Budowa aplikacji wielowarstwowych. Obsługa zdarzeń, zastosowanie walidatorów, wykonanie listy typu Drop Down List.

Laboratorium 4 część 2

Technologie internetowe

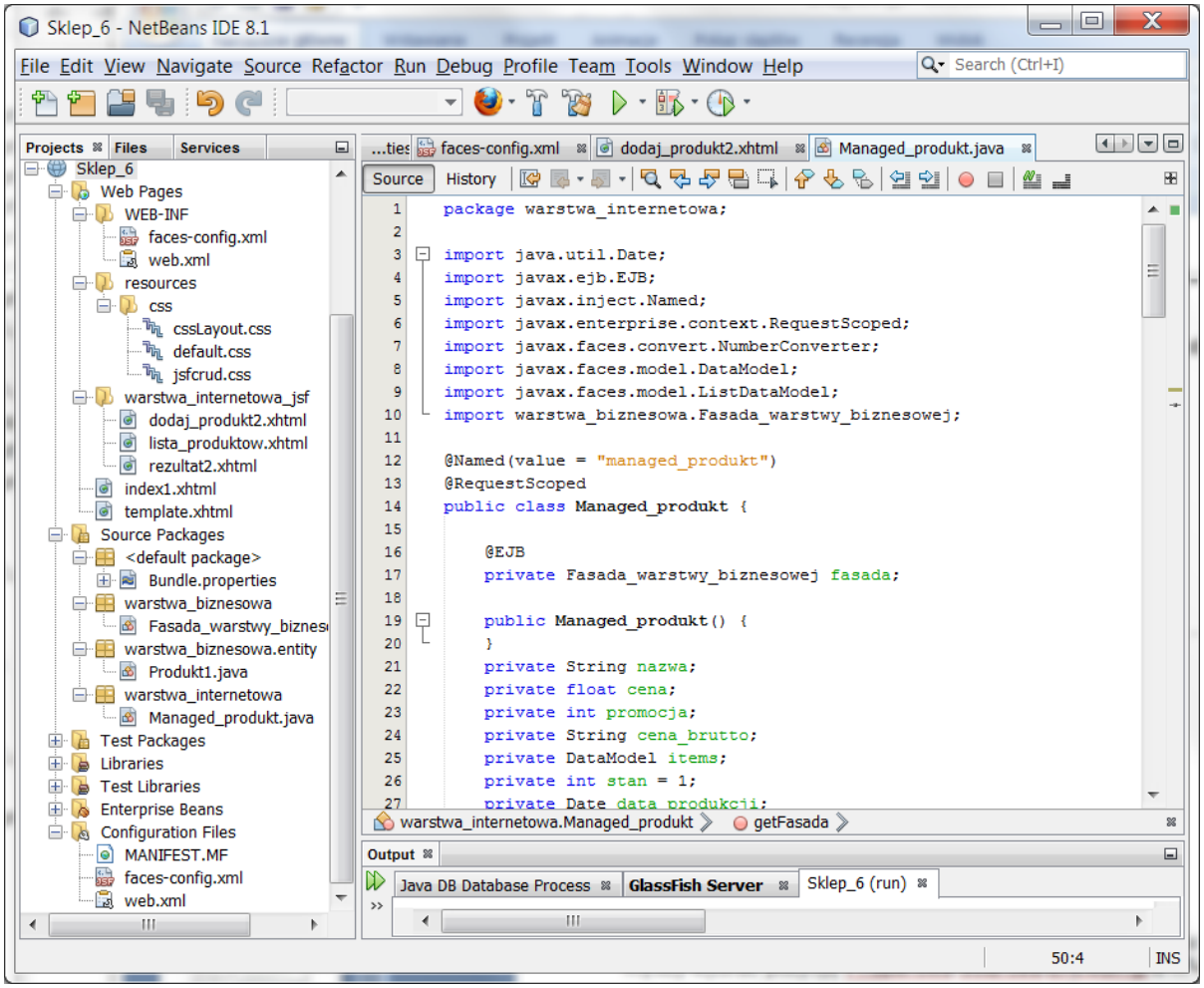
Zofia Kruczkiewicz

Wykaz pytań dotyczących materiału wykorzystanego w lab4, które należy opracować (m.in, wykłady: 3,4, 5).

1. Jaką rolę pełni zdarzenie typu **valueChangeListener ?**
2. Należy scharakteryzować sposoby definiowania obsługi zdarzeń **valueChangeListener**:
   <f:valueChangeListener type="pomoc.Zmiana_danych"/>- co zonacza atrybut **type**?
   <f:valueChangeListener binding="#{managed_produkt.zmiana2}"/> - co oznacza atrybut **binding**?
   oraz jako atrybut **valueChangeListener** w znaczniku **<h:inputText**– jak należy zdefiniować wartość tego atrybutu?.
3. Jaką rolę pełni zdarzenie **actionListener?**
4. Co jest przypisane do atrybutu **action?** Co musi realizować ten atrybut?
   <h:commandLink action="#{managed_produkt.dodaj_produkt}" value="OK" />
5. Co jest przypisane do atrybutu **action** oraz atrybutu **actionListener**? Jaka jest kolejność wykonywanych czynności zdefiniowanych za pomocą podanych atrybutów? Co musi realizować atrybut **action**?
   <h:commandLink action="#{managed_produkt.dane_produktu}" value="OK"
   actionListener="#{managed_produkt.dodaj_produkt}" />
6. Co realizuje atrybut **action?** Jaką rolę pełni znacznik **<f:actionListener**? Jaka jest kolejność wykonywanych czynności zdefiniowanych za pomocą atrybutu **action** i znacznika **<f:actionListener**?
   <h:commandLink action="rezultat2" value="OK" >
   <f:actionListener binding="#{managed_produkt}"/>
   </h:commandLink>
7. Należy opisać rolę znacznika
   <f:validateLongRange minimum="#{managed_produkt.min}" maximum="#{managed_produkt.max}"/>
   zagnieżdżonego w znaczniku **<h:inputText**. Jaką rolę pełnią atrybuty tego znacznika?
8. Należy opisać rolę atrybutu **validator** znacznika **<h:inputText**. Należy podać, jak definiuje się wartość tego atrybutu.
   validator="#{managed_produkt.zakrespromocji},,
9. Jaką rolę pełni znacznik **<h:selectOneMenu** przy wprowadzaniu danych
10. Jaki atrybut znacznika **<h:selectOneMenu** służy do pobrania wybranej wartości?
11. Jaką rolę pełni znacznik **<f:selectItems value="#{managed_produkt.itemsAvailableSelectOne}"/>** zagnieżdżony w znaczniku **<h:selectOneMenu.** Jaką rolę pełni atrybut **value** tego zagnieżdżonego znacznika – należy wyjaśnić, co jest przypisane do tego atrybutu i jaką pełni rolę?

1. Czynności początkowe

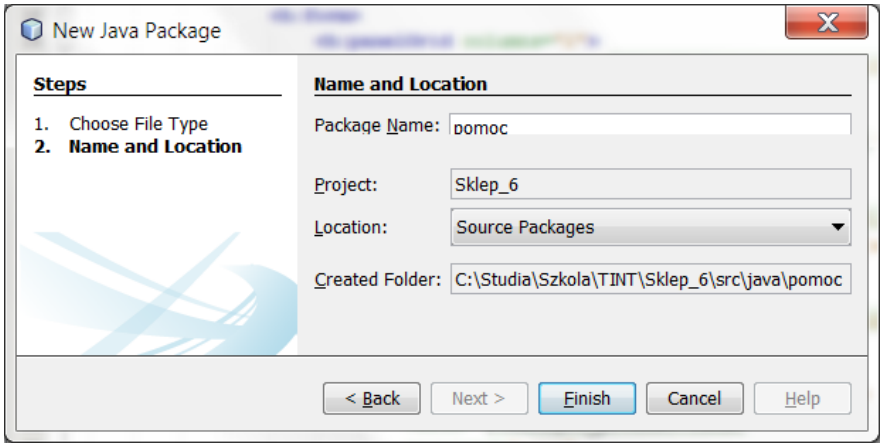
- Należy wykonać kopię programu **Sklep_6,** wykonanego podczas lab4 jako **Sklep_5.**

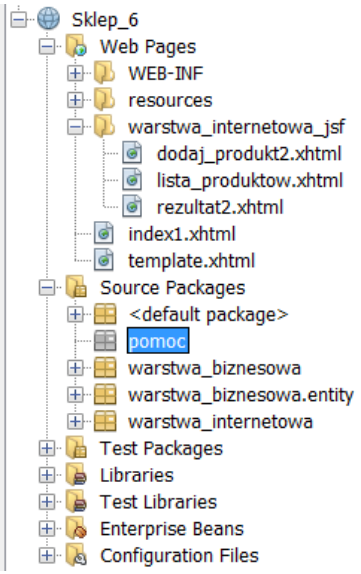


- Ustawić kodowanie UTF-8; po zaznaczeniu nazwy projektu w oknie **Projects** prawym klawiszem myszy wybrać pozycję **Properties/Sources/Encoding/UTF-8**

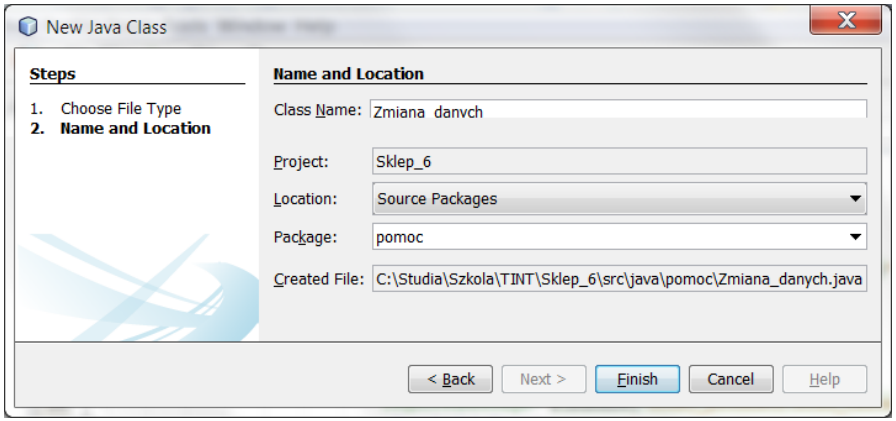
2. Obsługa zdarzeń typu **valueChangeListener**

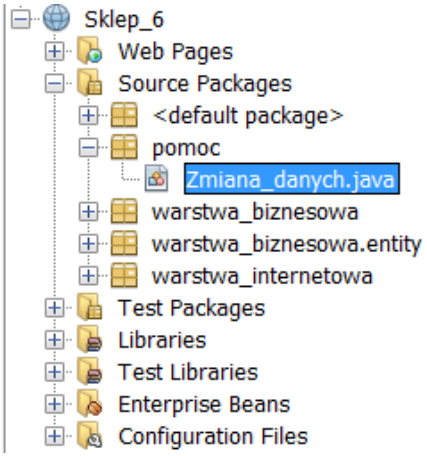
2.1. Należy wykonać pakiet o nazwie pomoc – po kliknięciu prawym klawiszem myszy na nazwę projektu należy wybrać kolejno pozycje **New/Other/Java/Java Package** i po kliknięciu na klawisz **Next** w polu **Package Name** wpisać nazwę nowego pakietu: **pomoc.**





2.2. 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.**





2.3 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);   }
   }
}
```

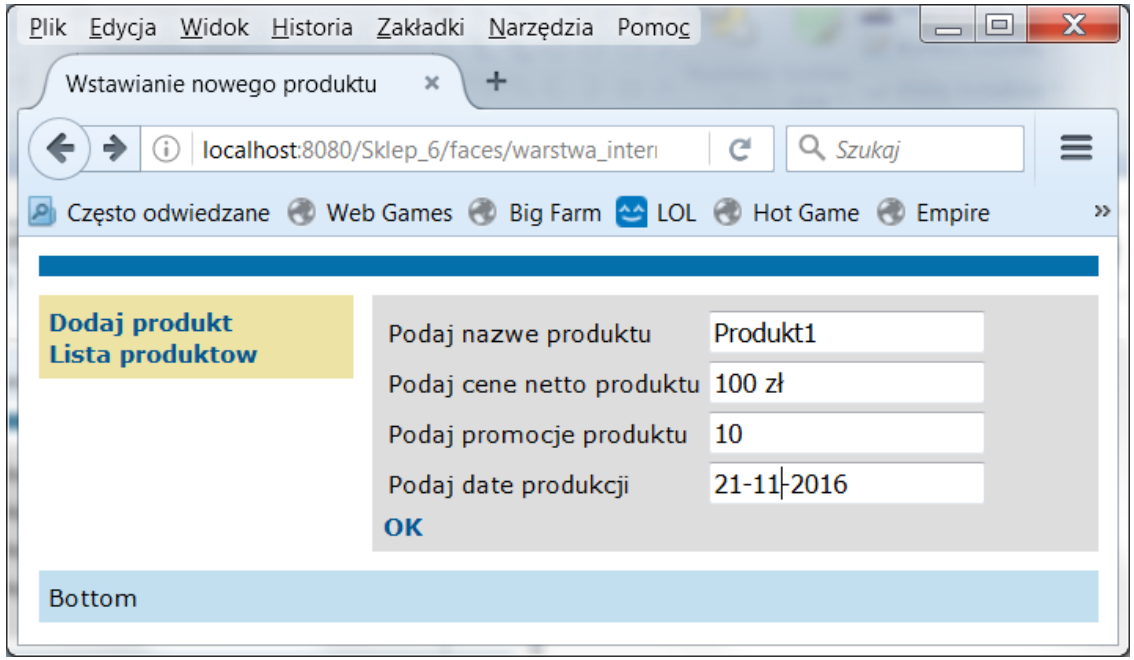
Jeśli wystąpią błędy konwersji (int), należy zastosować konwersję (Integer)

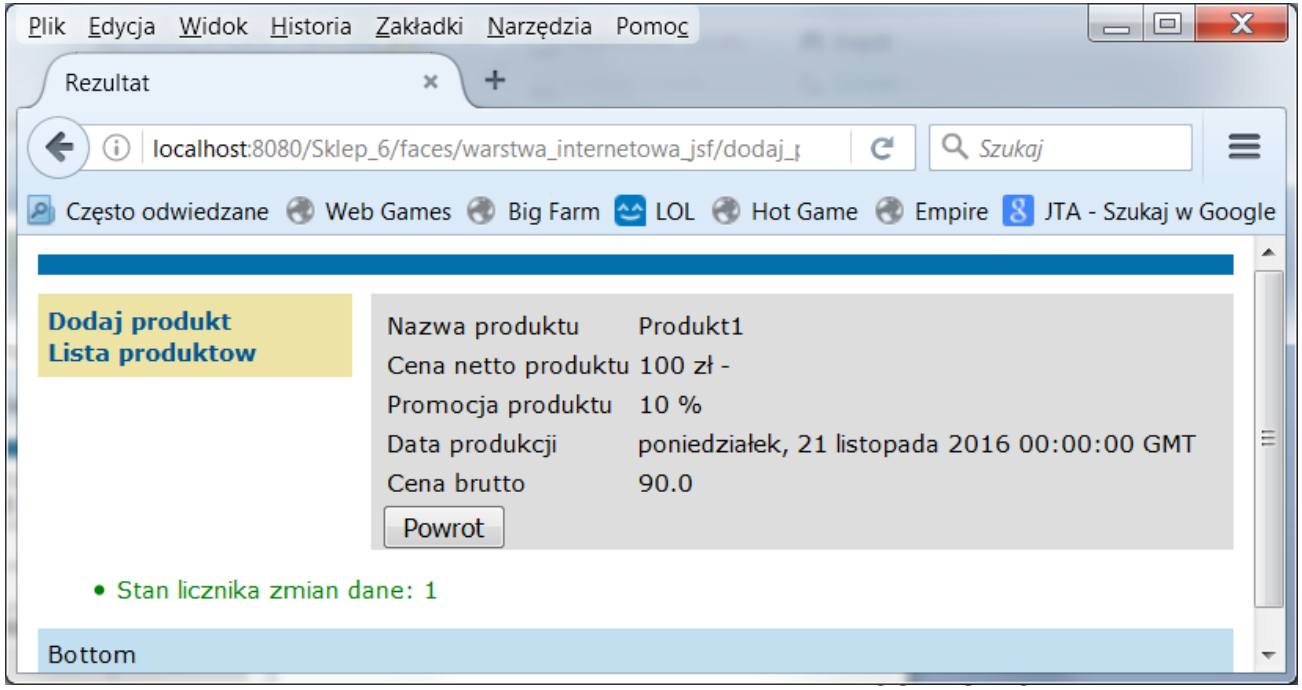
Klucz pozwala przechowywać informacje w kolekcji implementującej interfejs Map, pochodzące z różnych komponentów UI służących do wprowadzania danych.

2.4. Zastosowanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener – wg str. 44-45, wykład 6 (odwołanie do ścieżki pakietowej za pomocą atrybutu type).**

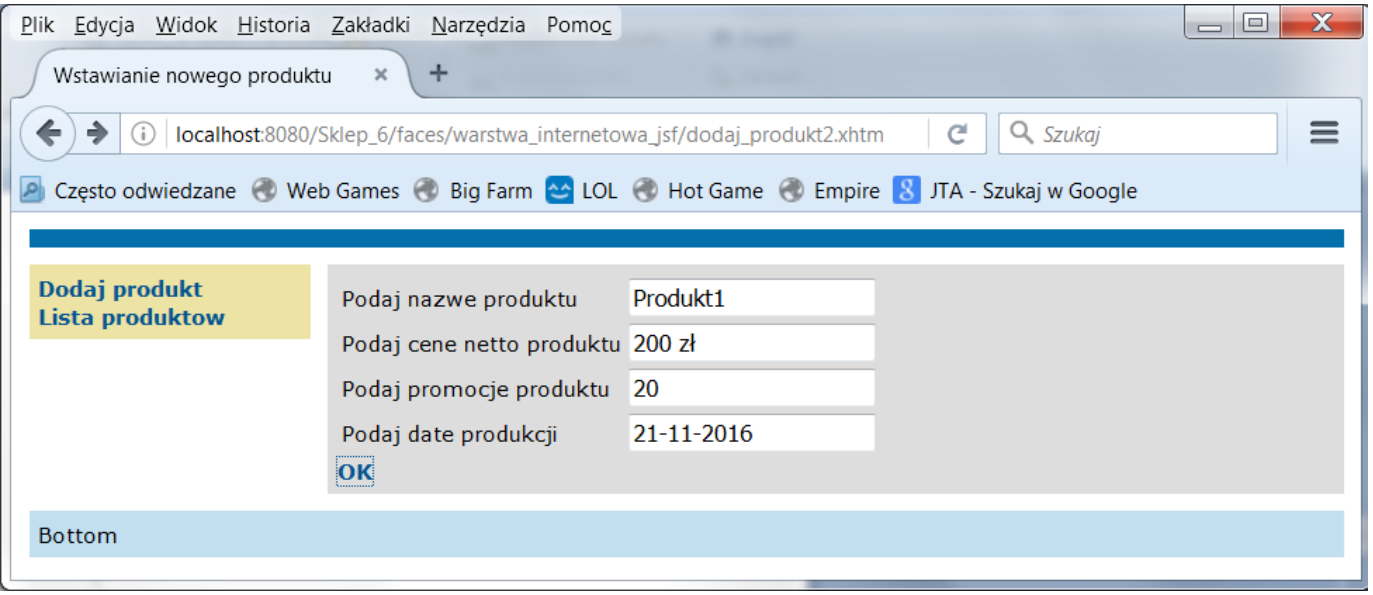
```
<h:outputLabel value="#{bundle['dodaj_produkt2.nazwa']}"  for="nazwa" />
<h:inputText
        id="nazwa"
        title="#{bundle['dodaj_produkt2.nazwa1']}"
        value="#{managed_produkt.nazwa}"
        required="true"
        requiredMessage="#{bundle['dodaj_produkt2.blad_nazwa']}" >
        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

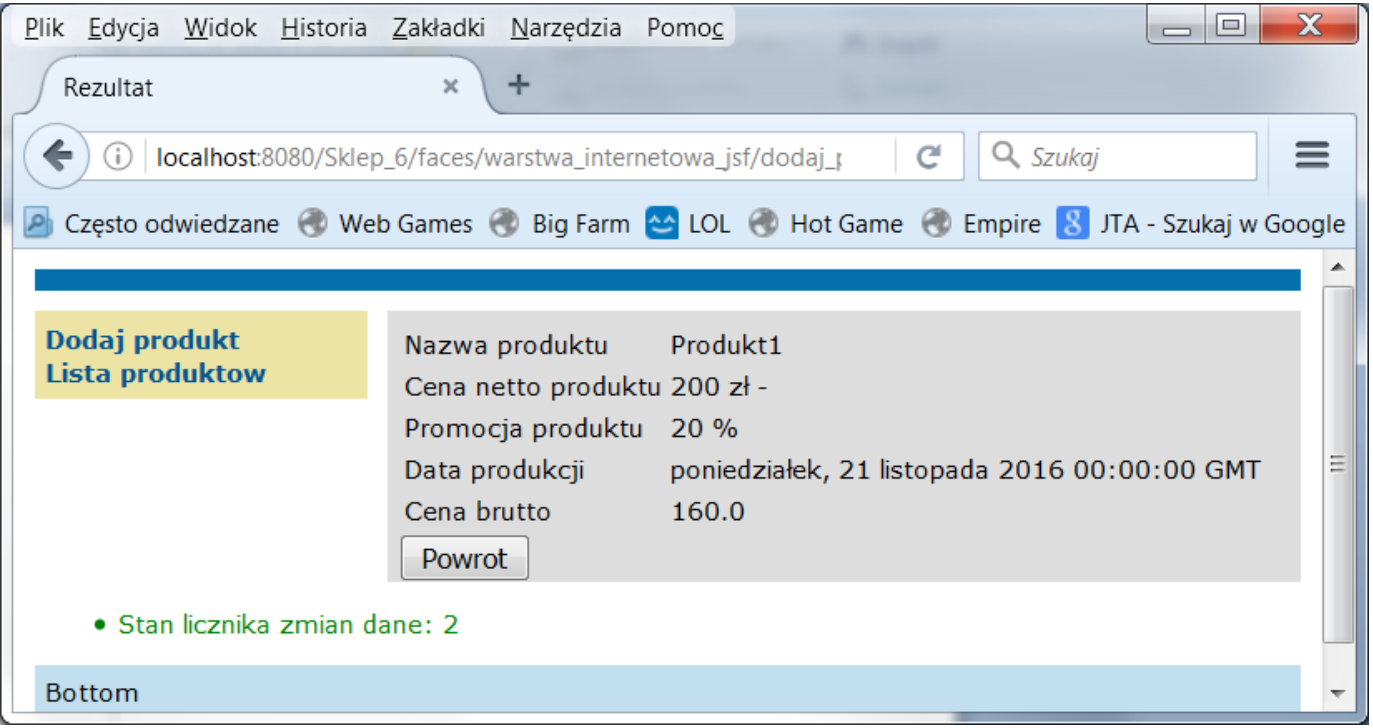
2.5 Prezentacja wyniku



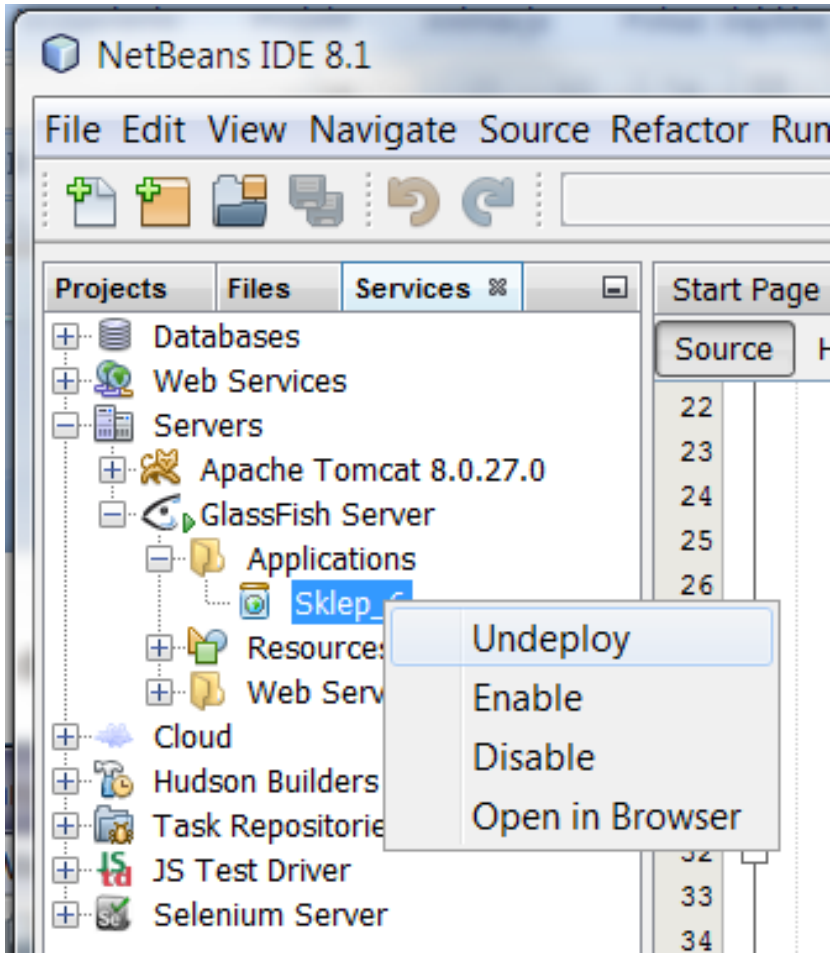


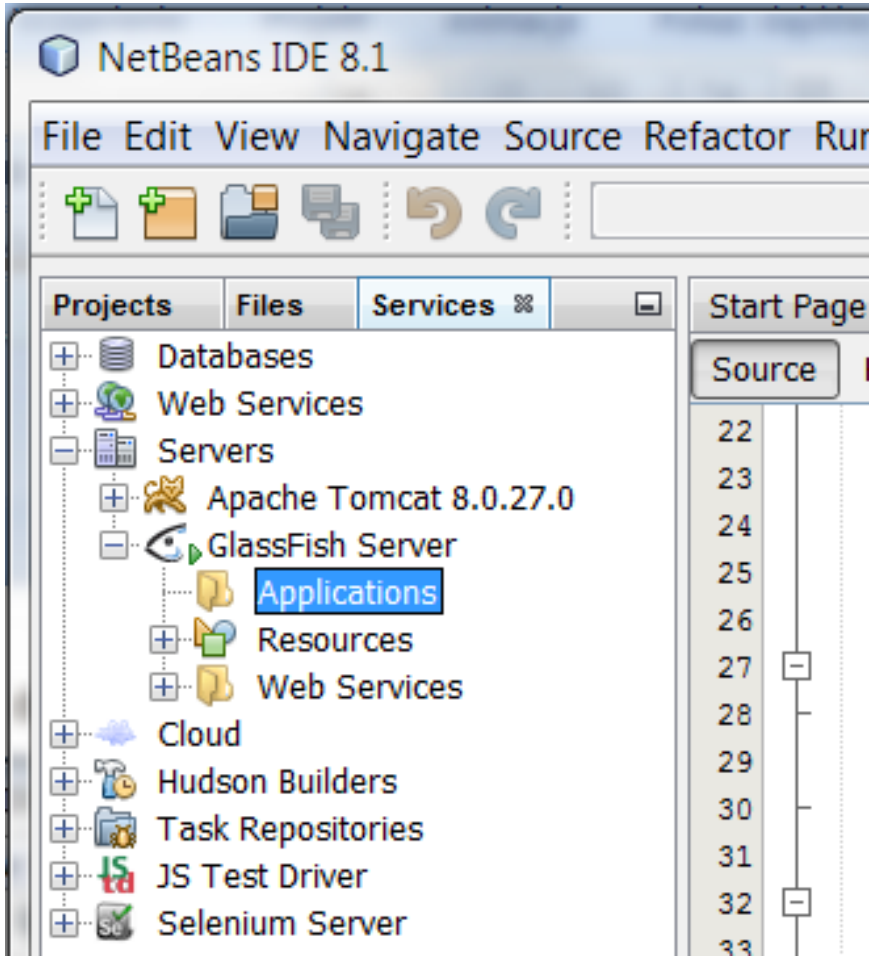
2.5 Prezentacja wyniku (cd)





2.6. **Uwaga dodatkowa:** w celu dokonania zmian w kodzie Java aplikacji należy dokonać operacji **Undeploy**





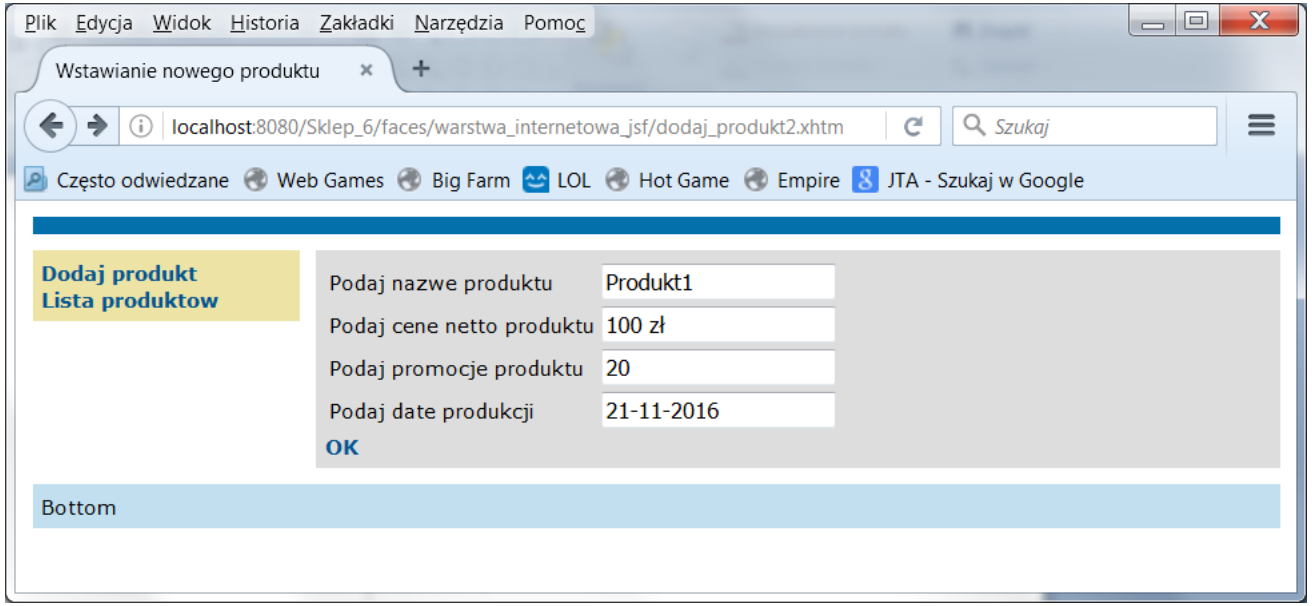
2.7. Zastosowanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener – wg str. 44-45, wykład 6 (odwołanie do obiektu za pomocą atrybutu binding)**

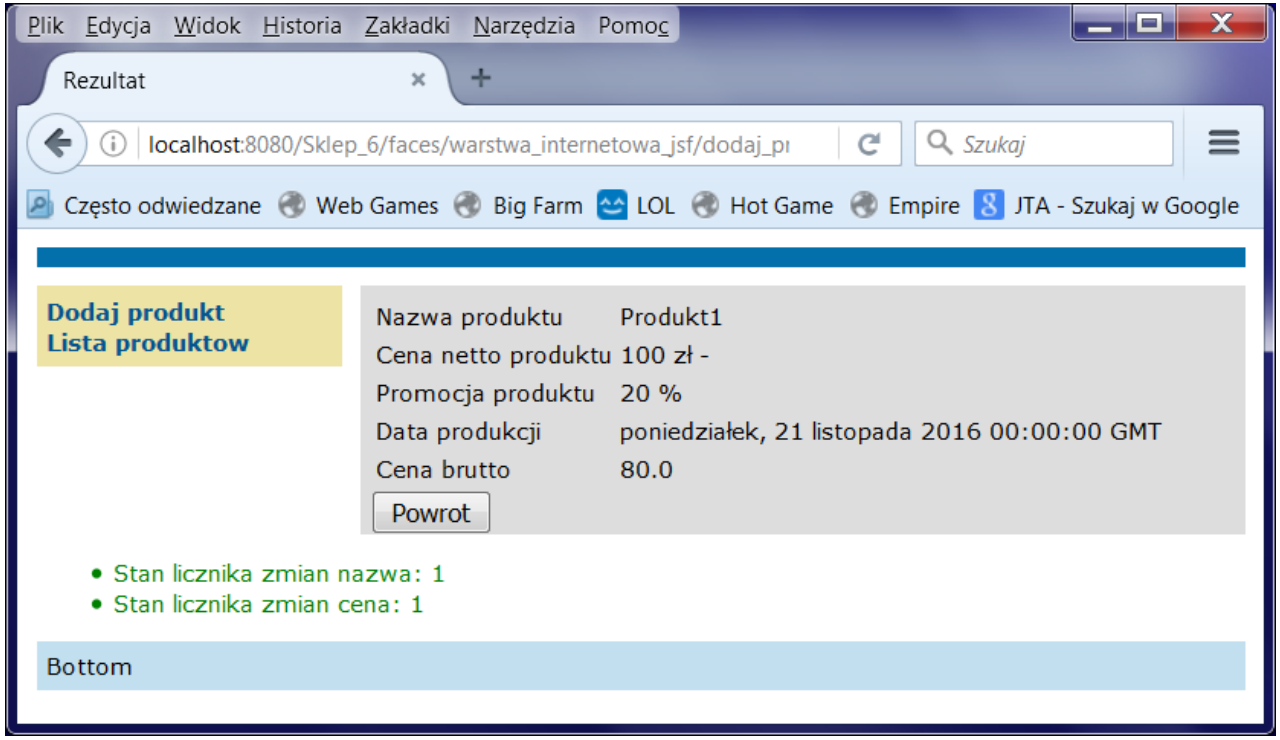
```
<h:outputLabel value="#{bundle['dodaj_produkt2.nazwa']}"  for="nazwa" />
<h:inputText
    id="nazwa”    title="#{bundle['dodaj_produkt2.nazwa1']}"
    value="#{managed_produkt.nazwa}"
    required="true”    requiredMessage="#{bundle['dodaj_produkt2.blad_nazwa']}" >
            <f:valueChangeListener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

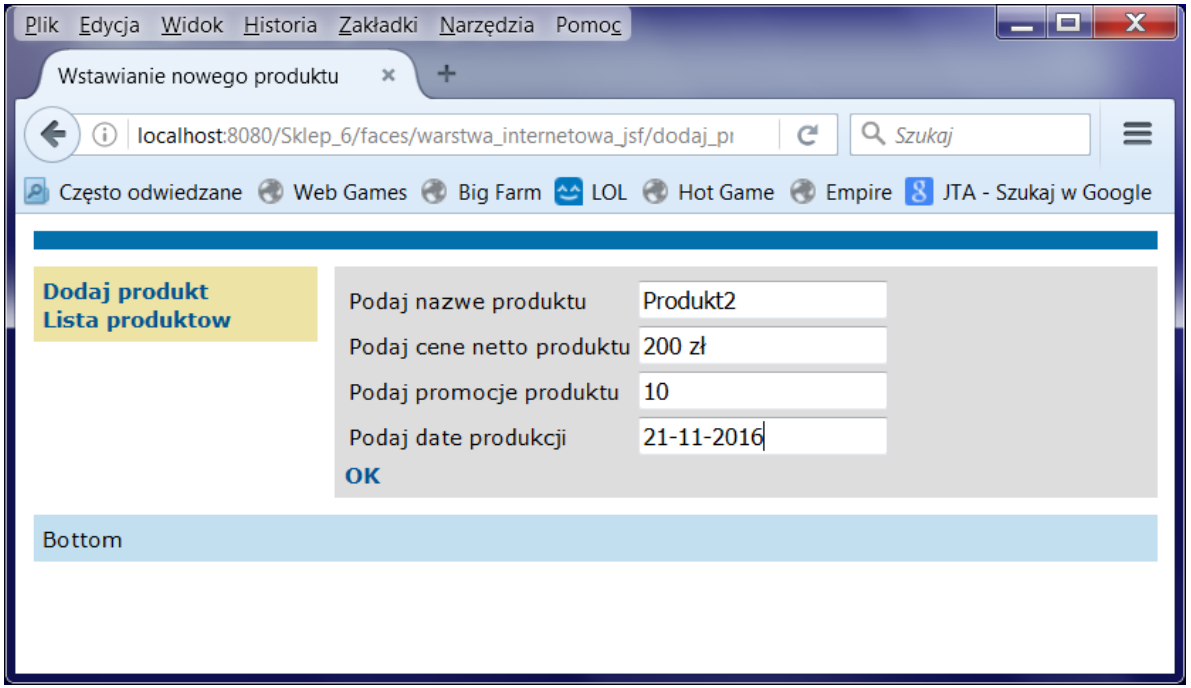
2.8. Zastosowanie definicji klasy **Zmiana_danych** implementującej interfejs **ValueChangeListener – wg str. 45,wykład 6.** Odwołanie do obiektu za pomocą atrybutu **binding** wymaga utworzenie obiektu typu **Zmiana_danych**. Ponieważ obecnie obsługą zdarzeń objęto pola **nazwa** i **cena** na stronie **dodaj_produkt2.xhtml, wykonano dwa niezależnie obiekty typu Zmiana_danych**

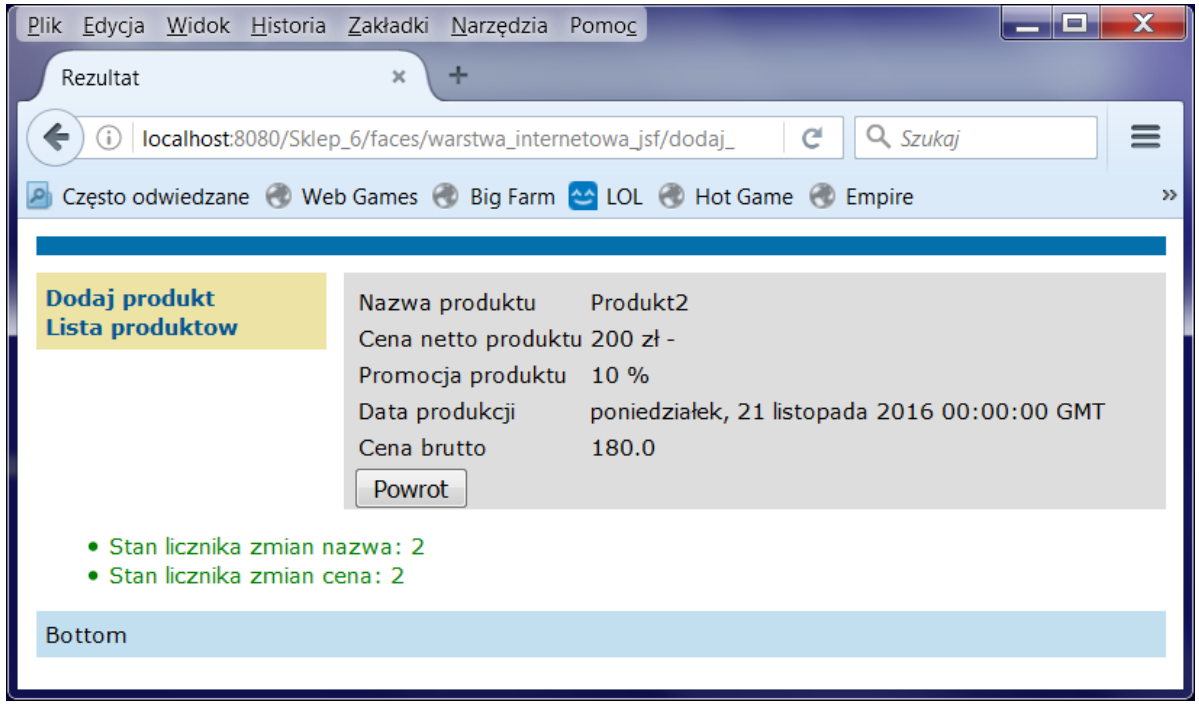
```
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt {
    @EJB
    private Fasada_warstwy_biznesowej fasada;
    public Managed_produkt() { }
    private String nazwa;
    private float cena;
    private int promocja;
    private String cena_brutto;
    private DataModel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

2.9 Prezentacja wyniku









2.10. Zadanie do wykonania:

Należy dodać obsługę zdarzeń dla wybranego atrybutu ze strony **dodaj_produkt2.xhtml z** wykorzystaniem atrybutu **binding** znacznika

<f: valueChangeListener

3. Obsługa zdarzeń typu **actionListener**

3.1. Obsługa zdarzeń typu **ActionListener (str. 46-47, wykład6, przykład1)** – 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]));
      }
  }
```

3.2. Obsługa zdarzeń typu **ActionListener (str. 46, wykład6)** – zmiana definicji znacznika **<h:commandLink** na stronie **dodaj_produkt2.xhtml** oraz modyfikacja metod obsługujących ten znacznik w klasie typu **Managed_produkt**. Stan po zmianie.

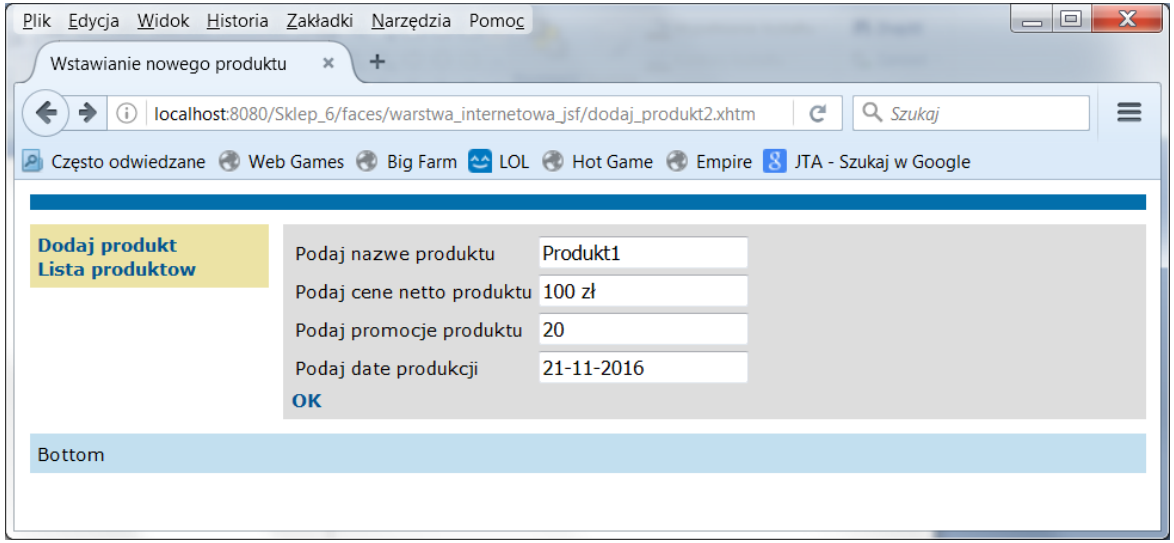
Strona dodaj_produkt2.xhtml

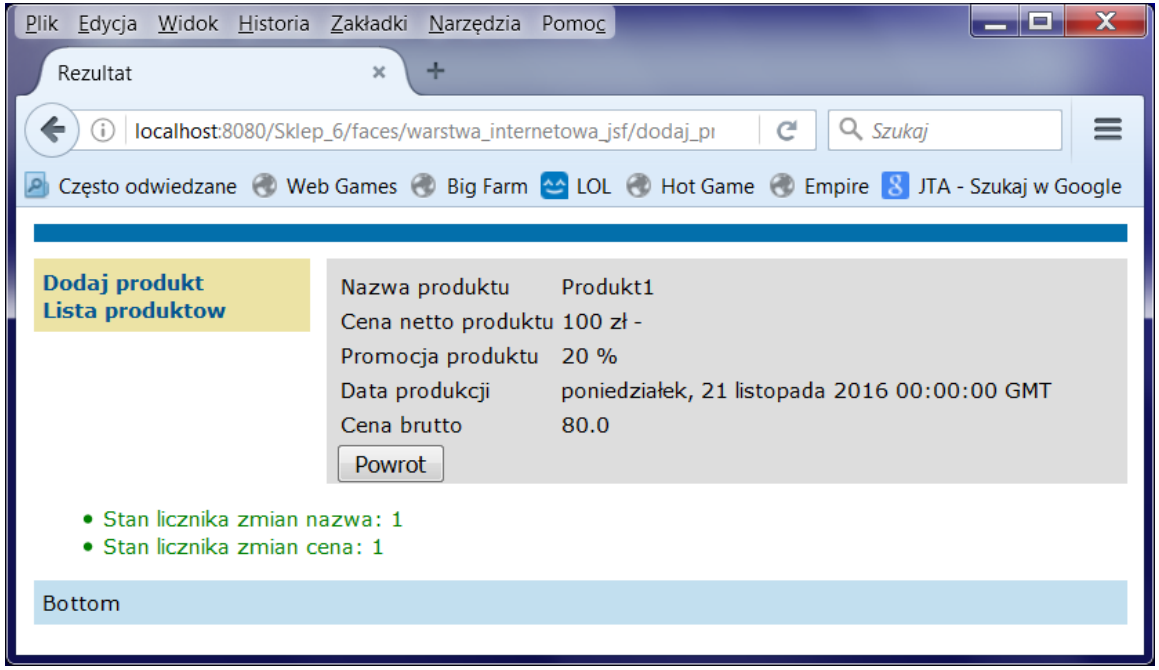
```
<h:commandLink  action="#{managed_produkt.dane_produktu}" value="OK"
                actionListener="#{managed_produkt.dodaj_produkt}" />
```

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";
   }
```

3.3. Prezentacja wyniku





3.4. Obsługa zdarzeń typu **ActionListener (str. 46, wykład6)** – zmiana definicji znacznika **<h:commandLink** na stronie **dodaj_produkt2.xhtml** oraz modyfikacja metod obsługujących ten znacznik w klasie typu **Managed_produkt**. Klasa ta implementuje interfejs **ActionListener.** Stan po zmianie.

Strona dodaj_produkt2.xhtml

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

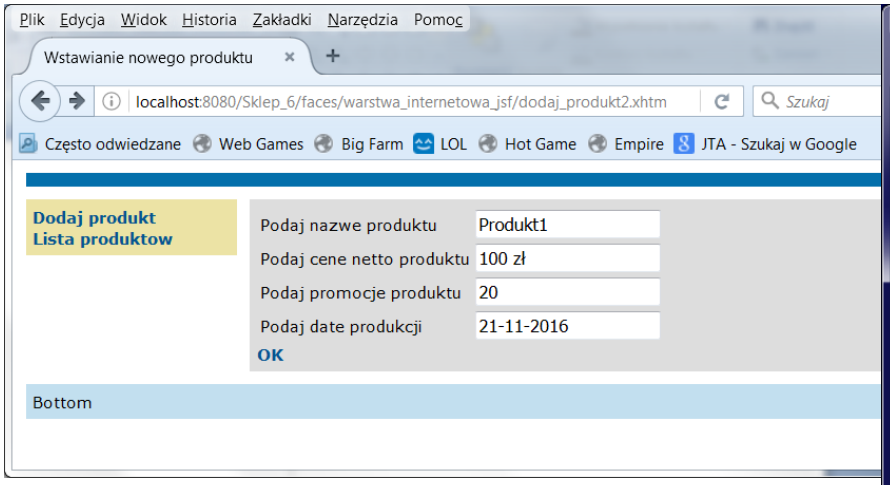
Klasa typu Managed_produkt

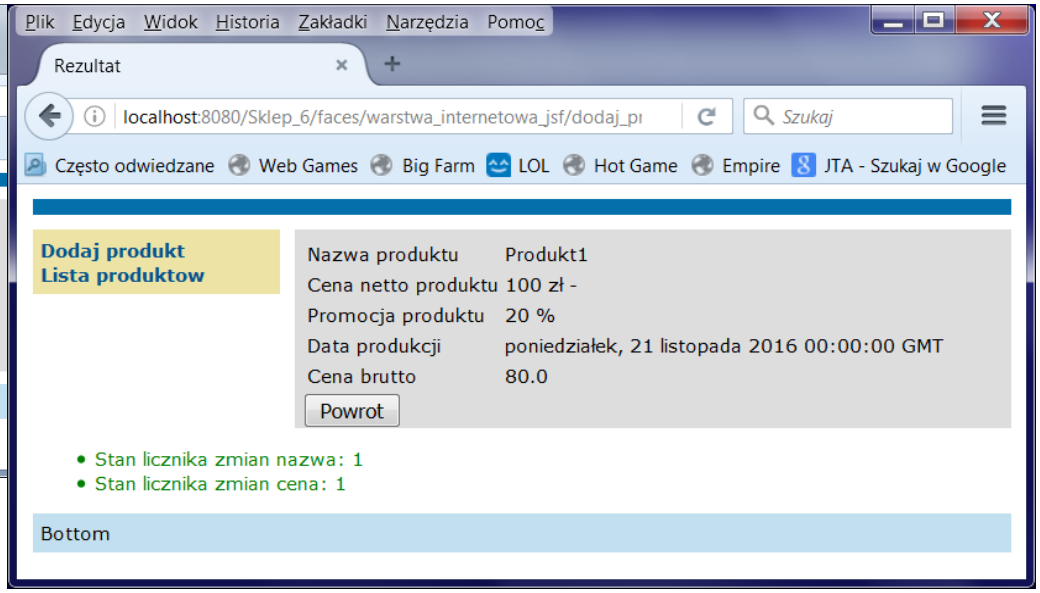
```
package warstwa_internetowa;

import java.util.Date;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.faces.convert.NumberConverter;
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 pomoc.Zmiana_danych;
import warstwa_biznesowa.Fasada_warstwy_biznesowej;

@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt implements ActionListener{
```

3.5. Obsługa zdarzeń typu **ActionListener (str. 46, wykład6)** – zmiana definicji znacznika **<h:commandLink** na stronie **dodaj_produkt2.xhtml** oraz modyfikacja metod obsługujących ten znacznik w klasie typu **Managed_produkt**. Klasa ta implementuje interfejs **ActionListener.** Stan po zmianie (cd)

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

3.6. Prezentacja wyniku





3.7. Zadanie do wykonania:

Należy dodać obsługę zdarzeń dla znacznika <h:commandLink tak, aby jego definicja była następująca:

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

definiując odpowiednio w klasie typu Managed_produkt metodę

public void processAction(ActionEvent event) throws AbortProcessingException

{ /*…*/}

4. Walidacja danych

4.1. Pierwszy sposób przy wprowadzaniu danych (**str. 58, wykład 6**) – zastosowanie walidatora LongRangeValidator

Plik dodaj_produkt2.xhtml

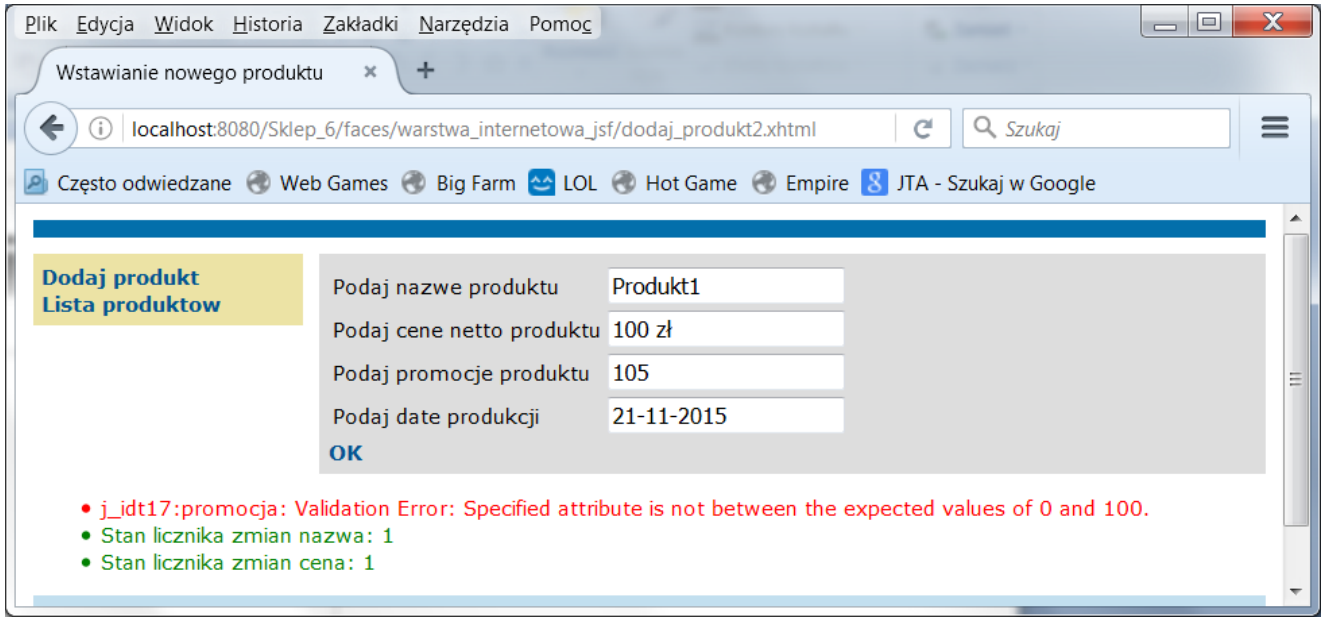
```
<h:outputLabel value="#{bundle['dodaj_produkt2.promocja']}" for="promocja" />
<h:inputText
    id="promocja"     title="#{bundle['dodaj_produkt2.promocja1']}"
    value="#{managed_produkt.promocja}"
    required="true"  requiredMessage="#{bundle['dodaj_produkt2.blad_promocja']}" >
    <f:converter converterId="javax.faces.Integer"  />
    <f:validateLongRange minimum="#{managed_produkt.min}"   maximum="#{managed_produkt.max}"/>
</h:inputText>
```

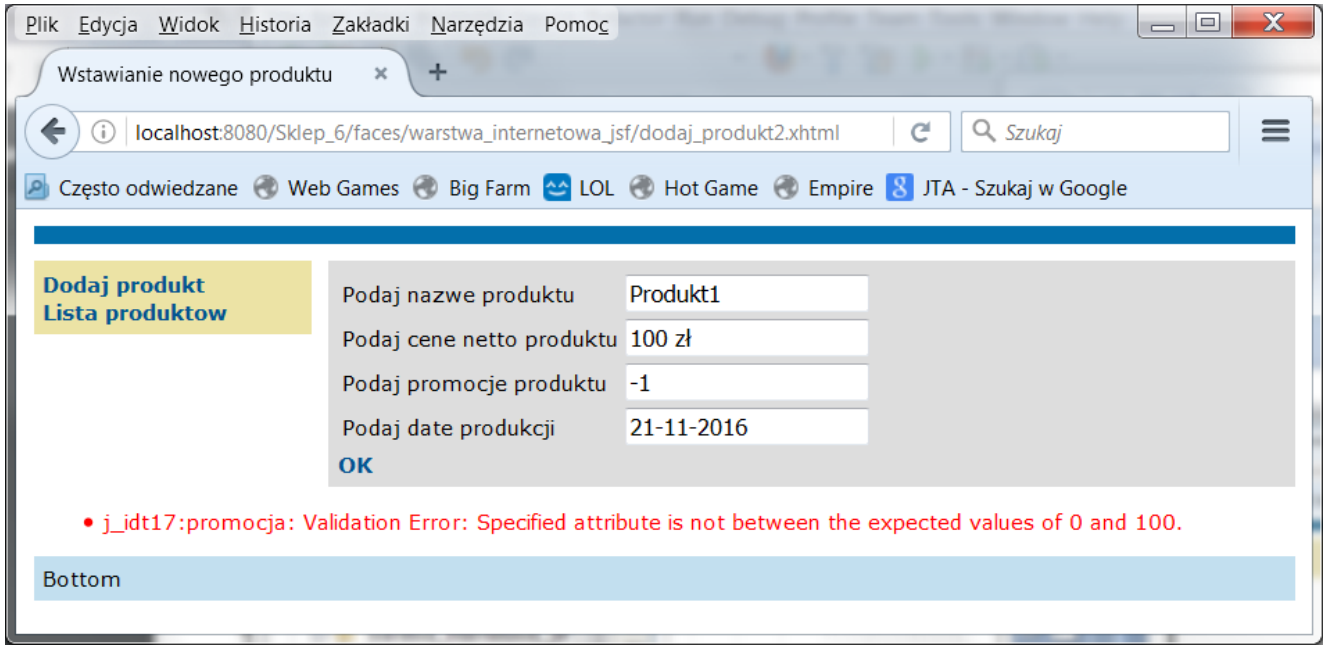
==========

Klasa Managed_produkt

```
public int getMin() {
        return 0;
    }

    public int getMax() {
        return 100;
    }
```

4.2. Prezentacja wyników





4.3. Drugi sposób przy wprowadzaniu danych – zastosowanie metody do walidacji danych o następującej liście parametrów:
(FacesContext context, UIComponent toValidate, Object value) do obsługi atrybutu **validator** znacznika do wprowadzania danych.

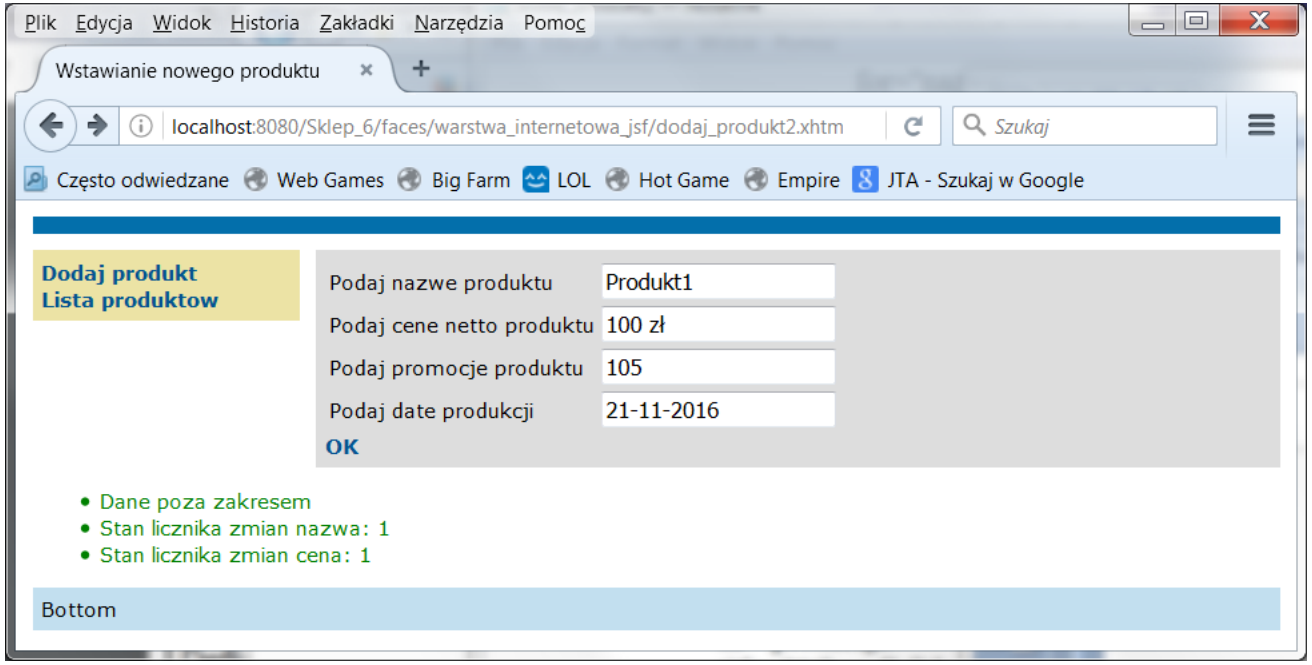
Plik dodaj_produkt2.xhtml

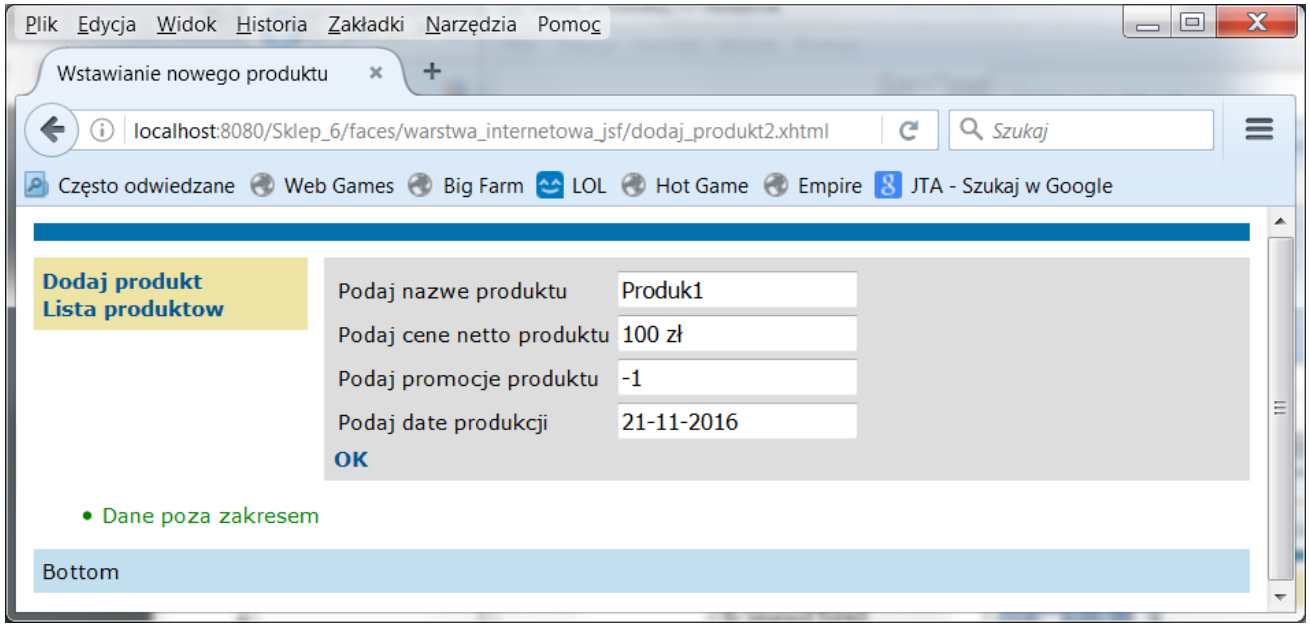
```
<h:outputLabel value="#{bundle['dodaj_produkt2.promocja']}" for="promocja" />
<h:inputText
    id="promocja"     title="#{bundle['dodaj_produkt2.promocja1']}"
    value="#{managed_produkt.promocja}"
    required="true"  requiredMessage="#{bundle['dodaj_produkt2.blad_promocja']}"
    validator="#{managed_produkt.zakrespromocji}">
    <f:converter converterId="javax.faces.Integer"  />
</h:inputText>
```

4.4. Drugi sposób przy wprowadzaniu danych – definicja metody do walidacji danych o następującej liście parametrów: **(FacesContext context, UIComponent toValidate, Object value)**

Klasa Managed_produkt (należy uzupełnić importy klas przez wykorzystanie pozycji Fix Imports)

```
package warstwa_internetowa;
import java.util.Date;
import javax.ejb.EJB;
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.component.UIInput;
import javax.faces.context.FacesContext;
import javax.faces.convert.NumberConverter;
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 pomoc.Zmiana_danych;
import warstwa_biznesowa.Fasada_warstwy_biznesowej;
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt implements ActionListener{
/*……*/
 public void zakrespromocji(FacesContext context, UIComponent toValidate, Object value) {
       stan = 1;
       int input =((Long) value).intValue();
       if (input < getMin() || input > getMax()) {
          ((UIInput) toValidate).setValid(false);
          FacesMessage message = new FacesMessage("Dane poza zakresem");
          context.addMessage(toValidate.getClientId(context), message);
          stan = 0;
       }
   }
```

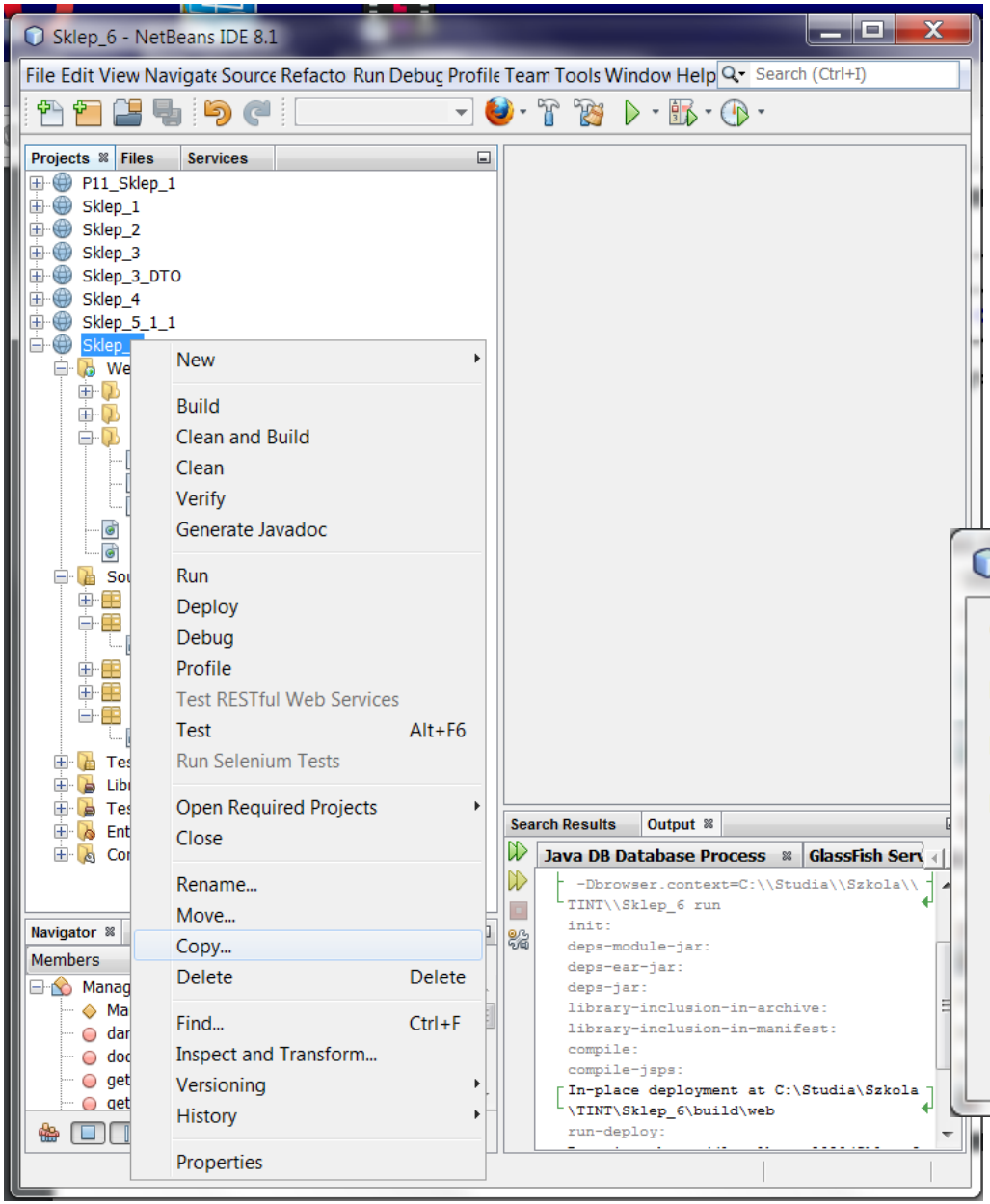


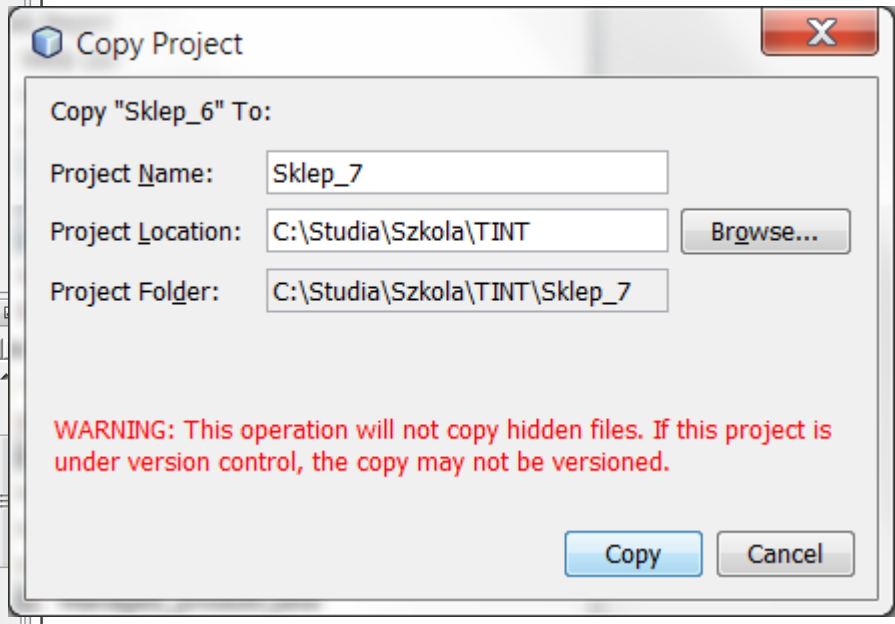


4.6. Zadanie do wykonania:
Zastosować jeden z wybranych sposobów walidacji do pola cena

5. Zastosowanie komponentu **h:selectOneMenu** (typu Drop Down List) na stronie lista_produktow2.xhtml do wyboru promocji

5.1. Należy wykonać kopię programu Sklep_6 o nazwie Sklep_7





5.2. Dodanie do strony **dodaj_produkt2.xhtml** komponentu typu **h:selectOneMenu** – do wprowadzania promocji (zmiana dotychczas stosowanego znacznika **<h:inputText** do wprowadzania promocji). Należy usunąć niepotrzebny kod z klasy typu **Managed_produkt.**

```
<h:outputLabel value="#{bundle['dodaj_produkt2.promocja']}" for="promocja" />
<h:selectOneMenu
        id="promocja"
        title="#{bundle['dodaj_produkt2.promocja1']}"
        value="#{managed_produkt.promocja}"
        required="true"
        requiredMessage="#{bundle['dodaj_produkt2.blad_promocja']}" >
          <f:selectItems value="#{managed_produkt.itemsAvailableSelectOne}"/>
</h:selectOneMenu>
```

5.3. Należy dodać do pakietu pomoc **nową klasę JSFPomoc** i wstawić następujący kod metody getSelectItems do tej klasy

```
package pomoc;

import java.util.List;
import javax.faces.model.SelectItem;

public class JSFPomoc {
    public static SelectItem[] getSelectItems(List<?> entities, boolean selectOne) {
        int size = selectOne ? entities.size() + 1 : entities.size();
        SelectItem[] items = new SelectItem[size];
        int i = 0;
        if (selectOne) {
            items[0] = new SelectItem("", "---");
            i++;  }
        for (Object x : entities) {
            items[i++] = new SelectItem(x, x.toString());
        }
        return items;
    }
}
```

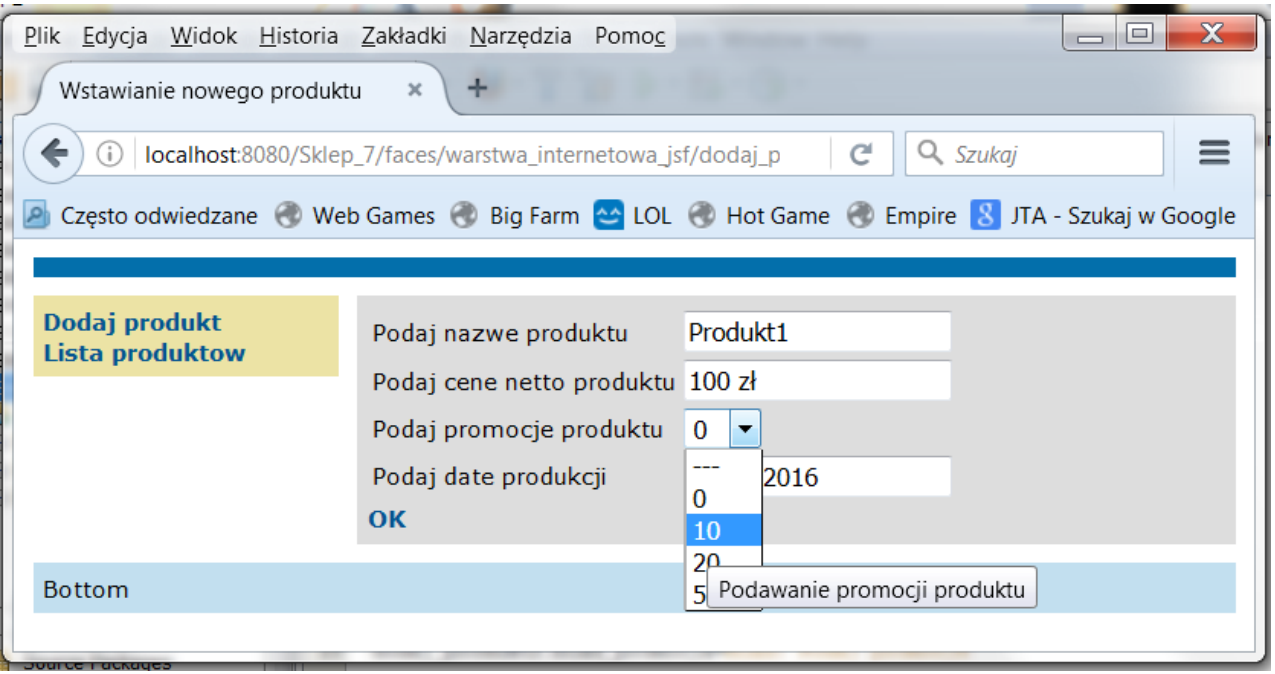
5.4. Dodanie metody w klasie **Fasada_warstwy_biznesowej** do generowania modelu listy rozwijanej, zawierającej wykaz promocji do wyboru z listy

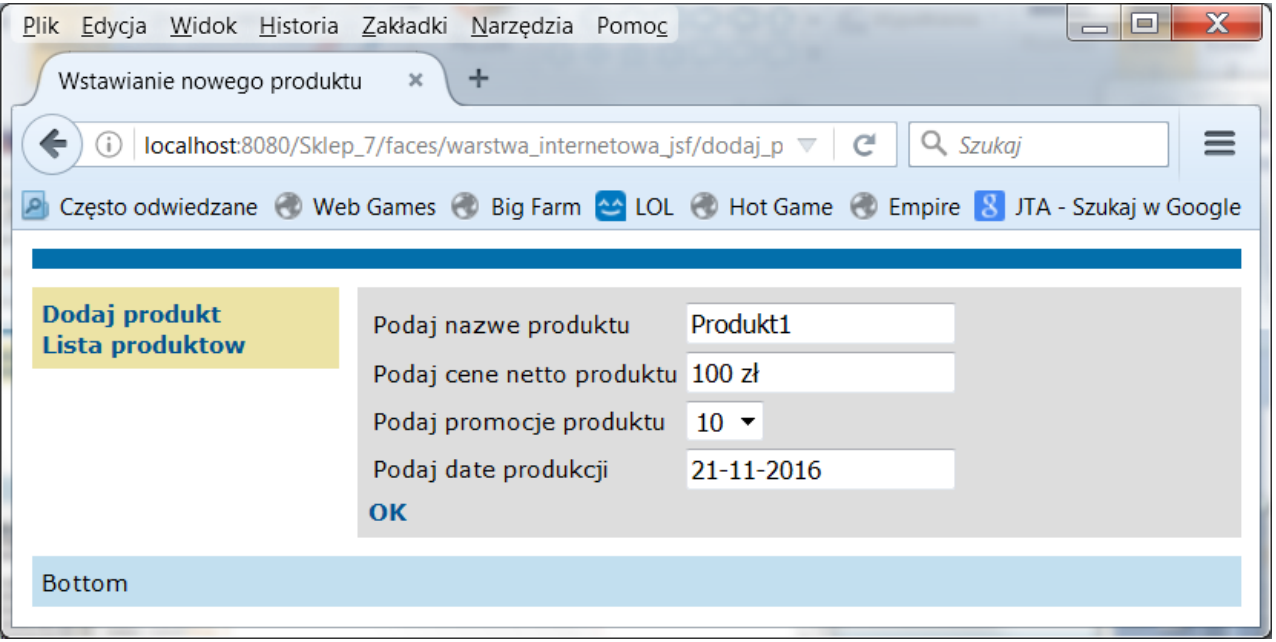
```
public ArrayList<Integer> findAll() {
    ArrayList<Integer> pom = new ArrayList();
    pom.add(0);
    pom.add(10);
    pom.add(20);
    pom.add(50);
    return pom;
  }
```

5.5. Dodanie do klasy **Managed_produkt** metody obsługującej wybór z listy – właściwość managed_produkt.itemsAvailableSelectOne dla znacznika f:selectItems value

```
public SelectItem[] getItemsAvailableSelectOne() {
        return JSFPomoc.getSelectItems(fasada.findAll(), true);
    }
```

5.6. Prezentacja działania programu (1)





5.7. Prezentacja programu (2)