Zastosowanie słuchaczy zdarzeń

wg

https://docs.oracle.com/javaee/7/JEETT.pdf

rozdziały 11-12

Technologie internetowe 6

1. Obsługa zdarzeń typu **valueChangeListener**

Rejestracja słuchaczy zdarzeń typu valueChangeListener w komponentach

1. Pierwszy sposób

Słuchacz zdarzeń może być metodą obiektów typu Managed Bean – wtedy referencja tej metody jest przypisana do atrybutu **valueChangeListener komponentu** (**wykład 3, Przykład 4, slajdy 36-39;**
http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/ti_/TINT_3.pdf)

2. Drugi sposób

Słuchacz zdarzeń może być instancją zdefiniowanej klasy – wtedy znaczniki **f:valueChangeListener** (**wykład 3, Przykład 5: slajdy 40-43; Przykład 6: slajdy 44-47**

http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/ti_/TINT_3.pdf)

Rejestracja słuchaczy zdarzeń typu Value-Change w komponentach

f:valueChangeListener

Drugi sposób

Atrybuty:

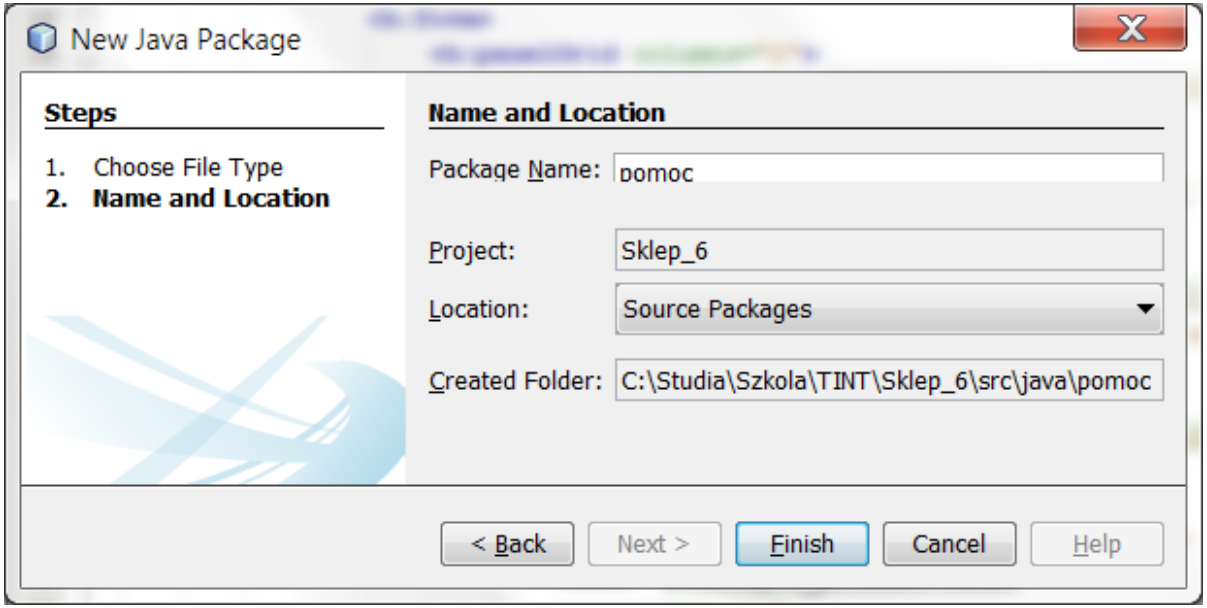
type - **wskazanie na nazwę pakietową klasy**, zawierającą definicję słuchacza zdarzeń typu **ValueChangeListener**. Można użyć literał lub wyrażenie, wskazujące np. na klasę typu **javax.faces.event.ValueChangeEvent.**

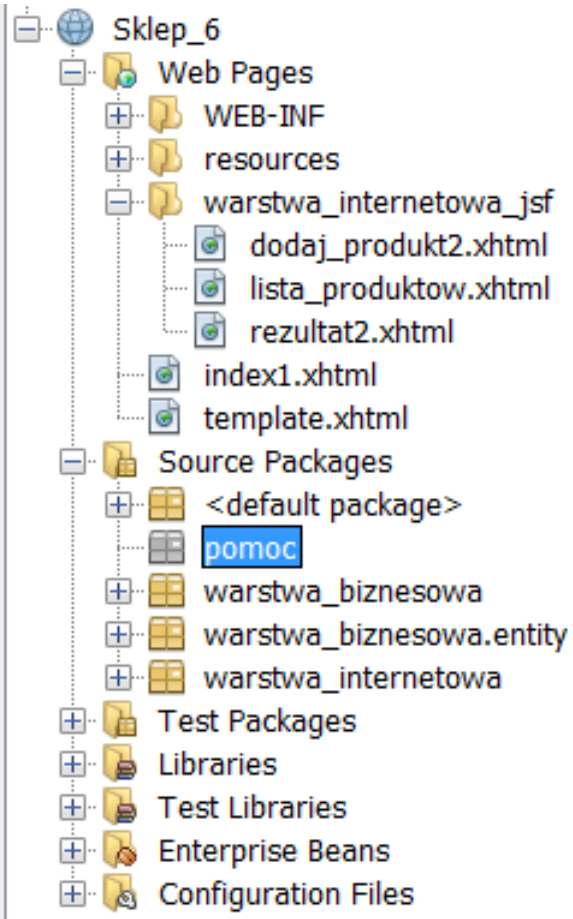
W podanym przykładzie jest: **warstwa_internetowa.nameChanged**

binding - **wskazanie na obiekt**, który implementuje słuchacza zdarzeń typu **ValueChangeListener**. Można użyć jedynie wyrażenia, które wskazuje na właściwość obiektu typu **Managed Bean**, która zwraca referencję do obiektu implementującego słuchacza zdarzeń typu **ValueChangeListener** (podobnie jak w przypadku konwerterów)

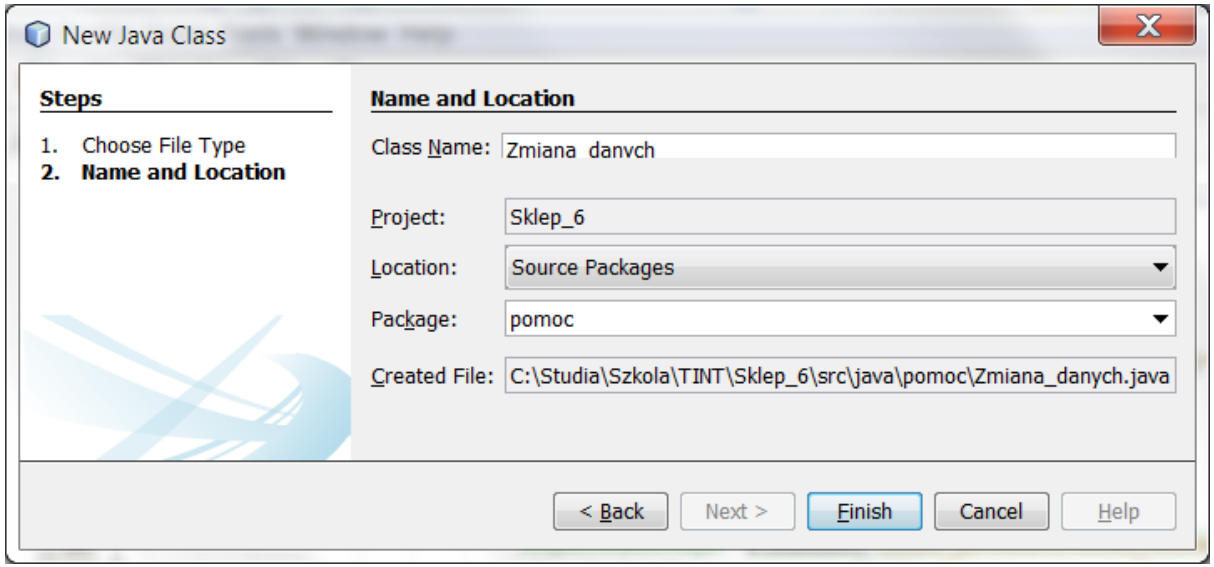
Przykład 1 – rejestracja obsługi valueChangeListener za pomocą atrybutu type

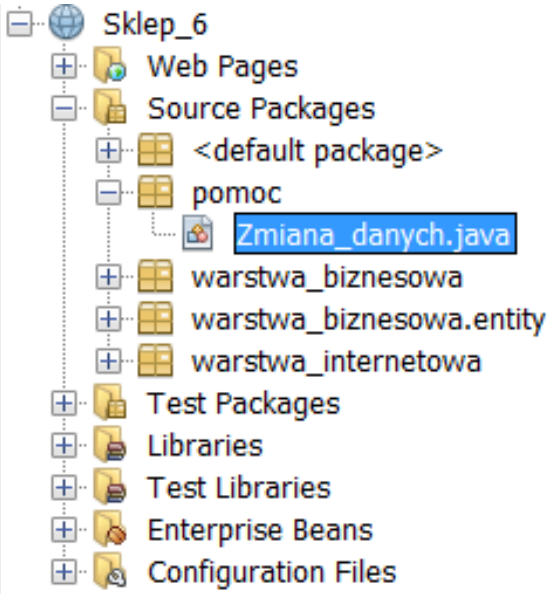
1. Należy wykonać pakiet o nazwie **pomoc** – po kliknięciu prawym klawiszem myszy na nazwę projektu należy wybrać kolejno pozycje **New/Other/Java/Java Package** i po kliknięciu na klawisz **Next** w polu **Package Name** wpisać nazwę nowego pakietu: **pomoc.**





2. W pakiecie **pomoc** należy utworzyć nową klasę **Zmiana_danych**: po kliknięciu na pakiet **pomoc** prawym klawiszem myszy należy wybrać kolejno pozycje: **New/Other/Java/Java Class** i po kliknięciu na klawisz **Next** w polu **Class Name** wpisać nazwę nowej klasy: **Zmiana_danych.**





3 Wykonanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener.** Możliwa kontrola zdarzeń w wielu komponentach typu UI

```
package pomoc;
import javax.faces.application.FacesMessage;
import javax.faces.context.FacesContext;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ValueChangeEvent;
import javax.faces.event.ValueChangeListener;
public class Zmiana_danych implements ValueChangeListener{
    int licznik;
    String klucz;
    public Zmiana_danych(String klucz_)
          {     klucz = klucz_;    }
    public Zmiana_danych()
         {     klucz="dane";    }
```

```
@Override
  public void processValueChange(ValueChangeEvent event) throws
                                                    AbortProcessingException {
      String nazwa;
      FacesContext context = FacesContext.getCurrentInstance();
      String clientId = event.getComponent().getClientId();
      nazwa = "" + event.getNewValue();
      if (!nazwa.equals("")) {
         if (context.getExternalContext().getSessionMap().containsKey(klucz))
            licznik = (int) context.getExternalContext().getSessionMap().get(klucz);
         licznik++;
         FacesMessage message =
               new FacesMessage("Stan licznika zmian " + klucz + ": " + licznik);
         context.getExternalContext().getSessionMap().put(klucz, licznik);
         context.addMessage(clientId, message);   }
   }
}
```

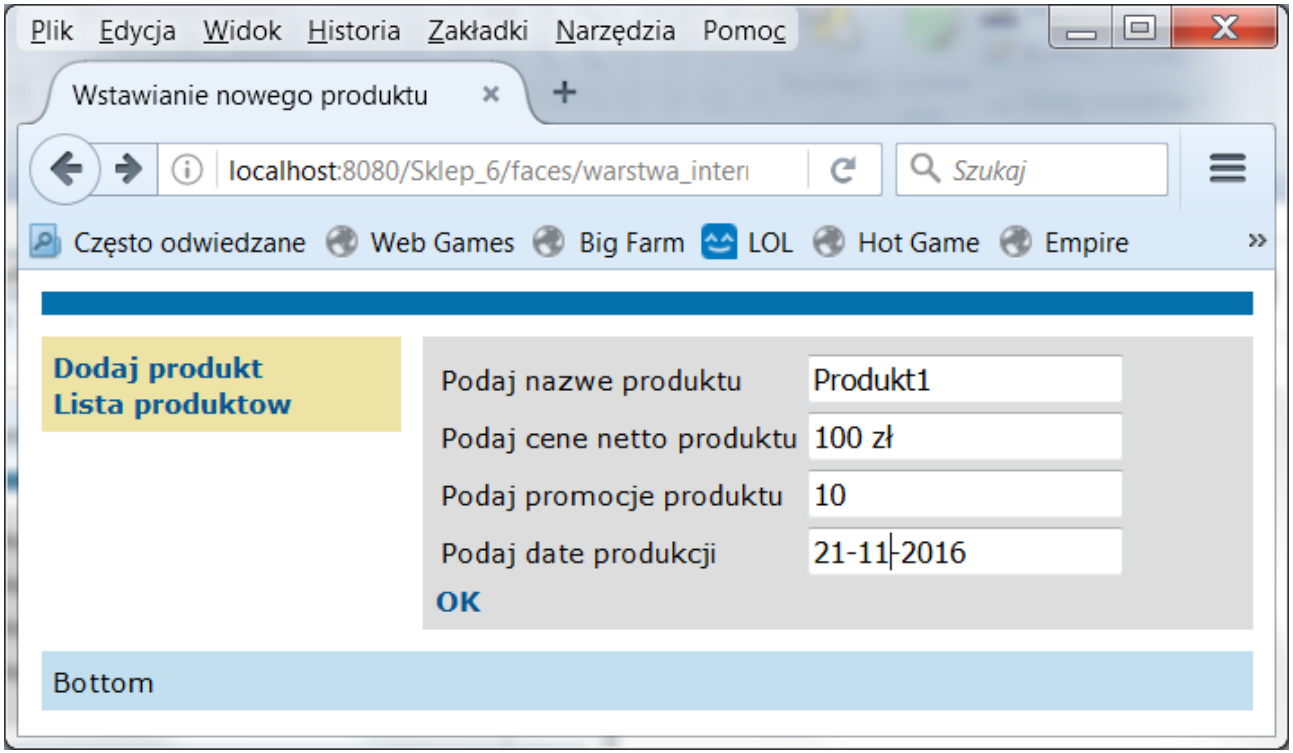
Jeśli wystąpią błędy konwersji (int), należy zastosować konwersję (Integer)

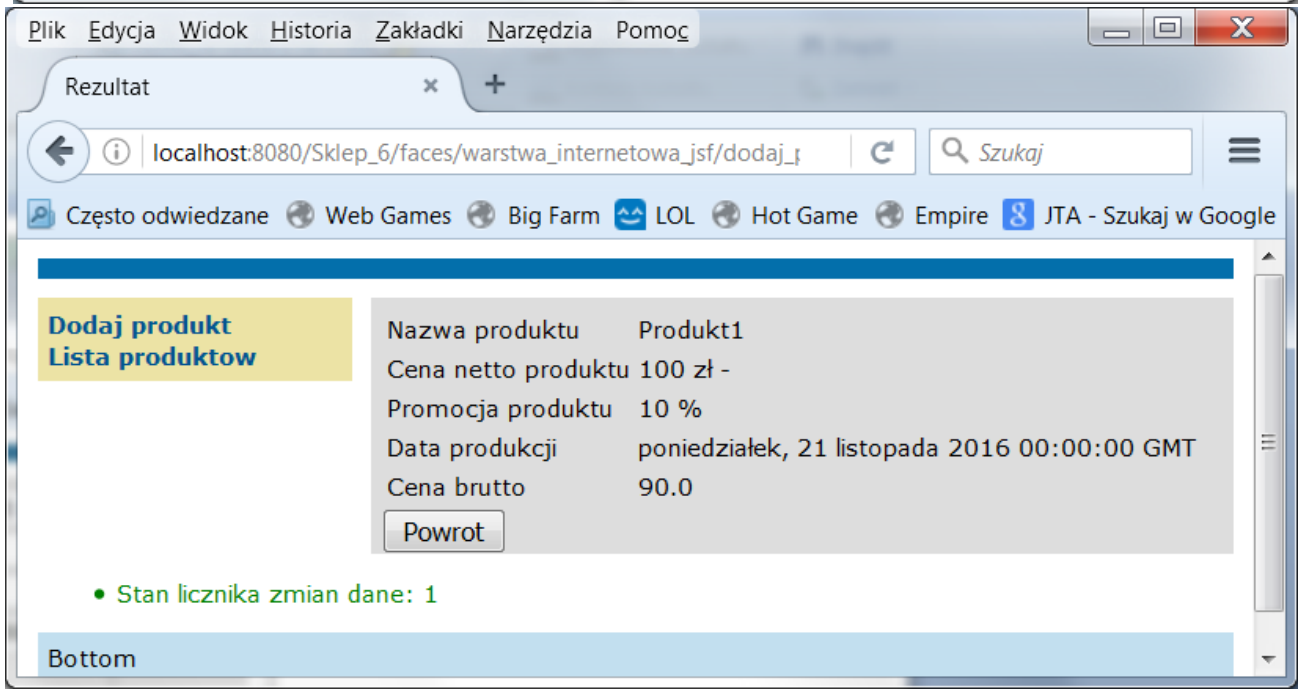
Klucz pozwala przechowywać informacje w kolekcji implementującej interfejs Map, pochodzące z różnych komponentów UI służących do wprowadzania danych.

4. Zastosowanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener – (odwołanie do ścieżki pakietowej za pomocą atrybutu type).**

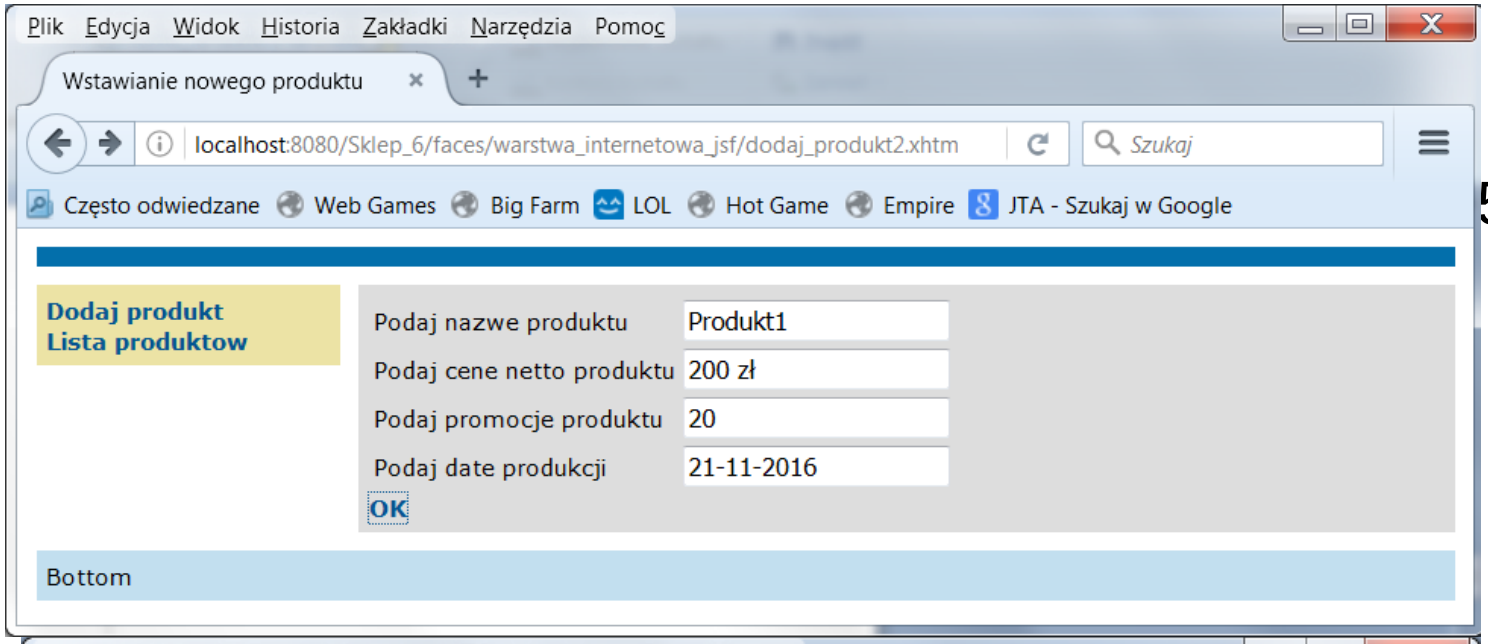
```
<h:outputLabel value="#{bundle['dodaj_produkt2.nazwa']}"
  for="nazwa" />
<h:inputText
      id="nazwa"
      title="#{bundle['dodaj_produkt2.nazwa1']}"
      value="#{managed_produkt.nazwa}"
      required="true"
      requiredMessage="#{bundle['dodaj_produkt2.blad_nazwa']}" >
        <f:valueChangeListener type="pomoc.Zmiana_danych"/>
</h:inputText>
```

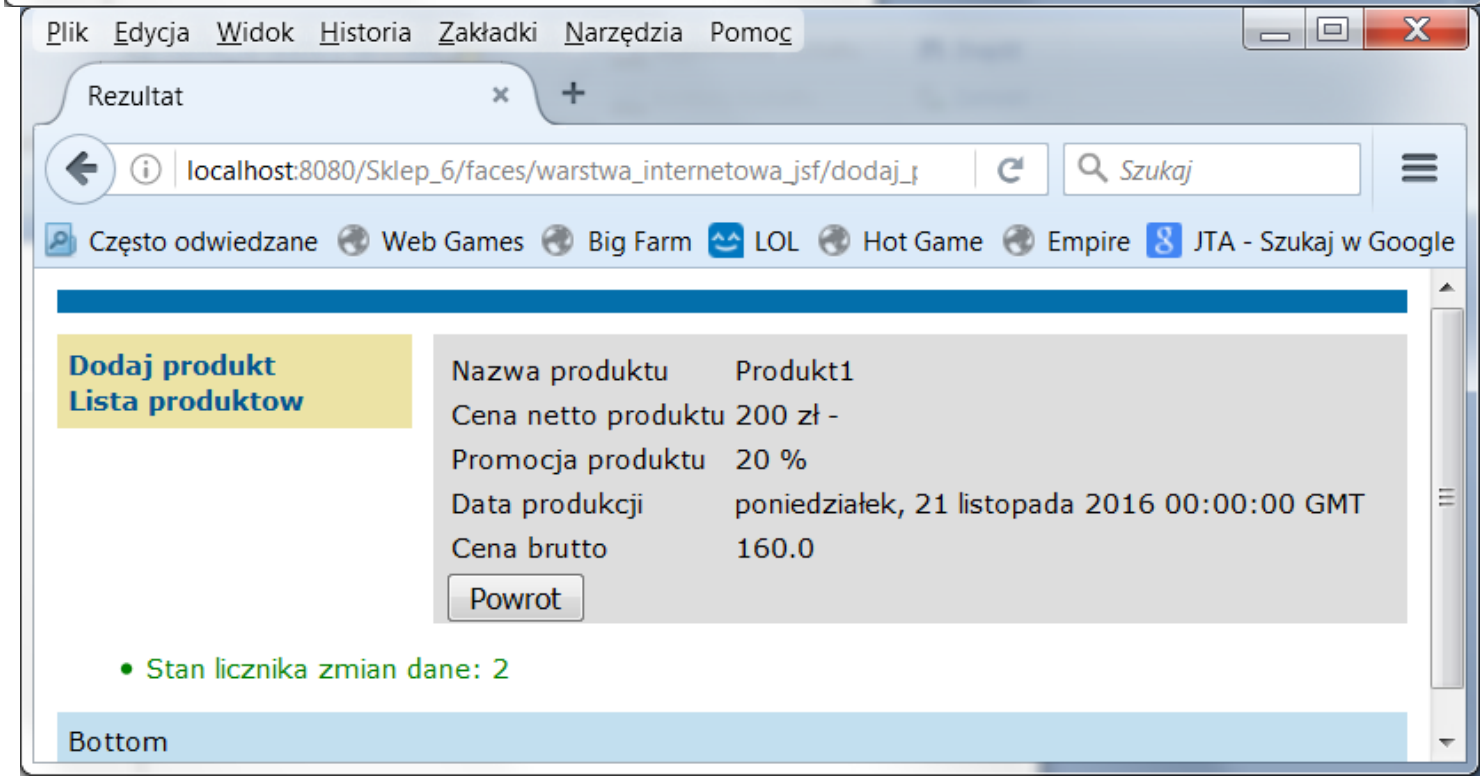
5 Prezentacja wyniku





5. Prezentacja wyniku (cd)





Przykład 2. – rejestracja obsługi valueChangeListener za pomocą atrybutu binding

1. Zastosowanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener**

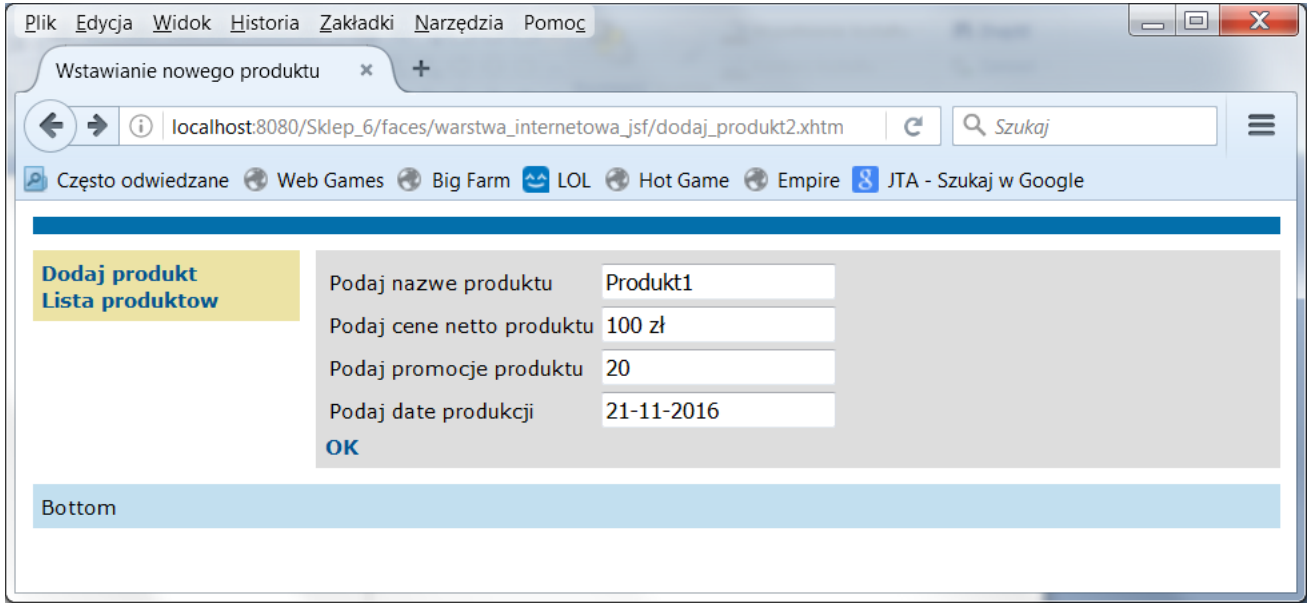
```
<h:outputLabel value="#{bundle['dodaj_produkt2.nazwa']}"  for="nazwa" />
<h:inputText
    id="nazwa”    title="#{bundle['dodaj_produkt2.nazwa1']}"
    value="#{managed_produkt.nazwa}"
    required="true”    requiredMessage="#{bundle['dodaj_produkt2.blad_nazwa']}" >
            <f:valueChangeListener binding="#{managed_produkt.zmiana1}"/>
</h:inputText>
<h:outputLabel value="#{bundle['dodaj_produkt2.cena']}" for="cena" />
<h:inputText
     id="cena"          title="#{bundle['dodaj_produkt2.cena1']}"
     value="#{managed_produkt.cena}"
     required="true"     requiredMessage="#{bundle['dodaj_produkt2.blad_cena']}"
     converter="#{managed_produkt.number_convert}"
     converterMessage="Blad! Poprawny format: 0,0 zł lub 0 zł" >
            <f:valueChangeListener binding="#{managed_produkt.zmiana2}"/>
</h:inputText>
```

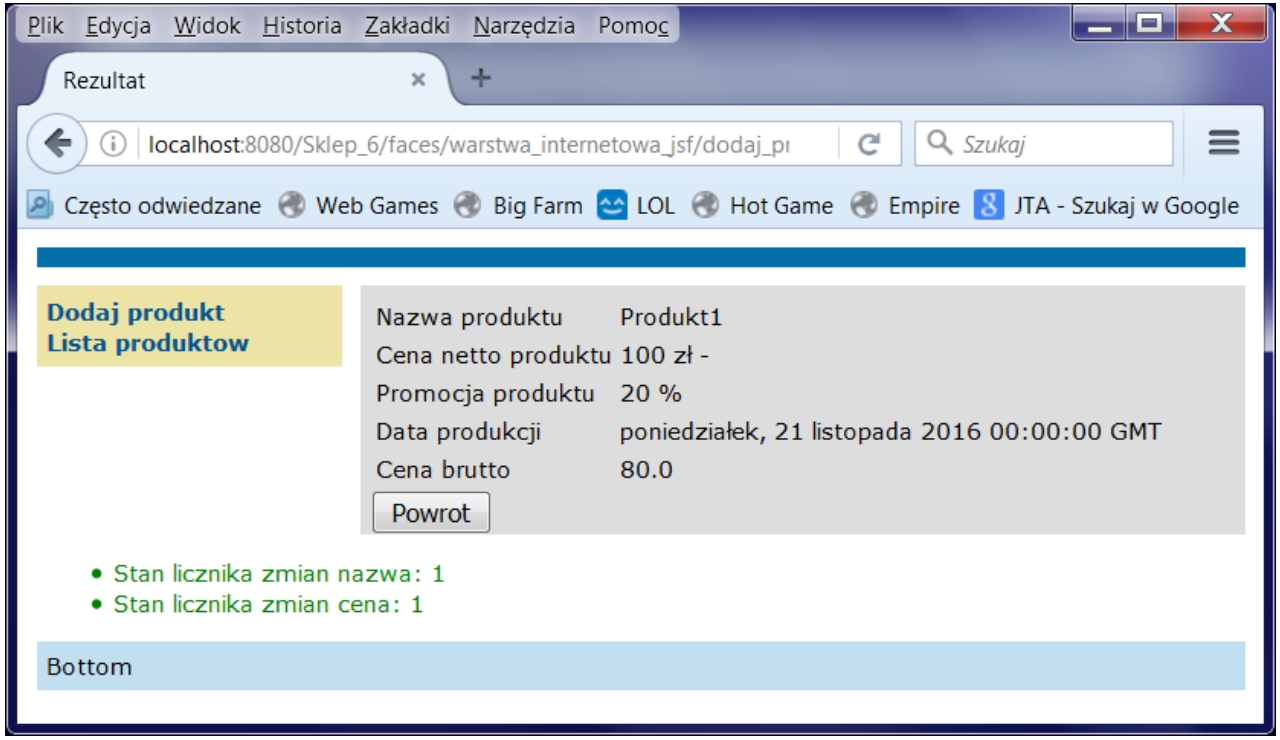
2. Zastosowanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener.** Odwołanie do obiektu za pomocą atrybutu **binding** wymaga utworzenie obiektu typu **Zmiana_danych**. Ponieważ obecnie obsługą zdarzeń objęto pola **nazwa** i **cena** na stronie **dodaj_produkt2.xhtml, wykonano dwa niezależnie obiekty typu Zmiana_danych**

```
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt {
    @EJB
    private Fasada_warstwy_biznesowej fasada;
    public Managed_produkt() { }
    private String nazwa;
    private float cena;
    private int promocja;
    private String cena_brutto;
    private DataModel items;
    private int stan = 1;
    private Date data_produkcji;
```

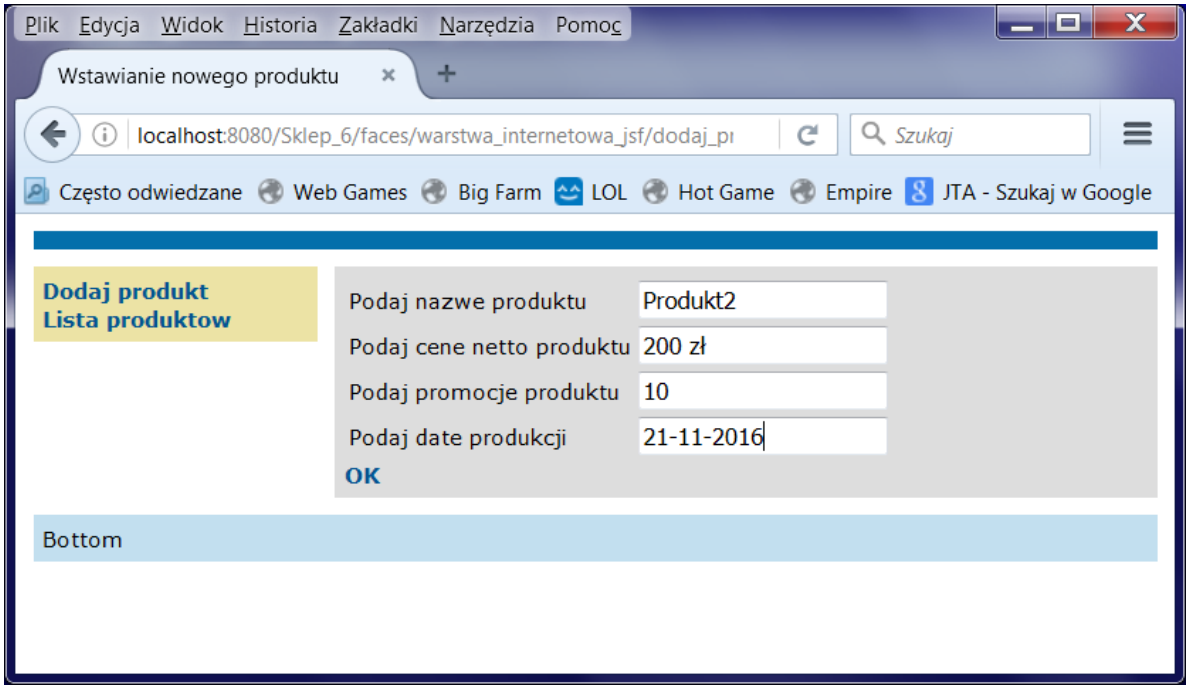
```
private Zmiana_danych zmiana1= new Zmiana_danych("nazwa");

private Zmiana_danych zmiana2= new Zmiana_danych("cena");

public Zmiana_danych getZmiana1()
    {  return zmiana1;   }
public void setZmiana1(Zmiana_danych zmiana)
   {  this.zmiana1 = zmiana;    }


public Zmiana_danych getZmiana2()
  {  return zmiana2;    }
public void setZmiana2(Zmiana_danych zmiana2)
 {  this.zmiana2 = zmiana2;    }
```

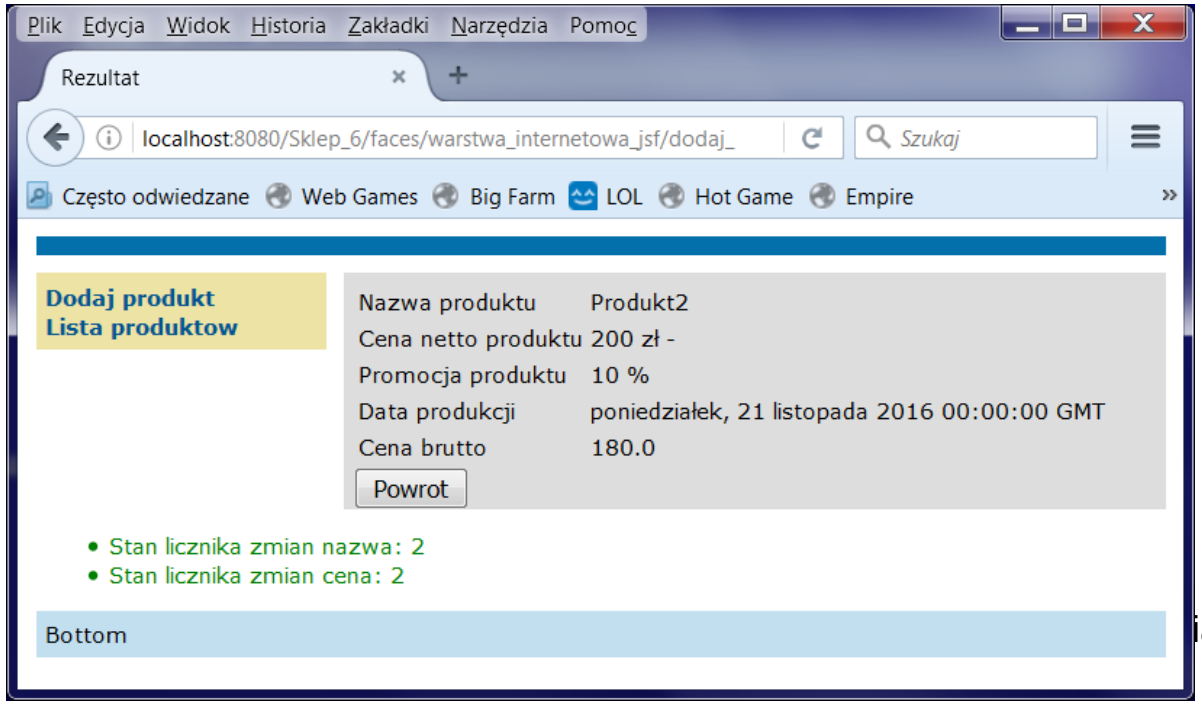
3 Prezentacja wyniku





3 Prezentacja wyniku (cd)





2. Obsługa zdarzeń typu **actionListener**

Rejestracja słuchaczy zdarzeń typu actionListener w komponentach

1. Pierwszy sposób

Słuchacz zdarzeń może być metodą obiektów typu Managed Bean – wtedy referencja tej metody jest przypisana do atrybutu **actionListener komponentu**

1. Drugi sposób

Słuchacz zdarzeń może być instancją zdefiniowanej klasy – wtedy znaczniki **f:actionListener** są powiązane z takim słuchaczem i zagnieżdżone w znaczniku komponentu, który generuje zdarzenie

Obsługa zdarzeń typu **ActionListener** – zmiana definicji znacznika **<h:commandLink** na stronie **dodaj_produkt2.xhtml** oraz modyfikacja metod obsługujących ten znacznik w klasie typu **Managed_produkt**.

Stan przed zmianą.

Strona dodaj_produkt2.xhtml

```
<h:commandLink
    action="#{managed_produkt.dodaj_produkt}"
    value="OK" />
```

Klasa typu Managed_produkt:

```
public String dodaj_produkt() {
        String[] dane = {nazwa, "" + cena, "" + promocja};
        fasada.utworz_produkt(dane, data_produkcji);
       dane_produktu();
       return "rezultat2";
    }
    public void dane_produktu() {
        stan = 1;
        String[] dane = fasada.dane_produktu();
        if (dane == null) {
            stan = 0;
        } else {
            nazwa = dane[0];
            cena = Float.parseFloat(dane[1]);
            promocja = Integer.parseInt(dane[2]);
            cena_brutto = dane[3];
            data_produkcji.setTime(Long.parseLong(dane[4]));
        }
    }
```

Przykład 1 (**pierwszy sposób**) **przy wprowadzaniu danych** – obsługa zdarzenia typu **actionListener** podczas kliknięcia na komponent typu **h:commandLink**

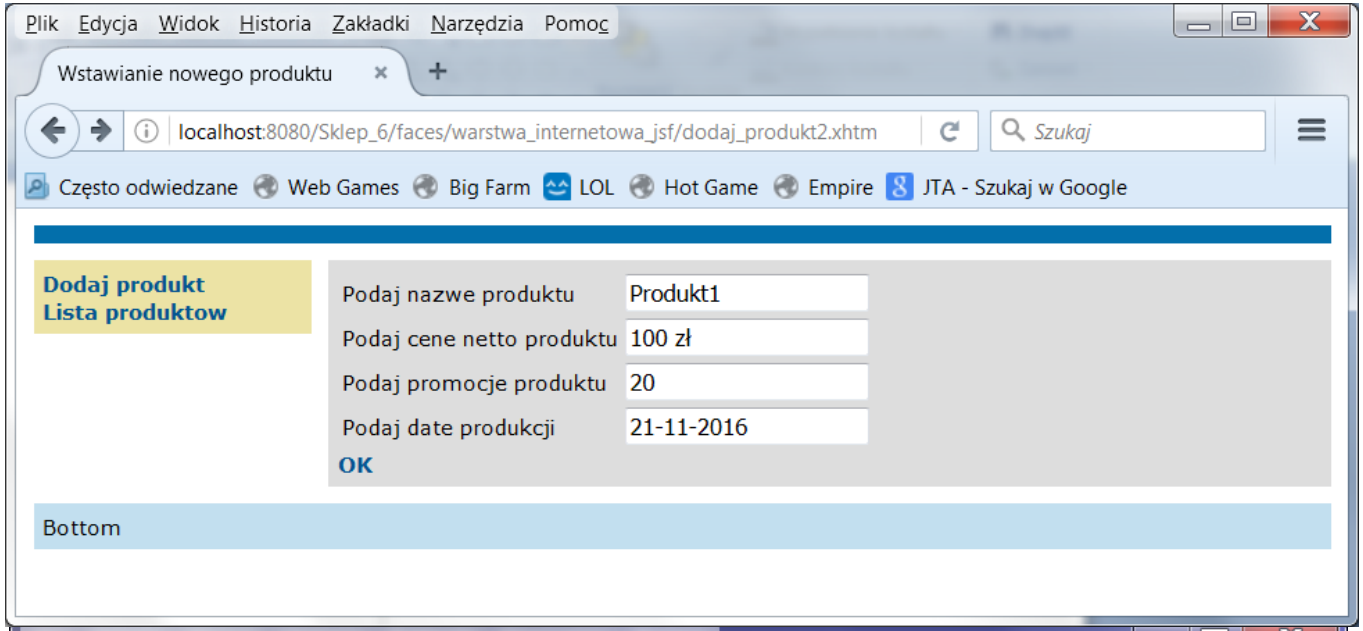
Fragment strony **dodaj_produkt2.xhtml:**

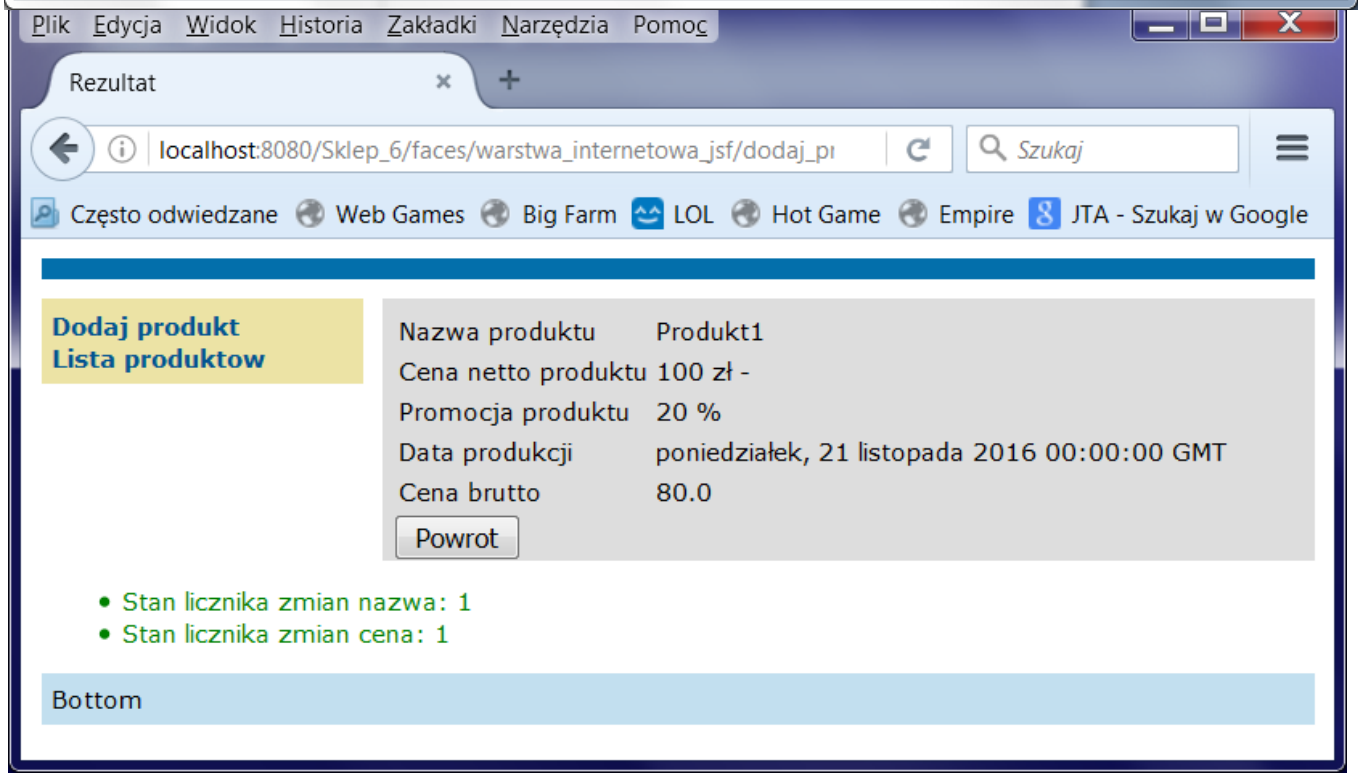
```
<h:commandLink  action="rezultat2" value="OK"
    actionListener="#{managed_produkt.dodaj_produkt}"/>
```

W klasie typu **Managed Bean** dokonano zmiany metody **dodaj_produkt**

```
public void dodaj_produkt() {
      String[] dane = {"" + nazwa, "" + cena, "" + promocja};
      fasada.utworz_produkt(dane, data_produkcji);
      dane_produktu();
      // return "rezultat2";
}
```

3.3. Prezentacja wyniku





Przykład 2 (**pierwszy sposób**) **przy wprowadzaniu danych** – obsługa zdarzenia typu **actionListener** podczas kliknięcia na komponent typu **h:commandLink**

Obsługa zdarzeń typu **ActionListener** – zmiana definicji znacznika **<h:commandLink** na stronie **dodaj_produkt2.xhtml** oraz modyfikacja metod obsługujących ten znacznik w klasie typu **Managed_produkt**. Stan po zmianie.

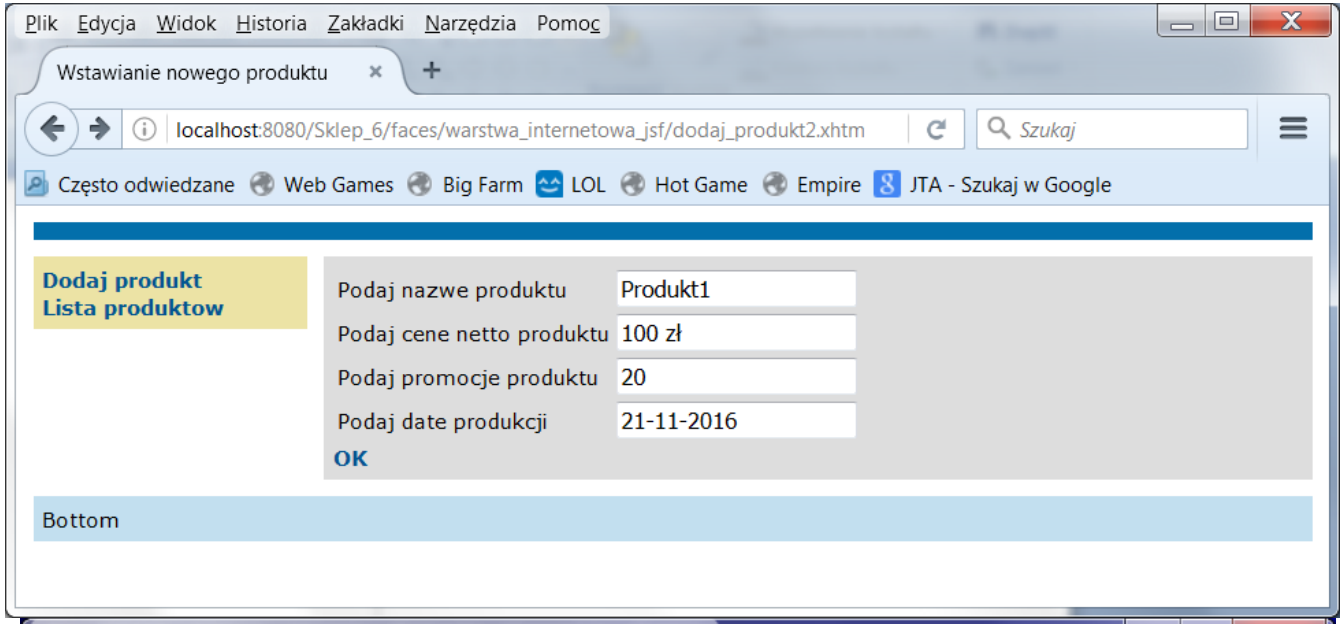
Strona dodaj_produkt2.xhtml:

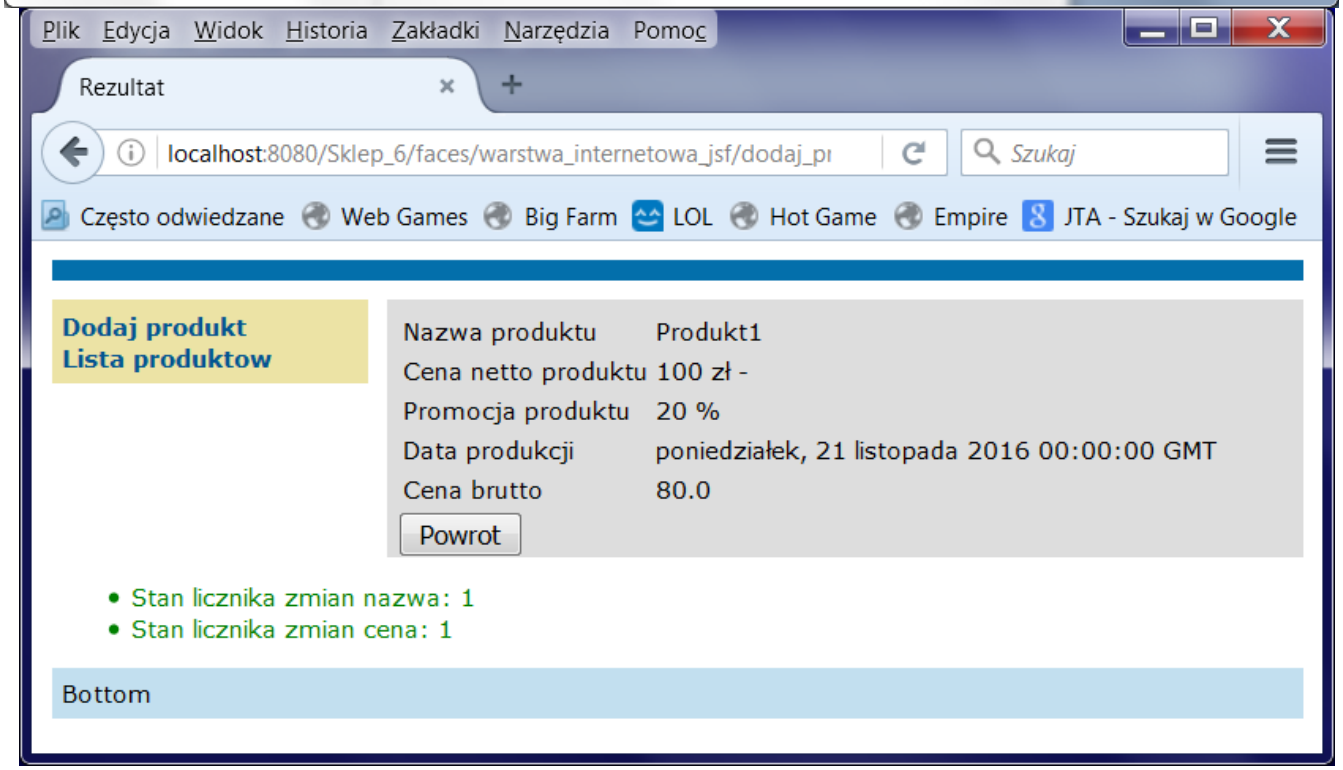
```
<h:commandLink  action="#{managed_produkt.dane_produktu}"
    value="OK"
    actionListener="#{managed_produkt.dodaj_produkt}" />
```

Klasa typu Managed_produkt

```
public void dodaj_produkt() {
        String[] dane = {nazwa, "" + cena, "" + promocja};
        fasada.utworz_produkt(dane, data_produkcji);
}
    public String dane_produktu() {
        stan = 1;
        String[] dane = fasada.dane_produktu();
        if (dane == null) {
            stan = 0;
        } else {
            nazwa = dane[0];
            cena = Float.parseFloat(dane[1]);
            promocja = Integer.parseInt(dane[2]);
            cena_brutto = dane[3];
            data_produkcji.setTime(Long.parseLong(dane[4]));
        }
        return "rezultat2";
    }
```

Prezentacja wyniku





Rejestracja słuchaczy zdarzeń typu **Action Listener** w komponentach (**drugi sposób**)

f:actionListener

Atrybuty:

type - **wskazanie na nazwę pakietową klasy**, zawierającą definicję słuchacza zdarzeń typu **ActionListener**. Można użyć literał lub wyrażenie, wskazujące np. na klasę typu **javax.faces.event.ActionListener**

binding - **wskazanie na obiekt**, który implementuje słuchacza zdarzeń typu **ActionListene**r. Można użyć jedynie wyrażenia, które wskazuje na właściwość obiektu typu **Managed Bean**, która zwraca referencję do obiektu implementującego słuchacza zdarzeń typu **ActionListener** (podobnie jak w przypadku konwerterów)

Przykład 3 (**drugi sposób**) **przy wprowadzaniu danych** – obsługa zdarzenia typu **actionListener** podczas kliknięcia na komponent typu **h:commandLink** (na stronie **dodaj_produkt2**)

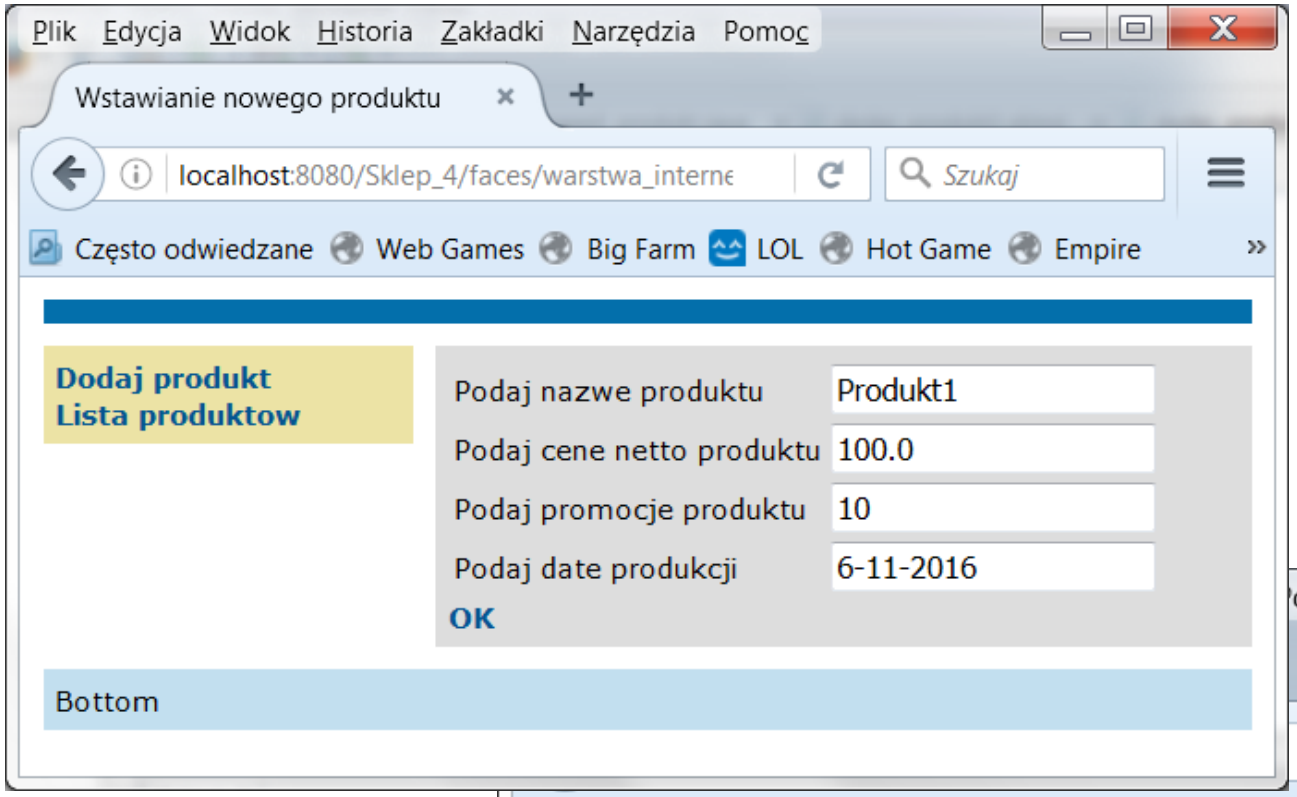
Fragment strony **dodaj_produkt2.xhtml:**

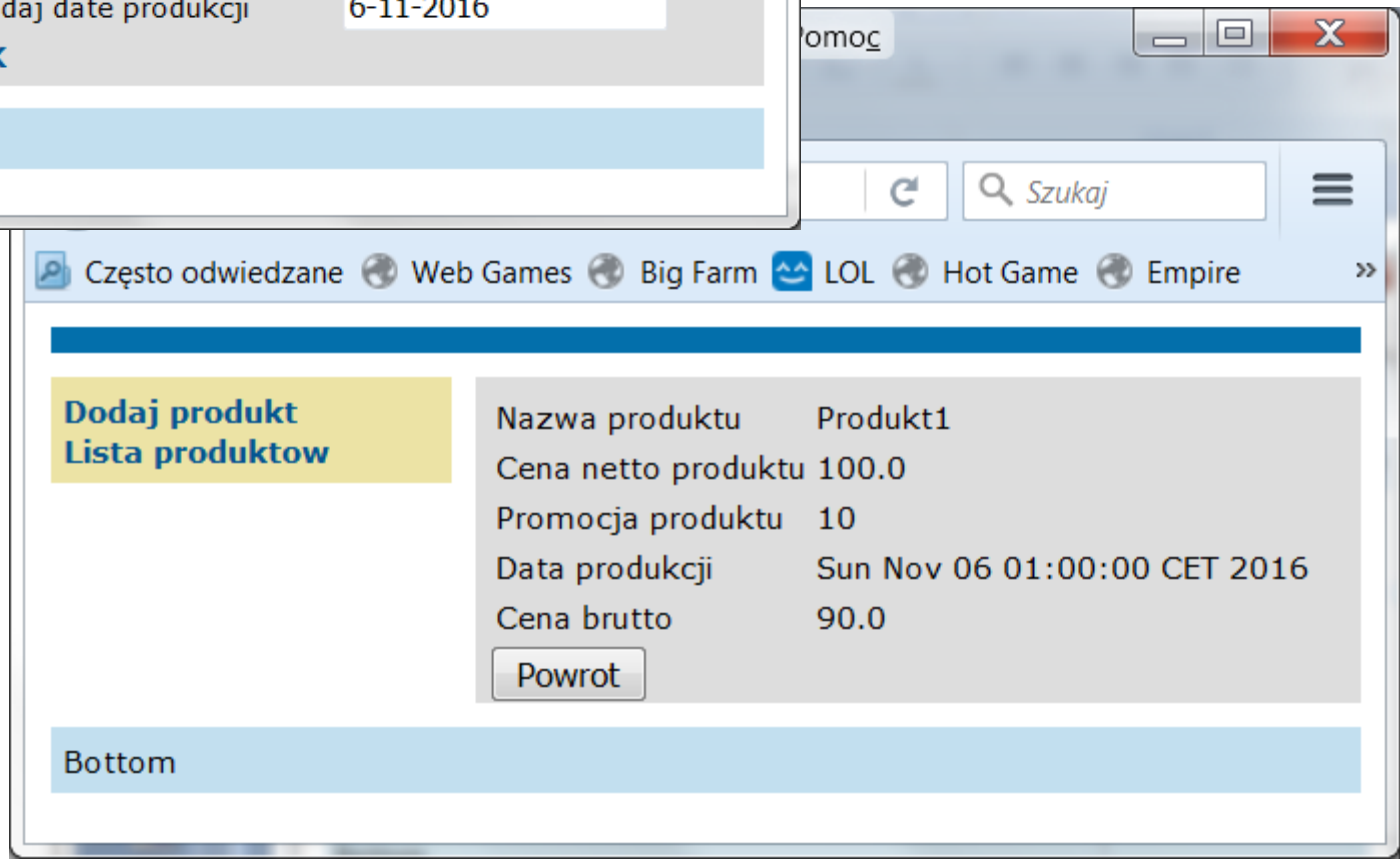
```
<h:commandLink  action="rezultat2" value="OK" >
    <f:actionListener  binding="#{managed_produkt}"/>
</h:commandLink>
```

Przykład 3 (cd) - Klasa typu **Managed Bean** implementuje interfejs **ActionListene**r:

```
package warstwa_internetowa;
import java.util.Date;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;
import warstwa_biznesowa.Fasada_warstwy_biznesowej;
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt implements ActionListener { //
.....
public void processAction(ActionEvent event)  throws
                                        AbortProcessingException {
 //public void dodaj_produkt() {
    String[] dane = {"" + nazwa, "" + cena, "" + promocja};
    fasada.utworz_produkt(dane, data_produkcji);
    dane_produktu();
   // return "rezultat2";
  }
```

Przykład 3 (cd)
Wynik działania w przypadku zastosowania obu sposobów rejestrowania obsługi zdarzeń typu **actionListener**





Przykład 4 (**drugi sposób**) **przy wprowadzaniu danych** – obsługa zdarzenia typu **actionListener** podczas kliknięcia na komponent typu **h:commandLink** (na stronie **dodaj_produkt2**)

Fragment strony **dodaj_produkt2.xhtml:**

```
<h:commandLink  action="rezultat2" value="OK" >
    <f:actionListener  binding="#{managed_produkt}"/>
</h:commandLink>
```

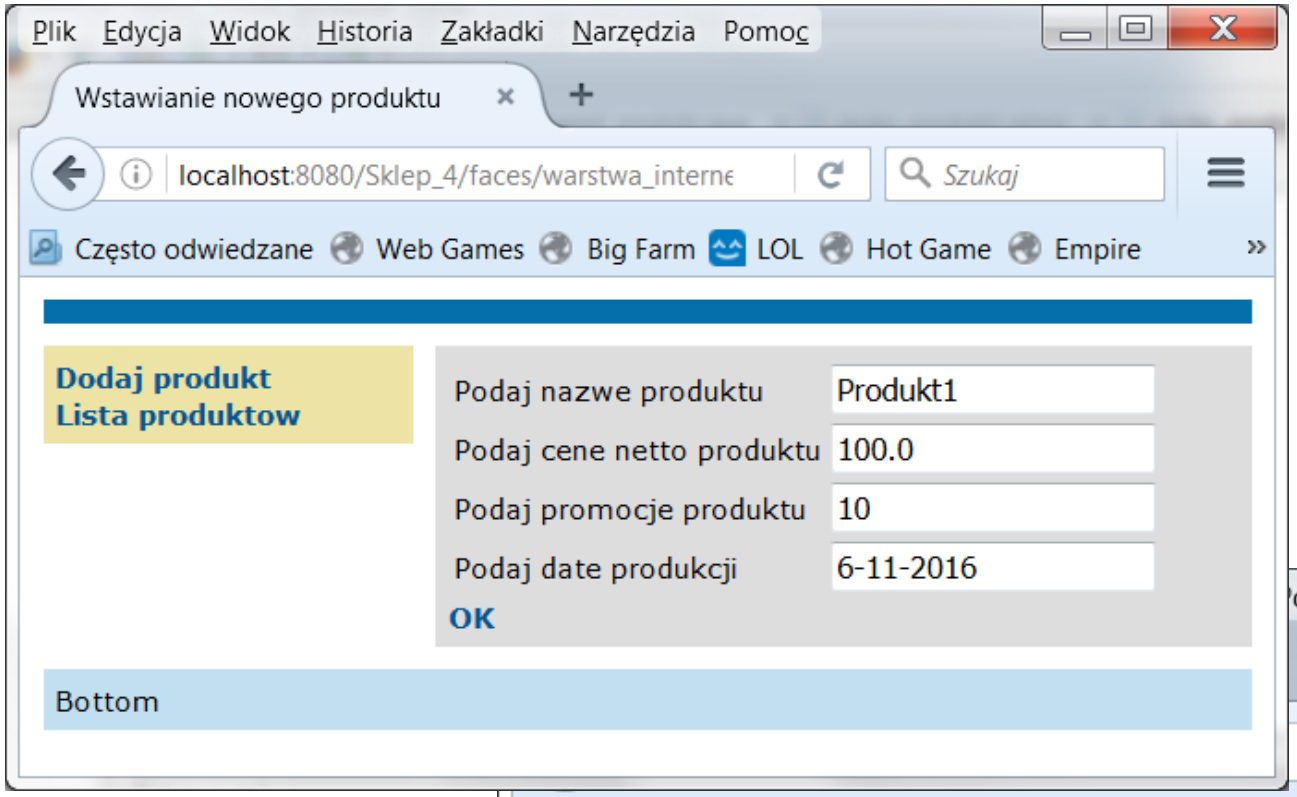
Przykład 4 (cd) - Klasa typu **Managed Bean** implementuje interfejs **ActionListene**r:

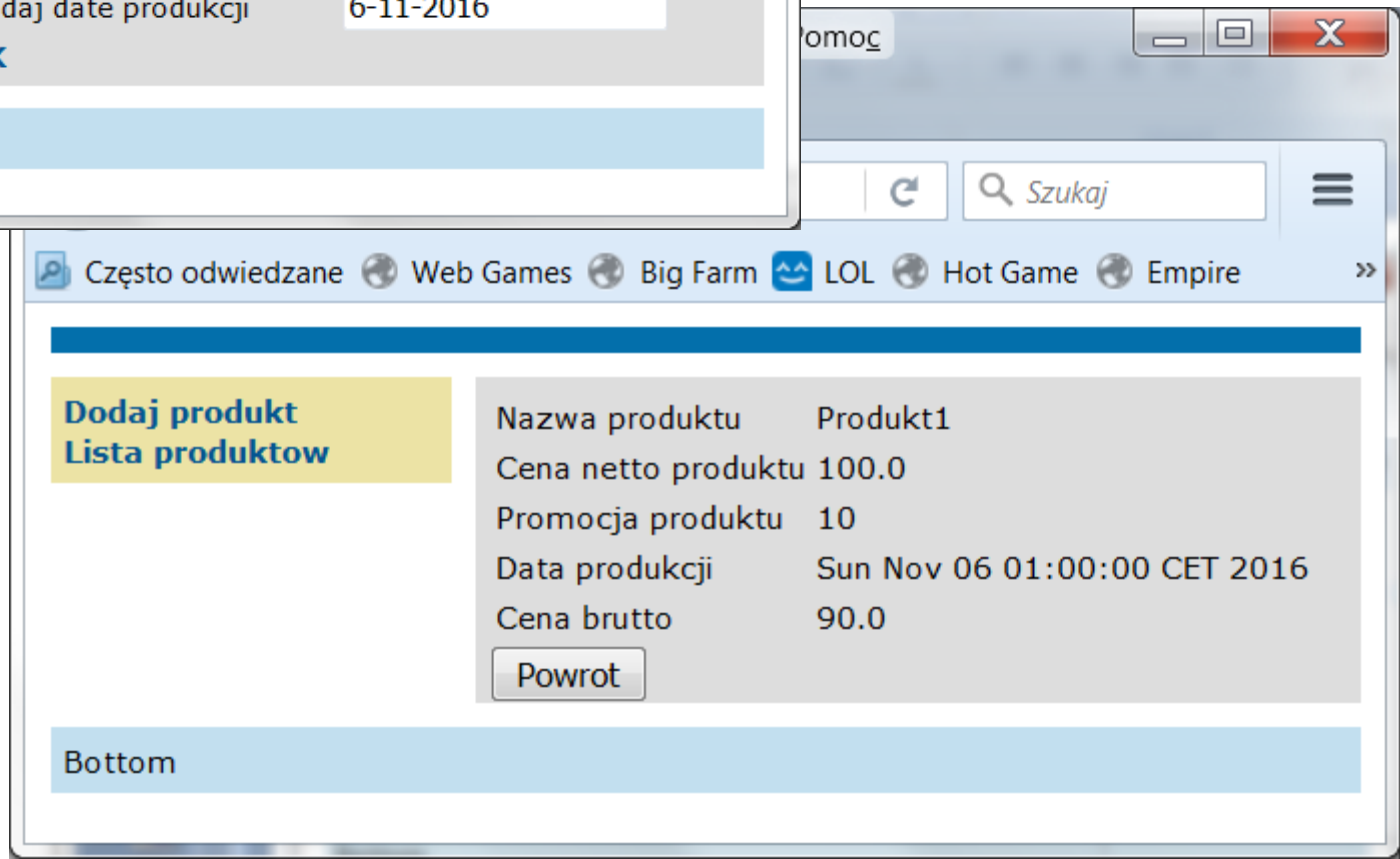
```
package warstwa_internetowa;
import java.util.Date;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;
import warstwa_biznesowa.Fasada_warstwy_biznesowej;
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt implements ActionListener {
//.....
```

```
public void processAction(ActionEvent event)  throws
                                        AbortProcessingException {
    dodaj_produkt();
}
```

```
 public void dodaj_produkt() {
        String[] dane = {nazwa, "" + cena, "" + promocja};
        fasada.utworz_produkt(dane, data_produkcji);
        dane_produktu();
}
    public void dane_produktu() {
        stan = 1;
        String[] dane = fasada.dane_produktu();
        if (dane == null) {
            stan = 0;
        } else {
            nazwa = dane[0];
            cena = Float.parseFloat(dane[1]);
            promocja = Integer.parseInt(dane[2]);
            cena_brutto = dane[3];
            data_produkcji.setTime(Long.parseLong(dane[4]));
        }
}
```

Przykład 4 (cd)
Wynik działania w przypadku zastosowania obu sposobów rejestrowania obsługi zdarzeń typu **actionListener**





Przykład 5 (**drugi sposób**) **przy wprowadzaniu danych** – obsługa zdarzenia typu **actionListener** podczas kliknięcia na komponent typu **h:commandLink** (na stronie **dodaj_produkt2**)

Fragment strony **dodaj_produkt2.xhtml:**

```
<h:commandLink
action="#{managed_produkt.dane_produktu}"/> value="OK" >
    <f:actionListener  binding="#{managed_produkt}"/>
</h:commandLink>
```

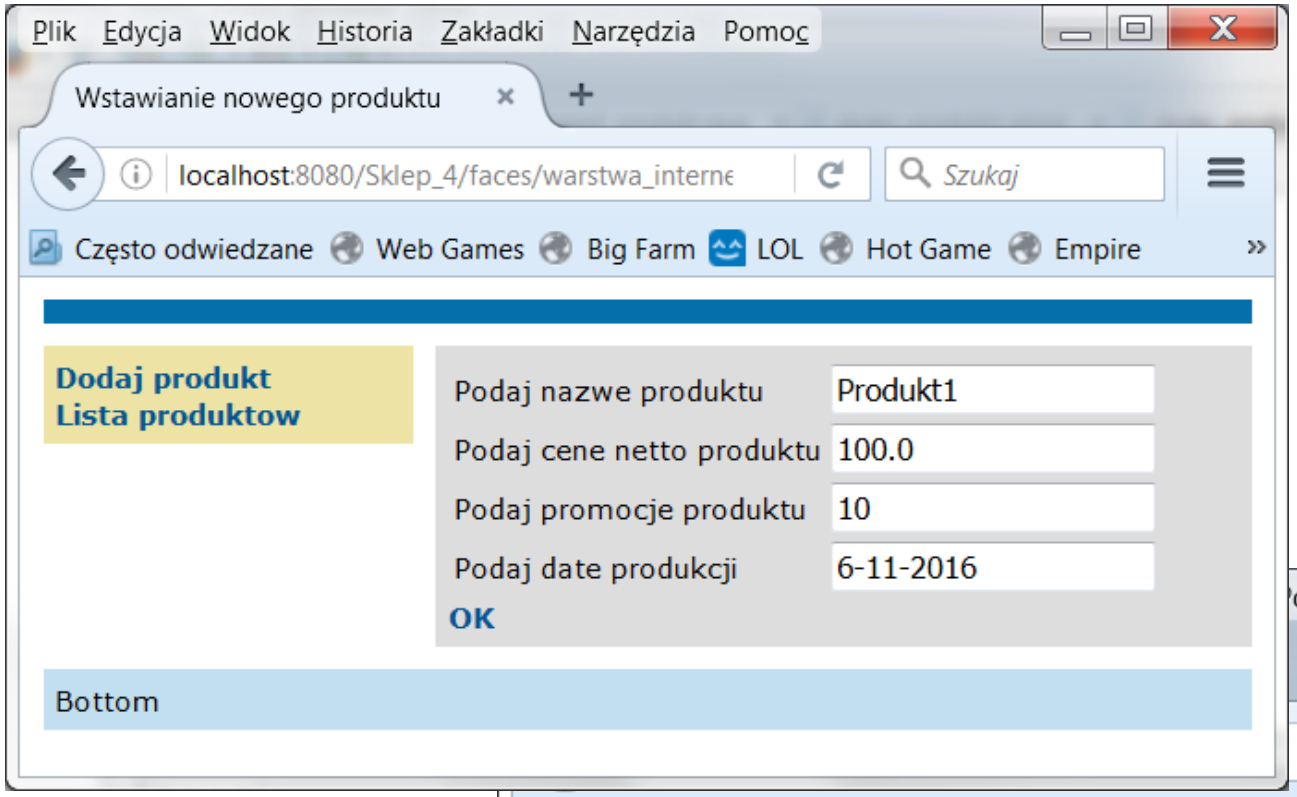
Przykład 5 (cd) - Klasa typu **Managed Bean** implementuje interfejs **ActionListene**r:

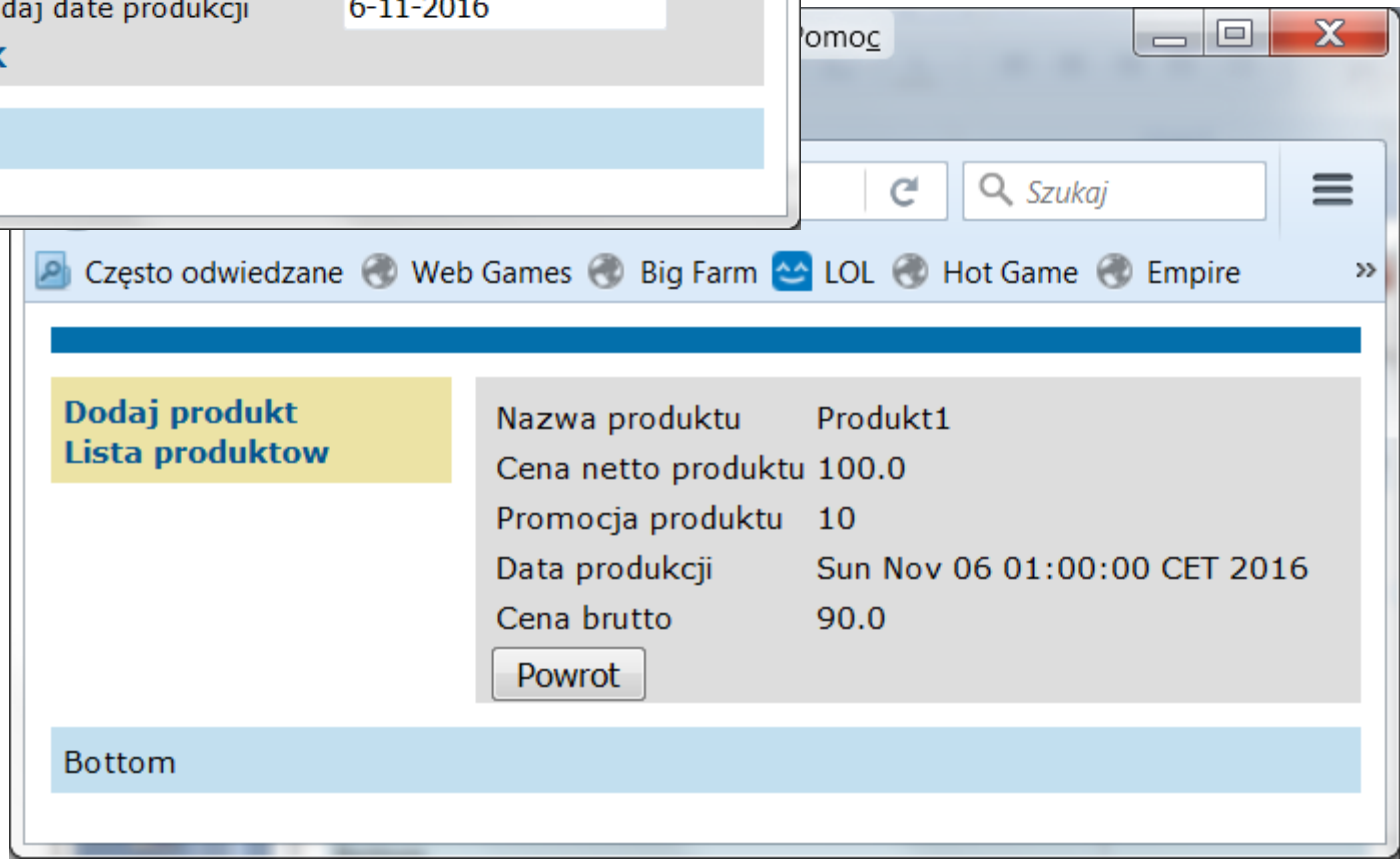
```
package warstwa_internetowa;
import java.util.Date;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;
import javax.faces.model.DataModel;
import javax.faces.model.ListDataModel;
import warstwa_biznesowa.Fasada_warstwy_biznesowej;
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt implements ActionListener {
//.....
```

```
public void processAction(ActionEvent event)  throws
                                        AbortProcessingException {
    dodaj_produkt();
}
```

```
 public void dodaj_produkt() {
        String[] dane = {nazwa, "" + cena, "" + promocja};
        fasada.utworz_produkt(dane, data_produkcji);
}
    public String dane_produktu() {
        stan = 1;
        String[] dane = fasada.dane_produktu();
        if (dane == null) {
            stan = 0;
        } else {
            nazwa = dane[0];
            cena = Float.parseFloat(dane[1]);
            promocja = Integer.parseInt(dane[2]);
            cena_brutto = dane[3];
            data_produkcji.setTime(Long.parseLong(dane[4]));
        }
        return "rezultat2";
}
```

Przykład 5 (cd)
Wynik działania w przypadku zastosowania obu sposobów rejestrowania obsługi zdarzeń typu **actionListener**



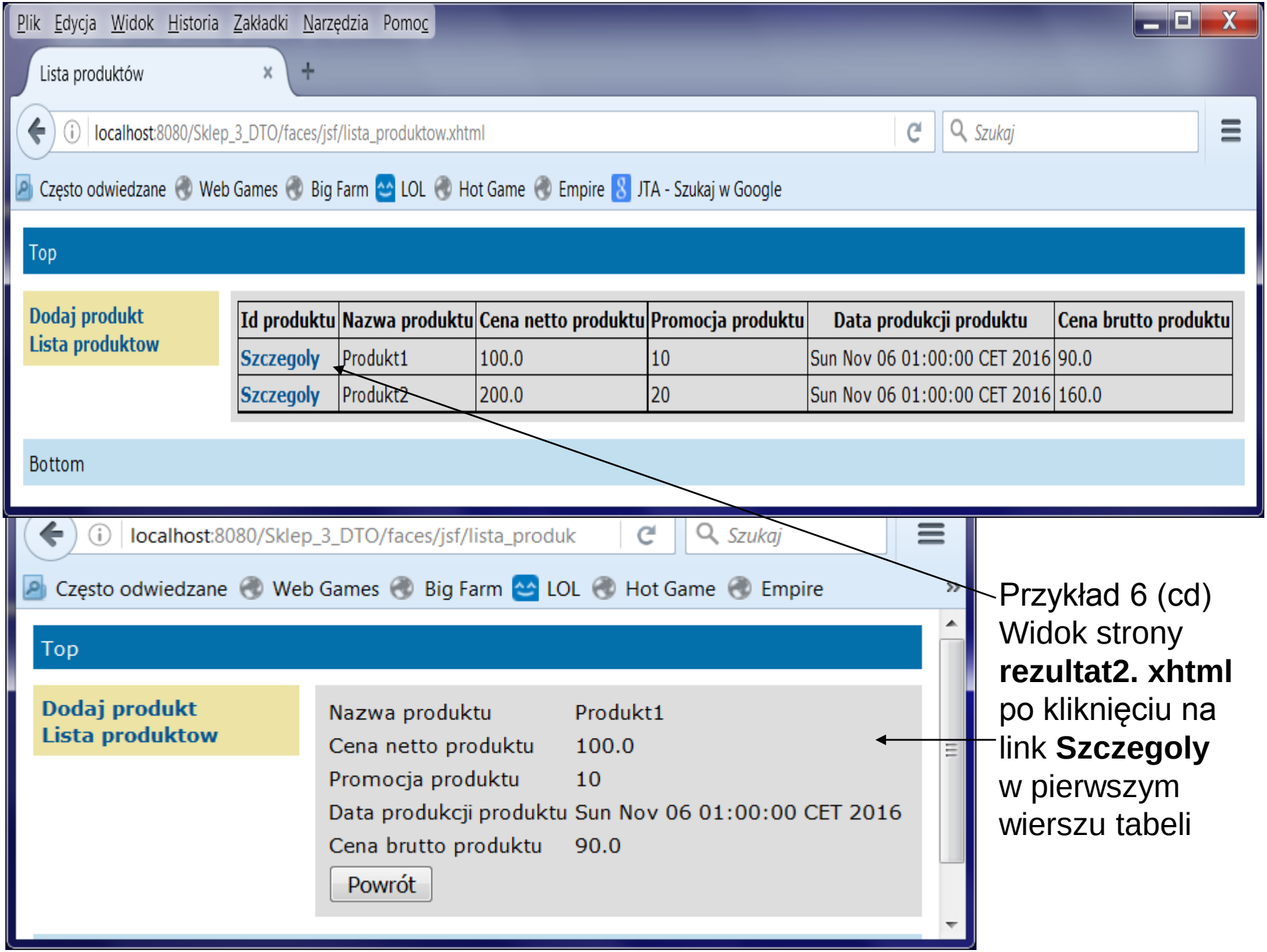


Przykład 6 – obsługa zdarzenia za pomocą słuchacza zdarzeń, zarejestrowanego za pomocą znacznika **f:setPropertyActionListener**

```
<h:dataTable value="#{managed_produkt.items}" var="item"
  border="0 "   cellpadding="2"   cellspacing="0"   rowClasses="jsfcrud_odd_row,jsfcrud_even_row"
  rules="all"   style="border:solid 1px"  lang="pl">
    <h:column>
      <f:facet name="header">
          <h:outputText value="#{bundle['lista_produktow.id']}"/>
      </f:facet>
      <h:commandLink   action= "rezultat2"  value="Szczegoly">
          <f:setPropertyActionListener
                  target="#{managed_produkt.produkt_dto}"
                  value="#{item}"/>
      </h:commandLink>
    </h:column>
```

Metoda umożliwiająca przejście na stronę rezultat2.xhtml i wyświetlenie wybranego **item**

Pobranie obiektu **item** w fazie „żądanie” przez podstawienie do obiektu **produkt_dto**



Przykład 6 (cd)
Widok strony **rezultat2. xhtml** po kliknięciu na link **Szczegoly** w pierwszym wierszu tabeli