implementar as regras de negócio que não necessitam manter estado conversacional. Contudo, além das regras de negócio, devemos nos preocupar com a interface dos usuários. Hoje em dia, na maioria dos casos, essa interface é web.

A seguir, vamos implementar um sistema que realiza operações matemáticas básicas para exemplificar a utilização da arquitetura EJB em conjunto com as tecnologias Java para desenvolvimento de interfaces web. Suponha que todo o sistema (session beans e a camada web) esteja no mesmo servidor de aplicação.

A seguinte interface Java será utilizada para definir as operações de um SLSB.

```

1 public interface Calculadora {
2     double soma(double a, double b);
3 }
```

Código Java 2.10: Calculadora.java

Depois de definir a interface, devemos implementar as operações do SLSB através de uma classe java com as anotações apropriadas.

```

1 @Stateless
2 @Local(Calculadora.class)
3 public class CalculadoraBean implements Calculadora {
4     public double soma(double a, double b) {
5         return a + b;
6     }
7 }
```

Código Java 2.11: CalculadoraBean.java

Perceba que o acesso local foi definido para esse SLSB pois ele será acessado por uma camada web no mesmo servidor de aplicação. Nesse momento, o SLSB está pronto.

O próximo passo é implementar a camada web. A interface Calculadora que define as operações do SLSB precisa estar no classpath da camada web diferentemente da classe CalculadoraBean que implementa as operações.

Suponha que a camada web da nossa aplicação utiliza apenas Servlets. Podemos injetar, através da anotação @EJB, o SLSB em uma Servlet.

```

1  @WebServlet("/soma")
2  public class SomaServlet extends HttpServlet {
3      @EJB
4      private Calculadora calculadora;
5
6      protected void service(HttpServletRequest req, HttpServletResponse res) {
7          double a = Double.parseDouble(req.getParameter("a"));
8          double b = Double.parseDouble(req.getParameter("b"));
9
10         double resultado = this.calculadora.soma(a, b);
11
12         PrintWriter out = response.getWriter();
13         out.println("<html><body><p>");
14         out.println("Soma: " + resultado);
15         out.println("</p></body></html>");
16     }
17 }
```

Código Java 2.12: SomaServlet.java

Agora, suponha que a camada web utilize JSF. Podemos injetar o SLSB em um managed bean também através da anotação @EJB.

```

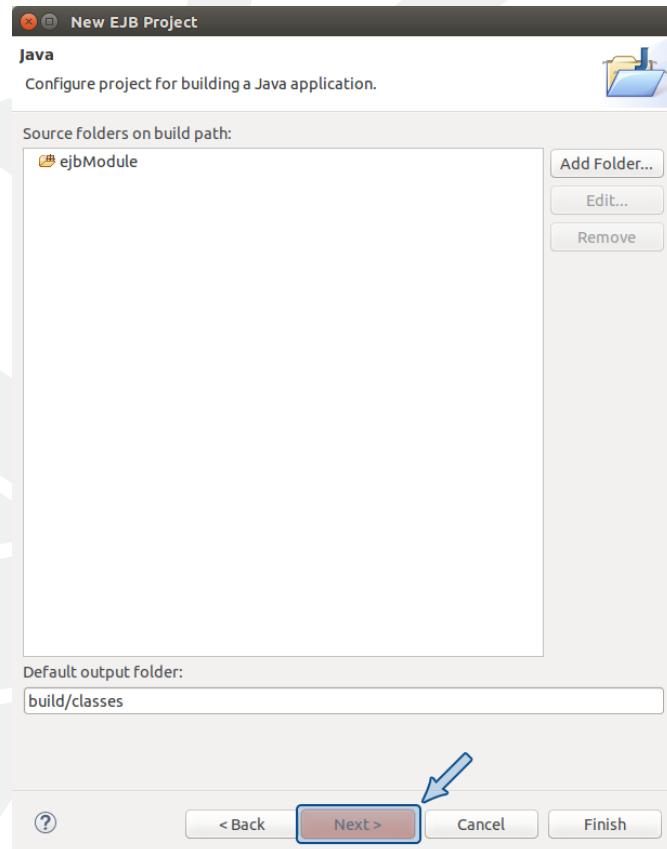
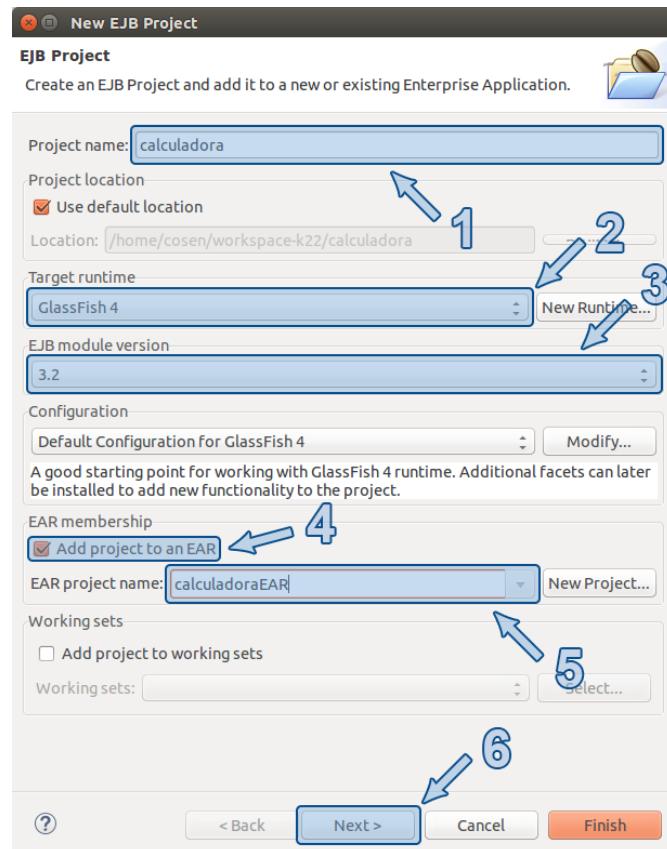
1  @ManagedBean
2  public class CalculadoraMB {
3      @EJB
4      private Calculadora calculadora;
5
6      private double a;
7
8      private double b;
9
10     private double resultado;
11
12     public void soma() {
13         this.resultado = this.calculadora.soma(a,b);
14     }
15
16     // GETTERS AND SETTERS
17 }
```

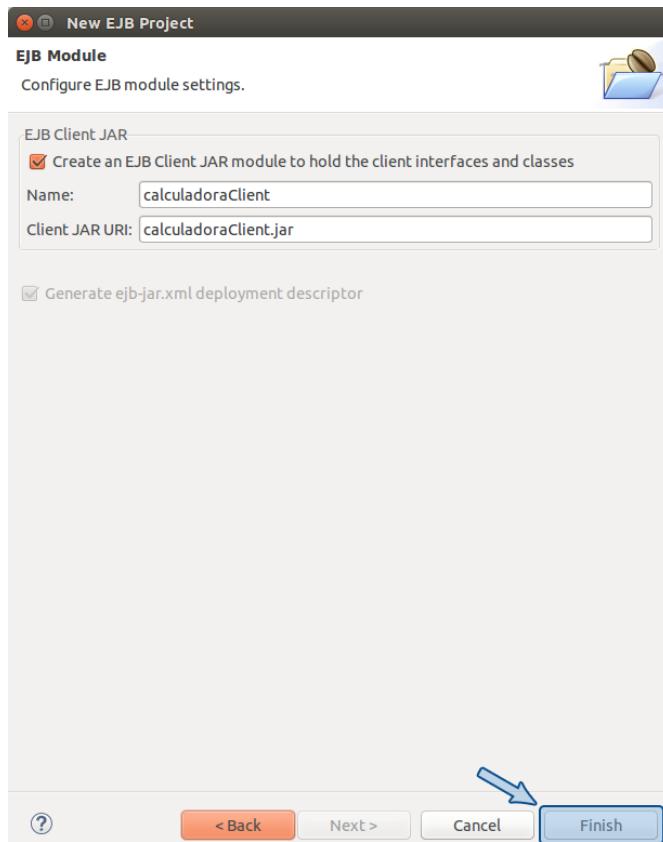
Código Java 2.13: CalculadoraMB.java



Exercícios de Fixação

- 1 Crie um EJB project no eclipse. Você pode digitar “CTRL+3” em seguida “new EJB project” e “ENTER”. Depois, siga exatamente as imagens abaixo.





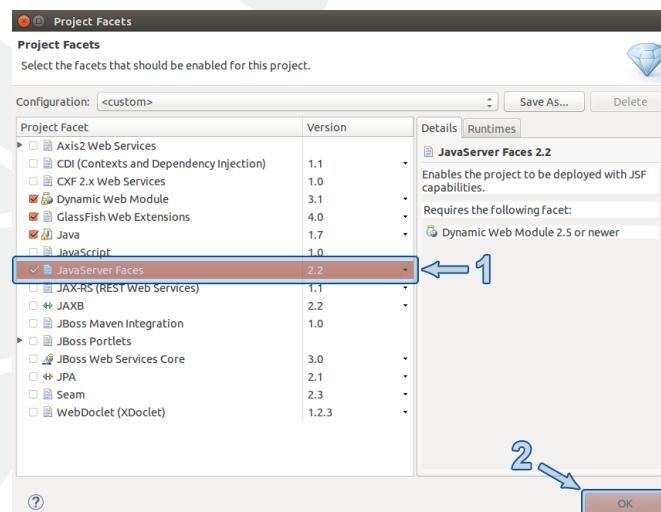
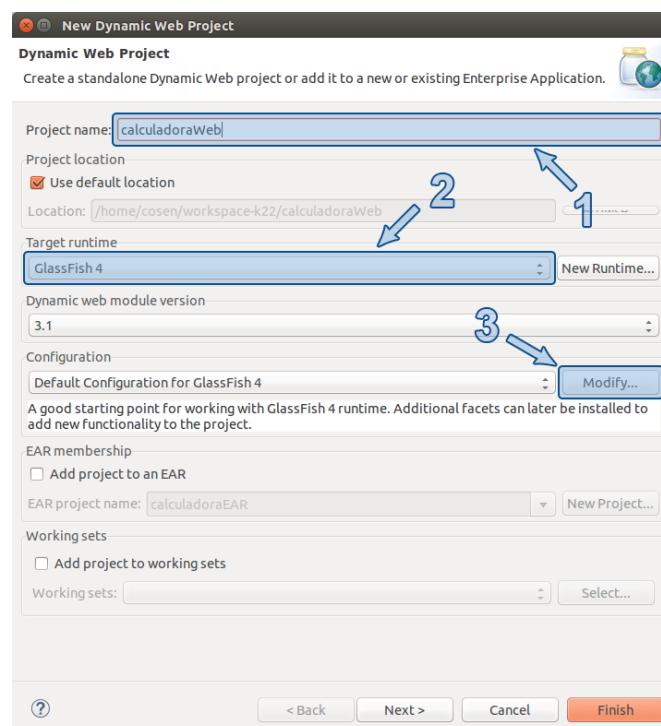
OBS: Os seguintes três projetos serão criados:

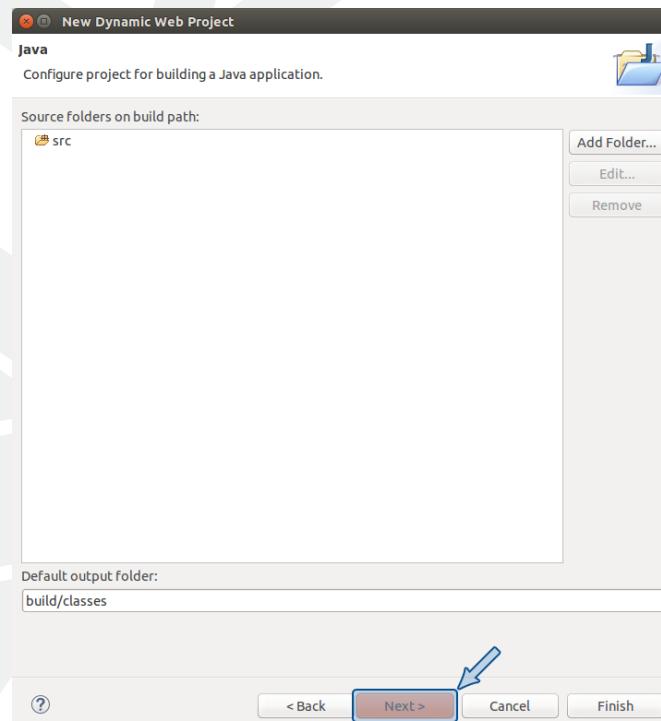
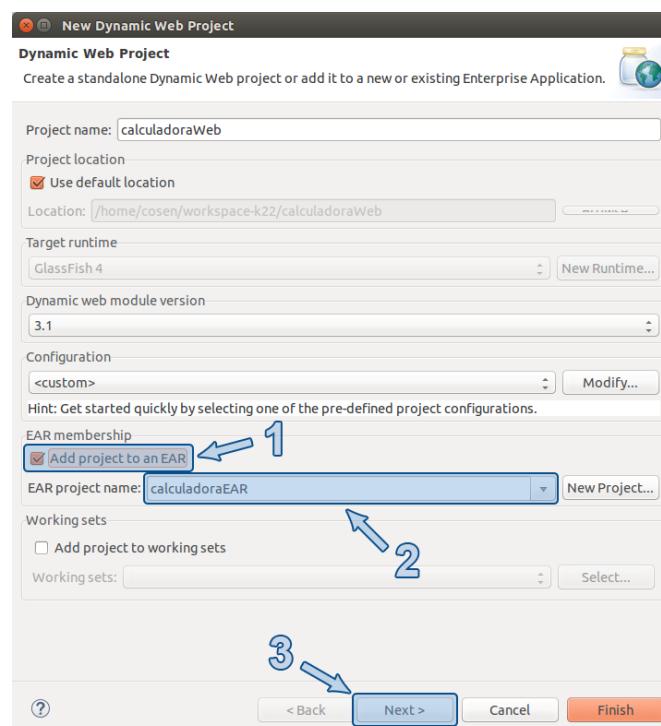
calculadora: As classes que implementam os SLSB devem ser colocadas nesse projeto.

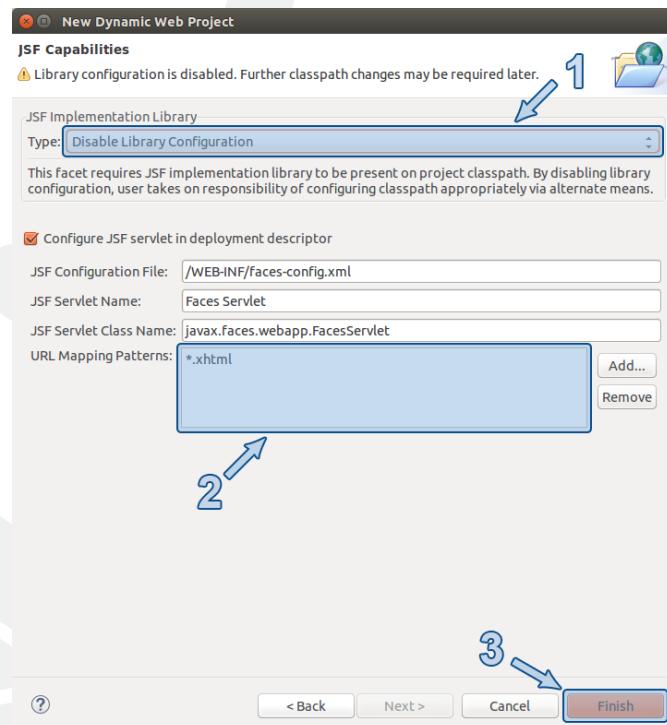
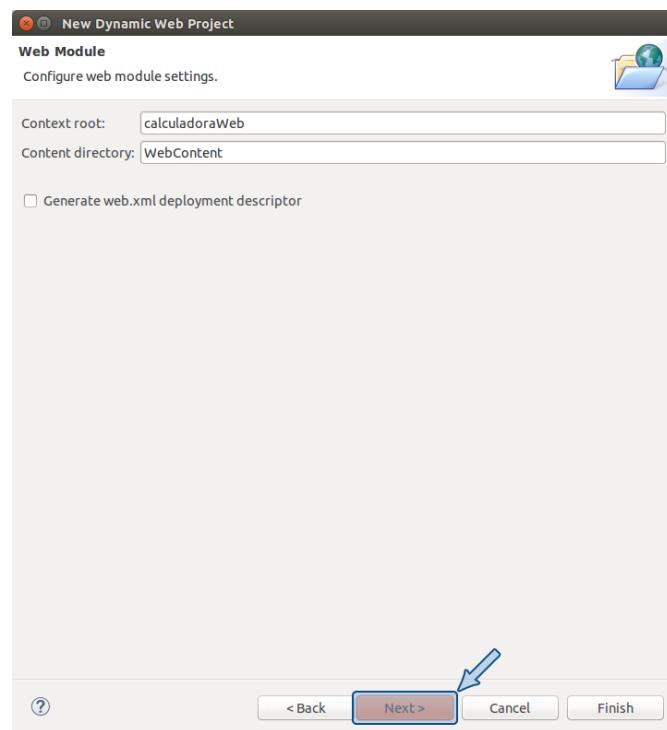
calculadoraClient: As interfaces que definem as operações dos SLSB devem ser colocadas nesse projeto.

calculadoraEAR: Esse projeto empacota todos os módulos da aplicação Java EE.

- 2 Crie um Dynamic Web Project no eclipse para implementar a camada web. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.

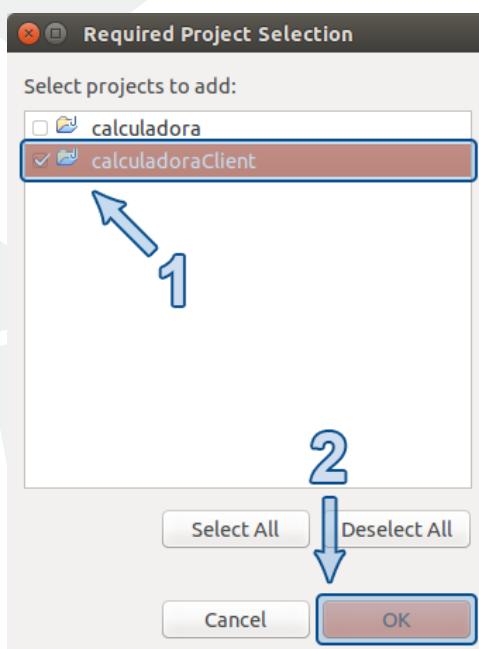
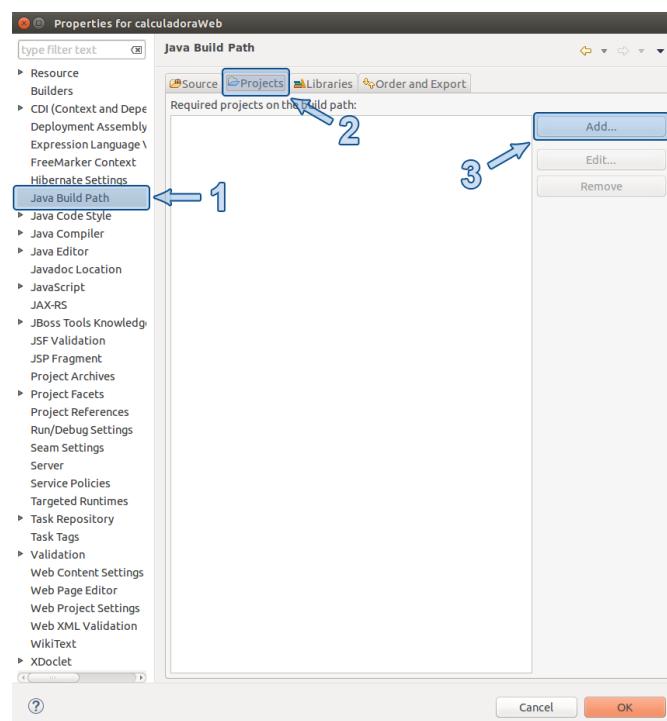


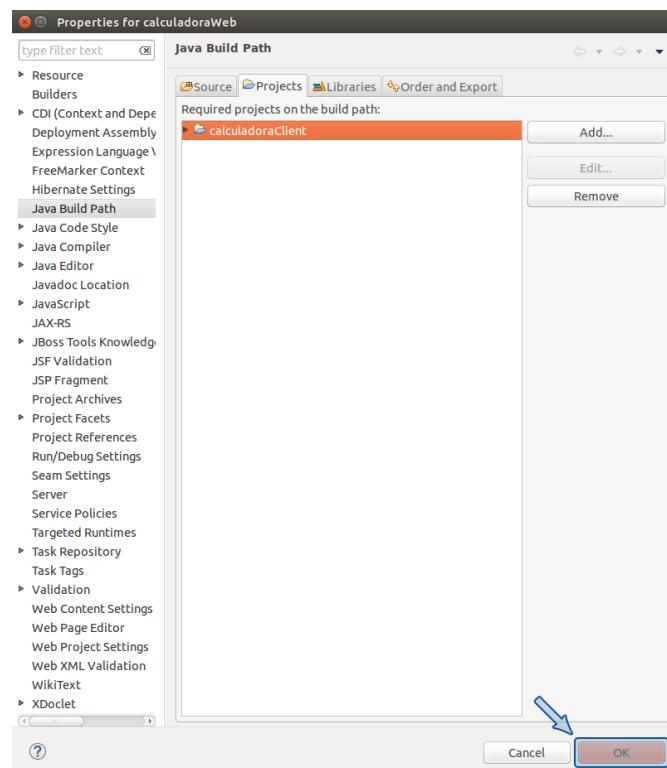




OBS: O módulo web já está vinculado ao projeto **calculadoraEAR**.

- 3 Adicione o projeto **calculadoraClient** como dependência do projeto **calculadoraWeb**. Abra as propriedades do projeto **calculadoraWeb**. Você pode selecionar o projeto **calculadoraWeb** e digitar "ALT+ENTER". Depois, siga as imagens abaixo.





- 4 No projeto **calculadoraClient**, crie uma interface java chamada **Calculadora** em um pacote chamado **br.com.k19.sessionbeans**.

```

1 package br.com.k19.sessionbeans;
2
3 public interface Calculadora {
4     double soma(double a, double b);
5 }
```

Código Java 2.14: Calculadora.java

- 5 No projeto **calculadora**, crie uma classe java chamada **CalculadoraBean** em um pacote chamado **br.com.k19.sessionbeans**.

```

1 package br.com.k19.sessionbeans;
2
3 import javax.ejb.Local;
4 import javax.ejb.Stateless;
5
6 @Stateless
7 @Local(Calculadora.class)
8 public class CalculadoraBean implements Calculadora {
9     public double soma(double a, double b) {
10         return a + b;
11     }
12 }
```

Código Java 2.15: CalculadoraBean.java

- 6 No projeto **calculadoraWeb**, crie uma classe java chamada **CalculadoraMB** em um pacote chamado **br.com.k19.managedbeans**.

```

1 package br.com.k19.managedbeans;
2
3 import javax.ejb.EJB;
4 import javax.faces.bean.ManagedBean;
5
6 import br.com.k19.sessionbeans.Calculadora;
7
8 @ManagedBean
9 public class CalculadoraMB {
10
11     @EJB
12     private Calculadora calculadora;
13
14     private double a;
15
16     private double b;
17
18     private double resultado;
19
20     public void soma() {
21         this.resultado = this.calculadora.soma(a, b);
22     }
23
24     // GETTERS AND SETTERS
25 }
```

Código Java 2.16: CalculadoraMB.java

Não esqueça de adicionar os métodos getters e setters

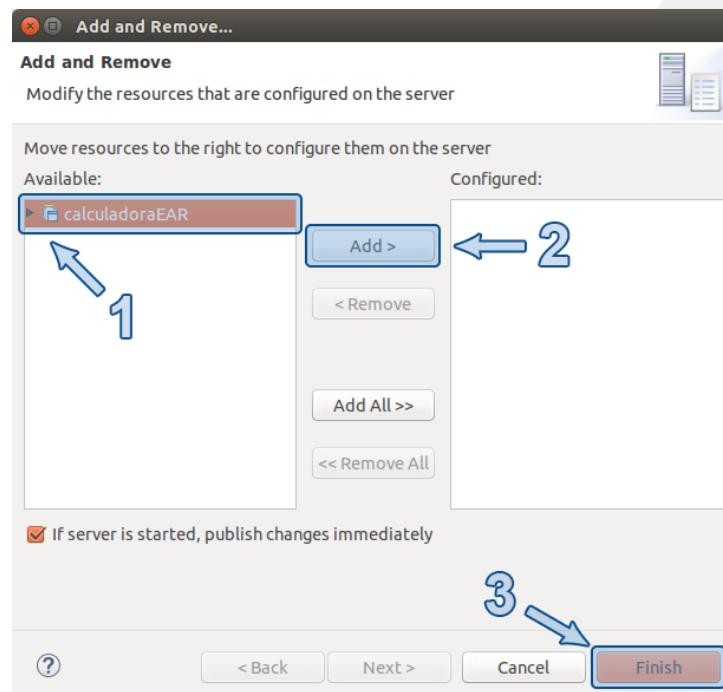
- 7 Crie uma simples tela na aplicação web para utilizar o managed bean. Adicione o arquivo **soma.xhtml** na pasta **WebContent** do projeto **calculadoraWeb** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:h="http://java.sun.com/jsf/html">
6
7 <h:head>
8     <title>Calculadora - Soma</title>
9 </h:head>
10
11 <h:body>
12     <h:form>
13         <h:outputLabel value="Valor A: " />
14         <h:inputText value="#{calculadoraMB.a}" />
15
16         <h:outputLabel value="Valor B: " />
17         <h:inputText value="#{calculadoraMB.b}" />
18
19         <h:commandButton action="#{calculadoraMB.soma}" value="soma" />
20
21         <h:outputLabel value="Resultado: " />
22         <h:outputText value="#{calculadoraMB.resultado}" />
23     </h:form>
24 </h:body>
25 </html>
```

Código XHTML 2.1: soma.xhtml

- 8 Adicione o projeto **calculadoraEAR** no **Glassfish**. Clique com o botão direito no **Glassfish** utilizando view Servers e escolha a opção “Add and Remove”.



- 9 Certifique-se que o **JBoss** e o **Wildfly** estejam parados. Inicie o **Glassfish**. Acesse a url <http://localhost:8080/calculadoraWeb/soma.xhtml> e teste o funcionamento da aplicação.



Cliente Java Web Local - EJB 3.1

Como dito anteriormente, na versão 3.1, quando o acesso a um SLSB é local, não é mais necessário definir uma interface Java nem utilizar a anotação **@Local**.

Além disso, as regras de empacotamento foram simplificadas. Os Session Beans podem ser empacotados no módulo web. Isso simplifica bastante o funcionamento das IDEs como o eclipse. Perceberemos essa diferença o exercício seguinte.

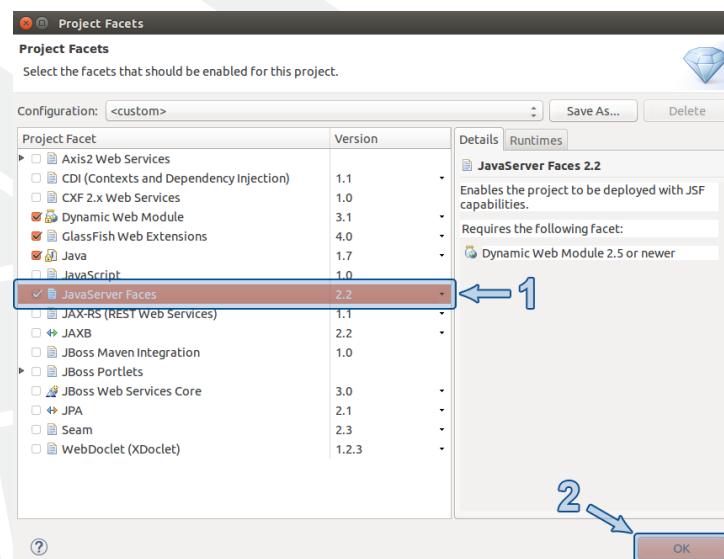
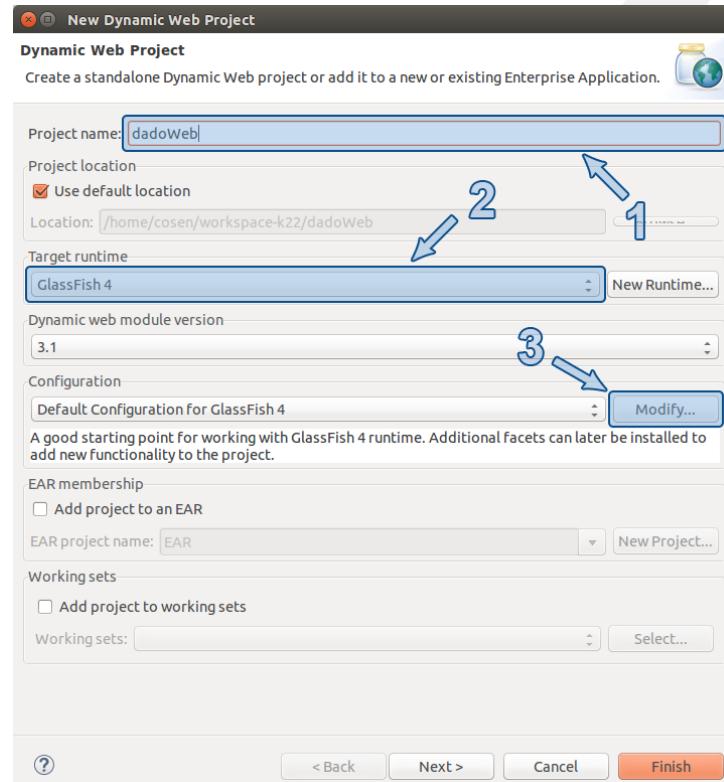


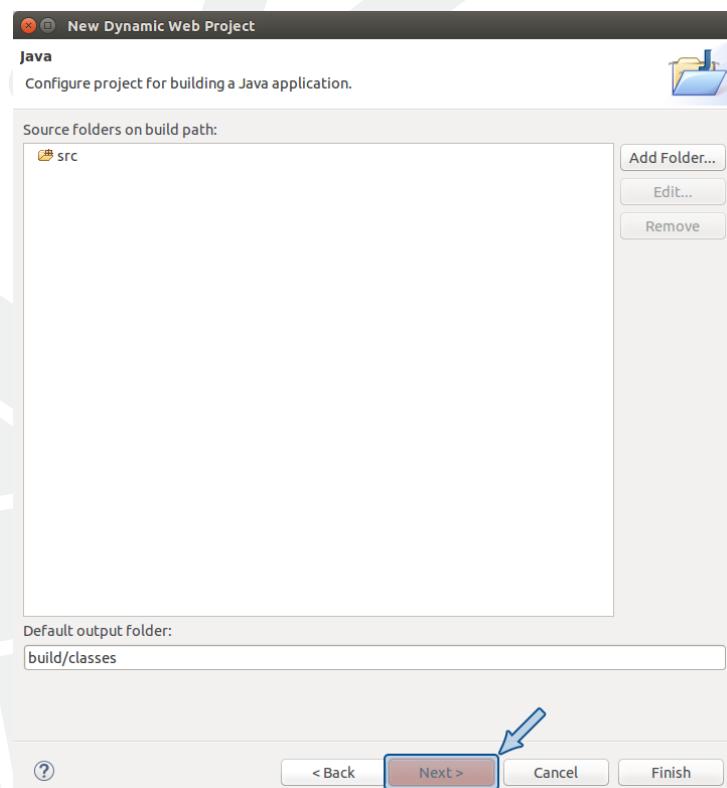
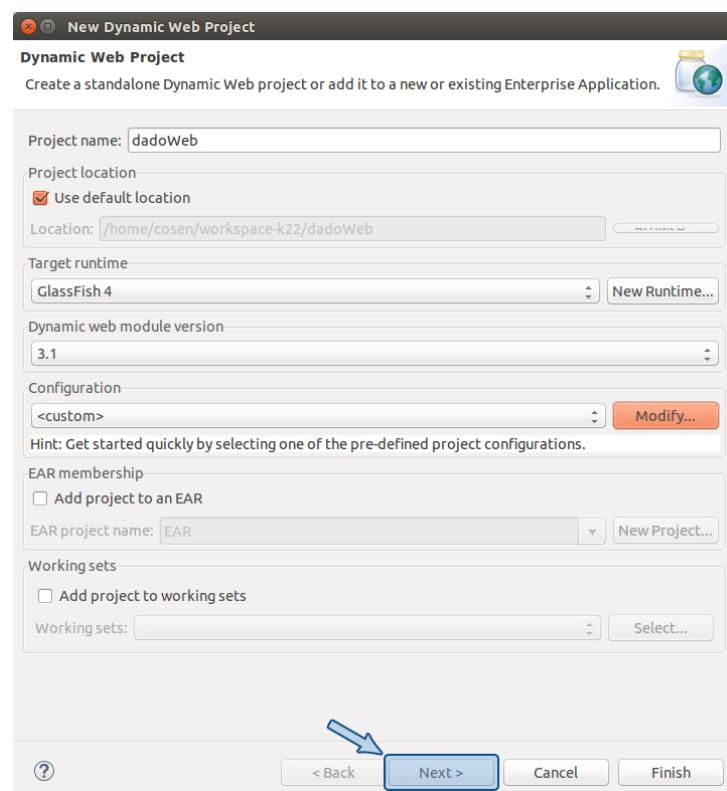
Exercícios de Fixação

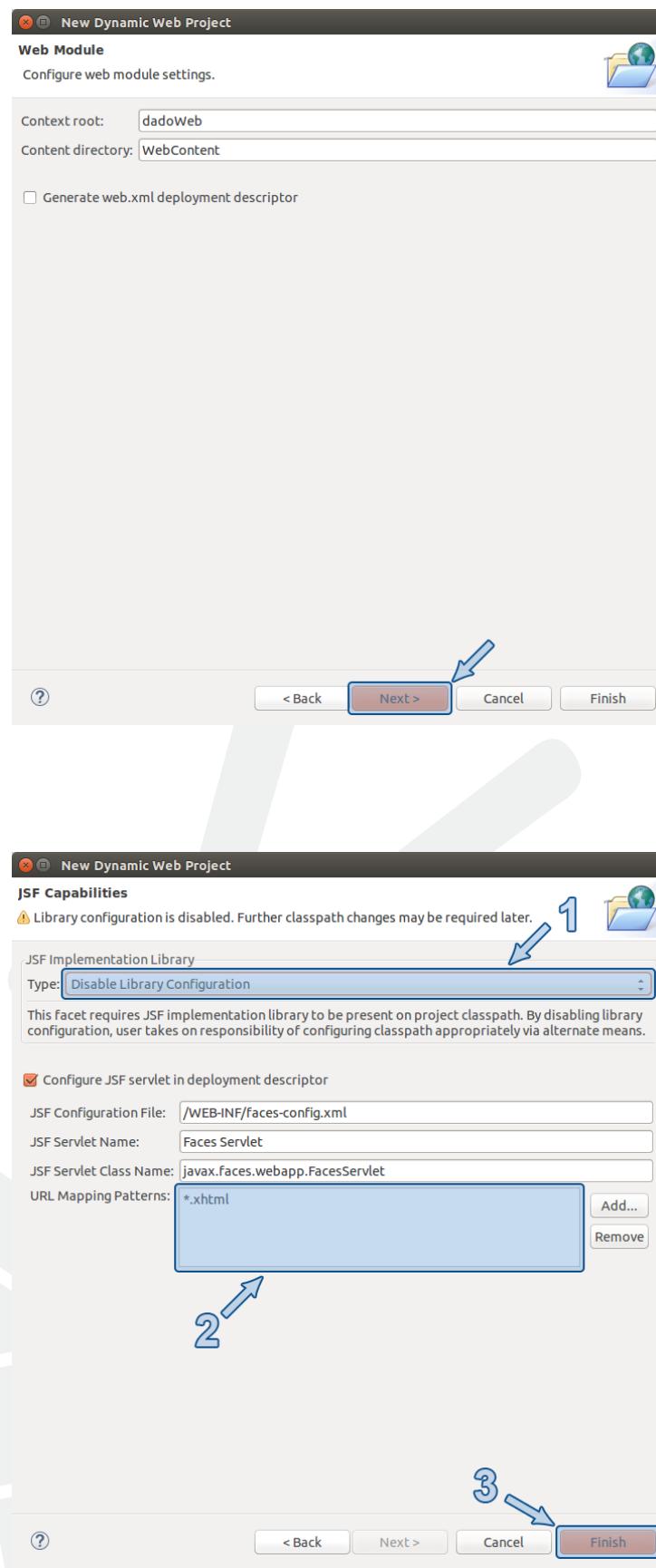
- 10 Para não confundir, feche os projetos **calculadora**, **calculadoraClient**, **calculadoraEAR** e **calculadoraWeb**. Para isso, clique com o botão direito do mouse sobre esses projetos e selecione a opção “Close Project”.

- 11 Crie um Dynamic Web Project no eclipse para implementar a camada web. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens

abaixo.







- 12 No projeto **dadoWeb**, adicione um pacote chamado **br.com.k19.sessionbeans** e acrescente nele uma classe chamada **LancadorDeDadoBean**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.Random;
4
5 import javax.ejb.Stateless;
6
7 @Stateless
8 public class LancadorDeDadoBean {
9     private Random gerador = new Random();
10
11     public int lanca() {
12         return this.gerador.nextInt(6) + 1;
13     }
14 }
```

Código Java 2.17: LancadorDeDadoBean.java

- 13 No projeto **dadoWeb**, adicione um pacote chamado **br.com.k19.managedbeans** e acrescente nele uma classe chamada **DadoMB**.

```

1 package br.com.k19.managedbeans;
2
3 import javax.ejb.EJB;
4 import javax.faces.bean.ManagedBean;
5
6 import br.com.k19.sessionbeans.LancadorDeDadoBean;
7
8 @ManagedBean
9 public class DadoMB {
10     @EJB
11     private LancadorDeDadoBean lancadorDeDadoBean;
12
13     private int resultado;
14
15     public void lancaDado(){
16         this.resultado = this.lancadorDeDadoBean.lanca();
17     }
18
19     public int getResultado() {
20         return resultado;
21     }
22
23     public void setResultado(int resultado) {
24         this.resultado = resultado;
25     }
26 }
```

Código Java 2.18: DadoMB.java

- 14 Crie uma simples tela na aplicação web para utilizar o managed bean. Adicione o arquivo **dado.xhtml** na pasta **WebContent** do projeto **dadoWeb** com o seguinte conteúdo.

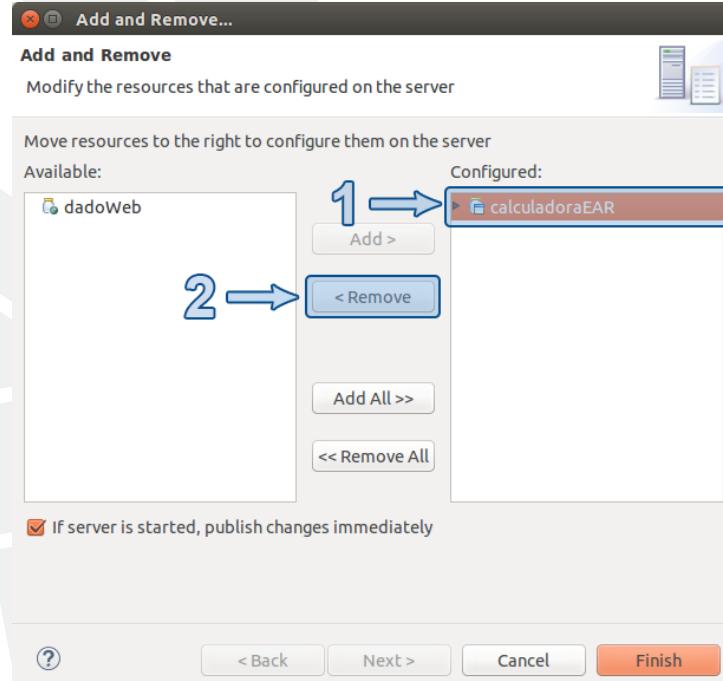
```

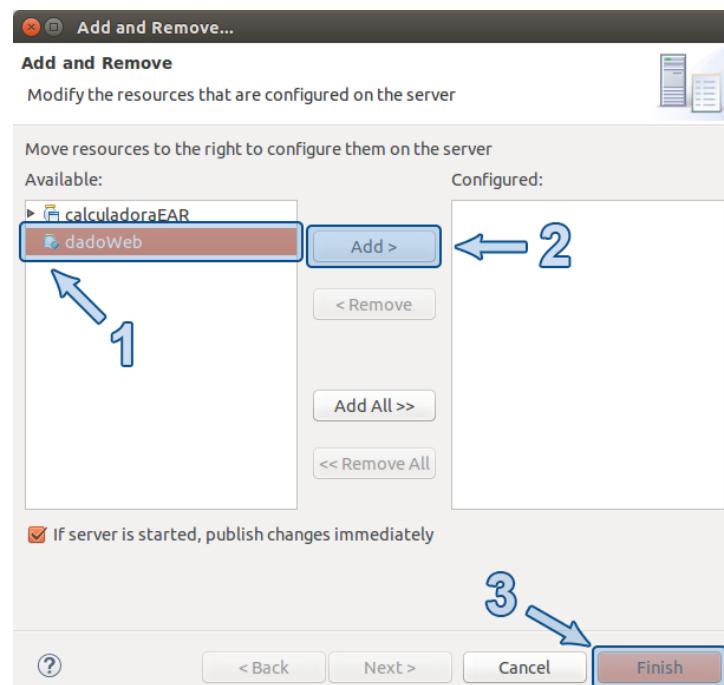
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:h="http://java.sun.com/jsf/html">
6
7 <h:head>
```

```
8 <title>Lançador de dado</title>
9 </h:head>
10
11 <h:body>
12   <h:form>
13     <h:commandButton action="#{dadoMB.lancaDado}" value="Lança o Dado" />
14
15     <h:outputLabel value="Resultado: " />
16     <h:outputText value="#{dadoMB.resultado}" />
17   </h:form>
18 </h:body>
19 </html>
```

Código XHTML 2.2: *dado.xhtml*

- 15 Remova o projeto **calculadoraEAR** e adicione o projeto **dadoWeb** no **Glassfish**. Clique com o botão direito no **Glassfish** da view Servers e escolha a opção “Add and Remove”. Depois, siga as imagens abaixo.





- 16 Certifique-se que o **Glassfish** está executando. Acesse a url <http://localhost:8080/dadoWeb/dado.xhtml> e teste o funcionamento da aplicação.



Cliente Java SE Remoto

Vimos os SLSBs sendo acessados localmente por aplicações web implantadas no mesmo servidor de aplicação. Contudo, eles podem ser acessados remotamente, ou seja, podem ser acessados por aplicações fora do mesmo servidor de aplicação. Inclusive, um SLSB pode ser acessado por aplicações Java SE.

Quando o acesso é local, podemos injetar um SLSB através da anotação @EJB no componente que necessita dos serviços implementados pelo SLSB. Agora, quando o acesso é remoto, não temos o recurso de injeção de dependência. Dessa forma, os SLSBs devem ser obtidos de outra maneira.

Todo SLSB implantado em um servidor de aplicação recebe um “nome”. Toda aplicação fora desse servidor de aplicação pode utilizar esse nome para obter a referência remota do SLSB.

Antes da versão 3.1, os nomes dos Session Beans não eram padronizados. Consequentemente, cada servidor de aplicação possuía uma regra diferente para nomear os Session Beans. A partir da versão 3.1, os nomes foram padronizados e portanto são portáveis (iguais em todos os servidores de aplicação). Consulte a especificação para conhecer as regras de nomenclatura <http://jcp.org/en/jsr/detail?id=318>.

Uma aplicação Java remota deve acessar o serviço de nomes (JNDI) do servidor de aplicação no qual o SLSB que ela deseja utilizar está implantado. O trecho de código Java para fazer uma consulta por um SLSB no JNDI teria o seguinte padrão.

```

1 InitialContext ic = new InitialContext();
2 StatelessSessionBean statelessSessionBean =
3     (StatelessSessionBean) ic.lookup("java:global/aplicacaoWeb/StatelessSessionBean");

```

Uma vez com a referência do SLSB, a aplicação pode chamar as operações normalmente como se o Session Bean estivesse local. Contudo, é importante ressaltar que as chamadas são remotas e portanto mais demoradas.



Exercícios de Fixação

- 17** Adicione uma interface chamada **LancadorDeDado** no pacote **br.com.k19.sessionbeans** do projeto **dadoWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 public interface LancadorDeDado {
4     int lanca();
5 }

```

Código Java 2.20: LancadorDeDado.java

- 18** Altere a classe **LancadorDeDadoBean** do projeto **dadoWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.Random;
4
5 import javax.ejb.Remote;
6 import javax.ejb.Stateless;
7
8 @Stateless
9 @Remote(LancadorDeDado.class)
10 public class LancadorDeDadoBean implements LancadorDeDado {
11     private Random gerador = new Random();
12
13     public int lanca(){
14         return this.gerador.nextInt(6) + 1;
15     }
16 }

```

Código Java 2.21: LancadorDeDadoBean.java

- 19** Altere a classe **DadoMB** do projeto **dadoWeb**.

```

1 package br.com.k19.managedbeans;
2
3 import javax.ejb.EJB;
4 import javax.faces.bean.ManagedBean;
5
6 import br.com.k19.sessionbeans.LancadorDeDado;
7
8 @ManagedBean
9 public class DadoMB {
10     @EJB
11     private LancadorDeDado lancadorDeDadoBean;
12 }

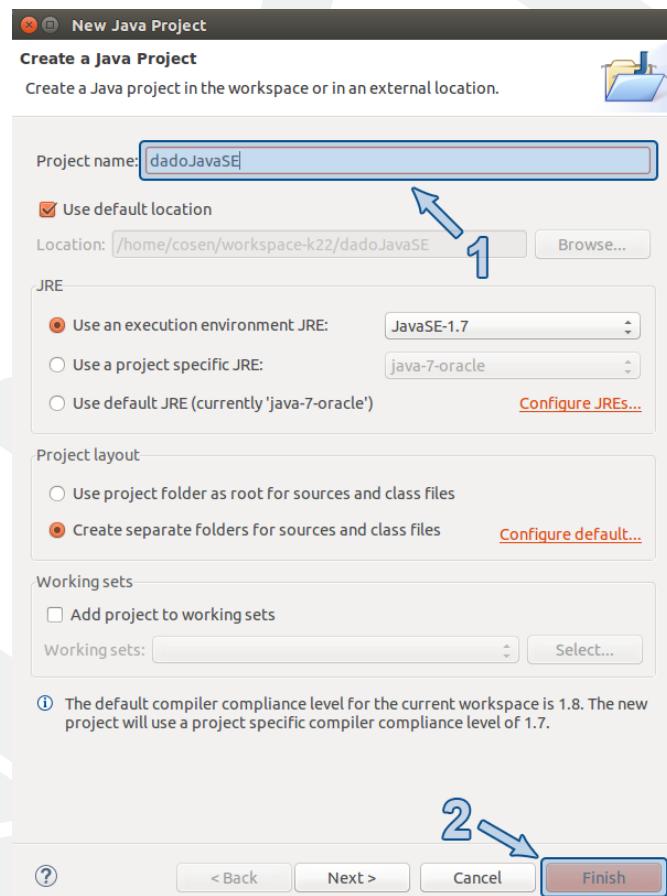
```

```

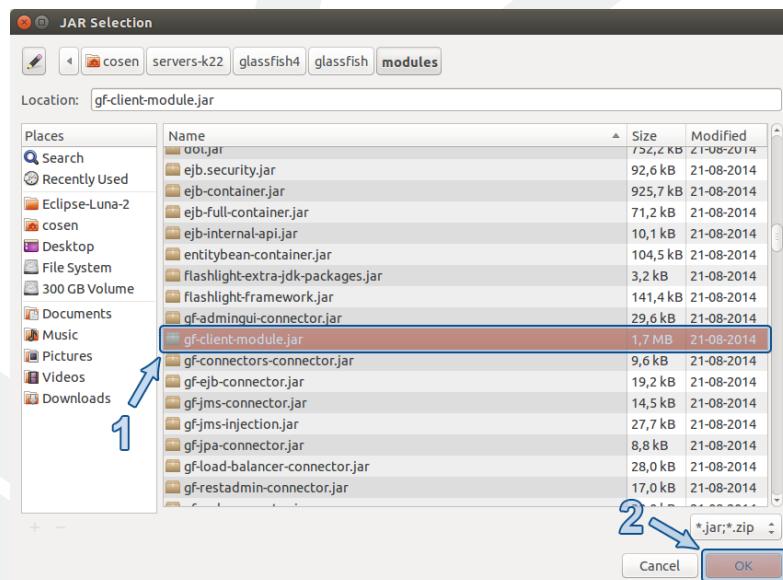
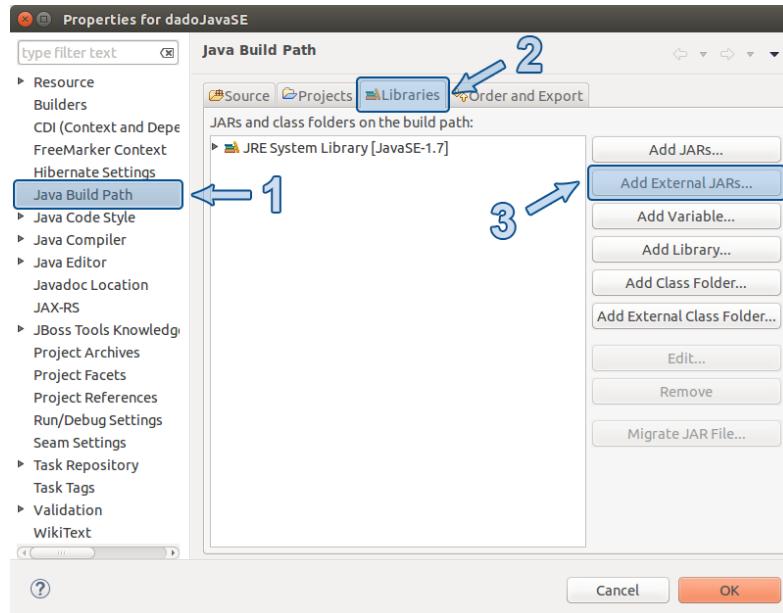
13 private int resultado;
14
15 public void lancaDados(){
16     this.resultado = this.lancadorDeDadosBean.lanca();
17 }
18
19 public int getResultado() {
20     return resultado;
21 }
22
23 public void setResultado(int resultado) {
24     this.resultado = resultado;
25 }
26 }
```

Código Java 2.22: DadoMB.java

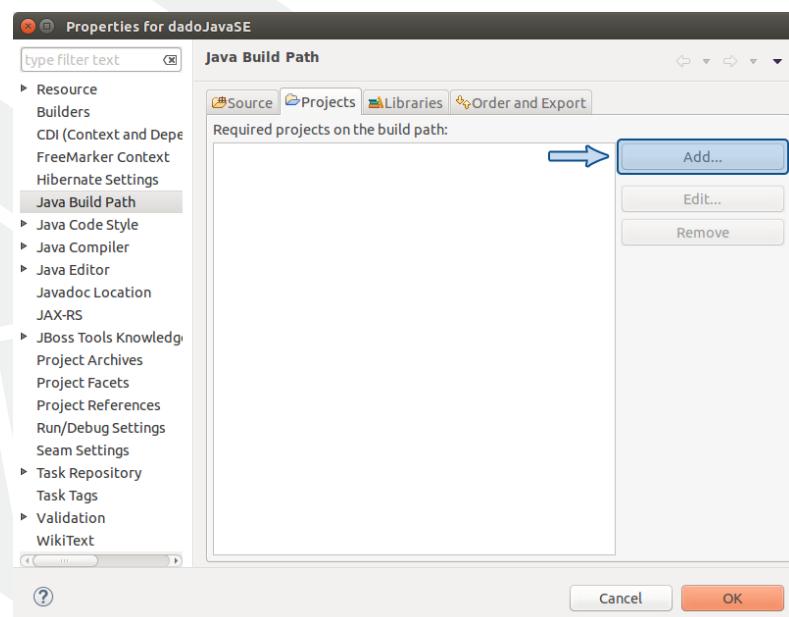
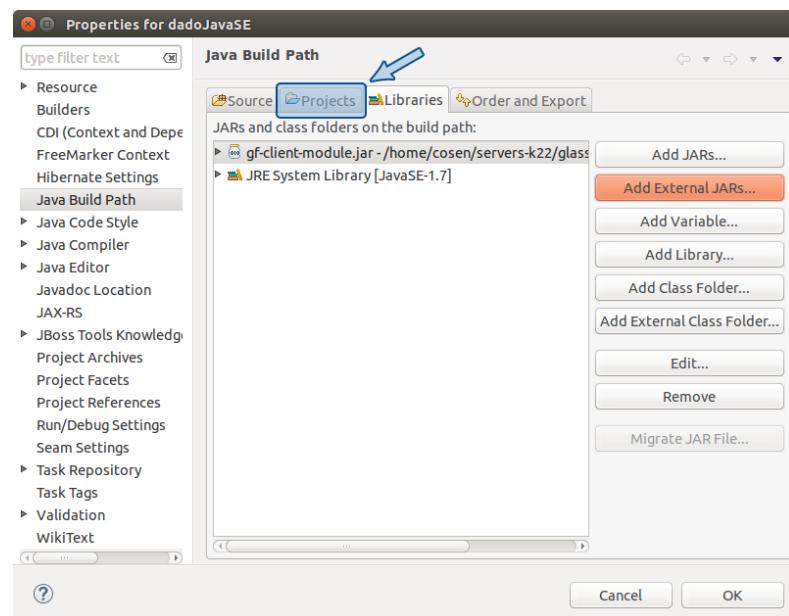
- 20** Crie um Java project no eclipse. Você pode digitar “CTRL+3” em seguida “new Java project” e “ENTER”. Depois, siga exatamente as imagens abaixo.

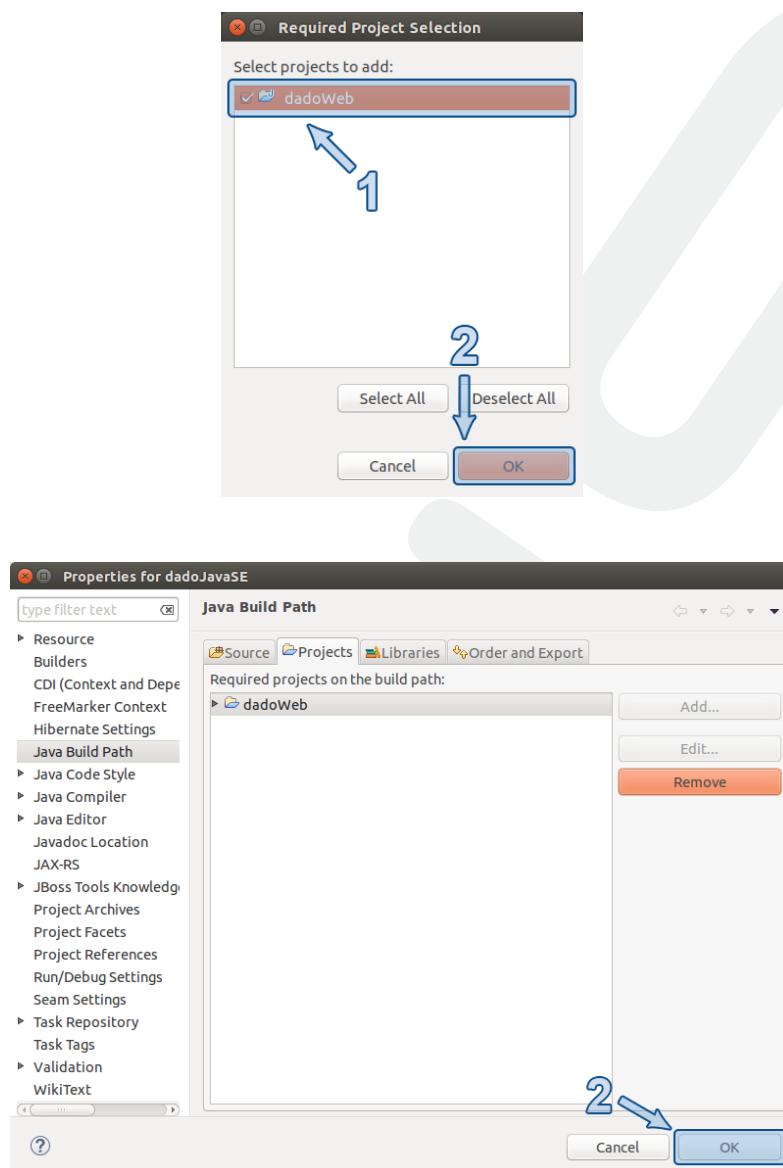


- 21** Adicione as bibliotecas do **Glassfish** necessárias para a consulta ao serviço de nomes. Abra as propriedades do projeto **dadoJavaSE**. Você pode selecionar o projeto **dadoJavaSE** e digitar “ALT+ENTER”. Depois, siga as imagens abaixo.



22 Adicione o projeto **dadoWeb** como dependência do projeto **dadoJavaSE**. Abra as propriedades do projeto **dadoJavaSE**. Você pode selecionar o projeto **dadoJavaSE** e digitar “ALT+ENTER”. Depois, siga as imagens abaixo





- 23 No projeto **dadoJavaSE**, adicione um pacote chamado **br.com.k19.testes** acrescenta nele uma classe chamada **TesteDeAcesso**.

```

1 package br.com.k19.testes;
2
3 import javax.naming.InitialContext;
4
5 import br.com.k19.sessionbeans.LancadorDeDado;
6
7 public class TesteDeAcesso {
8     public static void main(String[] args) throws Exception {
9         InitialContext ic = new InitialContext();
10
11         LancadorDeDado lancadorDeDado = (LancadorDeDado) ic
12             .lookup("java:global/dadoWeb/LancadorDeDadoBean");
13         System.out.println(lancadorDeDado.lanca());
14     }
15 }
```

Código Java 2.23: *TesteDeAcesso.java*

24 Republique o projeto **dadoWeb**. Você pode clicar com o botão direito nesse projeto na view Servers dentro do Glassfish e selecionar a opção “Full publish”.

25 Execute a classe **TesteDeAcesso** e confira o resultado no console



Ciclo de Vida

As instâncias dos SLSBs são administradas pelo EJB Container. Devemos entender o ciclo de vida desses objetos para utilizar corretamente a tecnologia EJB. Três aspectos fundamentais dos SLSBs nos ajudam a entender o ciclo de vida das instâncias.

1. Uma única instância de um SLSB pode atender chamadas de diversos clientes.
2. Uma instância de um SLSB não atende duas chamadas ao mesmo tempo. Em outras palavras, ela processa uma chamada de cada vez.
3. O EJB Container pode criar várias instâncias do mesmo SLSB para atender mais rapidamente as chamadas dos clientes.

Estados

O ciclo de vida das instâncias de um SLSB possui apenas dois estados.

1. NÃO EXISTE
2. PRONTO

NÃO EXISTE -> PRONTO

Antes de ser criada, dizemos que uma instância de um SLSB se encontra no estado NÃO EXISTE. Obviamente, nesse estado, uma instância não pode atender chamadas dos clientes.

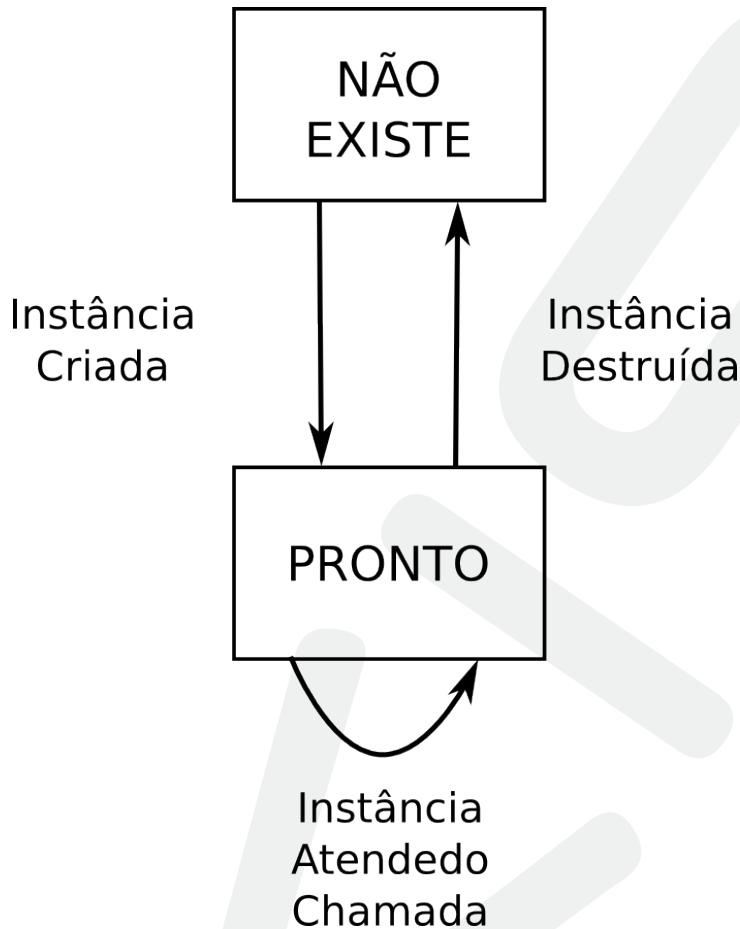
De acordo com a quantidade de chamadas e critérios de cada servidor de aplicação, o EJB Container pode criar novas instâncias de um SLSB. Cada instância criada passa para o estado PRONTO. No estado PRONTO, uma instância está apta a receber uma chamada.

PRONTO -> PRONTO

Quando uma chamada é realizada, o EJB Container seleciona uma instância entre as que estejam no estado PRONTO para realizar o atendimento. Enquanto, uma instância está atendendo uma chamada ela não pode atender outras chamadas. Depois de finalizar o atendimento, a instância volta para o estado PRONTO podendo receber outra chamada.

PRONTO -> NÃO EXISTE

Novamente, de acordo com a quantidade de chamadas e critérios de cada servidor de aplicação, o EJB Container pode destruir instâncias que estejam no estado PRONTO. Essas instâncias voltam para o estado NÃO EXISTE.



Escalabilidade e Pool

As características dos SLSBs favorecem a escalabilidade da aplicação pois, de acordo com a demanda, o EJB Container cria novas instâncias e cada instância pode atender vários clientes.

O EJB Container administra as instâncias criadas através de um Pool. Cada servidor de aplicação oferece configurações específicas para melhorar a eficiência no atendimento das chamadas. Por exemplo, o Glassfish permite que uma quantidade máxima de instâncias de um determinado SLSB seja definida pela aplicação.



Callbacks

Podemos associar lógicas específicas nas transições de estado no ciclo de vida dos SLSBs.

@PostConstruct

Podemos registrar um método de instância no EJB Container para que ele o execute em cada instância logo após ela ser criada. Esse registro é realizado através da anotação **@PostConstruct**.

1 @Stateless

```

2 public class CalculadoraBean {
3
4     @PostConstruct
5     public void inicializando() {
6         System.out.println("Mais uma calculadora criada...")
7     }
8
9     // METODOS DE NEGOCIO
10 }
```

Código Java 2.24: CalculadoraBean.java

O EJB Container utiliza o construtor sem argumentos para criar uma instância de um SLSB. Depois de chamar o construtor sem argumentos, o EJB Container injeta eventuais dependências na instância criada. Por fim, os métodos anotados com `@PostConstruct` são executados.

@PreDestroy

Também podemos registrar um método de instância no EJB Container para que ele o execute em cada instância imediatamente antes dela ser destruída. Esse registro é realizado através da anotação `@PreDestroy`.

```

1 @Stateless
2 public class CalculadoraBean {
3
4     @PreDestroy
5     public void destruindo() {
6         System.out.println("Mais uma calculadora será destruída...")
7     }
8
9     // METODOS DE NEGOCIO
10 }
```

Código Java 2.25: CalculadoraBean.java

Exercícios de Fixação

- 26 Adicione métodos de callbacks na classe **LancadorDeDadoBean** do projeto **dadoWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.Random;
4
5 import javax.annotation.PostConstruct;
6 import javax.annotation.PreDestroy;
7 import javax.ejb.Remote;
8 import javax.ejb.Stateless;
9
10 @Stateless
11 @Remote(LancadorDeDado.class)
12 public class LancadorDeDadoBean implements LancadorDeDado {
13     private Random gerador = new Random();
14     private static int contador;
15
16     @PostConstruct
17     public void inicializando(){
18         synchronized (LancadorDeDadoBean.class) {
19             LancadorDeDadoBean.contador++;
20             System.out.println("Criando um lançador de dados...");
```

```

21     System.out.println("Total: " + LancadorDeDadoBean.contador);
22 }
23 }
24
25 @PreDestroy
26 public void destruindo() {
27     synchronized (LancadorDeDadoBean.class) {
28         LancadorDeDadoBean.contador--;
29         System.out.println("Destruindo um lançador de dados...");
30         System.out.println("Total: " + LancadorDeDadoBean.contador);
31     }
32 }
33
34 public int lanca(){
35     return this.gerador.nextInt(6) + 1;
36 }
37 }
```

Código Java 2.26: LancadorDeDadoBean.java

- 27** Adicione um teste no pacote **br.com.k19.testes** do projeto **dadoJavaSE** para fazer consultas em paralelo ao SLSB que lança moedas.

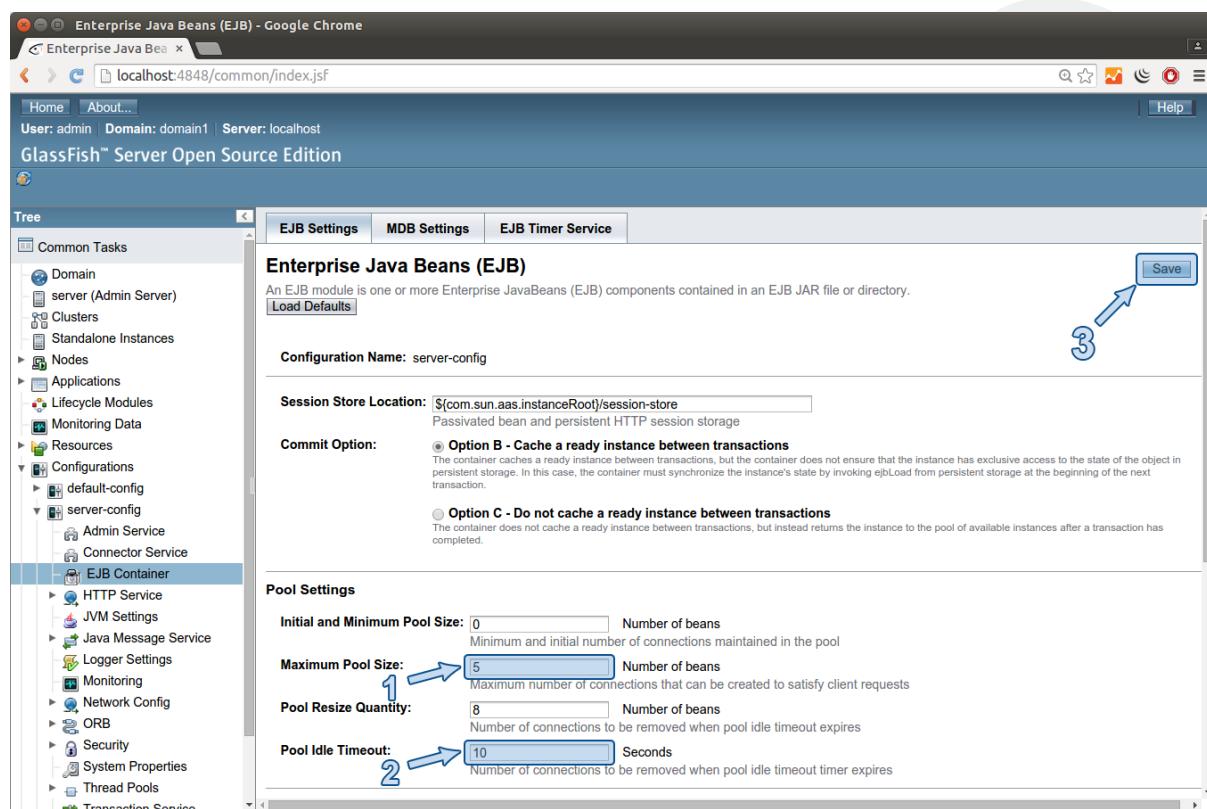
```

1 package br.com.k19.testes;
2
3 import javax.naming.InitialContext;
4
5 import br.com.k19.sessionbeans.LancadorDeDado;
6
7 public class TesteCicloDeVidaSLSB {
8     public static void main(String[] args) throws Exception {
9         InitialContext ic = new InitialContext();
10
11        for (int i = 0; i < 100; i++) {
12            final LancadorDeDado lancadorDeDado = (LancadorDeDado) ic
13                .lookup("java:global/dadoWeb/LancadorDeDadoBean");
14
15            Thread thread = new Thread(new Runnable() {
16
17                @Override
18                public void run() {
19                    for (int i = 0; i < 100; i++) {
20                        System.out.println(lancadorDeDado.lanca());
21                    }
22                }
23            });
24            thread.start();
25        }
26    }
27 }
```

Código Java 2.27: TesteCicloDeVidaSLSB.java

- 28** Reinicie o **Glassfish**. Depois, execute a classe **TesteCicloDeVidaSLSB** observe o log do servidor para conferir as mensagens dos métodos de callback. Chame o instrutor caso não consiga visualizar o log do glassfish.

- 29** Altere as configurações do pool de SLSB do Glassfish. Acesse a interface de administração do Glassfish no endereço <http://localhost:4848/>. Faça as modificações indicadas na imagem abaixo.



- 30** Reinicie o **Glassfish**. Depois, execute a classe `TesteCicloDeVidaSLSB` observe o log do servidor para conferir as mensagens dos métodos de callback. Observe que o servidor tentará manter no Pool de SLSB com apenas 5 instâncias do `LancadorDeDadoBean`. Observe também que depois de 10 segundos de ociosidade as instâncias são destruídas. Chame o instrutor caso não consiga visualizar o log do glassfish.



Métodos Assíncronos

A partir da versão 3.1 do EJB, podemos definir métodos assíncronos nos session beans. Geralmente, métodos assíncronos são utilizados para implementar tarefas demoradas. Para definir um método assíncrono, basta utilizar a anotação `@Asynchronous`. Se essa anotação for aplicada na classe que define um session bean, todos os métodos de negócios desse session bean serão assíncronos. Por outro lado, podemos aplicar essa anotação somente nos métodos de negócios que devem ser assíncronos.

```

1 @Stateless
2 @Asynchronous
3 public class MeuSessionBean {
4
5     public Future<Integer> metodoAssincrono1() { . . . }
6
7     public Future<String> metodoAssincrono2() { . . . }
8 }
```

Código Java 2.28: `MeuSessionBean.java`

```

1 @Stateless
2 public class MeuSessionBean {
3
4     @Asynchronous
5     public Future<Integer> metodoAssincrono1() {...}
6
7     @Asynchronous
8     public Future<String> metodoAssincrono2() {...}
9
10    public String metodoSincrono1() {...}
11
12    public Integer metodoSincrono2() {...}
13 }

```

Código Java 2.29: MeuSessionBean.java

Observe que o retorno dos métodos assíncronos utiliza a interface **java.util.concurrent.Future**. Essa interface permite verificar se a tarefa já foi concluída através do método **isDone()**.

```

1 MeuSessionBean bean = ...
2 Future<String> future = bean.metodoAssincrono2();
3 // executa alguma coisa enquanto o session bean trabalha
4
5 // verifica se a tarefa terminou
6 if(future.isDone()){
7
8 }

```

A interface Future também permite que o resultado seja recuperado através do método **get()**.

```

1 MeuSessionBean bean = ...
2 Future<String> future = bean.metodoAssincrono2();
3 // executa alguma coisa enquanto o session bean trabalha
4
5 // verifica se a tarefa terminou
6 if(future.isDone()){
7     String resultado = future.get();
8 }

```



Exercícios de Fixação

31 Acrescente o método **calculaFrequencia()** na classe **LancadorDeDadoBean** do projeto **dadoWeb**.

Obs: Faça o importe dos seguintes itens:

- `java.util.Map`
- `java.util.HashMap`
- `java.util.concurrent.Future`

```

1 ...
2 @Asynchronous
3 public Future<Map<Integer, Integer>> calculaFrequencia() {
4     Map<Integer, Integer> map = new HashMap<Integer, Integer>();
5     map.put(1, 0);
6     map.put(2, 0);
7     map.put(3, 0);
8     map.put(4, 0);

```

```

9  map.put(5, 0);
10 map.put(6, 0);
11
12 for (int i = 0; i < 500; i++) {
13     int v = this.gerador.nextInt(6) + 1;
14     map.put(v, map.get(v) + 1);
15     try {
16         Thread.sleep(100);
17     } catch (InterruptedException e) {
18     }
19     System.out.println(i);
20 }
21 return new AsyncResult<Map<Integer, Integer>>(map);
22 }
23 ...

```

Código Java 2.32: LancadorDeDadoBean.java

32 Adicione o método **calculaFrequencia()** na interface **LancadorDeDado** do projeto **dadoWeb**.

Obs: Faça o importe dos seguintes itens:

- `java.util.Map`
- `java.util.concurrent.Future`

```

1 ...
2 Future<Map<Integer, Integer>> calculaFrequencia();
3 ...

```

Código Java 2.33: LancadorDeDado.java

33 Adicione uma classe chamada **TesteAsynchronous** no projeto **dadoJavaSE**.

```

1 package br.com.k19.testes;
2
3 import java.util.Map;
4 import java.util.concurrent.Future;
5
6 import javax.naming.InitialContext;
7
8 import br.com.k19.sessionbeans.LancadorDeDado;
9
10 public class TesteAsynchronous {
11     public static void main(String[] args) throws Exception {
12         InitialContext ic = new InitialContext();
13
14         LancadorDeDado lancadorDeDado = (LancadorDeDado) ic
15             .lookup("java:global/dadoWeb/LancadorDeDadoBean");
16         Future<Map<Integer, Integer>> future = lancadorDeDado.calculaFrequencia();
17
18         System.out.print("Aguardando ");
19
20         while(!future.isDone()){
21             System.out.print("*");
22             Thread.sleep(1000);
23         }
24         System.out.println("\n" + future.get());
25     }
26 }

```

Código Java 2.34: TesteAsynchronous.java

34 Republique o projeto **dadoWeb**. Você pode clicar com o botão direito nesse projeto na view Servers dentro do Glassfish e selecionar a opção “Full publish”.

35 Execute a classe TesteAsynchronous e verifique o console do cliente e do servidor.



STATEFUL SESSION BEANS



Caracterizando os SFSBs

Stateful Session Bean é o segundo tipo de Session Bean. Muitas vezes, utilizaremos a sigla SFSB para fazer referência a esse tipo de componente. A ideia fundamental por trás dos SFSBs é a necessidade de manter estado conversacional.

Carrinho de Compras

Para exemplificar, suponha o funcionamento de um carrinho de compras de uma loja virtual. As regras de negócio do carrinho podem ser implementadas através de alguns métodos.

```
1 class CarrinhoBean {  
2       
3     public void adiciona(Produto produto) {  
4         // lógica para adicionar produto  
5     }  
6       
7     public void remove(Produto produto) {  
8         // lógica para remover produto  
9     }  
10      
11    public void finalizaCompra(){  
12        // lógica para finalizar a compra  
13    }  
14 }
```

Código Java 3.1: CarrinhoBean.java

Há duas necessidades fundamentais no exemplo do carrinho de compras que devemos observar. Primeiro, uma instância da classe CarrinhoBean não deve atender vários clientes para não misturar produtos escolhidos por clientes diferentes. Segundo, os produtos adicionados devem ser mantidos entre as chamadas dos métodos da classe CarrinhoBean. Em outras palavras, é necessário manter estado conversacional.

Provavelmente, o estado do carrinho, ou seja, os produtos adicionados seria mantido em uma lista ou em um conjunto.

```
1 class CarrinhoBean {  
2       
3     private Set<Produto> produtos = new HashSet<Produto>();  
4       
5     public void adiciona(Produto produto) {  
6         this.produtos.add(produto);  
7     }  
8       
9     public void remove(Produto produto) {  
10        this.produtos.remove(produto);  
11    }  
12 }
```

```

12
13     public void finalizaCompra(){
14         // lógica para finalizar a compra
15     }
16 }
```

Código Java 3.2: CarrinhoBean.java

Prova Digital

Outro exemplo, suponha o funcionamento de um sistema para aplicar provas que permita que os usuários respondam as questões em qualquer ordem ou modifiquem respostas já realizadas antes de finalizar a prova. As respostas poderiam ser mantidas em um mapa.

```

1 class ProvaBean {
2     private Map<Integer, Character> respostas = new HashMap<Integer, Character>();
3
4     public void responde(Integer questao, Character resposta) {
5         this.respostas.put(questao, resposta);
6     }
7
8     public void finaliza(){
9         // lógica para finalizar a prova
10    }
11 }
```

Código Java 3.3: ProvaBean.java

Uma instância da classe ProvaBean não pode atender dois clientes para não misturar as respostas de dois usuários diferentes. Além disso, as respostas já realizadas devem ser mantidas entre as chamadas.

TrackList

Mais um exemplo, suponha o funcionamento de um player de vídeo que permite que os usuários selecionem um conjunto de vídeos para assistir.

```

1 class ListaDeVideos {
2     private List<Video> videos = new ArrayList<Video>();
3
4     public void adiciona(Video video) {
5         this.videos.add(video);
6     }
7
8     public void embaralha() {
9         Collections.shuffle(this.videos);
10    }
11 }
```

Código Java 3.4: ListaDeVideos.java

Novamente, cada instância da classe ListaDeVideos deve ser exclusiva de um cliente e os vídeos adicionados devem ser mantidos entre as chamadas dos métodos.



SFSB - EJB 3.0

O primeiro passo para implementar um SFSB é definir a sua interface de utilização através de uma interface java. Por exemplo, considere um SFSB que implemente o funcionamento do carrinho de compras. Uma possível interface de utilização para esse session bean seria:

```

1 public interface Carrinho {
2     void adiciona(Produto produto);
3     void remove(Produto produto);
4 }
```

Código Java 3.5: Carrinho.java

Após definir a interface de utilização, o segundo passo seria implementar as operações do SFSB através de uma classe java.

```

1 public class CarrinhoBean implements Carrinho {
2
3     private Set<Produto> produtos = new HashSet<Produto>();
4
5     public void adiciona(Produto produto) {
6         this.produtos.add(produto);
7     }
8
9     public void remove(Produto produto) {
10        this.produtos.remove(produto);
11    }
12 }
```

Código Java 3.6: CarrinhoBean.java

O terceiro passo é especificar o tipo de session bean que queremos utilizar. No caso do carrinho, o tipo seria SFSB. Essa definição é realizada através da anotação **@Stateful**.

```

1 @Stateful
2 public class CarrinhoBean implements Carrinho {
3     ...
4 }
```

Código Java 3.7: CarrinhoBean.java

Por fim, é necessário definir se o SFSB poderá ser acessado remotamente ou apenas localmente. Quando o acesso a um SLSB é local, ele só pode ser acessado por aplicações que estejam no mesmo servidor de aplicação que ele. Caso contrário, quando o acesso a um SLSB é remoto, ele pode ser acessado tanto por aplicações que estejam no mesmo servidor de aplicação quanto aplicações que não estejam.

A definição do tipo de acesso é realizada através das anotações: **@Local** e **@Remote**.

```

1 @Stateful
2 @Remote(Carrinho.class)
3 public class CarrinhoBean implements Carrinho {
4     ...
5 }
```

Código Java 3.8: CarrinhoBean.java

```

1 @Stateful
2 @Local(Carrinho.class)
3 public class CarrinhoBean implements Carrinho {
4     ...
5 }
```

Código Java 3.9: CarrinhoBean.java



SFSB - EJB 3.1

Na versão 3.1, quando o acesso a um SFSB é local, não é mais necessário definir uma interface java nem utilizar a anotação @Local. Então, bastaria implementar uma classe java com a anotação @Stateful.

```
1 @Stateful
2 public class CarrinhoBean {
3
4     private Set<Produto> produtos = new HashSet<Produto>();
5
6     public void adiciona(Produto produto) {
7         this.produtos.add(produto);
8     }
9
10    public void remove(Produto produto) {
11        this.produtos.remove(produto);
12    }
13 }
```

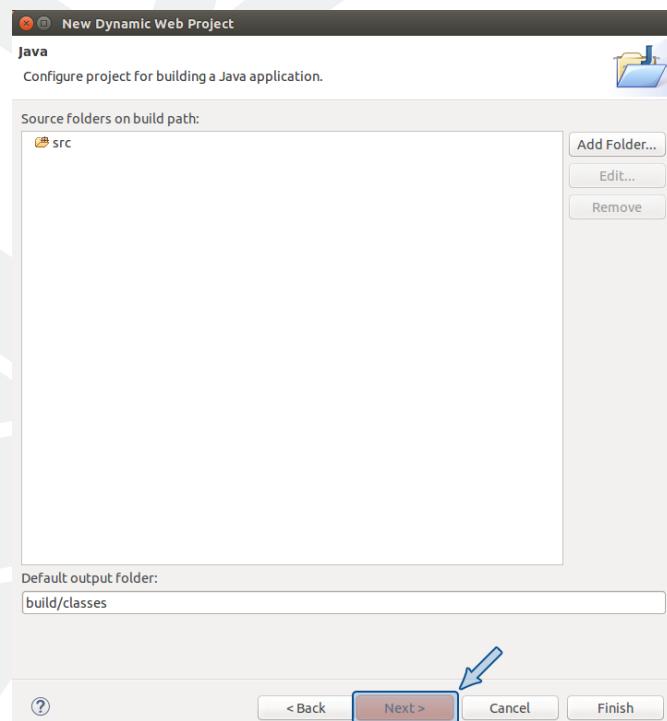
Código Java 3.10: CarrinhoBean.java

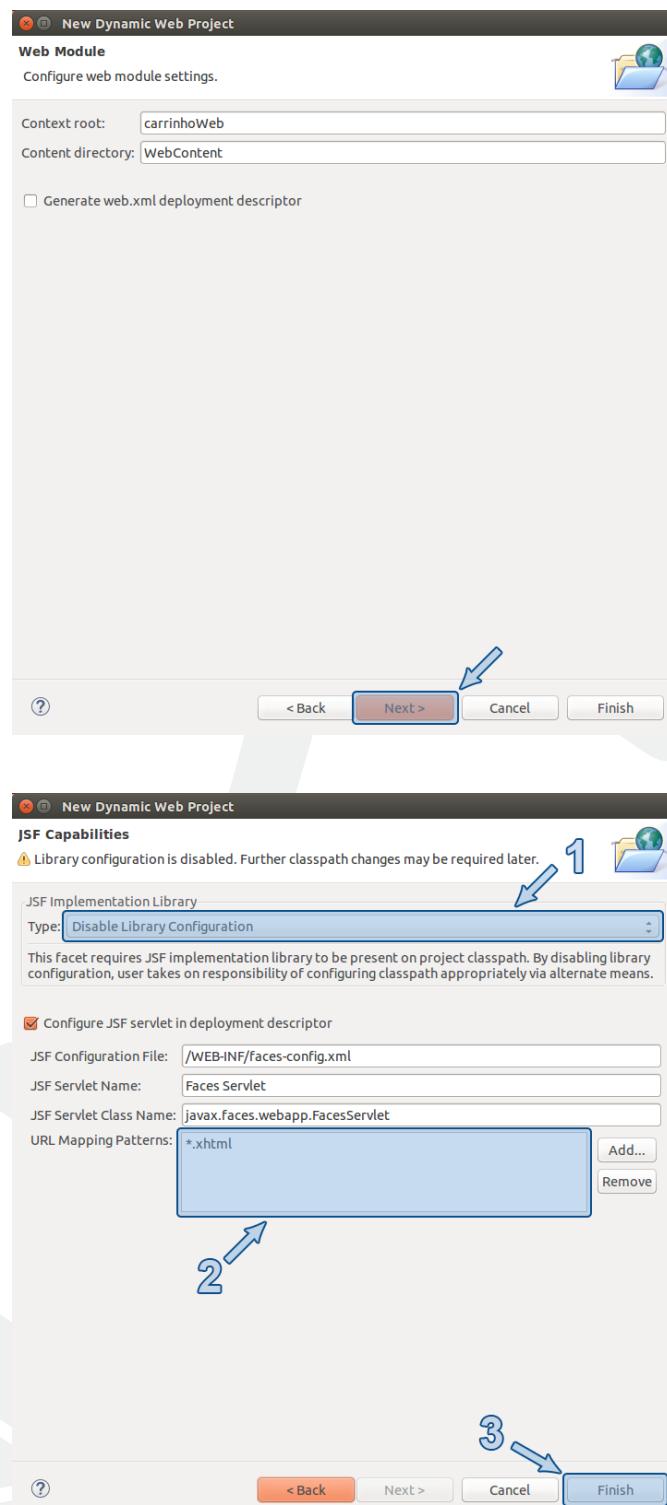


Exercícios de Fixação

- 1 Para não confundir, feche os projetos **dadoWeb** e **dadoJavaSE**. Para isso, clique com o botão direito do mouse sobre esses projetos e selecione a opção “Close Project”.

- 2 Crie um Dynamic Web Project no eclipse para implementar o funcionamento do carrinho de compras. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.





- 3 Crie um pacote chamado **br.com.k19.sessionbeans** e adicione a seguinte classe.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.HashSet;
4 import java.util.Set;
5 
```

```

6 import javax.ejb.Stateful;
7
8 @Stateful
9 public class CarrinhoBean {
10
11     private Set<String> produtos = new HashSet<String>();
12
13     public void adiciona(String produto) {
14         this.produtos.add(produto);
15     }
16
17     public void remove(String produto) {
18         this.produtos.remove(produto);
19     }
20
21     public Set<String> getProdutos() {
22         return produtos;
23     }
24 }
```

Código Java 3.11: CarrinhoBean.java

- 4 Crie um pacote chamado **br.com.k19.managedbeans** e adicione a seguinte classe.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.ejb.EJB;
7 import javax.faces.bean.ManagedBean;
8 import javax.faces.bean.SessionScoped;
9
10 import br.com.k19.sessionbeans.CarrinhoBean;
11
12 @ManagedBean
13 @SessionScoped
14 public class CarrinhoMB {
15     @EJB
16     private CarrinhoBean carrinhoBean;
17
18     private String produto;
19
20     public List<String> getProdutos() {
21         return new ArrayList<String>(this.carrinhoBean.getProdutos());
22     }
23
24     public void adiciona() {
25         this.carrinhoBean.adiciona(this.produto);
26     }
27
28     public void remove(String produto) {
29         this.carrinhoBean.remove(produto);
30     }
31
32     public void setProduto(String produto) {
33         this.produto = produto;
34     }
35
36     public String getProduto() {
37         return produto;
38     }
39 }
```

Código Java 3.12: CarrinhoMB.java

- 5 Adicione o arquivo **produtos.xhtml** na pasta **WebContent** do projeto **carrinhoWeb** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:h="http://java.sun.com/jsf/html">
6
7 <h:head>
8   <title>Carrinho de Compras</title>
9 </h:head>
10
11 <h:body>
12   <h:form>
13     <h:outputLabel value="Produto: "/>
14     <h:inputText value="#{carrinhoMB.produto}" />
15     <h:commandButton action="#{carrinhoMB.adiciona}" value="Adiciona no carrinho"/>
16
17     <hr/>
18
19     <h:outputLabel value="Produtos no carrinho: "/>
20     <h:dataTable value="#{carrinhoMB.produtos}" var="p">
21       <h:column>
22         <h:outputText value="#{p}" />
23       </h:column>
24       <h:column>
25         <h:commandLink action="#{carrinhoMB.remove(p)}" value="remove" />
26       </h:column>
27     </h:dataTable>
28   </h:form>
29 </h:body>
30 </html>
```

Código XHTML 3.1: produtos.xhtml

- 6 Remova o projeto **dadoWeb** do **Glassfish**. Adicione o projeto **carrinhoWeb** no **JBoss**. Certifique-se que o **Glassfish** e o **Wildfly** estejam parados. Execute o **JBoss** e teste a aplicação acessando a url <http://localhost:8080/carrinhoWeb/produtos.xhtml>.



Ciclo de Vida

As instâncias dos SFSBs são administradas pelo EJB Container. Devemos entender o ciclo de vida desses objetos para utilizar corretamente a tecnologia EJB. Para entender mais facilmente o ciclo de vida das instâncias dos SFSBs, devemos sempre ter em mente que cada instância atende apenas um cliente.

Estados

O ciclo de vida das instâncias de um SFSB possui três estados.

1. NÃO EXISTE
2. PRONTO
3. PASSIVADO

NÃO EXISTE -> PRONTO

Antes de ser criada, dizemos que uma instância de um SFSB se encontra no estado NÃO EXISTE. Obviamente, nesse estado, uma instância não pode atender as chamadas do seu cliente.

Quando um cliente recebe por injeção ou recupera por lookup um SFSB, o EJB Container cria uma nova instância desse SFSB para atender exclusivamente esse cliente. Nesse instante, logo após ser criada, a instância se encontra no estado PRONTO e pode atender as chamadas do seu respectivo cliente.

PRONTO -> PASSIVADO

Uma instância de um SFSB fica ociosa enquanto não estiver processando uma chamada do seu cliente. Para não consumir a memória do computador, depois de um certo tempo de ociosidade, o EJB Container pode transferir o conteúdo de uma instância ociosa para dispositivos secundários de armazenamento (hard disk). Esse processo de transferência é chamado de **passivação**. Após ser passivada, a instância ociosa se encontrará no estado PASSIVADO.

Outros fatores além da ociosidade podem levar o EJB Container decidir passivar instâncias dos SFSBs. Por exemplo, quando um certo limite de instâncias no estado PRONTO (ocupando memória) for atingido.

PASSIVADA -> PRONTO

Se o cliente de uma instância passivada realizar uma chamada a ela, o EJB Container realizará automaticamente o processo de ativação. Esse processo consiste na transferência do conteúdo da instância passivada para a memória principal novamente. Após ser ativada, a instância se encontrará no estado PRONTO e apta a atender a chamada realizada.

PRONTO -> NÃO EXISTE

Em determinadas situações, uma instância de um SFSB pode não ser mais útil. Em outras palavras, o cliente correspondente pode não precisar mais dela. Por exemplo, quando o cliente de um carrinho de compras finaliza a compra, a instância que representa o carrinho pode ser destruída (não compensa reaproveitar o mesmo carrinho para outro cliente).

O EJB Container é o responsável por destruir uma instância de um SFSB que não é mais útil. Por outro lado, a aplicação é responsável por determinar quando uma instância se torna inútil. Adicionando um método de negócio anotado com **@Remove** a aplicação declara que após a execução desse método a instância não é mais necessária.

No exemplo do carrinho de compras, poderíamos definir um método para implementar a lógica de finalizar compra e anotá-lo com **@Remove**. Dessa forma, logo após a execução desse método a instância seria destruída pelo EJB Container.

```

1 @Stateful
2 public class CarrinhoBean {
3
4     private Set<String> produtos = new HashSet<String>();
5
6     public void adiciona(String produto){
7         this.produtos.add(produto);
8     }
9
10    public void remove(String produto){

```

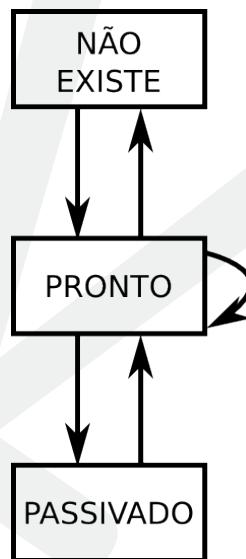
```

11     this.produtos.remove(produto);
12 }
13
14 public Set<String> getProdutos() {
15     return produtos;
16 }
17
18 @Remove
19 public void finalizaCompra(){
20     // lógica para finalizar compra
21 }
22 }
```

Código Java 3.13: CarrinhoBean.java

PASSIVADO -> PRONTO -> NÃO EXISTE

Uma instância pode ser passivada porque ficou ociosa quando estava no estado PRONTO. Ou seja, o respectivo cliente não realizou nenhuma chamada durante um “grande” período de tempo. Da mesma maneira, quando passivada, uma instância pode não receber uma chamada do seu cliente durante um “grande” período de tempo. Nesses casos, o EJB Container pode assumir que o cliente não chamará mais a instância passivada e portanto ela não é mais útil. Quando uma instância passivada não é mais útil, o EJB Container ativa e depois a destrói.



Callbacks

Podemos associar lógicas específicas nas transições de estado no ciclo de vida dos SFSBs.

@PostConstruct

Podemos registrar um método de instância no EJB Container para que ele o execute em cada instância logo após ela ser criada. Esse registro é realizado através da anotação **@PostConstruct**.

```

1 @Stateful
2 public class CarrinhoBean {
```

```

4  @PostConstruct
5  public void inicializando() {
6      System.out.println("Mais um carrinho criado...");
7  }
8
9  // METODOS DE NEGOCIO
10 }
```

Código Java 3.14: CarrinhoBean.java

O EJB Container utiliza o construtor sem argumentos para criar uma instância de um SLSB. Depois de chamar o construtor sem argumentos, o EJB Container injeta eventuais dependências na instância criada. Por fim, os métodos anotados com `@PostConstruct` são executados.

@PreDestroy

Também podemos registrar um método de instância no EJB Container para que ele o execute em cada instância imediatamente antes dela ser destruída. Esse registro é realizado através da anotação `@PreDestroy`.

```

1 @Stateful
2 public class CarrinhoBean {
3
4     @PreDestroy
5     public void destruindo() {
6         System.out.println("Mais um carrinho será destruído...");
7     }
8
9     // METODOS DE NEGOCIO
10 }
```

Código Java 3.15: CarrinhoBean.java

@PrePassivate

Também podemos registrar um método de instância no EJB Container para que ele o execute em cada instância imediatamente antes dela ser passivada. Esse registro é realizado através da anotação `@PrePassivate`.

```

1 @Stateful
2 public class CarrinhoBean {
3
4     @PrePassivate
5     public void passivando() {
6         System.out.println("Mais um carrinho será passivado...");
7     }
8
9     // METODOS DE NEGOCIO
10 }
```

Código Java 3.16: CarrinhoBean.java

@PostActivate

Também podemos registrar um método de instância no EJB Container para que ele o execute em cada instância imediatamente depois dela ser ativada. Esse registro é realizado através da anotação `@PostActivate`.

```
1 @Stateful
```

```

1 public class CarrinhoBean {
2
3     @PostActivate
4     public void ativando() {
5         System.out.println("Mais um carrinho foi ativado...");}
6
7 }
8
9 // METODOS DE NEGOCIO
10 }
```

Código Java 3.17: CarrinhoBean.java



Exercícios de Fixação

- 7 Adicione uma interface chamada **Carrinho** no pacote **br.com.k19.sessionbeans** do projeto **carrinhoWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.Set;
4
5 public interface Carrinho {
6
7     void adiciona(String produto);
8
9     void remove(String produto);
10
11    Set<String> getProdutos();
12
13    void finalizaCompra();
14 }
```

Código Java 3.18: Carrinho.java

- 8 Altere a classe **CarrinhoBean** do projeto **carrinhoWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.HashSet;
4 import java.util.Set;
5
6 import javax.annotation.PostConstruct;
7 import javax.annotation.PreDestroy;
8 import javax.ejb.PostActivate;
9 import javax.ejb.PrePassivate;
10 import javax.ejb.Remote;
11 import javax.ejb.Remove;
12 import javax.ejb.Stateful;
13
14 @Stateful
15 @Remote(Carrinho.class)
16 public class CarrinhoBean implements Carrinho {
17
18     private Set<String> produtos = new HashSet<String>();
19     private static int contadorTotal;
20     private static int contadorAtivos;
21     private int id;
22
23     public void adiciona(String produto) {
24         this.produtos.add(produto);
```

```

25 }
26
27 public void remove(String produto) {
28     this.produtos.remove(produto);
29 }
30
31 public Set<String> getProdutos() {
32     return produtos;
33 }
34
35 @Remove
36 public void finalizaCompra() {
37     System.out.println("Finalizando a compra");
38 }
39
40 @PostConstruct
41 public void postContract() {
42     synchronized (CarrinhoBean.class) {
43         CarrinhoBean.contadorTotal++;
44         CarrinhoBean.contadorAtivos++;
45         this.id = CarrinhoBean.contadorTotal;
46
47         System.out.println("PostConstruct");
48         System.out.println("ID: " + this.id);
49         System.out.println("ContatorTotal: " + CarrinhoBean.contadorTotal);
50         System.out.println("ContatorAtivos: " + CarrinhoBean.contadorAtivos);
51     }
52 }
53
54 @PrePassivate
55 public void prePassivate() {
56     synchronized (CarrinhoBean.class) {
57         CarrinhoBean.contadorAtivos--;
58
59         System.out.println("PrePassivate");
60         System.out.println("ID: " + this.id);
61         System.out.println("ContatorTotal: " + CarrinhoBean.contadorTotal);
62         System.out.println("ContatorAtivos: " + CarrinhoBean.contadorAtivos);
63     }
64 }
65
66 @PostActivate
67 public void postActivate() {
68     synchronized (CarrinhoBean.class) {
69         CarrinhoBean.contadorAtivos++;
70
71         System.out.println("PostActivate");
72         System.out.println("ID: " + this.id);
73         System.out.println("ContatorTotal: " + CarrinhoBean.contadorTotal);
74         System.out.println("ContatorAtivos: " + CarrinhoBean.contadorAtivos);
75     }
76 }
77
78 @PreDestroy
79 public void preDestroy() {
80     synchronized (CarrinhoBean.class) {
81         CarrinhoBean.contadorAtivos--;
82
83         System.out.println("PreDestroy");
84         System.out.println("ID: " + this.id);
85         System.out.println("ContatorTotal: " + CarrinhoBean.contadorTotal);
86         System.out.println("ContatorAtivos: " + CarrinhoBean.contadorAtivos);
87     }
88 }
89 }

```

Código Java 3.19: CarrinhoBean.java

- 9 Altere a classe **CarrinhoMB** do projeto **carrinhoWeb**.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.ejb.EJB;
7 import javax.faces.bean.ManagedBean;
8 import javax.faces.bean.SessionScoped;
9
10 import br.com.k19.sessionbeans.Carrinho;
11
12 @ManagedBean
13 @SessionScoped
14 public class CarrinhoMB {
15     @EJB
16     private Carrinho carrinhoBean;
17
18     private String produto;
19
20     ...
21 }
```

Código Java 3.20: CarrinhoMB.java

- 10 Republique o projeto **carrinhoWeb**. Você pode clicar com o botão direito nesse projeto na view Servers dentro do JBoss e selecionar a opção “Full publish”.

- 11 Adicione um usuário no JBoss com as seguintes características:

Tipo: Application User

Username: k19

Password: 1234

Utilize o script **add-user.sh** para adicionar o usuário **k19** no JBoss. Siga os passos abaixo.

```

cosen@k19:~/Desktop/jboss-as-7.1.1.Final/bin$ ./add-user.sh

What type of user do you wish to add?
  a) Management User (mgmt-users.properties)
  b) Application User (application-users.properties)
(a): b

Enter the details of the new user to add.
Realm (ApplicationRealm) :
Username : k19
Password :
Re-enter Password :
What roles do you want this user to belong to? (Please enter a comma separated list,
or leave blank for none) :
About to add user 'k19' for realm 'ApplicationRealm'
Is this correct yes/no? yes
Added user 'k19' to file '/home/cosen/Desktop/jboss-as-7.1.1.Final/standalone/
configuration/application-users.properties'
Added user 'k19' to file '/home/cosen/Desktop/jboss-as-7.1.1.Final/domain/
configuration/application-users.properties'
```

Terminal 3.1: Criando um usuário no JBoss

- 12** Configure o cache de SFSBs do JBoss. Modifique o arquivo **JBOSS_HOME/standalone/configuration/standalone.xml**.

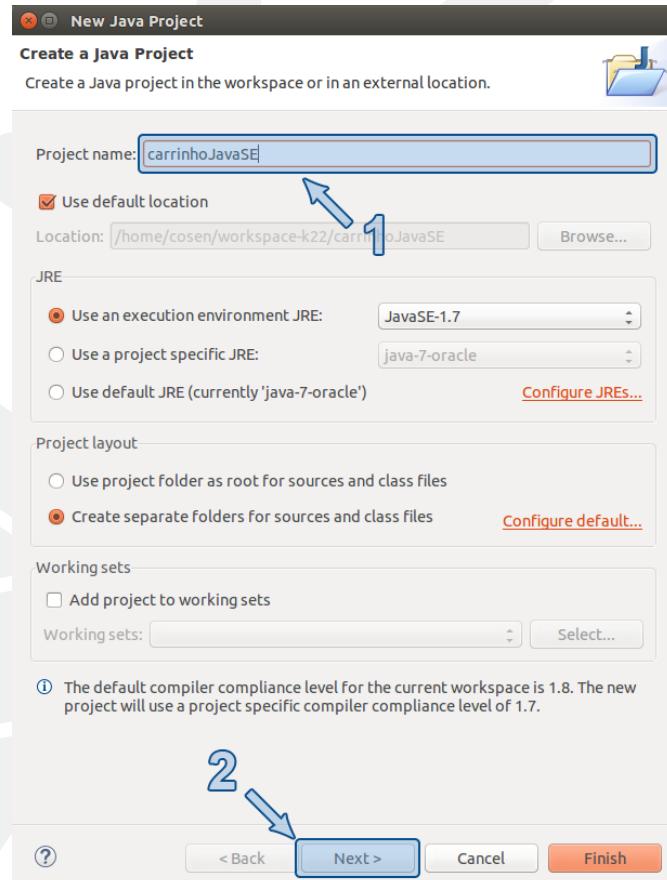
```

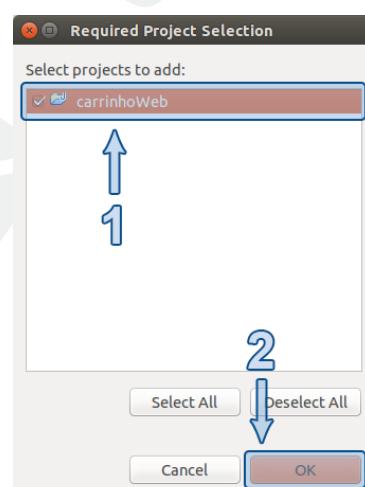
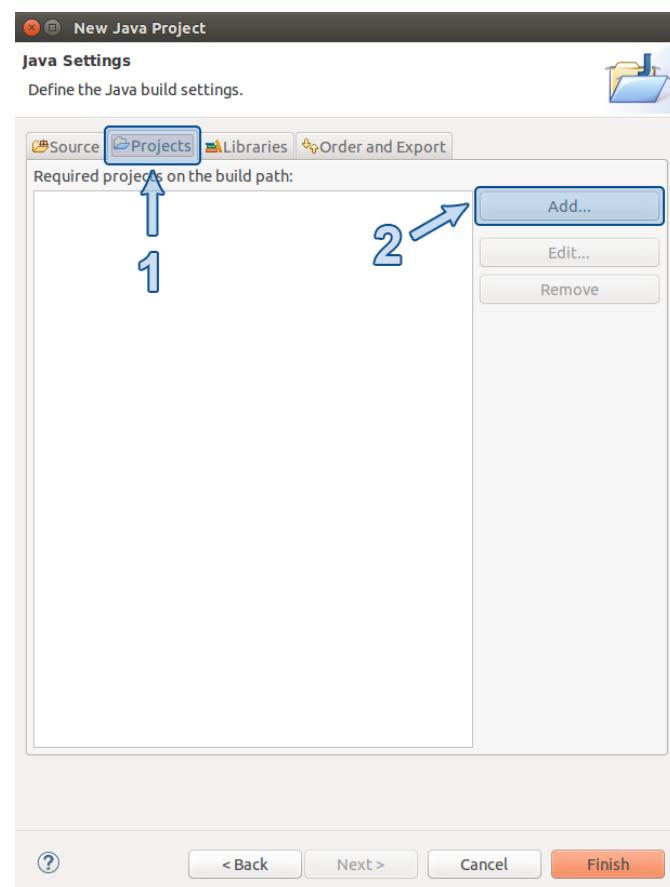
1 ...
2 <stateful default-access-timeout="5000" cache-ref="passivating"/>
3 ...
4 <cache name="passivating" passivation-store-ref="file" aliases="SimpleStatefulCache" />
5 ...
6 <file-passivation-store name="file"
7   idle-timeout="10"
8   idle-timeout-unit="SECONDS"
9   max-size="5"/>
10 ...

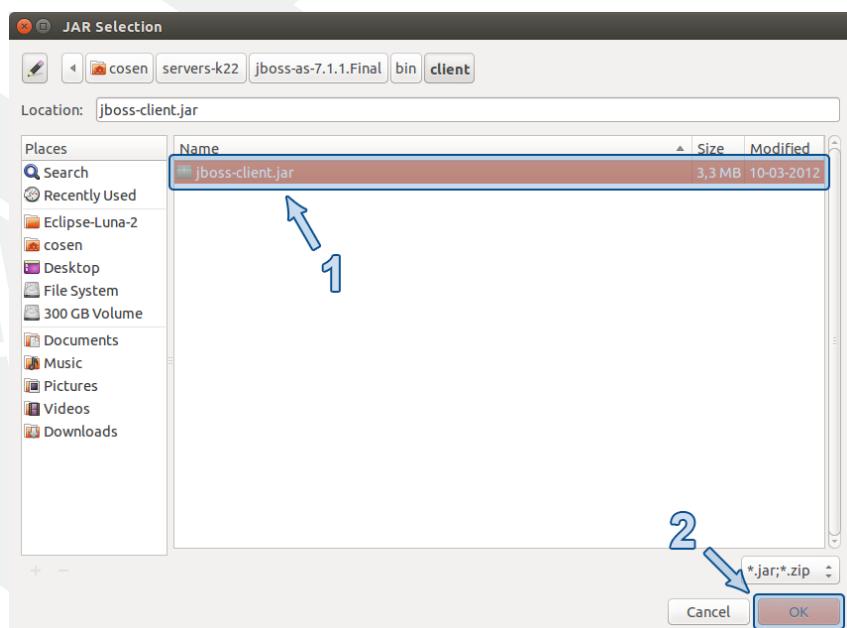
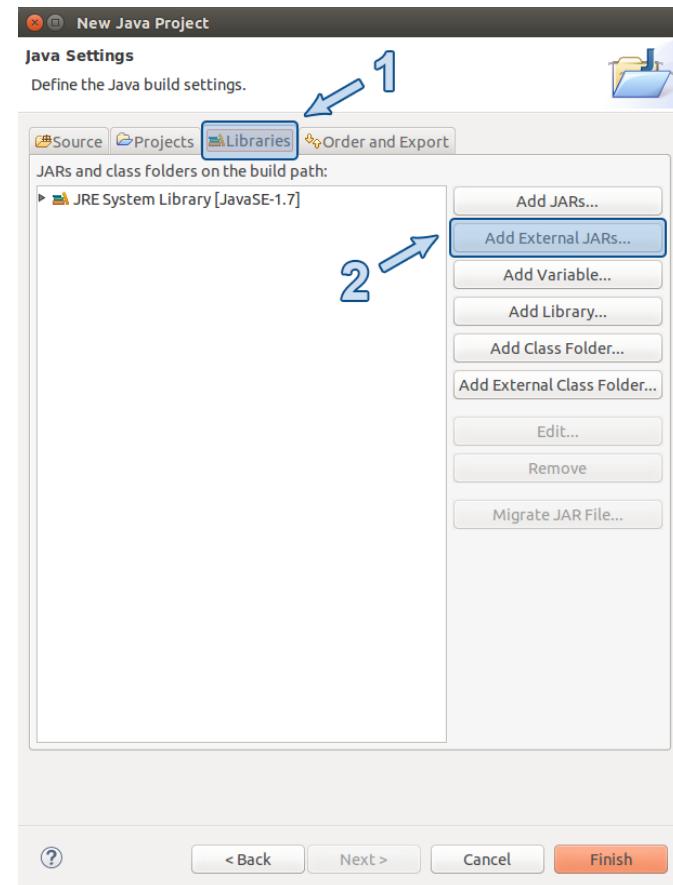
```

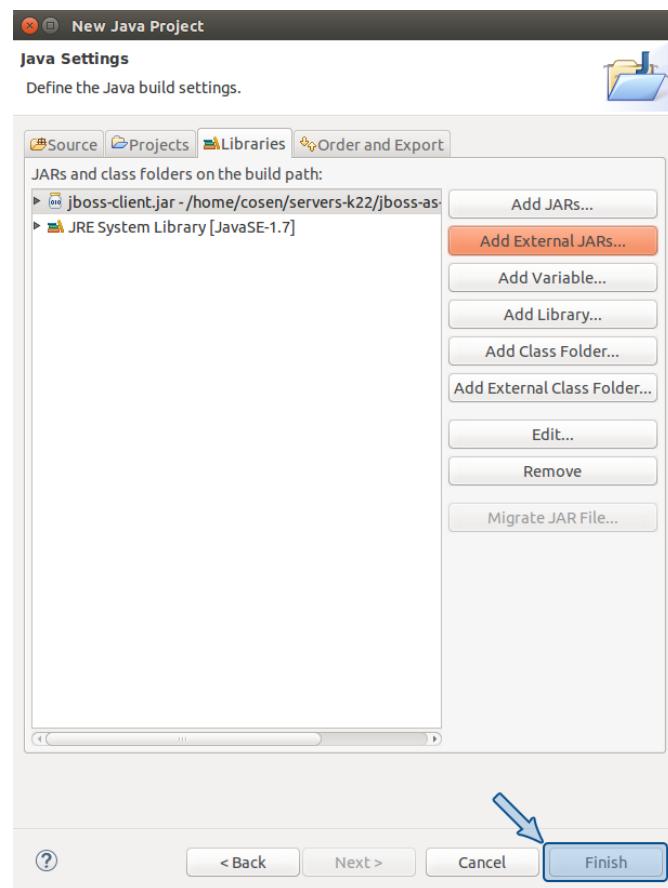
Código XML 3.1: standalone.xml

- 13** Crie um Java project no eclipse. Você pode digitar “CTRL+3” em seguida “new Java project” e “ENTER”. Depois, siga exatamente as imagens abaixo.









- 14 Adicione o arquivo **jboss-ejb-client.properties** na pasta **src** do projeto **carrinhoJavaSE**.

```

1 endpoint.name=client-endpoint
2 remote.connections=default
3 remote.connection.default.port=4447
4 remote.connection.default.host=localhost
5 remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
6 remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=<--  
false

```

Arquivo de Propriedades 3.1: jboss-ejb-client.properties

- 15 Crie uma classe chamada **TesteCicloDeVidaSFSB** em um pacote chamado **br.com.k19.testes** no projeto **carrinhoJavaSE**.

```

1 package br.com.k19.testes;
2
3 import java.util.Properties;
4
5 import javax.naming.Context;
6 import javax.naming.InitialContext;
7
8 import br.com.k19.sessionbeans.Carrinho;
9
10 public class TesteCicloDeVidaSFSB {
11     public static void main(String[] args) throws Exception {
12         Properties props = new Properties();
13         props.put(Context.INITIAL_CONTEXT_FACTORY,

```

```
14     "org.jboss.naming.remote.client.InitialContextFactory");
15 props.put(Context.PROVIDER_URL,
16           "remote://127.0.0.1:4447");
17 props.put(Context.SECURITY_PRINCIPAL, "k19");
18 props.put(Context.SECURITY_CREDENTIALS, "1234");
19
20 InitialContext ic = new InitialContext(props);
21
22 Carrinho[] carrinhos = new Carrinho[6];
23
24 for (int i = 0; i < carrinhos.length; i++) {
25     carrinhos[i] = (Carrinho)
26         ic.lookup("carrinhoWeb/CarrinhoBean!br.com.k19.sessionbeans.Carrinho");
27     carrinhos[i].adiciona("Chaveiro - K19");
28     carrinhos[i].adiciona("Caneta - K19");
29     Thread.sleep(1000);
30 }
31
32 carrinhos[0].adiciona("Borracha - K19");
33
34 Thread.sleep(5000);
35
36     carrinhos[0].finalizaCompra();
37 }
38 }
```

Código Java 3.21: TesteCicloDeVidaSFSB.java

- 16** Reinicie o JBoss. Depois, execute a classe **TesteCicloDeVidaSFSB** e observe o console do servidor.



SINGLETON SESSION BEANS



Caracterizando os Singleton Session Beans

Singleton Session Bean é o terceiro tipo de session bean. Este tipo de session bean surgiu na versão 3.1 da especificação Enterprise Java Beans. A ideia fundamental dos Singleton Session Beans é a necessidade de compartilhar dados transientes entre todos os usuários de uma aplicação EJB.

Número de usuários conectados

Para exemplificar, suponha que seja necessário contabilizar a quantidade de usuários conectados à aplicação. Esse serviço pode ser implementado através da classe abaixo.

```
1 class ContadorDeUsuariosBean {  
2     private int contador = 0;  
3  
4     public void adiciona() {  
5         this.contador++;  
6     }  
7  
8     public int getContador() {  
9         return this.contador;  
10    }  
11 }  
12 }
```

Código Java 4.1: ContadorDeUsuariosBean.java

Uma única instância da classe ContadorDeUsuariosBean deve ser criada para contabilizar corretamente o número de usuários conectados. Além disso, o contador de usuários conectados não precisa ser persistido entre duas execuções da aplicação.

Sistema de chat

Outro exemplo, suponha o funcionamento de um sistema de chat no qual as salas são criadas dinamicamente pelos usuários durante a execução. Podemos definir alguns métodos para implementar esse sistema.

```
1 class ChatBean {  
2     private Set<String> salas = new HashSet<String>();  
3  
4     public void criaSala(String sala) {  
5         this.salas.add(sala);  
6     }  
7  
8     public List<String> listaSalas(){  
9         return new ArrayList<String>(this.salas);  
10    }  
11 }
```

Código Java 4.2: ChatBean.java

As salas são criadas dinamicamente e todos os usuários compartilham todas as salas. Se o sistema “cair” por qualquer que seja o motivo não é necessário guardar as salas pois na próxima execução novas salas serão criadas pelos usuários. Uma única instância da classe ChatBean deve ser criada já que as salas serão compartilhadas entre todos os usuários.

Trânsito Colaborativo

Mais um exemplo, suponha um sistema colaborativo para informar o grau de congestionamento nas vias de uma cidade. As regras desse sistema poderiam ser implementadas através de alguns métodos.

```

1 class TransitoBean {
2     private Map<String, List<Integer>> vias = new HashMap<String, List<Integer>>();
3
4     public void registra(String via, Integer velocidade) {
5         if(this.vias.containsKey(via)) {
6             this.vias.get(via).add(velocidade);
7         }
8     }
9
10    public List<Integer> getVelocidadesRegistradas(String via) {
11        return this.vias.get(via);
12    }
13 }
```

Código Java 4.3: TransitoBean.java

Os dados sobre o trânsito são fornecidos pelos usuários e todos podem consultar as mesmas informações. A princípio, não é necessário manter esses dados persistidos.



Implementação

Para implementar um Singleton Session Bean podemos definir uma interface java com as assinaturas dos métodos de negócio. Por exemplo, suponha que um Singleton Session Bean será utilizado para implementar um sistema de chat.

```

1 public interface Chat {
2     void criaSala(String sala);
3     List<String> listaSalas();
4 }
```

Código Java 4.4: Chat.java

Após definir a interface de utilização, o segundo passo seria implementar as operações do session bean através de uma classe java.

```

1 public class ChatBean implements Chat {
2     private Set<String> salas = new HashSet<String>();
3
4     public void criaSala(String sala) {
5         this.salas.add(sala);
6     }
7 }
```

```

8
9     public List<String> listaSalas(){
10        return new ArrayList<String>(this.salas);
11    }
12 }
```

Código Java 4.5: ChatBean.java

O terceiro passo é especificar o tipo de session bean que queremos utilizar. No caso do chat, o tipo seria Singleton. Essa definição é realizada através da anotação **@Singleton**.

```

1 @Singleton
2 public class ChatBean implements Chat {
3 ...
4 }
```

Código Java 4.6: ChatBean.java

Por fim, é necessário definir se o session bean poderá ser acessado remotamente ou apenas localmente. Quando o acesso a um session bean é local, ele só pode ser acessado por aplicações que estejam no mesmo servidor de aplicação que ele. Caso contrário, quando o acesso a um session bean é remoto, ele pode ser acessado tanto por aplicações que estejam no mesmo servidor de aplicação quanto aplicações que não estejam.

A definição do tipo de acesso é realizada através das anotações: **@Local** e **@Remote**.

```

1 @Singleton
2 @Remote(Chat.class)
3 public class ChatBean implements Chat {
4 ...
5 }
```

Código Java 4.7: ChatBean.java

```

1 @Singleton
2 @Local(Chat.class)
3 public class ChatBean implements Chat {
4 ...
5 }
```

Código Java 4.8: ChatBean.java

Singleton Session Beans Locais

Quando o acesso a um Singleton Session Bean é local, não é necessário definir uma interface java nem utilizar a anotação **@Local**. Então, bastaria implementar uma classe java com a anotação **@Singleton**.

```

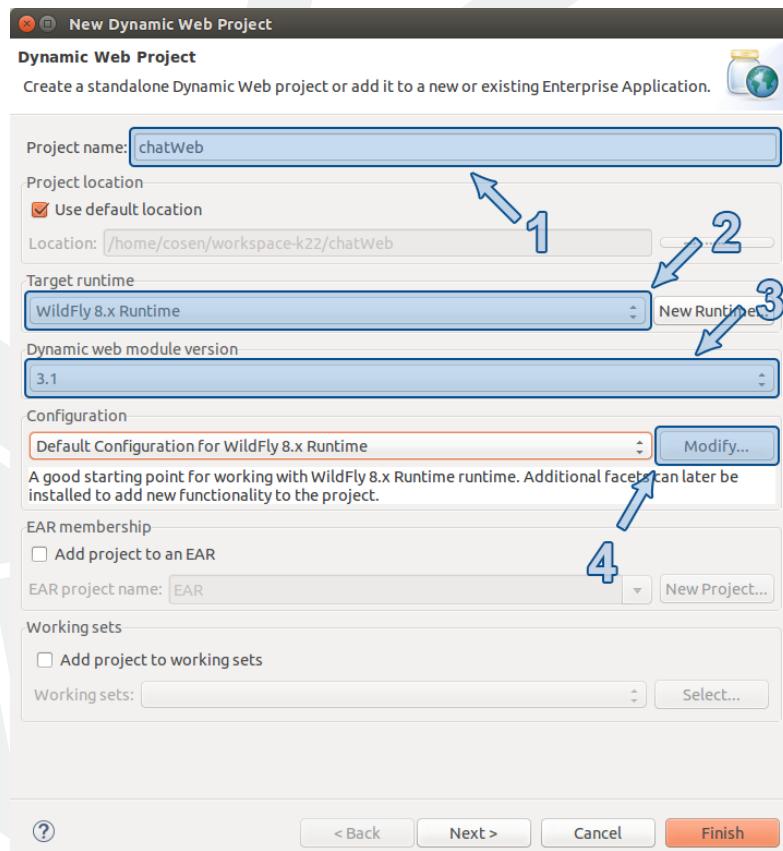
1 @Singleton
2 public class ChatBean {
3
4     private Set<String> salas = new HashSet<String>();
5
6     public void criaSala(String sala) {
7         this.salas.add(sala);
8     }
9
10    public List<String> listaSalas(){
11        return new ArrayList<String>(this.salas);
12    }
13 }
```

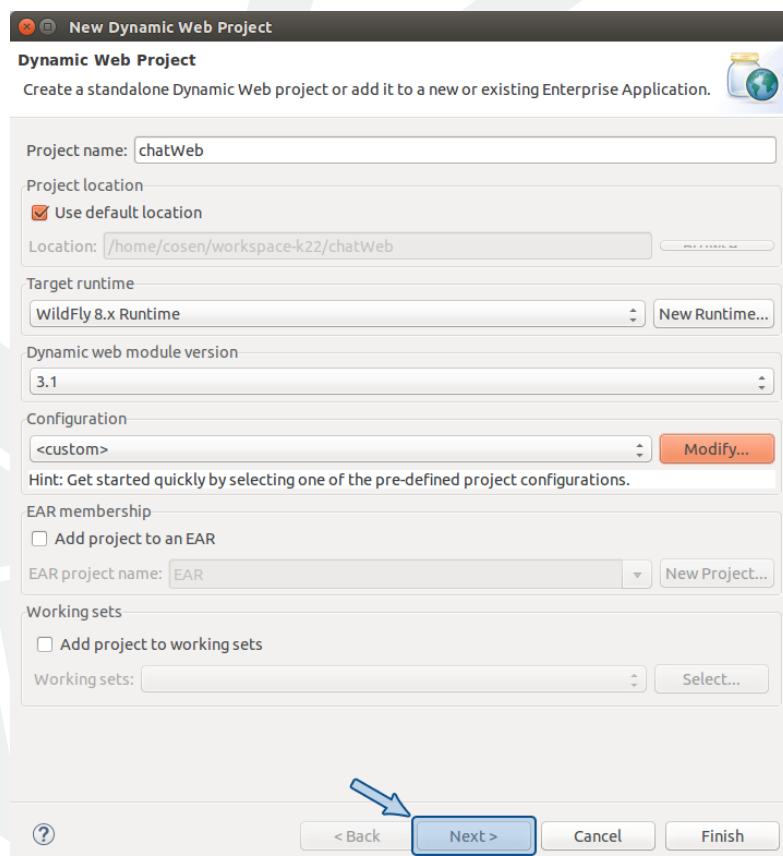
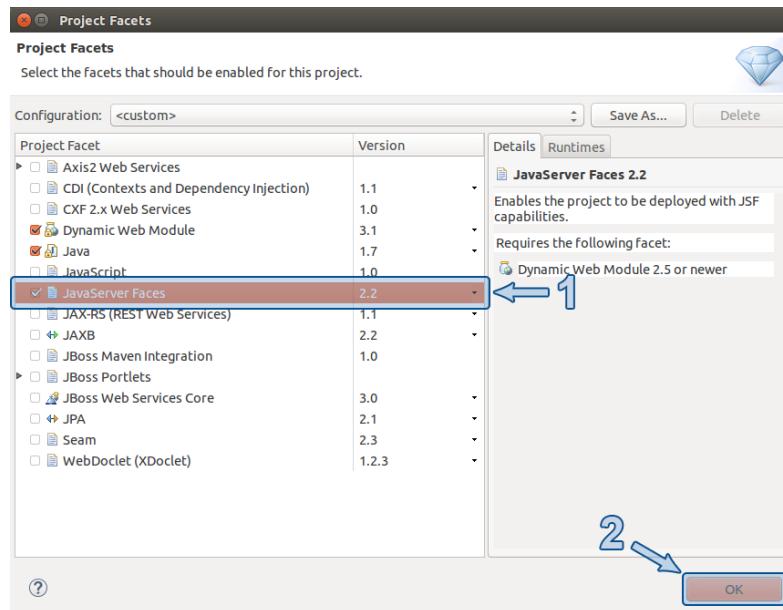
13 }

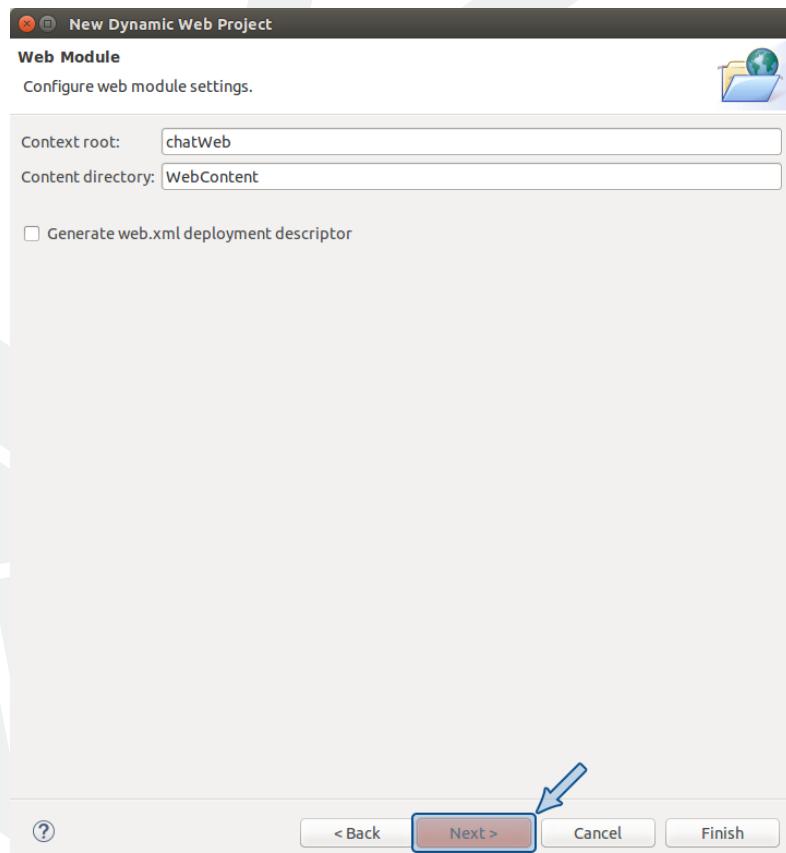
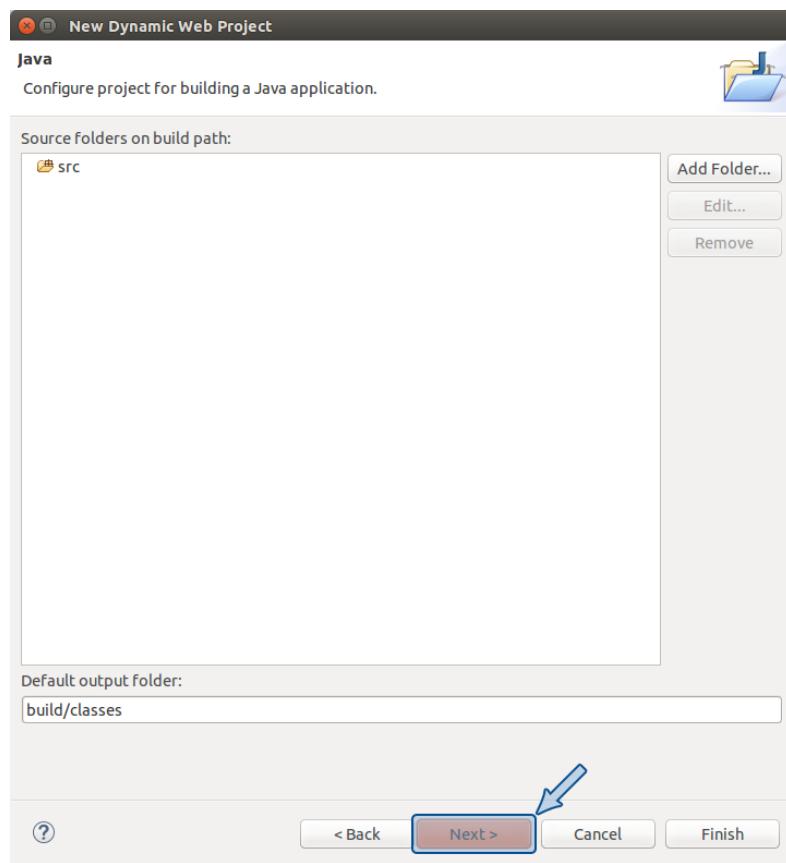
Código Java 4.9: ChatBean.java

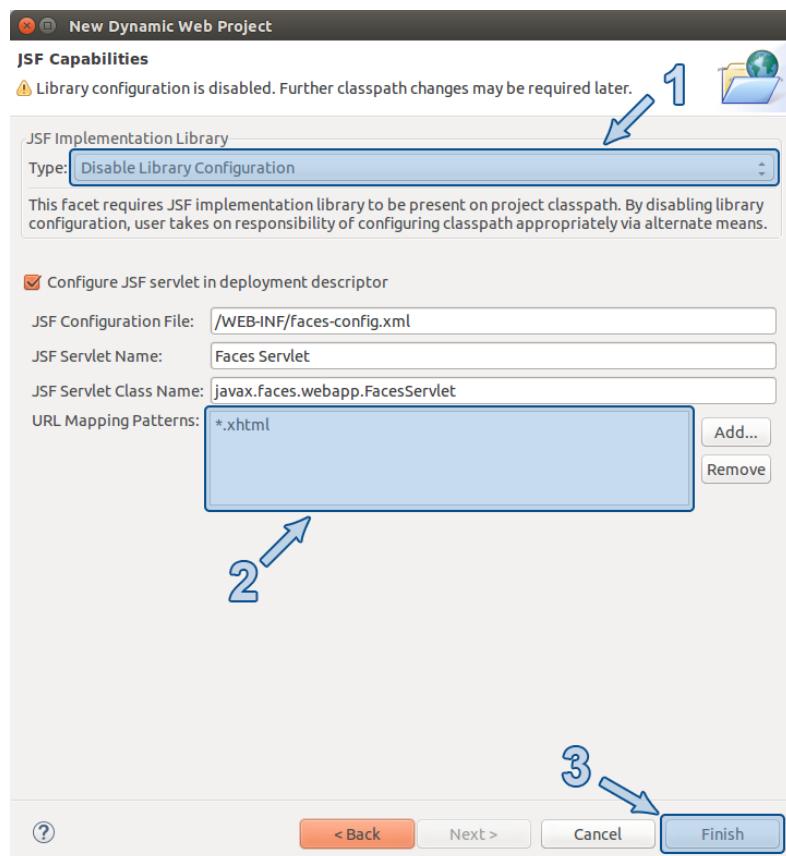
Exercícios de Fixação

- 1 Para não confundir, feche os projetos **carrinhoWeb** e **carrinhoJavaSE**. Para isso, clique com o botão direito do mouse sobre esses projetos e selecione a opção “Close Project”.
- 2 Crie um Dynamic Web Project no eclipse para implementar o funcionamento de um sistema de chat. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.









- 3 Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **chatWeb** e adicione a seguinte classe nesse pacote.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.ArrayList;
4 import java.util.HashSet;
5 import java.util.List;
6 import java.util.Set;
7
8 import javax.ejb.Singleton;
9
10 @Singleton
11 public class ChatBean {
12
13     private Set<String> salas = new HashSet<String>();
14
15     public void criaSala(String sala) {
16         this.salas.add(sala);
17     }
18
19     public List<String> listaSalas(){
20         return new ArrayList<String>(this.salas);
21     }
22 }
```

Código Java 4.10: ChatBean.java

- 4 Crie um pacote chamado **br.com.k19.managedbeans** no projeto **chatWeb** e adicione a seguinte classe.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.ejb.EJB;
6 import javax.faces.bean.ManagedBean;
7
8 import br.com.k19.sessionbeans.ChatBean;
9
10 @ManagedBean
11 public class ChatMB {
12
13     @EJB
14     private ChatBean chatBean;
15
16     private String sala;
17
18     public void adicionaSala() {
19         this.chatBean.criaSala(this.sala);
20     }
21
22     public List<String> getSalas() {
23         return this.chatBean.listaSalas();
24     }
25
26     public void setSala(String sala) {
27         this.sala = sala;
28     }
29
30     public String getSala() {
31         return sala;
32     }
33 }
```

Código Java 4.11: ChatMB.java

- 5 Adicione o arquivo **chat.xhtml** na pasta **WebContent** do projeto **chatWeb** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5       xmlns:ui="http://java.sun.com/jsf/facelets"
6       xmlns:h="http://java.sun.com/jsf/html"
7       xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10    <title>Chat</title>
11 </h:head>
12
13 <h:body>
14    <h:form>
15        <h:outputLabel value="Nova Sala: "/>
16        <h:inputText value="#{chatMB.sala}" />
17        <h:commandButton value="Criar" action="#{chatMB.adicionaSala}" />
18
19        <hr/>
20
21        <h:dataTable value="#{chatMB.salas}" var="sala">
22            <h:column>
23                <h:outputText value="#{sala}" />
24            </h:column>
```

```

25 </h: dataTable>
26 </h: form>
27 </h: body>
28 </html>
```

Código XHTML 4.1: chat.xhtml

- 6** Remova o projeto **carrinhoWeb** do **JBoss**. Adicione o projeto **chatWeb** no **Wildfly**. Certifique-se que o **Glassfish** e o **JBoss** estejam parado. Inicie o **Wildfly** e teste a aplicação acessando a url <http://localhost:8080/chatWeb/chat.xhtml>. Utilize pelo menos dois navegadores diferentes para acessar a aplicação e observe que as mesmas salas são apresentadas nesses navegadores.



Ciclo de Vida

As instâncias dos Singleton Session Beans são administradas pelo EJB Container. Devemos entender o de ciclo de vida desses objetos para utilizar corretamente a tecnologia EJB. Para entender mais facilmente o ciclo de vida das instâncias dos Singleton Session Beans, devemos sempre ter em mente que o EJB Container cria apenas uma instância de cada session bean desse tipo.

Estados

O ciclo de vida das instâncias dos Singleton Session Beans possui dois estados.

1. NÃO EXISTE
2. PRONTO

NÃO EXISTE -> PRONTO

Antes de ser criada, dizemos que uma instância de um Singleton Session Bean se encontra no estado NÃO EXISTE. Obviamente, nesse estado, uma instância não pode atender as chamadas dos clientes da aplicação.

O EJB Container cria apenas uma instância para cada Singleton Session Bean. Por padrão, o EJB Container é quem decide quando a criação da instância de um Singleton Session Bean deve ser realizada. Contudo, é possível determinar que essa criação seja realizada na inicialização da aplicação através da anotação **@Startup**.

```

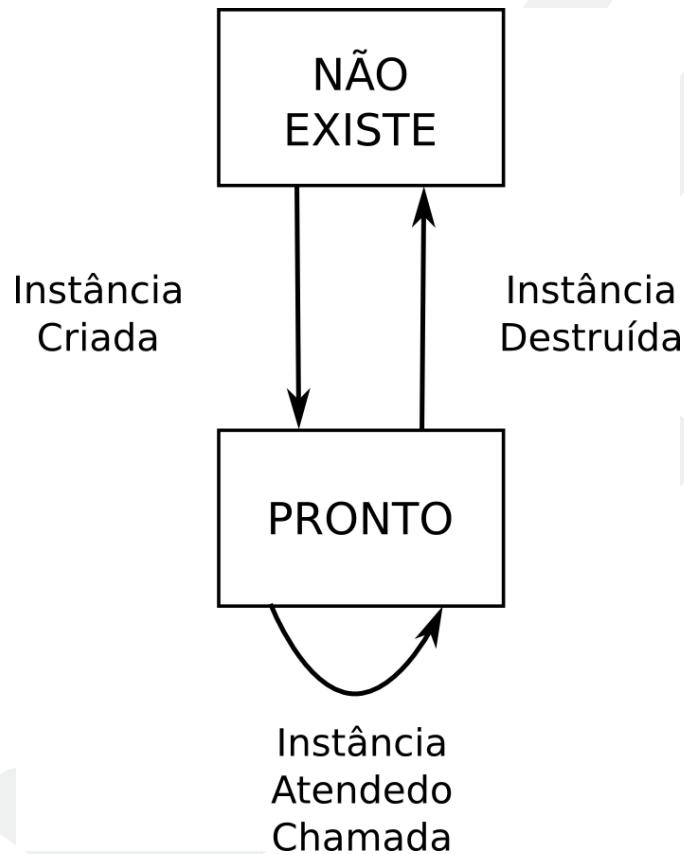
1 @Singleton
2 @Startup
3 class ContadorDeUsuariosBean {
4
5     private int contador = 0;
6
7     public void adiciona() {
8         this.contador++;
9     }
10
11    public int getContador() {
12        return this.contador;
13    }
14}
```

Código Java 4.12: ContadorDeUsuariosBean.java

Quando a instância de um Singleton Session Bean é criada, ela passa para do estado NÃO EXISTE para o estado PRONTO e pode atender as chamadas dos clientes da aplicação.

PRONTO -> NÃO EXISTE

Quando a aplicação é finalizada, o EJB Container destrói as instâncias dos Singleton Session Beans. Dessa forma, elas passam do estado PRONTO para o NÃO EXISTE.



Callbacks

Podemos associar lógicas específicas nas transições de estado no ciclo de vida dos Singleton Session Beans.

@PostConstruct

Podemos registrar um método de instância no EJB Container para que ele o execute em cada instância logo após ela ser criada. Esse registro é realizado através da anotação **@PostConstruct**.

```

1  @Singleton
2  class ContadorDeUsuariosBean {
3
4      @PostConstruct
5      public void inicializando() {
6          System.out.println("Contador de usuários criado...");
7      }
8
9      // METODOS DE NEGOCIO
10 }
  
```

Código Java 4.13: ContadorDeUsuariosBean.java

O EJB Container utiliza o construtor sem argumentos para criar a instância de um Singleton Session Bean. Depois de chamar o construtor sem argumentos, o EJB Container injeta eventuais dependências na instância criada. Por fim, os métodos anotados com @PostConstruct são executados.

@PreDestroy

Também podemos registrar um método de instância no EJB Container para que ele o execute em cada instância imediatamente antes dela ser destruída. Esse registro é realizado através da anotação **@PreDestroy**.

```

1  @Singleton
2  class ContadorDeUsuariosBean {
3
4      @PreDestroy
5      public void destruindo() {
6          System.out.println("Contador de usuários será destruído...");
7      }
8
9      // MÉTODOS DE NEGÓCIO
10 }
```

Código Java 4.14: ContadorDeUsuariosBean.java



Exercícios de Fixação

- 7 Altere a classe **ChatBean** do projeto **chatWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.ArrayList;
4 import java.util.HashSet;
5 import java.util.List;
6 import java.util.Set;
7
8 import javax.annotation.PostConstruct;
9 import javax.annotation.PreDestroy;
10 import javax.ejb.Singleton;
11
12 @Singleton
13 public class ChatBean {
14
15     private Set<String> salas = new HashSet<String>();
16
17     public void criaSala(String sala) {
18         this.salas.add(sala);
19     }
20
21     public List<String> listaSalas(){
22         return new ArrayList<String>(this.salas);
23     }
24
25     @PostConstruct
26     public void postConstruct() {
27         System.out.println("Criando o ChatBean...");
28     }
29
30     @PreDestroy
31     public void preDestroy() {
32         System.out.println("Destruindo o ChatBean...");
33     }
34 }
```

34 }

Código Java 4.15: ChatBean.java

8 Teste os métodos de callback seguindo os passos abaixo.

- Reinicie o **Wildfly** e observe que a mensagem de criação não aparece no console;
- Acesse o endereço <http://localhost:8080/chatWeb/chat.xhtml> e observe que agora a mensagem aparece no console;
- Tente acessar o mesmo endereço através de outros navegadores e repare que a mensagem de criação não aparece novamente no console.
- Pare o **Wildfly** e observe no console a mensagem de destruição.

9 Altere novamente a classe **ChatBean**.

```

1 ...
2 @Singleton
3 @Startup
4 public class ChatBean {
5 ...
6 }
```

Código Java 4.16: ChatBean.java

10 Inicie o **Wildfly** e observe que a mensagem de criação já aparece no console.



Concorrência

Um Singleton Session Bean suporta acesso concorrente. Em outras palavras, a instância de um Singleton Session Bean pode processar diversas chamadas a métodos de negócio simultaneamente.

Há dois modos de gerenciamento de acesso à instância de um Singleton Session Bean:

- Container Managed Concurrency (CMC)
- Bean Managed Concurrency (BMC)

O modo padrão de gerenciamento é o CMC. Opcionalmente, podemos declarar o modo CMC através da anotação **@ConcurrencyManagement**. Mas, lembre-se que não é necessário pois esse é o modo padrão.

```

1 @Singleton
2 @ConcurrencyManagement(ConcurrencyManagementType.CONTAINER)
3 public class ContadorDeUsuariosBean {
4 ...
5 }
```

Código Java 4.17: ContadorDeUsuariosBean.java

Para selecionar o modo BMC, devemos utilizar a anotação **@ConcurrencyManagement**.

```

1 @Singleton
2 @ConcurrencyManagement(ConcurrencyManagementType.BEAN)
3 public class ContadorDeUsuariosBean {
4     ...
5 }
```

Código Java 4.18: ContadorDeUsuariosBean.java

CMC

No modo CMC, todo método de negócio é associado ao **Read Lock** ou ao **Write Lock**. Chamadas a métodos associados ao Read Lock podem ser executadas simultaneamente. Por outro lado, chamadas a métodos associados ao Write Lock são executadas uma de cada vez.

Por padrão, no CMC, os métodos são associados ao Write Lock. Opcionalmente, podemos declarar o tipo de Lock através da anotação **@Lock**. Mas, lembre-se que não é necessário pois o Write Lock é associado aos métodos de negócio por padrão.

```

1 @Singleton
2 @Lock(LockType.WRITE)
3 public class ContadorDeUsuariosBean {
4     private int contador = 0;
5
6     public void adiciona() {
7         this.contador++;
8     }
9
10    public int getContador() {
11        return this.contador;
12    }
13 }
```

Código Java 4.19: ContadorDeUsuariosBean.java

```

1 @Singleton
2 public class ContadorDeUsuariosBean {
3     private int contador = 0;
4
5     @Lock(LockType.WRITE)
6     public void adiciona() {
7         this.contador++;
8     }
9
10    public int getContador() {
11        return this.contador;
12    }
13 }
```

Código Java 4.20: ContadorDeUsuariosBean.java

A anotação **@Lock** pode ser aplicada na classe do session bean ou diretamente nos métodos de negócio.

Para associar o Read Lock aos métodos de negócio, devemos utilizar a anotação **@Lock** na classe do session bean ou nos métodos de negócio.

```

1 @Singleton
2 @Lock(LockType.READ)
3 public class ContadorDeUsuariosBean {
4     private int contador = 0;
5 }
```

```

6  public void adiciona() {
7      this.contador++;
8  }
9
10 public int getContador() {
11     return this.contador;
12 }
13 }
```

Código Java 4.21: ContadorDeUsuariosBean.java

```

1 @Singleton
2 public class ContadorDeUsuariosBean {
3     private int contador = 0;
4
5     public void adiciona() {
6         this.contador++;
7     }
8
9     @Lock(LockType.READ)
10    public int getContador() {
11        return this.contador;
12    }
13 }
```

Código Java 4.22: ContadorDeUsuariosBean.java

BMC

No modo BMC, o controle de concorrência deve ser implementado dentro dos métodos de negócio. Você pode utilizar a instrução **synchronized** ou os recursos do pacote **java.util.concurrent**. Para isso, você precisa ter bons conhecimento de programação concorrente.



Exercícios de Fixação

- 11** Crie uma interface chamada **Contador** no pacote **br.com.k19.sessionbeans** do projeto **chatWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 public interface Contador {
4
5     void incrementa();
6
7     int getValor();
8 }
```

Código Java 4.23: Contador.java

- 12** Crie uma classe chamada **ContadorBean** no pacote **br.com.k19.sessionbeans** do projeto **chatWeb**.

```

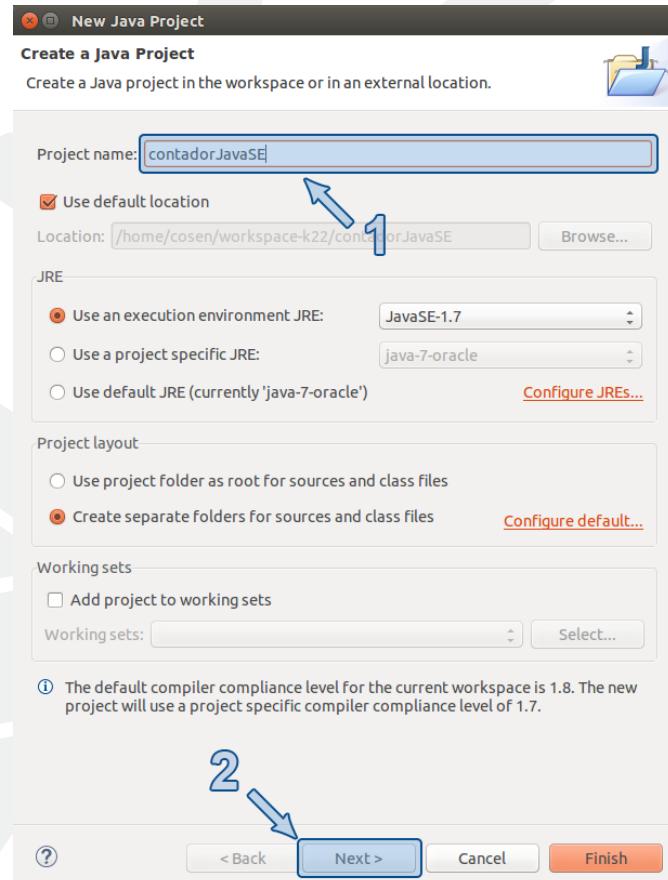
1 package br.com.k19.sessionbeans;
2
3 import javax.ejb.Lock;
4 import javax.ejb.LockType;
5 import javax.ejb.Remote;
6 import javax.ejb.Singleton;
7
```

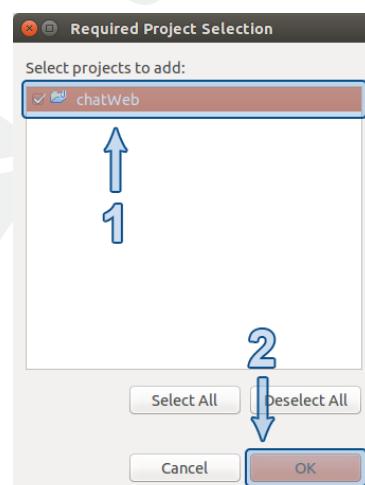
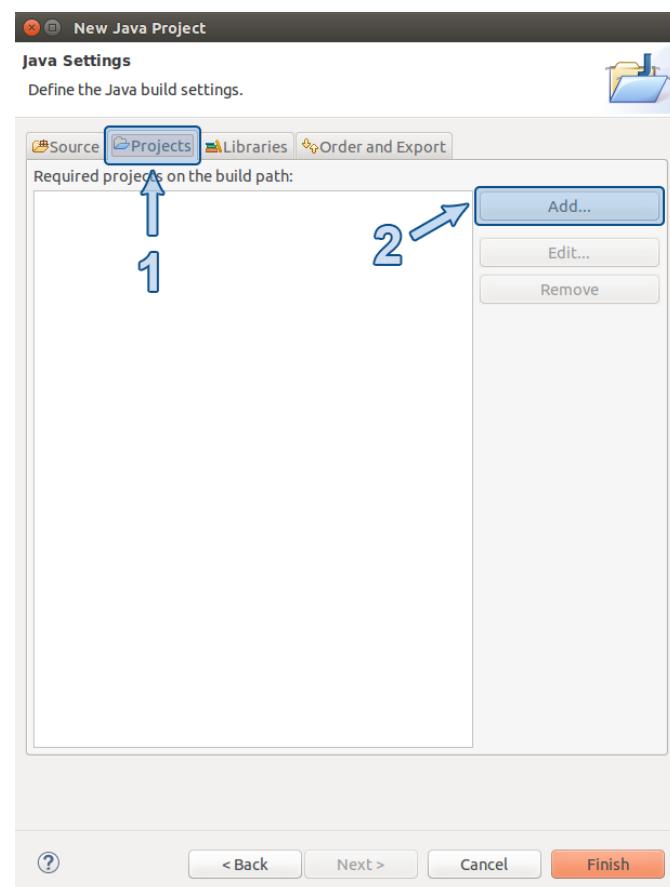
```

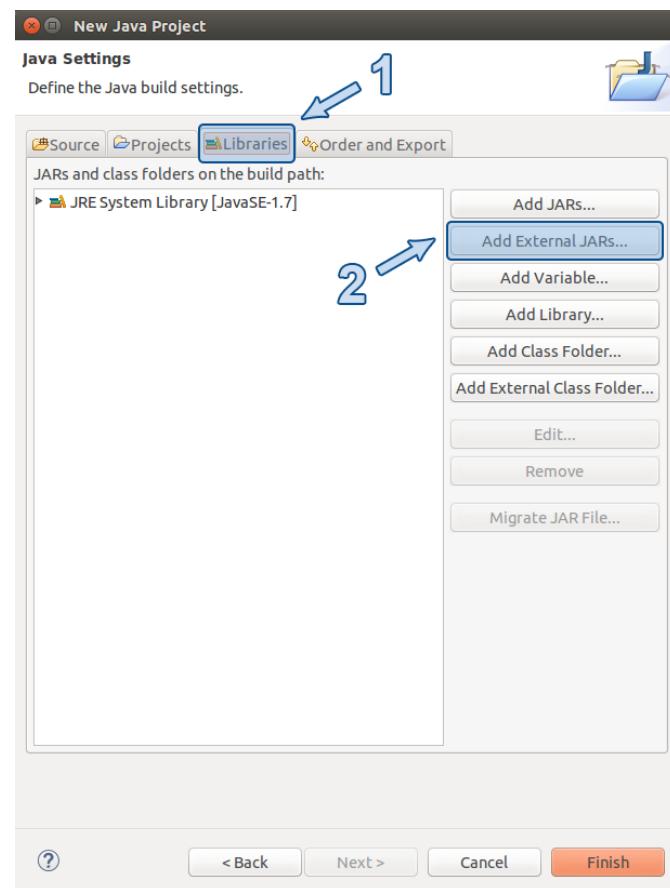
8 @Singleton
9 @Lock(LockType.READ)
10 @Remote(Contador.class)
11 public class ContadorBean implements Contador{
12
13     private int valor;
14
15     public void incrementa() {
16         this.valor++;
17     }
18
19     public int getValor() {
20         return this.valor;
21     }
22 }
```

Código Java 4.24: ContadorBean.java

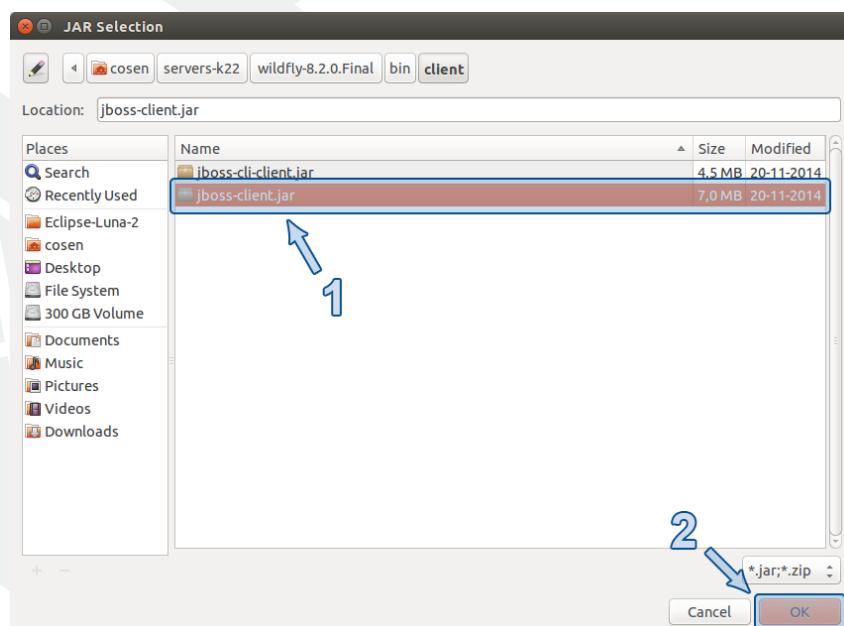
- 13** Crie um Java project no eclipse. Você pode digitar “CTRL+3” em seguida “new Java project” e “ENTER”. Depois, siga exatamente as imagens abaixo.

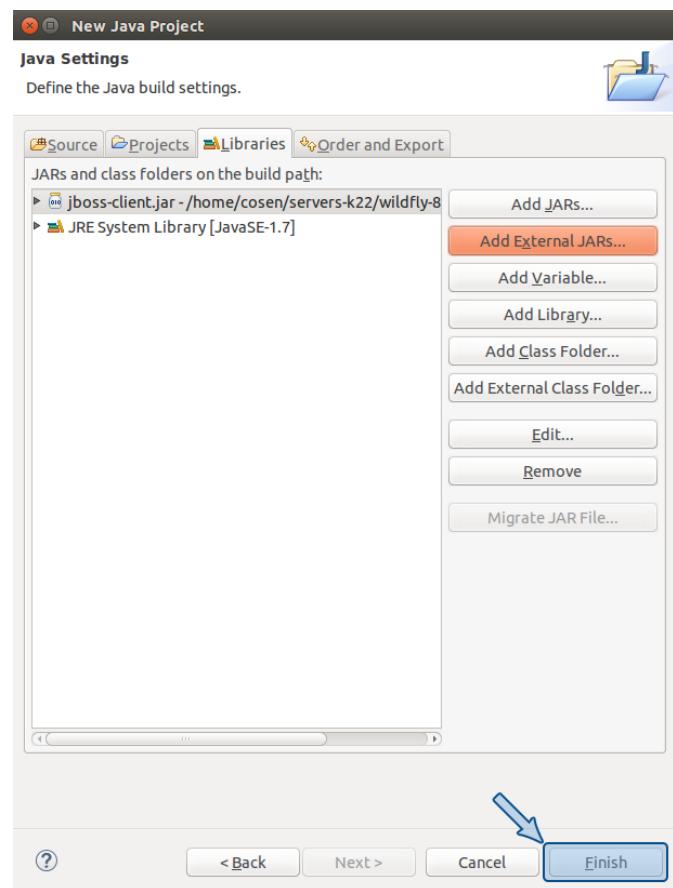






Importante: Selecione o arquivo **jboss-client.jar** contido na pasta **client** que está na pasta **bin** do diretório de instalação do **Wildfly**.





- 14 Adicione um usuário no Wildfly com as seguintes características:

Tipo: Application User

Username: k19

Password: singleton123#

Utilize o script **add-user.sh** para adicionar o usuário **k19** no JBoss. Siga os passos abaixo.

```
osen@k19:~/Desktop/wildfly-8.2.0.Final/bin$ ./add-user.sh
What type of user do you wish to add?
 a) Management User (mgmt-users.properties)
 b) Application User (application-users.properties)
(a): b

Enter the details of the new user to add.
Using realm 'ApplicationRealm' as discovered from the existing property files.
Username : k19
Password recommendations are listed below. To modify these restrictions edit
the add-user.properties configuration file.
- The password should not be one of the following restricted values {root,
admin, administrator}
- The password should contain at least 8 characters, 1 alphabetic character(s),
1 digit(s), 1 non-alphanumeric symbol(s)
- The password should be different from the username
Password :
Re-enter Password :
What groups do you want this user to belong to? (Please enter a comma separated
```

```

list, or leave blank for none)[ ]:
About to add user 'k19' for realm 'ApplicationRealm'
Is this correct yes/no? yes
Added user 'k19' to file '/home/cosen/servers-k22/wildfly-8.2.0.Final/standalone
/configuration/application-users.properties'
Added user 'k19' to file '/home/cosen/servers-k22/wildfly-8.2.0.Final/domain
/configuration/application-users.properties'
Added user 'k19' with groups to file '/home/cosen/servers-k22/wildfly-8.2.0.Final/
standalone/configuration/application-roles.properties'
Added user 'k19' with groups to file '/home/cosen/servers-k22/wildfly-8.2.0.Final/
domain/configuration/application-roles.properties'
Is this new user going to be used for one AS process to connect to another AS process?
e.g. for a slave host controller connecting to the master or for a Remoting
connection for server to server EJB calls.
yes/no? yes
To represent the user add the following to the server-identities definition
<secret value="c2luZ2x1dG9uMTIzJCU=" />

```

Terminal 4.1: Criando um usuário no JBoss

- 15** Adicione o arquivo **jboss-ejb-client.properties** na pasta **src** do projeto **contadorJavaSE**.

```

1 endpoint.name=client-endpoint
2 remote.connectionprovider.create.options.org.xnio.Options.SSL_ENABLED=false
3
4 remote.connections=default
5
6 remote.connection.default.host=localhost
7 remote.connection.default.port = 8080
8 remote.connection.default.connect.options.org.xnio.Options.SASL_POLICY_NOANONYMOUS=<-->
    false
9
10 remote.connection.default.username=k19
11 remote.connection.default.password=singleton123#

```

Arquivo de Propriedades 4.1: jboss-ejb-client.properties

- 16** Crie uma classe chamada **TesteDeAcessoConcorrente** em um pacote chamado **br.com.k19.testes** no projeto **contadorJavaSE**.

```

1 package br.com.k19.testes;
2
3 import java.util.Properties;
4
5 import javax.naming.Context;
6 import javax.naming.InitialContext;
7
8 import br.com.k19.sessionbeans.Contador;
9
10 public class TesteDeAcessoConcorrente {
11     public static void main(String[] args) throws Exception {
12         Properties props = new Properties();
13         props.put(Context.INITIAL_CONTEXT_FACTORY,
14             "org.jboss.naming.remote.client.InitialContextFactory");
15         props.put(Context.PROVIDER_URL, "http-remoting://localhost:8080");
16         props.put(Context.SECURITY_PRINCIPAL, "k19");
17         props.put(Context.SECURITY_CREDENTIALS, "singleton123#");
18
19         InitialContext ic = new InitialContext(props);
20
21         final Contador contador = (Contador) ic
22             .lookup("chatWeb/ContadorBean!br.com.k19.sessionbeans.Contador");
23
24         final Thread[] threads = new Thread[20];
25
26         System.out.println("Contador = " + contador.getValor());

```

```

27     System.out.println("Incrementando " + threads.length * threads.length
28             + " vezes");
29     for (int i = 0; i < threads.length; i++) {
30         threads[i] = new Thread(new Runnable() {
31             @Override
32             public void run() {
33                 for (int i = 0; i < threads.length; i++) {
34                     contador.incrementa();
35                 }
36             }
37         });
38         threads[i].start();
39     }
40
41     for (Thread thread : threads) {
42         thread.join();
43     }
44
45     System.out.println("Contador = " + contador.getValor());
46 }
47 }
```

Código Java 4.25: TesteDeAcessoConcorrente.java

- 17** Reinicie o **Wildfly**. Execute algumas vezes a classe **TesteDeAcessoConcorrente** e observe que o valor do contador **não** estará correto.

- 18** Modifique a classe **ContadorBean** do projeto **chatWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import javax.ejb.Lock;
4 import javax.ejb.LockType;
5 import javax.ejb.Remote;
6 import javax.ejb.Singleton;
7
8 @Singleton
9 @Remote(Contador.class)
10 public class ContadorBean implements Contador{
11
12     private int valor;
13
14     public void incrementa() {
15         this.valor++;
16     }
17
18     @Lock(LockType.READ)
19     public int getValor() {
20         return this.valor;
21     }
22 }
```

Código Java 4.26: ContadorBean.java

- 19** Reinicie o **Wildfly**. Execute algumas vezes a classe **TesteDeAcessoConcorrente** e observe que o valor do contador estará correto.

PERSISTÊNCIA



Data Sources

Aplicações Java se comunicam com banco de dados através de conexões JDBC. Para estabelecer uma conexão JDBC, algumas informações como usuário, senha e base de dados são necessárias.

As configurações relativas às conexões JDBC podem ser definidas nas aplicações ou nos servidores de aplicação. Quando definidas em uma aplicação serão utilizadas somente por essa aplicação. Quando definidas em um servidor de aplicação podem ser utilizadas em diversas aplicações.

Em um servidor de aplicação, as configurações JDBC são definidas em componentes chamados **Data Sources**. A criação de Data Sources depende do servidor de aplicação utilizado. Em particular, no Glassfish, os Data Sources podem ser criados através da interface de administração.

Os Data Sources permitem que uma única configuração JDBC seja utilizada por diversas aplicações. Eles também permitem que outros tipos de configurações sejam compartilhadas. Por exemplo, a configuração de um Connection Pool.

Além disso, através de Data Sources podemos utilizar facilmente o serviço de transações dos servidores de aplicação. Esse serviço é definido pela especificação Java Transaction API (JTA).



Exercícios de Fixação

- Vamos configurar um Data Source no Glassfish. Copie o arquivo **mysql-connector-java-VERSAO-bin.jar** que se encontra na pasta **K19-Arquivos/mysql-connector-java-VERSAO** da Área de Trabalho para a pasta **glassfishv4/glassfish/lib**.



Importante

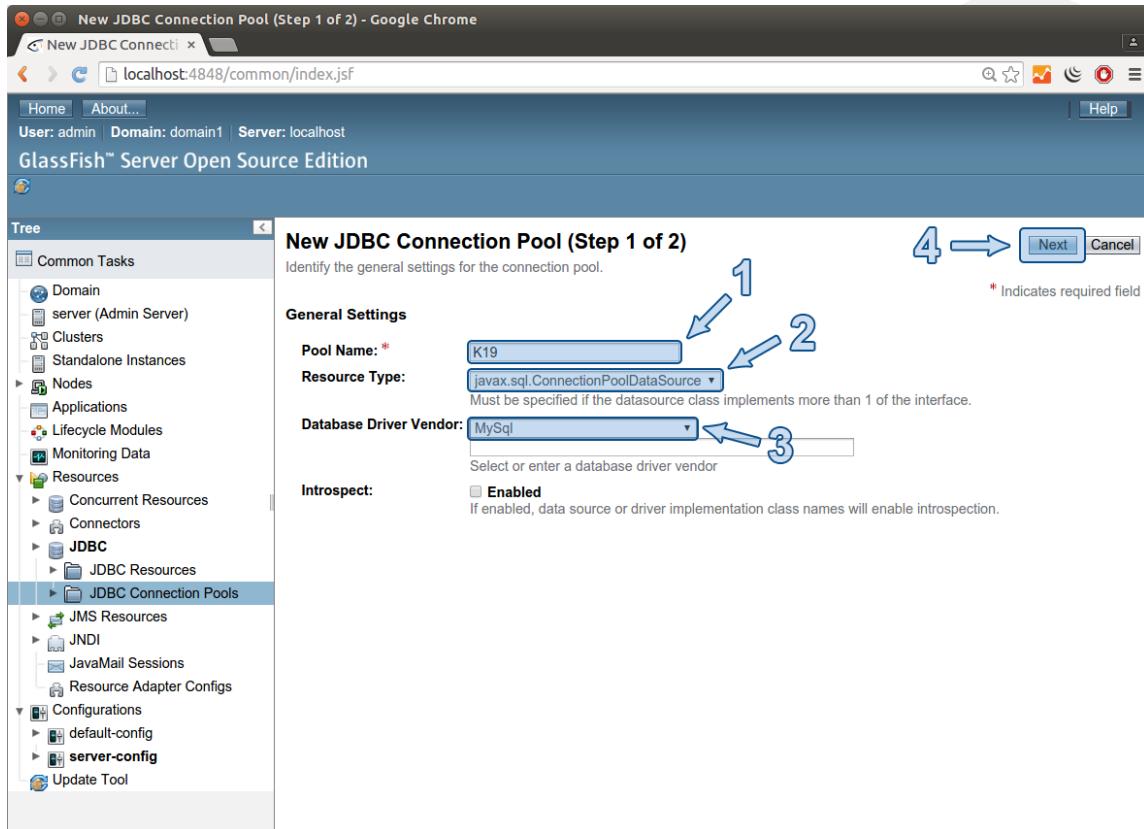
Você também pode obter o driver JDBC do MySQL através do site da K19: www.k19.com.br/arquivos.

- Acesse o MySQL Server através do MySQL Workbench ou através do cliente de linha de comando; apague a base de dados **k22_glassfish** caso ela exista; crie uma base de dados chamada **k22_glassfish**.

- 3 Reinicie o **Glassfish**; abra a interface de administração acessando a url **localhost:4848**. Siga os passos abaixo:

The screenshot shows the GlassFish Admin Console interface for managing JDBC connection pools. The URL is `localhost:4848/common/index.jsf`. The left sidebar has a tree view with nodes like 'Domain', 'Nodes', 'Resources' (selected), 'JDBC Resources' (selected), and 'JDBC Connection Pools' (selected). The main content area is titled 'JDBC Connection Pools' and contains a table of pools. The table has columns: Select, Pool Name, Resource Type, Classname, and Description. There are two rows: 'DerbyPool' (javax.sql.DataSource, org.apache.derby.jdbc.ClientDataSource) and '__TimerPool' (javax.sql.XADataSource, org.apache.derby.jdbc.EmbeddedXADataSource). A blue arrow labeled '1' points to the 'JDBC Connection Pools' node in the sidebar. A blue arrow labeled '2' points to the 'New...' button in the top right of the table header.

| Select | Pool Name | Resource Type | Classname | Description |
|--------|-------------|------------------------|--|-------------|
| | DerbyPool | javax.sql.DataSource | org.apache.derby.jdbc.ClientDataSource | |
| | __TimerPool | javax.sql.XADataSource | org.apache.derby.jdbc.EmbeddedXADataSource | |



Defina os seguintes valores para as seguintes propriedades:

DatabaseName: k22_glassfish

Password: root

ServerName: localhost

URL: jdbc:mysql://localhost:3306/k22_glassfish

url: jdbc:mysql://localhost:3306/k22_glassfish

User: root

Depois, clique em **finish**. Para testar o Connection Pool K19, siga os passos abaixo:

Edit JDBC Connection Pool

Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.

General Settings

- Pool Name:** K19
- Resource Type:** javax.sql.ConnectionPoolDataSource
- Datasource Classname:** com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource
- Driver Classname:** (empty)
- Ping:** Enabled
- Deployment Order:** 100
- Description:** (empty)

Pool Settings

- Initial and Minimum Pool Size:** 8 Connections
- Maximum Pool Size:** 32 Connections

JDBC Connection Pools

To store, organize, and retrieve data, most applications use relational databases. Java EE applications access relational databases through the JDBC API. Before an application can access a database, it must get a connection.

| Select | Pool Name | Resource Type | Classname | Description |
|--------------------------|-----------|------------------------------------|---|-------------|
| <input type="checkbox"/> | DerbyPool | javax.sql.DataSource | org.apache.derby.jdbc.ClientDataSource | |
| <input type="checkbox"/> | K19 | javax.sql.ConnectionPoolDataSource | com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource | |
| <input type="checkbox"/> | TimerPool | javax.sql.XADatasource | org.apache.derby.jdbc.EmbeddedXADatasource | |

The screenshot shows the GlassFish administration interface. On the left, there's a navigation tree with various nodes like Domain, Clusters, Standalone Instances, Nodes, Applications, Lifecycle Modules, Monitoring Data, Resources, JDBC, JNDI, JavaMail Sessions, Resource Adapter Configs, Configurations, default-config, and server-config. Under the JDBC node, 'JDBC Resources' and 'JDBC Connection Pools' are listed, with 'K19' selected. The main panel is titled 'Edit JDBC Connection Pool' and contains tabs for General, Advanced, and Additional Properties. The General tab is active, showing a yellow box with a green checkmark and the text 'Ping Succeeded'. Below this, it says 'Modify an existing JDBC connection pool. A JDBC connection pool is a group of reusable connections for a particular database.' There are buttons for Load Defaults, Flush, and Ping (which has a blue arrow pointing to it). Other fields include Pool Name (K19), Resource Type (javax.sql.ConnectionPoolDataSource), Datasource Classname (com.mysql.jdbc.jdbc2.optional.MysqlConnectionPoolDataSource), Driver Classname (empty), Ping (Enabled checked), Deployment Order (100), and Description (empty). In the Pool Settings section, Initial and Minimum Pool Size is set to 8. A note at the bottom states 'Minimum and initial number of connections maintained in the pool'.

OBS: A mensagem “Ping Succeeded” deve aparecer.

Agora vamos criar o Data Source. Siga os passos abaixo:

JDBC Resources

JDBC resources provide applications with a means to connect to a database.

| Select | JNDI Name | Logical JNDI Name | Enabled | Connection Pool | Description |
|--------------------------|-----------------|-----------------------------|-------------------------------------|-----------------|-------------|
| <input type="checkbox"/> | jdbc/_TimerPool | | <input checked="" type="checkbox"/> | _TimerPool | |
| <input type="checkbox"/> | jdbc/_default | java:comp/DefaultDataSource | <input checked="" type="checkbox"/> | DerbyPool | |

New JDBC Resource

Specify a unique JNDI name that identifies the JDBC resource you want to create. The name must contain only alphanumeric, underscore, dash, or dot characters.

JNDI Name: * 1

Pool Name: 2

Description:

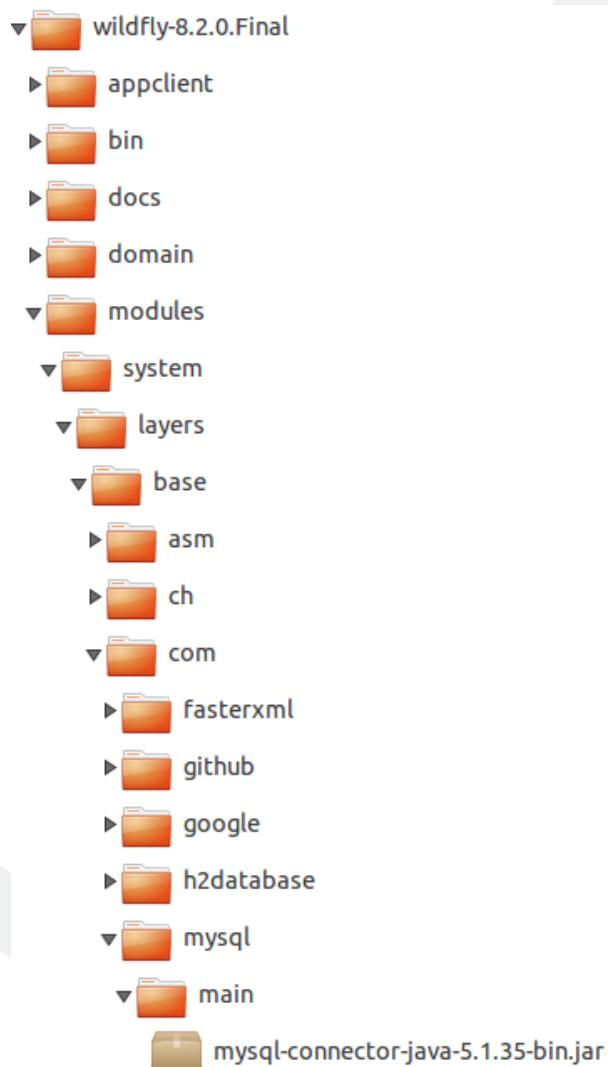
Status: Enabled

Additional Properties (0)

Add Property Delete Properties

| Select | Name | Value | Description |
|-----------------|------|-------|-------------|
| No items found. | | | |

- 4** Agora, vamos configurar um Data Source no Wildfly. Crie uma pasta chamada **mysql** dentro da pasta **modules/system/layers/base/com** do Wildfly. Dentro da pasta **mysql** crie uma pasta chamada **main**. Copie o arquivo **mysql-connector-java-VERSAO-bin.jar** que se encontra na pasta **K19-Arquivos/mysql-connector-java-VERSAO** da Área de Trabalho para a pasta **main**.



Importante

Você também pode obter o driver JDBC do MySQL através do site da K19: www.k19.com.br/arquivos.

- 5** Na pasta **modules/system/layers/base/com/mysql/main** do Wildfly, crie um arquivo chamado **module.xml**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <module xmlns="urn:jboss:module:1.0" name="com.mysql">
3   <resources>
4     <resource-root path="mysql-connector-java-5.1.35-bin.jar"/>
5   </resources>
6   <dependencies>
  
```

```

7 <module name="javax.api"/>
8 </dependencies>
9 </module>

```

Código XML 5.1: module.xml

Na linha 4, indique corretamente o nome correspondente ao arquivo do driver que você está utilizando.

6 Acesse o MySQL Server através do MySQL Workbench ou através do cliente de linha de comando; apague a base de dados **k22_wildfly** caso ela exista; crie uma base de dados chamada **k22_wildfly**.

7 Altere o arquivo de configuração do Wildfly (**standalone/configuration/standalone.xml**). Adicione dentro da tag **<datadources>** a configuração de um novo Data Source.

```

1 ...
2 <datasource jndi-name="java:/jdbc/K19" pool-name="K19" jta="true"
3   enabled="true" use-java-context="true" use-ccm="true">
4   <connection-url>jdbc:mysql://localhost:3306/k22_wildfly</connection-url>
5   <driver>com.mysql</driver>
6   <security>
7     <user-name>root</user-name>
8     <password>root</password>
9   </security>
10  </datasource>
11 ...

```

Código XML 5.2: standalone.xml

Depois, adicione dentro da tag **<drivers>** a seguinte configuração.

```

1 ...
2 <driver name="com.mysql" module="com.mysql">
3   <xa-datasource-class>
4     com.mysql.jdbc.optional.MysqlXADataSource
5   </xa-datasource-class>
6   <driver-class>com.mysql.jdbc.Driver</driver-class>
7 </driver>
8 ...

```

Código XML 5.3: standalone.xml

8 Pare o **Glassfish**. Inicie o **Wildfly** e observe se o Data Source foi criado corretamente. No console, deve aparecer uma mensagem semelhante a que é apresentada abaixo.

Bound data source [java:/jdbc/K19]



persistence.xml

Em um ambiente Java EE, diversas configurações relativas à persistência são realizadas nos Data Sources. Contudo, algumas configurações ainda devem ser realizadas pelas aplicações. A especificação JPA determina que cada aplicação contenha um arquivo de configurações chamado **persistence.xml** dentro de uma pasta chamada **META-INF** no classpath da aplicação.

No arquivo `persistece.xml` podemos definir qual Data Source será utilizado pela aplicação.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <provider>org.hibernate.ejb.HibernatePersistence</provider>
10    <jta-data-source>jdbc/K19</jta-data-source>
11
12    <properties>
13      <property name="hibernate.show_sql" value="true" />
14      <property name="hibernate.format_sql" value="true" />
15      <property name="hibernate.hbm2ddl.auto" value="update" />
16      <property name="hibernate.dialect" value="org.hibernate.dialect.←
17        MySQL5InnoDBDialect"/>
18    </properties>
19  </persistence-unit>
20</persistence>

```

Código XML 5.4: persistece.xml



Entity Beans

As regras de negócio de uma aplicação EJB são implementadas nos session beans. Por outro lado, os dados da aplicação que devem ser persistidos são armazenados em objetos chamados **Entity Beans**. São exemplos de entity beans que poderiam formar uma aplicação:

- clientes
- produtos
- pedidos
- funcionários
- fornecedores



Entity Classes e Mapeamento

Os Entity Beans são definidos por classes java (Entity Classes). As Entity Classes devem ser mapeadas para tabelas no banco de dados através de anotações ou XML. As principais anotações de mapeamento são:

@Entity É a principal anotação do JPA. Ela que deve aparecer antes do nome de uma classe. E deve ser definida em todas as classes que terão objetos persistidos no banco de dados.

As classes anotadas com `@Entity` são mapeadas para tabelas. Por convenção, as tabelas possuem os mesmos nomes das classes. Mas, podemos alterar esse comportamento utilizando a anotação `@Table`.

Os atributos declarados em uma classe anotada com `@Entity` são mapeados para colunas na tabela correspondente à classe. Outra vez, por convenção, as colunas possuem os mesmos no-

mes dos atributos. E novamente, podemos alterar esse padrão utilizando para isso a anotação `@Column`.

@Id Utilizada para indicar qual atributo de uma classe anotada com `@Entity` será mapeado para a chave primária da tabela correspondente à classe. Geralmente o atributo anotado com `@Id` é do tipo `Long`.

@GeneratedValue Geralmente vem acompanhado da anotação `@Id`. Serve para indicar que o valor de um atributo que compõe uma chave primária deve ser gerado pelo banco no momento em que um novo registro é inserido.

Supondo uma aplicação que administra livros e autores. As seguintes classes são exemplos de Entity Classes mapeadas com anotações que poderiam ser utilizadas no contexto dessa aplicação:

```

1 @Entity
2 public class Livro {
3
4     @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
5     private Long id;
6
7     private String nome;
8
9     private Double preco;
10
11    // GETTERS AND SETTERS
12 }
```

Código Java 5.1: *Livro.java*

```

1 @Entity
2 public class Autor {
3
4     @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
5     private Long id;
6
7     private String nome;
8
9     @ManyToMany
10    private List<Livro> livros;
11
12    // GETTERS AND SETTERS
13 }
```

Código Java 5.2: *Autor.java*

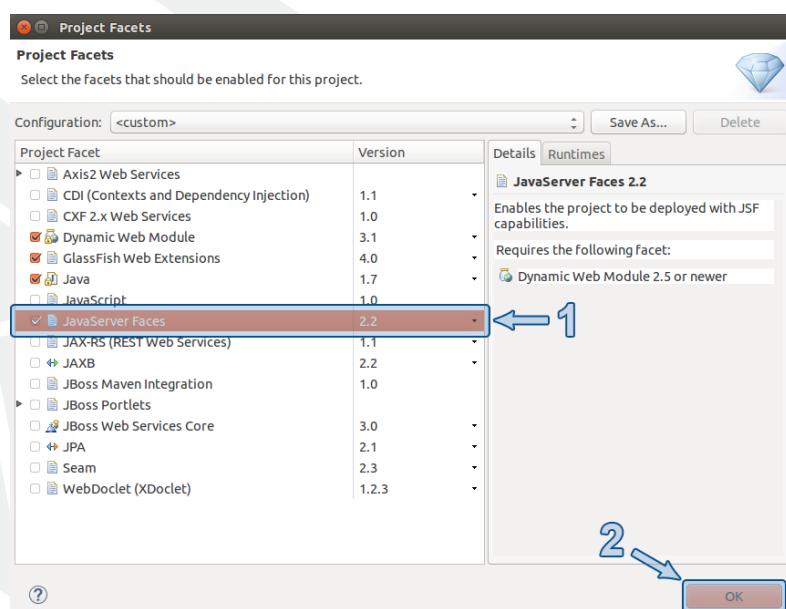
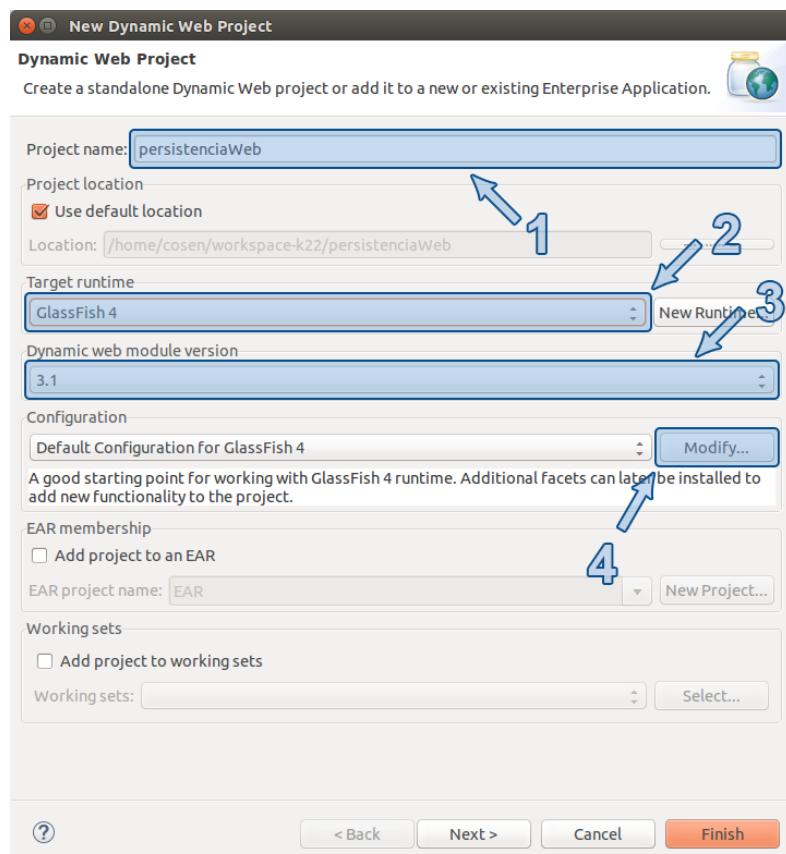
Consulte a apostila do curso “K21 - Persistência com JPA 2 e Hibernate” para obter detalhes sobre o mapeamento das Entity Classes <http://www.k19.com.br/downloads/apostilas-java>.

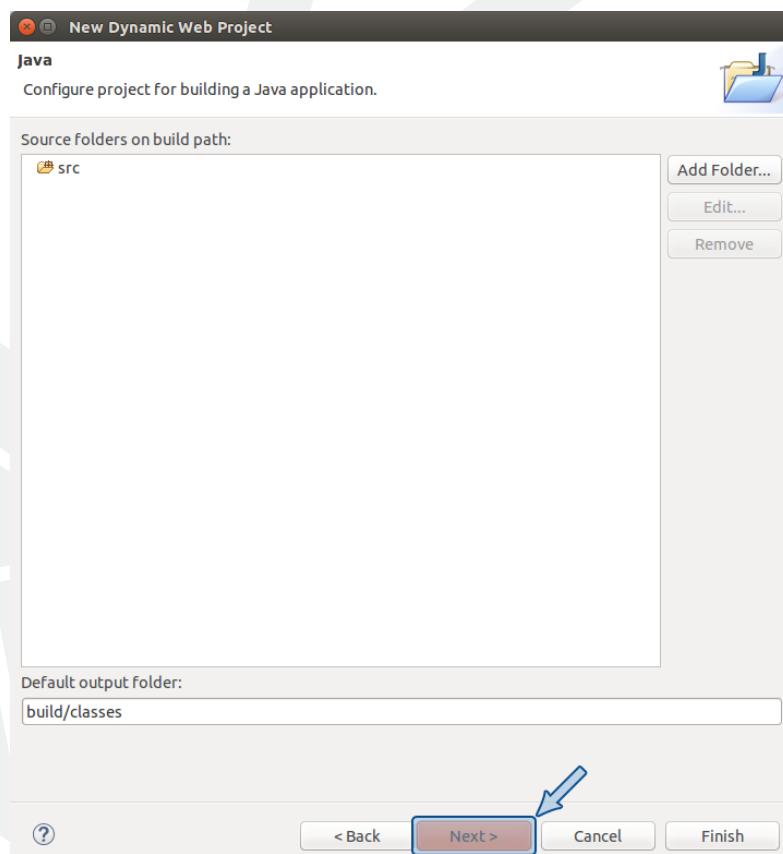
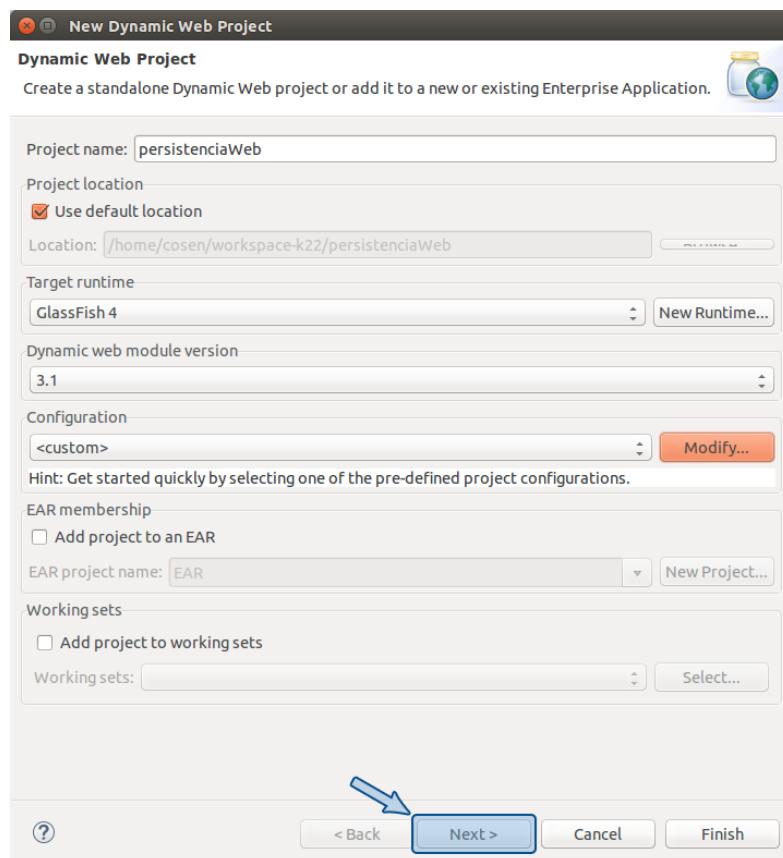


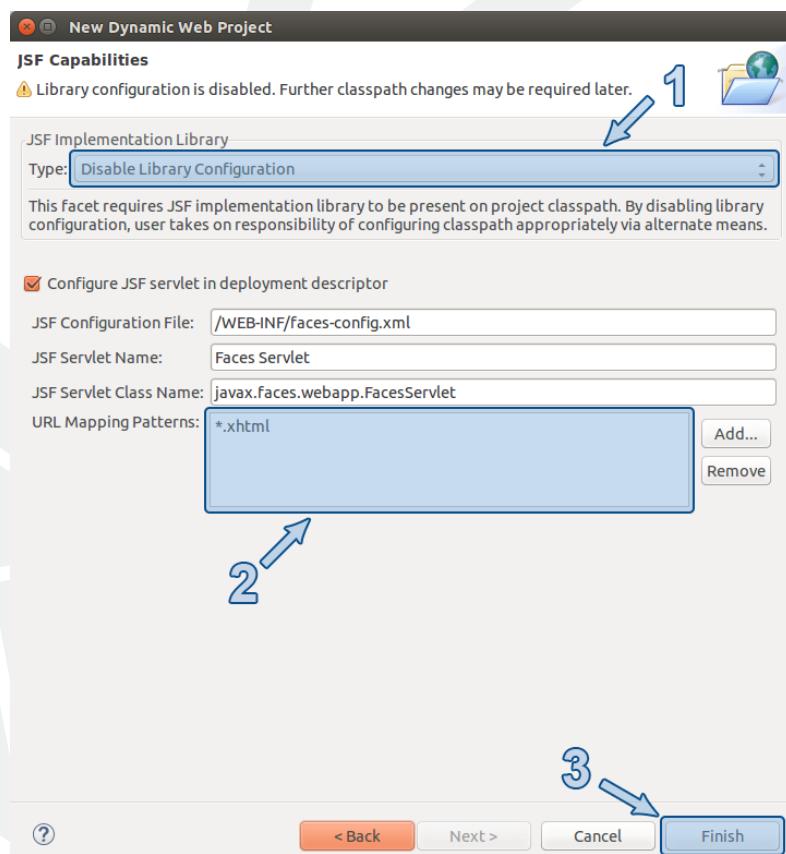
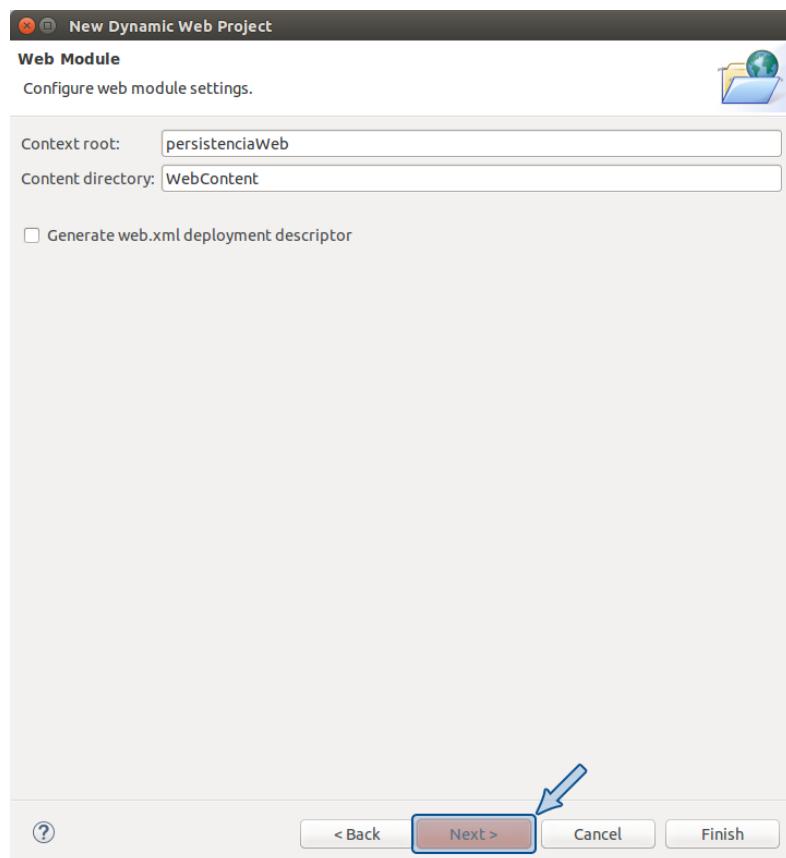
Exercícios de Fixação

- 9 Para não confundir, feche os projetos **chatWeb** e **contadorJavaSE**. Para isso, clique com o botão direito do mouse sobre esses projetos e selecione a opção “Close Project”.

- 10 Crie um Dynamic Web Project no eclipse chamado **persistenciaWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.







- 11 Adicione uma pasta chamada **META-INF** na pasta **src** do projeto **persistenciaWeb**.

- 12 Faça as configurações de persistência adicionando o arquivo **persistence.xml** na pasta **META-INF**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11   <properties>
12     <property
13       name="javax.persistence.schema-generation.database.action"
14       value="create"/>
15   </properties>
16   </persistence-unit>
17 </persistence>
```

Código XML 5.5: *persistence.xml*

- 13 Crie um pacote chamado **br.com.k19.entidades** no projeto **persistenciaWeb** e adicione nesse pacote uma Entity Class para definir os livros de uma editora.

```

1 package br.com.k19.entidades;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.GenerationType;
6 import javax.persistence.Id;
7
8 @Entity
9 public class Livro {
10
11   @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
12   private Long id;
13
14   private String nome;
15
16   private Double preco;
17
18   // GETTERS AND SETTERS
19 }
```

Código Java 5.3: *Livro.java*

- 14 Adicione no pacote **br.com.k19.entidades** uma Entity Class para definir autores dos livros de uma editora.

```

1 package br.com.k19.entidades;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.persistence.Entity;
7 import javax.persistence.GeneratedValue;
8 import javax.persistence.GenerationType;
```

```

9 import javax.persistence.Id;
10 import javax.persistence.ManyToMany;
11
12 @Entity
13 public class Autor {
14
15     @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
16     private Long id;
17
18     private String nome;
19
20     @ManyToMany
21     private List<Livro> livros = new ArrayList<Livro>();
22
23     // GETTERS AND SETTERS
24 }
```

Código Java 5.4: Autor.java



Entity Managers

Os Entity Managers são objetos que administram os Entity Beans. As principais responsabilidades dos Entity Managers são:

- Recuperar as informações armazenadas no banco de dados.
- Montar Entity Beans com os dados obtidos do banco de dados através de consultas.
- Sincronizar o conteúdo dos Entity Beans com os registros das tabelas do banco de dados.

Consulte a apostila do curso “Persistência com JPA 2” para obter detalhes sobre o funcionamento dos Entity Managers <http://online.k19.com.br>.

Obtendo Entity Managers

Em um ambiente Java SE, o controle da criação e do fechamento dos Entity Managers é responsabilidade das aplicações. Uma aplicação Java SE deve chamar, explicitamente, o método `createEntityManager()` em uma Entity Manager Factory para obter um novo Entity Manager ou o método `close()` em um Entity Manager para fechá-lo.

```
1 EntityManager manager = factory.createEntityManager();
```

```
1 manager.close();
```

Por outro lado, em um ambiente Java EE, o gerenciamento dos Entity Managers pode ser atribuído ao servidor de aplicação. Nesse caso, para uma aplicação Java EE obter um Entity Manager, ela pode utilizar o recurso de Injeção de Dependência oferecido pelo servidor de aplicação. Por exemplo, dentro de um Session Bean, podemos pedir a injeção de um Entity Manager através da anotação `@PersistenceContext`.

```

1 @Stateless
2 public class CalculadoraBean {
3
4     @PersistenceContext
5     private EntityManager manager;
6 }
```

```
7 // Resto do código  
8 }
```

Código Java 5.7: CalculadoraBean.java



Entity Manager Factories

As Entity Managers Factories são objetos responsáveis pela criação de Entity Managers de acordo com as configurações definidas no arquivo `persistence.xml`, nos Data Sources e através das anotações de mapeamento nas Entity Classes.

Obtendo Entity Manager Factories

Em um ambiente Java SE, uma Entity Manager Factory é obtida através do método estático `createEntityManagerFactory()` da classe **Persistence**.

```
1 EntityManagerFactory factory = Persistence.createEntityManagerFactory("unidade");
```

O método `createEntityManagerFactory()` deve ser chamado apenas uma vez a cada execução da aplicação. Não é necessário chamá-lo mais do que uma vez porque a aplicação não necessita de mais do que uma Entity Manager Factory e o custo de criação desse objeto é alto.

Por outro lado, em um ambiente Java EE, o controle sobre a criação das Entity Manager Factories é responsabilidade do servidor de aplicação. Inclusive, o servidor de aplicação evita a criação de fábricas desnecessárias.

Se uma aplicação Java EE deseja obter a Entity Manager Factory criada pelo servidor de aplicação, ela deve utilizar a anotação `@PersistenceUnit` para pedir a injeção desse objeto.

```
1 @Stateless  
2 public class CalculadoraBean {  
3  
4     @PersistenceUnit  
5     private EntityManagerFactory factory;  
6  
7     // Resto do código  
8 }
```

Código Java 5.9: CalculadoraBean.java

Em geral, as aplicações Java EE não necessitam interagir diretamente com as Entity Manager Factories. Na verdade, o comum é utilizar diretamente os Entity Managers que são obtidos com a anotação `@PersistenceContext`.

```
1 @Stateless  
2 public class CalculadoraBean {  
3  
4     @PersistenceContext  
5     private EntityManager manager;  
6  
7     // Resto do código  
8 }
```

Código Java 5.10: CalculadoraBean.java



Exercícios de Fixação

- 15** Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **persistenciaWeb** e adicione nesse pacote um SLSB para funcionar como repositório de livros.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.ejb.Stateless;
6 import javax.persistence.EntityManager;
7 import javax.persistence.PersistenceContext;
8 import javax.persistence.TypedQuery;
9
10 import br.com.k19.entidades.Livro;
11
12 @Stateless
13 public class LivroRepositorio {
14
15     @PersistenceContext
16     private EntityManager manager;
17
18     public void adiciona(Livro livro) {
19         this.manager.persist(livro);
20     }
21
22     public List<Livro> getLivros() {
23         TypedQuery<Livro> query = this.manager.createQuery(
24             "select x from Livro x", Livro.class);
25
26         return query.getResultList();
27     }
28 }
```

Código Java 5.11: *LivroRepositorio.java*

- 16** Crie um pacote chamado **br.com.k19.managedbeans** no projeto **persistenciaWeb** e adicione nesse pacote um Managed Bean para oferecer algumas ações para as telas.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.ejb.EJB;
6 import javax.faces.bean.ManagedBean;
7
8 import br.com.k19.entidades.Livro;
9 import br.com.k19.sessionbeans.LivroRepositorio;
10
11 @ManagedBean
12 public class LivroMB {
13
14     @EJB
15     private LivroRepositorio repositorio;
16
17     private Livro livro = new Livro();
18
19     private List<Livro> livrosCache;
20
21     public void adiciona() {
22         this.repositorio.adiciona(this.livro);
```

```

23     this.livro = new Livro();
24     this.livrosCache = null;
25 }
26
27 public List<Livro> getLivros() {
28     if (this.livrosCache == null) {
29         this.livrosCache = this.repositorio.getLivros();
30     }
31     return this.livrosCache;
32 }
33
34 public Livro getLivro() {
35     return livro;
36 }
37
38 public void setLivro(Livro livro) {
39     this.livro = livro;
40 }
41 }
```

Código Java 5.12: LivroMB.java

- 17** Crie uma tela para cadastrar livros. Adicione na pasta **WebContent** um arquivo chamado **livros.xhtml** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:ui="http://java.sun.com/jsf/facelets"
6      xmlns:h="http://java.sun.com/jsf/html"
7      xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10    <title>Livros</title>
11 </h:head>
12
13 <h:body>
14    <h1>Novo Livro</h1>
15    <h:messages/>
16    <h:form>
17        <h:outputLabel value="Nome: "/>
18        <h:inputText value="#{livroMB.livro.nome}" />
19
20        <h:outputLabel value="Preço: "/>
21        <h:inputText value="#{livroMB.livro.preco}" />
22
23        <h:commandButton action="#{livroMB.adiciona}" value="Salvar"/>
24    </h:form>
25
26    <h1>Lista de Livros</h1>
27    <h:dataTable value="#{livroMB.livros}" var="livro">
28        <h:column>
29            <h:outputText value="#{livro.nome}" />
30        </h:column>
31        <h:column>
32            <h:outputText value="#{livro.preco}" />
33        </h:column>
34    </h:dataTable>
35 </h:body>
36 </html>
```

Código XHTML 5.1: livros.xhtml

- 18 Remova o projeto **chatWeb** do **Glassfish**. Adicione o projeto **persistenciaWeb** no **Glassfish**. Certifique-se que o **JBoss** e o **Wildfly** estejam parados. Inicie o **Glassfish** e teste a aplicação acessando a url <http://localhost:8080/persistenciaWeb/livros.xhtml>.
- 19 Acesse o MySQL Server através do MySQL Workbench ou através do cliente de linha de comando; Verifique as tabelas geradas na base de dados **k22_glassfish** e seus respectivos dados.
- 20 Remova o projeto **persistenciaWeb** do **Glassfish**. Adicione o projeto **persistenciaWeb** no **Wildfly**. Certifique-se que o **Glassfish** e o **JBoss** estejam parados. Inicie o **Wildfly** e teste a aplicação acessando a url <http://localhost:8080/persistenciaWeb/livros.xhtml>.
- 21 Acesse o MySQL Server através do MySQL Workbench ou através do cliente de linha de comando; Verifique as tabelas geradas na base de dados **k22_wildfly** e seus respectivos dados.



TRANSAÇÕES

Geralmente, uma aplicação realiza diversas tarefas diferentes. Também é comum e muitas vezes necessário dividir as tarefa em pequenos passos. Daí surge o conceito de transação. Uma transação é um conjunto de passos que devem ser executados em uma ordem específica para que uma determinada tarefa seja realizada. Tipicamente, as transações modificam informações armazenadas em **resources** (bases de dados, filas de mensagens, sistemas corporativos de informação - EIS, entre outros).



ACID

Além da restrição natural de ordem, as transações possuem outras quatro propriedades fundamentais: Atomicidade, Consistência, Isolamento e Durabilidade. A sigla ACID é utilizada para indicar a existência dessas propriedades.

Atomicidade: Todos os passos de uma transação devem ser executados com sucesso para que a própria transação seja executada com sucesso. Se algum passo falhar a transação falhará e todos os passos realizados até o momento da falha serão desfeitos.

Consistência: Não pode existir inconsistência nos dados da aplicação nem antes nem depois da execução de uma transação. Ou seja, uma transação leva a aplicação de um estado consistente para outro estado consistente.

Isolamento: Alterações realizadas por uma transação não finalizada não podem afetar operações que não fazem parte da transação.

Durabilidade: Após a confirmação de uma transação, as modificações realizadas por ela devem ser refletidas nos resources mesmo que aconteça uma falha de hardware.



Transação Local ou Distribuída

Quando as alterações realizadas por uma transação afetam apenas um resource (bases de dados, filas de mensagens, sistemas corporativos de informação - EIS, entre outros), dizemos que a transação é local. Caso contrário, se dois ou mais resources são modificados por uma única transação ela é dita distribuída.



JTA e JTS

Todo servidor de aplicação Java EE deve oferecer suporte para as aplicações utilizarem transações. As especificações relacionadas a esse tópico são: Java Transaction API - JTA e Java Transaction Service - JTS. Os documentos dessas especificações podem ser obtidos através do site: www.jcp.org.

A especificação Enterprise Java Beans (EJB) é fortemente integrada com as especificações JTA e JTS, simplificando bastante o trabalho dos desenvolvedores de aplicação EJB que não precisam em momento nenhum lidar diretamente com JTS e muito pouco com JTA.

Na arquitetura EJB, as aplicações podem gerenciar as transações de dois modos:

- Container Managed Transactions - CMT
- Bean Managed Transactions - BMT



Container Managed Transactions - CMT

Quando optamos pelo gerenciamento CMT, a responsabilidade de abrir, confirmar ou abortar transações é atribuída ao EJB Container. Contudo, a aplicação deve indicar ao EJB Container através de configurações quando ele deve abrir, confirmar ou abortar transações.

Podemos definir o modo CMT individualmente para cada Session Bean da nossa aplicação através da anotação **@TransactionManagement**.

```
1 @Stateful
2 @TransactionManagement(TransactionManagementType.CONTAINER)
3 public class CarrinhoBean {
4     ...
5 }
```

O modo CMT é padrão. Dessa forma, tecnicamente, não é necessário acrescentar a anotação **@TransactionManagement**. Mas, podemos adicioná-la com o intuito de explicitar a opção de gerenciamento transacional.

Atributo Transacional

O EJB Container abre, confirma, aborta ou suspende transações de acordo com o atributo transacional de cada método dos Session Beans em modo CMT. O atributo transacional de um método pode ser definido com um dos seguintes valores: **REQUIRED**, **REQUIRES_NEW**, **SUPPORTS**, **MANDATORY**, **NOT_SUPPORTED** e **NEVER**.

O comportamento do EJB Container é o seguinte:

| Atributo Transacional | Já existia uma transação aberta? | O que o EJB Container faz? |
|-----------------------|----------------------------------|---|
| REQUIRED | NÃO | Abre uma nova transação |
| REQUIRED | SIM | Usa a transação que já estava aberta |
| REQUIRES_NEW | NÃO | Abre uma nova transação |
| REQUIRES_NEW | SIM | Abre uma transação e Suspende a que estava aberta |
| SUPPORTS | NÃO | Não faz nada |
| SUPPORTS | SIM | Usa a transação que já estava aberta |
| MANDATORY | NÃO | Lança EJBTransactionRequiredException |
| MANDATORY | SIM | Usa a transação que já estava aberta |
| NOT_SUPPORTED | NÃO | Não faz nada |
| NOT_SUPPORTED | SIM | Suspende a que estava aberta |
| NEVER | NÃO | Não faz nada |
| NEVER | SIM | Lança EJBException |

O atributo transacional de um método pode ser definido pela anotação **@TransactionAttribute**.

```

1 @TransactionAttribute(TransactionAttributeType.REQUIRED)
2 public void adiciona(String produto){
3     ...
4 }
```

Quando queremos que todos os métodos de um Session Bean possuam o mesmo atributo transacional, devemos anotar a classe com **@TransactionAttribute**.

```

1 @Stateful
2 @TransactionManagement(TransactionManagementType.CONTAINER)
3 @TransactionAttribute(TransactionAttributeType.REQUIRED)
4 public class CarrinhoBean {
5     ...
6 }
```

Caso nenhum atributo transacional seja definido explicitamente, o EJB Container utilizará por padrão o **REQUIRED**.

Rollback com SessionContext

Quando algum erro é identificado pela aplicação, ela pode marcar a transação corrente para rollback através do `setRollbackOnly()` do Session Context que pode ser obtido através de injeção com a anotação `@Resource`.

```

1 @Stateful
2 public class CarrinhoBean {
3
4     @Resource
5     private SessionContext context;
6
7     public void adiciona(String produto){
8         if(produto == null){
```

```

9     context.setRollbackOnly();
10    }
11    ...
12  }
13 }
```

Código Java 6.4: CarrinhoBean.java

Rollback com Exceptions

Quando exceptions ocorrem, transações podem ser abortadas pelo EJB Container. Devemos entender a classificação das exceptions para saber quando as transações serão abortadas.

Na arquitetura EJB, as exceptions são classificadas em dois grupos:

System Exceptions: Todas **Unchecked Exceptions** e as `java.rmi.RemoteException`, por padrão, são consideradas System Exceptions.

Application Exceptions: Todas **Checked Exceptions** exceto as `java.rmi.RemoteException`, por padrão, são consideradas Application Exceptions.

Por padrão, quando um método de um Session Bean lança uma System Exception, o EJB Container aborta a transação corrente. Por outro lado, quando uma Application Exception é lançada, o EJB Container não aborta a transação corrente.

Podemos utilizar a anotação `@ApplicationException` para alterar a classificação de uma System Exception.

```

1 @ApplicationException
2 public class ValorNegativoException extends RuntimeException {
3
4 }
```

Código Java 6.5: ValorNegativoException.java

A mesma anotação pode alterar o comportamento padrão para rollback das Application Exceptions.

```

1 @ApplicationException(rollback=true)
2 public class ValorNegativoException extends RuntimeException {
3
4 }
```

Código Java 6.6: ValorNegativoException.java



Bean Managed Transactions - BMT

Quando optamos pelo gerenciamento BMT, a responsabilidade de abrir, confirmar ou abortar transações é atribuída a aplicação. Podemos definir o modo BMT individualmente para cada Session Bean da nossa aplicação através da anotação `@TransactionManagement`.

```
1 @Stateful
```

```

2 @TransactionManagement(TransactionManagementType.BEAN)
3 public class CarrinhoBean {
4 ...
5 }
```

Código Java 6.7: CarrinhoBean.java

No modo BMT, devemos injetar um **UserTransaction** através da anotação `@Resource`. Esse objeto permite que a aplicação abra, confirme ou aborte transações.

```

1 @Stateful
2 @TransactionManagement(TransactionManagementType.BEAN)
3 public class CarrinhoBean {
4     @Resource
5     private UserTransaction ut;
6
7     public void adiciona(Produto p) {
8         try {
9             ut.begin();
10            // IMPLEMENTACAO
11            ut.commit();
12        } catch(ProdutoInvalidoException e) {
13            ut.rollback();
14        } catch (Exception e) {
15            e.printStackTrace();
16        }
17    }
18 }
```

Código Java 6.8: CarrinhoBean.java

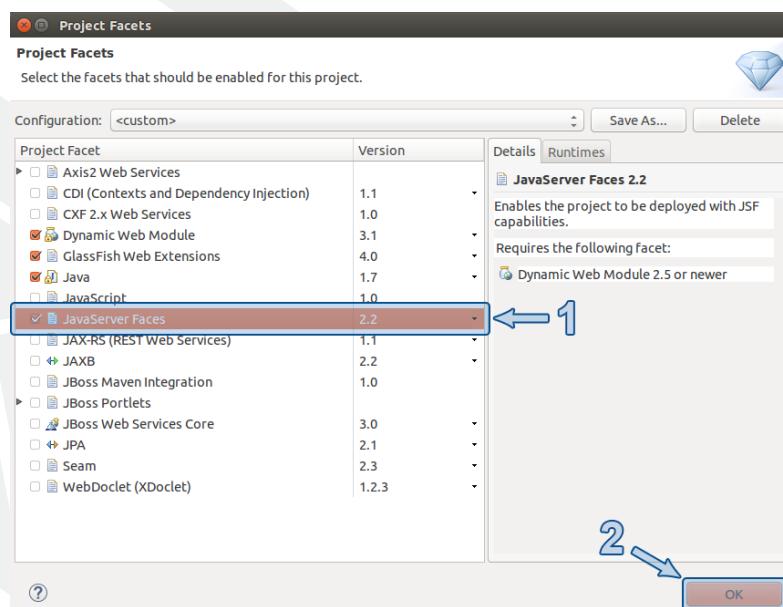
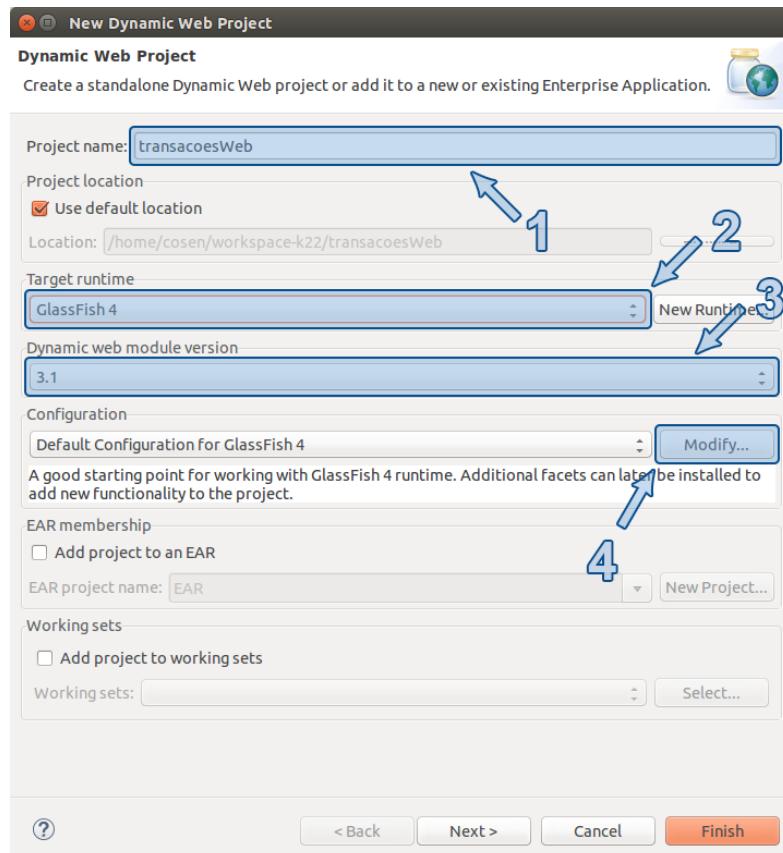
O modo BMT permite um controle maior sobre as transações. Contudo, o modo CMT é mais simples de utilizar e mais fácil de manter.

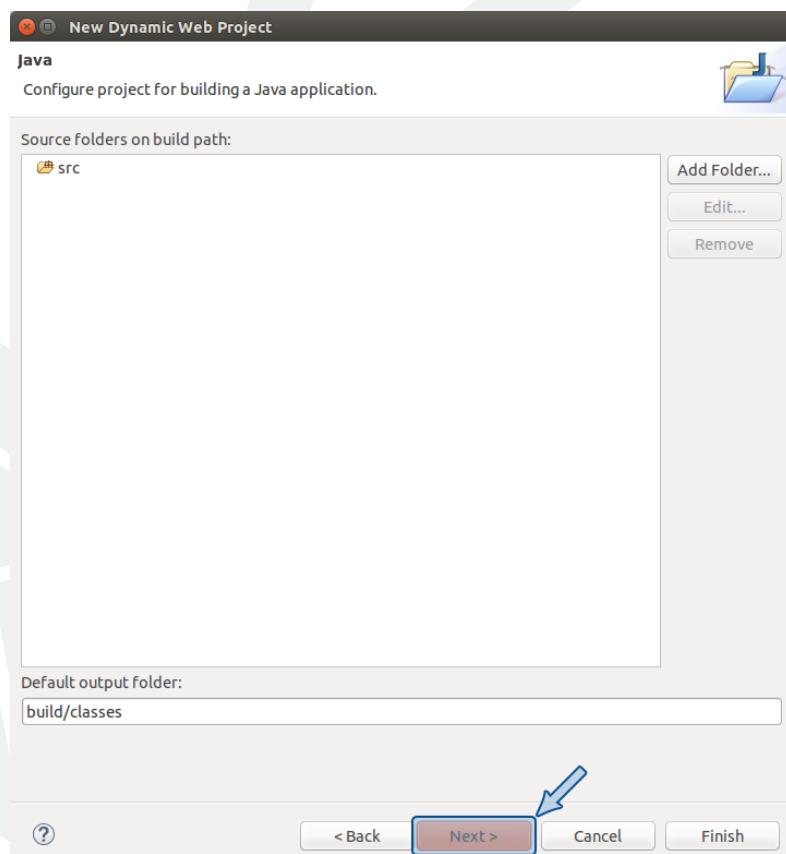
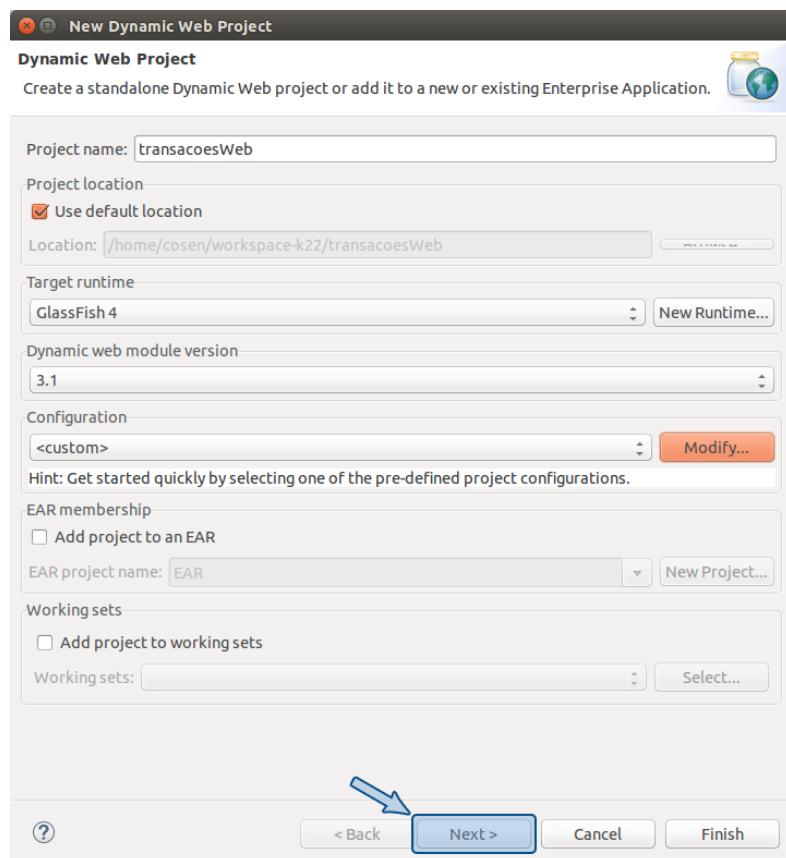


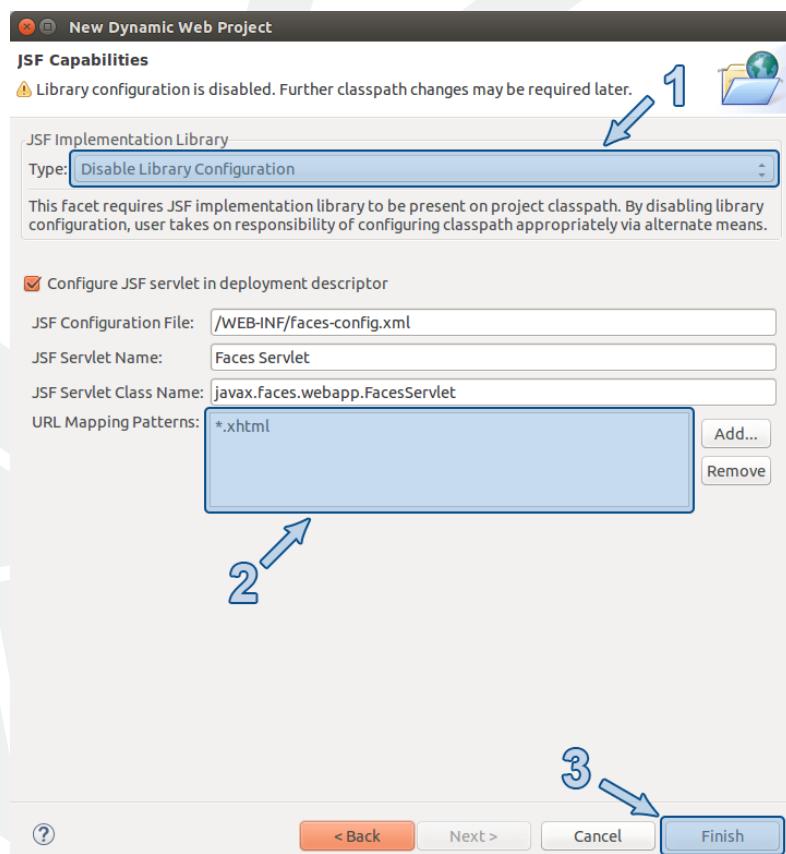
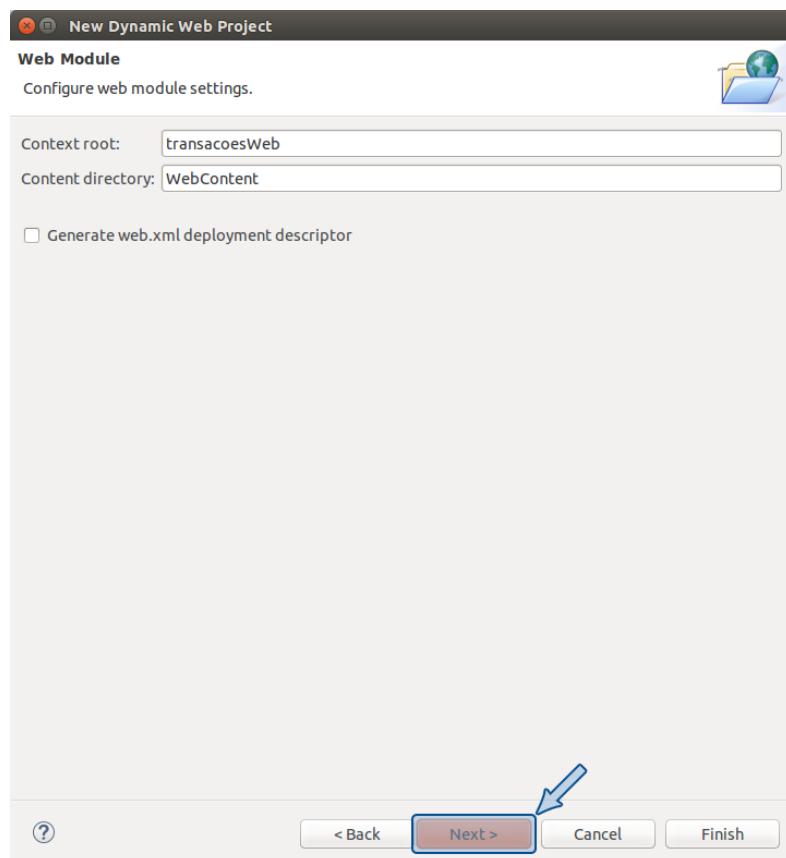
Exercícios de Fixação

- 1 Para não confundir, feche o projeto **persistenciaWeb**. Para isso, clique com o botão direito do mouse sobre esse projeto e selecione a opção “Close Project”.

- 2 Crie um Dynamic Web Project no eclipse chamado **transacoesWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.







- 3** Adicione uma pasta chamada **META-INF** na pasta **src** do projeto **transacoesWeb**.
- 4** Faça as configurações de persistência adicionando o arquivo **persistence.xml** na pasta **META-INF**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="create"/>
15    </properties>
16  </persistence-unit>
17</persistence>
```

Código XML 6.1: *persistence.xml*

- 5** Crie um pacote chamado **br.com.k19.entidades** no projeto **transacoesWeb** e adicione nesse pacote uma Entity Class para definir os produtos de uma loja.

```

1 package br.com.k19.entidades;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.GenerationType;
6 import javax.persistence.Id;
7
8 @Entity
9 public class Produto {
10
11   @Id @GeneratedValue(strategy=GenerationType.IDENTITY)
12   private Long id;
13
14   private String nome;
15
16   private double preco;
17
18   // GETTERS AND SETTERS
19 }
```

Código Java 6.9: *Produto.java*

- 6** Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **transacoesWeb** e adicione nesse pacote um SLSB para funcionar como repositório de produtos.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.annotation.Resource;
6 import javax.ejb.SessionContext;
7 import javax.ejb.Stateless;
8 import javax.persistence.EntityManager;
```

```

9 import javax.persistence.PersistenceContext;
10 import javax.persistence.TypedQuery;
11
12 import br.com.k19.entidades.Produto;
13
14 @Stateless
15 public class ProdutoRepositorio {
16
17     @PersistenceContext
18     private EntityManager manager;
19
20     @Resource
21     private SessionContext context;
22
23     public void adiciona(Produto produto) {
24         this.manager.persist(produto);
25         if (produto.getPreco() < 0) {
26             this.context.setRollbackOnly();
27         }
28     }
29
30     public List<Produto> getProdutos() {
31         TypedQuery<Produto> query = this.manager.createQuery(
32             "select x from Produto x", Produto.class);
33
34         return query.getResultList();
35     }
36 }
```

Código Java 6.10: ProdutoRepositorio.java

- 7 Crie um pacote chamado **br.com.k19.managedbeans** no projeto **transacoesWeb** e adicione nesse pacote um Managed Bean para oferecer algumas ações para as telas.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.ejb.EJB;
6 import javax.faces.bean.ManagedBean;
7
8 import br.com.k19.entidades.Produto;
9 import br.com.k19.sessionbeans.ProdutoRepositorio;
10
11 @ManagedBean
12 public class ProdutoMB {
13
14     @EJB
15     private ProdutoRepositorio repositorio;
16
17     private Produto produto = new Produto();
18
19     private List<Produto> produtosCache;
20
21     public void adiciona() {
22         this.repositorio.adiciona(this.produto);
23         this.produto = new Produto();
24         this.produtosCache = null;
25     }
26
27     public List<Produto> getProdutos() {
28         if (this.produtosCache == null) {
29             this.produtosCache = this.repositorio.getProdutos();
30         }
31         return this.produtosCache;
32     }
33 }
```

```

34     public void setProduto(Produto produto) {
35         this.produto = produto;
36     }
37
38     public Produto getProduto() {
39         return produto;
40     }
41 }
```

Código Java 6.11: ProdutoMB.java

- 8** Adicione um arquivo chamado **produtos.xhtml** na pasta **WebContent** do projeto **transacoesWeb**.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4  <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9  <h:head>
10    <title>Produtos</title>
11  </h:head>
12
13 <h:body>
14   <h1>Novo Produto</h1>
15   <h:form>
16     <h:outputLabel value="Nome: "/>
17     <h:inputText value="#{produtoMB.produto.nome}" />
18
19     <h:outputLabel value="Preço: "/>
20     <h:inputText value="#{produtoMB.produto.preco}" />
21
22     <h:commandButton action="#{produtoMB.adiciona}" value="Salvar"/>
23   </h:form>
24
25   <h1>Lista de Produtos</h1>
26   <h:dataTable value="#{produtoMB.produtos}" var="produto">
27     <h:column>
28       <h:outputText value="#{produto.nome}" />
29     </h:column>
30     <h:column>
31       <h:outputText value="#{produto.preco}" />
32     </h:column>
33   </h:dataTable>
34 </h:body>
35 </html>
```

Código XHTML 6.1: produtos.xhtml

- 9** Remova o projeto **persistenciaWeb** do **Wildfly**. Adicione o projeto **transacoesWeb** no **Glassfish**. Certifique-se que o **JBoss** e o **Wildfly** estejam parados. Inicie o **Glassfish** e teste a aplicação acessando a url <http://localhost:8080/transacoesWeb/produtos.xhtml>.

- 10** Adicione o projeto **transacoesWeb** no **glassfish**. Clique com o botão direito no **glassfish** da view **Servers** e escolha a opção “Add and Remove”.

Adicione alguns produtos e observe que produtos com preço negativo não são persistidos devido

ao rollback.



SEGURANÇA

Para muitas aplicações, a segurança é um aspecto obrigatório. Da segurança podemos extrair dois processos fundamentais: **Autenticação** e **Autorização**.

O processo de autenticação consiste na identificação dos usuários através de algum tipo de certificado (usuário e senha). Já o processo de autorização determina o que cada usuário autenticado pode acessar dentro da aplicação.

Na plataforma Java, esses dois processos são padronizados pela especificação JAAS (Java Authentication and Authorization Service).



Realms

Em um ambiente Java EE, para realizar o processo de autenticação, devemos criar um ou mais **Realms**. Um Realm é uma base de dados na qual os usuários de uma ou mais aplicações estão cadastrados.

Infelizmente, as configurações necessárias para criar um Realm não são padronizadas, ou seja, cada servidor de aplicação as realiza da sua própria maneira. Veremos no exercício como criar um Realm no Glassfish.



Exercícios de Fixação

- 1 Com o **Glassfish** executando, abra a interface de administração acessando a url `localhost:4848`. Siga os passos abaixo:

Realms

Create, modify, or delete security (authentication) realms.

Configuration Name: server-config

Realms (3)

| Select | Name | Class Name |
|--------|-------------|---|
| | admin-realm | com.sun.enterprise.security.auth.realm.file.FileRealm |
| | certificate | com.sun.enterprise.security.auth.realm.certificate.CertificateRealm |
| | file | com.sun.enterprise.security.auth.realm.file.FileRealm |

New Realm

Create a new security (authentication) realm. Valid realm types are PAM, OSGi, File, Certificate, LDAP, JDBC, Digest, Oracle Solaris, and Custom.

* Indicates required field

Configuration Name: server-config

Name: * K19-Realm

Class Name: com.sun.enterprise.security.auth.realm.file.FileRealm

Choose a realm class name from the drop-down list or specify a custom class

Properties specific to this Class

JAAS Context: * fileRealm

Identifier for the login module to use for this realm

Key File: * \${com.sun.aas.instanceRoot}/config/K19-RealmFile

Full path and name of the file where the server will store all user, group, and password information for this realm

Assign Groups:

Comma-separated list of group names

- 2 Adicione um usuário chamado **K19** dentro de um grupo chamado **admin** com a senha **K19**. Siga os passos abaixo:

Edit Realm

Edit an existing security (authentication) realm.

Configuration Name: server-config

Realm Name: K19-Realm

Class Name: com.sun.enterprise.security.auth.realm.file.FileRealm

Properties specific to this Class

JAAS Context: * fileRealm
Identifier for the login module to use for this realm

Key File: * \${com.sun.aas.instanceRoot}/config/K19-RealmFile
Full path and name of the file where the server will store all user, group, and password information for this realm

Assign Groups:
Comma-separated list of group names

Additional Properties (0)

File Users - Google Chrome

File Users

Manage user accounts for the currently selected security realm.

Configuration Name: server-config

Realm Name: K19-Realm

File Users (0)

New... Delete

| Select | User ID | Description |
|-----------------|---------|-------------|
| No items found. | | |

File Users

Manage user accounts for the currently selected security realm.

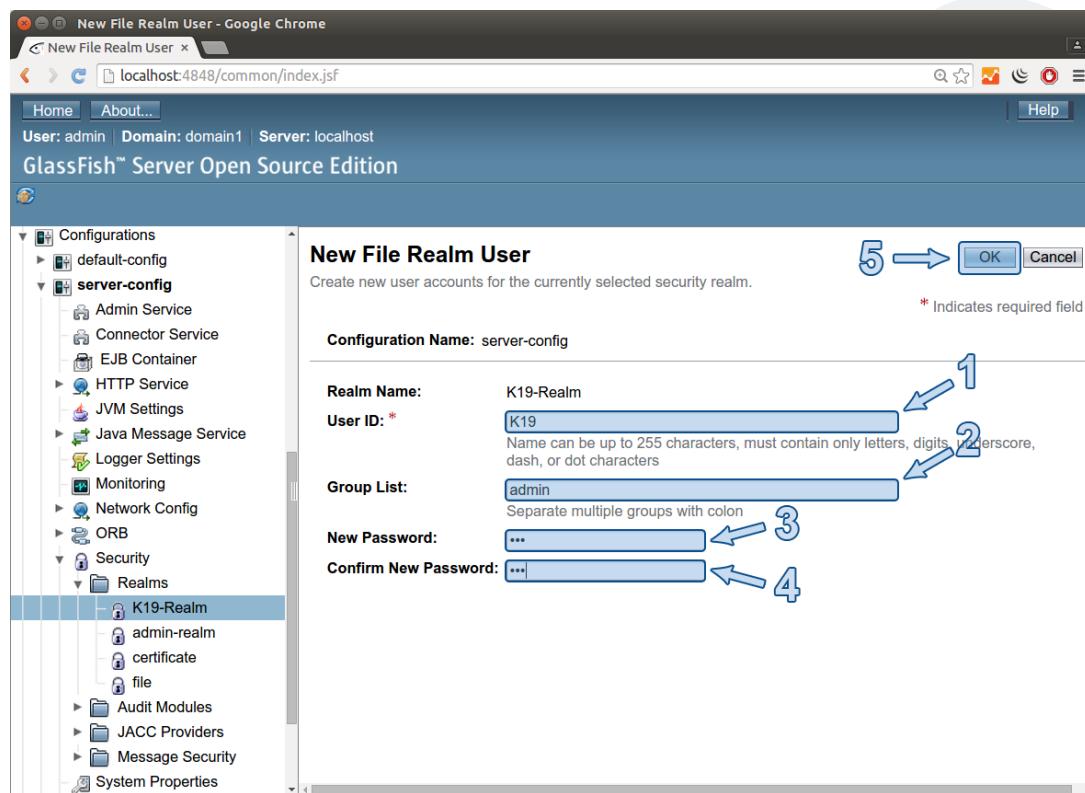
Configuration Name: server-config

Realm Name: K19-Realm

File Users (0)

New... Delete

| Select | User ID | Description |
|-----------------|---------|-------------|
| No items found. | | |



- 3 Repita o processo do exercício anterior e cadastre os seguintes usuários:

| Usuário | Grupo | Senha |
|---------|-------|-------|
| keizo | admin | keizo |
| afk | users | afk |

- 4 Acesse o MySQL Server através do MySQL Workbench ou através do cliente de linha de comando; crie tabelas chamadas **Usuario**, **Grupo** e **Usuario_Grupo** na base de dados **k22_glassfish**.

```

1 CREATE TABLE `Usuario` (
2   `nome` varchar(255) NOT NULL,
3   `senha` varchar(255) NOT NULL,
4   PRIMARY KEY (`nome`)
5 ) ENGINE=InnoDB DEFAULT CHARSET=utf8

```

```

1 CREATE TABLE `Grupo` (
2   `nome` varchar(255) NOT NULL,
3   PRIMARY KEY (`nome`)
4 ) ENGINE=InnoDB DEFAULT CHARSET=utf8

```

```

1 CREATE TABLE `Usuario_Grupo` (
2   `Usuario_nome` varchar(255) NOT NULL,
3   `grupos_nome` varchar(255) NOT NULL,
4   KEY `fk_Usuario_Grupo_1`(`Usuario_nome`),
5   KEY `fk_Usuario_Grupo_2`(`grupos_nome`),
6   CONSTRAINT `fk1` FOREIGN KEY (`Usuario_nome`) REFERENCES `Usuario`(`nome`),
7   CONSTRAINT `fk2` FOREIGN KEY (`grupos_nome`) REFERENCES `Grupo`(`nome`)
8 ) ENGINE=InnoDB DEFAULT CHARSET=utf8

```

5 Adicione registros nas tabelas com o código a seguir.

```

1 INSERT INTO Usuario (nome, senha) VALUES ('K19', md5('K19'));
2 INSERT INTO Usuario (nome, senha) VALUES ('keizo', md5('keizo'));
3 INSERT INTO Usuario (nome, senha) VALUES ('afk', md5('afk'));
4
5 INSERT INTO Grupo (nome) VALUES('admin');
6 INSERT INTO Grupo (nome) VALUES('users');
7
8 INSERT INTO Usuario_Grupo (Usuario_nome, grupos_nome) VALUES('K19', 'admin');
9 INSERT INTO Usuario_Grupo (Usuario_nome, grupos_nome) VALUES('keizo', 'admin');
10 INSERT INTO Usuario_Grupo (Usuario_nome, grupos_nome) VALUES('afk', 'users');
```

6 Com o Glassfish executando, abra a interface de administração acessando a url localhost:4848. Siga os passos abaixo:

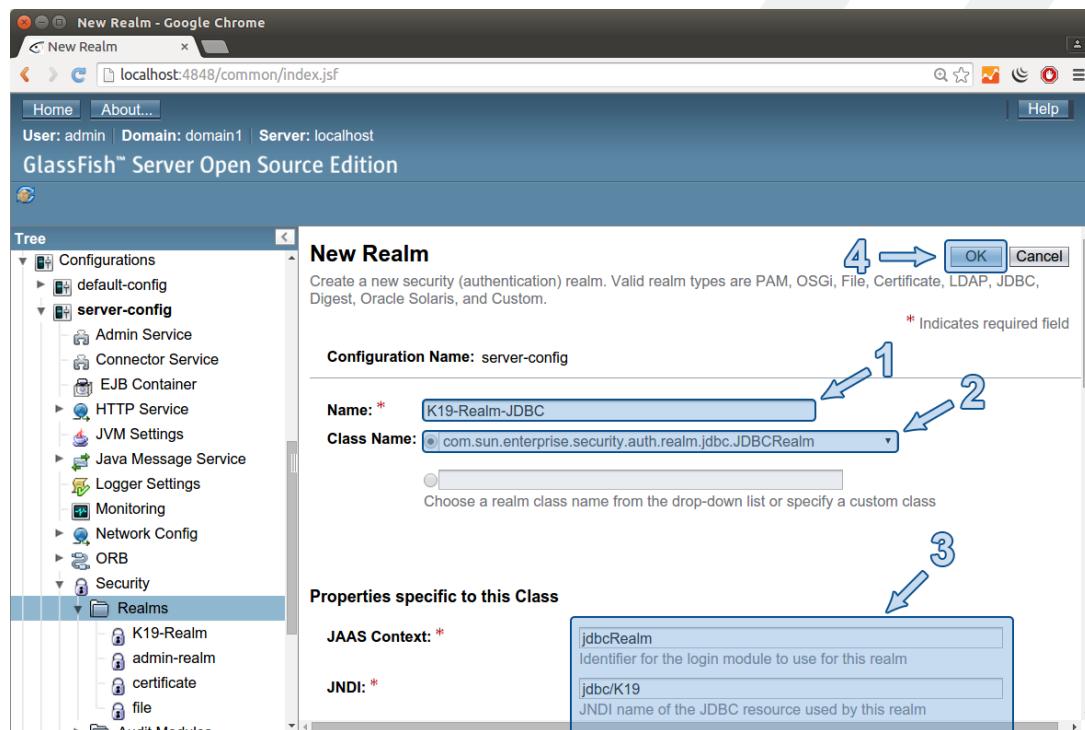
The screenshot shows the GlassFish administration interface. On the left, there's a navigation tree under 'server-config' with sections like Admin Service, Connector Service, EJB Container, HTTP Service, JVM Settings, Java Message Service, Logger Settings, Monitoring, Network Config, ORB, Security, and Realms. Under 'Realms', there are four entries: K19-Realm, admin-realm, certificate, and file. In the center, a table titled 'Realms (4)' lists these realms with columns 'Select', 'Name', and 'Class Name'. The 'Name' column contains K19-Realm, admin-realm, certificate, and file. The 'Class Name' column contains com.sun.enterprise.security.auth.realm.file.FileRealm, com.sun.enterprise.security.auth.realm.file.FileRealm, com.sun.enterprise.security.auth.realm.certificate.CertificateRealm, and com.sun.enterprise.security.auth.realm.file.FileRealm respectively. At the top of the table area, there are buttons for 'New...', 'Edit...', and 'Delete'. A blue arrow points to the 'New...' button.

| Select | Name | Class Name |
|--------|-------------|---|
| | K19-Realm | com.sun.enterprise.security.auth.realm.file.FileRealm |
| | admin-realm | com.sun.enterprise.security.auth.realm.file.FileRealm |
| | certificate | com.sun.enterprise.security.auth.realm.certificate.CertificateRealm |
| | file | com.sun.enterprise.security.auth.realm.file.FileRealm |

Importante: Na próxima tela preencha as seguintes propriedades.

- Name: K19-Realm-JDBC
- Class Name: com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm
- JAAS Context: jdbcRealm
- JNDI: jdbc/K19
- User Table: Usuario
- User Name Column: nome
- Password Column: senha
- Group Table: Usuario_Grupo

- Group Table User Name Column: Usuario_nome
- Group Name Column: grupos_nome
- Digest Algorithm: MD5
- Password Encryption Algorithm: MD5



Autenticação - Aplicações Web

Geralmente, o processo de autenticação é realizado na camada web. Portanto, vamos restringir a nossa discussão a esse tipo de aplicação.

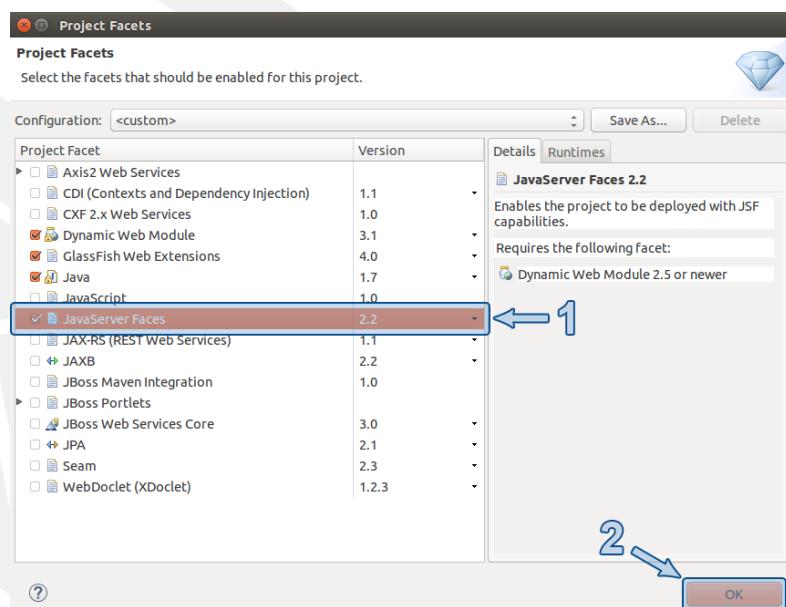
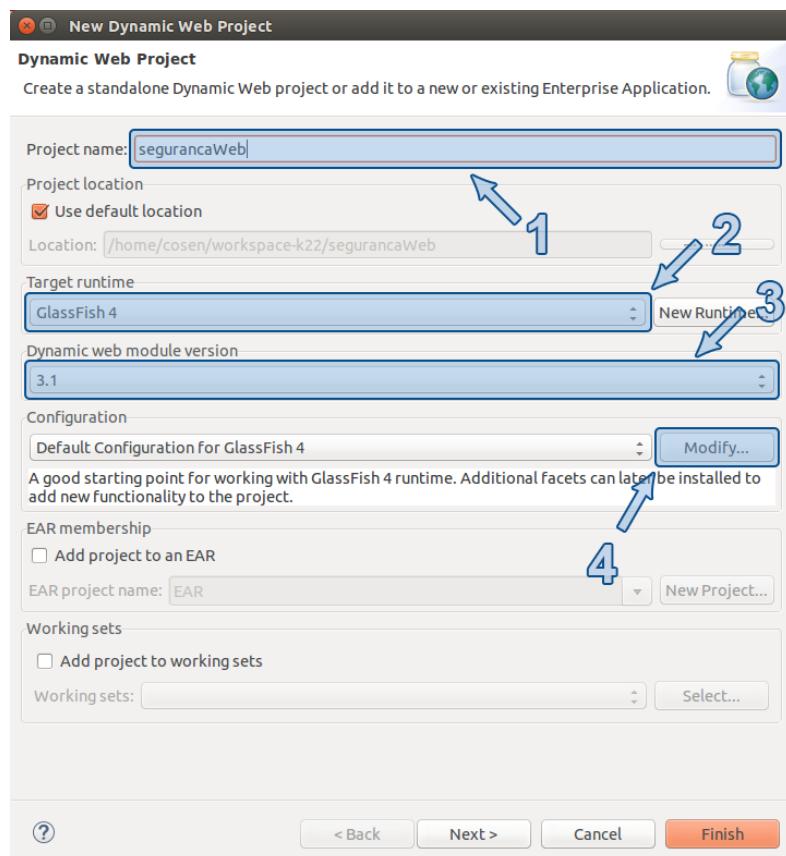
A maior parte das configurações referentes ao processo de autenticação que as aplicações web devem realizar são definidas no arquivo **web.xml**. Contudo, alguns servidores de aplicação exigem configurações extras. Veremos no exercício como configurar uma aplicação web no Glassfish para realizar o processo de autenticação.

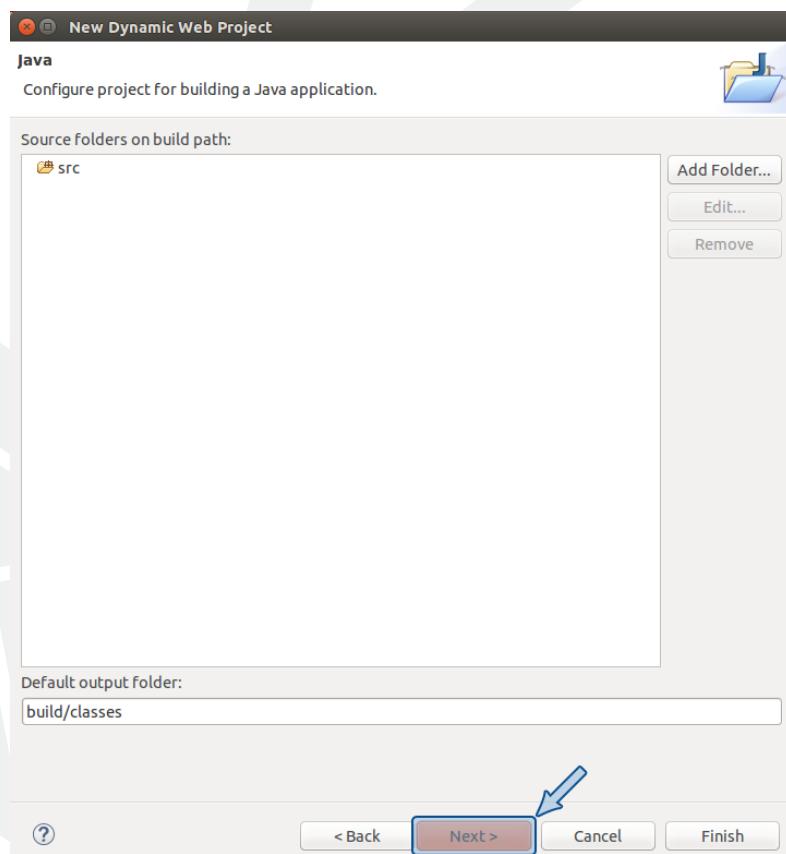
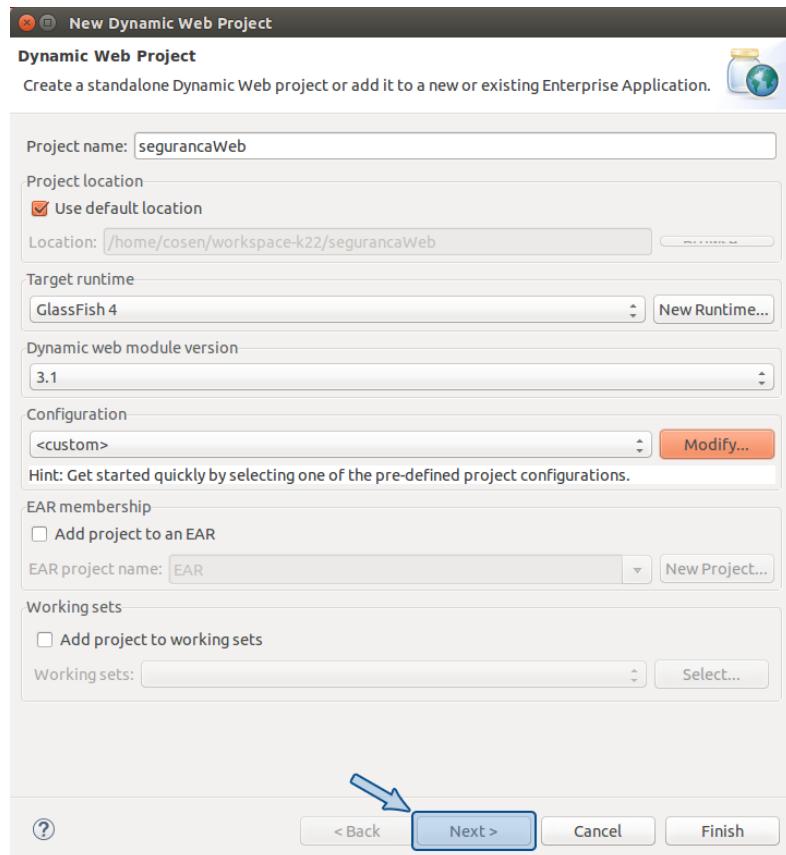


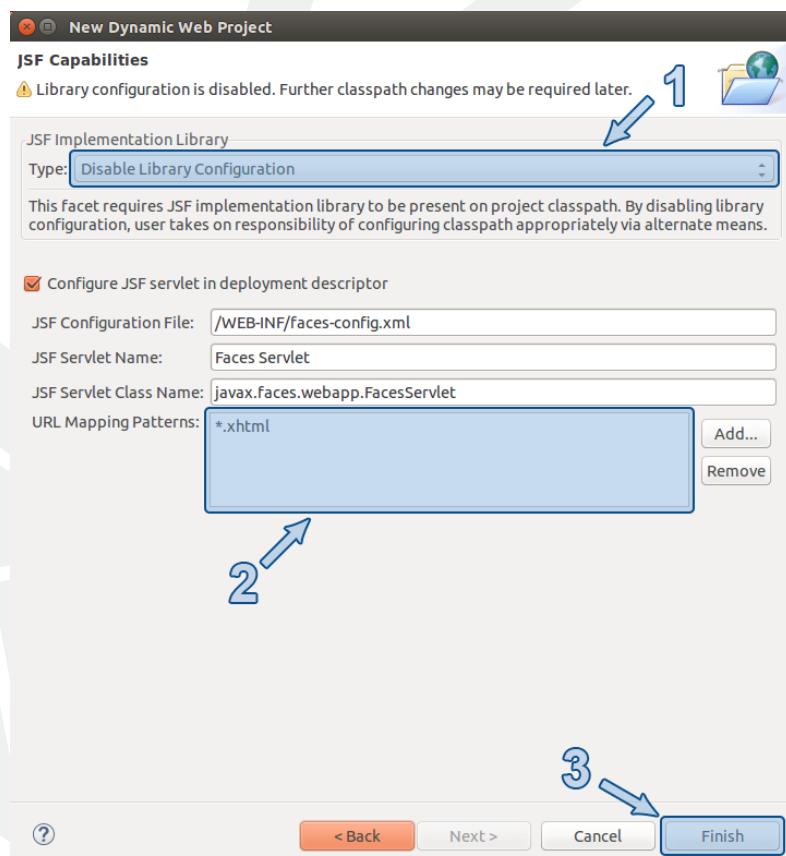
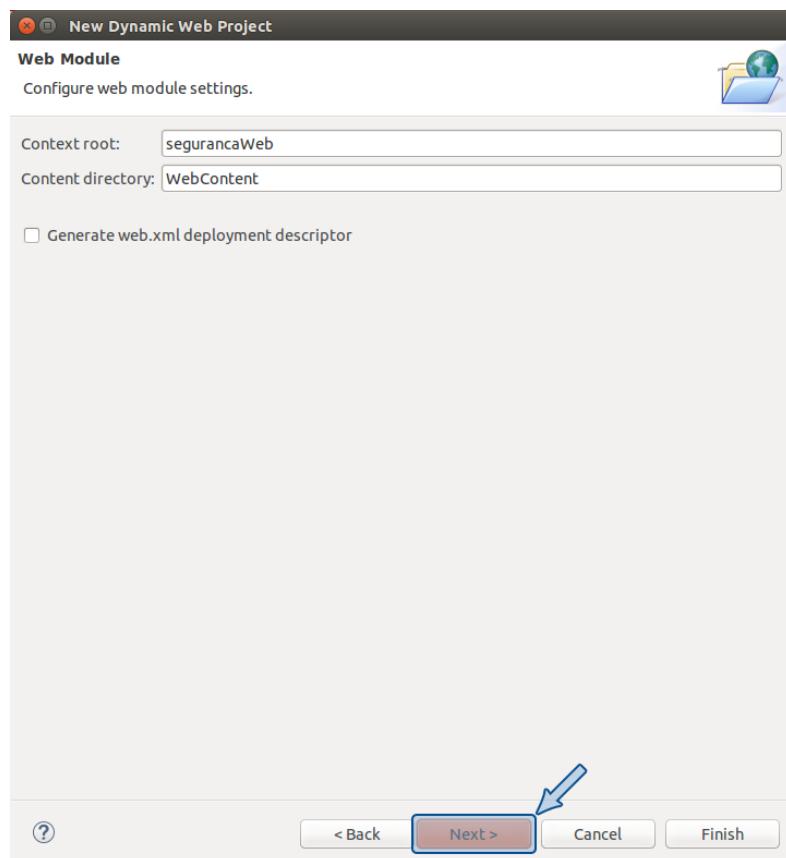
Exercícios de Fixação

- 7 Para não confundir, feche o projeto **transacoesWeb**. Para isso, clique com o botão direito do mouse sobre esse projeto e selecione a opção “Close Project”.

- 8 Crie um Dynamic Web Project no eclipse chamado **segurancaWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.







- 9 Acrescente no arquivo **WebContent/WEB-INF/glassfish-web.xml** do projeto **segurancaWeb** o seguinte trecho de código logo após o elemento **context-root**.

```

1 ...
2 <security-role-mapping>
3   <role-name>ADMIN</role-name>
4   <group-name>admin</group-name>
5 </security-role-mapping>
6
7 <security-role-mapping>
8   <role-name>USERS</role-name>
9   <group-name>users</group-name>
10 </security-role-mapping>
11 ...

```

Código XML 7.1: glassfish-web.xml

Essa configuração é específica do Glassfish.

Os grupos são utilizados pelo Glassfish e os roles pela aplicação.

- 10 Acrescente no arquivo **WebContent/WEB-INF/web.xml** do projeto **segurancaWeb** o seguinte trecho de código dentro do elemento **web-app**.

```

1 ...
2 <welcome-file-list>
3   <welcome-file>index.xhtml</welcome-file>
4 </welcome-file-list>
5
6 <login-config>
7   <auth-method>FORM</auth-method>
8   <realm-name>K19-Realm</realm-name>
9   <form-login-config>
10    <form-login-page>/login.xhtml</form-login-page>
11    <form-error-page>/acesso-negado.xhtml</form-error-page>
12  </form-login-config>
13 </login-config>
14
15 <security-constraint>
16   <web-resource-collection>
17     <web-resource-name>resources</web-resource-name>
18     <url-pattern>/*</url-pattern>
19     <http-method>GET</http-method>
20     <http-method>POST</http-method>
21   </web-resource-collection>
22   <auth-constraint>
23     <role-name>ADMIN</role-name>
24     <role-name>USERS</role-name>
25   </auth-constraint>
26 </security-constraint>
27 ...

```

Código XML 7.2: web.xml

- 11 Adicione um arquivo chamado **login.xhtml** na pasta **WebContent** do projeto **segurancaWeb** com seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets">

```

```

6  xmlns:h="http://java.sun.com/jsf/html"
7  xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10 <title>Segurança</title>
11 </h:head>
12
13 <h:body>
14 <form method="post" action="j_security_check">
15   Usuário: <input type="text" name="j_username" />
16   Senha: <input type="password" name="j_password" />
17   <input type="submit" value="Login" />
18 </form>
19 </h:body>
20 </html>

```

Código XHTML 7.1: login.xhtml

- 12** Adicione um arquivo chamado **acesso-negado.xhtml** na pasta **WebContent** do projeto **segurancaWeb** com seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10 <title>Segurança</title>
11 </h:head>
12
13 <h:body>
14 <h1>Acesso Negado</h1>
15 <h:link outcome="/login" value="Tentar novamente"/>
16 </h:body>
17 </html>

```

Código XHTML 7.2: acesso-negado.xhtml

- 13** Adicione um arquivo chamado **menu.xhtml** em uma pasta chamada **includes** dentro de **Web-Content/WEB-INF** no projeto **segurancaWeb** com seguinte conteúdo.

```

1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:ui="http://java.sun.com/jsf/facelets"
3   xmlns:h="http://java.sun.com/jsf/html"
4   xmlns:f="http://java.sun.com/jsf/core">
5
6 <h:form>
7   <h:panelGrid>
8     <h:commandLink action="#{autenticadorMB.sair}" value="Sair" />
9   </h:panelGrid>
10 </h:form>
11
12 </ui:composition>

```

Código XHTML 7.3: menu.xhtml

- 14** Adicione um arquivo chamado **index.xhtml** na pasta **WebContent** do projeto **segurancaWeb**

com seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10   <title>Segurança</title>
11 </h:head>
12
13 <h:body>
14   <h1>Autenticado</h1>
15   <ui:include src="/WEB-INF/includes/menu.xhtml"/>
16 </h:body>
17 </html>
```

Código XHTML 7.4: index.xhtml

- 15 Crie um pacote chamado **br.com.k19.managedbeans** no projeto **segurancaWeb** e adicione nesse pacote um Managed Bean.

```

1 package br.com.k19.managedbeans;
2
3 import javax.faces.bean.ManagedBean;
4 import javax.faces.context.ExternalContext;
5 import javax.faces.context.FacesContext;
6 import javax.servlet.http.HttpSession;
7
8 @ManagedBean
9 public class AutenticadorMB {
10
11   public String sair(){
12     FacesContext fc = FacesContext.getCurrentInstance();
13     ExternalContext ec = fc.getExternalContext();
14     HttpSession session = (HttpSession) ec.getSession(false);
15     session.invalidate();
16
17     return "/login";
18   }
19 }
```

Código Java 7.1: AutenticadorMB.java

- 16 Remova o projeto **transacoesWeb** do **Glassfish**. Adicione o projeto **segurancaWeb** no **Glassfish**. Certifique-se que o **JBoss** e o **Wildfly** estejam parados. Inicie o **Glassfish 3.1.2** e teste a aplicação acessando a url <http://localhost:8080/segurancaWeb>.

Obs: Os usuários cadastrados no K19-Realm são:

| Usuário | Grupo | Senha |
|---------|-------|-------|
| K19 | admin | K19 |
| keizo | admin | keizo |
| afk | users | afk |

- 17 Altere o arquivo **WebContent/WEB-INF/web.xml** para utilizar o JDBC Realm.

```

1 . .
2 <login-config>
3   <auth-method>FORM</auth-method>
4   <realm-name>K19-Realm-JDBC</realm-name>
5   <form-login-config>
6     <form-login-page>/login.xhtml</form-login-page>
7     <form-error-page>/acesso-negado.xhtml</form-error-page>
8   </form-login-config>
9 </login-config>
10 </login-config>
11 . .
12 . .

```

Código XML 7.3: web.xml

- 18 Reinicie o **Glassfish** e teste a aplicação acessando a url <http://localhost:8080/segurancaWeb>.

- 19 Adicione uma pasta chamada **META-INF** na pasta **src** do projeto **segurancaWeb**. Na pasta **META-INF**, crie o arquivo **persistence.xml**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="none"/>
15    </properties>
16  </persistence-unit>
17 </persistence>

```

Código XML 7.4: persistence.xml

- 20 Crie uma classe chamada **Grupo** em um pacote chamado **br.com.k19.entidades** no projeto **segurancaWeb**.

```

1 package br.com.k19.entidades;
2
3 import javax.persistence.Entity;
4 import javax.persistence.Id;
5
6 @Entity
7 @Table(name="Grupo")
8 public class Grupo {
9
10   @Id
11   private String nome;
12
13   public String getNome() {
14     return nome;
15   }

```

```

16  public void setNome(String nome) {
17      this.nome = nome;
18  }
19
20  @Override
21  public String toString() {
22      return this.nome;
23  }
24
25 }
```

Código Java 7.2: Grupo.java

- 21** Crie uma classe chamada **Usuario** no pacote **br.com.k19.entidades** no projeto **segurançaWeb**.

```

1 package br.com.k19.entidades;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.persistence.Entity;
7 import javax.persistence.FetchType;
8 import javax.persistence.Id;
9 import javax.persistence.ManyToMany;
10
11 @Entity
12 @Table(name="Usuario")
13 public class Usuario {
14
15     @Id
16     private String nome;
17
18     private String senha;
19
20     @ManyToMany(fetch=FetchType.EAGER)
21     private List<Grupo> grupos = new ArrayList<Grupo>();
22
23     // GETTERS AND SETTERS
24 }
```

Código Java 7.3: Usuario.java

- 22** Crie uma classe chamada **GrupoRepositorio** em um pacote chamado **br.com.k19.sessionbeans** no projeto **segurançaWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.ejb.Stateless;
6 import javax.persistence.EntityManager;
7 import javax.persistence.PersistenceContext;
8 import javax.persistence.TypedQuery;
9
10 import br.com.k19.entidades.Grupo;
11
12 @Stateless
13 public class GrupoRepositorio {
14
15     @PersistenceContext
16     private EntityManager manager;
17
18
19     public void adiciona(Grupo g){
```

```

20     this.manager.persist(g);
21 }
22
23 public List<Grupo> buscaTodos(){
24     TypedQuery<Grupo> query = this.manager.createQuery("select x from Grupo x", Grupo.class);
25     return query.getResultList();
26 }
27 }
```

Código Java 7.4: GrupoRepositorio.java

- 23** Crie uma classe chamada **UsuarioRepositorio** no pacote **br.com.k19.sessionbeans** no projeto **segurançaWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.ejb.Stateless;
6 import javax.persistence.EntityManager;
7 import javax.persistence.PersistenceContext;
8 import javax.persistence.TypedQuery;
9
10 import br.com.k19.entidades.Usuario;
11
12 @Stateless
13 public class UsuarioRepositorio {
14
15     @PersistenceContext
16     private EntityManager manager;
17
18
19     public void adiciona(Usuario u){
20         this.manager.persist(u);
21     }
22
23     public List<Usuario> buscaTodos(){
24         TypedQuery<Usuario> query =
25             this.manager.createQuery("select x from Usuario x", Usuario.class);
26         return query.getResultList();
27     }
28 }
```

Código Java 7.5: UsuarioRepositorio.java

- 24** Crie uma classe chamada **UsuarioMB** em um pacote chamado **br.com.k19.managedbeans** no projeto **segurançaWeb**.

```

1 package br.com.k19.managedbeans;
2
3 import java.math.BigInteger;
4 import java.security.MessageDigest;
5 import java.security.NoSuchAlgorithmException;
6 import java.util.List;
7
8 import javax.ejb.EJB;
9 import javax.faces.bean.ManagedBean;
10
11 import br.com.k19.entidades.Grupo;
12 import br.com.k19.entidades.Usuario;
13 import br.com.k19.sessionbeans.GrupoRepositorio;
14 import br.com.k19.sessionbeans.UsuarioRepositorio;
```

```
15  @ManagedBean
16  public class UsuarioMB {
17
18      @EJB
19      private UsuarioRepositorio usuarioRepositorio;
20
21      @EJB
22      private GrupoRepositorio grupoRepositorio;
23
24      private Usuario usuario = new Usuario();
25
26      private List<String> nomesDosGrupos;
27
28      private List<Usuario> usuarios;
29
30      private List<Grupo> grupos;
31
32
33      public void adiciona() throws NoSuchAlgorithmException {
34          // Associando os Grupos ao novo Usuário
35          for (String nomeDoGrupo : this.nomesDosGrupos) {
36              Grupo g = new Grupo();
37              g.setNome(nomeDoGrupo);
38              this.usuario.getGrupos().add(g);
39          }
40
41          // Criptografando a senha do novo Usuário
42          MessageDigest md = MessageDigest.getInstance("MD5"); // or "SHA-1"
43          md.update(this.usuario.getSenha().getBytes());
44          BigInteger hash = new BigInteger(1, md.digest());
45          String senhaCriptografada = hash.toString(16);
46          while (senhaCriptografada.length() < 32) { // 40 for SHA-1
47              senhaCriptografada = "0" + senhaCriptografada;
48          }
49          this.usuario.setSenha(senhaCriptografada);
50
51          // Salvando o usuário
52          this.usuarioRepositorio.adiciona(this.usuario);
53          this.usuario = new Usuario();
54          this.usuarios = null;
55      }
56
57      public List<Grupo> getGrupos() {
58          if (this.grupos == null) {
59              this.grupos = this.grupoRepositorio.buscaTodos();
60          }
61
62          return this.grupos;
63      }
64
65      public List<Usuario> getUsuarios() {
66          if (this.usuarios == null) {
67              this.usuarios = this.usuarioRepositorio.buscaTodos();
68          }
69
70          return this.usuarios;
71      }
72
73      public Usuario getUsuario() {
74          return usuario;
75      }
76
77      public void setUsuario(Usuario usuario) {
78          this.usuario = usuario;
79      }
80
81      public List<String> getNomesDosGrupos() {
82          return nomesDosGrupos;
83      }
84  }
```

```

85     public void setNomesDosGrupos(List<String> nomesDosGrupos) {
86         this.nomesDosGrupos = nomesDosGrupos;
87     }
88 }
```

Código Java 7.6: UsuarioMB.java

- 25 Adicione um arquivo chamado **usuarios.xhtml** na pasta **WebContent** do projeto **segurancaWeb** com seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:ui="http://java.sun.com/jsf/facelets"
6      xmlns:h="http://java.sun.com/jsf/html"
7      xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10    <title>Segurança</title>
11 </h:head>
12
13 <h:body>
14
15    <ui:include src="/WEB-INF/includes/menu.xhtml"/>
16
17    <h1>Novo Usuário</h1>
18
19    <h:messages />
20
21    <h:form>
22        <h:panelGrid columns="2">
23            <h:outputLabel value="Nome" />
24            <h:inputText value="#{usuarioMB.usuario.nome}" required="true" />
25            <h:outputLabel value="Senha" />
26            <h:inputSecret value="#{usuarioMB.usuario.senha}" required="true" />
27
28            <h:selectManyCheckbox value="#{usuarioMB.nomesDosGrupos}"
29                required="true">
30                <f:selectItems value="#{usuarioMB.grupos}" var="g"
31                    itemLabel="#{g.nome}" itemValue="#{g.nome}" />
32            </h:selectManyCheckbox>
33        </h:panelGrid>
34
35        <h:commandButton value="Salvar" action="#{usuarioMB.adiciona}" />
36    </h:form>
37
38    <hr />
39
40    <h:dataTable value="#{usuarioMB.usuarios}" var="u" border="1">
41        <h:column>
42            <f:facet name="header">Nome do Usuário</f:facet>
43            <h:outputText value="#{u.nome}" />
44        </h:column>
45
46        <h:column>
47            <f:facet name="header">Grupos</f:facet>
48            <h:outputText value="#{u.grupos}" />
49        </h:column>
50    </h:dataTable>
51 </h:body>
52 </html>
```

Código XHTML 7.5: index.xhtml

- 26 Modifique o arquivo **WebContent/WEB-INF/includes/menu.xhtml** do projeto **segurancaWeb**.

```

1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:ui="http://java.sun.com/jsf/facelets"
3   xmlns:h="http://java.sun.com/jsf/html"
4   xmlns:f="http://java.sun.com/jsf/core">
5
6   <h:form>
7     <h:panelGrid>
8       <h:commandLink action="#{autenticadorMB.sair}" value="Sair" />
9       <h:commandLink action="/usuarios" value="Usuários" />
10      </h:panelGrid>
11    </h:form>
12  </ui:composition>
```

Código XHTML 7.6: menu.xhtml

- 27 Reinicie o **Glassfish** e teste a aplicação acessando a url <http://localhost:8080/segurancaWeb>.

Obs: Os usuários cadastrados no K19-Realm-JDBC são:

| Usuário | Grupo | Senha |
|---------|-------|-------|
| K19 | admin | K19 |
| keizo | admin | keizo |
| afk | users | afk |

- 28 Adicione e teste novos usuários através do endereço <http://localhost:8080/segurancaWeb/usuarios.xhtml>



Autorização - Aplicações EJB

Podemos limitar o acesso dos usuários aos métodos de um Session Bean. Por exemplo, é possível declarar que um determinado método de um Session Bean só pode ser chamado por usuários administradores ou moderadores.

@RolesAllowed

Restrições de acesso podem ser definidas pela anotação **@RolesAllowed** que pode ser aplicada na classe ou nos métodos de um Session Bean. Se aplicada na classe valerá para todos os métodos. Se aplicada ao mesmo tempo na classe e em algum método, valerá as restrições definidas no método.

```

1 @RolesAllowed({"administrador", "moderador"})
2 public void adiciona(Produto produto){
3   this.manager.persist(produto);
4 }
```

```

1 @RolesAllowed({"administrador", "moderador"})
2 @Stateful
3 class CarrinhoBean {
4   ...
5 }
```

@PermitAll

Podemos utilizar a anotação **@PermitAll** para permitir que qualquer tipo de usuário tenha acesso. Para conseguir o mesmo efeito com a anotação **@RolesAllowed**, teríamos que listar todos os Roles. Além disso, caso um Role fosse criado ou destruído, alterações seriam necessárias.

```
1 @PermitAll
2 public void adiciona(Produto produto){
3     this.manager.persist(produto);
4 }
```

```
1 @PermitAll
2 @Stateful
3 class CarrinhoBean {
4     ...
5 }
```

Código Java 7.10: CarrinhoBean.java

@DenyAll

O funcionamento da anotação **@DenyAll** é exatamente o oposto da **@PermitAll**. Podemos utilizar a anotação **@DenyAll** em aplicações que são implantadas em ambientes diferentes. Sendo que em determinados ambientes certas funcionalidades devem ser desabilitadas.

```
1 @DenyAll
2 public void adiciona(Produto produto){
3     this.manager.persist(produto);
4 }
```

```
1 @DenyAll
2 @Stateful
3 class CarrinhoBean {
4     ...
5 }
```

Código Java 7.12: CarrinhoBean.java

@RunAs

Eventualmente, um Session Bean chama outro Session Bean. Suponha, que os métodos do primeiro possam ser executados por usuários moderadores e os métodos do segundo por administradores. Para que o primeiro Session Bean possa chamar o Segundo, temos que definir o papel de administrador para o primeiro Session Bean através da anotação **@RunAs**.

```
1 @Stateless
2 @RunAs("administrador")
3 class MensagemRepositorio {
4
5     @PersistenceContext
6     private EntityManager manager;
7
8     @EJB
9     private TopicoRepositorio topicoRepositorio;
10
11    @RolesAllowed({"moderador"})
12    public void remove(Long id){
13        Mensagem m = this.manager.find(Mensagem.class, id);
14        this.manager.remove(m);
```

```

15     Topico t = m.getTopico();
16
17     if(t.getMensagens().size() == 1){
18         this.topicoRepositorio.remove(t);
19     }
20 }
21 }
22 }
```

Código Java 7.13: MensagemRepository.java



Exercícios de Fixação

- 29** No pacote **br.com.k19.sessionbeans** do projeto **segurançaWeb**, adicione uma classe chamada **TarefasBean**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.annotation.security.RolesAllowed;
7 import javax.ejb.Singleton;
8
9 @Singleton
10 public class TarefasBean {
11
12     private List<String> tarefas = new ArrayList<String>();
13
14     @RolesAllowed({"ADMIN", "USERS"})
15     public void adiciona(String tarefa){
16         this.tarefas.add(tarefa);
17     }
18
19     @RolesAllowed({"ADMIN", "USERS"})
20     public List<String> listaTarefas() {
21         return this.tarefas;
22     }
23
24     @RolesAllowed({"ADMIN"})
25     public void remove(String tarefa){
26         this.tarefas.remove(tarefa);
27     }
28 }
```

Código Java 7.14: TarefaBean.java

- 30** No pacote **br.com.k19.managedbeans** do projeto **segurançaWeb**, adicione uma classe chamada **TarefasMB**.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.ejb.EJB;
6 import javax.faces.bean.ManagedBean;
7
8 import br.com.k19.sessionbeans.TarefasBean;
9
10 @ManagedBean
```

```

11 public class TarefasMB {
12
13     @EJB
14     private TarefasBean tarefasBean;
15
16     private String tarefa;
17
18     public void adiciona(){
19         this.tarefasBean.adiciona(this.tarefa);
20     }
21
22     public void remove(String tarefa){
23         this.tarefasBean.remove(tarefa);
24     }
25
26     public List<String> getTarefas(){
27         return this.tarefasBean.listaTarefas();
28     }
29
30     // GETTERS AND SETTERS
31 }
```

Código Java 7.15: TarefaMB.java

- 31** Crie um arquivo chamado **tarefas.xhtml** na pasta **WebContent** do projeto **segurancaWeb**.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5      xmlns:ui="http://java.sun.com/jsf/facelets"
6      xmlns:h="http://java.sun.com/jsf/html"
7      xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10    <title>Segurança</title>
11 </h:head>
12
13 <h:body>
14    <ui:include src="/WEB-INF/includes/menu.xhtml"/>
15    <h1>Nova Tarefa</h1>
16    <h:form>
17        <h:outputLabel value="Tarefa: " />
18        <h:inputText value="#{tarefasMB.tarefa}" />
19
20        <h:commandButton action="#{tarefasMB.adiciona}" value="Salvar" />
21
22        <h1>Lista de Tarefas</h1>
23        <h:dataTable value="#{tarefasMB.tarefas}" var="tarefa">
24            <h:column>
25                <h:outputText value="#{tarefa}" />
26            </h:column>
27            <h:column>
28                <h:commandLink action="#{tarefasMB.remove(tarefa)}" >remove</h:commandLink>
29            </h:column>
30        </h:dataTable>
31    </h:form>
32 </h:body>
33 </html>
```

Código XHTML 7.7: tarefas.xhtml

- 32** Modifique o arquivo **WebContent/WEB-INF/includes/menu.xhtml** do projeto **segurancaWeb**.

```
1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:ui="http://java.sun.com/jsf/facelets"
3   xmlns:h="http://java.sun.com/jsf/html"
4   xmlns:f="http://java.sun.com/jsf/core">
5
6   <h:form>
7     <h:panelGrid>
8       <h:commandLink action="#{autenticadorMB.sair}" value="Sair" />
9       <h:commandLink action="/usuarios" value="Usuários" />
10      <h:commandLink action="/tarefas" value="Tarefas" />
11    </h:panelGrid>
12  </h:form>
13 </ui:composition>
```

Código XHTML 7.8: menu.xhtml

- 33 Observe que usuários com role USERS não conseguem remover tarefas.

INTERCEPTADORES

Uma aplicação EJB pode definir, através de métodos de callback, lógicas a serem executadas pelo EJB Container quando uma instância de um Session Bean muda de estado.

- O método de callback “PostConstruct” é executado quando uma instância de um Session Bean de qualquer tipo muda do estado NÃO EXISTE para o PRONTO.
- O método de callback “PreDestroy” é executado quando uma instância de um Session Bean muda do estado PRONTO para o NÃO EXISTE.
- O método de callback “PrePassivate” é executado quando uma instância de um Stateful Session Bean muda do estado PRONTO para o PASSIVADO.
- O método de callback “PostActivate” é executado quando uma instância de um Stateful Session Bean muda do estado PASSIVADO para o PRONTO.

Há um método de callback para cada uma das seguintes transições: NÃO EXISTE->PRONTO, PRONTO->PASSIVADO, PASSIVADO->PRONTO e PRONTO->NÃO EXISTE. Além dessas transições, podemos considerar que toda vez que um método de negócio é chamado ocorre a transição PRONTO->PRONTO. Não há método de callback para essa transição especial. Contudo, na arquitetura EJB, podemos utilizar a ideia de interceptadores para conseguir executar lógicas antes ou depois da execução de um método de negócio.

É comum utilizar interceptadores para tarefas que não estão diretamente relacionadas às regras de negócio implementadas nos Session Beans. Por exemplo, podemos implementar logging ou controle de acesso com interceptadores.



Interceptor Methods

A lógica de um interceptador é definida dentro de um método anotado com **@AroundInvoke** ou registrado através de XML. Não há restrições em relação a visibilidade desse método, ou seja, ele pode ser público, protegido, padrão ou privado. Contudo, ele deve possuir uma assinatura compatível com o seguinte formato:

```
1 Object <METODO>(InvocationContext) throws Exception
```

Veja um exemplo concreto de método interceptador:

```
1 @AroundInvoke
2 public Object interceptor(InvocationContext ic) throws Exception {
3     // IDA
4     System.out.println("ANTES DO MÉTODO DE NEGÓCIO");
5
6     // CHAMANDO O MÉTODO DE NEGÓCIO E PEGANDO O SEU RETORNO
7     Object retornoDoMetodoDeNegocio = ic.proceed();
```

```

8 // VOLTA
9 System.out.println("DEPOIS DO MÉTODO DE NEGÓCIO");
10 return retornoDoMetodoDeNegocio;
11 }
12 }
```



Internal Interceptors

Um interceptador interno é criado quando um método interceptador é definido dentro de um Session Bean. Cada Session Bean pode ter no máximo um interceptador interno. Um interceptador interno atua em todos os métodos de negócios do seu respectivo Session Bean.

```

1 @Stateless
2 class CalculadoraBean {
3
4     // MÉTODOS DE NEGÓCIO
5
6     // MÉTODOS DE CALLBACK
7
8     // MÉTODO INTERCEPTADOR
9     @AroundInvoke
10    public Object interceptador(InvocationContext ic) throws Exception {
11        // IDA
12        System.out.println("ANTES DO MÉTODO DE NEGÓCIO");
13
14        // CHAMANDO O MÉTODO DE NEGÓCIO E PEGANDO O SEU RETORNO
15        Object retornoDoMetodoDeNegocio = ic.proceed();
16
17        // VOLTA
18        System.out.println("DEPOIS DO MÉTODO DE NEGÓCIO");
19        return retornoDoMetodoDeNegocio;
20    }
21 }
```

Código Java 8.3: CalculadoraBean.java



External Interceptors

Os interceptadores externos são criados quando um método interceptador é definido fora de um Session Bean em uma classe comum. Novamente, não mais do que um método interceptador pode ser definido em uma mesma classe.

```

1 class LoggingInterceptor {
2
3     @AroundInvoke
4     public Object interceptador(InvocationContext ic) throws Exception {
5        // IDA
6        System.out.println("ANTES DO MÉTODO DE NEGÓCIO");
7
8        // CHAMANDO O MÉTODO DE NEGÓCIO E PEGANDO O SEU RETORNO
9        Object retornoDoMetodoDeNegocio = ic.proceed();
10
11        // VOLTA
12        System.out.println("DEPOIS DO MÉTODO DE NEGÓCIO");
13        return retornoDoMetodoDeNegocio;
14    }
15 }
```

Código Java 8.4: LoggingInterceptor.java

Method-Level Interceptors

Interceptadores externos podem ser associados a métodos de negócio através da anotação **@Interceptors**.

```

1 @Stateless
2 class CalculadoraBean {
3
4     @Interceptors({LoggingInterceptor.class})
5     public double soma(double a, double b){
6         return a + b;
7     }
8 }
```

Código Java 8.5: CalculadoraBean.java

Vários interceptadores externos podem ser associados a um método de negócio através da anotação **@Interceptors**.

```

1 @Stateless
2 class CalculadoraBean {
3
4     @Interceptors({LoggingInterceptor.class, SegurancaInterceptor.class})
5     public double soma(double a, double b){
6         return a + b;
7     }
8 }
```

Código Java 8.6: CalculadoraBean.java

Class-Level Interceptors

Interceptadores externos também podem ser associados a Session Beans através da anotação **@Interceptors**. Quando associado a um Session Bean, um interceptador externo será aplicado a todos os métodos de negócio desse Session Bean.

```

1 @Stateless
2 @Interceptors({LoggingInterceptor.class})
3 class CalculadoraBean {
4
5     public double soma(double a, double b){
6         return a + b;
7     }
8 }
```

Código Java 8.7: CalculadoraBean.java

Vários interceptadores externos podem ser associados a um Session Bean através da anotação **@Interceptors**.

```

1 @Stateless
2 @Interceptors({LoggingInterceptor.class, SegurancaInterceptor.class})
3 class CalculadoraBean {
4
5     public double soma(double a, double b){
6         return a + b;
7     }
8 }
```

Código Java 8.8: CalculadoraBean.java

Default Interceptors

Interceptadores externos também podem ser associados a métodos de negócios através de configurações adicionadas no arquivo de configuração do EJB, o **ejb-jar.xml**. Esse arquivo deve ser colocado em uma pasta chamada **META-INF** dentro do módulo EJB da aplicação.

Por exemplo, suponha que o interceptador externo definido pela classe `LoggingInterceptor` tenha que ser aplicado em todos os métodos de negócios de todos os Session Beans.

```

1 <interceptor-binding>
2   <ejb-name>*</ejb-name>
3   <interceptor-class>interceptadores.LoggingInterceptor</interceptor-class>
4 </interceptor-binding>
```

Ou podemos aplicar a apenas um Session Bean.

```

1 <interceptor-binding>
2   <ejb-name>sessionbeans.CalculadoraBean</ejb-name>
3   <interceptor-class>INTERCEPTOR</interceptor-class>
4 </interceptor-binding>
```



Excluindo Interceptadores

Podemos excluir os Default Interceptors e os Class-Level Interceptors através das anotações **@ExcludeDefaultInterceptors** e **@ExcludeClassInterceptors** respectivamente.

A anotação `@ExcludeDefaultInterceptors` pode ser aplicada em um método de negócios ou no Session Bean.

```

1 @Stateless
2 @ExcludeDefaultInterceptors
3 class CalculadoraBean {
4
5   public double soma(double a, double b){
6     return a + b;
7   }
8 }
```

Código Java 8.9: *CalculadoraBean.java*

A anotação `@ExcludeClassInterceptors` pode ser aplicada em um método de negócios.

```

1 @Stateless
2 @Interceptors({LoggingInterceptor.class, SegurancaInterceptor.class})
3 class CalculadoraBean {
4
5   @ExcludeClassInterceptors
6   public double soma(double a, double b){
7     return a + b;
8   }
9 }
```

Código Java 8.10: *CalculadoraBean.java*



Invocation Context

Um método interceptador recebe um Invocation Context como parâmetro. Através dos Invocation Context, os métodos interceptadores podem acessar a instância do Session Bean que será utilizada para atender a chamada, descobrir qual método de negócio será executado, quais parâmetros foram passados e até mesmo trocar os parâmetros antes de chegar no método de negócio. Veja os métodos disponíveis nessa interface.

```

1 public interface InvocationContext {
2     public Object getTarget();
3     public Method getMethod();
4     public Object[] getParameters();
5     public void setParameters(Object[]);
6     public java.util.Map<String, Object> getContextData();
7     public Object proceed() throws Exception;
8 }
```

Código Java 8.11: InvocationContext.java



Ordem dos Interceptadores

A ordem na qual os interceptadores são executados pode afetar o funcionamento da aplicação. A especificação dos interceptadores define a seguinte ordem:

1. Default Interceptors
2. Class-Level Interceptors
3. Method-Level Interceptors
4. Internal Interceptors

Quando dois ou mais Default Interceptors estão associados a um método de negócio, eles serão executados na ordem em que foram definidos no **ejb-jar.xml**.

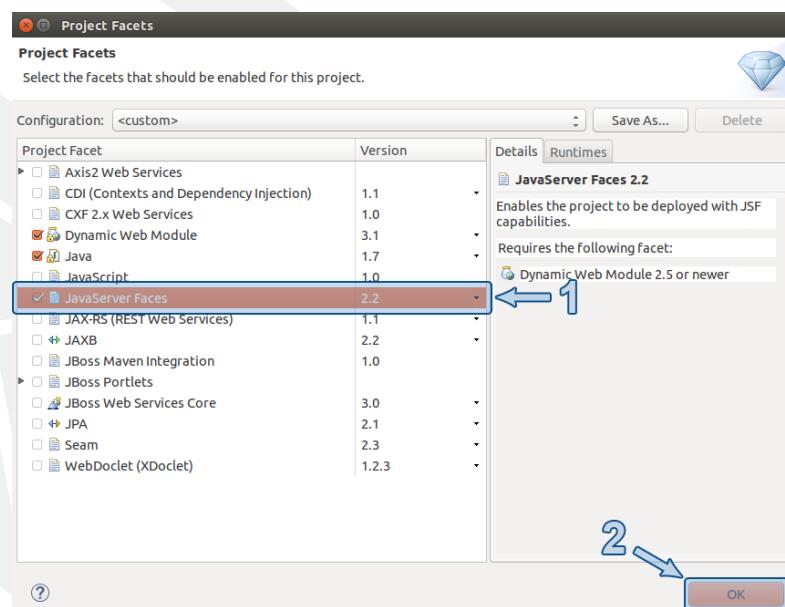
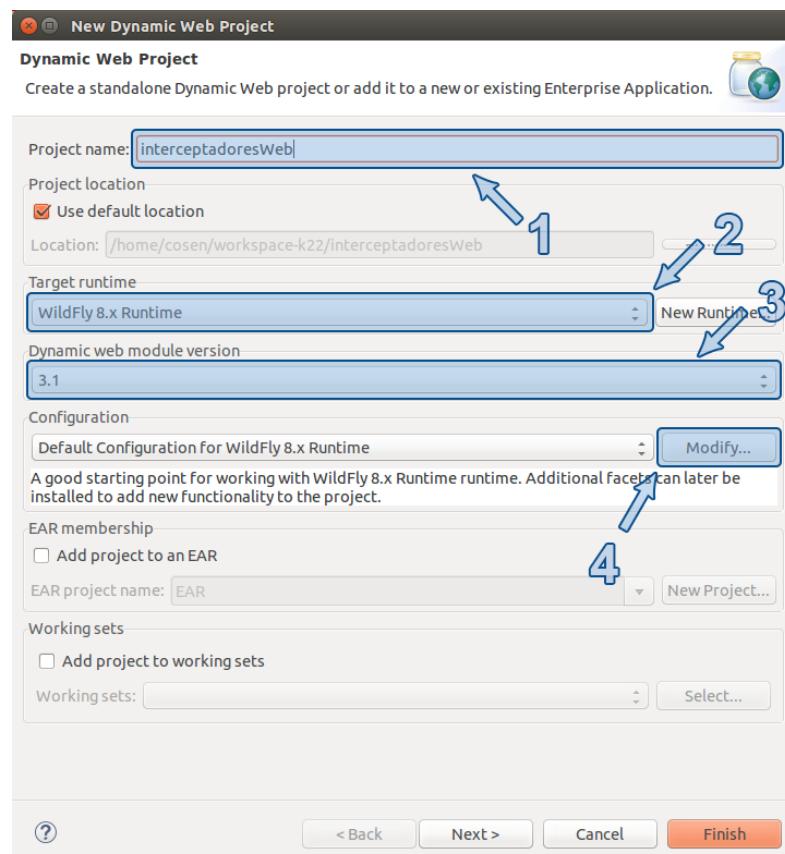
Quando dois ou mais Class-Level Interceptors estão associados a um método de negócio, eles serão executados na ordem em que foram declarados na anotação **@Interceptors**.

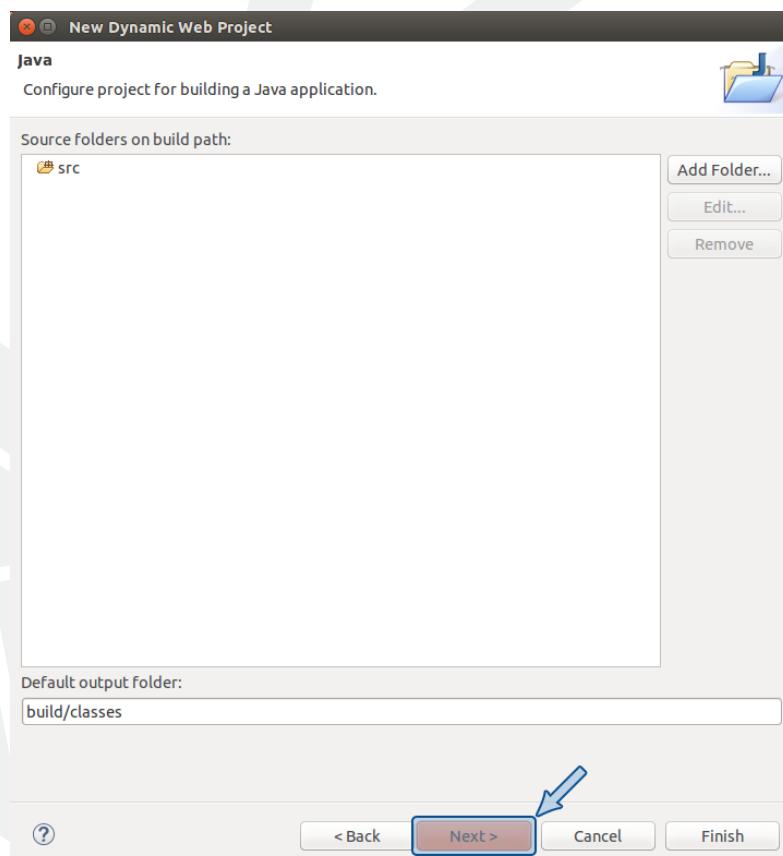
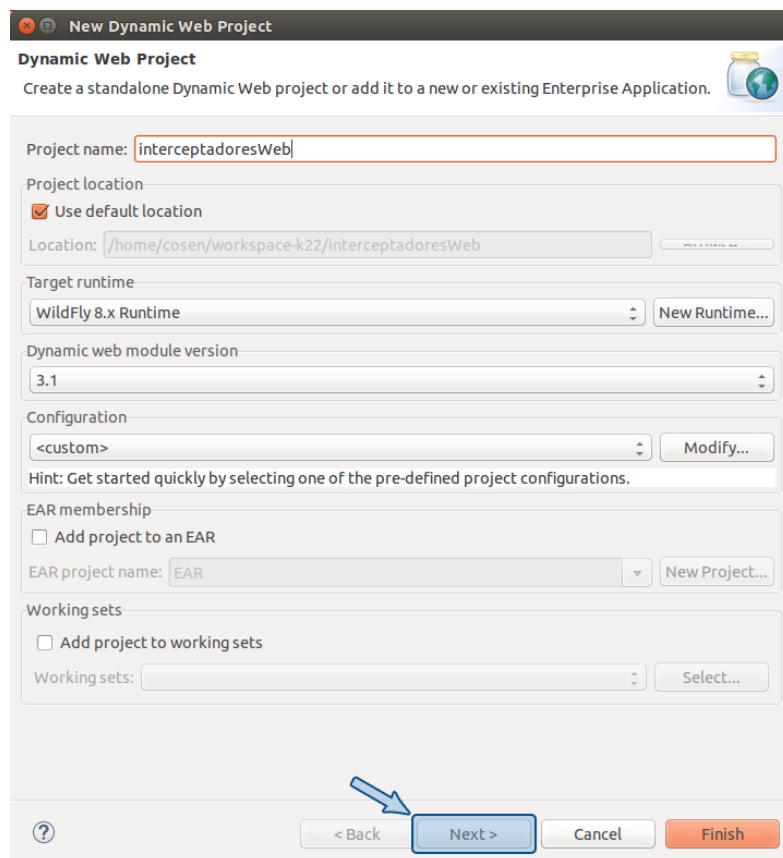
Quando dois ou mais Method-Level Interceptors estão associados a um método de negócio, eles serão executados na ordem em que foram declarados na anotação **@Interceptors**.

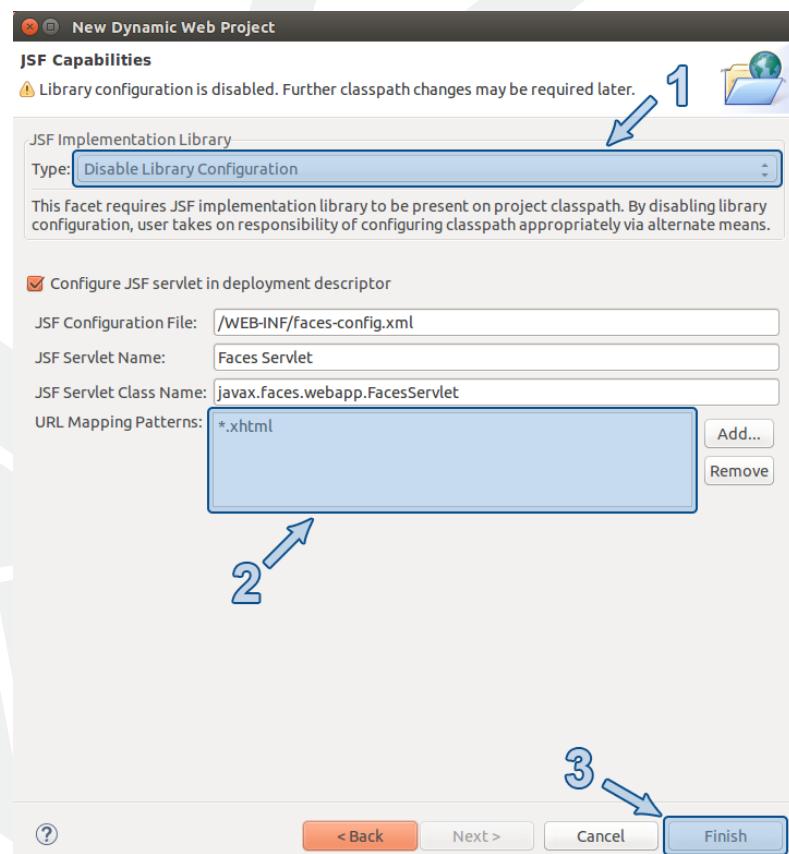
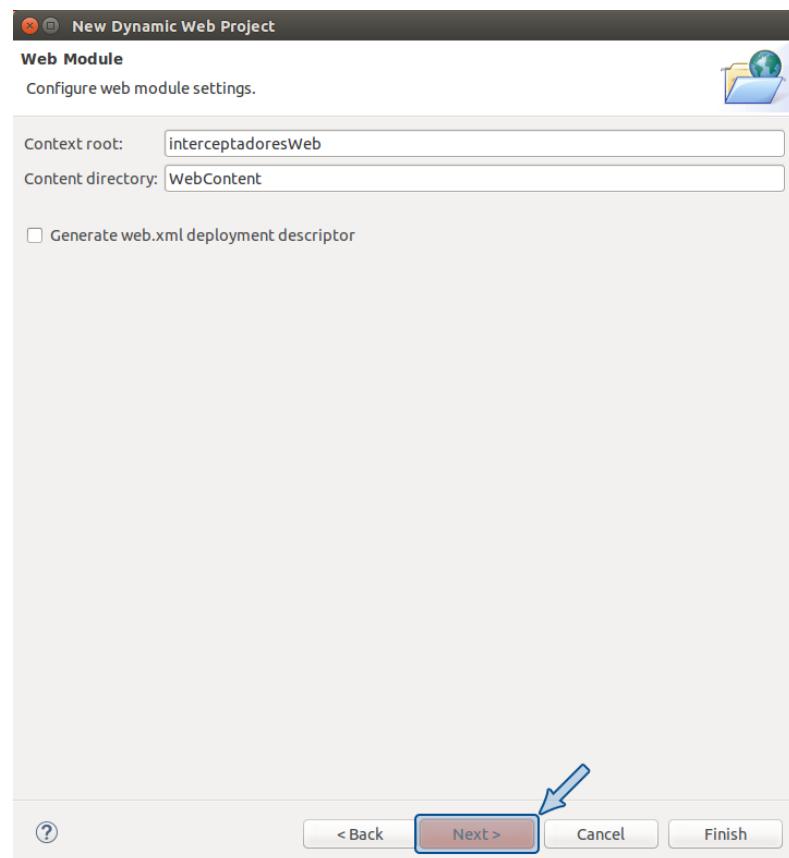


Exercícios de Fixação

- 1 Para não confundir, feche o projeto **segurançaWeb**. Para isso, clique com o botão direito do mouse sobre esse projeto e selecione a opção “Close Project”.
- 2 Crie um Dynamic Web Project no eclipse chamado **interceptadoresWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.







- 3** Adicione uma pasta chamada **META-INF** na pasta **src** do projeto **interceptadoresWeb**. Na pasta **META-INF**, crie o arquivo **persistence.xml**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="create"/>
15    </properties>
16  </persistence-unit>
17 </persistence>
```

Código XML 8.1: persistence.xml

- 4** Crie um pacote chamado **br.com.k19.entidades** no projeto **interceptadoresWeb** e adicione nesse pacote um Entity Bean para modelar mensagens.

```

1 package br.com.k19.entidades;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Mensagem {
9
10   @Id @GeneratedValue
11   private Long id;
12
13   private String texto;
14
15   // GETTERS AND SETTERS
16 }
```

Código Java 8.12: Mensagem.java

- 5** Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **interceptadoresWeb** e adicione nesse pacote um SLSB para funcionar como repositório de mensagens

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.ejb.Stateless;
6 import javax.persistence.EntityManager;
7 import javax.persistence.PersistenceContext;
8 import javax.persistence.TypedQuery;
9
10 import br.com.k19.entidades.Mensagem;
11
12 @Stateless
13 public class MensagemRepositorio {
```

```

15  @PersistenceContext
16  private EntityManager manager;
17
18  public void adiciona(Mensagem mensagem) {
19      this.manager.persist(mensagem);
20  }
21
22  public List<Mensagem> getMensagens() {
23      TypedQuery<Mensagem> query = this.manager.createQuery(
24          "select x from Mensagem x", Mensagem.class);
25
26      return query.getResultList();
27  }
28 }
```

Código Java 8.13: MensagemRepository.java

- 6** Crie um pacote chamado **br.com.k19.managedbeans** no projeto **interceptadoresWeb** e adicione nesse pacote um Managed Bean para oferecer algumas ações para as telas.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.ejb.EJB;
6 import javax.faces.bean.ManagedBean;
7
8 import br.com.k19.entidades.Mensagem;
9 import br.com.k19.sessionbeans.MensagemRepository;
10
11 @ManagedBean
12 public class MensagemMB {
13
14     @EJB
15     private MensagemRepository repositorio;
16
17     private Mensagem mensagem = new Mensagem();
18
19     private List<Mensagem> mensagensCache;
20
21     public void adiciona(){
22         this.repositorio.adiciona(this.mensagem);
23         this.mensagem = new Mensagem();
24         this.mensagensCache = null;
25     }
26
27     public List<Mensagem> getMensagens(){
28         if(this.mensagensCache == null){
29             this.mensagensCache = this.repositorio.getMensagens();
30         }
31         return this.mensagensCache;
32     }
33
34     public Mensagem getMensagem() {
35         return mensagem;
36     }
37
38     public void setMensagem(Mensagem mensagem) {
39         this.mensagem = mensagem;
40     }
41 }
```

Código Java 8.14: MensagemMB.java

- 7 Crie uma tela para cadastrar mensagens. Adicione na pasta **WebContent** do projeto **interceptadoresWeb** um arquivo chamado **mensagens.xhtml** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10  <title>Mensagens</title>
11 </h:head>
12
13 <h:body>
14  <h1>Nova Mensagem</h1>
15  <h:form>
16    <h:outputLabel value="Mensagem: "/>
17    <h:inputText value="#{mensagemMB.mensagem.texto}" />
18
19    <h:commandButton action="#{mensagemMB.adiciona}" value="Salvar"/>
20  </h:form>
21
22  <h1>Lista de Mensagens</h1>
23  <h:dataTable value="#{mensagemMB.mensagens}" var="mensagem">
24    <h:column>
25      <h:outputText value="#{mensagem.id}" />
26    </h:column>
27    <h:column>
28      <h:outputText value="#{mensagem.texto}" />
29    </h:column>
30  </h:dataTable>
31 </h:body>
32 </html>
```

Código XHTML 8.3: *mensagens.xhtml*

- 8 Implemente um interceptador externo para realizar o logging da aplicação. Crie um pacote chamado **br.com.k19.interceptadores** no projeto **interceptadoresWeb** e adicione nesse pacote a seguinte classe.

```

1 package br.com.k19.interceptadores;
2
3 import javax.interceptor.AroundInvoke;
4 import javax.interceptor.InvocationContext;
5
6 public class LoggingInterceptor {
7
8     @AroundInvoke
9     public Object interceptador(InvocationContext ic) throws Exception {
10         System.out.println("CHAMANDO O MÉTODO: " + ic.getMethod());
11
12         Object retornoDoMetodoDeNegocio = ic.proceed();
13
14         System.out.println("MÉTODO " + ic.getMethod() + " FINALIZADO");
15         return retornoDoMetodoDeNegocio;
16     }
17 }
```

Código Java 8.15: *LoggingInterceptor.java*

- 9 Adicione na pasta **WEB-INF** do projeto **interceptadoresWeb** o arquivo **ejb-jar.xml** com o se-

guinte conteúdo.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <ejb-jar version="3.1" xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/←
      javaee/ejb-jar_3_1.xsd">
5
6   <interceptors>
7     <interceptor>
8       <interceptor-class>
9         br.com.k19.interceptadores.LoggingInterceptor
10      </interceptor-class>
11    </interceptor>
12  </interceptors>
13
14  <assembly-descriptor>
15    <interceptor-binding>
16      <ejb-name>*</ejb-name>
17      <interceptor-class>
18        br.com.k19.interceptadores.LoggingInterceptor
19      </interceptor-class>
20    </interceptor-binding>
21  </assembly-descriptor>
22</ejb-jar>
```

Código XHTML 8.4: ejb-jar.xml

- 10** Remova o projeto **segurançaWeb** do **Glassfish**. Adicione o projeto **interceptadoresWeb** no **Wildfly**. Certifique-se que o **Glassfish** e o **JBoss** estejam parados. Inicie o **Wildfly** e teste a aplicação acessando a url <http://localhost:8080/interceptadoresWeb/mensagens.xhtml>. Verifique o log do **Wildfly** para observar as mensagens do interceptador.

- 11** Implemente um interceptador externo para eliminar palavras proibidas das mensagens adicionadas o logging da aplicação. Adicione no pacote **br.com.k19.interceptadores** do projeto **interceptadoresWeb** a seguinte classe.

```

1 package br.com.k19.interceptadores;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.interceptor.AroundInvoke;
7 import javax.interceptor.InvocationContext;
8
9 import br.com.k19.entidades.Mensagem;
10
11 public class CensuraInterceptor {
12
13   private List<String> palavrasProibidas = new ArrayList<String>();
14
15   public CensuraInterceptor(){
16     this.palavrasProibidas.add("coca-cola");
17     this.palavrasProibidas.add("fiat");
18     this.palavrasProibidas.add("sony");
19   }
20
21   @AroundInvoke
22   public Object interceptador(InvocationContext ic) throws Exception {
23     Object[] parameters = ic.getParameters();
24     Mensagem mensagem = (Mensagem)parameters[0];
25
26     for (String palavraProibida : this.palavrasProibidas) {
```

```

27     String textoOriginal = mensagem.getTexto();
28     String textoCensurado = textoOriginal.replaceAll(palavraProibida, "!CENSURADO!" );
29     mensagem.setTexto(textoCensurado);
30 }
31 return ic.proceed();
32 }
33 }
```

Código Java 8.16: CensuraInterceptor.java

- 12** Associe o interceptador de censura ao método de adicionar mensagens do session bean **MensagemRepository**. Não esqueça de adicionar o import da anotação **import javax.interceptor.Interceptors**.

```

1 ...
2 @Interceptors({CensuraInterceptor.class})
3 public void adiciona(Mensagem mensagem) {
4     this.manager.persist(mensagem);
5 }
6 ...
```

Código Java 8.17: MensagemRepository.java

- 13** Altere o arquivo **persistence.xml** do projeto **interceptadoresWeb**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="none"/>
15    </properties>
16  </persistence-unit>
17 </persistence>
```

Código XML 8.2: persistence.xml

- 14** Adicione mensagens com palavras proibidas através da url <http://localhost:8080/interceptadoresWeb/mensagens.xhtml>.



SCHEDULING

Algumas aplicações possuem a necessidade de agendar tarefas para serem executadas periodicamente ou uma única vez após um determinado tempo. Por exemplo, suponha uma aplicação que calcula o salário dos funcionários de uma empresa de acordo com as horas registradas. Possivelmente, esse cálculo deve ser realizado uma vez por mês.

Outro exemplo, suponha que uma empresa vende seus produtos através da internet. As entregas só são realizadas após a confirmação dos pagamentos. Quando um cliente realiza um pedido, o sistema da empresa deve esperar alguns dias para verificar se o pagamento correspondente foi realizado para que a entrega possa ser liberada.



Timers

Para agendar tarefas, podemos criar alarmes (timers) através do **TimerService**. Por exemplo, suponha que seja necessário executar uma tarefa uma única vez depois de 30 minutos.

```
1 timerService.createTimer(30 * 60 * 1000, "info");
```

O primeiro parâmetro do método `createTimer()` é quantidade de tempo que ele deve esperar para “disparar” e o segundo é uma informação que podemos associar ao timer. Também, podemos criar um alarme periódico que “dispara” a cada 30 minutos através do `TimerService` utilizando a sobrecarga do método `createTimer()`.

```
1 timerService.createTimer(30 * 60 * 1000, 30 * 60 * 1000, "info");
```

Podemos obter o `TimerService` por injeção através da anotação **@Resource**.

```
1 @Stateless
2 class FolhaDePagamentoBean {
3
4     @Resource
5     private TimerService timerService;
6
7     ...
8 }
```

Código Java 9.3: FolhaDePagamentoBean.java

Os alarmes não podem ser criados para Stateful Session Beans. Essa funcionalidade deve ser adicionada em versões futuras da especificação Enterprise Java Beans.



Métodos de Timeout

Quando um alarme (timer) “dispara”, o EJB Container executa um método de timeout no Bean que criou o alarme. Para definir um método de timeout devemos utilizar a anotação **@Timeout**.

```

1  @Stateless
2  class PedidoBean {
3
4      @Resource
5      private TimerService timerService;
6
7      public void registraPedido(Pedido pedido){
8          this.timerService.createTimer(5 * 24 * 60 * 60 * 1000, pedido);
9      }
10
11     @Timeout
12     public void verificaPagamento(Timer timer){
13         Pedido pedido = (Pedido)timer.getInfo();
14         // verifica o pagamento do pedido
15     }
16 }
```

Código Java 9.4: PedidoBean.java



Timers Automáticos

Na versão 3.1 da especificação Enterprise Java Beans, os alarmes podem ser criados e automaticamente associados a métodos de timeout através da anotação **@Schedule**.

```

1  @Stateless
2  class FolhaDePagamentoBean {
3
4      @Schedule(dayOfMonth="1")
5      public void calculaSalarios(){
6          // implementacao
7      }
8 }
```

Código Java 9.5: FolhaDePagamentoBean.java

Os atributos da anotação **@Schedule** são:

| Atributo | Valores |
|------------|--|
| second | [0 ... 59] |
| minute | [0 ... 59] |
| hour | [0 ... 23] |
| dayOfMonth | [1 ... 31] [-7 ... -1] quantidade de dias para o término do mês. [1st, 2nd, 3rd, 4th, 5th, Last] [Sun, Mon, Tue, Wed, Thu, Fri, Sat] |
| month | [1 ... 12] [Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec] |
| dayOfWeek | [0 ... 7] [Sun, Mon, Tue, Wed, Thu, Fri, Sat] |
| year | ano com 4 dígitos |

Podemos utilizar lista de valores nos atributo da anotação **@Schedule**.

```
1 @Schedule(minute="0, 15, 30, 45")
```

Também podemos utilizar intervalos de valores.

```
1 @Schedule(minute="0-10")
```

Utilizando o wildcard *, todos os valores possíveis são associados a um atributo da anotação @Schedule.

```
1 @Schedule(second="*", minute="*", hour="*")
```

Podemos aplicar expressões aos atributos second, minute e hour.

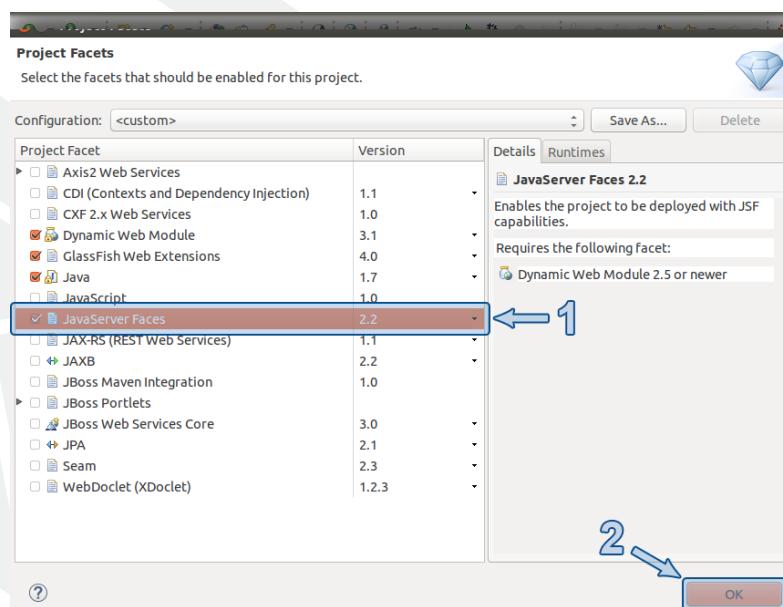
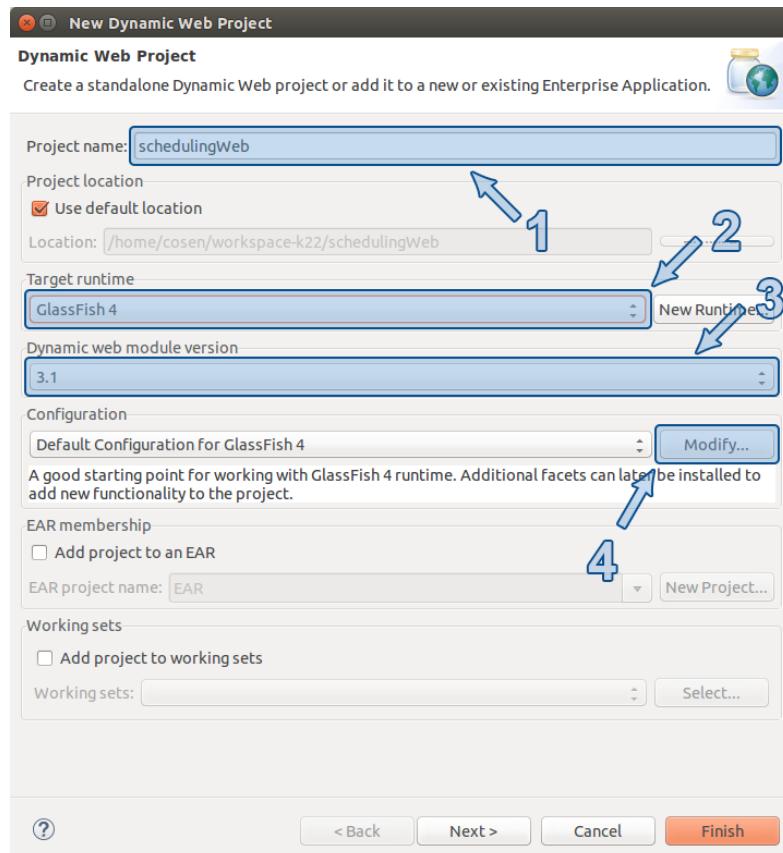
```
1 // equivale a @Schedule(second="0, 15, 30, 45")
2 @Schedule(second="*/15")
```

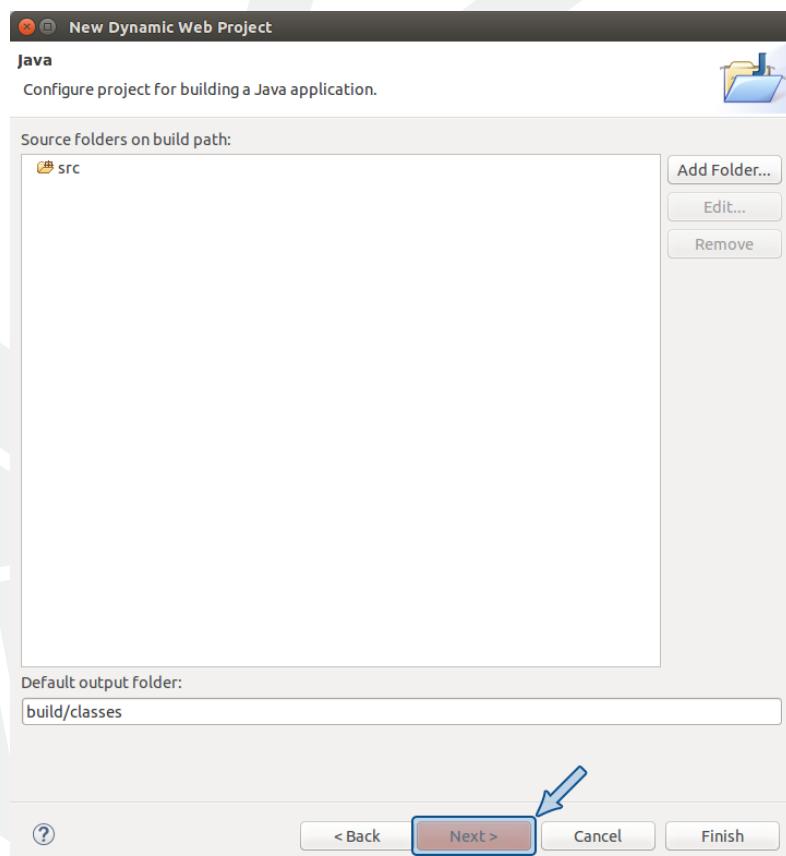
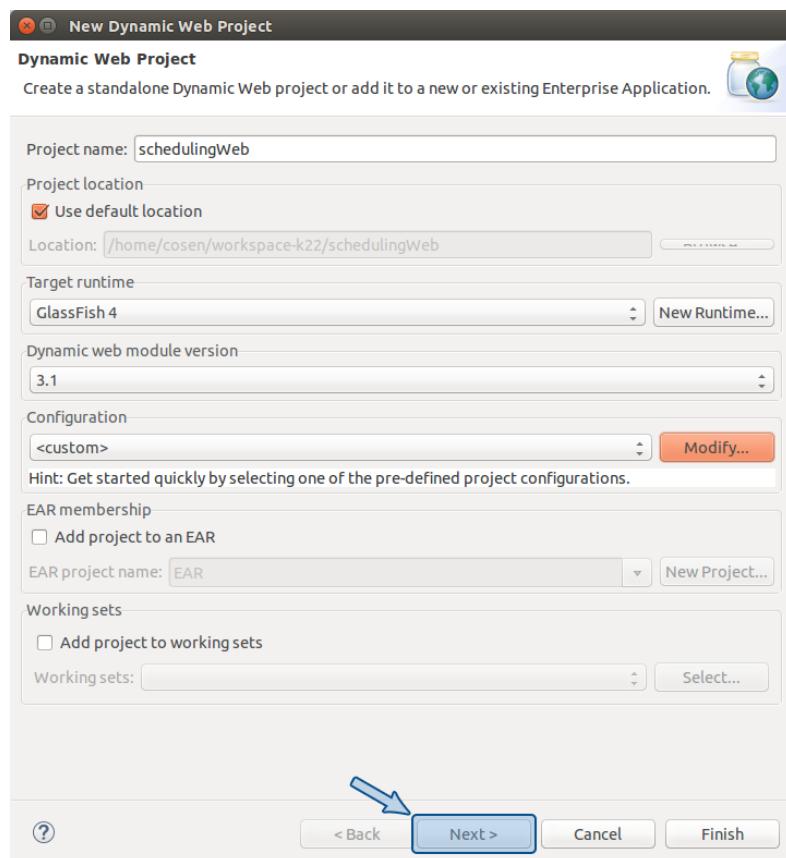
```
1 // equivale a @Schedule(second="30, 35, 40, 45, 50, 55")
2 @Schedule(second="30/5")
```

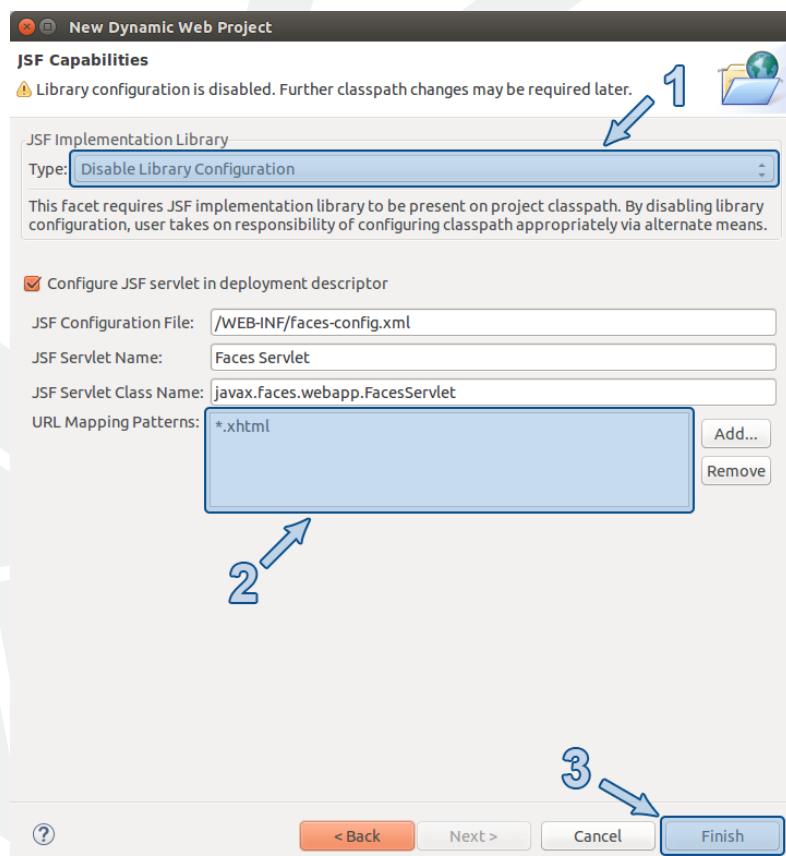
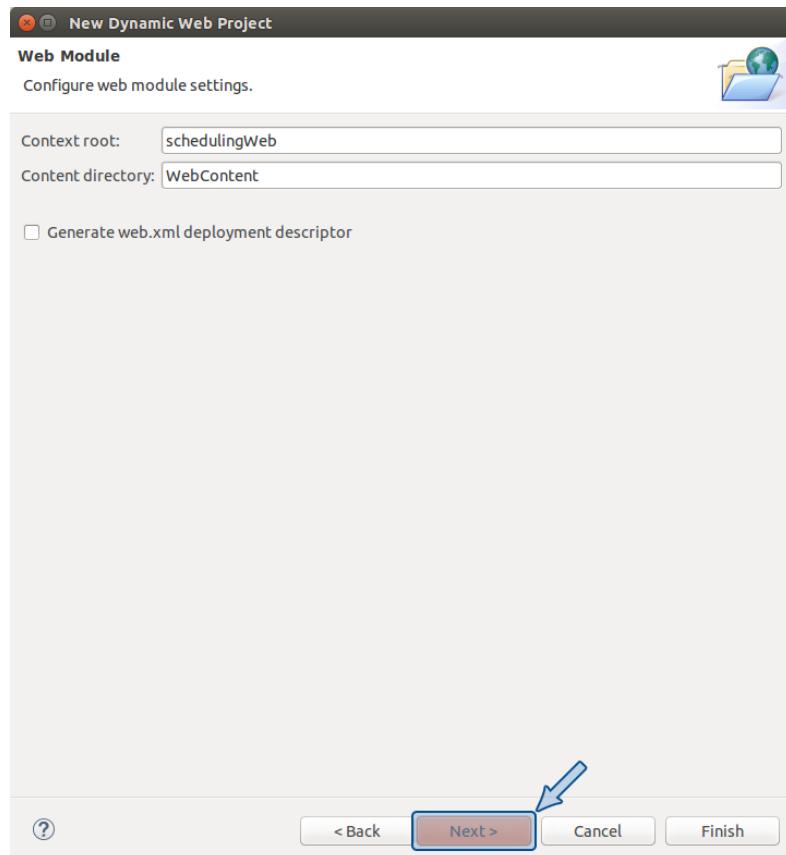


Exercícios de Fixação

- 1 Para não confundir, feche os projetos **interceptadoresWeb**. Para isso, clique com o botão direito do mouse sobre esse projeto e selecione a opção “Close Project”.
- 2 Crie um Dynamic Web Project no eclipse chamado **schedulingWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.







- 3 Adicione uma pasta chamada **META-INF** na pasta **src** do projeto **schedulingWeb**. Na pasta **META-INF**, crie o arquivo **persistence.xml**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="create"/>
15    </properties>
16  </persistence-unit>
17 </persistence>
```

Código XML 9.1: *persistence.xml*

- 4 Crie um pacote chamado **br.com.k19.entidades** no projeto **schedulingWeb** e adicione nesse pacote um Entity Bean para modelar produtos.

```

1 package br.com.k19.entidades;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Produto {
9
10   @Id @GeneratedValue
11   private Long id;
12
13   private String nome;
14
15   private double preco;
16
17   // GETTERS AND SETTERS
18 }
```

Código Java 9.11: *Produto.java*

- 5 Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **schedulingWeb** e adicione nesse pacote um SLSB para funcionar como repositório de produtos.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.ejb.Stateless;
6 import javax.persistence.EntityManager;
7 import javax.persistence.PersistenceContext;
8 import javax.persistence.TypedQuery;
9
10 import br.com.k19.entidades.Produto;
11
12 @Stateless
```

```

13 public class ProdutoRepositorio {
14
15     @PersistenceContext
16     private EntityManager manager;
17
18     public void adiciona(Produto produto) {
19         this.manager.persist(produto);
20     }
21
22     public List<Produto> getProdutos() {
23         TypedQuery<Produto> query = this.manager.createQuery(
24             "select x from Produto x", Produto.class);
25
26         return query.getResultList();
27     }
28 }
```

Código Java 9.12: ProdutoRepositorio.java

- 6** Crie um pacote chamado **br.com.k19.managedbeans** no projeto **schedulingWeb** e adicione nesse pacote um Managed Bean para oferecer algumas ações para as telas.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.ejb.EJB;
6 import javax.faces.bean.ManagedBean;
7
8 import br.com.k19.entidades.Produto;
9 import br.com.k19.sessionbeans.ProdutoRepositorio;
10
11 @ManagedBean
12 public class ProdutoMB {
13
14     @EJB
15     private ProdutoRepositorio repositorio;
16
17     private Produto produto = new Produto();
18
19     private List<Produto> produtosCache;
20
21     public void adiciona(){
22         this.repositorio.adiciona(this.produto);
23         this.produto = new Produto();
24         this.produtosCache = null;
25     }
26
27     public List<Produto> getProdutos(){
28         if(this.produtosCache == null){
29             this.produtosCache = this.repositorio.getProdutos();
30         }
31         return this.produtosCache;
32     }
33
34     public void setProduto(Produto produto) {
35         this.produto = produto;
36     }
37
38     public Produto getProduto() {
39         return produto;
40     }
41 }
```

Código Java 9.13: ProdutoMB.java

- 7 Crie uma tela para cadastrar produtos. Adicione na pasta **WebContent** do projeto **schedulingWeb** um arquivo chamado **produtos.xhtml** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10  <title>Produtos</title>
11 </h:head>
12
13 <h:body>
14  <h1>Novo Produto</h1>
15  <h:form>
16    <h:outputLabel value="Nome: "/>
17    <h:inputText value="#{produtoMB.produto.nome}" />
18
19    <h:outputLabel value="Preço: "/>
20    <h:inputText value="#{produtoMB.produto.preco}" />
21
22    <h:commandButton action="#{produtoMB.adiciona}" value="Salvar"/>
23 </h:form>
24
25  <h1>Lista de Produtos</h1>
26  <h:dataTable value="#{produtoMB.produtos}" var="produto">
27    <h:column>
28      <h:outputText value="#{produto.nome}" />
29    </h:column>
30    <h:column>
31      <h:outputText value="#{produto.preco}" />
32    </h:column>
33  </h:dataTable>
34 </h:body>
35 </html>
```

Código XHTML 9.1: *produtos.xhtml*

- 8 Remova o projeto **interceptadoresWeb** do **Wildfly**. Adicione o projeto **schedulingWeb** no **Glassfish**. Certifique-se que o **JBoss** e o **Wildfly** estejam parados. Inicie o **Glassfish** e teste a aplicação acessando a url <http://localhost:8080/schedulingWeb/produtos.xhtml>.
- 9 Periodicamente um produto cadastrado deve ser escolhido e colocado em destaque. Implemente essa lógica através dos recursos de scheduling. Adicione a seguinte classe no pacote **br.com.k19.sessionbeans** do projeto **schedulingWeb**.

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4 import java.util.Random;
5
6 import javax.ejb.EJB;
7 import javax.ejb.Schedule;
8 import javax.ejb.Singleton;
9
10 import br.com.k19.entidades.Produto;
11
12 @Singleton
13 public class ProdutoDestaqueBean {
```

```

15  @EJB
16  private ProdutoRepositorio repositorio;
17
18  private Produto produtoDestaque;
19
20  @Schedule(second="*/5", minute="*", hour="*", persistent=false)
21  public void trocaProdutoDestaque(){
22      Random gerador = new Random();
23      List<Produto> produtos = this.repositorio.getProdutos();
24      int i = gerador.nextInt(produtos.size());
25      this.produtoDestaque = produtos.get(i);
26  }
27
28  public void setProdutoDestaque(Produto produtoDestaque) {
29      this.produtoDestaque = produtoDestaque;
30  }
31
32  public Produto getProdutoDestaque() {
33      return produtoDestaque;
34  }
35 }
```

Código Java 9.14: ProdutoDestaqueBean.java

- 10 Adicione na classe **ProdutoMB** do projeto **schedulingWeb** o seguinte atributo.

```

1 ...
2 @EJB
3 private ProdutoDestaqueBean produtoDestaqueBean;
4 ...
```

Código Java 9.15: ProdutoMB.java

- 11 Adicione na classe **ProdutoMB** do projeto **schedulingWeb** o seguinte método.

```

1 ...
2 public Produto getProdutoDestaque(){
3     return this.produtoDestaqueBean.getProdutoDestaque();
4 }
5 ...
```

Código Java 9.16: ProdutoMB.java

- 12 Acrescente na tela **produtos.xhtml** do projeto **schedulingWeb** o produto em destaque.

```

1 ...
2 <h1>Produto Destaque</h1>
3
4 <h:outputLabel value="Nome: "/>
5 <h:outputText value="#{produtoMB.produtoDestaque.nome}" />
6 <h:outputLabel value="Preço: "/>
7 <h:outputText value="#{produtoMB.produtoDestaque.preco}" />
8 ...
```

Código XHTML 9.2: produtos.xhtml

- 13 Altere o arquivo **persistence.xml** do projeto **schedulingWeb**.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5   http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="none"/>
15    </properties>
16  </persistence-unit>
17 </persistence>
```

Código XML 9.2: persistence.xml

- 14 Acesse periodicamente a url <http://localhost:8080/schedulingWeb/produtos.xhtml> para verificar que o produto em destaque muda de tempos em tempos.



CONTEXTS AND DEPENDENCY INJECTION - CDI

Aplicações corporativas costumam utilizar tanto o container WEB para a camada de apresentação quanto o container EJB para a camada de negócio. A integração entre o container WEB e o container EJB pode ser mais facilmente realizada através dos recursos definidos pela especificação **Contexts and Dependency Injection - CDI**.

Dos recursos existentes na arquitetura CDI, podemos destacar o mecanismo de Injeção de Dependência e o gerenciamento do ciclo de vida dos objetos através de contextos. De acordo com a especificação CDI, os seguintes tipos de objetos possuem suporte a esses dois recursos:

- Managed Beans
- Session Beans
- Objetos criados por Producer Methods
- Objetos disponibilizados por Producer Fields
- Resources (Java EE resources, Persistence Contexts, Persistence Units, Remote EJBs e Web Services)



Managed Beans

Na arquitetura Java EE, os Managed Beans são objetos gerenciados pelo container Java EE. O container deve oferecer um pequeno conjunto de serviços fundamentais aos Managed Beans.

A definição básica do conceito de Managed Beans está documentada na especificação **Java EE 6 Managed Beans**. Contudo essa especificação permite que outras especificações estendam a idéia original de Managed Beans.

Não devemos confundir o conceito de Managed Beans do Java EE com o conceito de Managed Bean do JSF. Na verdade, um Managed Bean do JSF é um caso particular de um Managed Bean do Java EE.

A especificação CDI estende a definição de Managed Beans. Na arquitetura CDI, os Managed Beans são definidos por classes que devem respeitar certas restrições. Na seção 3.1.1 da especificação CDI são definidas essas restrições.

A top-level Java class is a managed bean if it is defined to be a managed bean by any other Java EE specification, or if it meets all of the following conditions:

- It is not a non-static inner class.
- It is a concrete class, or is annotated `@Decorator`.
- It is not annotated with an EJB component-defining annotation or declared as an EJB bean class in `ejb-jar.xml`.
- It does not implement `javax.enterprise.inject.spi.Extension`.
- It has an appropriate constructor—either:
 - the class has a constructor with no parameters, or
 - the class declares a constructor annotated `@Inject`.

All Java classes that meet these conditions are managed beans and thus no special declaration is required to define a managed bean.

As classes que se encaixam nessas restrições atuam como fonte de objetos que serão administrados pelo container CDI e poderão ser injetados em outros objetos.



Producer Methods and Fields

Os Producer Methods são apenas métodos que produzem objetos que serão administrados pelo container CDI e injetados em outros objetos. Os Producer Methods devem ser anotados com `@Produces`.

```
1 @Produces
2 public List<Produto> listaProdutos() {
3     // implementacao
4 }
```

Atributos também podem ser utilizados como fonte de objetos para o container Java EE. Os Producer Fields devem ser anotados com `@Produces`.

```
1 @Produces
2 public List<Produto> produtos;
```



EL Names

Na arquitetura CDI, páginas JSP ou JSF podem acessar objetos através de EL. Somente objetos com um **EL Name** podem ser acessados por páginas JSP ou JSF. A princípio, os seguintes tipos de objetos podem possuir um EL Name:

- Managed Beans

- Session Beans
- Objetos criados por Producer Methods
- Objetos disponibilizados por Producer Fields

Devemos aplicar a anotação **@Named** aos objetos que devem possuir um EL Name. Utilizando essa anotação, automaticamente, os objetos receberão um EL Name que é determinado de acordo com o tipo do objeto.

```
1 @Named // Managed Bean - EL Name: geradorDeApostas
2 public class GeradorDeApostas {
3     // implementacao
4 }
```

Código Java 10.3: GeradorDeApostas.java

```
1 @Named
2 @Stateless // Session Bean - EL Name: geradorDeApostas
3 public class GeradorDeApostas {
4     // implementacao
5 }
```

Código Java 10.4: GeradorDeApostas.java

```
1 @Named
2 @Produces // Producer Method - EL Name: listaProdutos
3 public List<Produto> listaProdutos() {
4     // implementacao
5 }
6
7 @Named
8 @Produces // Producer Method - EL Name: produtos
9 public List<Produto> getProdutos() {
10    // implementacao
11 }
```

```
1 @Named
2 @Produces // Producer Field - EL Name: produtos
3 public List<Produto> produtos;
```

É possível alterar o padrão definindo EL Names diretamente na anotação **@Named**.

```
1 @Named("gerador")
2 public class GeradorDeApostas {
3     // implementacao
4 }
```

Código Java 10.7: GeradorDeApostas.java



beans.xml

Para modificar as configurações do CDI, é necessário adicionar um arquivo chamado **beans.xml** na pasta **META-INF** no classpath.

```
1 <?xml version="1.0" encoding="UTF-8"?>
```

```

2 <beans xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="
5     http://java.sun.com/xml/ns/javaee
6     http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
7 </beans>

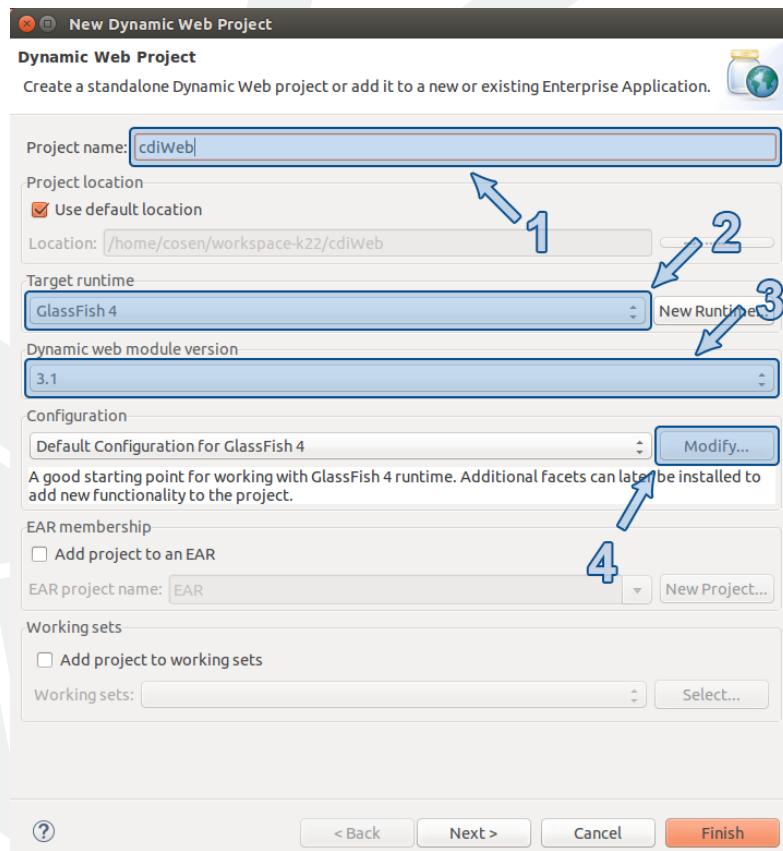
```

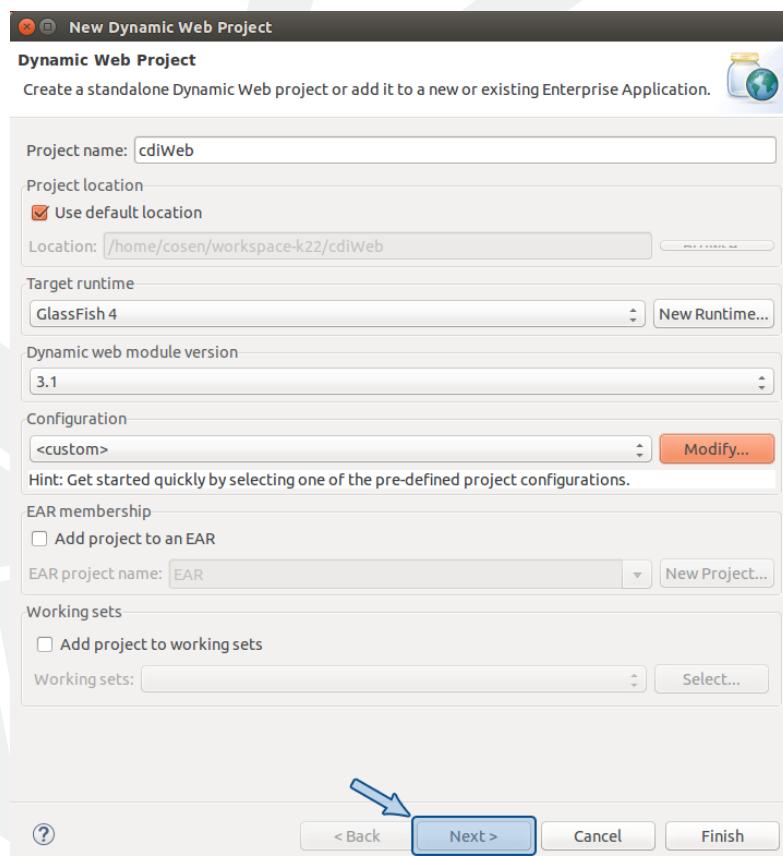
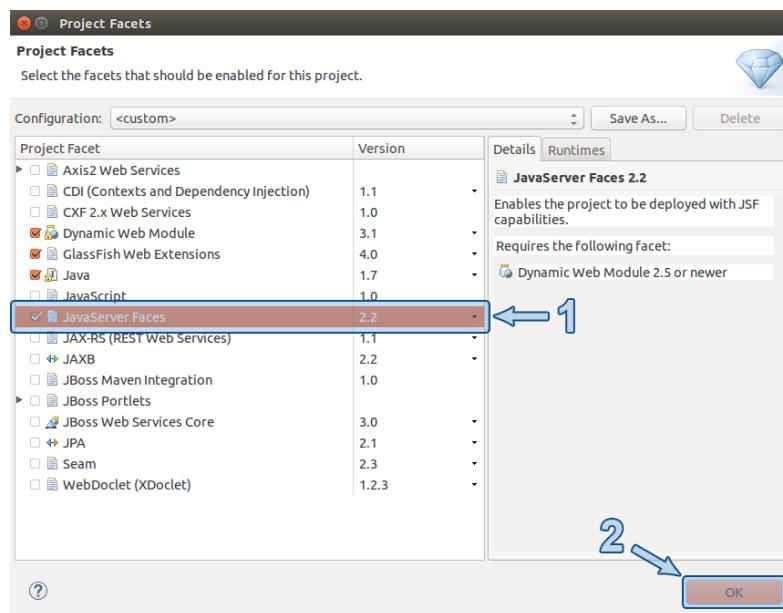
Código XML 10.1: beans.xml

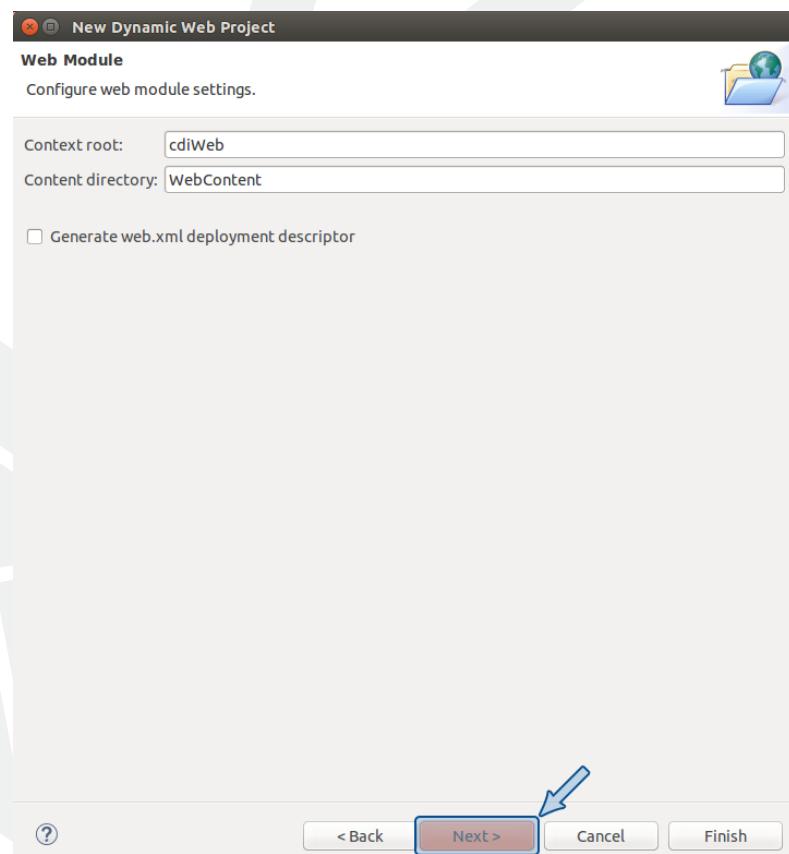
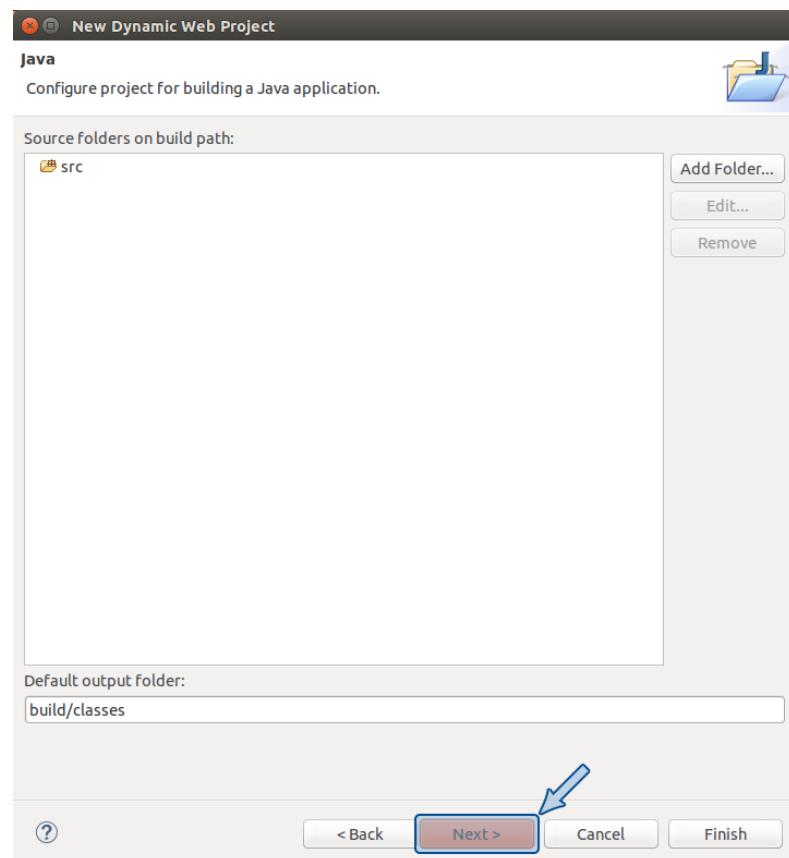


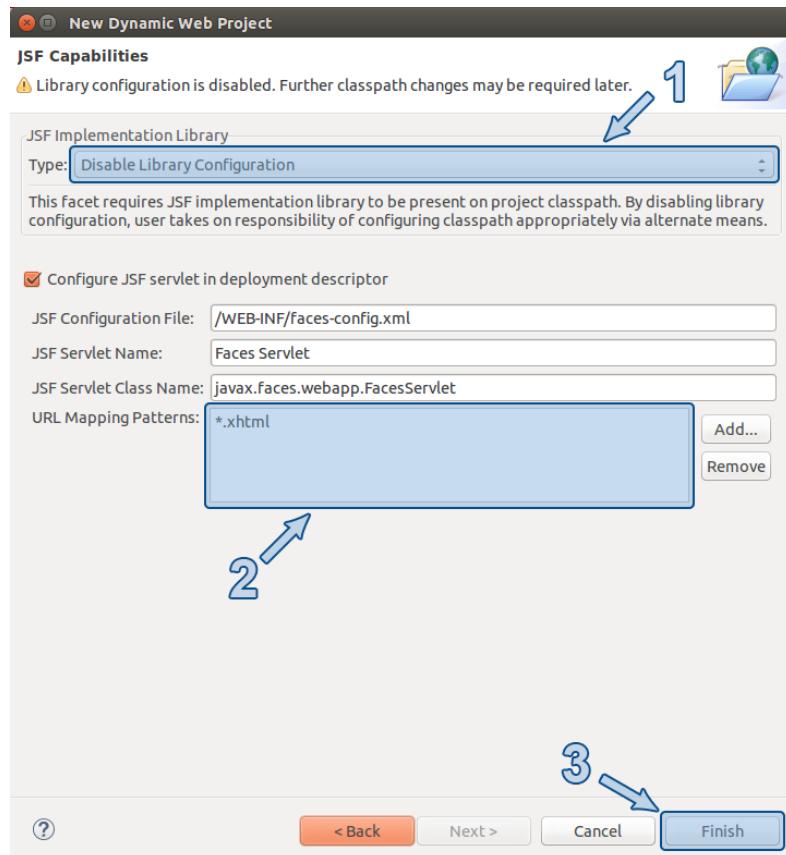
Exercícios de Fixação

- 1 Para não confundir, feche o projeto **schedulingWeb**. Para isso, clique com o botão direito do mouse sobre esse projeto e selecione a opção “Close Project”.
- 2 Crie um Dynamic Web Project no eclipse chamado **cdiWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.









- 3 Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **cdiWeb** e adicione nesse pacote um SLSB para funcionar como lançador de moeda.

```

1 package br.com.k19.sessionbeans;
2
3 import javax.ejb.Stateless;
4 import javax.inject.Named;
5
6 @Named
7 @Stateless
8 public class LancadorDeMoedaBean {
9
10    private String resultado;
11
12    public void lanca() {
13        if (Math.random() < 0.5) {
14            this.resultado = "CARA";
15        } else {
16            this.resultado = "COROA";
17        }
18    }
19
20    public void setResultado(String resultado) {
21        this.resultado = resultado;
22    }
23
24    public String getResultado() {
25        return resultado;
26    }
27 }
```

Código Java 10.8: LancadorDeMoedaBean.java

- 4 Crie uma tela para utilizar o lançador de moedas. Adicione na pasta **WebContent** do projeto **cdiWeb** um arquivo chamado **moeda.xhtml** com o seguinte conteúdo.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10  <title>Moeda</title>
11 </h:head>
12
13 <h:body>
14  <h1>Moeda</h1>
15  <h:form>
16    <h:commandButton action="#{lancadorDeMoedaBean.lanca}" value="Jogar"/>
17  </h:form>
18  <h2>Resultado: <h:outputText value="#{lancadorDeMoedaBean.resultado}" /></h2>
19 </h:body>
20 </html>
```

Código XHTML 10.1: *moeda.xhtml*

- 5 Remova o projeto **schedulingWeb** do **Glassfish**. Adicione o projeto **cdiWeb** no **Glassfish**. Certifique-se que o **JBoss** e o **Wildfly** estejam parado. Inicie o **Glassfish** e teste a aplicação acessando a url <http://localhost:8080/cdiWeb/moeda.xhtml>.

- 6 Crie um pacote chamado **br.com.k19.managedbeans** no projeto **cdiWeb** e adicione nesse pacote uma classe para gerar números aleatórios.

```

1 package br.com.k19.managedbeans;
2
3 import java.util.ArrayList;
4 import java.util.List;
5
6 import javax.enterprise.inject.Produces;
7 import javax.inject.Named;
8
9 public class GeradorDeNumeros {
10
11     @Named
12     @Produces
13     public List<Double> getNumeros(){
14         List<Double> numeros = new ArrayList<Double>();
15         for (int i = 0; i < 5; i++) {
16             numeros.add(Math.random());
17         }
18         return numeros;
19     }
20 }
```

Código Java 10.9: *GeradorDeNumeros.java*

- 7 Crie uma tela para utilizar o gerador de números. Adicione na pasta **WebContent** do projeto **cdiWeb** um arquivo chamado **numeros.xhtml** com o seguinte conteúdo.

```
1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
```

```

2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10  <title>Números</title>
11 </h:head>
12
13 <h:body>
14  <h1>Números</h1>
15
16 <h:dataTable value="#{numeros}" var="numero">
17  <h:column>
18    <h:outputText value="#{numero}" />
19  </h:column>
20 </h:dataTable>
21
22 </h:body>
23 </html>

```

Código XHTML 10.2: numeros.xhtml

- 8** Adicione o arquivo **beans.xml** na pasta **WEB-INF** do projeto **cdiWeb**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="
5     http://java.sun.com/xml/ns/javaee
6     http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
7 </beans>

```

Código XML 10.2: beans.xml

- 9** Acesse a url <http://localhost:8080/cdiWeb/numeros.xhtml>.



Escopos e Contextos

Os objetos administrados pelo container CDI são armazenados em **contextos**. Conceitualmente, um contexto é uma coleção de objetos relacionados logicamente que devem existir durante um período de tempo específico. A especificação CDI define quatro contextos padrões.

Request Context: Quando se trata de aplicações Java WEB, para cada requisição HTTP um novo Request Context é criado pelo container CDI e destruído no final do processamento da mesma requisição.

Múltiplos Request Contexts podem existir simultaneamente.

Session Context: Um Session Context está sempre associado a uma HTTP Session. Quando uma HTTP Session é criada pelo container WEB, o container CDI cria um Session Context associado a essa HTTP Session. Quando uma HTTP Session é destruída pelo container WEB, o container CDI também destrói o Session Context correspondente.

Múltiplos Session Contexts podem existir simultaneamente.

Application Context: O container CDI cria um Application Context quando a aplicação é inicializada e o destrói quando a aplicação é finalizada.

Múltiplos Application Contexts **não** podem existir simultaneamente.

Conversation Context: Há dois tipos de Conversation Context: **transient** e **long-running**. Um Conversation Context do tipo transient se comporta de maneira muito parecida com o Request Context. Basicamente, um Conversation do tipo long-running é criado na chamada do método `Conversation.begin()` e destruído quando o método `Conversation.end()` é executado.

Múltiplos Conversation Contexts podem existir simultaneamente.

Todo objeto administrado pelo container CDI possui um escopo. O escopo de um objeto define em qual contexto ele será armazenado quando criado pelo container CDI. A especificação CDI define cinco escopos padrões: Request, Session, Application, Conversation e Dependent.

Objetos com escopo Request, Session, Application e Conversation são armazenados no Request Context, Session Context, Application Context e Conversation Context respectivamente.

Um objeto com escopo Dependent pertence a outro objeto. O objeto dependente é armazenado indiretamente em algum contexto de acordo com o escopo do objeto a qual ele pertence.

As anotações: **@RequestScoped**, **@SessionScoped**, **@ApplicationScoped**, **@ConversationScoped** e **@Dependent** são utilizadas para definir o escopo dos objetos. Por padrão, se nenhuma anotação for definida o escopo dos objetos é o Dependent.

```

1 @RequestScoped
2 public class GeradorDeApostas {
3     // implementacao
4 }
```

Código Java 10.10: *GeradorDeApostas.java*

```

1 @Produces
2 @SessionScoped
3 public List<Produto> listaProdutos() {
4     // implementacao
5 }
```



Injection Points

Quando um objeto é criado pelo container CDI, todas as dependências são injetados pelo container nesse objeto. As dependências são outros objetos pertencentes ao mesmo contexto do objeto que está sendo criado. Se alguma dependência não estiver criada o container se encarrega de criá-la antes.

As dependências de um objeto são definidas através de **Injection Points**. Há três tipos de Injection Points:

Bean Constructors

As dependências de um objeto podem ser definidas através de construtores com a anotação `@Inject`.

```

1 public class CarrinhoDeCompras {
2
3     @Inject
4     public CarrinhoDeCompras(Usuario usuario) {
5
6     }
7 }
```

Código Java 10.12: CarrinhoDeCompras.java

Field

As dependências de um objeto podem ser definidas através de atributos com a anotação `@Inject`.

```

1 public class CarrinhoDeCompras {
2
3     @Inject
4     private Usuario usuario;
5 }
```

Código Java 10.13: CarrinhoDeCompras.java

Initializer methods

As dependências de um objeto podem ser definidas através de métodos inicializadores com a anotação `@Inject`.

```

1 public class CarrinhoDeCompras {
2
3     private Usuario usuario;
4
5     @Inject
6     public void setUsuario(Usuario usuario){
7         this.usuario = usuario;
8     }
9 }
```

Código Java 10.14: CarrinhoDeCompras.java



Exercícios de Fixação

- 10 Altere o método `getNumeros()` da classe **GeradorDeNumeros** do projeto **cdiWeb** para que ele adicione uma mensagem no console toda vez que for chamado.

```

1 ...
2 @Named
3 @Produces
4 public List<Double> getNumeros(){
5     System.out.println("GERANDO NÚMEROS");
6     List<Double> numeros = new ArrayList<Double>();
7     for (int i = 0; i < 5; i++) {
8         numeros.add(Math.random());
```

```
9     }
10    return numeros;
11 }
12 . . .
```

Código Java 10.15: GeradorDeNumeros.java

- 11 Acesse a url <http://localhost:8080/cdiWeb/numeros.xhtml> e depois observe as mensagens impressas no log do Glassfish.

- 12 Para evitar que o método **getNumeros** seja chamado mais do que uma vez por requisição HTTP, utilize o Resquet Scope. Não esqueça de importar a anotação **import javax.enterprise.context.RequestScoped;**.

```
1 . . .
2 @Named
3 @Produces
4 @RequestScoped
5 public List<Double> getNumeros(){
6     System.out.println("GERANDO NÚMEROS");
7     List<Double> numeros = new ArrayList<Double>();
8     for (int i = 0; i < 5; i++) {
9         numeros.add(Math.random());
10    }
11    return numeros;
12 }
13 . . .
```

Código Java 10.16: GeradorDeNumeros.java

- 13 Acesse novamente a url <http://localhost:8080/cdiWeb/numeros.xhtml> e depois observe as mensagens impressas no console do eclipse.



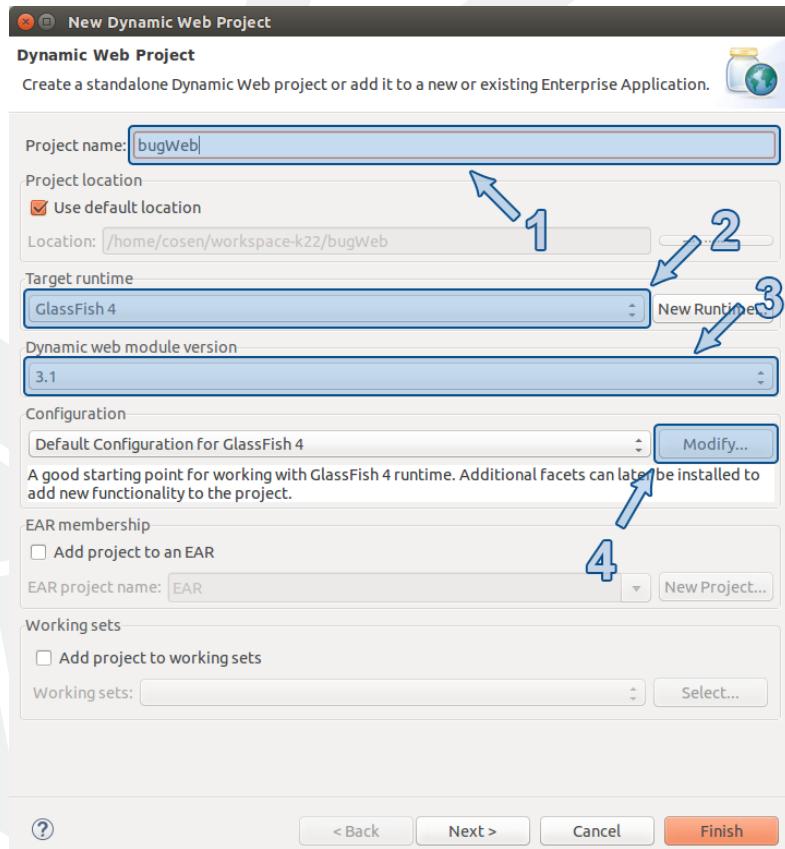
PROJETO

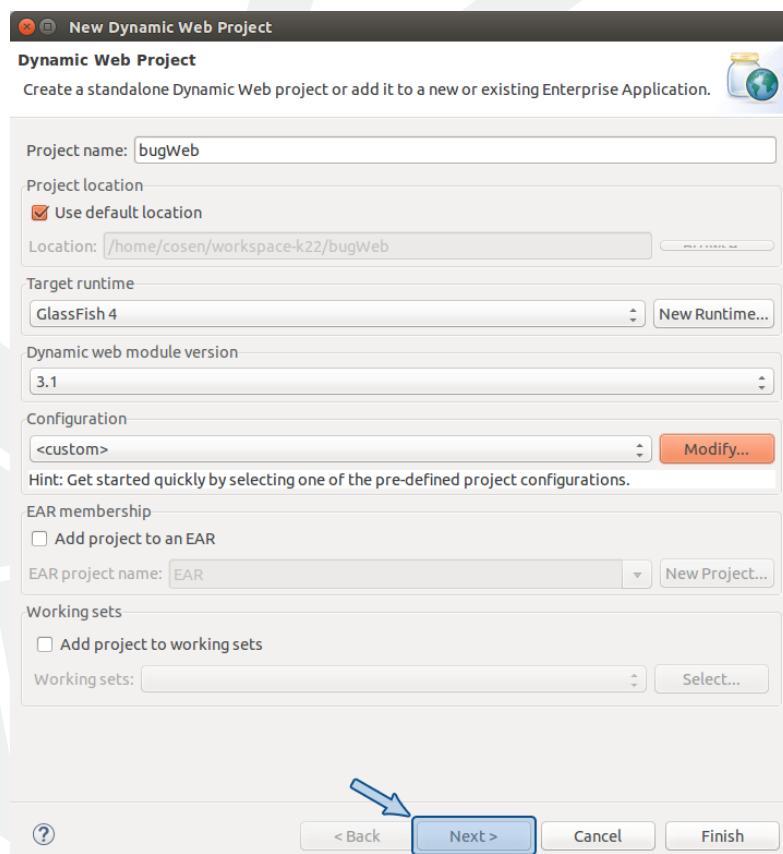
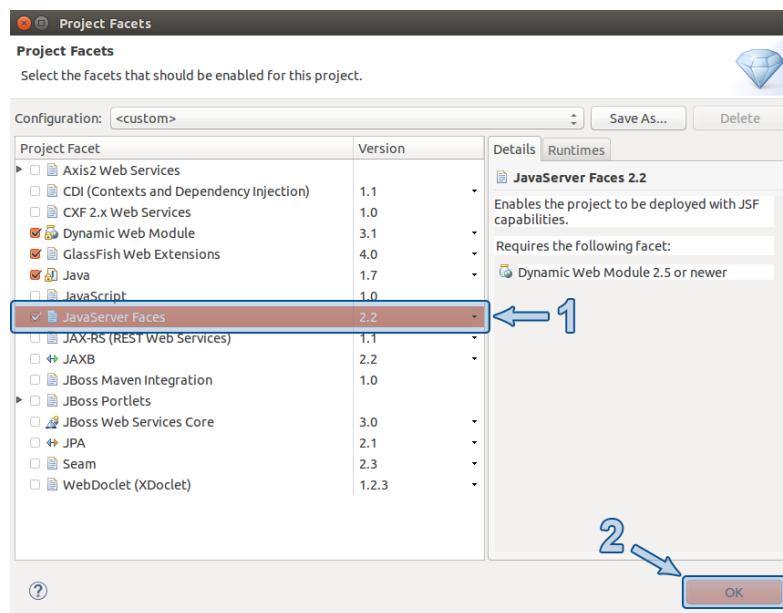
Neste capítulo, implementaremos um pequeno projeto para praticar os conceitos discutidos nos capítulos anteriores. Criaremos um sistema simples de cadastro de bugs.

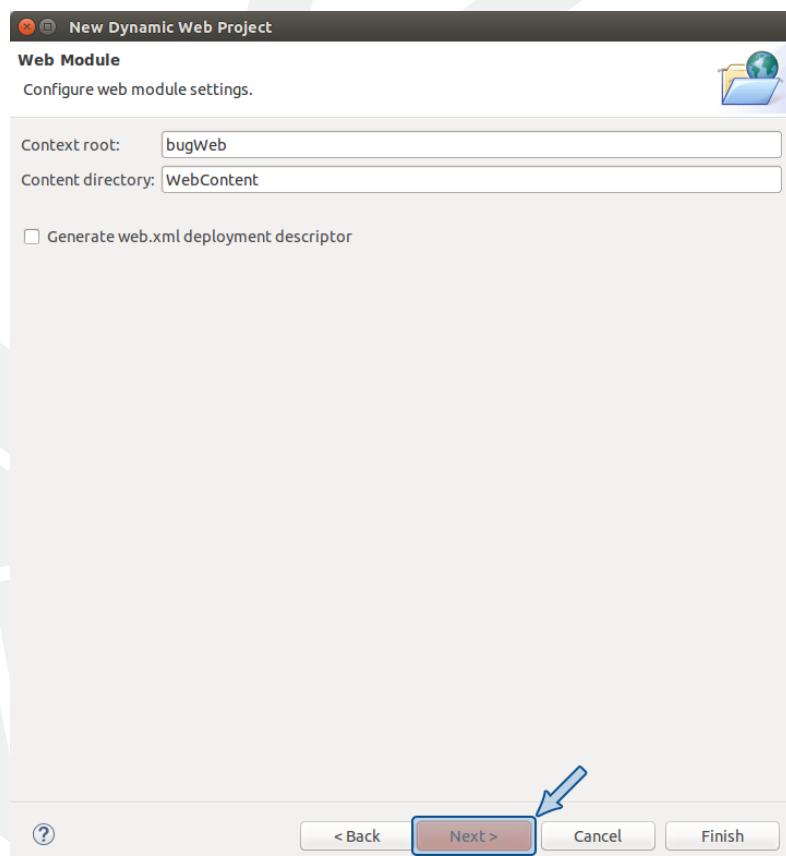
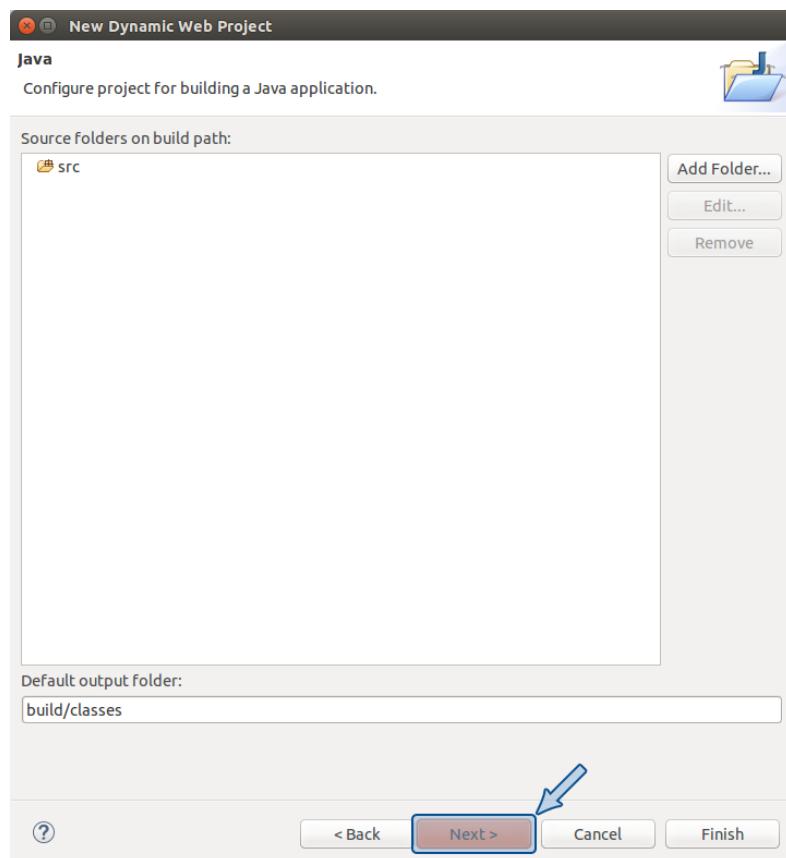


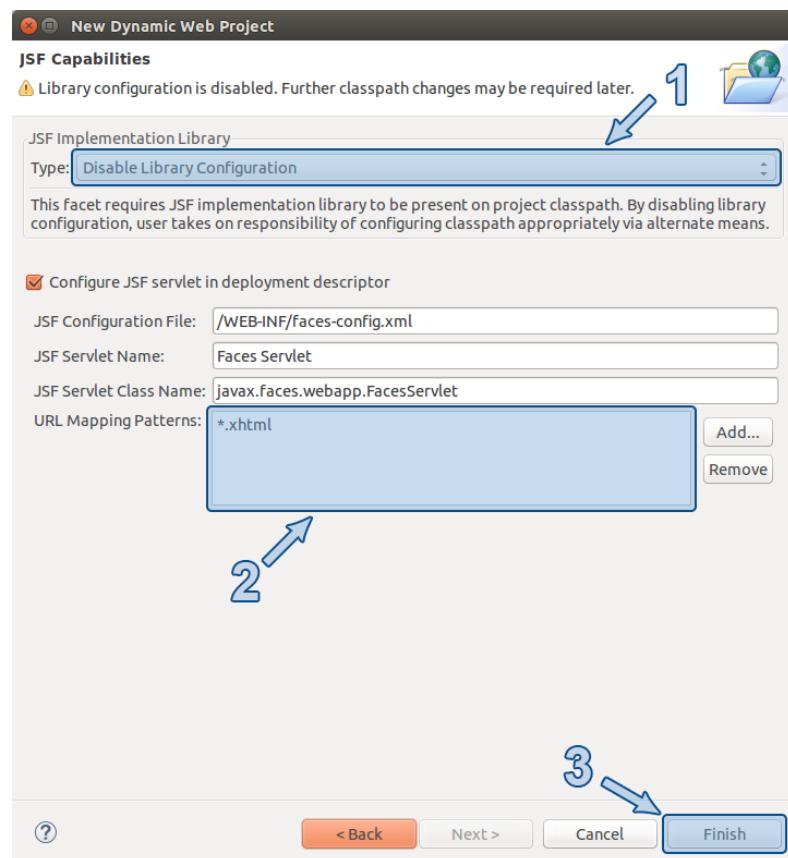
Exercícios de Fixação

- Crie um Dynamic Web Project no eclipse chamado **bugWeb**. Você pode digitar “CTRL+3” em seguida “new Dynamic Web Project” e “ENTER”. Depois, siga exatamente as imagens abaixo.









- 2** Adicione uma pasta chamada **META-INF** na pasta **src** do projeto **bugWeb**. Na pasta **META-INF**, crie o arquivo **persistence.xml**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
5     http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
6   version="2.1">
7
8   <persistence-unit name="K19" transaction-type="JTA">
9     <jta-data-source>jdbc/K19</jta-data-source>
10
11    <properties>
12      <property
13        name="javax.persistence.schema-generation.database.action"
14        value="create"/>
15    </properties>
16  </persistence-unit>
17</persistence>
```

Código XML A.1: persistence.xml

- 3** Adicione o arquivo **beans.xml** na pasta **WEB-INF** do projeto **bugWeb**.

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://java.sun.com/xml/ns/javaee"
3   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4   xsi:schemaLocation="
```

```

5   http://java.sun.com/xml/ns/javaee
6   http://java.sun.com/xml/ns/javaee/beans_1_0.xsd">
7 </beans>
```

Código XML A.2: beans.xml

- 4** Configure a aplicação **bugWeb** para que ele utilize o Realm **K19-Realm** criado no capítulo [7](#), adicionando no arquivo **web.xml** do projeto **bugWeb** as configurações necessárias.

```

1 ...
2 <login-config>
3   <realm-name>K19-Realm</realm-name>
4 </login-config>
5 ...
```

Código XML A.3: web.xml

- 5** Faça o mapeamento dos Groups do K19-Realm para os Roles da aplicação **bugWeb**.

```

1 ...
2 <security-role-mapping>
3   <role-name>ADMIN</role-name>
4   <group-name>admin</group-name>
5 </security-role-mapping>
6
7 <security-role-mapping>
8   <role-name>USERS</role-name>
9   <group-name>users</group-name>
10 </security-role-mapping>
11 ...
```

Código XML A.4: glassfish-web.xml

- 6** Crie um pacote chamado **br.com.k19.entities** no projeto **bugWeb**. Adicione nesse pacote um Entity Bean para modelar projetos e outro para modelar bugs.

```

1 package br.com.k19.entities;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6
7 @Entity
8 public class Project {
9
10   @Id @GeneratedValue
11   private Long id;
12
13   private String name;
14
15   private String description;
16
17   public Long getId() {
18     return id;
19   }
20
21   // GETTERS AND SETTERS
22 }
```

Código Java A.1: Project.java

```
1 package br.com.k19.entities;
2
3 import javax.persistence.Entity;
4 import javax.persistence.GeneratedValue;
5 import javax.persistence.Id;
6 import javax.persistence.ManyToOne;
7
8 @Entity
9 public class Bug {
10
11     @Id @GeneratedValue
12     private Long id;
13
14     private String description;
15
16     private String severity;
17
18     @ManyToOne
19     private Project project;
20
21     // GETTERS AND SETTERS
22 }
```

Código Java A.2: Bug.java

- 7 Crie um pacote chamado **br.com.k19.sessionbeans** no projeto **bugWeb**. Adicione nesse pacote um SLSB para funcionar como repositório de projetos e outro para funcionar como repositório de bugs.

```
1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.annotation.security.RolesAllowed;
6 import javax.ejb.Stateless;
7 import javax.ejb.TransactionAttribute;
8 import javax.ejb.TransactionAttributeType;
9 import javax.persistence.EntityManager;
10 import javax.persistence.PersistenceContext;
11 import javax.persistence.TypedQuery;
12
13 import br.com.k19.entities.Bug;
14 import br.com.k19.entities.Project;
15
16 @Stateless
17 @RolesAllowed({ "ADMIN", "USERS" })
18 public class ProjectRepository {
19
20     @PersistenceContext
21     private EntityManager manager;
22
23     public void add(Project project) {
24         this.manager.persist(project);
25     }
26
27     public void edit(Project project) {
28         this.manager.merge(project);
29     }
30
31     @RolesAllowed({ "ADMIN" })
32     public void removeById(Long id) {
33         Project project = this.manager.find(Project.class, id);
34
35         TypedQuery<Bug> query = this.manager.createQuery(
36             "select x from Bug x where x.project = :project", Bug.class);
37         query.setParameter("project", project);
```

```

38     List<Bug> bugs = query.getResultList();
39     for (Bug bug : bugs) {
40         this.manager.remove(bug);
41     }
42
43     this.manager.remove(project);
44 }
45
46 @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
47 public List<Project> findAll() {
48     TypedQuery<Project> query = this.manager.createQuery(
49         "select x from Project x", Project.class);
50     return query.getResultList();
51 }
52
53 @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
54 public Project findById(Long id) {
55     return this.manager.find(Project.class, id);
56 }
57 }
```

Código Java A.3: ProjectRepository.java

```

1 package br.com.k19.sessionbeans;
2
3 import java.util.List;
4
5 import javax.annotation.security.RolesAllowed;
6 import javax.ejb.Stateless;
7 import javax.ejb.TransactionAttribute;
8 import javax.ejb.TransactionAttributeType;
9 import javax.persistence.EntityManager;
10 import javax.persistence.PersistenceContext;
11 import javax.persistence.TypedQuery;
12
13 import br.com.k19.entities.Bug;
14
15 @Stateless
16 @RolesAllowed({ "ADMIN", "USERS" })
17 public class BugRepository {
18     @PersistenceContext
19     private EntityManager manager;
20
21     public void add(Bug bug) {
22         this.manager.persist(bug);
23     }
24
25     public void edit(Bug bug) {
26         this.manager.merge(bug);
27     }
28
29     @RolesAllowed({ "ADMIN" })
30     public void removeById(Long id) {
31         Bug bug = this.manager.find(Bug.class, id);
32         this.manager.remove(bug);
33     }
34
35     @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
36     public List<Bug> findAll() {
37         TypedQuery<Bug> query = this.manager.createQuery("select x from Bug x",
38             Bug.class);
39         return query.getResultList();
40     }
41
42     @TransactionAttribute(TransactionAttributeType.NOT_SUPPORTED)
43     public Bug findById(Long id) {
44         return this.manager.find(Bug.class, id);
45     }
46 }
```

Código Java A.4: BugRepository.java

- 8 Crie um pacote chamado **br.com.k19.managedbeans** no projeto **bugWeb**. Adicione nesse pacote um Managed Bean CDI para oferecer para as telas JSF as funcionalidades de CRUD relacionadas aos projetos.

```
1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.enterprise.context.RequestScoped;
6 import javax.inject.Inject;
7 import javax.inject.Named;
8
9 import br.com.k19.entities.Project;
10 import br.com.k19.sessionbeans.ProjectRepository;
11
12 @Named
13 @RequestScoped
14 public class ProjectMB {
15
16     @Inject
17     private ProjectRepository projectRepository;
18
19     private Project project = new Project();
20
21     private List<Project> projects;
22
23     public void save() {
24         if (this.getProject().getId() == null) {
25             this.projectRepository.add(this.getProject());
26         } else {
27             this.projectRepository.edit(this.getProject());
28         }
29         this.project = new Project();
30         this.projects = null;
31     }
32
33     public void delete(Long id) {
34         this.projectRepository.removeById(id);
35         this.projects = null;
36     }
37
38     public void prepareEdit(Long id) {
39         this.project = this.projectRepository.findById(id);
40     }
41
42     public Project getProject() {
43         return project;
44     }
45
46     public List<Project> getProjects() {
47         if (this.projects == null) {
48             this.projects = this.projectRepository.findAll();
49         }
50         return projects;
51     }
52 }
```

Código Java A.5: ProjectMB.java

- 9 Adicione no pacote **br.com.k19.managedbeans** um Managed Bean CDI para oferecer para as

telas JSF as funcionalidades de CRUD relacionadas aos bugs.

```
1 package br.com.k19.managedbeans;
2
3 import java.util.List;
4
5 import javax.enterprise.context.RequestScoped;
6 import javax.inject.Inject;
7 import javax.inject.Named;
8
9 import br.com.k19.entities.Bug;
10 import br.com.k19.entities.Project;
11 import br.com.k19.sessionbeans.BugRepository;
12 import br.com.k19.sessionbeans.ProjectRepository;
13
14 @Named
15 @RequestScoped
16 public class BugMB {
17
18     @Inject
19     private BugRepository bugRepository;
20
21     @Inject
22     private ProjectRepository projectRepository;
23
24     private Bug bug = new Bug();
25
26     private Long projectId;
27
28     private List<Bug> bugs;
29
30     public void save() {
31         Project project = this.projectRepository.findById(this.projectId);
32         this.bug.setProject(project);
33
34         if (this.getBug().getId() == null) {
35             this.bugRepository.add(this.getBug());
36         } else {
37             this.bugRepository.edit(this.getBug());
38         }
39         this.bug = new Bug();
40         this.bugs = null;
41     }
42
43     public void delete(Long id) {
44         this.bugRepository.removeById(id);
45         this.bugs = null;
46     }
47
48     public void prepareEdit(Long id) {
49         this.bug = this.bugRepository.findById(id);
50     }
51
52     public Bug getBug() {
53         return bug;
54     }
55
56     public List<Bug> getBugs() {
57         if (this.bugs == null) {
58             this.bugs = this.bugRepository.findAll();
59         }
60         return bugs;
61     }
62
63     public void setprojectId(Long projectId) {
64         this.projectId = projectId;
65     }
66
67     public Long getprojectId() {
68         return projectId;
```

```
69 }  
70 }
```

Código Java A.6: BugMB.java

- 10 Adicione no pacote **br.com.k19.managedbeans** um Managed Bean CDI para implementar o processo de login e logout.

```
1 package br.com.k19.managedbeans;  
2  
3 import javax.enterprise.context.RequestScoped;  
4 import javax.faces.context.FacesContext;  
5 import javax.inject.Named;  
6 import javax.servlet.ServletException;  
7 import javax.servlet.http.HttpServletRequest;  
8  
9 @Named  
10 @RequestScoped  
11 public class AuthenticatorMB {  
12     private String username;  
13     private String password;  
14  
15     public String login() throws ServletException {  
16         FacesContext context = FacesContext.getCurrentInstance();  
17         HttpServletRequest request = (HttpServletRequest) context  
18             .getExternalContext().getRequest();  
19         request.login(this.username, this.password);  
20  
21         return "/projects";  
22     }  
23  
24     public String logout() throws ServletException {  
25         FacesContext context = FacesContext.getCurrentInstance();  
26         HttpServletRequest request = (HttpServletRequest) context  
27             .getExternalContext().getRequest();  
28         request.logout();  
29         return "/login";  
30     }  
31  
32     // GETTERS AND SETTERS  
33 }
```

Código Java A.7: AuthenticatorMB.java

- 11 Crie um pacote chamado **br.com.k19.filters** no projeto **bugWeb**. Adicione nesse pacote um Filtro para verificar a autenticação dos usuários.

```
1 package br.com.k19.filters;  
2  
3 import java.io.IOException;  
4  
5 import javax.servlet.FilterChain;  
6 import javax.servlet.FilterConfig;  
7 import javax.servlet.ServletException;  
8 import javax.servlet.ServletRequest;  
9 import javax.servlet.ServletResponse;  
10 import javax.servlet.annotation.WebFilter;  
11 import javax.servlet.http.HttpServletRequest;  
12 import javax.servlet.http.HttpServletResponse;  
13  
14 @WebFilter(servletNames = { "Faces Servlet" })  
15 public class AuthenticatorFilter implements javax.servlet.Filter {
```

```

17  @Override
18  public void doFilter(ServletRequest request, ServletResponse response,
19      FilterChain chain) throws IOException, ServletException {
20      HttpServletRequest req = (HttpServletRequest) request;
21
22      if (req.getRemoteUser() == null && !req.getRequestURI().endsWith(req.get-
23          getContextPath() + "/login.xhtml")) {
24          HttpServletResponse res = (HttpServletResponse) response;
25          res.sendRedirect(req.getContextPath() + "/login.xhtml");
26      } else {
27
28          chain.doFilter(request, response);
29      }
30  }
31
32  @Override
33  public void init(FilterConfig filterConfig) throws ServletException {
34  }
35
36  @Override
37  public void destroy() {
38 }

```

Código Java A.8: AuthenticatorFilter.java

- 12** Crie um menu para as telas da aplicação **bugWeb**. Adicione um arquivo chamado **menu.xhtml** na pasta **WebContent** do projeto **bugWeb**.

```

1 <ui:composition xmlns="http://www.w3.org/1999/xhtml"
2   xmlns:ui="http://java.sun.com/jsf/facelets"
3   xmlns:h="http://java.sun.com/jsf/html"
4   xmlns:f="http://java.sun.com/jsf/core">
5
6   <h:form>
7     <h:panelGrid>
8       <h:commandLink action="#{authenticatorMB.logout}">logout</h:commandLink>
9
10      <h:outputLink value="#{request.contextPath}/projects.xhtml">Projects</h:-
11          outputLink>
12
13      <h:outputLink value="#{request.contextPath}/bugs.xhtml">Bugs</h:outputLink>
14    </h:panelGrid>
15  </h:form>
16 </ui:composition>

```

Código XHTML A.1: menu.xhtml

- 13** Crie um tela de login na aplicação **bugWeb**. Adicione um arquivo chamado **login.xhtml** na pasta **WebContent** do projeto **bugWeb**.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10   <title>Login</title>
11 </h:head>
12

```

```

13 <h:body>
14   <h1>Login</h1>
15   <h:form>
16     <h:panelGrid>
17       <h:outputLabel value="Username:> />
18       <h:inputText value="#{authenticatorMB.username}" />
19
20       <h:outputLabel value="Password:> />
21       <h:inputSecret value="#{authenticatorMB.password}" />
22
23       <h:commandButton action="#{authenticatorMB.login}" value="login" />
24     </h:panelGrid>
25   </h:form>
26 </h:body>
27 </html>

```

Código XHTML A.2: login.xhtml

- 14** Crie uma tela para administrar os projetos. Adicione um arquivo chamado **projects.xhtml** na pasta **WebContent** do projeto **bugWeb**.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10  <title>Projects</title>
11 </h:head>
12
13 <h:body>
14  <ui:include src="/menu.xhtml"/>
15
16  <hr/>
17
18  <h1>New Project</h1>
19  <h:form>
20    <h:panelGrid>
21      <h:inputHidden value="#{projectMB.project.id}" />
22
23      <h:outputLabel value="Name:> />
24      <h:inputText value="#{projectMB.project.name}" />
25
26      <h:outputLabel value="Description:> />
27      <h:inputTextarea value="#{projectMB.project.description}" />
28
29      <h:commandButton action="#{projectMB.save}" value="Save" />
30    </h:panelGrid>
31  </h:form>
32
33  <hr/>
34
35  <h1>Project List</h1>
36  <h: dataTable value="#{projectMB.projects}" var="project"
37   rendered="#{not empty projectMB.projects}" border="1">
38    <h:column>
39      <f:facet name="header">Id</f:facet>
40      #{project.id}
41    </h:column>
42
43    <h:column>
44      <f:facet name="header">Name</f:facet>
45      #{project.name}
46    </h:column>

```

```

47 <h:column>
48     <f:facet name="header">Description</f:facet>
49     #{project.description}
50 </h:column>
51
52 <h:column>
53     <f:facet name="header">Delete</f:facet>
54     <h:form>
55         <h:commandLink action="#{projectMB.delete(project.id)}" >delete</h:commandLink>
56     </h:form>
57 </h:column>
58 <h:column>
59     <f:facet name="header">Edit</f:facet>
60     <h:form>
61         <h:commandLink action="#{projectMB.prepareEdit(project.id)}" >edit</h:commandLink>
62     </h:form>
63 </h:column>
64 </h:datatable>
65 </h:body>
66 </html>

```

Código XHTML A.3: projects.xhtml

- 15 Crie uma tela para administrar os bugs. Adicione um arquivo chamado **bugs.xhtml** na pasta **WebContent** do projeto **bugWeb**.

```

1 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4 <html xmlns="http://www.w3.org/1999/xhtml"
5   xmlns:ui="http://java.sun.com/jsf/facelets"
6   xmlns:h="http://java.sun.com/jsf/html"
7   xmlns:f="http://java.sun.com/jsf/core">
8
9 <h:head>
10    <title>Bugs</title>
11 </h:head>
12
13 <h:body>
14    <ui:include src="/menu.xhtml"/>
15
16    <hr />
17
18    <h1>New Bug</h1>
19    <h:form>
20        <h:panelGrid>
21            <h:inputHidden value="#{bugMB.bug.id}" />
22
23            <h:outputLabel value="Severity:" />
24            <h:selectOneMenu value="#{bugMB.bug.severity}">
25                <f:selectItem itemValue="LOW" />
26                <f:selectItem itemValue="MEDIUM" />
27                <f:selectItem itemValue="HIGH" />
28            </h:selectOneMenu>
29
30
31            <h:outputLabel value="Description:" />
32            <h:inputTextarea value="#{bugMB.bug.description}" />
33
34            <h:outputLabel value="Project:" />
35            <h:selectOneMenu value="#{bugMB.projectId}">
36                <f:selectItems value="#{projectMB.projects}" var="project"
37                    itemLabel="#{project.name}" itemValue="#{project.id}" />
38            </h:selectOneMenu>

```

```

39      <h:commandButton action="#{bugMB.save}" value="Save" />
40  </h:panelGrid>
41 </h:form>
42
43  <hr />
44
45  <h1>Bug List</h1>
46  <h: dataTable value="#{bugMB.bugs}" var="bug"
47    rendered="#{not empty bugMB.bugs}" border="1">
48    <h: column>
49      <f: facet name="header">Id</f: facet>
50      #{bug.id}
51    </h: column>
52
53    <h: column>
54      <f: facet name="header">Project</f: facet>
55      #{bug.project.name}
56    </h: column>
57
58    <h: column>
59      <f: facet name="header">Severity</f: facet>
60      #{bug.severity}
61    </h: column>
62
63    <h: column>
64      <f: facet name="header">Description</f: facet>
65      #{bug.description}
66    </h: column>
67
68    <h: column>
69      <f: facet name="header">Delete</f: facet>
70      <h: form>
71        <h: commandLink action="#{bugMB.delete(bug.id)}">delete</h: commandLink>
72      </h: form>
73    </h: column>
74    <h: column>
75      <f: facet name="header">Edit</f: facet>
76      <h: form>
77        <h: commandLink action="#{bugMB.prepareEdit(bug.id)}">edit</h: commandLink>
78      </h: form>
79    </h: column>
80  </h: dataTable>
81 </h: body>
82 </html>

```

Código XHTML A.4: bugs.xhtml

- 16 Crie uma tela para os erros internos e os erros de autorização. Adicione um arquivo chamado **error.xhtml** na pasta **WebContent** do projeto **bugWeb**.

```

1  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
2   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
3
4  <html xmlns="http://www.w3.org/1999/xhtml"
5    xmlns:ui="http://java.sun.com/jsf/facelets"
6    xmlns:h="http://java.sun.com/jsf/html"
7    xmlns:f="http://java.sun.com/jsf/core">
8
9  <h:head>
10   <title>Error</title>
11 </h:head>
12
13 <h:body>
14   <h3>Internal Error or Client not authorized for this invocation.</h3>
15 </h:body>
16 </html>

```

Código XHTML A.5: error.xhtml

- 17 Configure a página de erro no arquivo **web.xml** do projeto **bugWeb**. Adicione o seguinte trecho de código nesse arquivo.

```
1 <error-page>
2   <exception-type>java.lang.Exception</exception-type>
3   <location>/error.xhtml</location>
4 </error-page>
```

Código XML A.5: web.xml

- 18 Teste a aplicação!!!