Zastosowanie walidatorów oraz komponentów wyboru

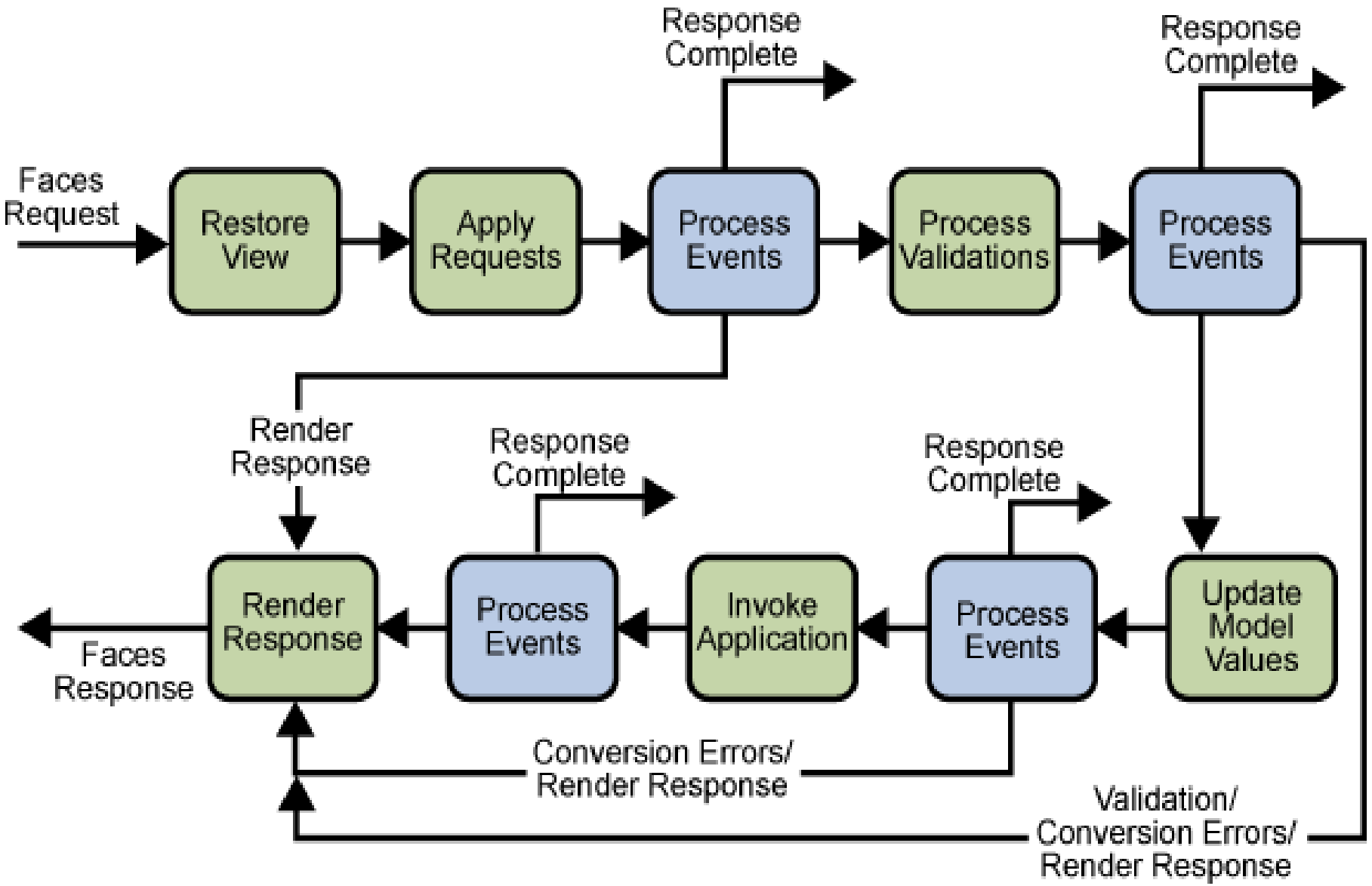
wg

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

Rozdziały 11-12 oraz rozdział 10

Technologie internetowe 7

Standard cyklu życia „Request-Response” dla JavaServer Faces



Cel zastosowania walidatorów

- **Walidatory** służą do sprawdzania poprawności danych odbieranych z komponentów wejściowych.
- **Walidatory** umożliwiają aplikacji wyrażenie ograniczeń na danych wejściowych formularza, aby zapewnić spełnienie niezbędnych wymagań przed przetworzeniem danych wejściowych.

Zastosowanie walidatorów domyślnych- przykład ich rejestrowania

```
<faces-config>
        <application>
                <default-validators>
                        <validator-id>javax.faces.Bean
                        </validator-id>
                </default-validators>
        <application/>
</faces-config>
```

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 o błędach można wykorzystać znaczniki typu **h:Message** oraz **h:Messages**

Uwaga:

Do walidacji można użyć walidatorów typu **Bean Validation definiowanych przez programistę**

Rejestracja walidatorów wartości komponentów

1. Należy odwołać się metody zdefiniowanej w komponencie typu Managed Bean, która przeprowadza walidację za pomocą atrybutu **validator** komponentu (**wykład 3, Przykład 3, slajdy 32-34;** http://zofia.kruczkiewicz.staff.iiar.pwr.wroc.pl/wyklady/ti_/TINT_3.pdf)
2. Należy zarejestrować walidator za pomocą znaczników **<f:>**
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:

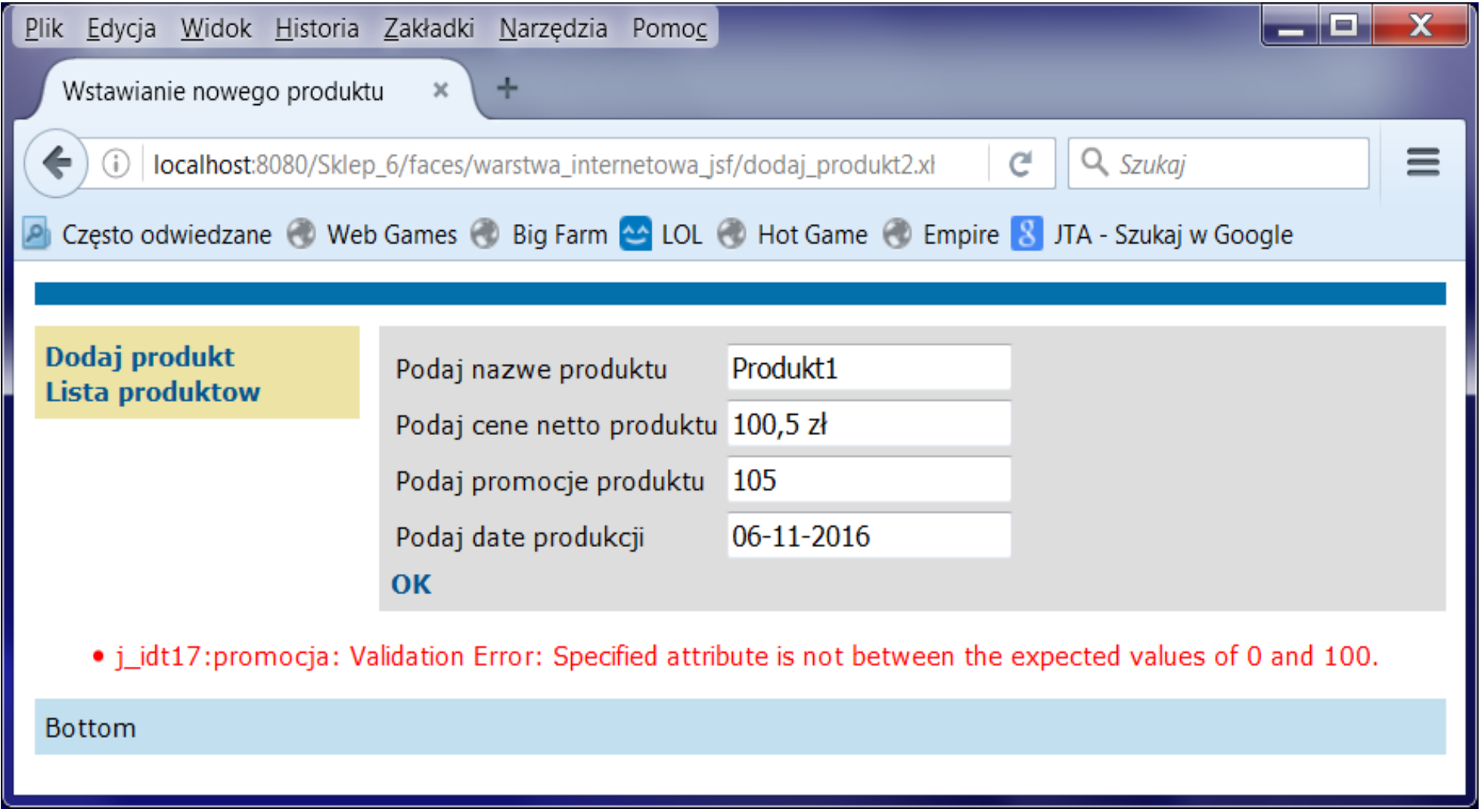
Standardowe walidatory działają jedynie w komponentach, które implementują **EditableValueHolder**.

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: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

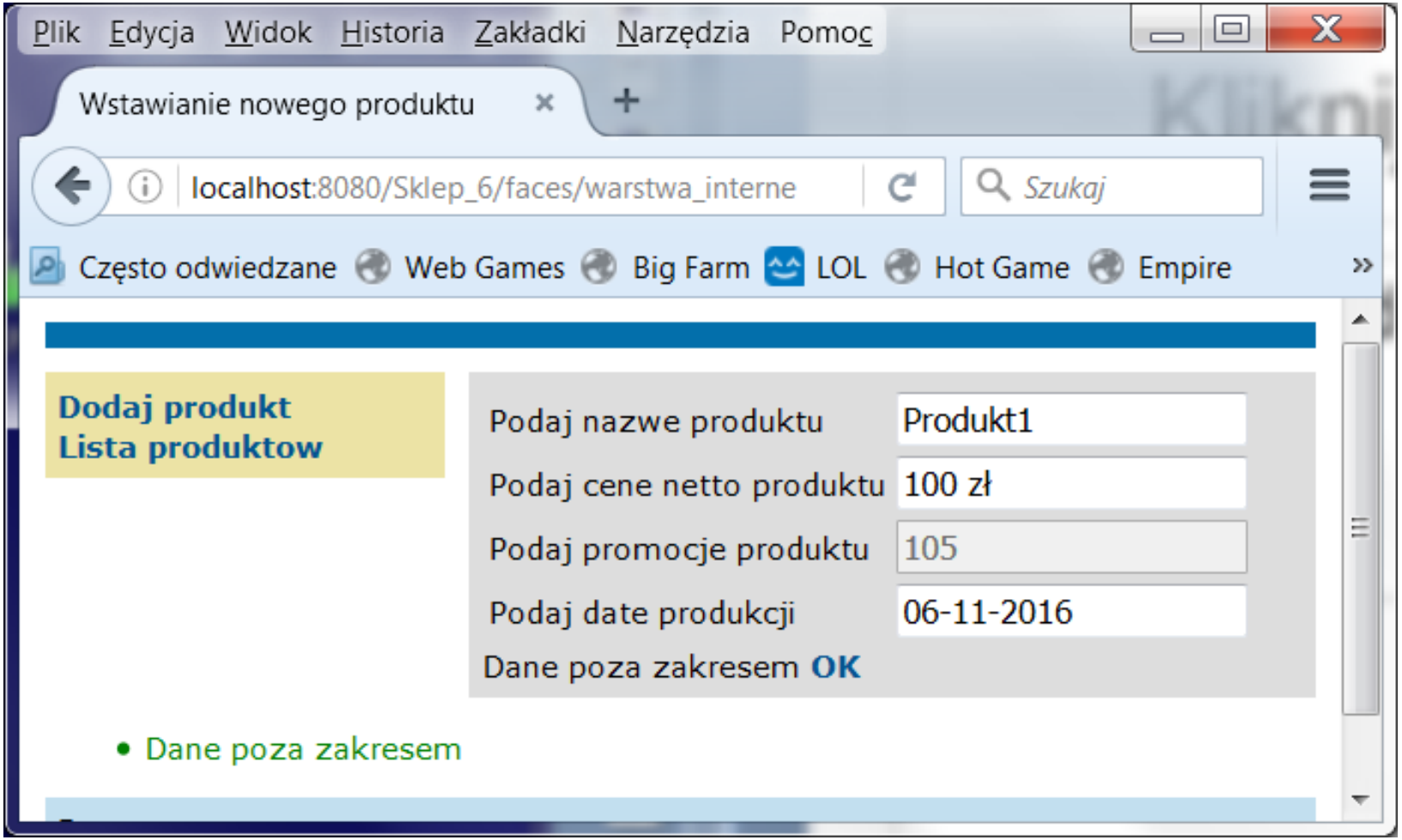
Wykorzystanie atrybutu **disabled** oraz **validator** w znaczniku **<h:inputText**

```
<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)

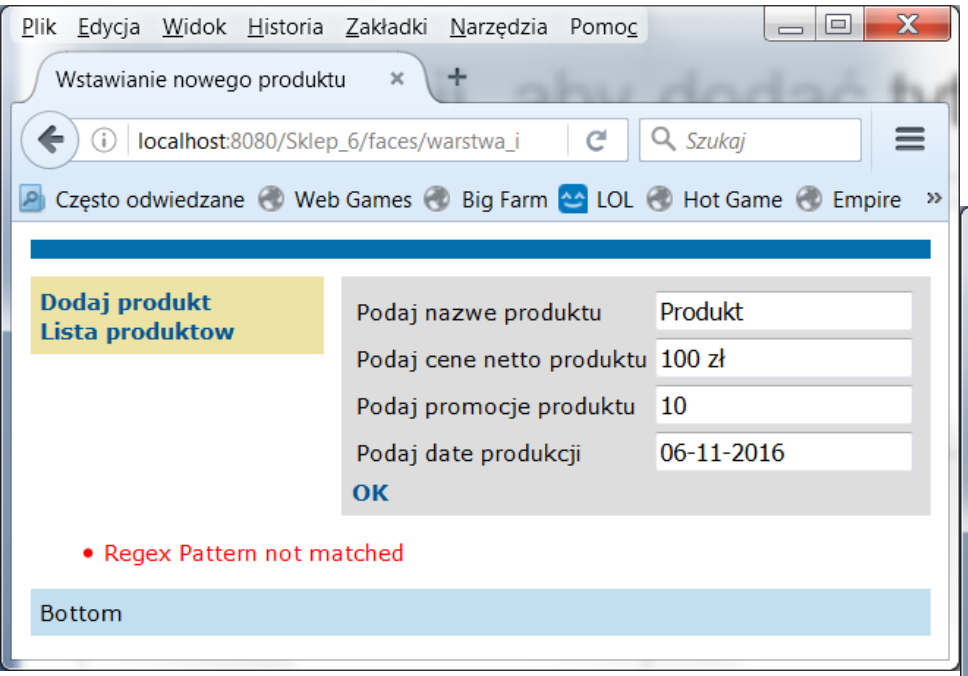


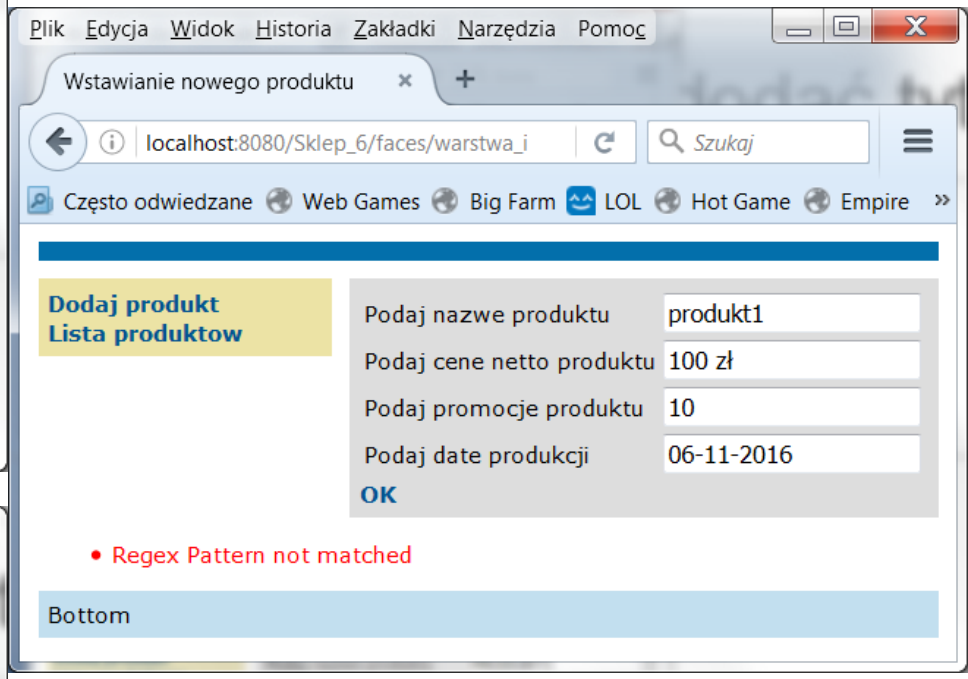
Przykład 3 - Wprowadzanie danych - Konwerter typu **validateRegex** z atrybutem **pattern** określającym wzór łańcucha.

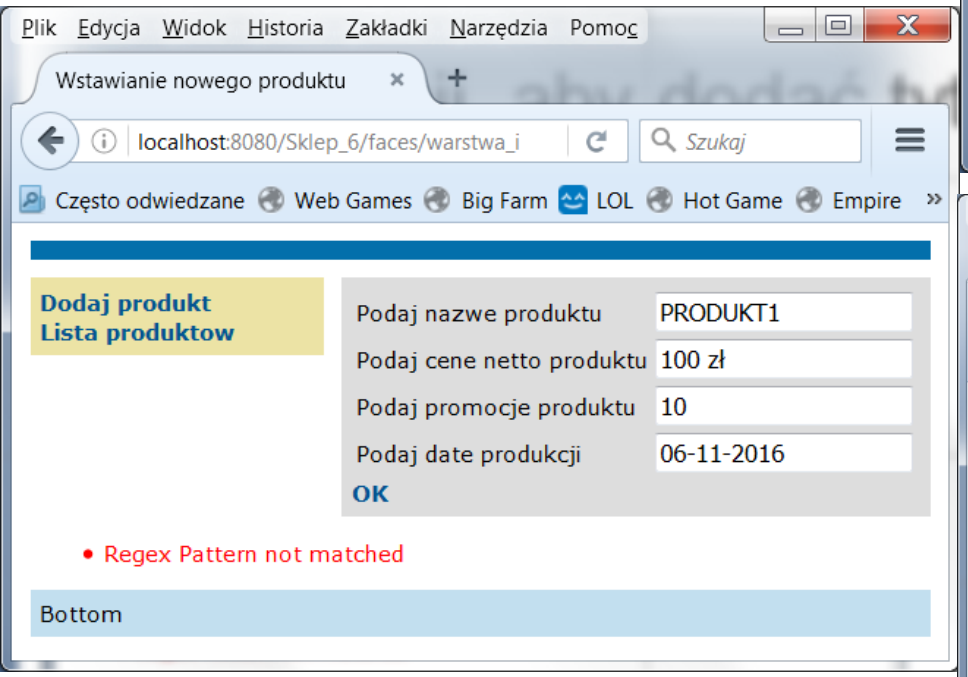
```
<h:outputLabel value="#{bundle['dodaj_produkt2.nazwa']}"
                    for="nazwa" />
<h:inputText
   id="nazwa"
   title="#{bundle['dodaj_produkt2.nazwa1']}"
   value="#{managed_produkt.nazwa}"
   required="true"
   requiredMessage="#{bundle['dodaj_produkt2.blad_nazwa']}" >
     <f:validateRegex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

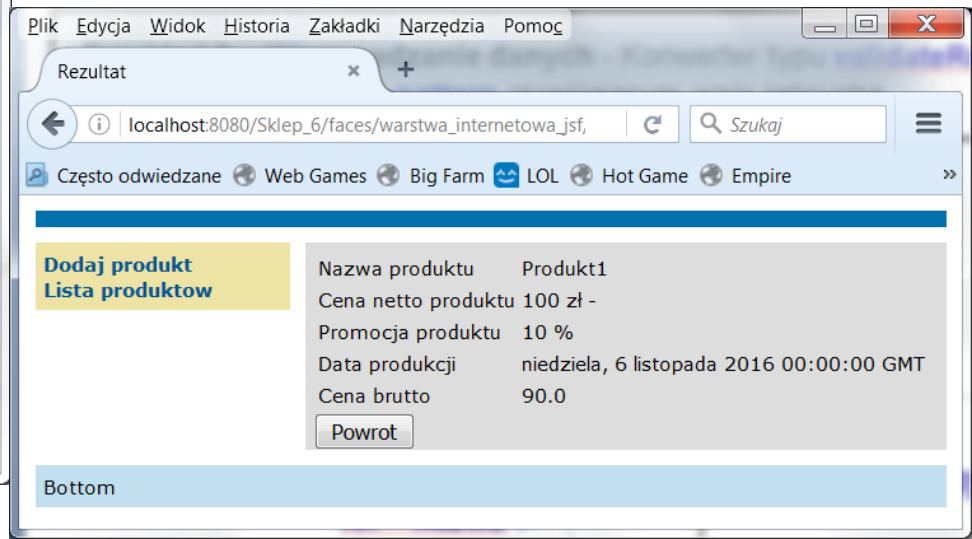
Wzorzec łańcucha:
- Przynajmniej jedna cyfra
- Przynajmniej jedna mała litera (alfabet angielski)
- Przynajmniej jedna duża litera (alfabet angielski)
- Liczba znaków zawarta między 4 a 10 znaków

Przykład 3 (cd) – wynik walidacji łańcucha









Odwołania do metod klasy typu Managed Bean

https://docs.oracle.com/javaee/7/JEETT.pdf (p. 11.4, 12.2)

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**

Komponenty implementujące znaczniki zawierające atrybuty: **action** i **actionListener**, **validator** i **valueChangeListener**

(https://docs.oracle.com/javaee/7/JEETT.pdf, p.7.4)

Tylko komponenty, które implementują **ActionSource**, mogą używać atrybuty **action** i **actionListener**

- UICommand
- UIViewAction

Są renederowane jako button lub link

Tylko komponenty, które implementują **EditableValueHolder**, mogą używać atrybutów **validator** i **valueChangeListener**:

- UIInPut
- UISelectBoolean
- UISelectOne
- UISelectMany
- UIViewParameter

Są renderowane jako elementy do wprowadzania danych

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

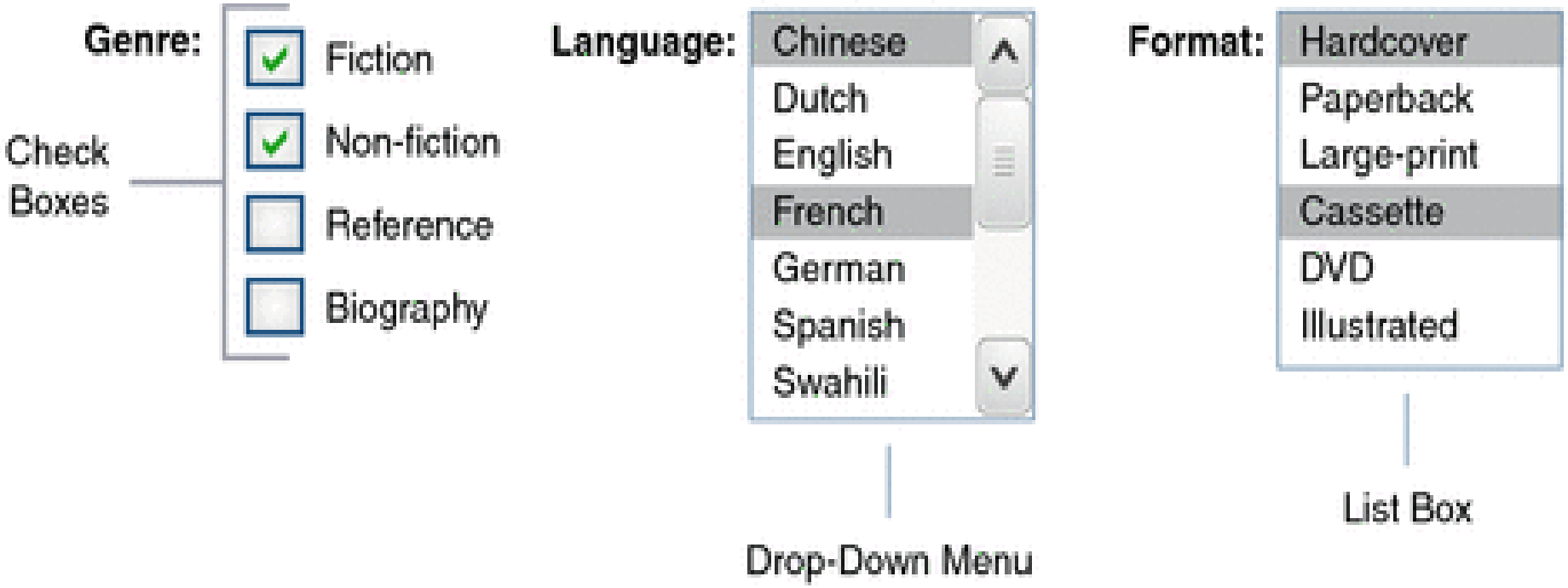
Komponenty wyświetlające komponenty wyboru wielu opcji - UISelectMany

h:selectManyCheckbox – wyświetlany jako zbiór check box (**wykład 4, slajdy 23-25**)

h:selectManyListbox - wyświetlany jako drop-down menu

h:selectManyMenu – wyświetlany jako list box

h:selectManyCheckbox	h:selectManyListbox	h:selectManyMenu



UISelectMany- h:selectManyRadio, h:selectManyListbox h:selectManyCheckbox

```
<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**

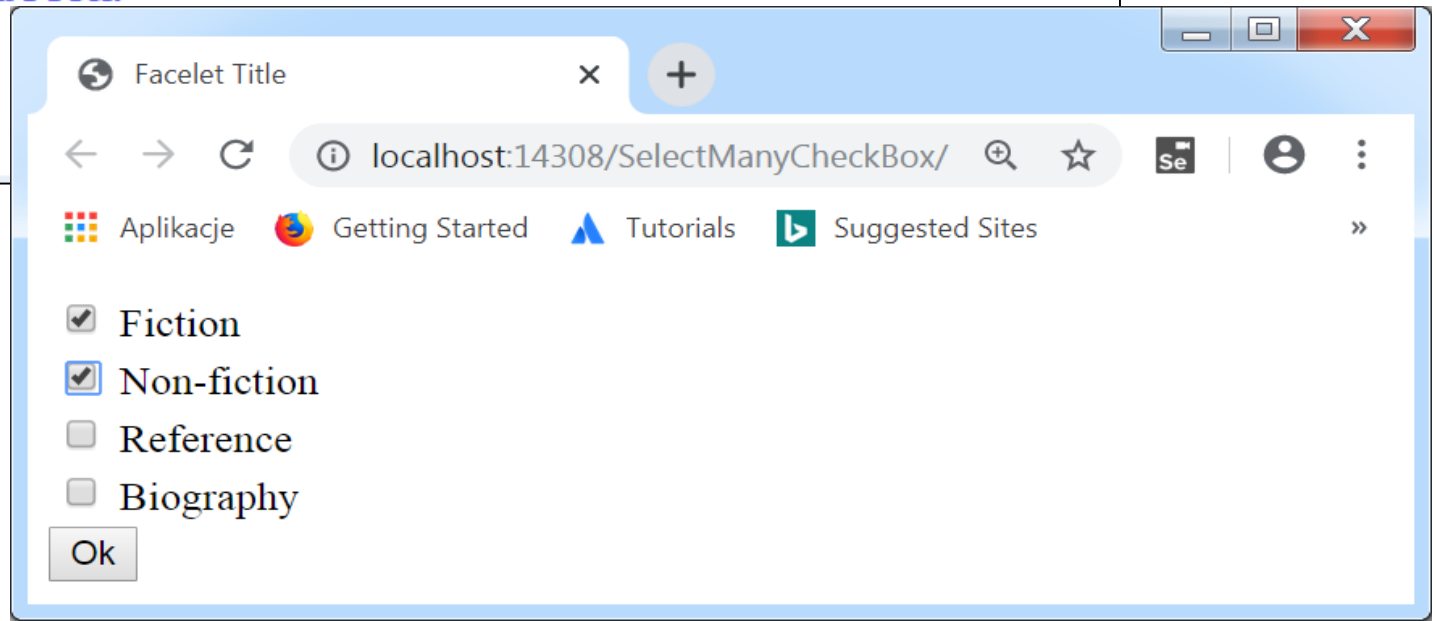
```
private static SelectItem[] newsletterItems = {
        new SelectItem("Duke's Quarterly"),
        new SelectItem("Innovator's Almanac"),
        new SelectItem("Duke's Diet and Exercise Journal"),
        new SelectItem("Random Ramblings") };

public void setNewsletters(String[] newsletters) {
    this.newsletters = newsletters;
  }
public String[] getNewsletters() {
    return this.newsletters;
 }
public SelectItem[] getNewsletterItems() {
    return newsletterItems;  }
```

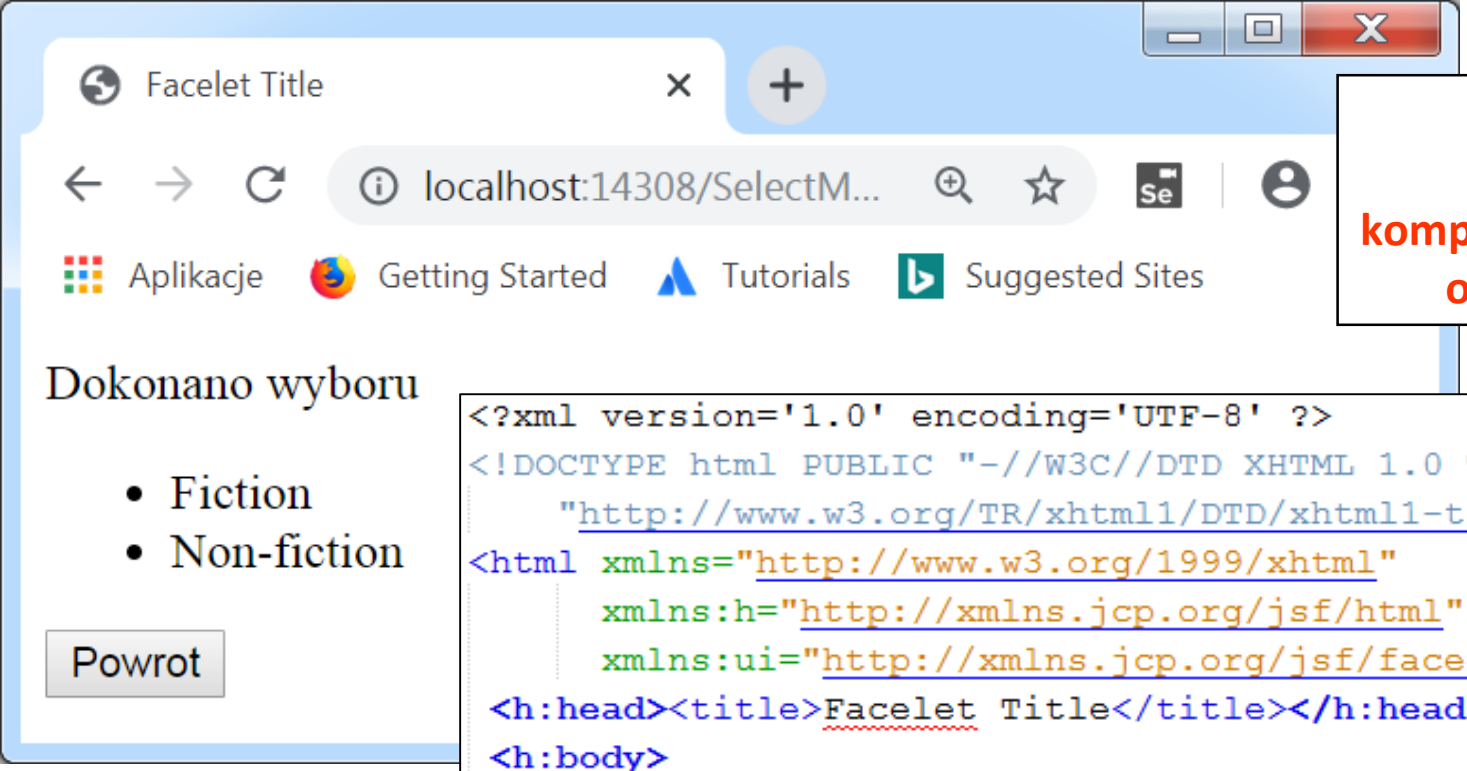
f:selectItems są reprezentowane przez różne typy pojemników: List, Set, Map, Collection zawierających elementy jako zwykłe obiekty Javy (POJOs – Plain Old Java Objects)

Przykład 1 - wykład 4 slajdy: 23-25

Komponenty wyświetlające komponenty wyboru wielu opcji - przykład

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
 <h:head><title>Facelet Title</title></h:head>
 <h:body>
   <h:form>
     <h:selectManyCheckbox  id="newslettercheckbox"
        layout="pageDirection" value="#{wybor.wybraneopcje}">
       <f:selectItems value="#{wybor.opcjewyboru}"/>
     </h:selectManyCheckbox>
     <h:commandButton  id="ok" value="Ok" action="rezultat">
     </h:commandButton>
   </h:form>
 </h:body>
</html>
```





cd Komponenty wyświetlające komponenty wyboru wielu opcji – przyklad cd

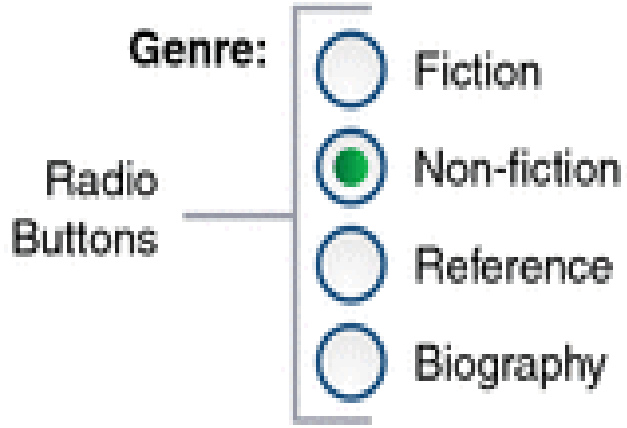
```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
 <h:head><title>Facelet Title</title></h:head>
 <h:body>
  <h:form>
    <h:outputText value="Dokonano wyboru"
                  rendered="#{!empty wybor.wybraneopcje}"/>
     <ul>
       <ui:repeat value="#{wybor.wybraneopcje}" var="nli">
         <li><h:outputText value="#{nli}" /></li>
       </ui:repeat>
     </ul>
     <h:commandButton id="Powrot" value="Powrot" action="index">
     </h:commandButton>
   </h:form>
  </h:body>
</html>
```

cd Komponenty wyświetlające komponenty wyboru wielu opcji – przyklad cd

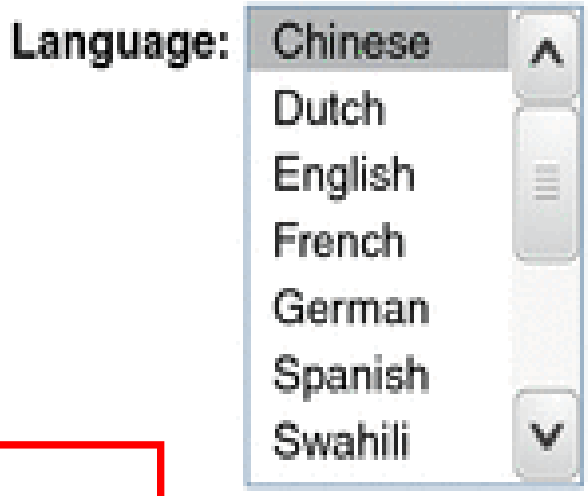
```
import javax.inject.Named;
import javax.enterprise.context.RequestScoped;
import javax.faces.model.SelectItem;

@Named(value = "wybor")
@RequestScoped
public class Wybor {
    private String[] wybraneopcje;
    private  SelectItem[] opcjewyboru = {
        new SelectItem("Fiction"),
        new SelectItem("Non-fiction"),
        new SelectItem("Reference"),
        new SelectItem("Biography")
    };
    public Wybor() { }
    public SelectItem[] getOpcjewyboru() {return opcjewyboru;}
    public void setOpcjewyboru(SelectItem[] opcjewyboru) {
        this.opcjewyboru = opcjewyboru;     }
    public String[] getWybraneopcje()     {return wybraneopcje;}
    public void setWybraneopcje(String[] wybraneopcje) {
        this.wybraneopcje = wybraneopcje; }
}
```

Komponenty wyświetlające komponenty wyboru jednej opcji – UISelectOne, UISelectBoolean

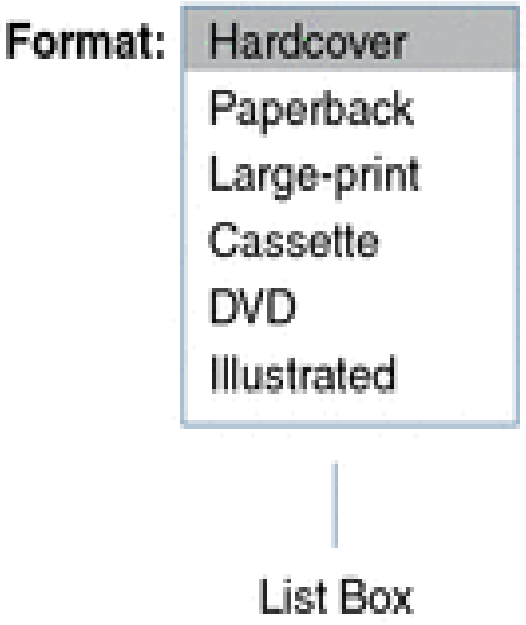
h:selectOneRadio

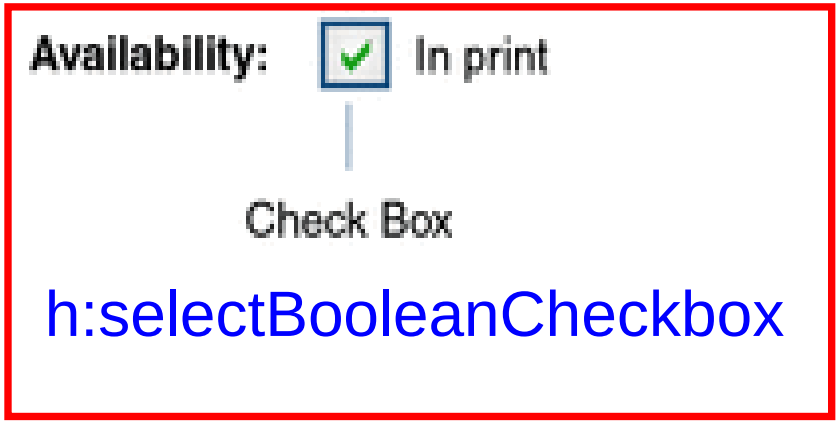


h:selectOneMenu



h:selectOneListbox





h:selectBooleanCheckbox

UISelectBoolean - h:selectBooleanCheckbox.

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

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

```
UIOutput specialOfferText = .....;
  public UIOutput getSpecialOfferText() {
       return this.specialOfferText;   }

  public void setSpecialOfferText(UIOutput specialOfferText) {
       this.specialOfferText = specialOfferText;   }
```

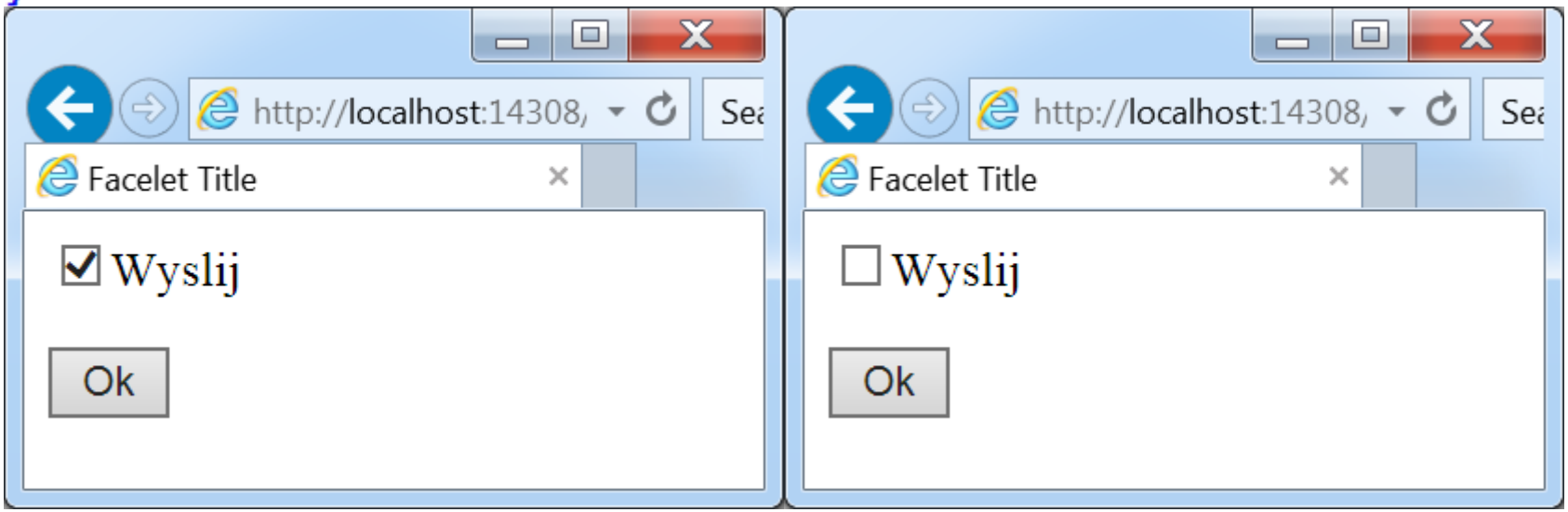
Zdefiniowana przez programistę obsługa wyświetlania wyboru

```
  UISelectBoolean specialOffer = .....;
public UISelectBoolean getSpecialOffer() {
       return this.specialOffer;   }

  public void setSpecialOffer(UISelectBoolean specialOffer) {
       this.specialOffer = specialOffer;    }
```

Zdefiniowana przez programistę obsługa wyboru

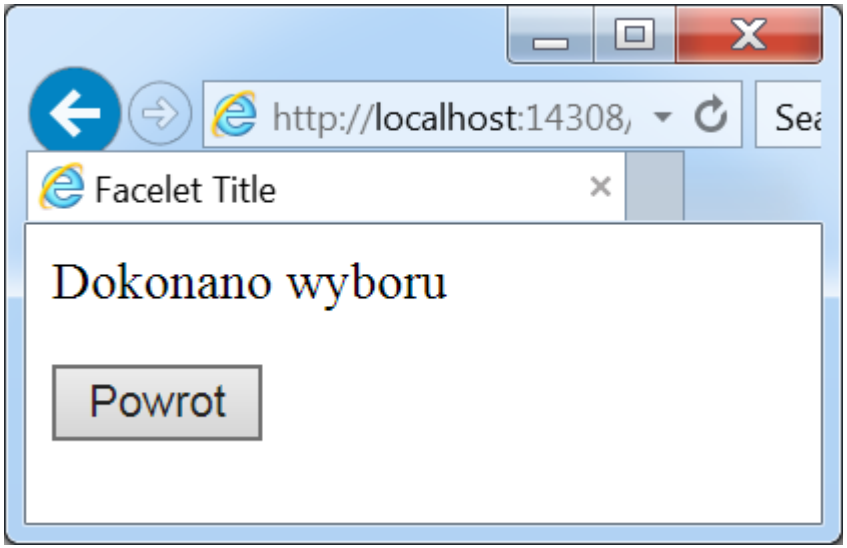
UISelectBoolean - h:selectBooleanCheckbox – Przykład 2

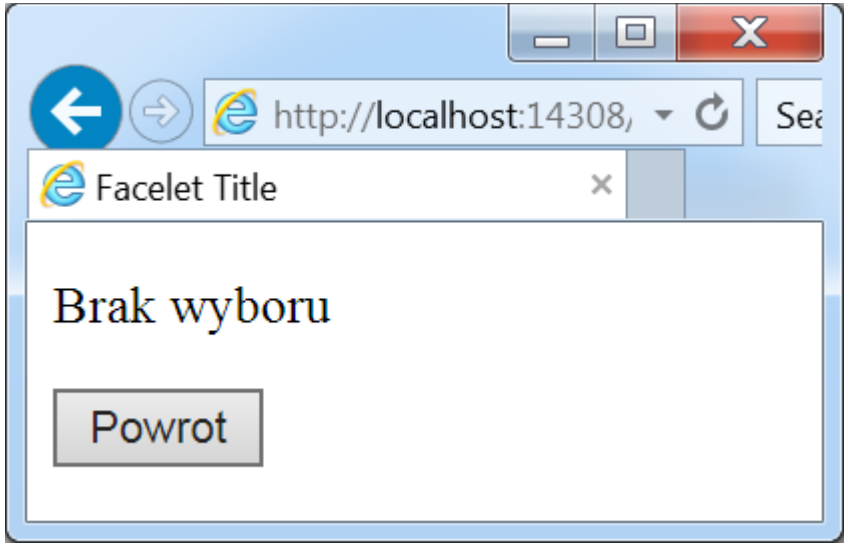
```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
 <h:head><title>Facelet Title</title></h:head>
 <h:body>
   <h:form>
      <h:selectBooleanCheckbox  id="wybor" value="#{wybor.wyborOpcji}"/>
      <h:outputLabel for="wybor" value="#{wybor.wyborOpcjiText}"/>
      <p></p>
      <h:commandButton  id="ok" value="Ok" action="rezultat"/>
   </h:form>
 </h:body>
</html>
```



UISelectBoolean - h:selectBooleanCheckbox. Przykład 2 cd

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">
 <h:head><title>Facelet Title</title></h:head>
 <h:body>
   <h:form>
     <h:outputText value="Dokonano wyboru" rendered="#{wybor.wyborOpcji}"/>
     <p></p>
     <h:outputText value="Brak wyboru" rendered="#{!wybor.wyborOpcji}"/>
     <p></p>
     <h:commandButton id="Powrot" value="Powrot" action="index"/>
   </h:form>
 </h:body>
</html>
```





UISelectBoolean - h:selectBooleanCheckbox. Przykład 2 cd

```
import javax.enterprise.context.RequestScoped;
import javax.inject.Named;

@Named(value = "wybor")
@RequestScoped
public class Wybor {
    boolean wyborOpcji = false;
    String wyborOpcjiText = "Wyslij";

    public String getWyborOpcjiText() { return wyborOpcjiText;  }

    public void setWyborOpcjiText(String wyborOpcjiText) {
        this.wyborOpcjiText = wyborOpcjiText;  }

    public boolean isWyborOpcji()     {  return wyborOpcji;   }

    public void setWyborOpcji(boolean wyborOpcji) {
        this.wyborOpcji = wyborOpcji;   }
}
```

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.**

Przykład 3 - 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 3(cd) Kod obsługujący wybór promocji z listy **Drop-Down Menu**

Kod w komponencie typu **Managed_produkt**

```
public SelectItem[] getItemsAvailableSelectOne() {
        return JSFPomoc.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;
    }
```

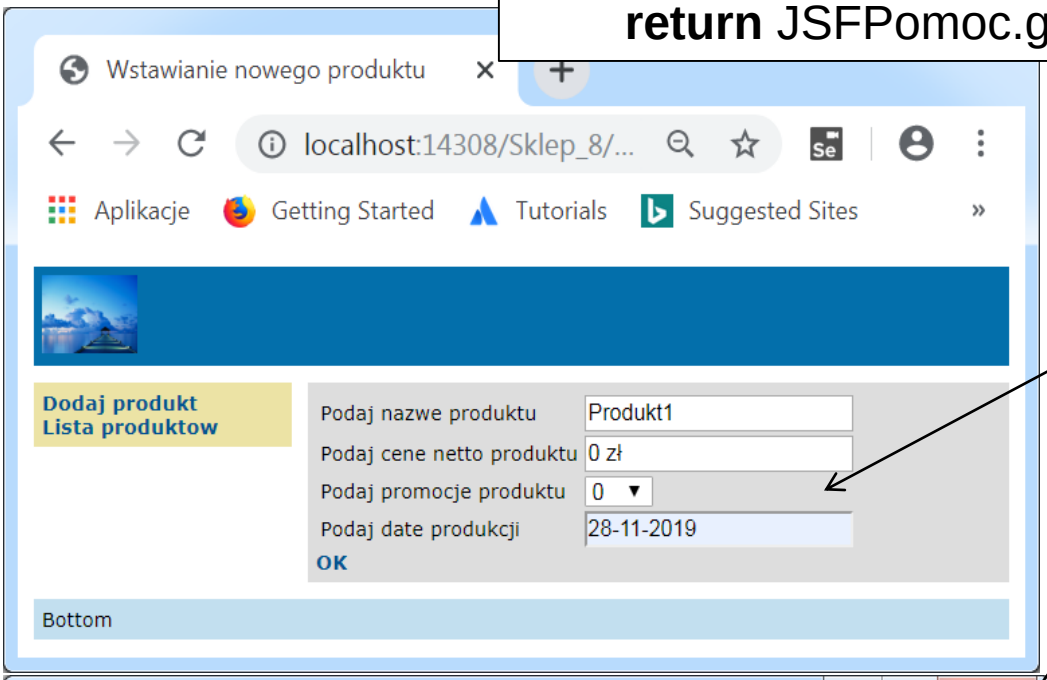
Przykład 3(cd) Kod z pomocniczej klasy **JSFPomoc**

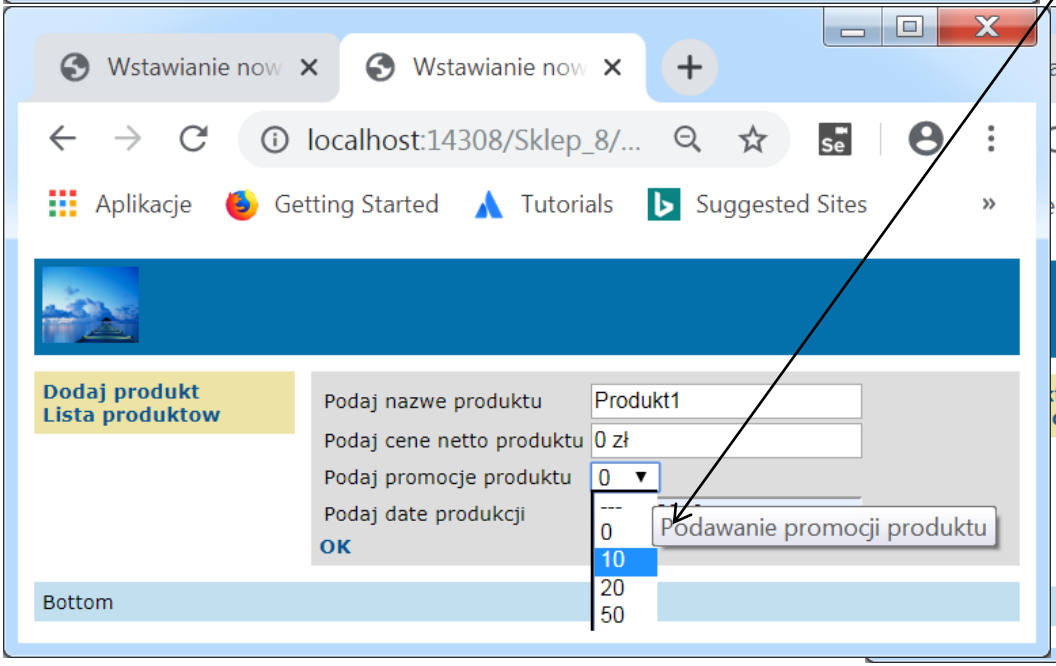
```
package jsf.util;
import java.util.List;
import javax.faces.application.FacesMessage;
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.Converter;
import javax.faces.model.SelectItem;
public class JSFPomoc {
public static SelectItem[] getSelectItems(List<?> 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;
    }
```

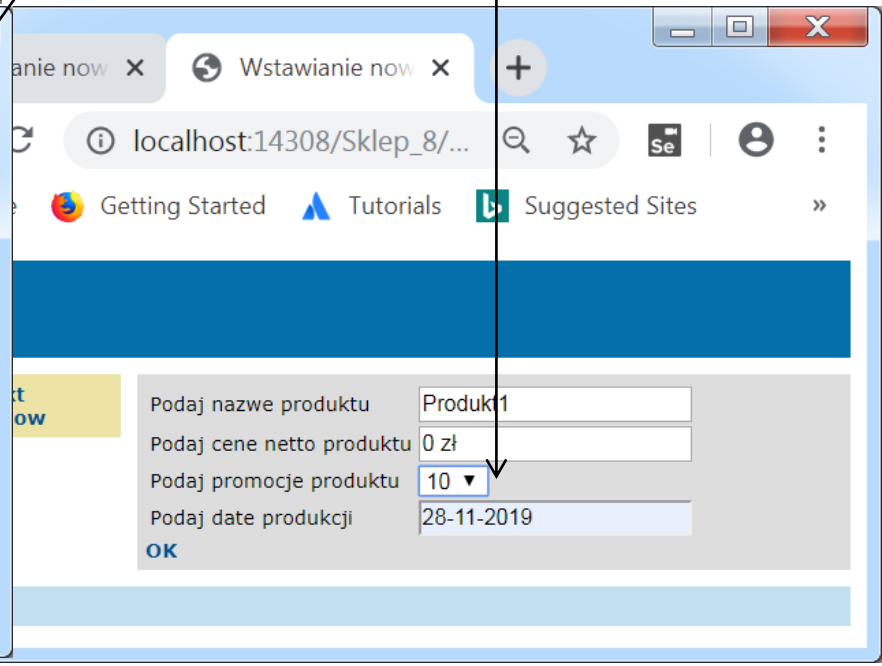
```
public SelectItem[] getItemsAvailableSelectOne() {
    return JSFPomoc.getSelectItems(fasada.findAll(), true);   }
```

Przykład 3 (cd)

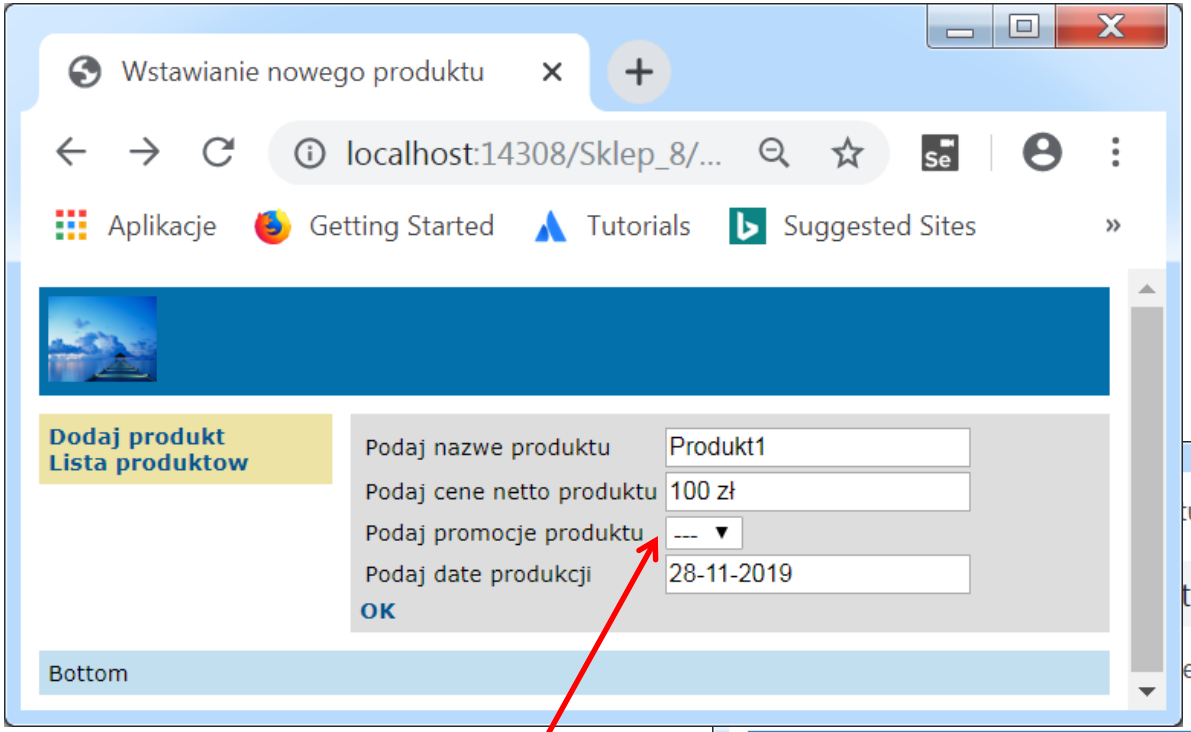
W przypadku braku wyboru promocji z listy domyślnie wybrana jest pierwsza pozycja
Wynik działania wyboru z listy.

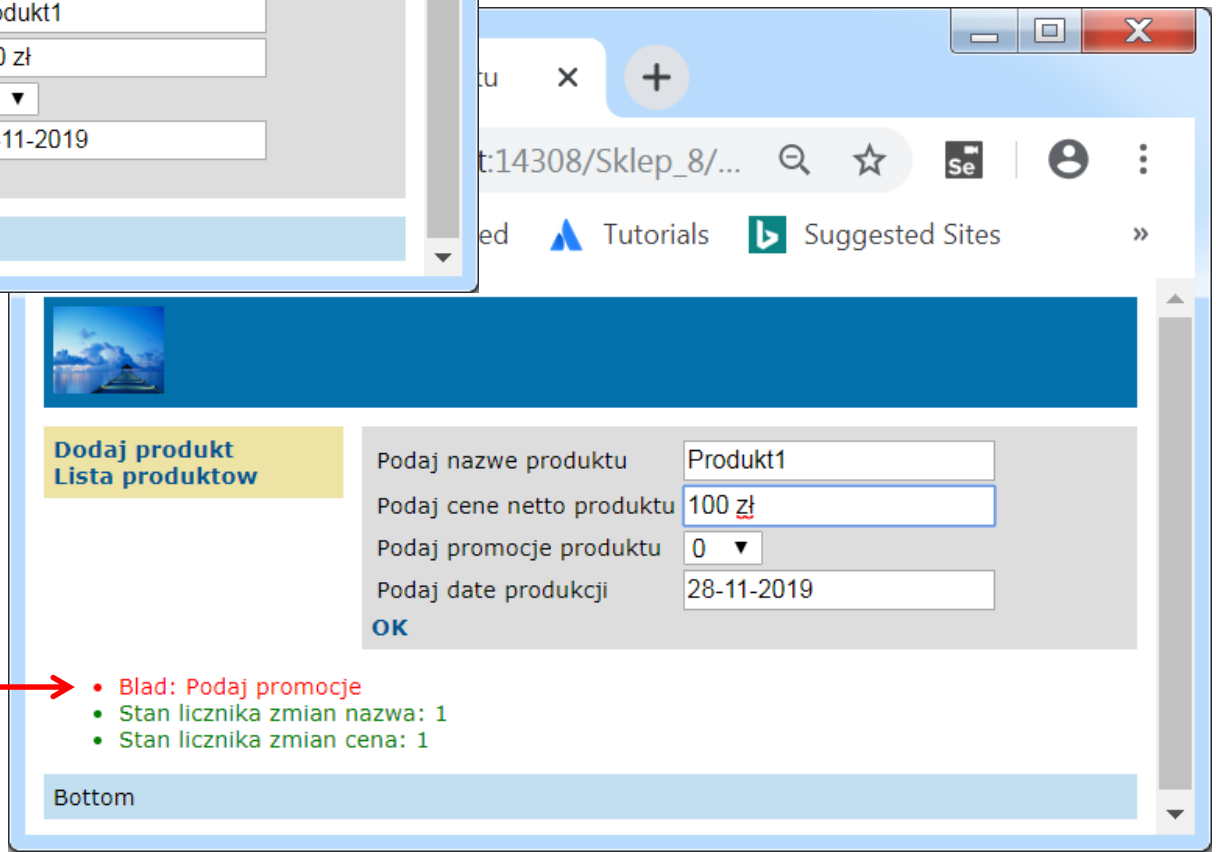






```
public SelectItem[] getItemsAvailableSelectOne() {
        return JSFPomoc.getSelectItems(fasada.findAll(), true);   }
```

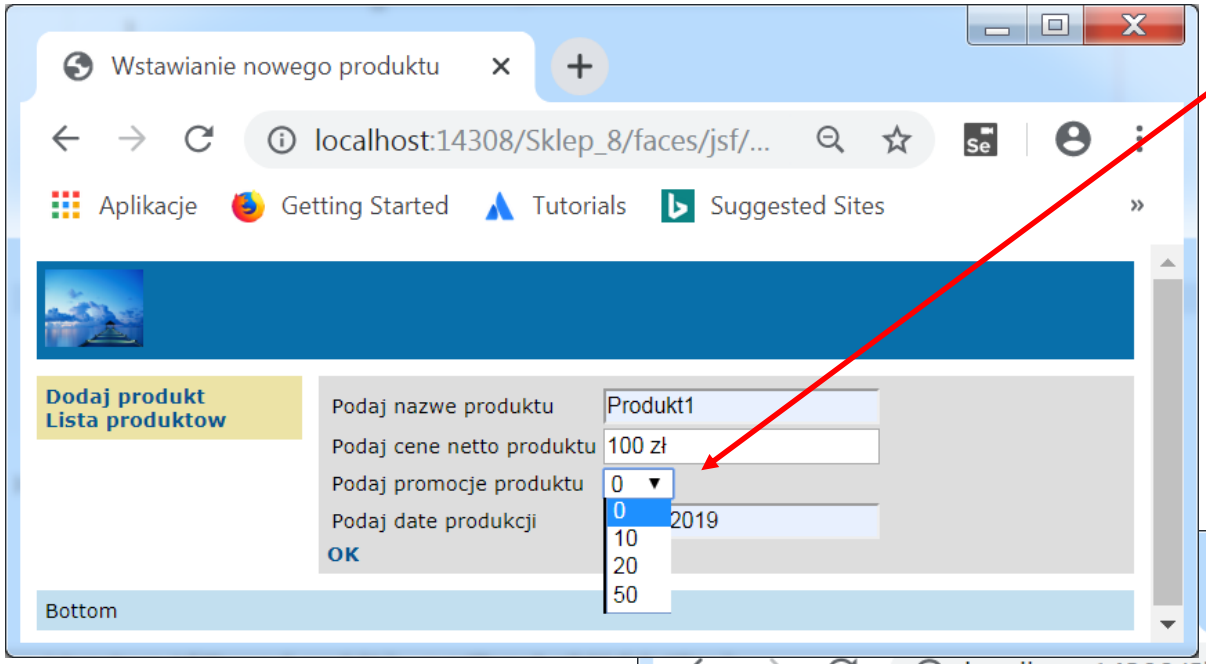


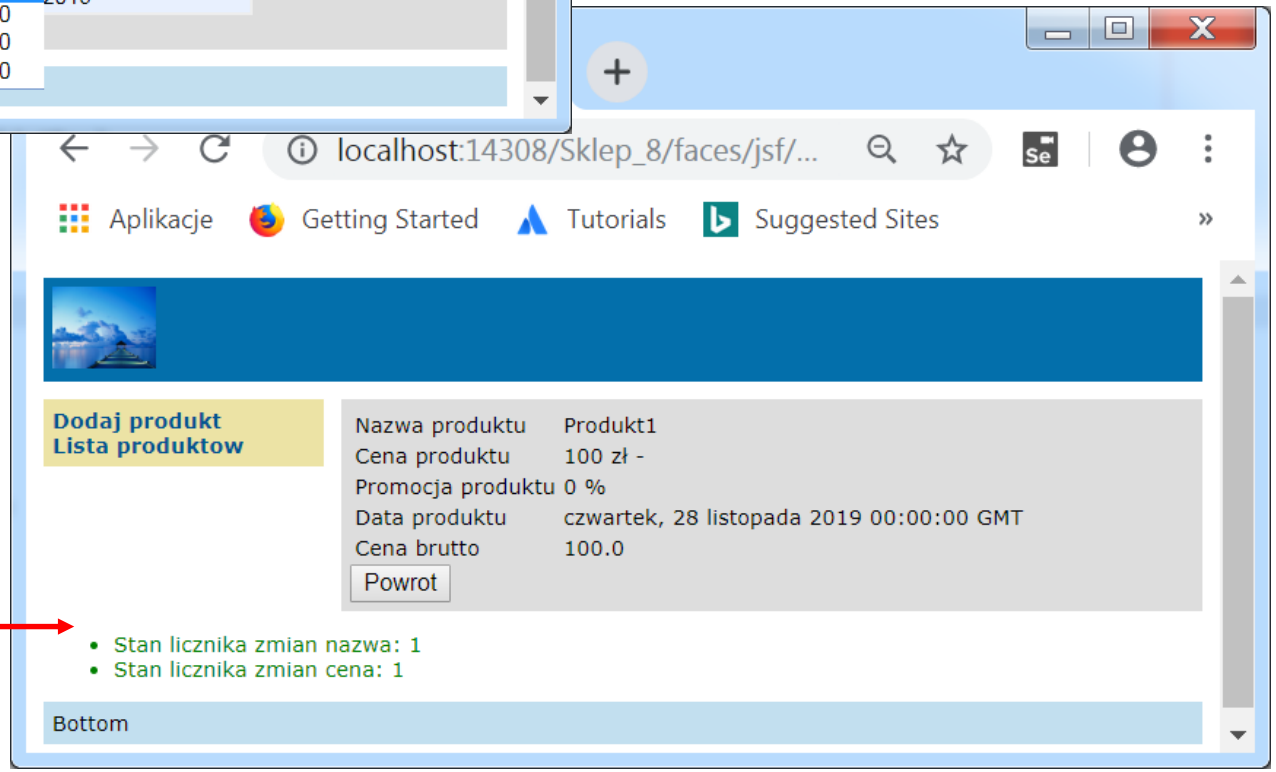


Przykład 3(cd)

W przypadku wyboru pozycji --- z listy promocji zostanie zgłoszony błąd. Wynik działania wyboru z listy.

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





Przykład 3(cd)
W przypadku braku wyboru promocji z listy domyślnie wybrana jest pierwsza pozycja, która teraz jest 0
Wynik działania wyboru z listy.