<http://www.javabeat.net/introduction-to-jboss-seam/>

Introduction to JBoss Seam

1) Introduction

Web Application Framework called ***“JBoss Seam”*** which combines the advantages from the two rapidly growing technologies ***Enterprise Java Beans 3.0***and ***Java Server Faces***. JBoss Seam, by sitting on top of J2EE provides a nice way of integration between JSF and EJB Components with other great functionalities. This article is an introductory article only and it covers the idea that gave birth to JBoss Seam, its advantages, the various modules involved along with a Sample Application.

2) The Need for JBoss Seam

JBoss Seam provides a ***Light-weight Container*** for J2EE standards and it addresses the long-standing issues in any Typical Web Application like ***State Management*** and ***Better Browser Navigation***. It also neatly provides an integration between the two popular technologies, ***Java Server Faces*** (in the UI tier) and ***Enterprise Java Beans*** (EJB 3 in the Server Side).

Let us assume an imaginary application The Web Application is a simple Registration Application, which provides the user with a View which contains username and password text-fields. User can click the submit button after filling both the fields. If the username password information given by the user is not found in the database, the user is assumed to be a new user and he is greeted with a welcome message; else an error page is displayed with appropriate message.  
Let us analysis the roles of ***JSF*** and ***EJB 3.0*** in this Web Application. More specifically, we will analysis the various components in both the client as well the server tier along with their responsibilities.  
In the Client side, for designing and presenting the form with the input controls (text-field and button) to the user, we may have written a userinput.jsp page with the set of JSF core tag libraries like <f:view> and <h:form>. And then, a JSF Managed Bean called UserBean encapsulated with properties(username and password) may have been coded which serves as a Model. The UI components values within the JSP page would have got bound to the properties of the Managed Bean with the help of JSF Expression Language. Since the logic is to query from the database for the existence of the username and the password, a Stateless Session Facade Bean would have been written whose sole purpose is to persistence the client information to the database. For persistence the information, the Session Bean may depend on the EntityManager API for querying and persisting entities.  
The ***Data Traversal Logic*** from JSF to EJB Session Bean should have be taken care by the ***Managed Bean*** only. The Managed Bean apart from representing a Model may also act as a ***Listener*** in handling the button click events. Say, after clicking the register button, one of the action methods within the Managed Bean would have been called by the framework, and here the Bean might have a ***JNDI Look-up*** to get an instance of the Session Bean to persisting or querying the user information. If we look at carefully, the ***JSF Managed Bean*** is serving as an ***intermediary*** between the transfer of Data from the Client to the Server. Within this Managed Bean is the code for getting a reference to the Session Bean for doing various other functionalities. Wouldn’t a direct commnication between ***JSF UI Components*** and the ***Enterprise Bean Components*** be nice? There is no purpose of the intermediate Managed Bean in this case. JBoss Seam provides a very good solution for this. Not only this many of the outstanding problems that are faced in a Web Application are also addressed and given solution in this Framework.

3) Advantages of JBoss Seam

* Integration of JSF and EJB
* Stateful Web Applications
* Dependency Bijection Support

3.1) Integration of JSF and EJB

Today the Web Application World see more and more matured technologies that are focusing to establish an easy-to-use development by reducing lots and lots of boiler-plate code along with some other added functionalities in their own domains. Let us consider JSF and EJB technologies to extend further discussion regarding this.  
EJB 3.0 which is a specification given from Sun has gained much popularity because of its simplified yet robust programming model. Much of the middle-ware related services like ***Security***, ***Transactions***, ***Connection Pooling*** etc is delegated to the container itself. Comparing to its predecessors, EJB 3.0 offers a ***POJO programming Model***. No need for your beans to extend or implement EJB specific classes or interfaces. And also, along with the new specification [Java Persistence API (JPA)](http://www.javabeat.net/jpa/), an unified programming model to access the underlying database along with rich set of features are now possible thereby completely eliminating the heavy-headed entity beans.  
[**What is JPA?**](http://www.javabeat.net/jpa/):Java Persistence API (JPA) provides POJO (Plain Old Java Object) standard and object relational mapping (OR mapping) for data persistence among applications. Persistence, which deals with storing and retrieving of application data, can now be programmed with Java Persistence API starting from EJB 3.0 as a result of JSR 220. This API has borrowed many of the concepts and standards from leading persistence frameworks like Toplink (from Oracle) and Hibernate (from JBoss). One of the great benefits of JPA is that it is an independent API and can nicely integrate with J2EE as well as J2SE applications.  
***Java Server Faces*** provides a ***Component-based approach*** for developing User Interface Components in a Web Application. It hides most of the boiler-plate code and provides a higher-level abstraction over the client request and the server response objects. Using Java Server Faces, a Web Application can be viewed by components making events and listeners executing the appropriate logic for the Events. No need for the traditional HttpServletRequest and HttpServletResponse object to extract the client input parameters and to generate the response.  
JBoss provides a framework in which the ***Events*** that are emitted by the ***JSF UI Components*** can be directly handled by the***Enterprise Beans***. No need for the intermediate Managed Beans which establishes the data transfer from and to JSF and EJB.

3.2) Dependency Bijection Support

Before getting into ***Dependency Bijection***, it is wise to look at the two types: namely ***Dependency Injection***and***Dependency Outjection***

**3.2.1) Dependency Injection**  
This model is used when a ***Component*** or a ***Service*** which is running inside some ***Framework*** or a ***Container*** is well known in the early stages so that the Framework/Container can create instances of them thereby taking the burden from the Clients. These type of model is used heavily in most of the J2EE Components, to name a few, EJB, Servlets, JMS etc.  
For example, consider the following piece of code,  
**MySessionBean.java**

|  |  |  |
| --- | --- | --- |
| 1 | @Stateless | |
| 2 |  | public class MySessionBean{ | |

|  |  |
| --- | --- |
| 4 | @PersistentContext |

|  |  |  |
| --- | --- | --- |
| 5 | private EntityManager entityManager; | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | public void query(){ |
| 8 | // Do something with EntityManager reference. | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | } | | |
| 10 | | } |

In the above code, the Session Bean (MySessionBean) is depending on the services of EntityManager. So, before the session bean starts using the EntityManager object, the EntityManager instance should be available. This is made simple because the EJB Container will creates a new instance of the EntityManager object based on the various parameters taken from the Configuration File and injects this instance to the EntityManager reference. This way of doing is called Dependency Injection.  
In Seam, injection of a Component by the Seam framework is done by the use of @In Annotation. For example, consider the following piece of code,  
**MyComponent.java**

|  |  |  |
| --- | --- | --- |
| 1 | public class MyComponent{ | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | @In |
| 4 | private MyInjectorComponent compToBeInjected; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | } | |

When the Application gets deployed, Seam Framework will keep track of these Annotated Components and during run-time, upon creation of MyComponent class and before the invocation of any other methods, the Container will inject and instance of an object of type MyInjectorComponent to the compToBeInjected reference.

**3.2.2) Dependency Outjection**  
In Dependency Injection, usually the Framework injects the services to components that are in need of, whereas the reverse happens in ***Dependency Outjection***. Consider the following piece of code,  
**MyComponent.java**

|  |  |  |
| --- | --- | --- |
| 1 | public class MyComponent{ | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | @Out |
| 4 | private MyService myService; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | public MyComponent(){ | |

|  |  |  |
| --- | --- | --- |
| 7 | } | |
| 8 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | public MyService getMyService(){ | | |
| 10 | | return createMyService(); |

|  |  |  |
| --- | --- | --- |
| 11 | } | |
| 12 |  |

|  |  |  |
| --- | --- | --- |
| 13 | private MyService createMyService(){ | |
| 14 | // My own way of creating |

|  |  |  |
| --- | --- | --- |
| 15 | } | |
| 16 | } |

**MyServiceClient.java**

|  |  |  |
| --- | --- | --- |
| 1 | public class MyServiceClient{ | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | @In |
| 4 | private MyService myService; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | } | |

In the above code, MyComponent class is acting as a factory class for creating MyService objects. Assume that this MyComponent class is a controlled class meaning that is being managed by the Container or the Framework wherever it is embedded. Situations may arise such that the MyService object might be needed by the Container so that the Container can use this MyService to inject dependencies on other components or clients. For instance, MyServiceClient is in need of MyService object and the Container can inject this reference from the object that it has taken from MyComponent class. This model is ***Dependency Outjection*** where a Service is taken by the Container from the Component.  
Seam supports both these models in the form of @In and @Out annotations. For example consider the following class,  
**MyClass.java**

|  |  |  |
| --- | --- | --- |
| 1 | public class MyClass{ | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | @In |
| 4 | private MyServiceOne service1; | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | @out | |

|  |  |  |
| --- | --- | --- |
| 7 | private MyServiceTwo service2; | |
| 8 | } |

The above code can be read like this. The container injects the reference to service1 because it is annotated with @In(meaning for Injection). And at a later stage, the Container may taken the reference of service2 (because of @Out) thereby serving it to other needful Components.

3.3) Stateful Web Applications

Since the underlying protocol used in a Web Application is ***Http***, all Web Application are ***Stateless in nature***. Precisely it means that all the requests that are coming from the Client Browser are treated as individual requests only. The Server shows no partiality for the Client Requests. It is up to the Application or the Framework to identify whether requests are coming from the same Client or not. Session Management in Web Application is a time-consuming job and typically Servlets/Jsp provides various ways to manage sessions. Even in this case, Application Developers still have to depend on HttpSession like classes for creating session objects and storing/retrieving objects from the session.  
JBoss Seam provides an excellent way to ***Manage States*** amongst multiple client requests. The ***State Management Facility***is tightly integrated with Seam Components in the form of various ***Contexts***. Following are the most commonly used Contexts in Seam.

* Conversation – A ***Conversation*** generally represents multiple requests that come from the same client. Components that come under this category can remember the state of the client and this forms the heart of ***State Management*** in Seam.
* Application – This context generally holds information that is used by various components within the same Application.
* Session – Components that fall under this category simply uses ***Http Session Api*** for state management.
* Page – The scope of the components managing the information is restricted to the ***Current Page*** only. All the stored information is lost when the user leaves this current page.
* Stateless – Components that falls under this category don’t manage state between multiple Client Requests.

4) Seam Components

Components in Seam take over the functionality as well the Business Logic in a Web Application. The good thing is that all Seam Components are ***POJO Objects*** only meaning that Components don’t have to extend or implement any Seam specific Classes or Interfaces. The Seam Component Model follows the standard ***Java Bean Component Architecture***. Even it is possible to ***Annotate Enterprise Beans*** (Session Beans, Entity Beans and Message Driven Beans) as Components. Seam provides an excellent framework for ***Integrating Enterprise Beans into the Seam Framework***.  
As mentioned before, Components in Seam carries the business process involved in a typical Web Application. Seam Components may also act as ***Listeners*** for JSF UI Components or they may even interact with the Database. Since Seam provides a good deal of Integration with ***EJB Components***, Developers generally prefer having ***Stateful Session Beans*** as Listeners and the ***Entity Beans*** for interacting with the Database.  
Usually Seam Components are identified by ***Name***, ***Scope*** and ***Role***. These are ***Declarative Identifications*** meaning that they are represented directly in the Source Code with the help of Annotations. For example, consider the following Seam Component,  
**MyStatefulBeanImpl.java**

|  |  |
| --- | --- |
| 1 | @Stateful |
| 2 | @Name("MyStatefulBean") | |

|  |  |
| --- | --- |
| 3 | @Scope(ScopeType.SESSION) |
| 4 | public MyStatefulBeanImpl implements MyStatefulBean{ | |

|  |  |
| --- | --- |
| 5 |  |
| 6 | public String statefulMethod(){ | |

|  |  |  |
| --- | --- | --- |
| 7 | } | |
| 8 | } |

In the above code, the @Name Annotation represents the name of a Seam Component using which other Managed Seam Components can access it. @Scope defines how life this Component will survive during the Web Application. Since the above class is actually a Session Bean, it has been marked with @Stateful Annotation.

5) Support for Annotations

Most of the boiler-plate needed for the various parts of the Application is taken care by the Seam because of the declarative style of programming available in the form of Annotations. Most of the Annotations that we see in Seam are a mix of EJB 3.0 and Seam specific Annotations. They are a number of Annotations defined in the Seam Api and this section just provides an overview of the most commonly used Annotations. For a complete list of Annotations and their purpose, have a look at the Seam Documentation.

5.1) Annotations for Seam Components

Components in Seam are really POJO’s and they play a major role in keeping the ***Application stateful***. Following are the most commonly used Annotations related to Seam Components are given below.

* @Name
* @Scope
* @JndiName

**5.1.1) @Name Annotation**  
This Annotation specifies the name of a ***Seam Component*** so that other Components (Seam Components or JSF Pages) can refer the Component by its name. This Annotation is mandatory for a class that is going to act as a Seam Component. Following is an example of the Seam Component.  
**MyComponent.java**

|  |  |
| --- | --- |
| 1 | @Name("MyComponent") |
| 2 | public class MyComponent{ | |

|  |  |  |
| --- | --- | --- |
| 3 | // Some Functionalities here. | |
| 4 | } |

**5.1.2) @Scope Annotation**  
This Annotation, if given, tells to the Seam Framework about the scope (or the life-time) for a Seam Component. The values for this Annotation are taken from the org.jboss.seam.ScopeType Enum and can be any of the following values:APPLICATION, CONVERSATION, SESSION, PAGE, STATELESS etc. Following is a sample code snippet for using the @ScopeAnnotation.  
**MyComponent.java**

|  |  |  |
| --- | --- | --- |
| 1 | @Scope(ScopeType.SESSION) | |
| 2 | @Name("MyComponent") |

|  |  |
| --- | --- |
| 3 | public class MyComponent{ |
| 4 | // Some Functionalities here. | |

|  |  |
| --- | --- |
| 5 | } |

**5.1.3) @JndiAnnotation Annotation**  
This Annotation should not be applied to normal Seam Components but to Components that represent any of ***EJB*** or ***JMS Service***. Following is an example of one such component.  
**MySessionBean.java**

|  |  |  |
| --- | --- | --- |
| 1 | @JndiName("ejb/session/MySessionBean") | |
| 2 | Public class MySessionBean{ |

|  |  |
| --- | --- |
| 3 |  |
| 4 | } | |

5.2) Annotations for Components Lifecycle

All Components in Seam have well defined ***Life-cycle Management*** as defined by the Seam Framework. Management of Component Life-cycle is important as it has dependencies over the state Management of Web Applications. Following are the most commonly used Annotations for Life-cycle Management.

* @Create
* @Destroy

**5.2.1) @Create Annotation**  
When a method inside a Seam Component is marked with @Create Annotation, then this method will be called immediately after the creation of Seam Component. Most of the costly Resources like ***Initializing*** File or Database can be coded in this method. Following is the sample code snippet for the same,  
**MyDatabaseUtils.java**

|  |  |
| --- | --- |
| 1 | @Name("MyDatabaseUtils") |
| 2 | Public class MyDatabaseUtils{ | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | @Create() | |

|  |  |  |
| --- | --- | --- |
| 5 | public void initDatabase(){ | |
| 6 | } |

|  |  |
| --- | --- |
| 7 | } |

**5.2.2) @Destroy Annotation**  
If @Create Annotation for a Seam Component deals with initialization Stuffs for a object, then the code inside the @DestroyAnnotation can be used to ***Release Resources or References*** to other objects. This method will be called before a Component is going to get removed from any of the Context as mentioned by its Scope.  
**MyDatabaseUtils.java**

|  |  |
| --- | --- |
| 1 | @Name("MyDatabaseUtils") |
| 2 | Public class MyDatabaseUtils{ | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | @Destroy() | |

|  |  |  |
| --- | --- | --- |
| 5 | public void closeDbResources(){ | |
| 6 | } |

|  |  |
| --- | --- |
| 7 | } |

6) Events

***Events*** can occur in a Seam Application through various ***Sources***. And the events emitted by the Sources are captured and handled by the ***Listeners***. Let us look into the various Sources that are responsible for emitting Events. Source can be any one of the following items.

* Seam Page Events
* Seam Component Driven Events
* JSF Components Emitting Events

Let us look into the details one by one.

6.1) Seam Page Events

Before Seam renders a page to Display, it checks whether the page has registered itself for firing any Events. If that’s the case, then the Framework will fire Events on behalf of the Web Pages. For example, consider the following situation. If the Home Page of a Web-site is eager to display the hits to this page, then the following could be possible,  
**pages.xml**

|  |  |
| --- | --- |
| 1 | <pages> |
| 2 | <page view-id = "/hit-count.jsp" action = "#{Hitter.initHits}"> | |

|  |  |  |
| --- | --- | --- |
| 3 | </page> | |
| 4 | </pages> |

The above is the pages.xml file and it should be placed in the WEB-INF directory of the Web Application module. The above code essentially tells that whenever a request comes for /hit-count.jsp, then call the method initHits() available in the Hitter class.

6.2) Seam Component Driven Events

It is possible by the Seam Components also to emit Events. There is a ***Low-degree of Coupling*** between the Initiation and the Handling of Events. Even ***Custom Events*** can be declared specified either in the Xml File or through Annotations. For example, consider the following code snippet,

**components.xml**

|  |  |
| --- | --- |
| 1 | <components> |
| 2 | <event type = "myEvent"> | |

|  |  |
| --- | --- |
| 3 | <action expression = "#{MyComponent.myCallBack1}"/> |
| 4 | <action expression = "#{MyComponent.myCallBack2}"/> |

|  |  |
| --- | --- |
| 5 | </event> |
| 6 | </components> | |

The above Xml File must be placed in the WEB-INF directory of the Web Application Module. If anyone raises an Event of type ('myEvent' is just a string), then the following methods defined inside the action element will be fired. For, example, consider the following code,

**MyEventInitiator.java**

|  |  |  |
| --- | --- | --- |
| 1 | public class MyEventInitiator{ | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | public void initEvent(){ |
| 4 | Events.instance().raiseEvent("myEvent"); | |

|  |  |  |
| --- | --- | --- |
| 5 | } | |
| 6 | } |

**MyComponent.java**

|  |  |  |
| --- | --- | --- |
| 1 | public MyComponent{ | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | public void myCallBack1(){ |
| 4 | System.out.println("My Call back method 1 called"); | |

|  |  |  |
| --- | --- | --- |
| 5 | } | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | public void myCallBack2(){ |
| 8 | System.out.println("My Call back method 2 called"); | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | } | | |
| 10 | | } |

If some one initiates the Event by calling MyEventInitiator.initEvent(), then a event of type myEvent gets raised. Since this Listeners (Actions) of this Event are defined in the components.xml file in the form ofMyComponent.myCallBack1() and MyComponent.myCallback2(), both these methods will be fired in the order in which they are defined.

6.3) JSF Components Emitting Events

As discussed previously, Seam components can act as ***Listeners*** for the ***JSF UI Components*** emitting Events. For example, consider the following piece of Code,  
**calculate.jsp**

|  |  |  |
| --- | --- | --- |
| 1 | <h:commandButton value = "Calculate Tax" action = "#{TaxCalculator.calculateTax}"> | |
| 2 | </h:commandButton> |

In the above snippet code, whenever the User clicks the Calculate Tax button, the calculateTax() method defined inside the TaxCalculator class will be called. Following is how the TaxCalculator component looks like,

**TaxCalculator.java**

|  |  |
| --- | --- |
| 1 | @Name("TaxCalculator") |
| 2 | @Scope(ScopeType.PAGE) |

|  |  |  |
| --- | --- | --- |
| 3 | public class TaxCalculator{ | |
| 4 |  |

|  |  |  |
| --- | --- | --- |
| 5 | public void calculateTax(){ | |
| 6 | } |

|  |  |
| --- | --- |
| 7 | } |

7) Sample Application

With the concepts and theories in mind, let us develop a minimal functionality Shopping Application using JBoss Seam Framework. The following are the list of softwares/products needed for the Sample Application to work.

The functionality of the ***Shopping Application*** is simple. It initially provides a page prompting for the user name before beginning the Shopping Application. Then a ***List of products*** along with ***Product Id***, ***Name***, ***Description***, ***Price***, ***Number of Items*** to be checked-out are shown to the user. The user can select the product items he wants and continues to check-out. After that a brief summary of the Product Information is shown to the User .  
Since the Shopping Application it is an ***Enterprise application***, it involves the ***Creation of Jar and War files*** and then packaging them as an ***Ear File***. Let us look into the various files and the directory structure required for completing this Application.

7.1) View Files

These set of files represents the view files rendered as a result of client making the client. Let us look into the various files involved one by one.

**7.1.1) index.html**  
**index.html**

|  |  |
| --- | --- |
| 1 | <html> |
| 2 | <head> | |

|  |  |  |
| --- | --- | --- |
| 3 | <meta http-equiv="Refresh" content="0; URL=user.seam"> | |
| 4 | </head> |

|  |  |
| --- | --- |
| 5 | </html> |

This is the file that will be request when the client starts the Application in the Web Browser. This file immediately redirects touser.seam. Later, we will see that the extension seam is mapped to xhtml (for Facelets). Here is the content ofuser.xhtml.

**7.1.2) user.xhtml**

|  |  |
| --- | --- |
| 1 | <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" |
| 2 | "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>"> |

|  |  |
| --- | --- |
| 3 | <html xmlns="<http://www.w3.org/1999/xhtml>" |
| 4 | xmlns:ui="<http://java.sun.com/jsf/facelets>" | |

|  |  |
| --- | --- |
| 5 | xmlns:h="<http://java.sun.com/jsf/html>" |
| 6 | xmlns:f="<http://java.sun.com/jsf/core>"> | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | <body> | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | <center> | |

|  |  |  |
| --- | --- | --- |
| 11 | <h2>Welcome to Shopping.</h2> | |
| 12 | </center> |

|  |  |
| --- | --- |
| 13 |  |
| 14 | <h:form> | |

|  |  |  |
| --- | --- | --- |
| 15 | <center> | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | To contine shopping, Please enter your name: | |
| 18 | <br/> |

|  |  |
| --- | --- |
| 19 |  |
| 20 | <h:inputText value="#{User.name}" size="20"/><br/> | |

|  |  |  |
| --- | --- | --- |
| 21 | <h:commandButton type="submit" value="Continue Shopping" action="product-list"/> | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | </center> |
| 24 | </h:form> |

|  |  |
| --- | --- |
| 25 |  |
| 26 | </body> | |

|  |  |
| --- | --- |
| 27 | </html> |

The things to note that is there is a Component called User with a property called name. As mentioned in the preceding sections, the user will be prompted to enter his name before continue with the shopping. The entered name will be mapped to the name property in the User class. Also note that when the clicks the “Continue Shopping” button, the action is re-directed to “product-list” which actually maps to product-list.xhtml (which is defined the navigation.xml) file. Given below are the contents of product-list.xhtml file.

**7.1.3) product-list.xhtml**

|  |  |
| --- | --- |
| 1 | <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" |
| 2 | "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>"> |

|  |  |
| --- | --- |
| 3 | <html xmlns="<http://www.w3.org/1999/xhtml>" |
| 4 | xmlns:ui="<http://java.sun.com/jsf/facelets>" | |

|  |  |
| --- | --- |
| 5 | xmlns:h="<http://java.sun.com/jsf/html>" |
| 6 | xmlns:f="<http://java.sun.com/jsf/core>"> | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | <body> | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | <center> | |

|  |  |  |
| --- | --- | --- |
| 11 | <h2>List of available products</h2> | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | Hi, <b>#{User.name}</b> |
| 14 | Following are the available producst in our shopping repository | |

|  |  |  |
| --- | --- | --- |
| 15 | </center> | |
| 16 |  |

|  |  |  |
| --- | --- | --- |
| 17 | <h:form> | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | <center> |
| 20 | <h:dataTable  columnClasses="list-column-center, list-column-right, | |

|  |  |  |
| --- | --- | --- |
| 21 | list-column-center, list-column-right" headerClass="list-header" | |
| 22 | rowClasses="list-row" styleClass="list-background" |

|  |  |  |
| --- | --- | --- |
| 23 | value="#{ShoppingManager.products}" var="aProduct"> | |
| 24 |  |

|  |  |
| --- | --- |
| 25 | <h:column> |
| 26 | <h:outputText value = "#{aProduct.id}"/> | |

|  |  |  |
| --- | --- | --- |
| 27 | </h:column> | |
| 28 |  |

|  |  |
| --- | --- |
| 29 | <h:column> |
| 30 | <h:outputText value = "#{aProduct.name}"/> | |

|  |  |  |
| --- | --- | --- |
| 31 | </h:column> | |
| 32 |  |

|  |  |
| --- | --- |
| 33 | <h:column> |
| 34 | <h:outputText value = "#{aProduct.description}"/> | |

|  |  |  |
| --- | --- | --- |
| 35 | </h:column> | |
| 36 |  |

|  |  |
| --- | --- |
| 37 | <h:column> |
| 38 | <h:outputText value = "#{aProduct.price}"/> | |

|  |  |  |
| --- | --- | --- |
| 39 | </h:column> | |
| 40 |  |

|  |  |
| --- | --- |
| 41 | <h:column> |
| 42 | <h:inputText value = "#{aProduct.noOfItems}" size = "5"/> | |

|  |  |  |
| --- | --- | --- |
| 43 | </h:column> | |
| 44 | </h:dataTable> |

|  |  |  |
| --- | --- | --- |
| 45 | </center> | |
| 46 |  |

|  |  |
| --- | --- |
| 47 | <center> |
| 48 | <h:commandButton action="#{ShoppingManager.checkout}" | |

|  |  |
| --- | --- |
| 49 | value="Checkout Products"/> |
| 50 | <h:commandButton action="user" value="Go to user page" type = "submit"/> | |

|  |  |  |
| --- | --- | --- |
| 51 | </center> | |
| 52 |  |

|  |  |  |
| --- | --- | --- |
| 53 | </h:form> | |
| 54 | </body> |

|  |  |
| --- | --- |
| 55 | </html> |

Note that a class called Product with properties ***name***, ***description***, ***price***, ***noOfItems***, ***amount***, ***id (Primary key)*** is defined which represent of the products in the shopping repository. Access to the product objects such as displaying is facilitated with the help of ShoppingManager Session Bean. Note that this Session Bean is configured to act as a ***Seam Component*** in this Application. As soon as this page loads, all the ***Product information*** from the Products Table, is fetched and it is displayed in this page.  
We have two buttons at the bottom of the page. One is to navigate to the Home page and the other is to checkout the products. As soon as the user clicks the “Checkout Products” button, the checkOut() method in the ShoppingManagerbean is fired which will calculate the amount of the individual products the user has selected. Upon completion of the method, checkout action is returned which is mapped with product-summary.xhtml page as defined in thenavigation.xml file.

**7.1.4) product-summary.xhtml**  
Given below are the contents of product-summary.xhtml file which will display the summary information of the user selected products along with the total price information.  
**product-summary.xhtml**

|  |  |
| --- | --- |
| 1 | <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" |
| 2 | "<http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd>"> |

|  |  |
| --- | --- |
| 3 | <html xmlns="<http://www.w3.org/1999/xhtml>" |
| 4 | xmlns:ui="<http://java.sun.com/jsf/facelets>" | |

|  |  |
| --- | --- |
| 5 | xmlns:h="<http://java.sun.com/jsf/html>" |
| 6 | xmlns:f="<http://java.sun.com/jsf/core>"> | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | <body> | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | <center> | |

|  |  |  |
| --- | --- | --- |
| 11 | <h2>Shopping Summary</h2> | |
| 12 | Hi, <b>#{User.name}</b> |

|  |  |  |
| --- | --- | --- |
| 13 | </center> | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | <h:form> |
| 16 | <center> |

|  |  |
| --- | --- |
| 17 | <h:dataTable  columnClasses="list-column-center, list-column-right, |
| 18 | list-column-center, list-column-right" headerClass="list-header" | |

|  |  |
| --- | --- |
| 19 | rowClasses="list-row" styleClass="list-background" |
| 20 | value="#{ShoppingManager.products}" var="aProduct"> | |

|  |  |
| --- | --- |
| 21 |  |
| 22 | <h:column> | |

|  |  |  |
| --- | --- | --- |
| 23 | <h:outputText value = "#{aProduct.id}"/> | |
| 24 | </h:column> |

|  |  |
| --- | --- |
| 25 |  |
| 26 | <h:column> | |

|  |  |  |
| --- | --- | --- |
| 27 | <h:outputText value = "#{aProduct.name}"/> | |
| 28 | </h:column> |

|  |  |
| --- | --- |
| 29 |  |
| 30 | <h:column> | |

|  |  |  |
| --- | --- | --- |
| 31 | <h:outputText value = "#{aProduct.description}"/> | |
| 32 | </h:column> |

|  |  |
| --- | --- |
| 33 |  |
| 34 | <h:column> | |

|  |  |  |
| --- | --- | --- |
| 35 | <h:outputText value = "#{aProduct.price}"/> | |
| 36 | </h:column> |

|  |  |
| --- | --- |
| 37 |  |
| 38 | <h:column> | |

|  |  |  |
| --- | --- | --- |
| 39 | <h:outputText value = "#{aProduct.noOfItems}" size = "5"/> | |
| 40 | </h:column> |

|  |  |
| --- | --- |
| 41 |  |
| 42 | <h:column> | |

|  |  |  |
| --- | --- | --- |
| 43 | <h:outputText value = "#{aProduct.amount}" size = "5"/> | |
| 44 | </h:column> |

|  |  |
| --- | --- |
| 45 |  |
| 46 | </h:dataTable> | |

|  |  |  |
| --- | --- | --- |
| 47 | </center> | |
| 48 |  |

|  |  |
| --- | --- |
| 49 | <center> |
| 50 | Total Price: | |

|  |  |  |
| --- | --- | --- |
| 51 | <b><h:outputText value = "#{ShoppingManager.totalPrice}" /></b> | |
| 52 | Happy Shopping. |

|  |  |  |
| --- | --- | --- |
| 53 | <h:commandButton action="user" value="Go to user page" type = "submit"/> | |
| 54 | </center> |

|  |  |
| --- | --- |
| 55 |  |
| 56 | </h:form> | |

|  |  |  |
| --- | --- | --- |
| 57 | </body> | |
| 58 | </html> |

7.2) Resource Files

In Seam terms, all the ***Configuration Information*** (web.xml, application.xml) and the various ***Resources Messages***are defined in terms of resources. Let us have a look into the various Configuration file.

**7.2.1) application.xml**  
**application.xml**

|  |  |
| --- | --- |
| 1 | <application> |
| 2 | <display-name>Seam Hello World</display-name> | |

|  |  |
| --- | --- |
| 3 |  |
| 4 | <module> | |

|  |  |
| --- | --- |
| 5 | <web> |
| 6 | <web-uri>app.war</web-uri> | |

|  |  |  |
| --- | --- | --- |
| 7 | <context-root>/Shopping</context-root> | |
| 8 | </web> |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | </module> | | |
| 10 | |  |

|  |  |
| --- | --- |
| 11 | <module> |
| 12 | <ejb>app.jar</ejb> | |

|  |  |  |
| --- | --- | --- |
| 13 | </module> | |
| 14 |  |

|  |  |
| --- | --- |
| 15 | <module> |
| 16 | <java>jboss-seam.jar</java> | |

|  |  |  |
| --- | --- | --- |
| 17 | </module> | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | <module> |
| 20 | <java>el-ri.jar</java> | |

|  |  |  |
| --- | --- | --- |
| 21 | </module> | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | <module> |
| 24 | <java>el-api.jar</java> | |

|  |  |  |
| --- | --- | --- |
| 25 | </module> | |
| 26 |  |

|  |  |
| --- | --- |
| 27 | </application> |

This is the ***Application Level Configuration File*** which contains entries for the ***Web*** and the ***Java Module*** along with the libraries that are specific to JBoss and Seam.

**7.2.2) ejb-jar.xml**  
**ejb-jar.xml**

|  |  |
| --- | --- |
| 1 | <ejb-jar> |
| 2 | <assembly-descriptor> | |

|  |  |
| --- | --- |
| 3 | <interceptor-binding> |
| 4 | <ejb-name>\*</ejb-name> | |

|  |  |  |
| --- | --- | --- |
| 5 | <interceptor-class>org.jboss.seam.ejb.SeamInterceptor</interceptor-class> | |
| 6 | </interceptor-binding> |

|  |  |  |
| --- | --- | --- |
| 7 | </assembly-descriptor> | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | <interceptors> | |
| 10 | | <interceptor> | |

|  |  |  |
| --- | --- | --- |
| 11 | <interceptor-class>org.jboss.seam.ejb.SeamInterceptor</interceptor-class> | |
| 12 | </interceptor> |

|  |  |  |
| --- | --- | --- |
| 13 | </interceptors> | |
| 14 | </ejb-jar> |

These entries are necessary to make all the Ejb Components look like Seam Components. This is achieved by ***intercepting***the functionality of the Ejb Components with the help of ***Interceptors***.

**7.2.3) persistence.xml**  
**persistence.xml**

|  |  |
| --- | --- |
| 1 | <persistence> |
| 2 | <persistence-unit name="ShoppingManager"> | |

|  |  |  |
| --- | --- | --- |
| 3 | <provider>org.hibernate.ejb.HibernatePersistence</provider> | |
| 4 | <jta-data-source>java:/DefaultDS</jta-data-source> |

|  |  |
| --- | --- |
| 5 |  |
| 6 | <properties> | |

|  |  |
| --- | --- |
| 7 | <property name="hibernate.dialect" value="org.hibernate.dialect.HSQLDialect"/> |
| 8 | <property name="hibernate.transaction.flush\_before\_completion" value="true"/> |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | <property name="hibernate.hbm2ddl.auto" value="create-drop"/> | | |
| 10 | | <property name="hibernate.show\_sql" value="true"/> |

|  |  |
| --- | --- |
| 11 | </properties> |
| 12 | </persistence-unit> | |

|  |  |
| --- | --- |
| 13 | </persistence> |

The above file represents the ***Persistence Unit*** that the Session Bean may use to store and retrieve information about the product Objects from the Product table. A persistence Unit generally contains information about its Database, Driver Name, Dialect etc.

**7.2.4) components.xml**  
Since we are using Session Bean for interacting with the Entity Object, we must specify the ***Jndi Name*** of the Session Bean so that Seam framework can lookup and create an instance of the Session Bean. This is achieved with the help of the following Xml File.  
**components.xml**

|  |  |
| --- | --- |
| 1 | <?xml version="1.0" encoding="UTF-8"?> |
| 2 | <components xmlns="<http://jboss.com/products/seam/components>" | |

|  |  |
| --- | --- |
| 3 | xmlns:core="<http://jboss.com/products/seam/core>" |
| 4 | xmlns:xsi="<http://www.w3.org/2001/XMLSchema-instance>" | |

|  |  |
| --- | --- |
| 5 | xsi:schemaLocation= |
| 6 | "<http://jboss.com/products/seam/core> | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | <http://jboss.com/products/seam/core-1.1.xsd> | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | |  | |

|  |  |  |
| --- | --- | --- |
| 11 | <http://jboss.com/products/seam/components> | |
| 12 |  |

|  |  |
| --- | --- |
| 13 |  |
| 14 | <http://jboss.com/products/seam/components-1.1.xsd>"> | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | <core:init jndi-pattern="Shopping/#{ejbName}/local" debug="true"/> | |

|  |  |  |
| --- | --- | --- |
| 17 | <core:manager conversation-timeout="120000"/> | |
| 18 |  |

|  |  |
| --- | --- |
| 19 | </components> |

**7.2.5) navigation.xml**  
All the navigation rules in this Shopping Application are logically expressed in terms of action in this navigation.xml file. The advantage of providing this kind of declarative mapping is that all the components will refer only to the ***Logical Outcome of the Action*** and not the ***Physical Location of the Page***.  
**A.a**

|  |  |  |
| --- | --- | --- |
| 1 | <?xml version="1.0" encoding="UTF-8"?> | |
| 2 | <!DOCTYPE faces-config |

|  |  |  |
| --- | --- | --- |
| 3 | PUBLIC "-//Sun Microsystems, Inc.//DTD JavaServer Faces Config 1.0//EN" | |
| 4 | "<http://java.sun.com/dtd/web-facesconfig_1_0.dtd>"> |

|  |  |
| --- | --- |
| 5 |  |
| 6 | <faces-config> | |

|  |  |
| --- | --- |
| 7 |  |
| 8 | <navigation-rule> | |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | <from-view-id>\*</from-view-id> | | |
| 10 | |  |

|  |  |
| --- | --- |
| 11 | <navigation-case> |
| 12 | <from-outcome>user</from-outcome> | |

|  |  |  |
| --- | --- | --- |
| 13 | <to-view-id>/user.xhtml</to-view-id> | |
| 14 | <redirect/> |

|  |  |  |
| --- | --- | --- |
| 15 | </navigation-case> | |
| 16 |  |

|  |  |
| --- | --- |
| 17 | <navigation-case> |
| 18 | <from-outcome>product-list</from-outcome> | |

|  |  |  |
| --- | --- | --- |
| 19 | <to-view-id>/product-list.xhtml</to-view-id> | |
| 20 | <redirect/> |

|  |  |  |
| --- | --- | --- |
| 21 | </navigation-case> | |
| 22 |  |

|  |  |
| --- | --- |
| 23 | <navigation-case> |
| 24 | <from-outcome>checkout</from-outcome> | |

|  |  |  |
| --- | --- | --- |
| 25 | <to-view-id>/product-summary.xhtml</to-view-id> | |
| 26 | <redirect/> |

|  |  |  |
| --- | --- | --- |
| 27 | </navigation-case> | |
| 28 | </navigation-rule> |

|  |  |
| --- | --- |
| 29 |  |
| 30 | </faces-config> | |

7.3) Source Files

Now let us look into the Various Java Source Files (representing Session Beans, Entities, etc). Following is the code for the Product Entity.

**7.3.1) Product.java**  
**Product.java**

|  |  |  |
| --- | --- | --- |
| 1 | package net.javabeat.articles.jboss.seam.shopping; | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | import org.jboss.seam.annotations.\*; | |
| 4 | import javax.persistence.\*; |

|  |  |  |
| --- | --- | --- |
| 5 | import java.io.Serializable; | |
| 6 |  |

|  |  |  |
| --- | --- | --- |
| 7 | import static org.jboss.seam.ScopeType.SESSION; | |
| 8 |  |

|  |  |  |
| --- | --- | --- |
| 9 | @Entity | |
| 10 | | @Name("person") | |

|  |  |
| --- | --- |
| 11 | @Scope (SESSION) |
| 12 | public class Product { | |

|  |  |
| --- | --- |
| 13 |  |
| 14 | private String name; | |

|  |  |  |
| --- | --- | --- |
| 15 | private String description; | |
| 16 | private double price; |

|  |  |
| --- | --- |
| 17 | private int noOfItems; |
| 18 | private double amount; |

|  |  |  |
| --- | --- | --- |
| 19 | private long id; | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | public Product() { | |
| 22 | } |

|  |  |
| --- | --- |
| 23 |  |
| 24 | public String getName() { | |

|  |  |  |
| --- | --- | --- |
| 25 | return name; | |
| 26 | } |

|  |  |
| --- | --- |
| 27 |  |
| 28 | public void setName(String name) { | |

|  |  |  |
| --- | --- | --- |
| 29 | this.name = name; | |
| 30 | } |

|  |  |
| --- | --- |
| 31 |  |
| 32 | public String getDescription() { | |

|  |  |  |
| --- | --- | --- |
| 33 | return description; | |
| 34 | } |

|  |  |
| --- | --- |
| 35 |  |
| 36 | public void setDescription(String description) { | |

|  |  |  |
| --- | --- | --- |
| 37 | this.description = description; | |
| 38 | } |

|  |  |
| --- | --- |
| 39 |  |
| 40 | public double getPrice() { | |

|  |  |  |
| --- | --- | --- |
| 41 | return price; | |
| 42 | } |

|  |  |
| --- | --- |
| 43 |  |
| 44 | public void setPrice(double price) { | |

|  |  |  |
| --- | --- | --- |
| 45 | this.price = price; | |
| 46 | } |

|  |  |
| --- | --- |
| 47 |  |
| 48 | public int getNoOfItems() { | |

|  |  |  |
| --- | --- | --- |
| 49 | return noOfItems; | |
| 50 | } |

|  |  |
| --- | --- |
| 51 |  |
| 52 | public void setNoOfItems(int noOfItems) { | |

|  |  |  |
| --- | --- | --- |
| 53 | this.noOfItems = noOfItems; | |
| 54 | } |

|  |  |
| --- | --- |
| 55 |  |
| 56 | public double getAmount(){ | |

|  |  |  |
| --- | --- | --- |
| 57 | return amount; | |
| 58 | } |

|  |  |
| --- | --- |
| 59 |  |
| 60 | public void setAmount(double amount){ | |

|  |  |  |
| --- | --- | --- |
| 61 | this.amount = amount; | |
| 62 | } |

|  |  |
| --- | --- |
| 63 |  |
| 64 | @Id @GeneratedValue | |

|  |  |  |
| --- | --- | --- |
| 65 | public long getId() { | |
| 66 | return id; |

|  |  |  |
| --- | --- | --- |
| 67 | } | |
| 68 |  |

|  |  |  |
| --- | --- | --- |
| 69 | public void setId(long id) { | |
| 70 | this.id = id; |

|  |  |  |
| --- | --- | --- |
| 71 | } | |
| 72 | } |

Note that apart from qualifying this as an Entity, it has to be qualified with Annotations like @Name and @Scope to tell to the Seam Framework that it is a ***Seam Component***. The ***Session Bean*** will store and retrieve the values from the Productobjects fetched from the Product Table.

**7.3.2) Session Bean Interface**  
Following is the interface definition of the Session Bean. This Bean has various functionalities like ***Creating, Retrieving and Destroying products***. Other that this it also performs the checkout operation which is nothing but the calculation of the no of items with the individual products along with the total amount calculation.  
**Shopping.java**

|  |  |  |
| --- | --- | --- |
| 1 | package net.javabeat.articles.jboss.seam.shopping; | |
| 2 |  |

|  |  |
| --- | --- |
| 3 | import javax.ejb.\*; |
| 4 | import java.util.\*; |

|  |  |  |
| --- | --- | --- |
| 5 | import net.javabeat.articles.jboss.seam.shopping.Product; | |
| 6 |  |

|  |  |
| --- | --- |
| 7 | @Local |
| 8 | public interface Shopping { | |

|  |  |  |
| --- | --- | --- |
| 9 |  | |
| 10 | | public List  getProducts(); | |

|  |  |
| --- | --- |
| 11 | public void createProducts(); |
| 12 | public void destroyProducts(); | |

|  |  |
| --- | --- |
| 13 | public String checkout(); |
| 14 | public double getTotalPrice(); | |

|  |  |
| --- | --- |
| 15 |  |
| 16 | } | |

**7.3.3) Session Bean Implementation**  
**ShoppingManager.java**

[view source](http://www.javabeat.net/introduction-to-jboss-seam/#viewSource)

[print](http://www.javabeat.net/introduction-to-jboss-seam/#printSource)[?](http://www.javabeat.net/introduction-to-jboss-seam/#about)

|  |  |  |
| --- | --- | --- |
| 1 | package net.javabeat.articles.jboss.seam.shopping; | |
| 2 | import java.util.List; |

|  |  |
| --- | --- |
| 3 | import javax.ejb.\*; |
| 4 | import net.javabeat.articles.jboss.seam.shopping.Product; | |

|  |  |  |
| --- | --- | --- |
| 5 | import net.javabeat.articles.jboss.seam.shopping.ProductDatabase; | |
| 6 | import org.jboss.seam.annotations.\*; |

|  |  |  |
| --- | --- | --- |
| 7 | import org.jboss.seam.ejb.\*; | |
| 8 |  |

|  |  |  |  |
| --- | --- | --- | --- |
| 9 | import static org.jboss.seam.ScopeType.SESSION; | | |
| 10 | |  |

|  |  |  |
| --- | --- | --- |
| 11 | import javax.persistence.\*; | |
| 12 |  |

|  |  |
| --- | --- |
| 13 | @Stateful |
| 14 | @Name("ShoppingManager") | |

|  |  |
| --- | --- |
| 15 | @Scope (SESSION) |
| 16 | public class ShoppingManager implements Shopping { | |

|  |  |
| --- | --- |
| 17 |  |
| 18 | private List  products; | |

|  |  |  |
| --- | --- | --- |
| 19 | private double totalPrice; | |
| 20 |  |

|  |  |  |
| --- | --- | --- |
| 21 | @PersistenceContext(unitName = "ShoppingManager") | |
| 22 | private EntityManager entityManager; |

|  |  |
| --- | --- |
| 23 |  |
| 24 | @Create | |

|  |  |  |
| --- | --- | --- |
| 25 | public void createProducts () { | |
| 26 |  |

|  |  |
| --- | --- |
| 27 | Product product = null; |
| 28 | product = createProduct("Sony Playstation", | |

|  |  |  |
| --- | --- | --- |
| 29 | "Gaming Console The Sony Playstation 2", 9000d, 0); | |
| 30 | entityManager.persist(product); |

|  |  |
| --- | --- |
| 31 |  |
| 32 | product = createProduct("Traffic Signal - DVD", | |

|  |  |  |
| --- | --- | --- |
| 33 | "SKU-CODE: Traffic Signal - Sony &amp; BMG", 240d, 0); | |
| 34 | entityManager.persist(product); |

|  |  |
| --- | --- |
| 35 |  |
| 36 | product = createProduct("DVD Player", | |

|  |  |  |
| --- | --- | --- |
| 37 | "Samsung P475 DVD Player", 3990d, 0); | |
| 38 | entityManager.persist(product); |

|  |  |
| --- | --- |
| 39 |  |
| 40 | product = createProduct("COMPAQ Notebook", | |

|  |  |  |
| --- | --- | --- |
| 41 | "COMPAQ Notebook PC - PRESARIO V6307", 49000d, 0); | |
| 42 | entityManager.persist(product); |

|  |  |
| --- | --- |
| 43 |  |
| 44 | products = entityManager.createQuery( | |

|  |  |  |
| --- | --- | --- |
| 45 | "select prdt from Product prdt").getResultList(); | |
| 46 | } |

|  |  |
| --- | --- |
| 47 |  |
| 48 | private Product createProduct(String name, String description, | |

|  |  |  |
| --- | --- | --- |
| 49 | double price, int noOfItems){ | |
| 50 |  |

|  |  |  |
| --- | --- | --- |
| 51 | Product product = new Product(); | |
| 52 | product.setName(name); |

|  |  |  |
| --- | --- | --- |
| 53 | product.setDescription(description); | |
| 54 | product.setPrice(price); |

|  |  |  |
| --- | --- | --- |
| 55 | product.setNoOfItems(noOfItems); | |
| 56 | return product; |

|  |  |  |
| --- | --- | --- |
| 57 | } | |
| 58 |  |

|  |  |  |
| --- | --- | --- |
| 59 | public List  getProducts() { | |
| 60 | return products; |

|  |  |  |
| --- | --- | --- |
| 61 | } | |
| 62 |  |

|  |  |
| --- | --- |
| 63 | public String checkout(){ |
| 64 | for(Product aProduct : products){ | |

|  |  |
| --- | --- |
| 65 | aProduct.setAmount(aProduct.getNoOfItems() \* aProduct.getPrice()); |
| 66 | totalPrice = totalPrice + (aProduct.getNoOfItems() \* aProduct.getPrice()); |

|  |  |
| --- | --- |
| 67 | } |
| 68 | return "checkout"; | |

|  |  |  |
| --- | --- | --- |
| 69 | } | |
| 70 |  |

|  |  |  |
| --- | --- | --- |
| 71 | public double getTotalPrice(){ | |
| 72 | return totalPrice; |

|  |  |  |
| --- | --- | --- |
| 73 | } | |
| 74 |  |

|  |  |
| --- | --- |
| 75 | @Destroy |
| 76 | @Remove() | |

|  |  |  |
| --- | --- | --- |
| 77 | public void destroyProducts(){ | |
| 78 | } |

|  |  |
| --- | --- |
| 79 | } |

As already explained, the above code makes use of EntityManager to create and retrieve products from the products Table.

8) Conclusion

This is an Introduction article about the evolving JBoss Seam Framework. It started with the necessity of having this Framework and then continued with the discussion of having some many advantages of using this framework. The advantages like ***“EJB-JSF”*** Integration, the need for a ***Stateful Web Application*** and the exiting ***Dependency Bijection Functionality*** is covered in much detailed manner. Then the various core modules of the Seam Framework like ***Seam Components***, the need and the usage of ***Seam Annotations*** over various modules along with the ***Various Types of Events***emited by Seam Sources are given much consideration and discussed briefly. Then the article finally ended up with a***Shopping Application*** thereby illustrating the various techniques and concepts available in JBoss Seam.