Zastosowanie konwerterów, słuchaczy zdarzeń, walidatorów oraz komponentów wyboru

wg

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

Technologie internetowe 6

Konwertery liczbowe i daty

Konwertery - javax.faces.convert

Klasa w pakiecie javax.faces.convert	Atrybut ConvertID
BigDecimalConverter	javax.faces.BigDecimal
BigIntegerConverter	javax.faces.BigInteger
BooleanConverter	javax.faces.Boolean
ByteConverter	javax.faces.Byte
CharacterConverter	javax.faces.Character
DateTimeConverter	javax.faces.DateTime
DoubleConverter	javax.faces.Double
EnumConverter	javax.faces.Enum
FloatConverter	javax.faces.Float
IntegerConverter	javax.faces.Integer
LongConverter	javax.faces.Long
NumberConverter	javax.faces.Number
ShortConverter	javax.faces.Short

Konwertery liczbowe

Atrybuty znacznika convertNumber

Atrybut	Typ	Opis
binding	NumberConverter	Używany do powiązania konwertera z właściwością obiektu typu Managed Bean
currencyCode	String	ISO 4217 - kod oznaczeń własnych programisty
currencySymbol	String	Własny symbol programisty
for	String	Używany w komponentach kompozytowych do powiązania z wybranym z wielu komponentów
groupingUsed	Boolean	Specyfikuje, czy w formatowanym wyjściu znajdują się separatory grupowania.
integerOnly	Boolean	Określa, czy tylko część całkowitą wartości będzie konwertowana.

locale	String lub Locale	Kod, którego numer stylu jest używany do formatowania lub analizowania danych.
maxFractionDigits	int	Maksymalna liczba cyfr sformatowanych w części ułamkowej formatowanego wyjścia
maxIntegerDigits	Int	Maksymalna liczba cyfr sformatowanych w części całkowitej formatowanego wyjścia.
minFractionDigits	int	Minimalna liczba cyfr sformatowanych w części ułamkowej formatowanego wyjścia
minIntegerDigits	int	Minimalna liczba cyfr sformatowanych w części całkowitej formatowanego wyjścia.
pattern	String	Niestandardowy wzór formatowania, który określa sposób sformatowana i analizowania ciągu znaków liczby.
type	String	Określa, czy ciąg znaków jest analizowany i formatowane jako liczba, waluta, lub procent. Jeśli nie podano, używa się formatu liczby.

Konwerter daty

Atrybuty znacznika convertDateTime

Atrybut	Typ	Opis
binding	DateTimeConverter	Używany do przypisania konwertera do właściwości obiektu typu Managed Bean
dateStyle	String	Definiuje format daty lub fragment zawierający datę, podany przez klasę **java.text.DateFormat**. Stosowany tylko wtedy, gdy typ jest datą lub data i czasem lub jeśli szablon nie zdefiniowany Poprawne wartości formatu: domyślna, krótka, średnia, długa i pełna. Jeśli nie podano wartości formatu, używana jest domyślna wartość.
for	String	Używany w złożonych komponentach. Odnosi się do jednego z obiektów nalżących do złożonego komponentu, wewnątrz którego ten znacznik jest zagnieżdżony.

locale	String or Locale	Kod kraju, którego predefiniowane style dla dat i czasu są używane podczas formatowania lub przetwarzania. Jeśli nie podano, wykorzystywany jest kod kraju, zwrócony przez **javax.faces.context.FacesContext.getLocale**.
pattern	String	Niestandardowy wzór formatowania, który określa, jak łańcuch data / czas powinien być sformatowany i analizowany. Jeżeli ten atrybut jest określony, ignorowane są atrybuty **dateStyle, timeStyle i type**.
timeStyle	String	Definiuje format czasu lub część łańcucha, dotyczący czasu w łańcuchu daty, określony przez **java.text.DateFormat**. Stosowany tylko wtedy, kiedy typ jest czasem i szablon nie jest zdefiniowany. Prawidłowe wartości formatu: domyślny, krótki, średnii, długi i pełny. Jeśli wartość nie jest określona, używana jest domyślna wartość formatu.
timeZone	String lub TimeZone	Strefa czasu, w której interpretuje się czas w łańcuchu daty
type	String	Określa, czy wartość ciągu będzie zawierać datę, czas, lub obie. Prawidłowe wartości to: data, czas, lub obie. Jeśli wartość nie jest określona, używana jest data.

Wpływ atrybutu *Locale* na formatowanie danych podczas konwersji

Jeśli w komponentach atrybut **locale** nie jest ustawiony, wtedy przez komponent jest pobierana domyślna wartość tego atrybutu za pomocą **javax.faces.context.FacesContext.getLocale**.

W celu uzyskania różnych formatów, należy ustawić wartość atrybutu locale indywidualnie w każdym komponencie, w przeciwnym wypadku zdefiniować domyślną wartość **locale** w pliku **faces-config.**

```
public class Locales {
public static void main(String[] args) {
        String pattern = "#######.###";
        double value = 111111.111;
        Locale locales[] = Locale.getAvailableLocales();
        for (Locale loc : locales) {
            NumberFormat number_f = NumberFormat.getNumberInstance(loc);
            DecimalFormat decimal_f = (DecimalFormat) number_f;
            decimal_f.applyPattern(pattern);
            String output = decimal_f.format(value);
            System.out.println(pattern + " " + output + " " + loc.toString());  }
    }
}
```

Pattern	**Wyjściowy łańcuch dla wartości:** 111111.111	**locale**
######.###	111111,111	**pl**
######.###	111111,111	**pl_PL**

######.###	111111,111	**fr**
######.###	111111,111	**fr_BE**
######.###	111111,111	**fr_CA**
######.###	111111,111	**fr_FR**
######.###	111111,111	**fr_LU**

######.###	111111,111	**de**
######.###	111111,111	**de_AT**
######.###	111111,111	**de_DE**
######.###	111111,111	**de_LU**

######.###	111111.111	**en**
######.###	111111.111	**en_AU**
######.###	111111.111	**en_CA**
######.###	111111.111	**en_GB**
######.###	111111.111	**en_IE**
######.###	111111.111	**en_IN**
######.###	111111.111	**en_MT**
######.###	111111.111	**en_NZ**
######.###	111111.111	**en_PH**
######.###	111111.111	**en_SG**
######.###	111111.111	**en_US**
######.###	111111.111	**en_ZA**

Rejestrowanie konwerterów

Sposoby rejestrowania konwerterów do komponentu (SR)

1. Zbindowanie atrybutu **value** komponentów do **właściwości (metody typu set i get)** obiektów typu Managed Bean, takiego samego typu jak konwerter.
2. Zagnieżdżenie znaczników jednego ze standardowych konwerterów **f:convertNumber** i **f:convertDateTime** wewnątrz znaczników komponentów
3. Powiązanie atrybutu **converter** w komponencie z instancją konwertera zdefiniowanego w obiekcie typu **Managed Bean (lub standardowym).**
4. Zagnieżdżenie znacznika **f:converter** wewnątrz znacznika komponentu i użycie albo jego atrybutu **converterId** lub jego atrybutu **binding** powiązanego z instancją konwertera zdefiniowanego w obiekcie typu **Managed Bean (lub standardowym).**

Przykład 1

Definicja typów liczbowych właściwości obiektu **Managed_produkt**

```
@ManagedBean
@RequestScoped
public class Managed_produkt {

    @EJB
    private Fasada_warstwy_biznesowej fasada;
    private DataModel items;
    private int stan = 1;
    private Produkt_dto produkt_dto = new Produkt_dto();
```

Sposób 1

```
public String dodaj_produkt() {
    fasada.utworz_produkt(produkt_dto);
    dane_produktu();
    return "rezultat2";
   }
```

```
public void dane_produktu() {
   stan = 1;
   produkt_dto = fasada.dane_produktu();
       if (produkt_dto == null) {
           stan = 0;
           produkt_dto=new Produkt_dto();
       }
    }
```

Metody **właściwości obiektu Managed_produkt,** używanych w atrybutach value komponentów wejściowych i wyjściowych

```
public Float getCena() {
      return produkt_dto.getCena();  }
public void setCena(Float cena) {
      this.produkt_dto.setCena(cena);  }

public Integer getPromocja() {
      return produkt_dto.getPromocja();  }
public void setPromocja(Integer promocja) {
      this.produkt_dto.setPromocja(stan);  }

public Float getCena_brutto() {
      return produkt_dto.getCena_brutto();  }
public void setCena_brutto(Float cena_brutto) {
      this.produkt_dto.setCena_brutto(cena_brutto);   }

public Date getData_produkcji() {
      return produkt_dto.getData_produkcji();  }
public void setCena_brutto(Date data_produkcji) {
      this.produkt_dto.setData_produkcji(data_produkcji);  }
```

Właściwość definiująca konwerter typu **Float**

Właściwość definiująca konwerter typu **Integer**

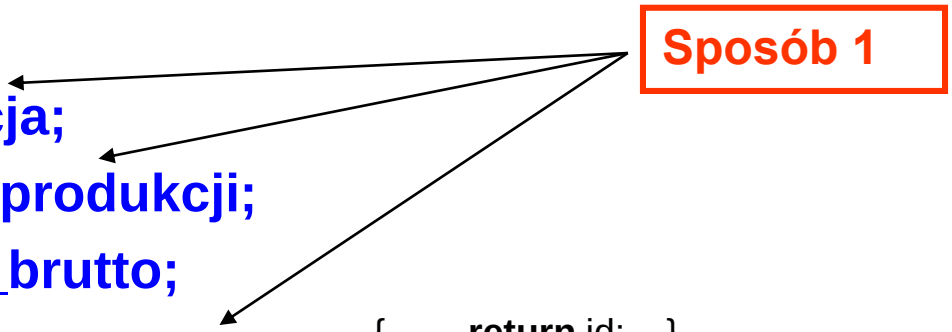
Właściwość definiująca konwerter typu **Float**

Właściwość definiująca konwerter **DataTimeConverter**

Metody definiujące właściwości obiektu **Managed_produkt** umożliwiające bindowanie danych w formularzach JSF – danymi sa pola obiektu **produkt_dto**

Obiekt typu Produkt_dto – implementacja wzorca projektowego Transfer Object

```
public class Produkt_dto {
    protected long id;
    protected String nazwa;
    protected float cena;
    protected int promocja;
    protected Date data_produkcji;
    protected float cena_brutto;
```




```
  public long getId()                                  {   return id;    }
  public void setId(long id)                           {   this.id = id;    }
  public String getNazwa()                             {   return nazwa;    }
  public void setNazwa(String nazwa)                   {   this.nazwa = nazwa;    }
  public float getCena()                               {   return cena;    }
  public void setCena(float cena)                      {   this.cena = cena;    }
  public int getPromocja()                             {   return promocja;    }
  public void setPromocja(int promocja)                {   this.promocja = promocja;    }
  public Date getData_produkcji()                      {   return data_produkcji;    }
  public void setData_produkcji(Date data_produkcji)   {   this.data_produkcji = data_produkcji;    }
  public float getCena_bruttto()                       {   return cena_brutto;    }
  public void setCena_brutto(float cena_brutto)        {   this.cena_brutto = cena_brutto;    }
}
```

Pierwszy sposób rejestrowania konwerterów (SR1)

Fragment pliku jsf, **służący do wprowadzania danych**, korzystający z **właściwości obiektu Managed_produkt** – typ liczbowy właściwości pozwala na formatowanie danych liczbowych przy wprowadzaniu danych (atrybut **value** komponentów **h:inputText**)

```
<h:outputLabel for="cena" value="#{bundle['dodaj_produkt2.podaj_cena']}" />
<h:inputText
      id="cena"
      title="#{bundle['dodaj_produkt2.podaj_cena']}"
      value="#{managed_produkt.cena}"
      required="true"
      requiredMessage="#{bundle['dodaj_produkt2.podaj_cena_blad']}" >
 </h:inputText>
 <h:outputLabel value="#{bundle['dodaj_produkt2.podaj_promocja']}"
                    for="promocja" />
 <h:inputText
       id="promocja"
       title="#{bundle['dodaj_produkt2.podaj_promocja']}"
       value="#{managed_produkt.promocja}"
       required="true"
       requiredMessage="#{bundle['dodaj_produkt2.podaj_promocja_blad']}" >
  </h:inputText>
```

Drugi sposób rejestrowania konwerterów (SR2)

Fragment pliku jsf, **służący do wprowadzania danych**, korzystający z **właściwości obiektu Managed_produkt oraz znacznika zagnieżdżonego f:convertDateTime**, który pozwala na formatowanie danych oraz przy wprowadzaniu danych (atrybut **pattern**)

```
<h:outputLabel value="#{bundle['dodaj_produkt2.podaj_data']}"
                                for="data" />
<h:inputText
    id="data„
    title="#{bundle['dodaj_produkt2.podaj_data']}"
    value="#{managed_produkt.data_produkcji}"
    required="true"
    requiredMessage="#{bundle['dodaj_produkt2.podaj_data_blad']}" >
        <f:convertDateTime pattern="dd-MM-yyyy" />
</h:inputText>
```

Pierwszy sposób rejestrowania konweterów (SR1)

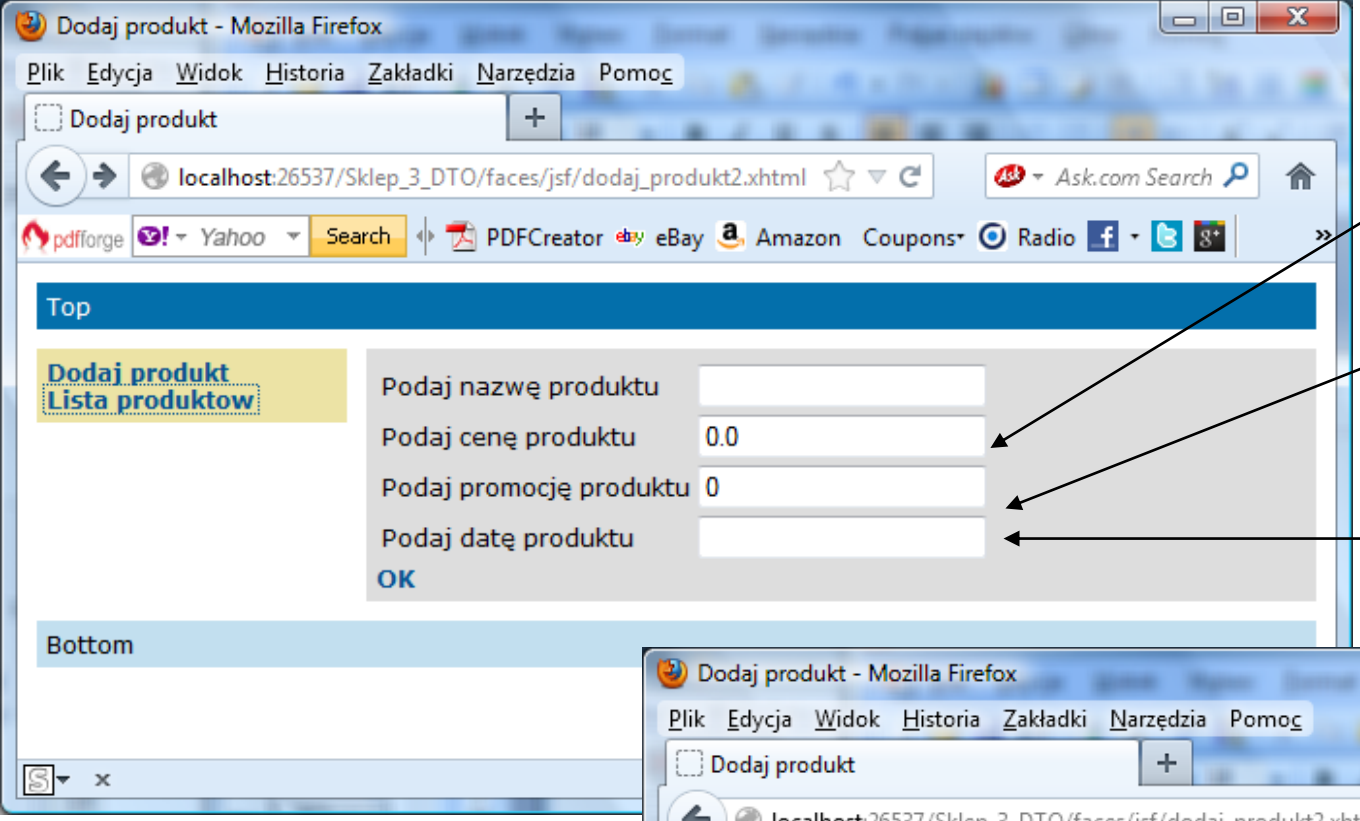
Fragment pliku jsf **służący do wyświetlania danych**, korzystający z **właściwości obiektu Managed_produkt** – typ liczbowy właściwości pozwala na formatowanie danych liczbowych przy wyprowadzaniu danych (atrybut **value** komponentów **h:outputText**)

```
<h:outputLabel  for="cena" value="#{bundle[‘lista_produktow.cena']}" />
<h:outputText  id="cena"
                value="#{managed_produkt.cena}"/>
<h:outputLabel for="promocja" value="#{bundle['lista_produktow.promocja']}"/>
<h:outputText  id="promocja"
               value="#{managed_produkt.promocja}"/>
<h:outputLabel  for="data " value="#{bundle['lista_produktow.data']}" />
<h:outputText  id="data"
               value="#{managed_produkt.data_produkcji}"/>
<h:outputLabel  for="brutto" value="#{bundle['lista_produktow.cenabrutto']}"  />
<h:outputText  id="brutto"
               value="#{managed_produkt.cena_brutto}" />
```

Pierwszy sposób rejestrowania konwerterów (SR1)

Fragment pliku jsf, **służący do wyświetlania zbioru danych w formie tabeli**, korzystający z komponentu **h:dataTable**, gdzie atrybut **value="#{managed_produkt.items}"** oraz **var="item"**, gdzie **item** jest obiektem typu **Produkt_dto**

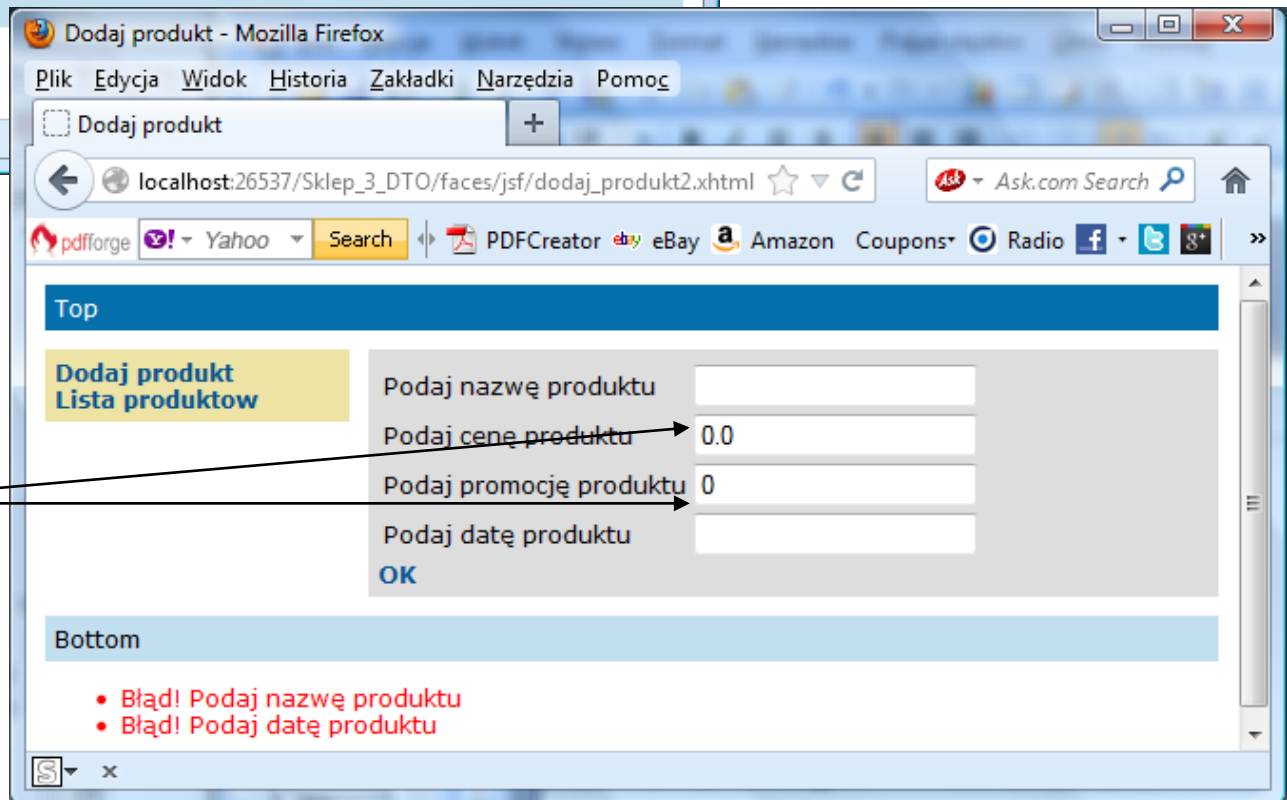
```
<h:column>
  <f:facet name="header"> <h:outputText value="#{bundle['lista_produktow.id']}"/>  </f:facet>
  <h:outputText  value="#{item.id}"/>
</h:column>
<h:column>
   <f:facet name="header"> <h:outputText value="#{bundle['lista_produktow.cena']}"/> </f:facet>
   <h:outputText  value="#{item.cena}"/>
 </h:column>
 <h:column>
    <f:facet name="header"><h:outputText value="#{bundle[‘lista_produktow.promocja']}"/></f:facet>
     <h:outputText  value="#{item.promocja}"/>
 </h:column>
 <h:column>
   <f:facet name="header"> <h:outputText value="#{bundle['lista_produktow.data']}"/></f:facet>
    <h:outputText  value="#{item.data_produkcji}"/>
 </h:column>
 <h:column>
      <f:facet name="header"> <h:outputText value="#{bundle['lista_produktow.data']}"/></f:facet>
       <h:outputText  value="#{item.cena_brutto}"/>
  </h:column>
```



Sposób 1

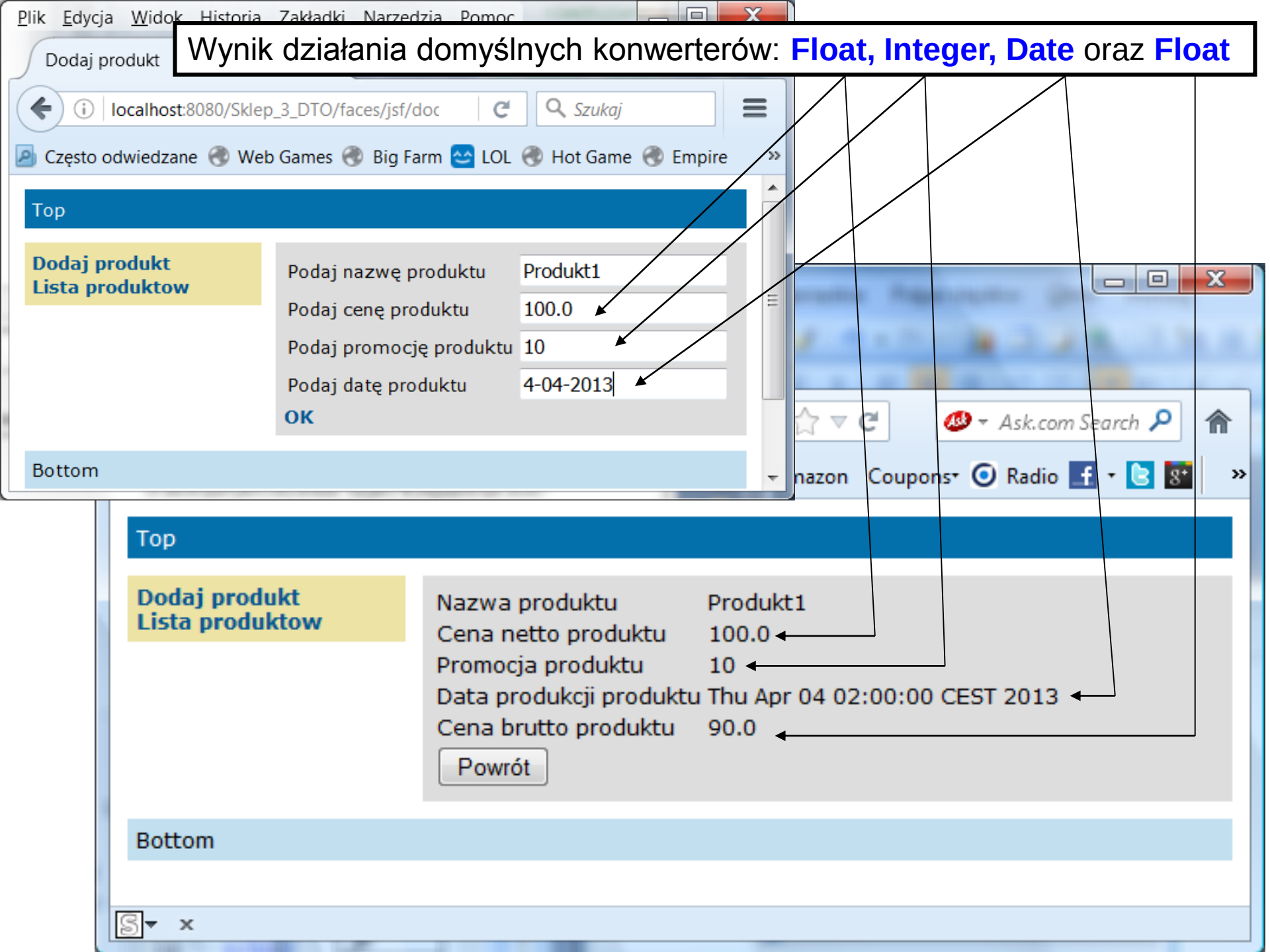
Sposób 1

Sposób 2

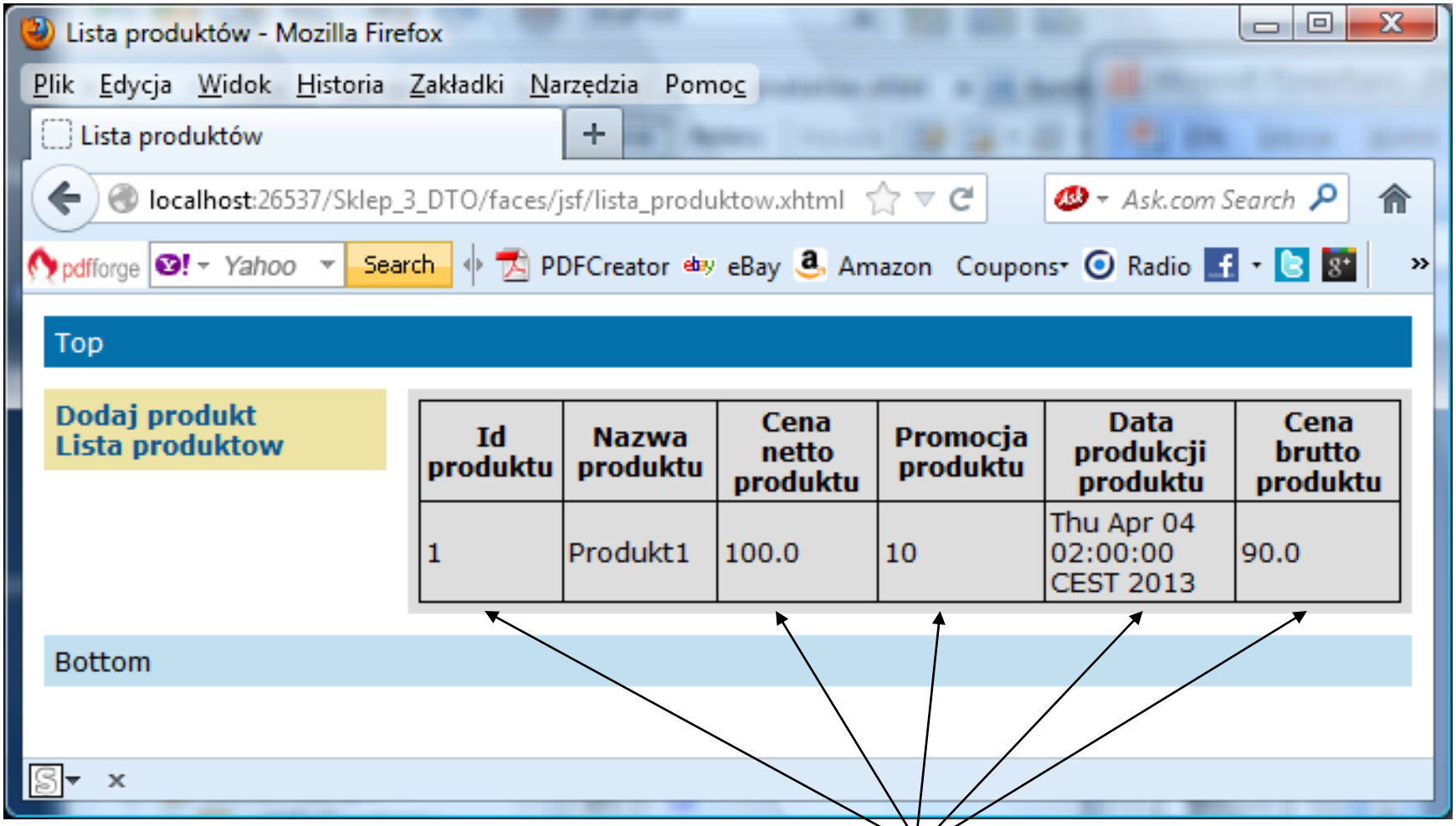


Widok strony po kliknięciu na przycisk **OK.**
Efekt uzyskany dzięki domyślnym wartościom pól wejściowych po zastosowaniu domyślnych konwerterów typu **Float i Integer** oraz polom **required i requiredMessage** w pozostałych polach wejściowych

Wynik działania domyślnych konwerterów: **Float, Integer, Date** oraz **Float**



Dane po konwersji w komponencie typu **dataTable**



Wynik działania domyślnych konwerterów:
Long, Float, Integer, Date oraz **Float**

Przykład 2

Drugi sposób rejestrowania konwerterów (SR2)

(P1) Wyświetlanie danych - Konwerter typu **convertNumber** z atrybutem **pattern** określającym sposób prezentowania wartości i oznaczeń wartości konwertowanej (zł w kodzie UTF-8)

```
<h:column>
    <f:facet name="header">  <h:outputText value="#{bundle['lista_produktow.cena']}"/> </f:facet>
      <h:outputText value="#{item.cena}">
            <f:convertNumber  pattern="####.## z&#322;"/>
      </h:outputText>
</h:column>
```

(P2) Wyświetlanie danych - Konwerter typu **convertNumber** z atrybutami **currencySymbol** (%) i **type** do specyfikowania własnego typu (programisty) i oznaczeń wartości konwertowanej

```
<h:column>
    <f:facet name="header">  <h:outputText value="#{bundle['lista_produktow.promocja']}"/>    </f:facet>
      <h:outputText value="#{item.promocja}">
            <f:convertNumber currencySymbol="%" type="currency"/>
      </h:outputText>
</h:column>
```

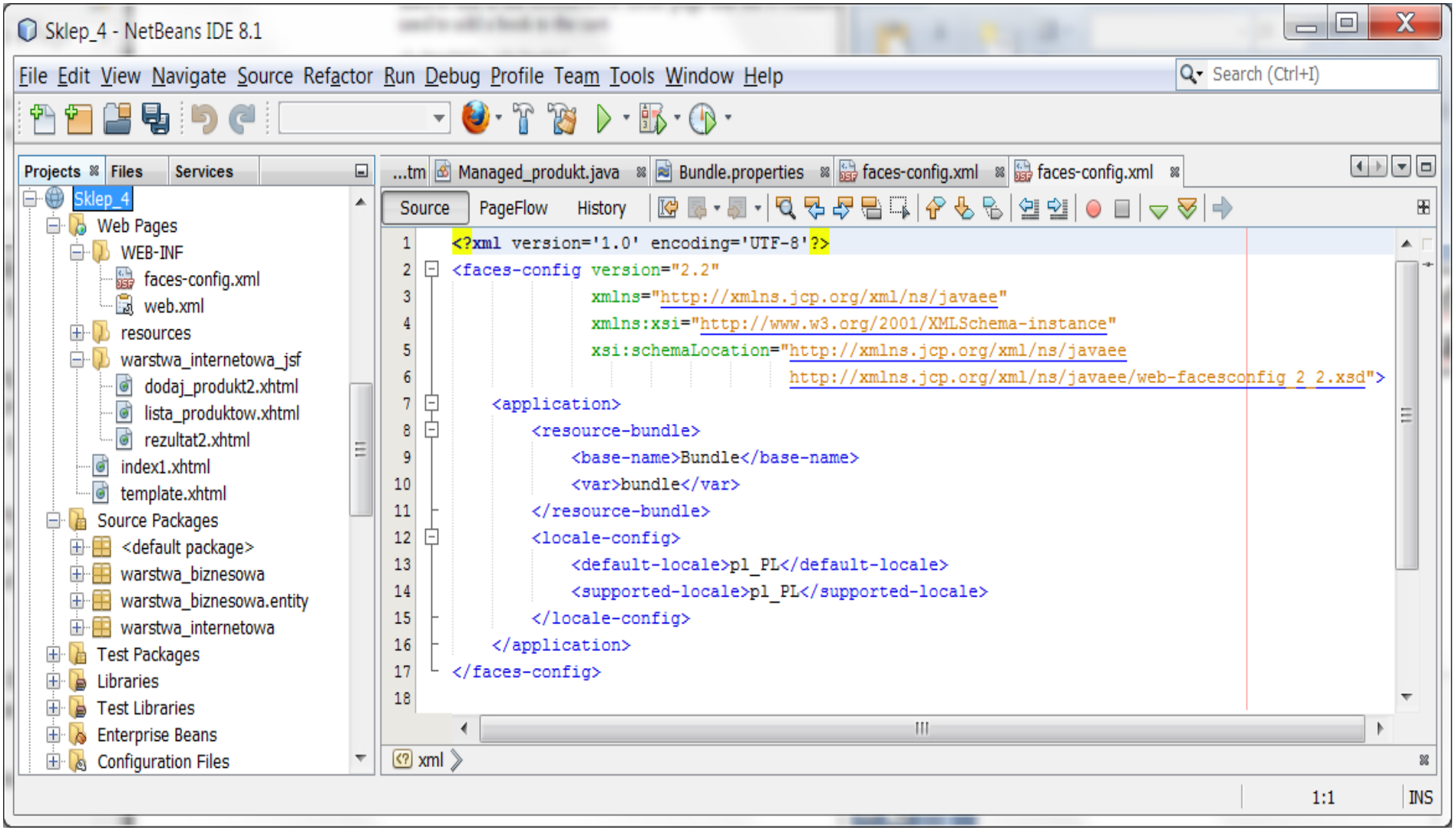
(P3) Wyświetlanie danych - Konwerter typu **convertNumber** z atrybutami **currencySymbol** (zł w kodzie UTF-8) i **type** do specyfikowania własnego typu i oznaczeń wartości konwertowanej **(SR2)**

```
<h:column>
  <f:facet name="header">  <h:outputText value="#{bundle['jsf.lista_produktow.cenabrutto']}"/>  </f:facet>
   <h:outputText value="#{item.cena_brutto}">
      <f:convertNumber currencySymbol="z&#322;" type="currency"/>
   </h:outputText>
 </h:column>
```

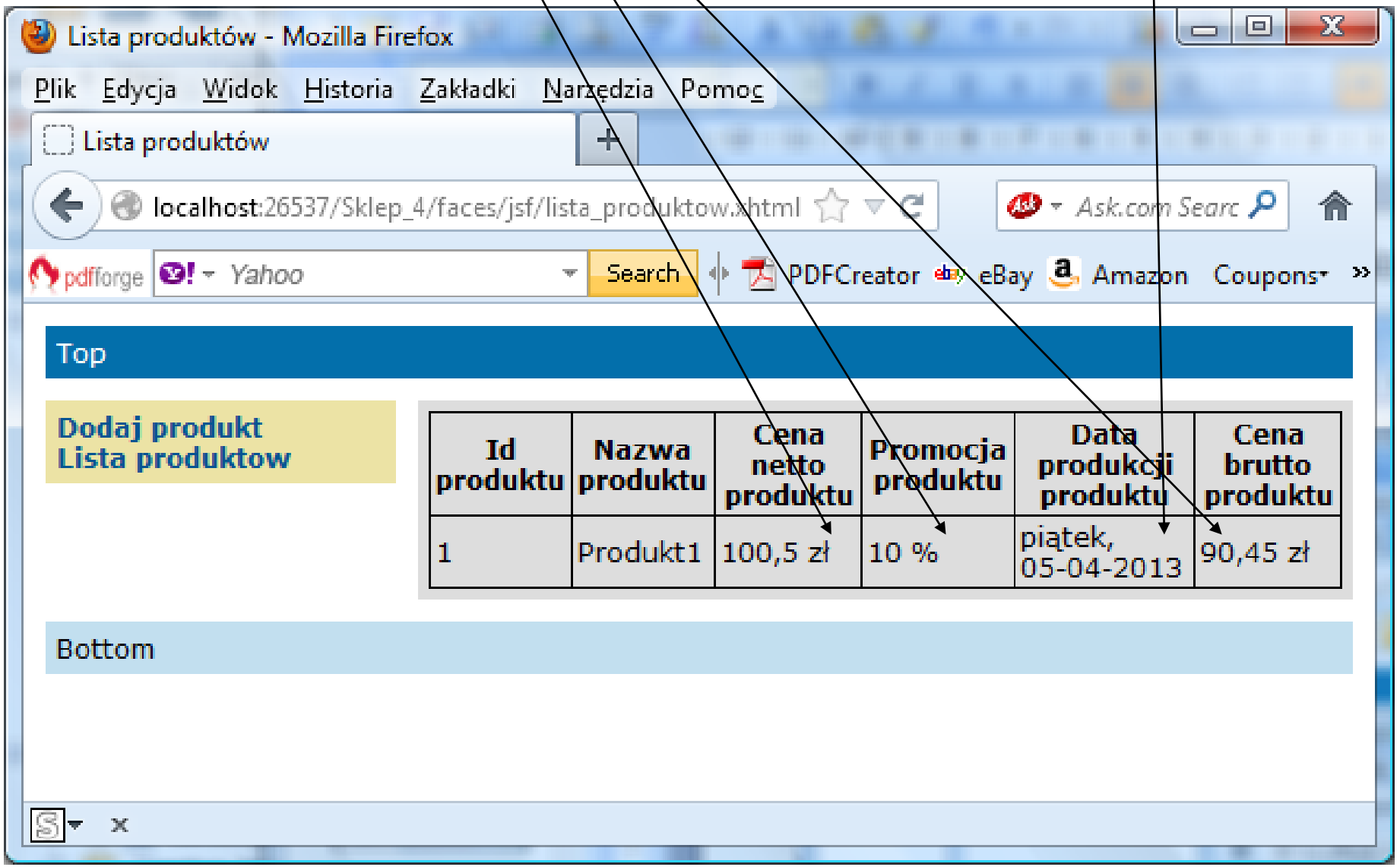
(P4) Wyświetlanie danych - Konwerter **convertDateTime**, który przekształca datę na **nazwa_dnia, numer_dnia-numer_miesiąca-rok** za pomocą atrybutu **pattern**. Nazwa dnia podawana jest w języku strefy czasowej podawanej przez **javax.faces.context.FacesContext.getLocale**, jeśli nie został zdefiniowany atrybut **locale**. **(SR2)**

```
<h:column>
  <f:facet name="header"> <h:outputText value="#{bundle['lista_produktow.data']}"/> </f:facet>
    <h:outputText value="#{item.data_produkcji}">
       <f:convertDateTime  pattern="EEEEEEEE, dd-MM-yyyy" />
    </h:outputText>
</h:column>
```

Zawartość pliku typu faces-config.xml, określająca domyślną wartość atrybutu **locale** aplikacji (**pl_PL**)



Działanie konwerterów typu **convertNumber (P1), (P2), (P3)** oraz **convertDateTime (P4) – pl_PL**



Przykład 3

Trzeci (SR3) i czwarty (SR4) sposób rejestrowania konwerterów

Należy dodać konwerter typu **NumberConverter**, który pozwala na wprowadzanie danych typu liczba rzeczywista, gdzie część ułamkowa jest oddzielona **przecinkiem** od części całkowitej liczby (**pl_PL**). Podczas wprowadzania dodawany jest symbol wartości **zł**

```
@ManagedBean
@RequestScoped
public class Managed_produkt {

    @EJB
    private Fasada_warstwy_biznesowej fasada;
    private DataModel items;
    private int stan = 1;
    private Produkt_dto produkt_dto = new Produkt_dto();
    private NumberConverter number_convert=new NumberConverter();

    public NumberConverter getNumber_convert() {
        this.number_convert.setPattern("######.## zł");
        return number_convert;
    }

    public void setNumber_convert(NumberConverter Number_convert) {
        this.number_convert = Number_convert;
    }
```

(P5) Wprowadzanie danych - Konwerter typu **converter** z atrybutem **pattern** określającym sposób wprowadzania danych ceny (w kodzie konwertera)**. (SR4)**

```
<h:outputLabel for="cena"  value="#{bundle['dodaj_produkt2.cena']}" />
<h:inputText
                    id="cena"       title="#{bundle['dodaj_produkt2.cena1']}"
    value="#{managed_produkt.cena}"
    required="true"  requiredMessage="#{bundle['dodaj_produkt2.blad_cena']}" >
      <f:converter  binding="#{managed_produkt.number_convert}"/>
</h:inputText>
```

(P6) Wprowadzanie danych - Konwerter typu **convertDateTime** z atrybutem **pattern** określającym sposób wprowadzania daty: **numer_dnia-numer_miesiaca-rok. (SR2)**

```
<h:outputLabel for="data" value="#{bundle['dodaj_produkt2.podaj_data']}" />
<h:inputText
                    id="data"     title="#{bundle['dodaj_produkt2.podaj_data']}"
  value="#{managed_produkt.data_produkcji}"
  required="true"  requiredMessage="#{bundle['dodaj_produkt2.podaj_data_blad']}" >
        <f:convertDate Time pattern="dd-MM-yyyy" />
</h:inputText>
```

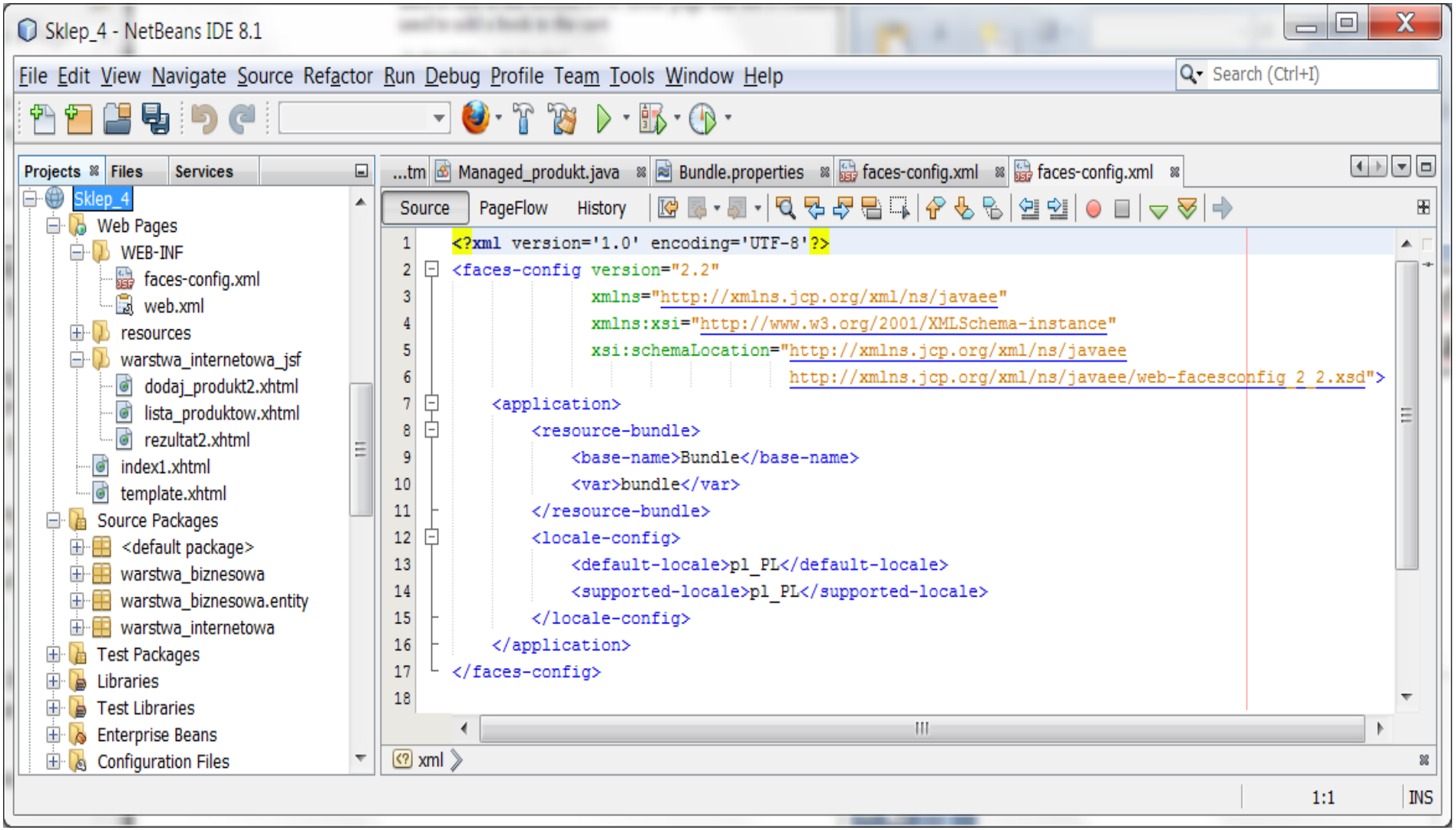
(P7) Wyświetlanie danych - Konwerter typu **convertDateTime** z atrybutami **dateStyle**, **locale**, **timeStyle** oraz **type**, gdzie dla wybranych wartości atrybutów uzyskano następującą postać wyświetlanej daty: np. **Saturday, February 2, 2013 12:00:00 AM GMT (SR2)**

```
<h:outputLabel value="#{bundle['lista_produktow.data']}"
                    for="data"/>
<h:outputText id="data" value="#{managed_produkt.data_produkcji}">
        <f:convertDateTime dateStyle="full"
                locale="en_US"  timeStyle="long" type="both"/>
</h:outputText>
```

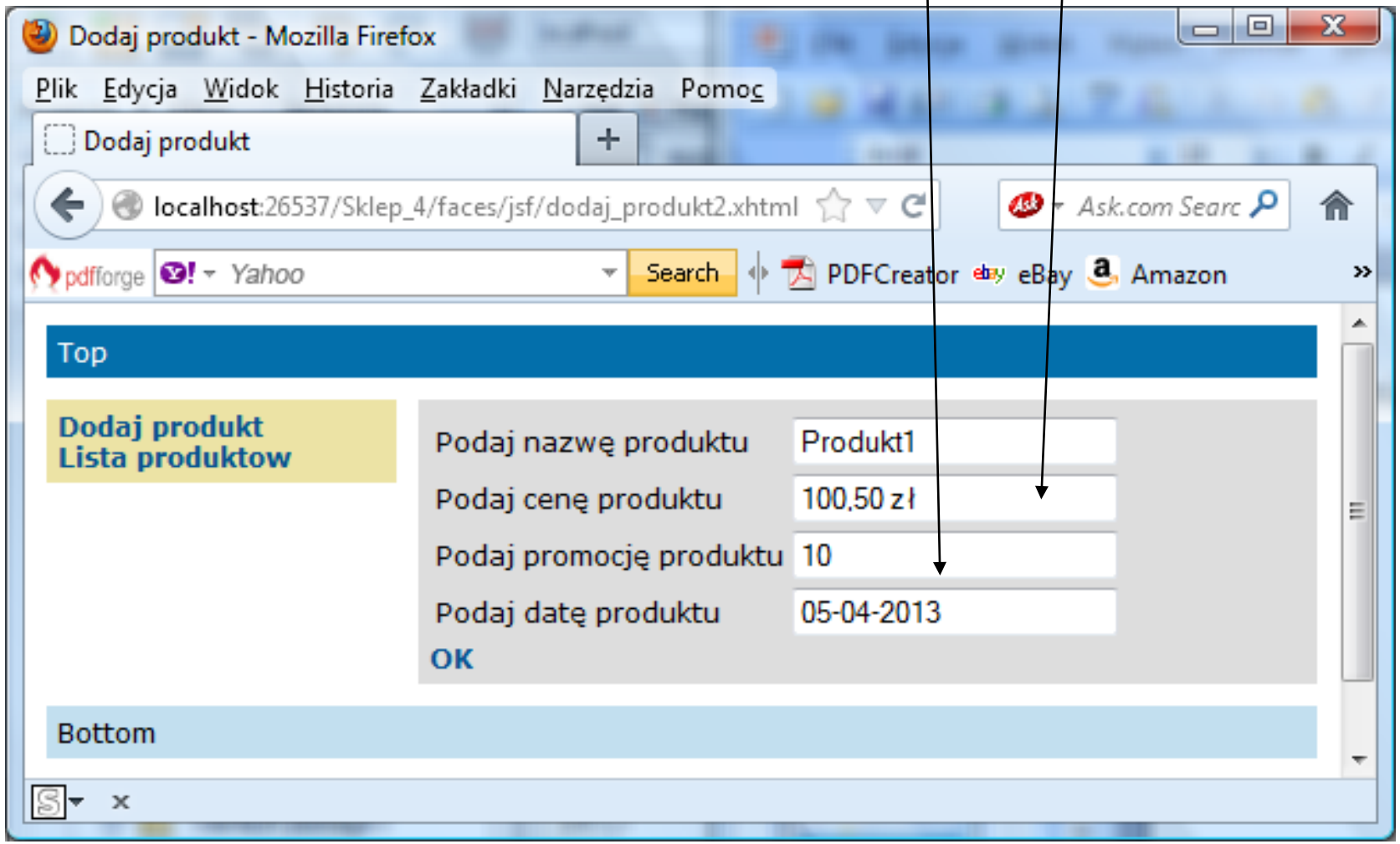
(P8) Wyświetlanie danych - Konwerter typu **convertNumber** z przykładu **(P5)** z atrybutami **binding** oraz **pattern**, gdzie dla wybranych wartości atrybutów uzyskano następującą postać wyświetlanej ceny: np. **120,5 zł –** lub **120.5 zł - (SR4)**

```
<h:outputLabel   for="cena" value="#{bundle['lista_produktow.cena']}"  />
 <h:outputText  id="cena" value="#{managed_produkt.cena}">
     <f:convertNumber  binding="#{managed_produkt.number_convert}"
                       pattern="####.### z&#322; -"/>
</h:outputText>
```

Zawartość pliku typu faces-config.xml, określająca domyślną wartość atrybutu **locale** aplikacji (**pl_PL**)

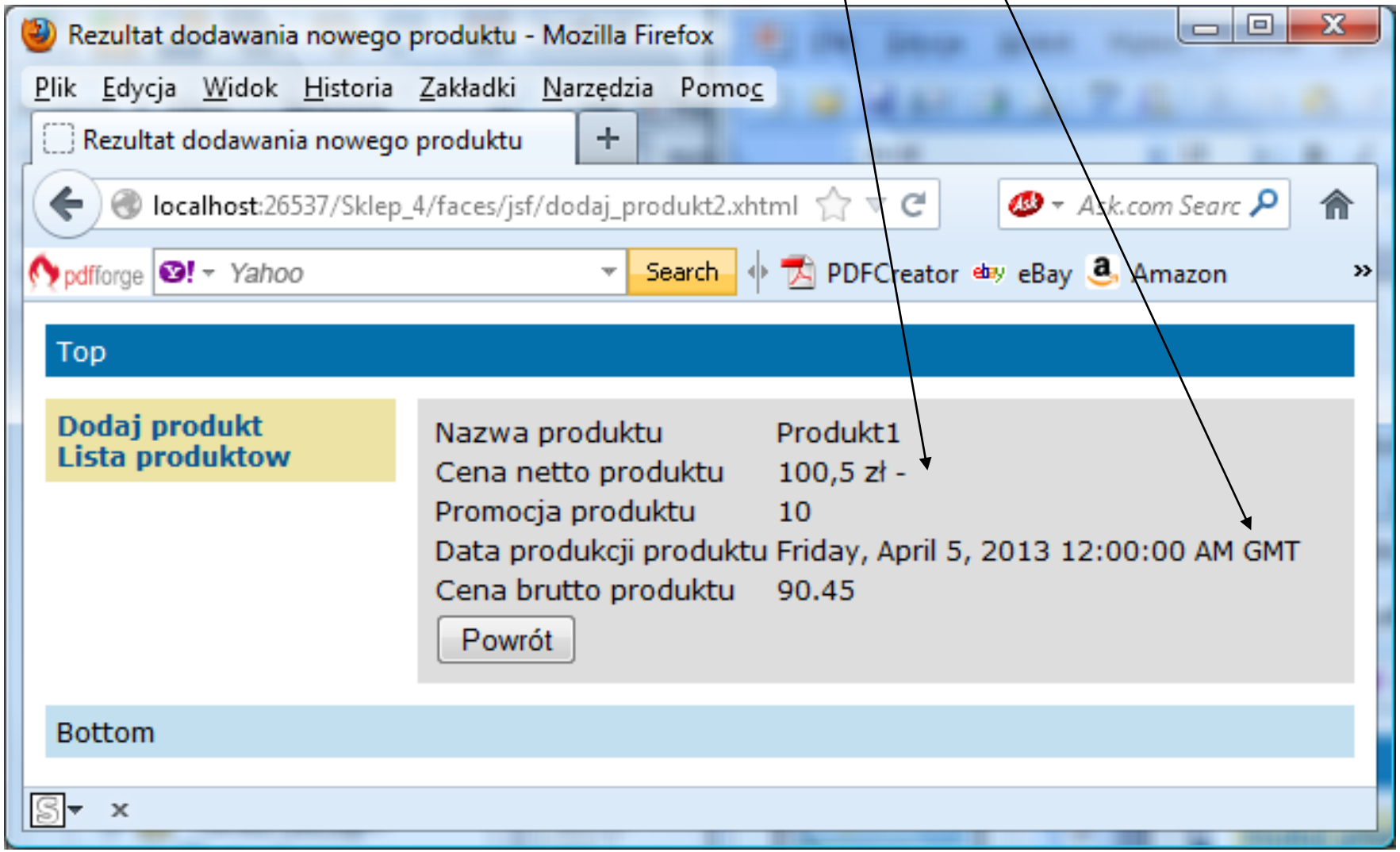


Działanie konwertera typu **NumberConverter (P5)** oraz konwertera typu **convertDateTime (P6)**



Działanie konwertera typu **convertDateTime (P7)** w wersji narzuconej **en_US**

oraz konwertera typu **NumberConverter (P8)** w wersji domyślnej **pl_PL**

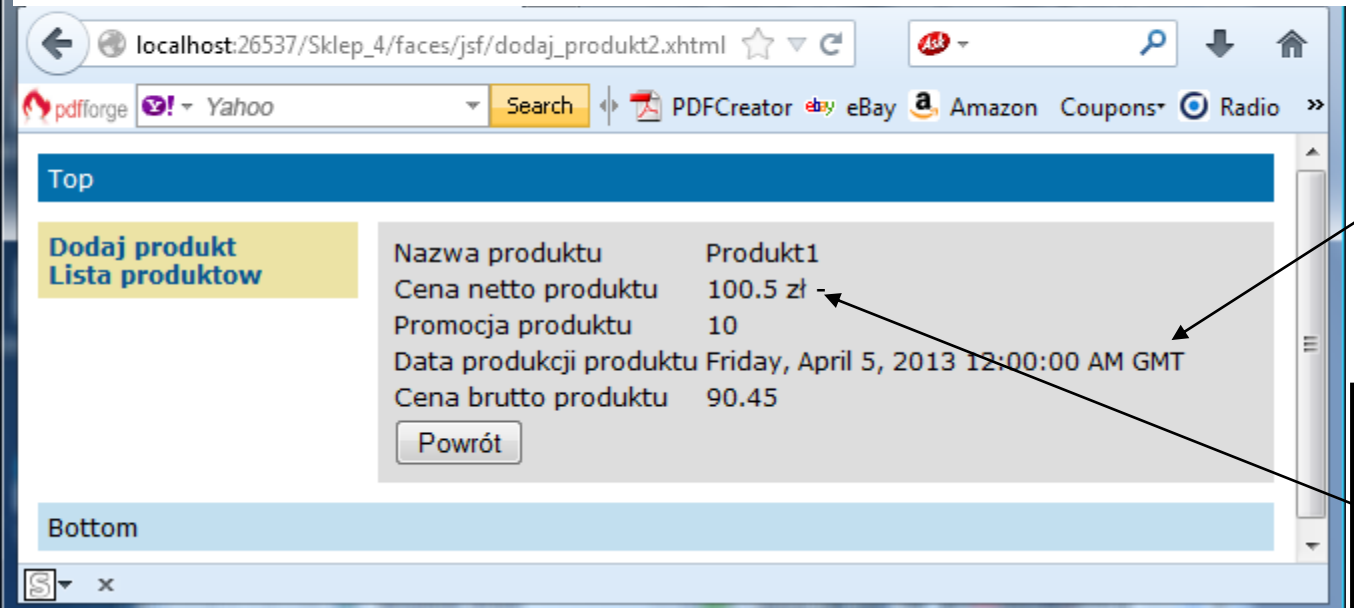


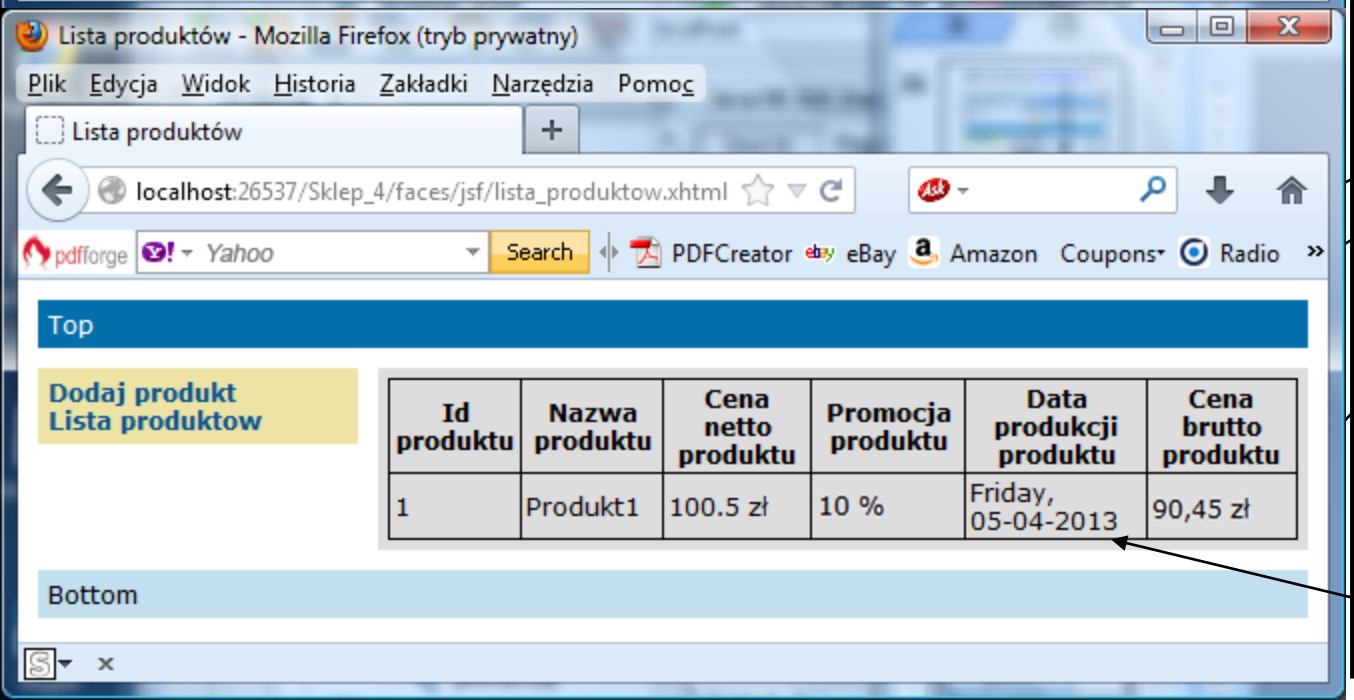
Zawartość pliku typu **faces-config.xml** , określająca domyślną wartość **locale** aplikacji (**en_US**)

```
<?xml version='1.0' encoding='UTF-8'?>
<!-- ======= FULL CONFIGURATION FILE ========================== -->
<faces-config version="2.1"
              xmlns="http://java.sun.com/xml/ns/javaee"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                                  http://java.sun.com/xml/ns/javaee/web-facesconfig_2_1.xsd">

    <application>
        <resource-bundle>
            <base-name>/Bundle</base-name>
            <var>bundle</var>
        </resource-bundle>
        <locale-config>
            <default-locale>en_US</default-locale>
            <supported-locale>pl_PL</supported-locale>
        </locale-config>
    </application>
</faces-config>
```

Efekt zmiany formatowania domyślnego zdefiniowanego za pomocą znacznika **default-locale** na **en_US** w pliku **faces-config.xml**





Atrybut **locale** ustawiony indywidulanie w **convertDateTime (P7)**

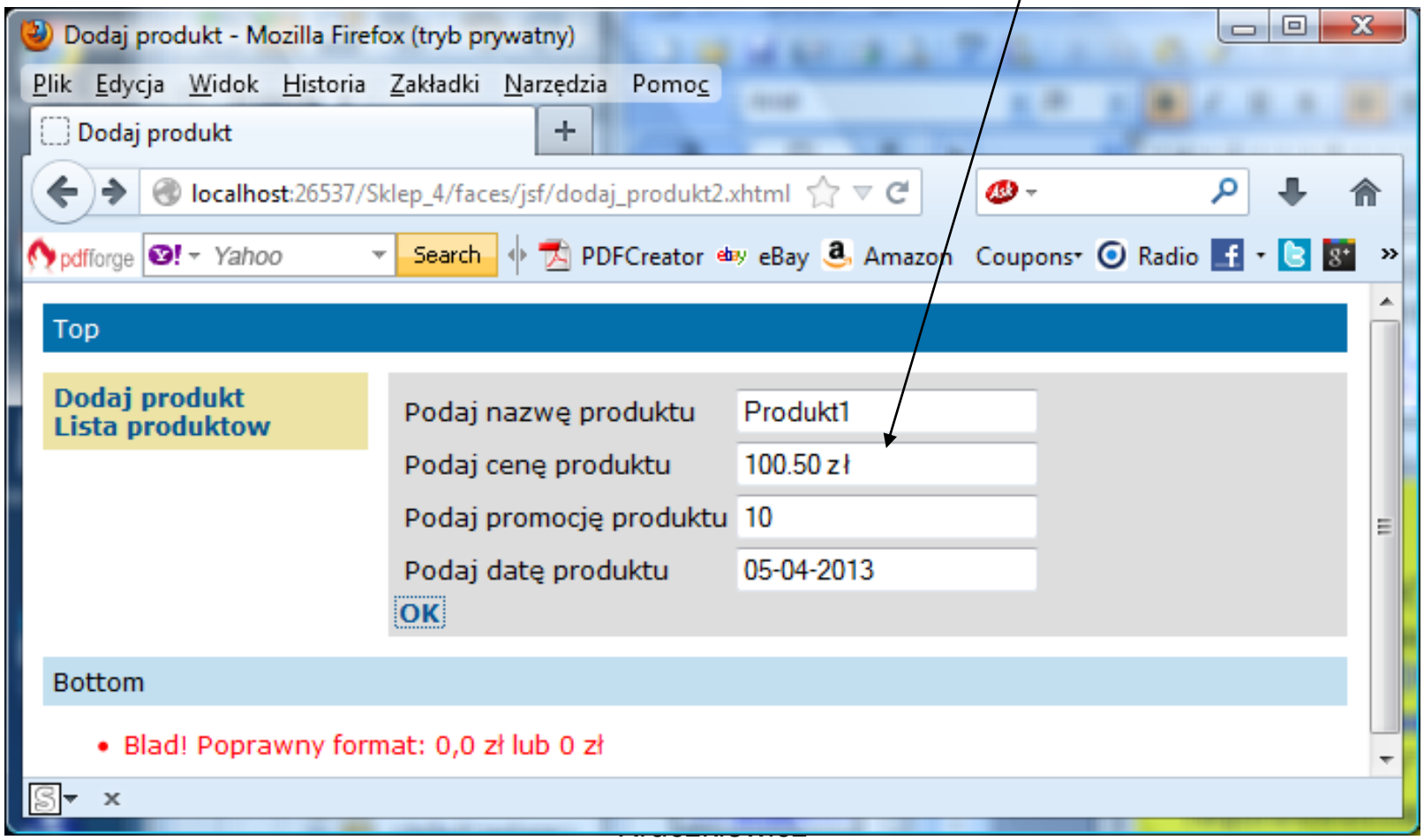
Działanie konwerterów typu **convertNumber (P8), (P1), (P2) (P3)**

oraz **convertDateTime (P4)**

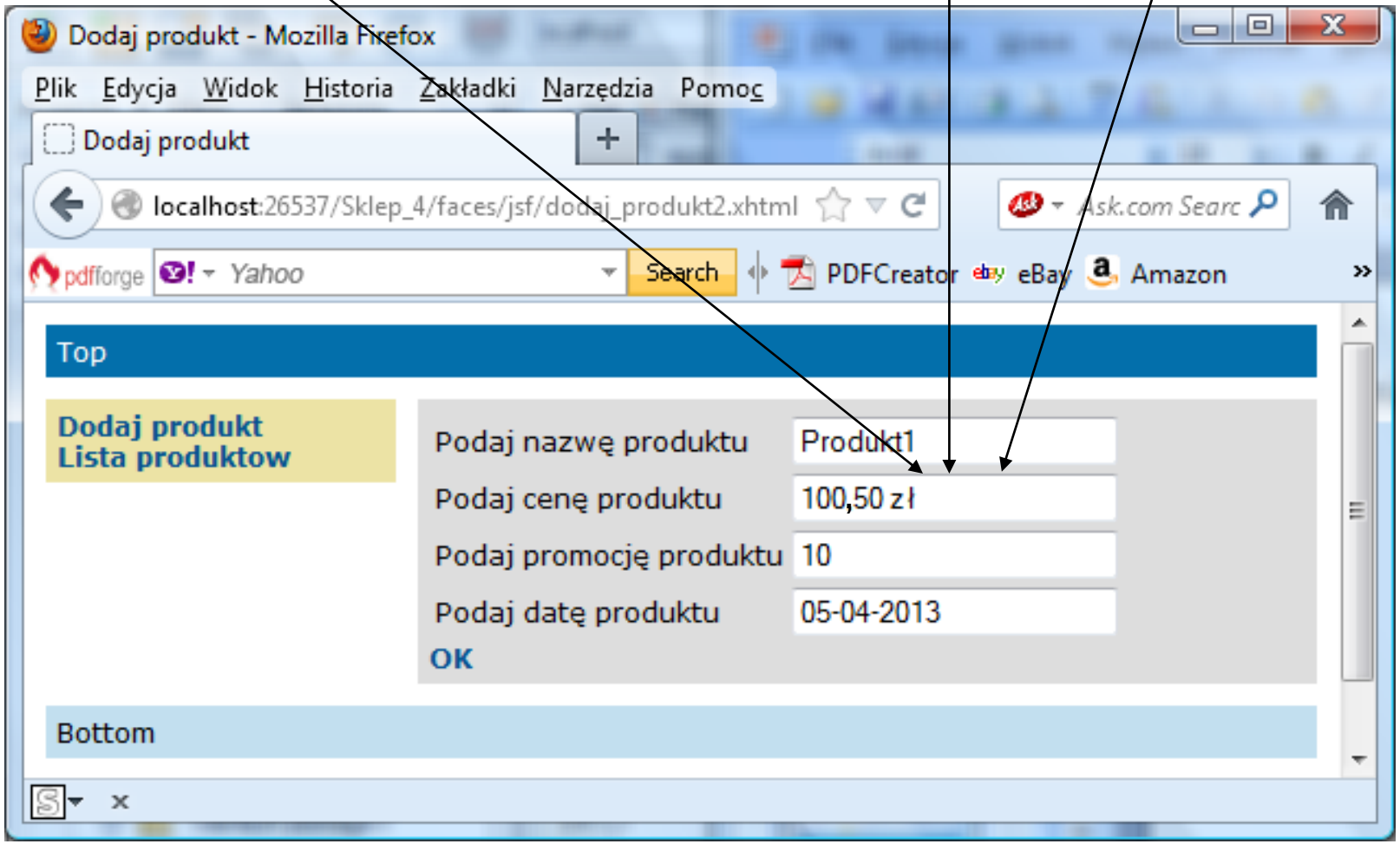
(P9) Wprowadzanie danych - Komponent typu **h:inputText** z atrybutem **converter,** odwołującym się do konwertera typu **NumberConverter** zdefiniowanego w klasie obiektu typu **Managed_produkt,** gdzie nadano wartość jego atrybutu **pattern** metodą **setPattern.** Widok taki sam, jak dla konwertera **(P5) - (SR3)**

```
<h:outputLabel value="#{bundle['dodaj_produkt2.cena']}"
                    for="cena" />
<h:inputText
    id="cena"
    title="#{bundle['dodaj_produkt2.podaj_cena']}"
    value="#{managed_produkt.cena}"
    converter="#{managed_produkt.number_convert}"
    converterMessage="Blad! Poprawny format: 0,0 zł lub 0 zł"
    required="true"
    requiredMessage="#{bundle['dodaj_produkt2.podaj_cena_blad']}" >
</h:inputText>
```

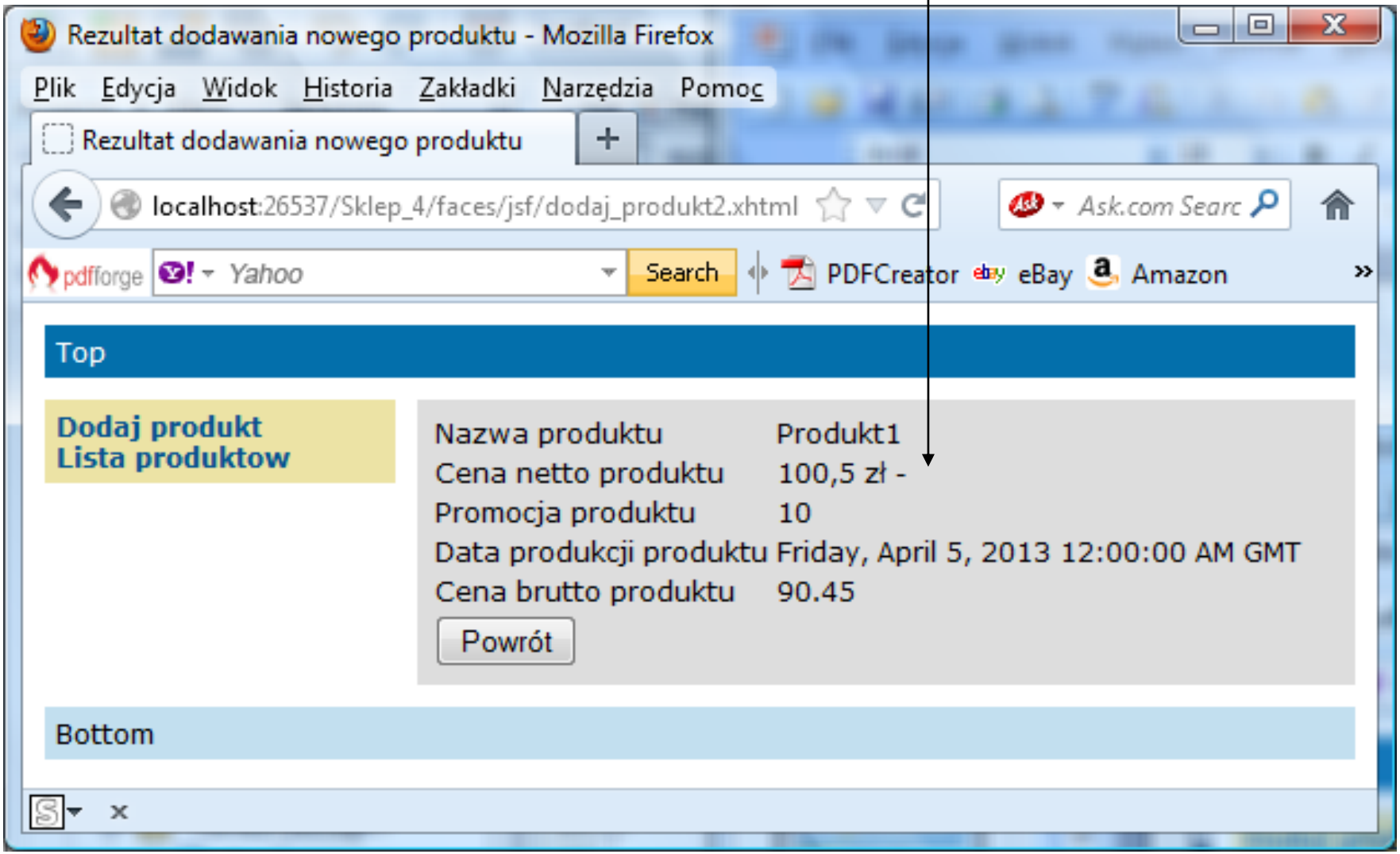
Efekt zastosowania definicji konwertera wg **3-go sposobu (SR3)** definiowania konwertera wartości liczbowych za pomoca atrybutu **converter** komponentu **(zastosowano znacznik <default-locale>pl_PL</default-locale>** w pliku **favces-congig.xml)** – należało część ułamkową oddzielić przecinkiem od części całkowitej. Komunikat o błędzie jest realizowany za pomocą atrybutu **converterMessage** komponentu **typu h:inputText**



Działanie konwertera typu **NumberConverter,** zdefiniowanego w kodzie komponentu typu **Managed_produkt**, zbindowanego **(SR4)** ze znacznikiem typu **f:convertNumber** lub **f:converter (P5)** lub atrybutem **(SR3) converter (P9) –pl_PL**



Działanie konwertera typu **NumberConverter (P8)**, zdefiniowanego w kodzie komponentu typu **Managed_produkt** zbindowanego **(SR4)** ze znacznikiem **f:convertNumber – pl_PL**



Linki do materiałów zawierających informację o formatowaniu

- http://docs.oracle.com/javase/tutorial/i18n/format/simpleDateFormat.html
- http://docs.oracle.com/javase/tutorial/i18n/format/decimalFormat.html

Obsługa zdarzeń

Rejestracja słuchaczy zdarzeń typu valueChangeListener w komponentach

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

2. **Słuchacz zdarzeń może być instancją zdefiniowanej klasy** – wtedy znaczniki **f:valueChangeListener** (**wykład 3, Przykłady 5 i 6, slajdy 40-47;** http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/ti_/TINT_3.pdf)

Rejestracja słuchaczy zdarzeń typu **Value-Change** w komponentach (drugi sposób)

f:valueChangeListener

Atrybuty:

type - **wskazanie na nazwę pakietową klasy (Przykład 6, wyk.3)**, 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 (Przykład 5, wyk.3)**, 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)

Rejestracja słuchaczy zdarzeń typu actionListener w komponentach

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

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

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 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**

// javax.faces.event.ActionEvent;

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

Przykład 2 (drugi sposób) **przy wprowadzaniu danych** – obsługa zdarzenia typu **actionListener** podczas kliknięcia na komponent typu **h:commandLink** (na stronie **dodaj_produkt2**)

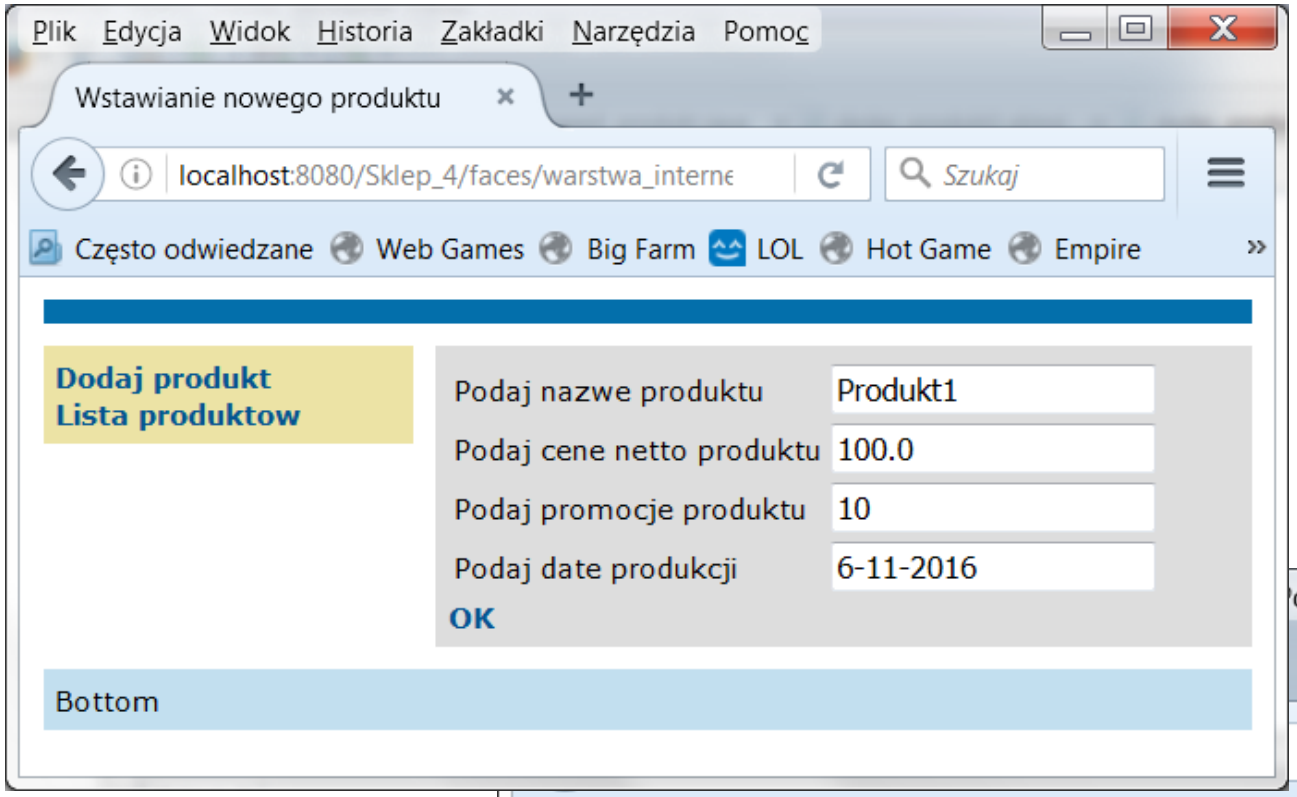
Fragment strony **dodaj_produkt2.xhtml:**

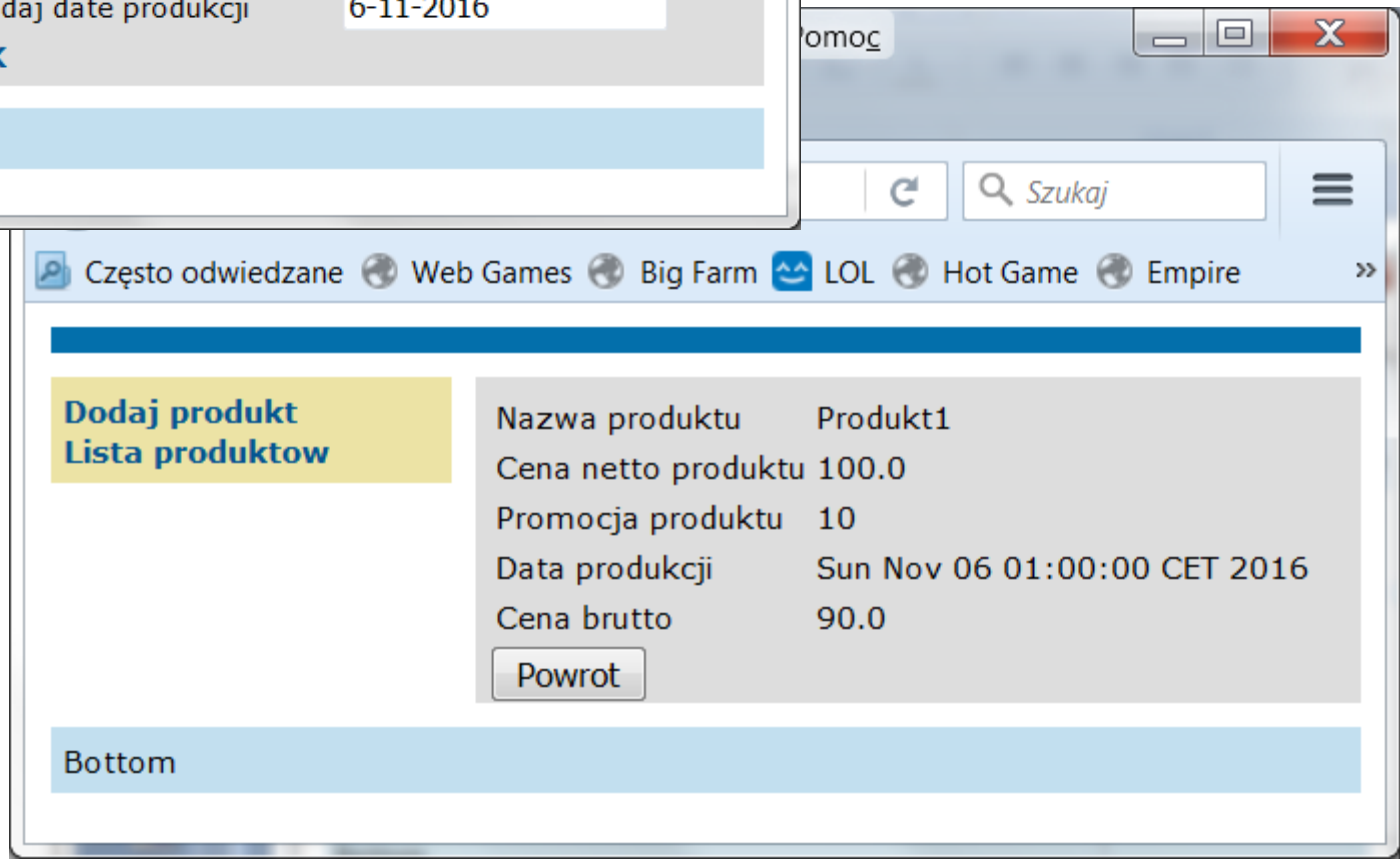
```
<h:commandLink  action="rezultat2" value="OK" >
    <f:actionListener  binding="#{managed_produkt}"/>
</h:commandLink>
```

Przykład 2 (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(ActionEvent event) {
    String[] dane = {"" + nazwa, "" + cena, "" + promocja};
    fasada.utworz_produkt(dane, data_produkcji);
    dane_produktu();
  // return "rezultat2";
  }
```

Przykłady 1 i 2 (cd)
Wynik działania w przypadku zastosowania obu sposobów rejestrowania obsługi zdarzeń typu **actionListener**



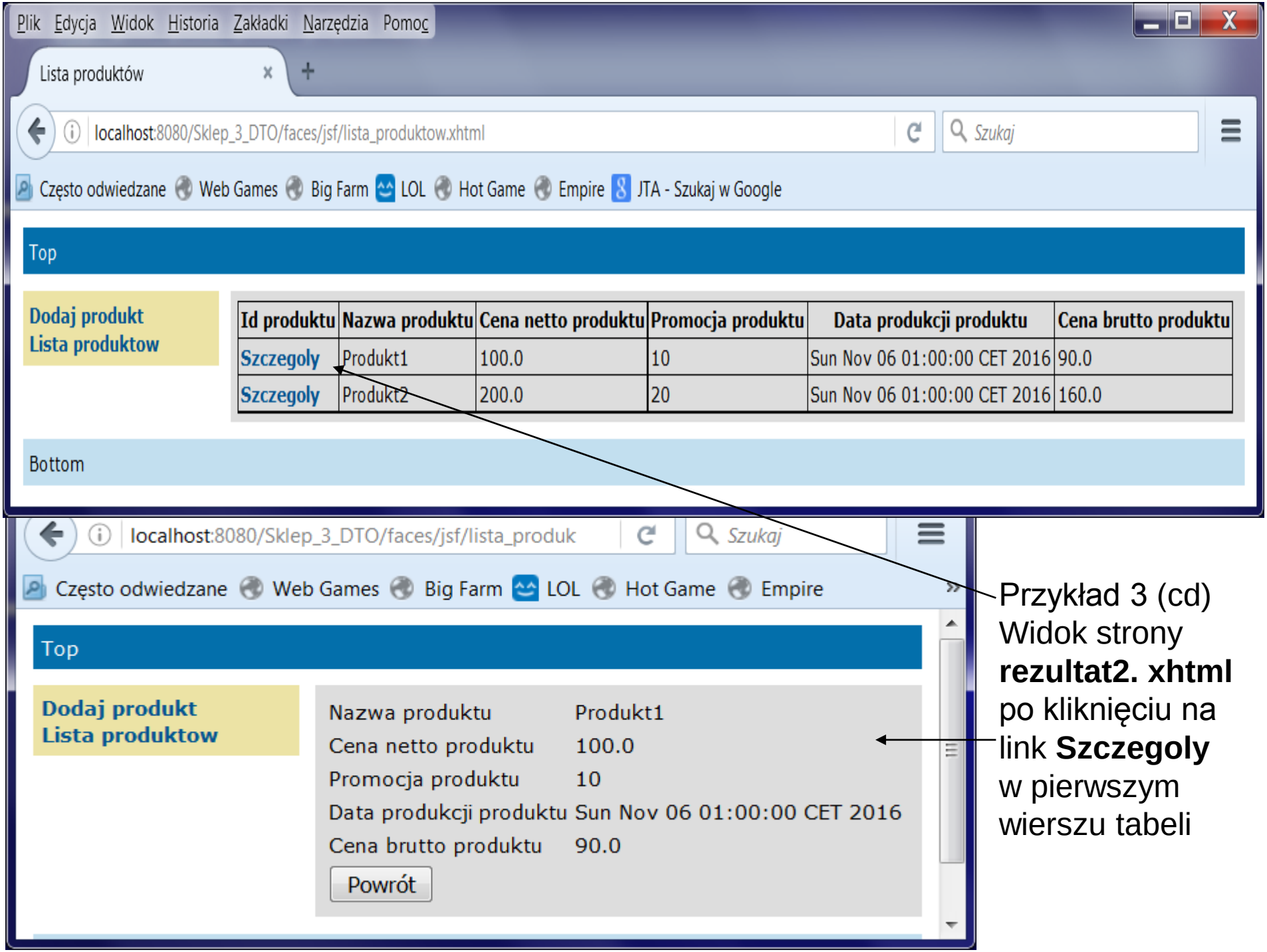


Przykład 3 – obsługa zdarzenia za pomocą słuchacza zdarzeń, zarejestrowanego za pomocą znacznika **f:setPropertyActionListener**

```
<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 3 (cd)
Widok strony **rezultat2. xhtml** po kliknięciu na link **Szczegoly** w pierwszym wierszu tabeli

Użycie standardowych walidatorów

Wykaz standardowych walidatorów implementujących interfejs **javax.faces.validator.Validator**

Klasa walidatora	Znacznik	Funkcja
BeanValidator	validateBean	Rejestruje walidator w komponencie
DoubleRangeValidator	validateDoubleRange	Sprawdza, czy wartość komponentu zawiera się w podanym przedziale. Wartość musi być reprezentowana w systemie zmiennopozycyjnym
LengthValidator	validateLength	Sprawdza, czy długość wartości komponentu zawiera się w podanym przedziale. Wartość musi być typu java.langString

Klasa walidatora	Znacznik	Funkcja
LongRangeValidator	validateLongRange	Sprawdza, czy wartość komponentu zawiera się w podanym przedziale. Wartość może być numeryczna lub typu java.langString, który można przekształcić na typ long
RegexValidator	validateRegEx	Sprawdza, czy wartość komponentu spełnia łańuch regularny typu java.util.regex
RequiredValidator	validateRequired	Sprawdza, czy wartość komponentu typu javax.faces.component.EditableValueHolder nie jest pusta

Typy komunikatów wyświetlanych przez standardowe walidatory

{1}: Validation Error: Value is greater than allowable maximum of "{0}„

gdzie:

{1} – jest zastępowane przez etykietę lub id komponentu

{0} – jest zastępowane przez wartość graniczną, używaną podczas walidacji

Do wyświetlania komunikatów obłędach można wykorzystać znaczniki typu **h:Message** oraz **h:Messages**

Uwaga:

Do walidacji można użyć walidatorów typu **Bean Validation**

Rejestracja walidatorów wartości komponentów

1. Należy zarejestrować walidator za pomocą znaczników **<f:>**
2. Należy odwołać się metody zdefiniowanej w komponencie typu Managed Bean, która przeprowadza walidację za pomocą atrybutu **validator** komponentu (**wykład 3, Przykład 3, slajdy 32-35;** http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/ti_/TINT_3.pdf)
3. Należy umieścić znacznik walidatora **<f:>** wewnątrz znacznika komponentu i użyć albo atrybut **validatorId** walidatora (podobnie używany jak atrybut **converterId** konwerterów) lub jego atrybut **binding** wskazujący na instancję walidatora.

Uwaga:

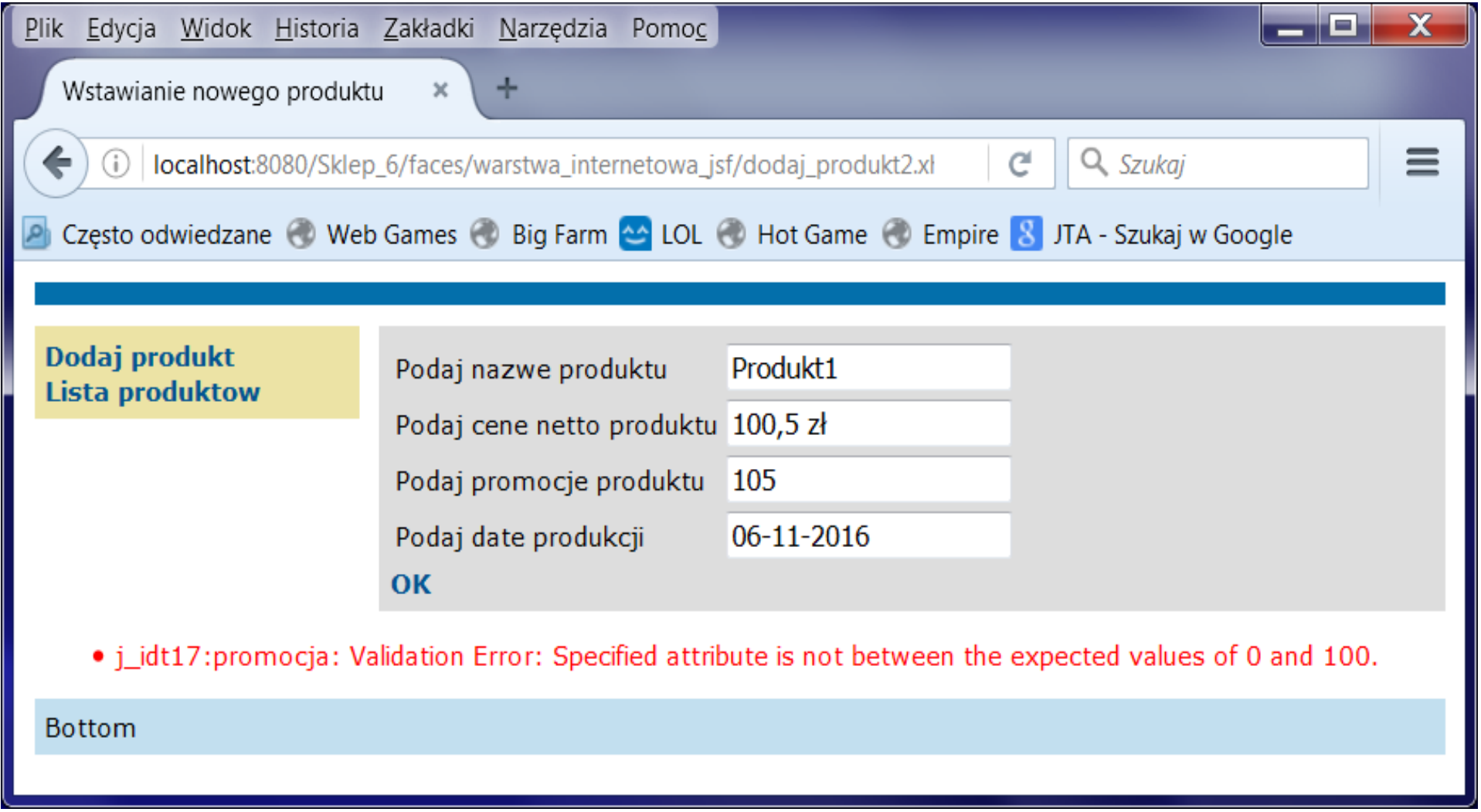
Standardowe walidatory działają jedynie w komponentach, które implementują **EditableValueHolder**.

Przykład 1 - Pierwszy sposób przy wprowadzaniu danych –
zastosowanie walidatora **LongRangeValidator**

```
<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.Long"  />
      <f:validateLongRange minimum="#{managed_produkt.min}"
                           maximum="#{managed_produkt.max}"/>
</h:inputText>
```

```
@Named(value = "managed_produkt")
@RequestScoped
public class Managed_produkt {
   public int getMin()   {   return 0;    }
   public int getMax()   {   return 100; }
```

Wynik działania walidatora



Przykład 2 - Drugi sposób przy wprowadzaniu danych – atrybut validator

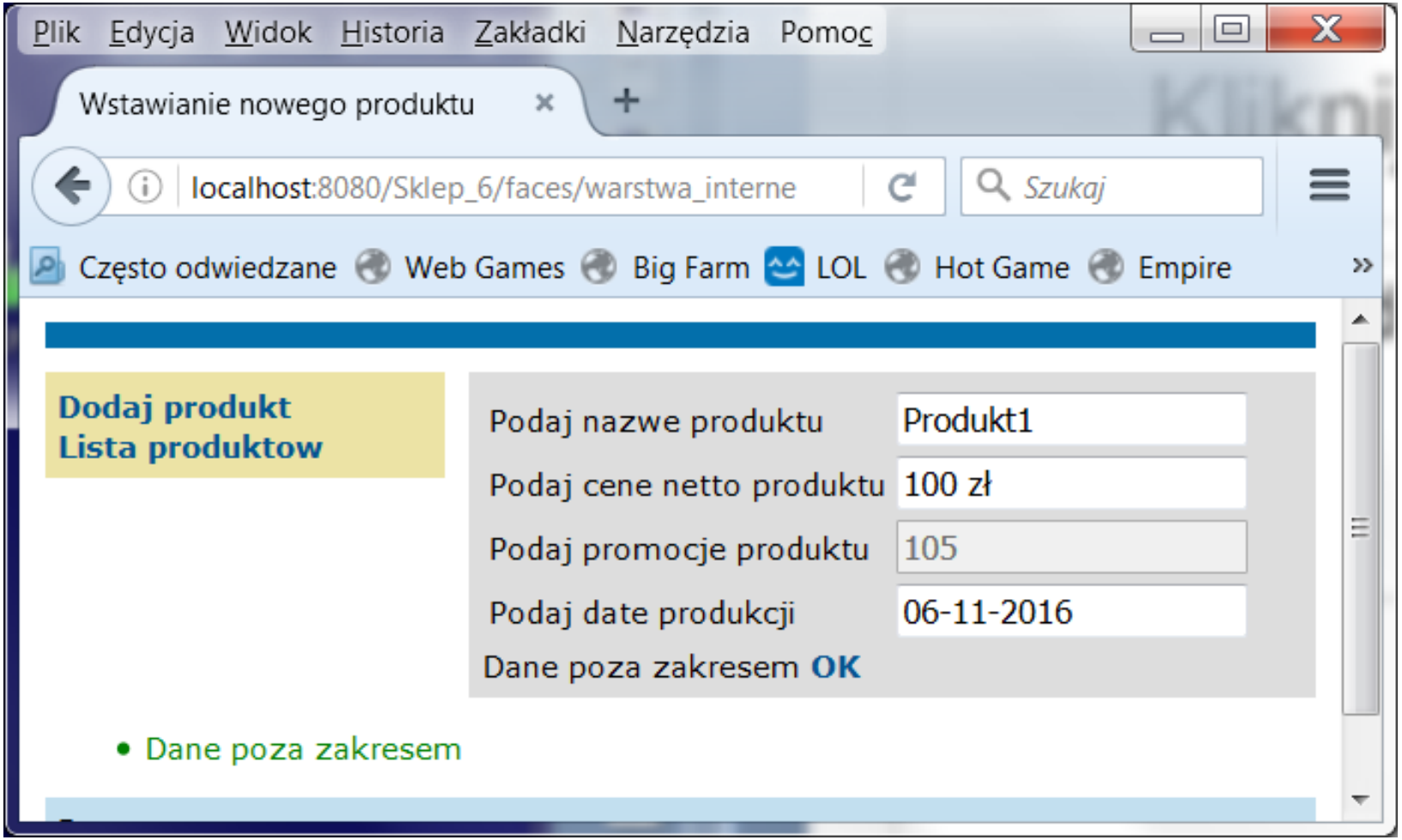
Wykorzystanie atrybutu **disabled** oraz **validator** w znaczniku **<h:inputText**

```
<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']}"
          disabled="#{managed_produkt.stan==0}"
          validator = "#{managed_produkt.zakrespromocji}">
            <f:converter converterId="javax.faces.Integer"  />
   </h:inputText>
```

Definicja metody do walidacji wartości promocji w komponencie **Managed_produkt**

```
public void zakrespromocji(FacesContext context,
                                   UIComponent toValidate, Object value) {
    stan = 1;
    int input = (Integer) value;
    if (input < getMin() || input > getMax()) {
       ((UIInput) toValidate).setValid(false);
       FacesMessage message = new FacesMessage("Dane poza zakresem");
       context.addMessage(toValidate.getClientId(context), message);
       stan = 0; }
 }
```

Drugi sposób przy wprowadzaniu danych – atrybut **validator**(cd)

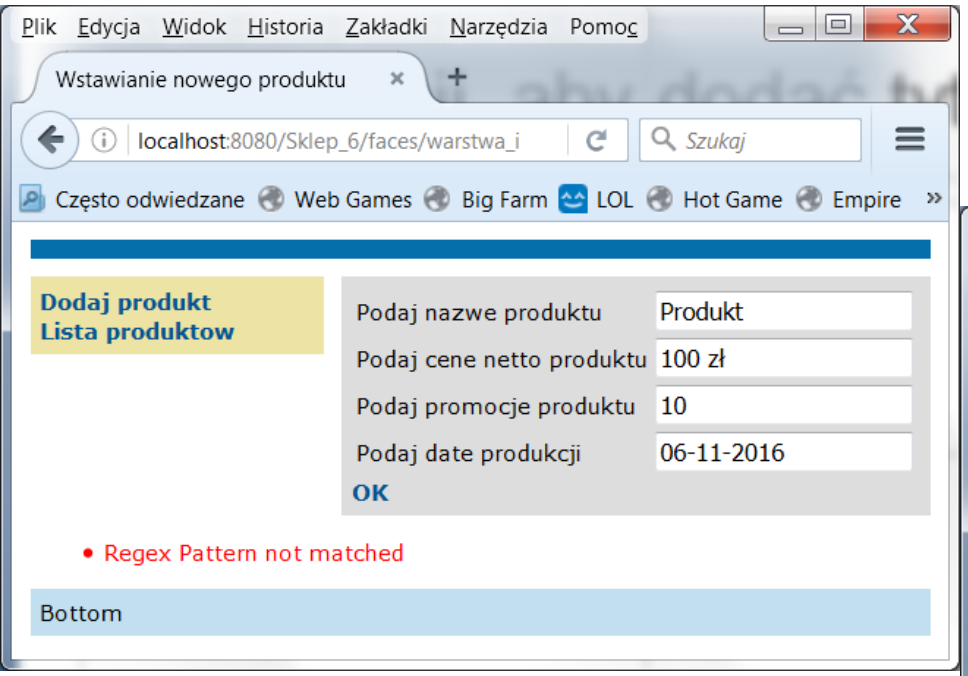


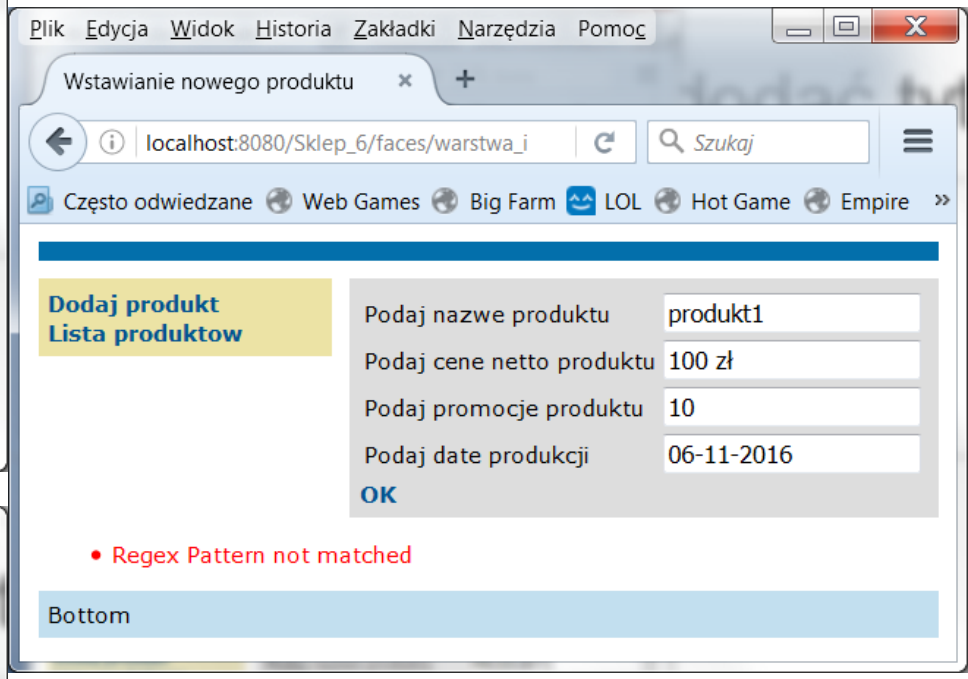
Przykład 3 - Wprowadzanie danych - Konwerter typu **validateRegex** z atrybutem **pattern** określającym wzór łańcucha.

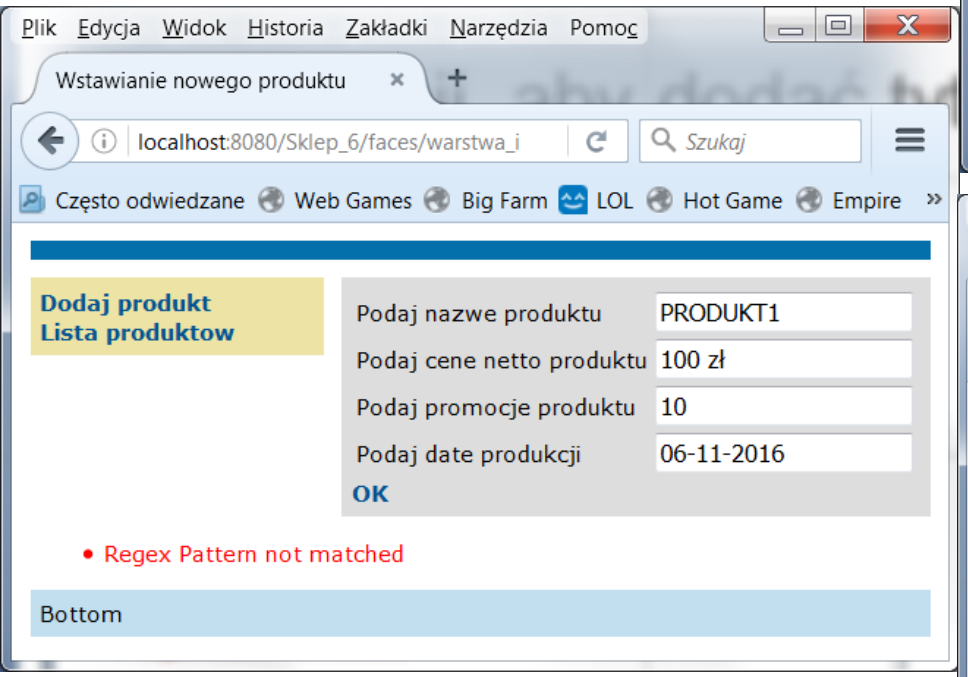
```
<h:outputLabel value="#{bundle['dodaj_produkt2.nazwa']}"
                    for="nazwa" />
<h:inputText
   id="nazwa"
   title="#{bundle['dodaj_produkt2.nazwa1']}"
   value="#{managed_produkt.nazwa}"
   required="true"
   requiredMessage="#{bundle['dodaj_produkt2.blad_nazwa']}" >
     <f:validateRegex pattern="((?=.*\d)(?=.*[a-z])(?=.*[A-Z]).{4,10})"
                      for="nazwa"/>
 </h:inputText>
```

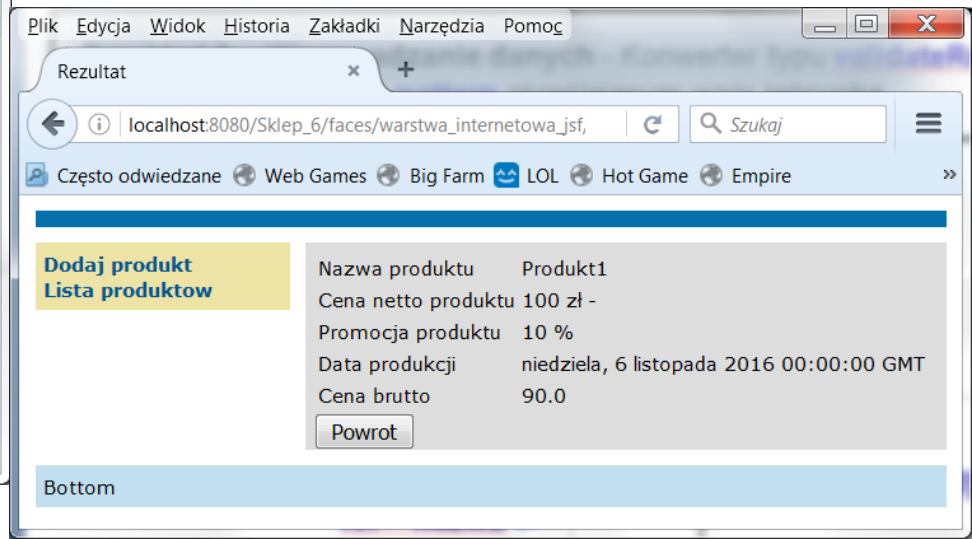
Wzorzec łańcucha:
- Liczba znaków zawarta między 4 a 10 znaków
- Przynajmniej jedna cyfra
- Przynajmniej jedna mała litera (alfabet angielski)
- Przynajmniej jedna duża litera (alfabet angielski)

Przykład 3 (cd) – wynik walidacji łańcucha









Odwołania do metod klasy typu Managed Bean

Atrybuty komponentów, które wskazują metody obiektów typu **Managed Bean**

Atrybut	Funkcja
action	Wskazuje na metodę obiektu typu Managed Bean, kiedy **wykonywana jest nawigacja dla** komponentu i zwracana jest wartość typu String, zawierająca nazwę strony
actionListener	Wskazuje na metodę obiektu typu Managed Bean, **która obsługuje zdarzenie**
validator	Wskazuje na metodę obiektu typu Managed Bean, **która obsługuje walidację**
valueChangeListener	Wskazuje na metodę obiektu typu Managed Bean, **która obsługuje zdarzenie zmiany wartości**

Bindowanie właściwości komponentów typu Managed Bean

Właściwości obiektów typu Managed Bean, tj. atrybuty prywatne wraz z metodami dostępu, mogą być powiązane z:

- z atrybutem **value** komponentu - **metody dostępu (set i get)** przyjmują typ parametru oraz zwracają typ wyniku odpowiedni do **typu atrybutu value komponentu**
- z wystąpieniem komponentu (atrybut **binding** komponentu) - metody dostępu przyjmują typ parametru oraz zwracają typ wyniku odpowiedni **do typu komponentu**
- z implementacją konwertera (atrybut **binding** znacznika konwetera)
- z implementacją walidatora (atrybut **binding** znacznika walidatora) metody
- z implementacją słuchacza zdarzeń (atrybut **binding** znacznika słuchacza zdarzeń)

metody dostępu przyjmują typ parametru oraz zwracają typ wyniku odpowiedni do typu **konwertera, walidatora i słuchacza zdarzeń**

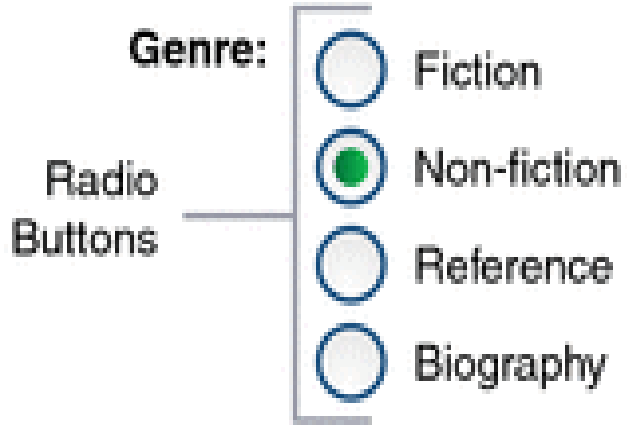
Dopuszczalne typy wartości atrybutu **value** komponentów

Typ komponentu	Typy wartości atrybutu value komponentu
UIInput, UIOutput UISelectItem UISelectOne	Każdy z podstawowych typów danych numerycznych oraz takich obiektów Javy, dla których istnieje implementacja javax.faces.convert.Converter
UIData	Tablica ziaren, lista ziaren, pojedyncze ziarno, java.sql.ResultSet, javax.servlet.jsp.jstl.sql.Result, javax.sql.RowSet
UISelectBoolean	Boolean lub boolean
UISelectItems	java.lang.String, Collection, Array, Map
UISelectMany	Tablica lub lista elementów typów standardowych

Tworzenie list wyboru

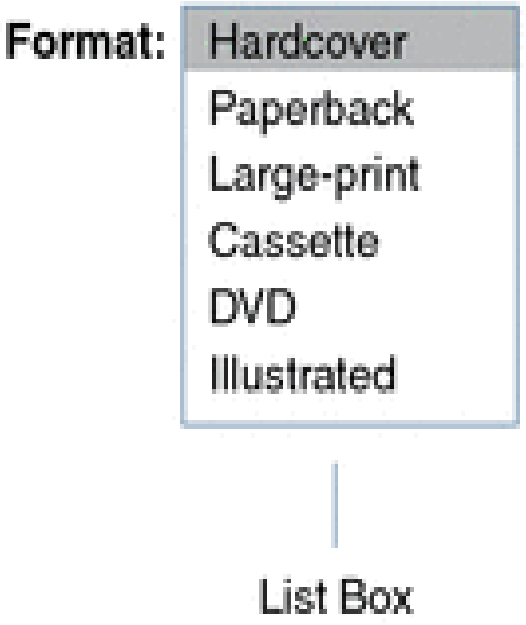
Komponenty wyświetlające komponenty wyboru jednej opcji – UISelectOne, UISelectBoolean

h:selectOneRadio



h:selectOneMenu

h:selectOneListbox



h:selectBooleanCheckbox

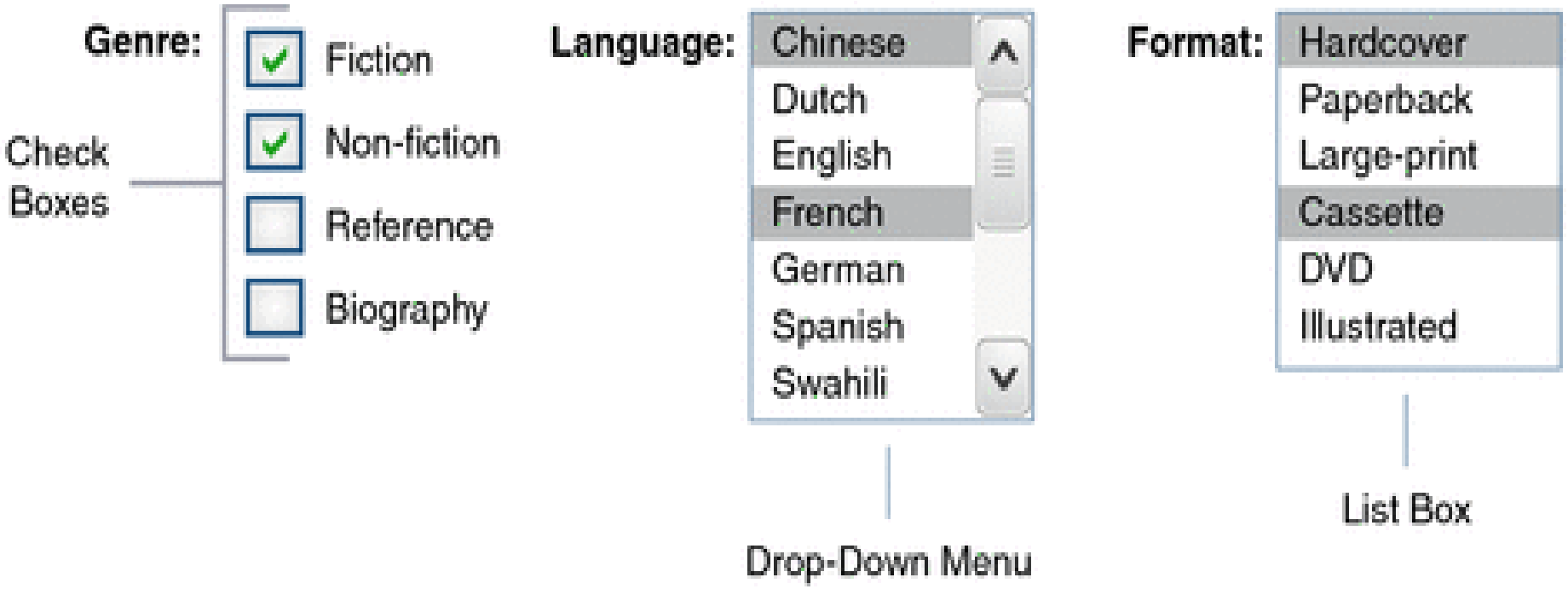
Komponenty wyświetlające komponenty wyboru wielu opcji - UISelectMany

h:selectManyCheckbox – wyświetlany jako zbiór check box

h:selectManyListbox - wyświetlany jako drop-down menu

h:selectManyMenu – wyświetlany jako list box

h:selectManyCheckbox	h:selectManyListbox	h:selectManyMenu



UISelectBoolean - h:selectBooleanCheckbox.

```
<h:selectBooleanCheckbox id="fanClub"  rendered="false"
        binding="#{cashierBean.specialOffer}" />
<h:outputLabel for="fanClub" rendered="false"
   binding="#{cashierBean.specialOfferText} " value="#{bundle.DukeFanClub}" />
</h:outputLabel>
```

//definicja kodu w komponencie **cashierBean** typu **Managed Bean**

```
  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;    }
```

Wartość wyświetlana

Wartość wybrana

UISelectOne- h:selectOneMenu, h:selectOneRadio, h:selectOneListbox

Przykład wyświetlania rezultatów wyboru (ComboBox, drop-down list)

```
<h:selectOneMenu  id="shippingOption" required="true"
                  value="#{cashier.shippingOption}">
    <f:selectItem itemValue="2" itemLabel="#{bundle.QuickShip}"/>
    <f:selectItem itemValue="5" itemLabel="#{bundle.NormalShip}"/>
    <f:selectItem itemValue= "7" itemLabel="#{bundle.SaverShip}"/>
</h:selectOneMenu>
```

//definicja kodu w komponencie **cashier** typu **Managed Bean**

```
private String shippingOption = "2";
public void setShippingOption(String shippingOption) {
     this.shippingOption = shippingOption;
 }
public String getShippingOption() {
     return this.shippingOption;
}
```

Atrybut **value** przechowuje aktualnie wybraną pozycję reprezentowaną przez **itemValue** lub pierwszą, jeśli nie dokonano wyboru. Atrybut **itemLabel** służy do wyświetlania pozycji wyboru.

UISelectItem – wspieranie wyboru jednego elementu

```
<f:selectItem itemValue="#{cashier.itemOne} "
              itemLabel="#{bundle.QuickShip}"/>
```

//definicja kodu w komponencie **cashier** typu **Managed Bean**

```
SelectItem itemOne = null;
SelectItem getItemOne() {
        return itemOne;
  }
void setItemOne(SelectItem item) {
        itemOne = item;
 }
```

Obiekt typu **SelectItem** reprezentuje dwie wartości typu String: etykietę i wartość wybranej opcji. Metody typu **getItemOne** i **setItemOne** służą do obsługi wyboru wartości **itemValue** w znacznikach **f:selectItem.**

UISelectMany- h:selectManyRadio i h:selectManyListbox

```
<h:selectManyCheckbox
            id="newslettercheckbox"
            layout="pageDirection"
            value="#{cashier.newsletters}">
      <f:selectItems value="#{cashier.newsletterItems}"/>
 </h:selectManyCheckbox>
```

//definicja kodu w komponencie **cashier** typu **Managed Bean**

```
    private String newsletters[] = new String[0];
    public void setNewsletters(String newsletters[])
          { this.newsletters = newsletters;
           }
    public String[] getNewsletters()
          { return this.newsletters; }
```

UISelectItems – wspieranie wyboru kilku elementów

```
<f:selectItems value="#{cashier.newsletterItems}"/>
```

//definicja kodu w komponencie **cashier** typu **Managed Bean**

```
private static 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;  }
```

f:selectItems są reprezentowane przez różne typy pojemników: List, Set, Map, Collection zawierających elementy jako zwykłe obiekty Javy (POJOs – Plain Old Java Objects)

Przykład 1 - wykorzystanie znacznika <h:selectOneMenu.
przy wprowadzaniu danych np. promocji.

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

Przykład 1(cd) Kod obsługujący wybór promocji z listy **Drop-Down Menu**

Kod w komponencie typu **Managed_produkt**

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

Kod w komponencie typu **Fasada_warstwy_biznesowej**

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

Przykład 1(cd) Kod z pomocniczej klasy **JsfUtil**

```
package jsf.util;
import java.util.List;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.model.SelectItem;
public class JsfUtil {
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;
    }
```

Przykład 1(cd)

W przypadku braku wyboru promocji z listy domyślnie wybrana jest pierwsza pozycja
Wynik działania wyboru z listy.