Questo capitolo offre una panoramica sui managed beans e spiega come scrivere metodi e proprietà degli stessi che vengono usati da un’applicazione JavaServer Faces. Inoltre introduce la funzionalità Bean Validation.

## 12.1 I Managed Beans nella tecnologia JavaServer Faces

Una tipica applicazione JavaServer Faces include uno o più managed beans, ognuno dei quali può essere associato con i componenti utilizzati in una pagina.

Questo capitolo introduce i concetti base di creazione, configurazione e utilizzo dei managed beans in un’applicazione.

## 1.2.1.1 Creare un managed bean

Un managed bean è creato con un costruttore senza argomenti, un set di proprietà e un set di metodi che eseguono determinate funzioni per un componente.

Ogni proprietà di un managed bean può essere legata a uno tra i seguenti elementi:

* Un valore di un componente
* Un’istanza di un componente
* Un’istanza di un convertitore
* Un’istanza di un listener
* Un’istanza di un validatore

Le funzioni più comuni che i metodi dei managed beans eseguono includono:

* Validazione dei dati dei componenti
* Gestione di un evento fatto partire dal componente
* Eseguire processamenti per determinare la prossima pagina verso cui l’applicazione deve navigare

Come per tutti i componenti JavaBeans, una proprietà consiste di un campo di dati privati e un set di metodi accessori, come mostrati nel seguente codice:

*private Integer userNumber = null;*

*...*

*public void setUserNumber(Integer user\_number) {*

*userNumber = user\_number;*

*}*

*public Integer getUserNumber() {*

*return userNumber;*

*}*

Quando si lega al valore di un componente, una proprietà di un bean può essere di un qualunque tipo primitivo e numerico o un qualunque tipo di oggetto Java tramite il quale l’applicazione ha accesso a un convertitore appropriato.

Ad esempio una proprietà può essere di tipo *java.util.Date* se l’applicazione ha accesso a un convertitore che possa convertire il tipo *Date* in una *String* e viceversa.

Vedere il capitolo 12.2 per informazioni sui tipi accettati dai tag dei componenti.

Quando una proprietà di un bean è legata a un’istanza di un componente, il tipo della proprietà deve essere lo stesso dell’oggetto componente. Ad esempio se un componente *javax.faces.component.UISelectBoolean* è legato alla proprietà, la proprietà deve accettare e ritornare un oggetto *UiSelectBoolean*.

Allo stesso modo, se la proprietà è legata a un convertitore, un validatore o un’istanza listener, la proprietà deve essere del tipo appropriato di convertitore, validatore o listener.

## 12.1.2 Usare le EL per riferirsi ai managed beans

Per legare i valori e gli oggetti dei componenti alle proprietà del managed bean o per riferirsi ai metodi del managed bean dai tags del componente, l’autore della pagina utilizza la sintassi Expression Language.

Come spiegato nel capitolo 9.1, ecco alcune delle funzionalità che offre EL:

* Analisi differita delle espressioni
* Abilità di usare una value expression sia per leggere che per scrivere dati
* Method expressions

L’analisi differita delle espressioni è importante perché il ciclo di vita di JavaServer Faces è diviso in molte fasi.

Gli eventi generati dai componenti, la conversione e validazione dei dati e la propagazione verso oggetti esterni sono eseguiti in maniera ordinata.

L’implementazione deve essere in grado di ritardare la valutazione delle espressioni fin quando non sia raggiunta la fase appropriata del ciclo vitale, perciò i tags degli attributi dell’implementazione utilizzano una sintassi di valutazione differita, che è distinguibile per l’utilizzo dei limitatori #{}.

Per immagazzinare i dati in oggetti esterni, praticamente tutti gli attributi dei tags JavaServer Faces utilizzano espressioni *lvalue*, che permettono sia di ottenere che di settare dati da oggetti esterni.

Infine, alcuni attributi dei tag dei componenti accettano espressioni di metodo (method expressions) che si riferiscono a metodi che gestiscono eventi dei componenti o validazioni o conversioni di dati dei componenti.

Per ilustrare un tag JavaServer Faces usande l’EL, il seguente tag si riferisce a un metodo che valida un input di un utente.

*<h:inputText id="inputGuess"*

*value="#{userNumberBean.userNumber}"*

*required="true" size="3"*

*disabled="#{userNumberBean.number eq userNumberBean.userNumber ...}"*

*validator="#{userNumberBean.validateNumberRange}">*

*</h:inputText>*

Questo tag unisce il valore *inputGuess* del componente alla proprietà *UserNumberBean.userNumber*  del managed bean utilizzando un’espressione *lvalue*.

Il tag utilizza un’espressione di metodo per riferirsi al metodo *UserNumberBean.validateNumberRange* che esegue la validazione del valore locale del componente.

Il valore locale è tutto ciò che l’utente digita nel campo corrispondente a questo tag.

Il metodo è invocato quando l’espressione viene analizzata.

Quasi tutti gli attributi dei tag JavaServer Faces accettano le value expressions.

In aggiunta alle proprietà del bean a cui fanno riferimento, le value expressions possono riferirsi a *list,maps,arrays*, oggetti impliciti e gruppi di risorse.

Un altro utilizzo delle value expressions è di collegare un’ istanza di un componente a una proprietà di un managed bean. L’autore di una pagina fa questo per riferirsi alla proprietà dall’attributo *binding*:

*<h:outputLabel for="fanClub"*

*rendered="false"*

*binding="#{cashierBean.specialOfferText}">*

*value="#{bundle.DukeFanClub}"/>*

*</h:outputLabel>*

Oltre a usare le espressioni con i tags standard dei componenti, si possono configurare le proprie proprietà personalizzate dei componenti per accettare espressioni tramite la creazione di istanze *javax.el.ValueExpression* o *javax.el.MethodExpression*.

## 12.2 Scrivere le proprietà dei Bean

Come spiegato nel capitolo precedente, una proprietà di un managed bean può essere legata a uno dei seguenti oggetti:

- Un valore di un componente

- Un’istanza di un componente

- Un’istanza di un convertitore

- Un’istanza di un listener

- Un’istanza di un validatore

Queste proprietà seguono le convenzioni di componenti JavaBeans (chiamati anche beans).

Il tag di un componente lega il valore del componente a una proprietà di un managed bean tramite l’utilizzo del proprio attributo *value* e lega l’istanza di un componente a una proprietà di un managed bean tramite l’attributo *binding*.

Allo stesso modo tutti i tag di convertitori, listeners e validatori usano l’attributo *binding* per legare le loro implementazioni associate alle proprietà dei managed bean.

Per legare un’istanza di un componente a una proprietà di un managed bean la proprietà deve avere lo stesso tipo del componente. Ad esempio se una proprietà di un managed bean è legata a un’istanza *UISelectBoolean* la proprietà deve accettare e ritornare un valore *UISelectBoolean*.

Allo stesso modo per legare l’implementazione di un convertitore, listener o validatore a una proprietà di un managed bean, la proprietà deve accettare e ritornare lo stesso tipo di oggetto.

Il resto del capitolo spiega come scrivere proprietà che possano essere legate al valore di un componente, a ulle sue istanze per gli oggetti componenti descritti nel capitolo 10.2 e a implementazioni di convertitori, validatori e listeners.

## 12.2.1 Scrivere proprietà legate ai valori dei componenti

Per scrivere una proprietà di un managed bean legata al valore di un componente, bisogna far coincidere il tipo della proprietà e del valore del componente.

La tabella sottostante elenca i tipi di valori accettabili:



Quando lega i componenti e le proprietà usando l’attributo *value* del tag del componente, l’autore della pagina deve assicurarsi che le proprietà corrispondenti abbiano lo stesso tipo del valore dei componenti.

## 12.2.1.1 Proprietà di UIInput e di UIOutput

Le classi *UIInput* e *UIOutput* del componente sono rappresentate dai tag dei componenti che iniziano con *h:input* e *h:output*.

Nel seguente esempio un tag *h:inputText* lega il nome del componente col nome della proprietà del managed bean chiamato *CashierBean*:

*<h:inputText id="name"*

*size="30"*

*value="#{cashierBean.name}"*

*...>*

*</h:inputText>*

Il seguente frammento di codice del managed bean *CashierBean* mostra il tipo della proprietà del bean legato al precedente tag del componente:

*protected String name = null;*

*public void setName(String name) {*

*this.name = name;*

*}*

*public String getName() {*

*return this.name;*

*}*

Come descritto nel capitolo 11.1, per convertire il valore di un componente input o output si può o applicare un convertitore o creare una proprietà del bean legata al componente con tipo coincidente.

Qua sotto c’è un tag d’esempio che mostra la date in cui un oggetto verrà imbarcato:

*<h:outputText value="#{cashierBean.shipDate}">*

*<f:convertDateTime type="date" dateStyle="full" />*

*</h:outputText>*

La proprietà del bean rappresentata da questo tag deve avere come tipo un *java.util.Date*.

Il seguente frammento di codice mostra la proprietà *shipDate* del managed bean *CashierBean* che è legata tramite il valore del tag del precedente esempio:

*private Date shipDate;*

*public Date getShipDate() {*

*return this.shipDate;*

*}*

*public void setShipDate(Date shipDate) {*

*this.shipDate = shipDate;*

*}*

## 12.2.1.2 Proprietà UIData

La classe UIData del componente è rappresentata dal tag del componente *h:dataTable*.

I componenti *UIData* devono essere legati a uno dei tipi delle proprietà di un managed bean elencati nella tabella a 12.1. I componenti *Data* sono spiegati nel capitolo 10.2.12.

Viene qui riportata parte del tag di inizio di *dataTable* dall’esempio contenuto in quel capitolo:

*<h:dataTable id="items"*

*...*

*value="#{cart.items}"*

*...*

*var="item">*

La value expression punta alla proprietà *items* del bean carrello della spesa chiamato *cart*.

Il bean *cart* conserva una map dei beans *ShoppingCartItem*.

Il metodo *getItems* del bean *cart* inserisce una *List* di istanze*ShoppingCartItem* salvate nella mappa quando l’utente aggiunge libri al carrello, come mostrato nel seguente frammento di codice:

*public synchronized List<ShoppingCartItem> getItems() {*

*List<ShoppingCartItem> results = new ArrayList<ShoppingCartItem>();*

*results.addAll(this.items.values());*

*return results;*

*}*

Tutti i componenti contenuti nel componente *UIData* sono legati alle proprietà del bean *cart* che è legato all’intero componente *UiData*.

Ad esempio, ecco il tag *h:outputText* che mostra il titolo del libro nella tabella:

*<h:commandLink action="#{showcart.details}">*

*<h:outputText value="#{item.item.title}"/>*

*</h:commandLink>*

Il titolo è un link alla pagina *bookdetails.xhtml*.

Il tag *h:outputText* utilizza la value expression *#{item.item.title}* per legare il componente *UIOutput* alla proprietà *title* dell’entità *Book*.

Il primo oggetto nell’espressione è l’istanza *ShoppingCartItem* a cui il tag *h:dataTable* fa riferimento mentre effettua il rendering della riga corrente.

Il secondo oggetto a cui fa riferimento è la proprietà *item* di *ShoppingCartItem* che ritorna un *Object* (in questo caso un libro).

Il *title* nell’espressione fa riferimento alla proprietà *title* di *Book*.

Il valore del componente *UIOutput* corrispondente a questo tag è legato alla proprietà *title* dell’entità *Book*:

*private String title;*

*...*

*public String getTitle() {*

*return title;*

*}*

*public void setTitle(String title) {*

*this.title = title;*

*}*

## 12.2.1.3 Proprietà di *UISelectBoolean*

La classe *UISelectBoolean* del componente è rappresentata dal tag del componente *h:selectBooleanCheckbox*.

Le proprietà del managed bean che gestiscono i dati dei componenti *UISelectBoolean* devono essere di tipo *boolean* o *Boolean*.

Il tag *selectBooleanCheckbox* dell’esempio nel capitolo 10.2.8 lega un componente a una proprietà.

Il seguente esempio mostra un tag che lega un valore di un componente a una proprietà *boolean*:

*<h:selectBooleanCheckbox title="#{bundle.receiveEmails}"*

*value="#{custFormBean.receiveEmails}">*

*</h:selectBooleanCheckbox>*

*<h:outputText value="#{bundle.receiveEmails}">*

Ed ecco una proprietà di esempio che può essere legata al componente rappresentato dal tag di esempio:

*private boolean receiveEmails = false;*

*...*

*public void setReceiveEmails(boolean receiveEmails) {*

*this.receiveEmails = receiveEmails;*

*}*

*public boolean getReceiveEmails() {*

*return receiveEmails;*

*}*

## 12.2.1.4 Proprietà di UISelectMany

La classe *UISelectMany* del componente è rappresentata dai tag dei componenti che iniziano con *h:SelectMany*.

Dato che un componente *UISelectMany*  permette a un utente di selezionare uno o più oggetti da una lista di oggetti, questo componente deve essere associato a una proprietà di un bean di tipo *List* o *array*.

Questa proprietà rappresenta un set di oggetti selezionati dalla lista degli oggetti disponibili.

Il seguente esempio di tag *selectManyCheckbox* viene dal capitolo 10.2.9:

*<h:selectManyCheckbox id="newslettercheckbox"*

*layout="pageDirection"*

*value="#{cashierBean.newsletters}">*

*<f:selectItems value="#{cashierBean.newsletterItems}"/>*

*</h:selectManyCheckbox>*

Sotto riportiamo la proprietà del bean che è associata al valore del tag *selectManyCheckbox* del precedente esempio:

*private String[] newsletters;*

*public void setNewsletters(String[] newsletters) {*

*this.newsletters = newsletters;*

*}*

*public String[] getNewsletters() {*

*return this.newsletters;*

*}*

I componenti *UISelectItem* e *UISelectItems* vengono usati per rappresentare tutti i valori in un componente *UISelectMany* e si rimanda ai capitoli a loro dedicati per informazioni.

## 12.2.1.5 Proprietà di *UISelectOne*

La classe *UISelectOne* del componenteè rappresentata dai tags del componente che iniziano con *h:selectOne*.

Le proprietà *UISelectOne* accettano gli stessi tipi delle proprietà *UIInput* e *UIOutput* perché il componente *UISelectOne* rappresenta il singolo oggetto selezionato del set i oggetti.

Tale oggetto può essere un qualunque tipo primitivo per cui si può applicare un convertitore.

Ecco un esempio di tag *h:selectOneMenu* preso da 10.2.8.2:

*<h:selectOneMenu id="shippingOption"*

*required="true"*

*value="#{cashierBean.shippingOption}">*

*<f:selectItem itemValue="2"*

*itemLabel="#{bundle.QuickShip}"/>*

*<f:selectItem itemValue="5"*

*itemLabel="#{bundle.NormalShip}"/>*

*<f:selectItem itemValue="7"*

*itemLabel="#{bundle.SaverShip}"/>*

*</h:selectOneMenu>*

Ed ecco riportata la proprietà del bean corrispondente a questo tag:

*private String shippingOption = "2";*

*public void setShippingOption(String shippingOption) {*

*this.shippingOption = shippingOption;*

*}*

*public String getShippingOption() {*

*return this.shippingOption;*

*}*

Notare che *shippingOption* rappresenta l’oggetto correntemente selezionato dalla lista nel componente *UISelectOne*.

## 12.2.1.6 Proprietà di *UISelectItem*

Un componente *UISelectItem* rappresenta un singolo valore in un set di valori contenuti in un componente *UISelectMany* o *UISelectOne*.

Un componente *UISelectItem* deve essere legato a una proprietà del managed bean di tipo *javax.faces.model.SelectItem*.

In oggetto *SelectItem* è composto da un *Object* che rappresenta il valore e da due stringhe rappresentanti la label e la descrizione dell’oggetto *UISelectItem*.

Il tag d’esempio preso dal sottocapitolo “Proprietà di *UISelectOne*” contiene tags *selectItem* che settano i valori della lista di oggetti nella pagina.

Riportiamo un esempio di una proprietà del bean che può settare i valori della lista:

*SelectItem itemOne = null;*

*SelectItem getItemOne(){*

*return itemOne;*

*}*

*void setItemOne(SelectItem item) {*

*itemOne = item;*

*}*

## 12.2.1.7 Proprietà di *UISelectItems*

I componenti *UISelectItems* sono figli dei componenti *UISelectMany* e *UISelectOne*.

Ogni componente *UISelectItems* è composto da un set di istanze *UISelectItem* o da una collezione di oggetti sotto forma di arraya, list o anche POJO’s.

Il seguente frammento di codice preso da *CashierBean* mostra come scrivere proprietà per i tags *selectItems* contenenti istanze *SelectItem*:

*private String[] newsletters;*

*private static final SelectItem[] newsletterItems = {*

*new SelectItem("Duke's Quarterly"),*

*new SelectItem("Innovator's Almanac"),*

*new SelectItem("Duke's Diet and Exercise Journal"),*

*new SelectItem("Random Ramblings")*

*};*

*...*

*public void setNewsletters(String[] newsletters) {*

*this.newsletters = newsletters;*

*}*

*public String[] getNewsletters() {*

*return this.newsletters;*

*}*

*public SelectItem[] getNewsletterItems() {*

*return newsletterItems;*

*}*

Qui la proprietà *newsletters* rappresenta l’oggetto *SelectItems* in cui la proprietà *newsletterItems* rappresenta un array statico di oggetti *SelectItem*.

La classe *selectItem* ha diversi costruttori, nell’esempio il primo argomento è un *Object* che rappresenta il valore dell’oggetto mentreil secondo una *String* che rappresenta la label che appare nel componente *UISelectMany* sulla pagina.

## 12.2.2 Scrivere proprietà legate a istanze del componente

Una proprietà legata a un’istanza di un componente ritorna e accetta un’istanza invece che un valore di un componente. I seguenti componenti uniscono in’istanza di un componente a una proprietà del managed bean:

*<h:selectBooleanCheckbox id="fanClub"*

*rendered="false"*

*binding="#{cashierBean.specialOffer}" />*

*<h:outputLabel for="fanClub"*

*rendered="false"*

*binding="#{cashierBean.specialOfferText}"*

*value="#{bundle.DukeFanClub}" />*

*</h:outputLabel>*

Il tag *selectBooleanCheckbox* renderizza una check box e lega il componente *fanClub*

*UISelectBoolean* alla proprietà *specialOffer* di *CashierBean*.

Il tag *outputLabel* lega il valore dell’attributo *value*, che rappresenta la label del check box, alla proprietà *specialOfferText* di *CashierBean*.

Se l’utente ordina più di 100$ di libri e clicca il bottone submit, il metodo *submit* di *CashierBean* setta entrambe le proprietà *rendered* a true, causando la visualizzazione del check box e della label quando la pagina viene ri-renderizzata.

Dato che i componenti corrispondenti ai tags di esempio sono legati alle proprietà del managed bean, tali proprietà devono avere un tipo che combacia con quello dei componenti.

*UIOutput specialOfferText = null;*

*UISelectBoolean specialOffer = null;*

*public UIOutput getSpecialOfferText() {*

*return this.specialOfferText;*

*}*

*public void setSpecialOfferText(UIOutput specialOfferText) {*

*this.specialOfferText = specialOfferText;*

*}*

*public UISelectBoolean getSpecialOffer() {*

*return this.specialOffer;*

*}*

*public void setSpecialOffer(UISelectBoolean specialOffer) {*

*this.specialOffer = specialOffer;*

*}*

## 12.2.3 Scrivere proprietà che si legano a convertitori, listeners e validatori

Tutti i tag standard di convertitori, listeners e validatori inclusi nella tecnologia JavaServer Faces supportano attributi di binding che permettono di legare tali implementazioni alle proprietà del managed bean.

Il seguente esempio mostra un tag standard *convertDateTime* che utilizza un’espressione di valore con il proprio attributo *binding* per legare l’istanza di *javax.faces.convert.DateTimeConverter* alla proprietà *convertDate* di *LoginBean*:

*<h:inputText value="#{loginBean.birthDate}">*

*<f:convertDateTime binding="#{loginBean.convertDate}" />*

*</h:inputText>*

La proprietà *convertDate* deve accettare e ritornare un oggetto *DatiTimeConverter* come mostrato qui:

*private DateTimeConverter convertDate;*

*public DateTimeConverter getConvertDate() {*

*...*

*return convertDate;*

*}*

*public void setConvertDate(DateTimeConverter convertDate) {*

*convertDate.setPattern("EEEEEEEE, MMM dd, yyyy");*

*this.convertDate = convertDate;*

*}*

Dato che il convertitore è legato a una proprietà del managed bean, esso può modificare gli attributi del convertitore o aggiungergli nuove funzionalità.

Nel caso del precedente esempio, la proprietà setta il pattern della data che il convertitore usa per analizzare l’input dell’utente nell’oggetto *Date*.

Le proprietà dei managed beans che legano le implementazioni di validatori o listeners sono scritte nello stesso modo.

## 12.3 Scrivere i metodi di un Managed Bean

I metodi di un managed bean possono eseguire diverse funzioni specifiche dell’applicazione per i componenti della pagina. Tali funzioni includono:

* Eseguire processamenti associati alla navigazione
* Gestire gli eventi azione
* Eseguire validazioni sui valori del componente
* Gestire eventi che cambiano i valori

Utilizzando un managed bean per eseguire queste funzioni, non si ha bisogno di implementare l’interfaccia *javax.faces.validator.Validator* per gestire la validazione o un’interfaccia listener per gestire gli eventi.

Inoltre, utilizzando un managed bean invece di un’implementazione di *Validator* per eseguire le validazioni, non si ha bisogno di creare tag personalizzate per l’implementazione di *Validator*.

In generale è buona abitudine includere questi metodi nello stesso managed bean che definisce le proprietà per i componenti che si riferiscono a tali metodi.

La ragione di ciò è che i metodi potrebbe aver bisogno di accedere ai dati del componente per determinare come gestire l’evento o eseguire la validazione associata al componente.

I prossimi capitoli spiegano come scrivere vari tipi di metodi dei managed beans.

## 12.3.1 Scrivere metodi per gestire la navigazione

Un **action method**, un metodo di un managed bean che gestisce il processamento della navigazione, deve essere un metodo *public* che non prende parametri e ritorna un *Object*, che è l’outcome logico che il sistema di navigazione utilizza per determinare la pagina successiva da visualizzare.

Il riferimento a questo metodo viene fatto tramite l’attributo *action* del tag del componente.

Il seguente action method è preso dal managed bean *CashierBean*, che è invocato quando un utente clicca sul bottone submit della pagina.

Se l’utente ha ordinato più di 100$ di libri, tale metodo setta le proprietò *rendered* dei componenti *funclub* e *specialOffer* a true, causandone la visualizzazione sulla pagina la prossima volta che la stessa viene renderizzata.

Dopo aver settato l’attributo *rendered* a true il metodo ritorna l’outcome logico *null* che fa in modo che la pagina venga ri-renderizzata senza creare una nuova view della stessa e conservando l’input dell’utente.

Se il metodo il metodo desse come return *purchase*, che è l’outcome logico usato per avanzare alla pagina di pagamento, la pagina verrebbe ri-renderizzata senza conservare i dati inseriti dall’utente che andrebbero persi.

Se l’utente non arriva a comprare 100$ di libri (poveraccio!) o se il componente *thankYou* è già stato renderizzato, il metodo ritorna *bookreceipt*.

L’implementazione di JavaServer Faces carica la pagina *bookreceipt.xhtml* dopo che il metodo ritorna:

*public String submit() {*

*...*

*if ((cart().getTotal()> 100.00) && !specialOffer.isRendered()) {*

*specialOfferText.setRendered(true);*

*specialOffer.setRendered(true);*

*return null;*

*} else if (specialOffer.isRendered() && !thankYou.isRendered()) {*

*thankYou.setRendered(true);*

*return null;*

*} else {*

*...*

*cart.clear();*

*return ("bookreceipt");*

*}*

*}*

Solitamente un action method ritorna un outcome di tipo *String*, come mostrato nell’esempio precedente.

In alternativa si può definire una classe *Enum* che incapsula tutte le possibili stringhe di outcome e poi creare un action method che ritorni una costante *Enum*, la quale rappresenta una *String* particolare di outcome definita dalla classe *Enum*.

Ciò è mostrato dall’esempio successivo:

*public enum Navigation {*

*main, accountHist, accountList, atm, atmAck, transferFunds,*

*transferAck, error*

*}*

Quando il risultato di ritorno è un outcome, un action method utilizza la notazione *dot (il punto fermo “.”)* per riferirsi all’outcome dalla classe *Enum*:

*public Object submit(){*

*...*

*return Navigation.accountHist;*

*}*

## 12.3.2 Scrivere un metodo per gestire un action event

Un metodo di un managed bean che gestisce un evento azione deve essere un metodo *public* che accetta un action event e ritorna *void*.

Il riferimento a tale metodo si effettua tramite l’utilizzo dell’attributo *actionListener* del tag del componente.

Solo i componenti che implementano *javax.faces.component.ActionSource* possono far riferimento a questo metodo.

Nel seguente esempio un metodo di un managed bean chiamato *ActionBean* processa l’evento rappresentato da un utente che clicca un dei link della pagina.

*public void chooseBookFromLink(ActionEvent event) {*

*String current = event.getComponent().getId();*

*FacesContext context = FacesContext.getCurrentInstance();*

*String bookId = books.get(current);*

*context.getExternalContext().getSessionMap().put("bookId", bookId);*

*}*

Questo metodo prende il componente che ha generato l’evento dall’oggetto evento, poi prende l’IS del componente, che è un codice di un libro.

Il metodo compara il codice con l’oggetto *HashMap* che contiene i codici dei libri e corrispondente valori degli ID dei vari libri.

Infine il metodo setta l’ID del libro usando il valore selezionato dall’oggetto *HashMap*.

## 12.3.3 Scrivere un metodo che esegue una validazione

Invece di implementare l’interfaccia *javax.faces.validator.Validator* per eseguire la validazione di un componente, si può includere un metodo nel managed bean che si prenda in carico la validazione degli input per tale componente.

Un metodo di un managed bean che performa la validazione deve accettare un *javax.faces.context.FacesContext*, il componente di cui deve validare i dati, e i dati da validare, proprio come fa il metodo *validate* dell’interfaccia *Validator*.

Un componente si riferisce al metodo del managed bean usando il proprio attributo *validator*.

Possono essere validati solo valori di componenti *UIInput* o di componenti che lo estendono.

Sotto viene riportato un esempio di un metodo di un managed bean che valida l’input dell’utente perso dall’esempio “quessnumber-cdi”:

*public void validateNumberRange(FacesContext context,*

*UIComponent toValidate,*

*Object value) {*

*if (remainingGuesses <= 0) {*

*((UIInput) toValidate).setValid(false);*

*FacesMessage message = new FacesMessage("No guesses left!");*

*context.addMessage(toValidate.getClientId(context), message);*

*return;*

*}*

*int input = (Integer) value;*

*if (input < minimum || input> maximum) {*

*((UIInput) toValidate).setValid(false);*

*FacesMessage message = new FacesMessage("Invalid guess");*

*context.addMessage(toValidate.getClientId(context), message);*

*}*

*}*

Il metodo *validateNumberRange* esegue due differenti validazioni:

* Se l’utente ha finito i tentativi, il metodo setta la proprietà *valid* del componente *UIInput* a false, quindi accoda un messaggio nell’istanza di *FacesContext* associando il messaggio con l’ID del componente, e ritorna.
* Se l’utente ha ancora dei tentativi, il metodo recuperare il valore locale del componente. Se il valore di input è fuori dal range, il metodo setta la proprietà *valid* di *UIInput* a false, accoda diversi messaggi dsull’istanza di *FacesContext* e ritorna.

## 12.3.4 Scrivere un metodo per gestire un evento value-change

Un managed bean che gestisce un evento che cambia il valore deve usare un metodo *public* che accetta un evento value-change e ritorna *void*.

Il riferimento a questo metodo si effettua tramite l’attributo *valueChangeListener* del componente.

Questo capitolo spiega come scrivere un metodo di un managed bean che rimpiazzi l’implementazione di *javax.faces.event.ValueChangeListener*.

Il seguente esempio è preso dal capitolo 11.2 dove il tag *h:inputTect* on *id* di *name* ha un’istanza *ValueChangeListener* registrata su di esso.

Questa istanza di *ValueChangeListener* gestisce l’evento generato dall’inserimento di un valore nel campo corrispondente al componente.

Quando l’utente inserisce un valore, si genera l’evento value-change e il metodo *processValueChange(ValueChangeEvent)* della classe *ValueChangeListener* viene invocato:

*<h:inputText id="name"*

*size="30"*

*value="#{cashierBean.name}"*

*required="true"*

*requiredMessage="#{bundle.ReqCustomerName}">*

*<f:valueChangeListener*

*type="javaeetutorial.dukesbookstore.listeners.NameChanged" />*

*</h:inputText>*

Invece di implementare *ValueChangeListener* si può scrivere un metodo di un managed bean per gestire questo evento. Per farlo si deve spostare il metodo *processValueChange(ValueChangeEvent)* dalla classe *ValueChangeListener* , chiamata *NamedChanged*, al proprio managed bean.

Ecco il codice del metodo del managed bean che processa l’evento:

*public void processValueChange(ValueChangeEvent event)*

*throws AbortProcessingException {*

*if (null != event.getNewValue()) {*

*FacesContext.getCurrentInstance().getExternalContext().*

*getSessionMap().put("name", event.getNewValue());*

*}*

*}*

Per fare in modo che questo metodo gestisca il *ValueChangeEvent* generato da un componente di input, far riferimento a questo metodo dall’attributo *valueChangeListener* del tag del componente.