Technologia i rozwiązania

Java EE 6
Tworzenie aplikacji w NetBeans 7

Obowiązkowa wiedza każdego programisty!

Helion

David R. Heffelfinger

[PACKT PUBLISHING]

Tytuł oryginału: Java EE 6 Development with NetBeans 7

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-246-8936-1

First published in the English language under the title ‘Java EE 6 Development with NetBeans’.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: http://helion.pl (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/jave6n.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/jave6n
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

5

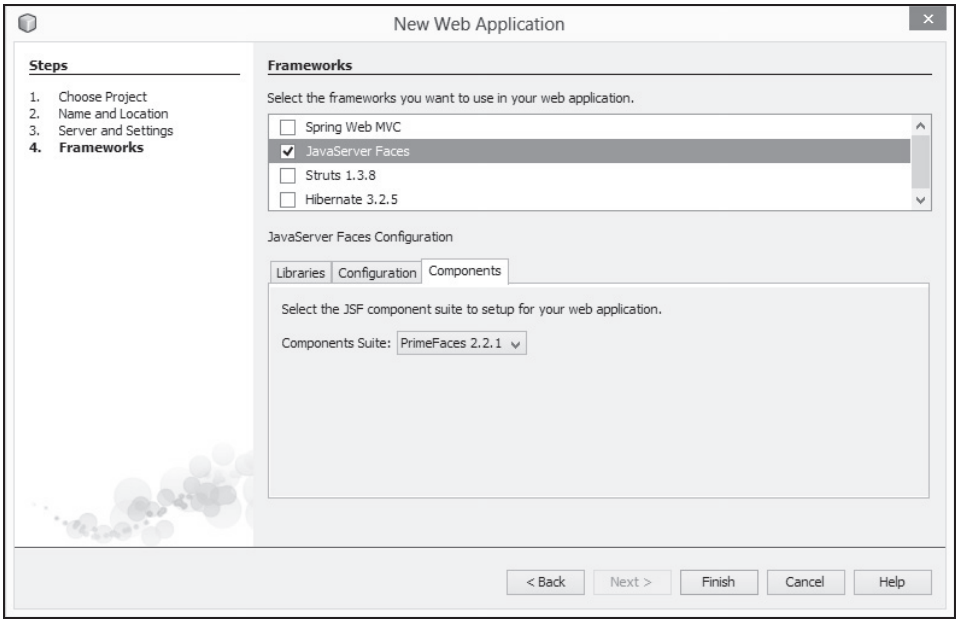
ROZDZIAŁ

Tworzenie eleganckich aplikacji sieciowych z wykorzystaniem biblioteki PrimeFaces

Jedną z zalet platformy JSF jest to, że umożliwia bardzo łatwe tworzenie niestandardowych komponentów. Dlatego powstało kilka bibliotek komponentów o otwartym dostępie do kodu źródłowego. Jedną z nich jest biblioteka PrimeFaces. Umożliwia ona łatwe tworzenie eleganckich aplikacji sieciowych. W wersji 7.0 środowiska NetBeans dostępna jest wbudowana biblioteka PrimeFaces.

Pierwszy projekt utworzony z wykorzystaniem biblioteki PrimeFaces

Aby zastosować bibliotekę PrimeFaces w projekcie, trzeba w standardowy sposób utworzyć aplikację sieciową w Javie. Gdy wybierzesz platformę JavaServer Faces, kliknij zakładkę *Components*, a następnie wybierz pozycję *PrimeFaces 2.2.1* jako pakiet komponentów (rysunek 5.1).



Rysunek 5.1. Tworzenie aplikacji z biblioteką PrimeFaces

Po utworzeniu projektu środowisko NetBeans doda potrzebne biblioteki. Na stronach JSF dostępny wtedy będzie mechanizm automatycznego uzupełniania projektu dla znaczników biblioteki PrimeFaces.

Jeśli wybierzesz PrimeFaces jako pakiet komponentów dla projektu JSF, środowisko NetBeans utworzy przykładowy projekt używający komponentów PrimeFaces. Kod wygenerowanego pliku wygląda tak:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:p="http://primefaces.prime.com.tr/ui"
   xmlns:h="http://java.sun.com/jsf/html">
   <h:head>
      <title>Facelet Title</title>
   </h:head>
   <h:body>
      <h:form>
         <p:commandButton value="Hello from PrimeFaces"
            onclick="dlg1.show();" type="button"
            />
         <p:dialog header="PrimeFaces Dialog" widgetVar="dlg1"
            width="500">
            For more information visit <a href="http://primefaces.
```

```
                org">http://primefaces.org</a>.
            </p:dialog>
        </h:form>
    </h:body>
</html>
```

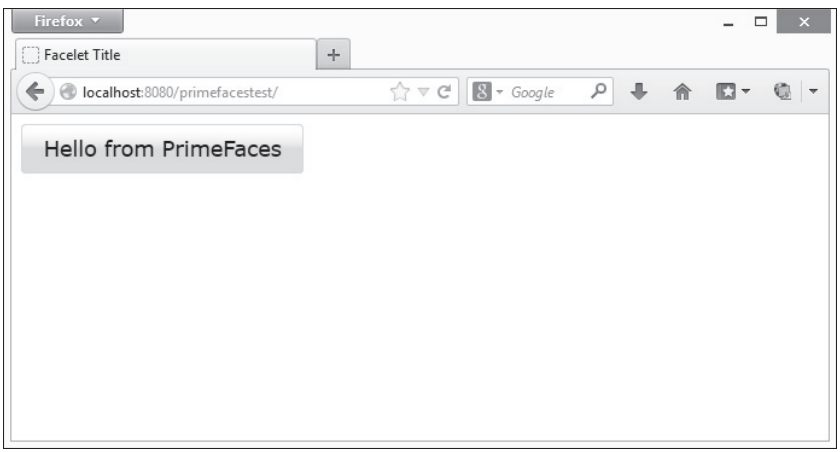
Strona ta wygląda prawie jak zwykła strona faceletu. Wyjątkiem jest kilka komponentów charakterystycznych dla biblioteki PrimeFaces.

Zwróć uwagę na to, że przestrzeń nazw biblioteki PrimeFaces (`xmlns:p="http://primefaces.prime.com.tr/ui"`) jest automatycznie dodawana do znacznika `<html>`. Ta przestrzeń nazw jest niezbędna do stosowania komponentów PrimeFaces na stronach. Zgodnie z konwencją znaczniki PrimeFaces mają przedrostek `p`.

Pierwszy komponent PrimeFaces na przedstawionej stronie to `<p:commandButton>`. Jest on podobny do standardowego komponentu przycisku polecenia JSF, jednak ma pewne dodatkowe zalety (na przykład wygląda elegancko bez konieczności ręcznego dodawania arkusza stylów CSS).

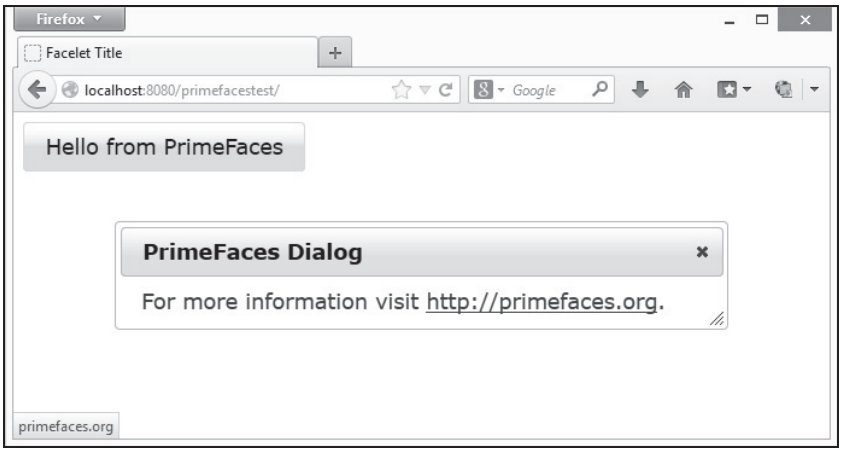
Inny komponent PrimeFaces na przykładowej stronie to `<p:dialog>`. Komponent ten jest wyświetlany jako okno dialogowe, które może znajdować się nad innymi komponentami strony. Za pomocą wartości atrybutu `widgetVar` można uzyskać dostęp do tego komponentu w innych komponentach ze strony. Okno dialogowe udostępnia w tym celu działający po stronie klienta interfejs API JavaScriptu. Najczęściej używane funkcje tego interfejsu API to `show()` i `hide()`. Służą one do wyświetlania i ukrywania okna dialogowego na stronie. Ten interfejs API wykorzystano w atrybucie `onclick` wspomnianego wcześniej przycisku polecenia.

Gdy uruchomisz aplikację, zobaczysz automatycznie wygenerowaną stronę (rysunek 5.2).



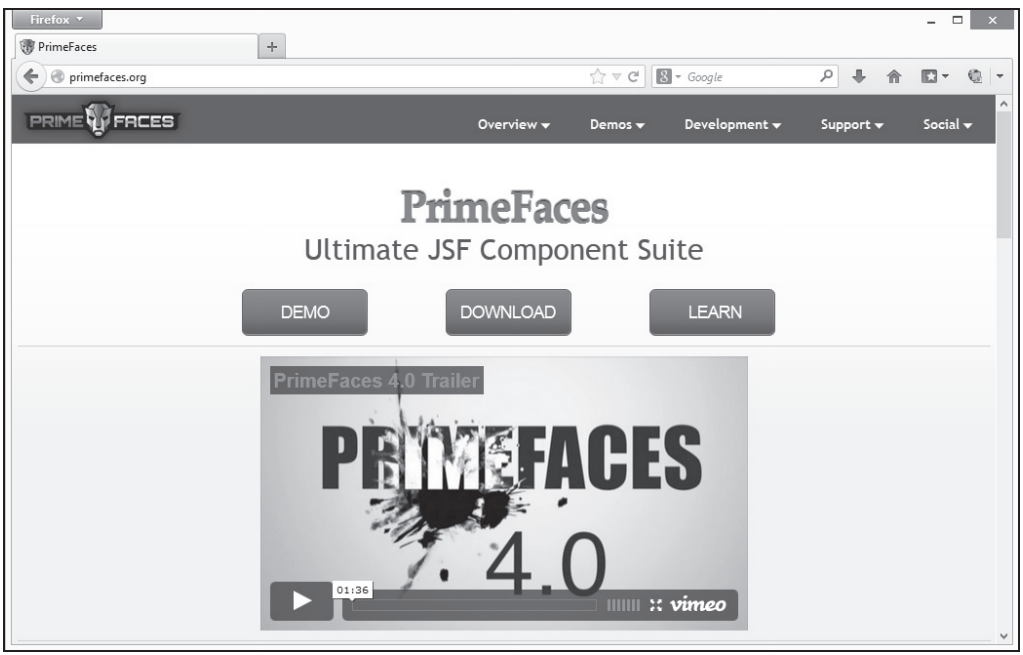
Rysunek 5.2. Automatycznie wygenerowana strona

Po kliknięciu przycisku pojawi się okno dialogowe (rysunek 5.3).



Rysunek 5.3. Okno dialogowe wyświetlane w odpowiedzi na kliknięcie przycisku

Gdy klikniesz odnośnik w oknie dialogowym, przejdziesz do witryny biblioteki PrimeFaces (rysunek 5.4).



Rysunek 5.4. Witryna poświęcona bibliotece PrimeFaces

Jak widać, biblioteka PrimeFaces umożliwia tworzenie eleganckich aplikacji sieciowych małym nakładem sił. Dalej zobaczysz, jak wykorzystać niektóre komponenty PrimeFaces, aby znacznie ułatwić i uprościć proces tworzenia aplikacji sieciowych.

Stosowanie komponentów PrimeFaces w aplikacjach JSF

Z tego podrozdziału dowiesz się, jak stosować proste komponenty PrimeFaces, aby ułatwić sobie pracę nad tworzeniem aplikacji JSF. Poniżej znajduje się kod prostej strony do wprowadzania danych na temat klientów:

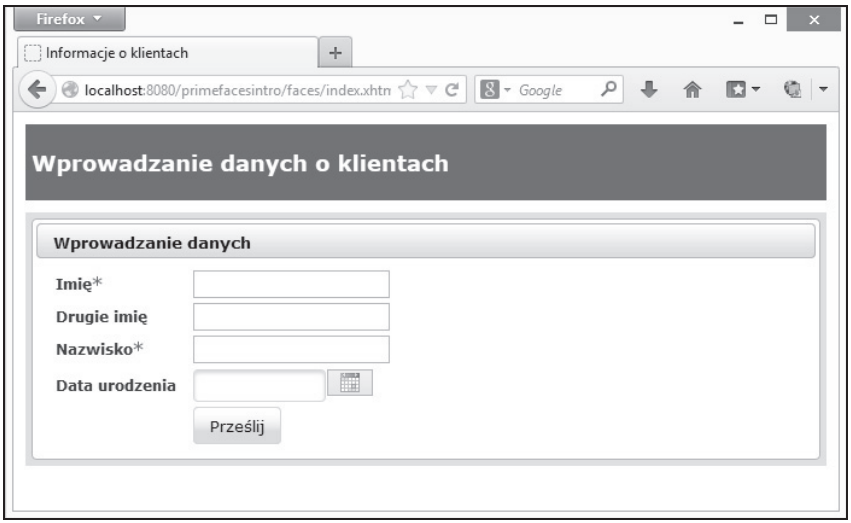
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:p="http://primefaces.prime.com.tr/ui"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets">
   <ui:composition template="./template.xhtml">
      <ui:define name="top">
         <h2>Wprowadzanie danych o klientach</h2>
      </ui:define>

      <ui:define name="content">
         <h:form>
            <p:messages/>
               <p:panel header="Wprowadzanie danych">
                  <h:panelGrid columns="2">
                     <h:outputLabel for="firstName" value="Imię"
                        styleClass="requiredLbl"/>
                     <h:inputText id="firstName" label="Imię"
                        value="#{customer.firstName}"
                        required="true"/>
                     <h:outputLabel for="middleName" value="Drugie imię"
                        styleClass="optionalLbl"/>
                     <h:inputText id="middleName" label="Drugie imię"
                        value="#{customer.middleName}"/>
                     <h:outputLabel for="lastName" value="Nazwisko"
                        styleClass="requiredLbl"/>
                     <h:inputText id="lastName" label="Nazwisko"
                        value="#{customer.lastName}"
                        required="true"/>
                     <h:outputLabel for="birthDate" value="Data urodzenia"
                        styleClass="optionalLbl"/>
                     <p:calendar id="birthDate"
                        value="#{customer.birthDate}"
                        showOn="button"
                        inputStyle="width:100px;"
                        navigator="true"/>
                  <h:panelGroup/>
```

```
                    <p:commandButton
                        value="Prześlij"
                        action="#{customerController.
                        saveCustomer}"
                        ajax="false"/>
                </h:panelGrid>
            </p:panel>
        </h:form>
    </ui:define>
</ui:composition>
</html>
```

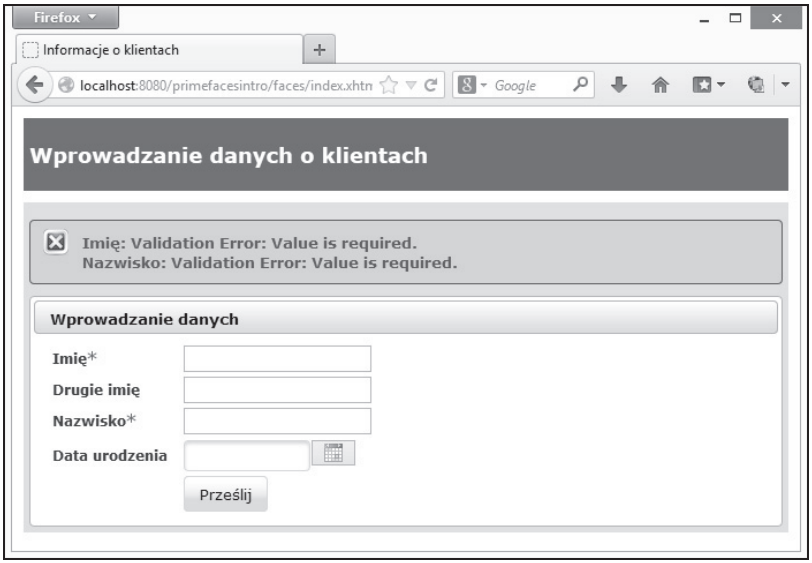
W przykładzie wykorzystano dostępny w środowisku NetBeans mechanizm generowania szablonów faceletu. Dzięki temu można „za darmo" uzyskać bardzo ciekawe style CSS. Więcej informacji o szablonach faceletów w środowisku NetBeans znajdziesz w poprzednim rozdziale.

Gdy uruchomisz projekt, przedstawiony kod zostanie wyświetlony w przeglądarce tak jak na rysunku 5.5.



Rysunek 5.5. Strona do wprowadzania danych w przeglądarce

Pierwszy nowy komponent PrimeFaces zastosowany na tej stronie to `<p:messages>`. Można go wykorzystać jako zamiennik standardowego komponentu JSF `<h:messages>`. Zaletą znacznika `<p:messages>` w porównaniu ze znacznikiem `<h:messages>` jest to, że komunikaty o błędach w znaczniku `<p:messages>` są domyślnie formatowane w elegancki sposób (rysunek 5.6).



Rysunek 5.6. Sformatowane komunikaty o błędach w znaczniku <p:messages>

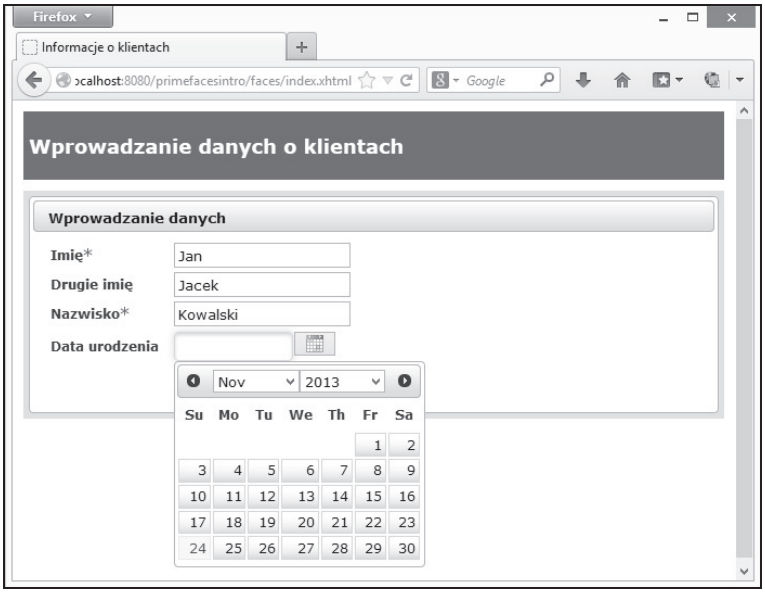
Podobnie jak w przypadku komponentu `<p:messages>`, biblioteka PrimeFaces udostępnia komponent `<p:message>`, który można potraktować jako zamiennik standardowego komponentu JSF `<h:message>` (w przykładzie nie zastosowano komponentu `<p:message>`). Znacznik `<p:message>`, podobnie jak `<h:message>`, należy stosować do wyświetlania błędów związanych ze sprawdzaniem poprawności obok pól z nieprawidłowymi danymi, a nie w górnej części strony.

Inny komponent PrimeFaces wykorzystany w przykładzie to `<p:calendar>`. Kalendarz ten można wykorzystać do wprowadzania dat. Gdy użytkownik kliknie ikonę wygenerowaną dla tego komponentu, pojawi się atrakcyjny widżet z kalendarzem (rysunek 5.7). Użytkownik może w nim wybrać datę, klikając ją.

Komponent kalendarza biblioteki PrimeFaces można w szerokim zakresie modyfikować. Domyślnie jest wyświetlany jako pole tekstowe, a gdy użytkownik je kliknie, pojawia się widżet z kalendarzem. W przykładzie atrybut `showOn` ma wartość `button`. To powoduje, że obok pola tekstowego widoczna jest ikona kalendarza. Jest to wizualna wskazówka informująca o specjalnym charakterze pola. Pozwala to także wprowadzić datę ręcznie, jeśli użytkownik woli zastosować to podejście.

Domyślnie listy miesięcy i lat są niedostępne, co utrudnia wprowadzanie dat z odległej przeszłości i przyszłości (domyślnym dniem jest zawsze dzień bieżący). Aby rozwiązać ten problem, można ustawić właściwość `navigator` na `true`, jak zrobiono w przykładowym kodzie.

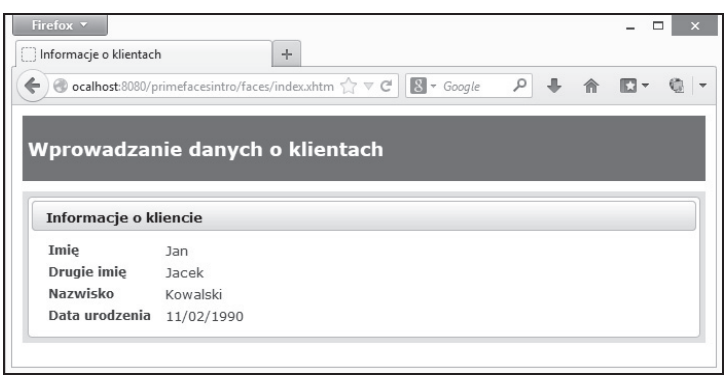
Ponadto można kontrolować wygląd pola tekstowego. Umożliwiają to atrybuty `inputStyle` i `inputStyleClass` komponentu `<p:calendar>`. Wartością atrybutu `inputStyle` musi być poprawny wewnątrzwierszowy kod CSS, natomiast w atrybucie `inputStyleClass` trzeba podać nazwę klasy CSS zdefiniowanej w jednym z arkuszy stylów CSS.



Rysunek 5.7. Komponent kalendarza umożliwia wybranie daty

Ostatnim nowym komponentem PrimeFaces użytym na przykładowej stronie jest `<p:command ↪Button>`. Domyślnie wyświetla on przycisk, który zgłasza ajaksowe żądanie aktualizacji części strony. Nie jest wtedy konieczne odświeżanie całej strony. Komponent ten można wykorzystać jako zamiennik standardowego komponentu JSF `<h:commandButton>`. W tym celu właściwość `ajax` należy ustawić na `false`, tak jak w przykładowym kodzie. Zaletą stosowania komponentu `<p:commandButton>` zamiast `<h:commandButton>` jest to, że ma on domyślnie elegancki wygląd. Dzięki temu nie trzeba tworzyć niestandardowych stylów CSS dla przycisków.

Gdy klikniesz przycisk na przykładowej stronie, przejdziesz do strony z potwierdzeniem (rysunek 5.8).



Rysunek 5.8. Strona z potwierdzeniem

Na stronie z potwierdzeniem nie ma żadnych nowych komponentów PrimeFaces, dlatego strona ta nie wymaga omówienia. Znajdziesz ją w dostępnym do pobrania kodzie dla tego rozdziału (plik *confirmation.xhtml*).

Widoki z zakładkami

W formularzach HTML często znajduje się dużo pól, przez co formularze mogą być bardzo długie. Programiści nieraz dzielą formularze na zakładki, dzięki czemu strona jest mniej przytłaczająca dla użytkowników. Zwykle tworzenie stron z zakładkami wymaga stosowania sztuczek z języków HTML i JavaScript. Jednak biblioteka PrimeFaces udostępnia komponent `<p:tabView>`, który można wykorzystać do łatwego generowania zakładek. W poniższym przykładzie pokazano, jak korzystać z tego komponentu.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:p="http://primefaces.prime.com.tr/ui"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core">
   <ui:composition template="./template.xhtml">
      <ui:define name="top">
         <h2>Wprowadzanie danych o klientach</h2>
      </ui:define>
      <ui:define name="content">
         <h:form>
            <p:messages/>
            <h:panelGrid columns="1" style="width: 100%">
               <p:tabView>
                  <p:tab title="Dane osobowe">
                     <h:panelGrid columns="2">
                        <h:outputLabel for="firstName"
                           value="Imię"
                           styleClass="requiredLbl"/>
                        <h:inputText id="firstName"
                           label="Imię"
                           value="#{customer.
                           firstName}"
                           required="true"/>
                        <h:outputLabel for="middleName"
                           value="Drugie imię"
                           styleClass="optionalLbl"/>
                        <h:inputText id="middleName"
                           label="Drugie imię"
                           value="#{customer.
                           middleName}"/>
```

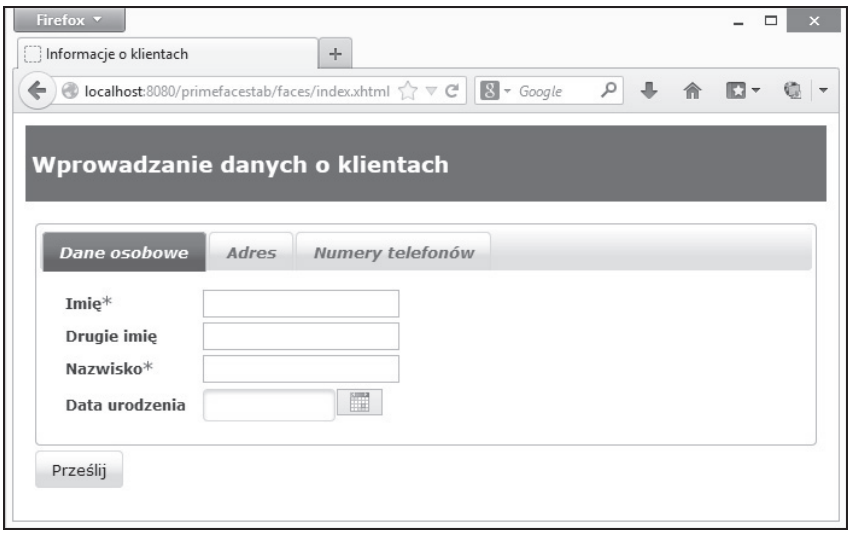
```
            <h:outputLabel for="lastName"
               value="Nazwisko"
               styleClass="requiredLbl"/>
            <h:inputText id="lastName" label="Nazwisko"
               value="#{customer.
               lastName}"
               required="true"/>
            <h:outputLabel for="birthDate"
               value="Data urodzenia"
               styleClass="optionalLbl"/>
            <p:calendar id="birthDate"
               value="#{customer.
               birthDate}"
               showOn="button"
               inputStyle="width:100px;"
               navigator="true"/>
         </h:panelGrid>
      </p:tab>
      <p:tab title="Adres">
         <h:panelGrid columns="2">
            <h:outputLabel for="line1" value="Wiersz 1."
               styleClass="requiredLbl"/>
            <h:inputText id="line1"
               value="#{customer.
               addrLine1}"
               required="true"/>
            <h:outputLabel for="line2" value="Wiersz 2."
               styleClass="optionalLbl"/>
            <h:inputText id="line2"
               value="#{customer.
               addrLine2}"/>
            <h:outputLabel for="city" value="Miasto"
               styleClass="requiredLbl"/>
            <h:inputText id="city"
               value="#{customer.
               addrCity}"
               required="true"/>
            <h:outputLabel for="state"
               value="Województwo"
               styleClass="requiredLbl"/>
            <h:selectOneMenu id="state"
               required="true"
               value="#{customer.addrState}">
               <f:selectItem itemValue=""
                  itemLabel=""/>
               <f:selectItem itemValue="LB"
                  itemLabel="Lubelskie"/>
               <f:selectItem itemValue="MZ"
                  itemLabel="Mazowieckie"/>
```

```
                    <f:selectItem itemValue="OP"
                       itemLabel="Opolskie"/>
                    <f:selectItem itemValue="PM"
                       itemLabel="Pomorskie"/>
                    <!-- Z uwagi na zwięzłość przykładu pozostałe
                         województwa zostały pominięte -->
                 </h:selectOneMenu>
                 <h:outputLabel for="zip" value="Kod"
                    styleClass="requiredLbl"/>
                 <h:inputText id="zip"
                    value="#{customer.
                    addrZip}"
                    required="true"/>
              </h:panelGrid>
           </p:tab>
           <p:tab title="Numery telefonu">
              <h:panelGrid columns="2">
                 <h:outputLabel for="homePhone"
                    value="Domowy"/>
                 <p:inputMask id="homePhone"
                    mask="(999)-999-999"
                    value="#{customer.
                    homePhone}"
                    size="12"
                    styleClass="optionalLbl"/>
                 <h:outputLabel for="mobilePhone"
                    value="Komórkowy"/>
                 <p:inputMask id="mobilePhone"
                    mask="(999)-999-999"
                    value="#{customer.
                    mobilePhone}"
                    size="12"
                    styleClass="optionalLbl"/>
                 <h:outputLabel for="workPhone"
                    value="Służbowy"/>
                 <p:inputMask id="workPhone"
                    mask="(999)-999-999"
                    value="#{customer.
                    workPhone}"
                    size="12"
                    styleClass="optionalLbl"/>
              </h:panelGrid>
           </p:tab>
        </p:tabView>
        <p:commandButton
           value="Prześlij"
           action="#{customerController.saveCustomer}"
           ajax="false"/>
     </h:panelGrid>
```

```
            </h:form>
        </ui:define>
    </ui:composition>
</html>
```

Jak widać w tym przykładzie, nadrzędnym komponentem interfejsu z zakładkami jest `<p:tabView>`. Wewnątrz niego trzeba umieścić jeden lub kilka komponentów `<p:tab>`. Każdy komponent `<p:tab>` zawiera tu pola na dane wejściowe. Pola te pojawiają się w odpowiednich zakładkach. Komponent `<p:tab>` ma atrybut `title`, którego wartość jest wyświetlana jako nagłówek zakładki.

Gdy uruchomisz projekt, zobaczysz, jak działa komponent zakładki (rysunek 5.9).

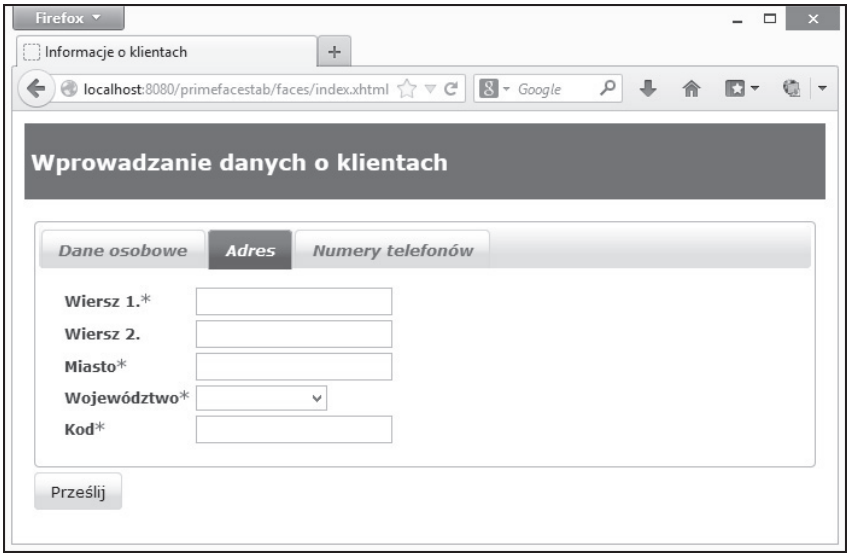


Rysunek 5.9. Zakładki na stronie

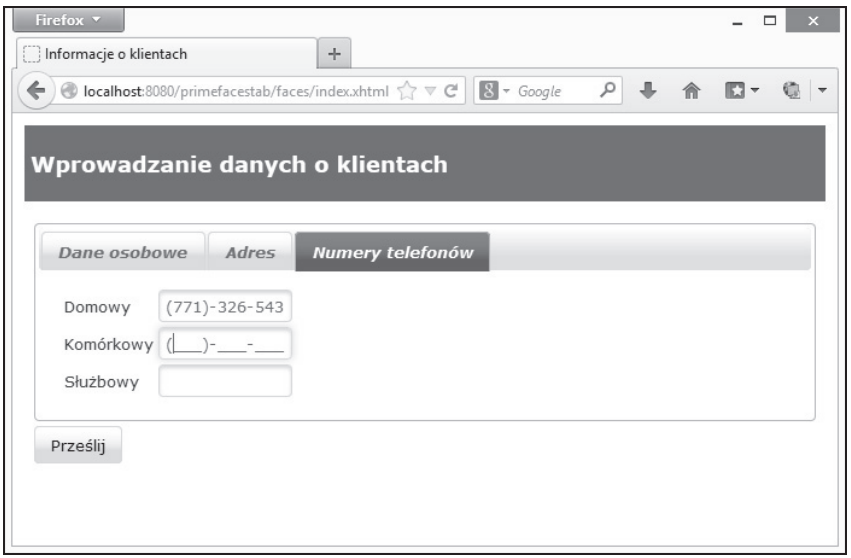
Po kliknięciu poszczególnych zakładek zobaczysz umieszczone na nich komponenty (rysunek 5.10).

Uważni Czytelnicy zapewne zauważyli, że w trzeciej zakładce pojawił się nowy komponent PrimeFaces — `<p:inputMask>`. Pozwala on uniemożliwić użytkownikom wprowadzenie źle sformatowanych danych. Tu komponent ten wykorzystano dla każdego pola na numer telefonu. Działanie tego komponentu przedstawia rysunek 5.11.

Jak widać na rysunku, gdy użytkownik kliknie komponent `inputMask`, automatycznie wyświetlany jest oczekiwany format. Użytkownik musi tylko uzupełnić puste miejsca, aby dane były poprawnie sformatowane. Tu oczekiwany jest numer telefonu w formacie `(xxx)xxx-xxx`, gdzie `x` to cyfra w systemie dziesiętnym. Przy definiowaniu masek cyfra `9` reprezentuje dowolną cyfrę, litera `a` reprezentuje dowolną literę, a gwiazdka (`*`) — dowolną cyfrę lub literę.



Rysunek 5.10. Komponenty z zakładki Adres



Rysunek 5.11. Działanie komponentu <p:inputMask>

Tu w celu formatowania danych przypisano do atrybutu `mask` komponentu `<p:inputMask>` wartość `(999)-999-999`, co pozwoliło uzyskać pożądaną maskę.

Komponent `<p:inputMask>` pozwala więc wymuszać poprawne formatowanie danych bez stosowania walidatorów JSF.

Pora wrócić do omawiania zakładek. Gdy użytkownik kliknie widoczny w dolnej części strony przycisk *Prześlij*, dane z wszystkich zakładek zostaną przesłane i potraktowane jak informacje z jednego znacznika `<h:form>`. Od tego momentu działa standardowy dla JSF proces przetwarzania żądań.

Interfejsy oparte na kreatorze

Inną (obok zakładek) popularną techniką dzielenia długich formularzy jest stosowanie kreatorów. Są one przydatne, gdy użytkownik musi uzupełnić pola w określonej kolejności. W poprzednim przykładzie nie można wymusić na użytkowniku wprowadzenia adresu przed podaniem numeru telefonu. Dlatego nie można sprawdzić, czy wprowadzone numery pasują do miejsca zamieszkania podanego w adresie. Aby rozwiązać ten problem, można wykorzystać interfejs oparty na kreatorze. Pożądany efekt można łatwo uzyskać za pomocą komponentu PrimeFaces `<p:wizard>`. Poniżej pokazano, jak zastosować ten komponent.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
   xmlns:p="http://primefaces.prime.com.tr/ui"
   xmlns:h="http://java.sun.com/jsf/html"
   xmlns:ui="http://java.sun.com/jsf/facelets"
   xmlns:f="http://java.sun.com/jsf/core">
   <ui:composition template="./template.xhtml">
      <ui:define name="top">
         <h2>Wprowadzanie danych o użytkownikach</h2>
      </ui:define>

      <ui:define name="content">
         <h:form>
            <h:panelGrid columns="1" style="width: 100%">
               <p:wizard>
                  <p:tab title="Dane osobowe"
                     id="personalInfo">
                     <p:panel header="Dane osobowe">
                        <p:messages/>
                        <h:panelGrid columns="2">

                           <h:outputLabel for="firstName"
                              value="Imię"
                              styleClass="requiredLbl"/>
                           <h:inputText id="firstName"
                              label="Imię"
                              value="#{customer.
                              firstName}"
                              required="true"/>
```

```
            <h:outputLabel for="middleName"
               value="Drugie imię"
               styleClass="optionalLbl"/>
            <h:inputText id="middleName"
               label="Drugie imię"
               value="#{customer.
               middleName}"
            />
            <h:outputLabel for="lastName"
               value="Nazwisko"
               styleClass="requiredLbl"/>
            <h:inputText id="lastName"
               label="Nazwisko"
               value="#{customer.
               lastName}"
               required="true"/>
            <h:outputLabel for="birthDate"
               value="Data urodzenia"
               styleClass="optionalLbl"/>
            <p:calendar id="birthDate"
               value="#{customer.
               birthDate}"
               showOn="button"
               inputStyle="width:100px;"
               navigator="true"/>
         </h:panelGrid>
      </p:panel>
   </p:tab>
   <!-- Zakładkę z numerami telefonów pominięto
        z uwagi na zwięzłość przykładu -->
   <p:tab title="Potwierdzenie" id="confirmation">
      <p:messages/>
      <h:panelGrid columns="2">
         <h:outputText value="Imię"
            styleClass="optionalLbl"/>
         <h:outputText id="firstNameTxt"
            value="#{customer.
            firstName}" />
         <h:outputText value="Drugie imię"
            styleClass="optionalLbl"/>
         <h:outputText id="middleNameTxt"
            value="#{customer.
            middleName}"/>
         <h:outputText value="Nazwisko"
            styleClass="optionalLbl"/>
         <h:outputText id="lastNameTxt"
            value="#{customer.
            lastName}" />
         <h:outputText value="Data urodzenia"
            styleClass="optionalLbl"/>
```

```
                        <h:outputText id="birthDateTxt"
                           value="#{customer.
                           formattedBirthDate}" />
                        <h:outputText value="Telefon domowy"
                           styleClass="optionalLbl"/>
                        <h:outputText id="homePhoneTxt"
                           value="#{customer.
                           homePhone}" />
                        <h:outputText value="Telefon komórkowy"
                           styleClass="optionalLbl"/>
                        <h:outputText id="mobilePhoneTxt"
                           value="#{customer.
                           mobilePhone}" />
                        <h:outputText value="Telefon służbowy"
                           styleClass="optionalLbl"/>
                        <h:outputText id="workPhoneTxt"
                           value="#{customer.
                           workPhone}" />
                     <h:panelGroup/>
                     <h:inputHidden value="#{customer.
                        addrState}"/>
                     <p:commandButton id="submitButton"
                        value="Prześlij"
                        actionListener=
                        "#{customerController.saveCustomer}"
                        ajax="false"/>
                     </h:panelGrid>
                  </p:tab>
               </p:wizard>
            </h:panelGrid>
         </h:form>
      </ui:define>
   </ui:composition>
</html>
```

Niektóre strony („zakładki") kreatora zostały pominięte w kodzie z uwagi na konieczność zachowania zwięzłości tego przykładu. Kompletną wersję znajdziesz w dostępnym do pobrania kodzie do tego rozdziału.

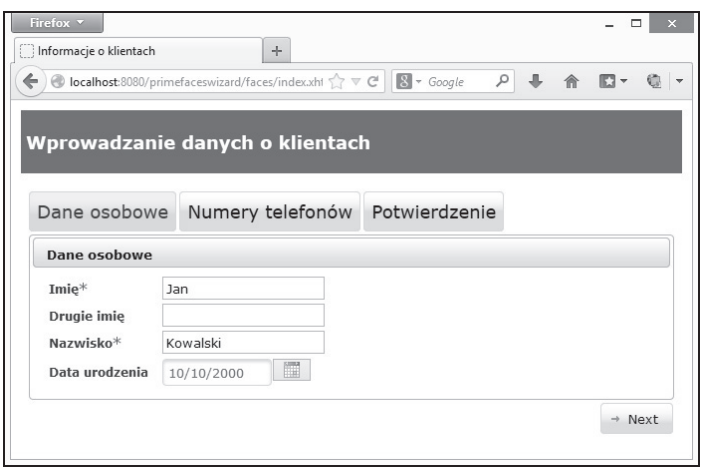
Jak widać, biblioteka PrimeFaces sprawia, że generowanie interfejsów opartych na kreatorze jest proste. Wystarczy zastosować komponent `<p:wizard>`, a następnie dodać komponent `<p:tab>` dla każdego etapu kreatora. W każdym komponencie `<p:tab>` można umieścić standardowy kod JSF.

W ostatniej zakładce znajduje się komponent `<p:commandButton>` służący do przesyłania danych na serwer. Wartość atrybutu `actionListener` tego komponentu to metoda z ziarna zarządzanego `CustomerController`.

```
package com.ensode.primefacesdemo.managedbeans;
import java.io.Serializable;
import javax.faces.bean.ManagedBean;
import javax.faces.bean.SessionScoped;
import javax.faces.event.ActionEvent;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
@ManagedBean
@SessionScoped
public class CustomerController implements Serializable {
   /** Tworzy nowy obiekt typu CustomerController */
   public CustomerController() {
   }
   public void saveCustomer(ActionEvent actionEvent) {
      // W rzeczywistej aplikacji należałoby zapisać dane.
      // W tym przykładzie kod tylko wyświetla komunikat.
      FacesMessage facesMessage = new FacesMessage(
         "Dane zostały zapisane");
      facesMessage.setSeverity(FacesMessage.SEVERITY_INFO);
      FacesContext.getCurrentInstance().addMessage(null,
         facesMessage);
   }
}
```

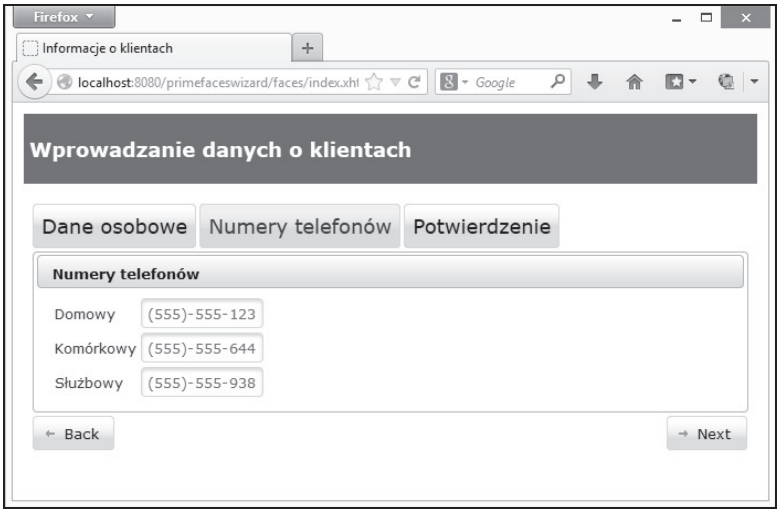
Metoda `saveCustomer()` tylko ustawia komunikat wyświetlany jako potwierdzenie na stronie. W rzeczywistej aplikacji oczywiście należałoby zapisać dane w bazie.

Na tym etapie można zobaczyć, jak działa komponent kreatora (rysunek 5.12).



Rysunek 5.12. Działanie komponentu kreatora

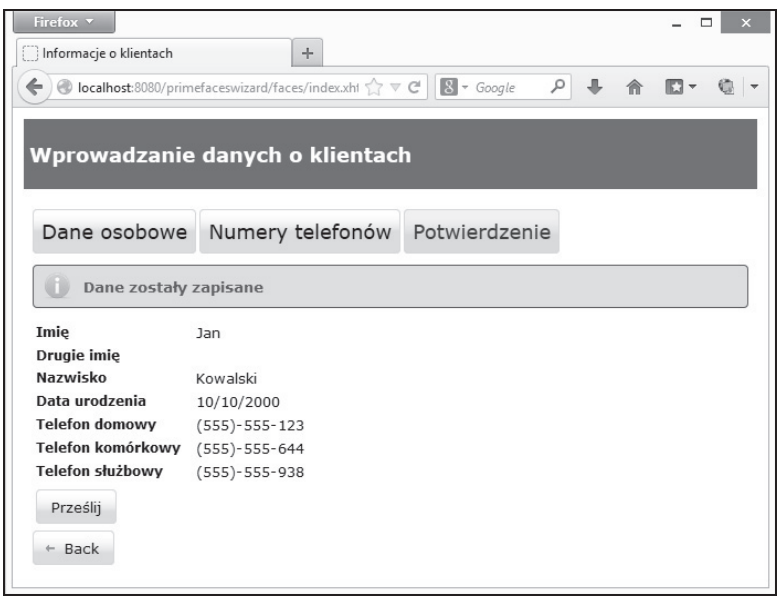
Zauważ, że komponent kreatora automatycznie dodaje przycisk *Next* w prawym dolnym rogu strony (rysunek 5.13). Kliknięcie tego przycisku powoduje przejście do następnej strony kreatora.



Rysunek 5.13. Przycisk Next pozwala przechodzić między stronami kreatora

Jak można się domyślić, komponent kreatora generuje zarówno przycisk *Next*, jak i przycisk *Back*.

Po przejściu w kreatorze do ostatniej strony i kliknięciu przycisku *Prześlij* pojawi się komunikat z potwierdzeniem wygenerowany przez metodę `saveCustomer()` z ziarna zarządzanego `CustomerController`. Komunikat ten jest wyświetlany przez komponenty `<p:messages>`, które zapewniają elegancki styl tekstu bez konieczności jawnego stosowania stylów CSS (rysunek 5.14).



Rysunek 5.14. Komponent <p:messages> zapewnia elegancki styl komunikatu

Dodatkowe informacje

W tym rozdziale przedstawiono tylko podstawowe informacje na temat możliwości biblioteki PrimeFaces. Pokaz ilustrujący wszystkie komponenty z tej biblioteki znajdziesz na stronie *http://www.primefaces.org/showcase/ui/home.jsf*. Więcej informacji na temat tej biblioteki zawiera witryna *http://www.primefaces.org*.

Podsumowanie

Ten rozdział zawiera wprowadzenie do biblioteki komponentów JSF PrimeFaces dostępnej w środowisku NetBeans od wersji 7.0. Pokazano tu, w jaki sposób biblioteka PrimeFaces umożliwia łatwe tworzenie eleganckich aplikacji z obsługą Ajaksa. Jednym z ważnych opisanych tu komponentów biblioteki PrimeFaces jest komponent zakładek, który umożliwia wygodne dzielenie stron na zakładki. Dowiedziałeś się też, jak stosować komponent kreatora z biblioteki PrimeFaces, pozwalający na łatwe tworzenie w aplikacjach sieciowych interfejsów opartych na kreatorach.

Skorowidz

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

U

W

Z

Ź

Ż

Java EE 6
Tworzenie aplikacji w NetBeans 7

Java EE to zbiór zaawansowanych technologii, pozwalających stworzyć nowoczesną aplikację o doskonałej architekturze. Jeżeli żądasz najwyższej wydajności, niezawodności, jakości oraz elastyczności, to Java EE jest prawdopodobnie najlepszym wyborem. Jeżeli chcesz się skupić na postawionym zadaniu i nie masz ochoty zmagać się z problemami z konfiguracją czy środowiskiem pracy, wybierz lidera wśród darmowych IDE. NetBeans zapewnia pełne wsparcie dla Javy EE, a praca z nim to czysta przyjemność.

W trakcie lektury tej wyjątkowej książki poznasz najlepsze techniki pracy z tym duetem. Zaznajomisz się z możliwościami Javy EE oraz przekonasz się, jak bardzo NetBeans może ułatwić Ci pracę. Na wstępie dowiesz się, jak zainstalować, dostosować do własnych potrzeb oraz przygotować do pracy środowisko NetBeans. Na kolejnych stronach zdobędziesz wiedzę związaną z serwletami, JSP oraz JSF. Ponadto nauczysz się korzystać z Java Persistence API oraz projektować i uruchamiać usługi sieciowe. Książka ta jest doskonałym źródłem wiedzy dla wszystkich programistów języka Java, chcących wykorzystać potencjał NetBeans w tworzeniu zaawansowanych aplikacji.

Poznaj najlepsze techniki pracy z NetBeans!

Dzięki tej książce:

- poznasz możliwości Javy EE
- dostosujesz NetBeans do własnych preferencji
- wykorzystasz w pełni wsparcie NetBeans dla Javy EE
- zwiększysz swoją produktywność!

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-8936-1

9 788324 689361

Cena: 59,00 zł