

Tworzenie aplikacji J2EE w technologii Struts

Plan prezentacji

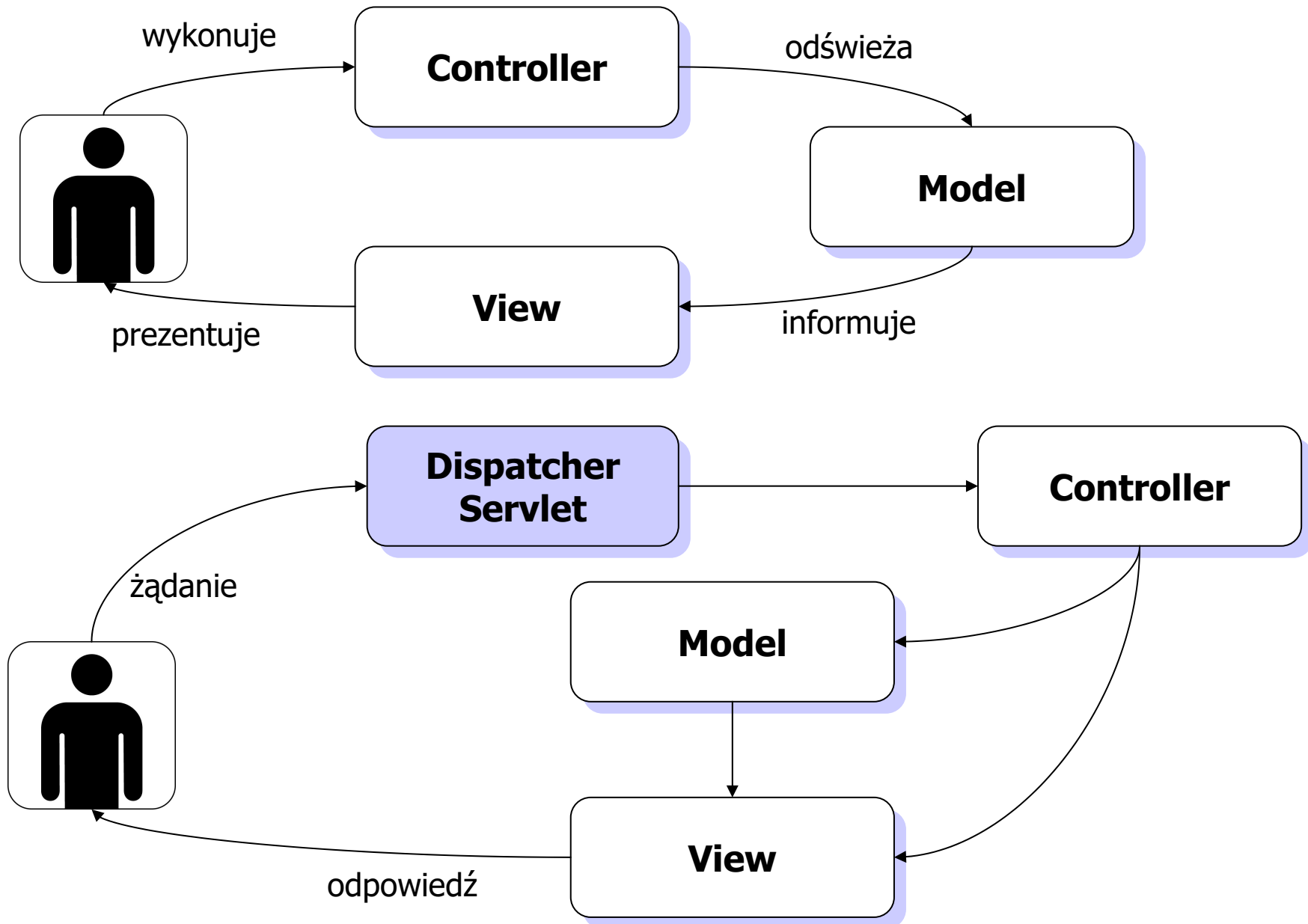
- Architektura MVC
- Wprowadzenie do Apache Struts
- Sterowanie w Apache Struts
- Komponenty Form Bean
- Zaawansowane elementy Struts

Architektura MVC

Architektura Model-View-Controller

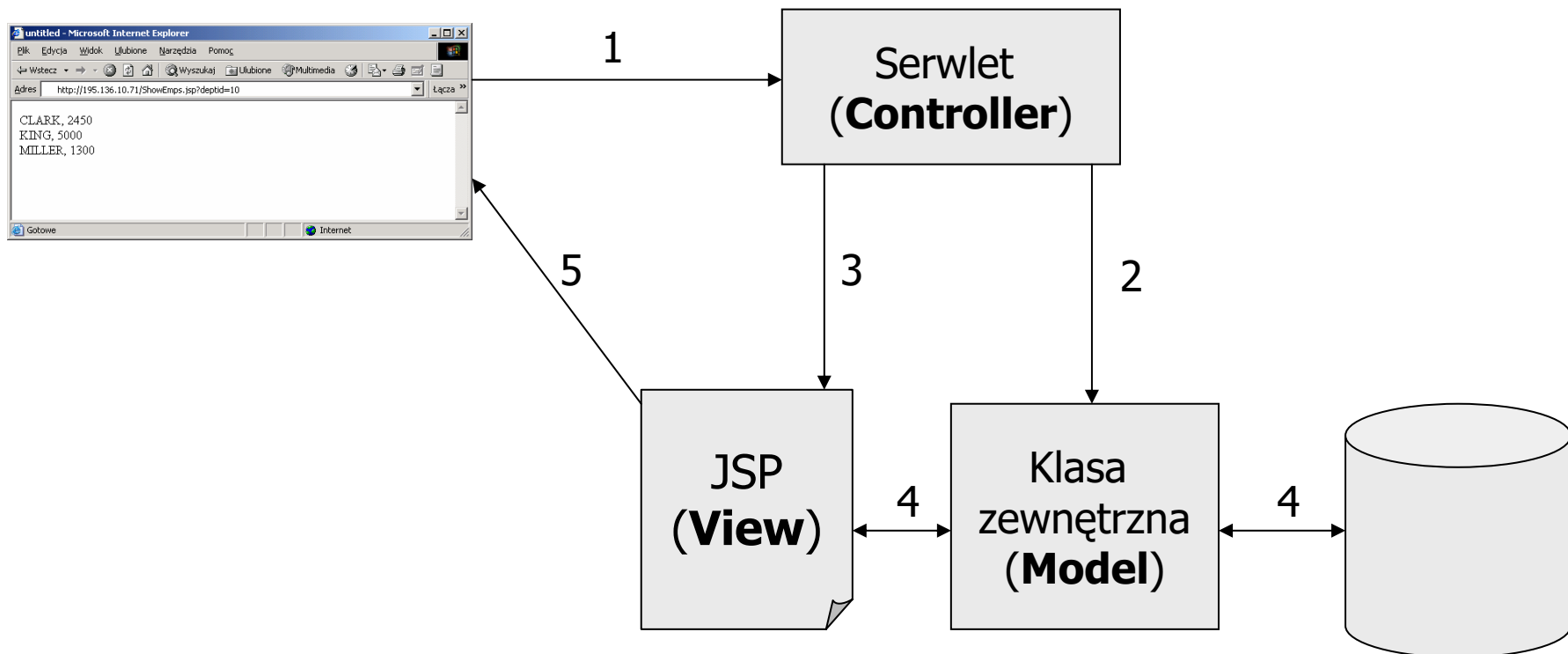
- Architektura MVC zakłada podział składników systemu aplikacyjnego na trzy kategorie:
 - **Model** (komponenty modelu): komponenty reprezentujące dane, na których operują aplikacje; komponenty modelu oferują także metody dostępu do danych
 - **View** (komponenty prezentacji): komponenty reprezentujące wizualizację (prezentację) danych dla użytkownika; komponenty prezentacji pobierają dane od komponentów modelu, a następnie wyświetlają je na ekranie użytkownika
 - **Controller** (komponenty sterujące): komponenty przechwytyjące żądania użytkowników i odwzorowujące je w wywołania metod komponentów modelu; następnie komponenty sterujące przekazują sterowanie do komponentów prezentacji

Uniwersalna architektura MVC



Architektura MVC dla J2EE

- Widoki najczęściej implementowane jako JSP (JSP Model 2)
- Rolę kontrolera pełni serwlet
- Model oparty o JavaBeans, EJB, POJO, Business Components, itp.



Wprowadzenie do Apache Struts

Czym jest Apache Struts?

- Apache Struts to rozwijana na licencji open source powłoka kontrolna wykorzystująca standardowe technologie (serwlety, komponenty JavaBean, XML) i ułatwiająca tworzenie aplikacji webowych zgodnie z paradygmatem Model 2 MVC.
- W zależności od punktu widzenia:
 - szkielet aplikacji MVC
 - zbiór klas i interfejsów użytkowych
 - zbiór bibliotek znaczników JSP

Bałagan na stronie <http://struts.apache.org>

- Struts Action Framework
 - podstawowy szkielet aplikacyjny zgodny z modelem MVC (oryginalny projekt Struts), aktualna wersja stabilna to 1.2.8
- Struts Shale Framework
 - całkowicie nowy szkielet aplikacyjny zgodny z modelem MVC i ściśle zintegrowany z technologią Java Server Faces, projekt w fazie eksperymentalnej
- Struts Extensions
 - EL, Extras, Flow, Struts-Faces, Struts-Taglibs, Scripting, Tiles
- Struts Titanium (Struts Ti)
 - propozycja Struts Action Framework 2 wynikająca z włączenia Open Symphony WebWork do projektu Struts
- Struts Classics
 - Struts 1.3 zostały podzielone na podprojekty, Struts Classics to ujednolicona dystrybucja Struts umożliwiająca tworzenie nowych podprojektów (Struts Action Framework Library?)

Zalety Struts

- Scentralizowane zarządzanie na podstawie plików konfiguracyjnych
 - praktycznie całość zarządzania przeniesiona do plików konfiguracyjnych XML, wysoka elastyczność aplikacji
- Komponenty Form Bean
 - użyteczny mechanizm obsługi danych wejściowych przesyłanych z formularzy HTML
- Bogaty zestaw bibliotek znaczników do obsługi formularzy i komponentów JavaBean
 - łatwość wyświetlania danych przechowywanych w komponentach JB
- Biblioteka znaczników HTML
- Automatyczna walidacja formularzy HTML

Wady Struts

- Duży koszt rozpoczęcia pracy ze środowiskiem
 - konieczność poznania obszernego API, długi czas uczenia się struktury aplikacji, duża inwestycja początkowa
- Niekompletna dokumentacja
 - w porównaniu z innymi technikami Struts są źle udokumentowane, zasoby książkowe i on-line są niewystarczające
- Brak przejrzystości
 - aplikacje Struts są trudne do zrozumienia, oceny i optymalizacji ze względu na dużą liczbę procesów i czynności odbywających się niewidocznie dla projektanta

Składniki Struts

- Model
 - reprezentuje stan aplikacji jako zbiór instancji komponentów JavaBean oraz zbiór akcji zmieniających stan aplikacji. W ramach modelu wyróżnia się dane wejściowe (dane wprowadzone przez użytkownika) i dane wyjściowe (odczytane przez komponenty modelu, być może na podstawie danych wejściowych).
 - logika biznesowa aplikacji jako interakcje między komponentami JavaBean zawartymi w modelu, możliwe wykorzystanie szkieletów:
 - Commons Chain of Responsibility
 - Spring
 - XWork
 - dostęp do danych zewnętrznych realizowany za pomocą różnych technologii:
 - Enterprise Java Beans
 - Hibernate
 - JDBC
 - O/R Bridge

Składniki Struts

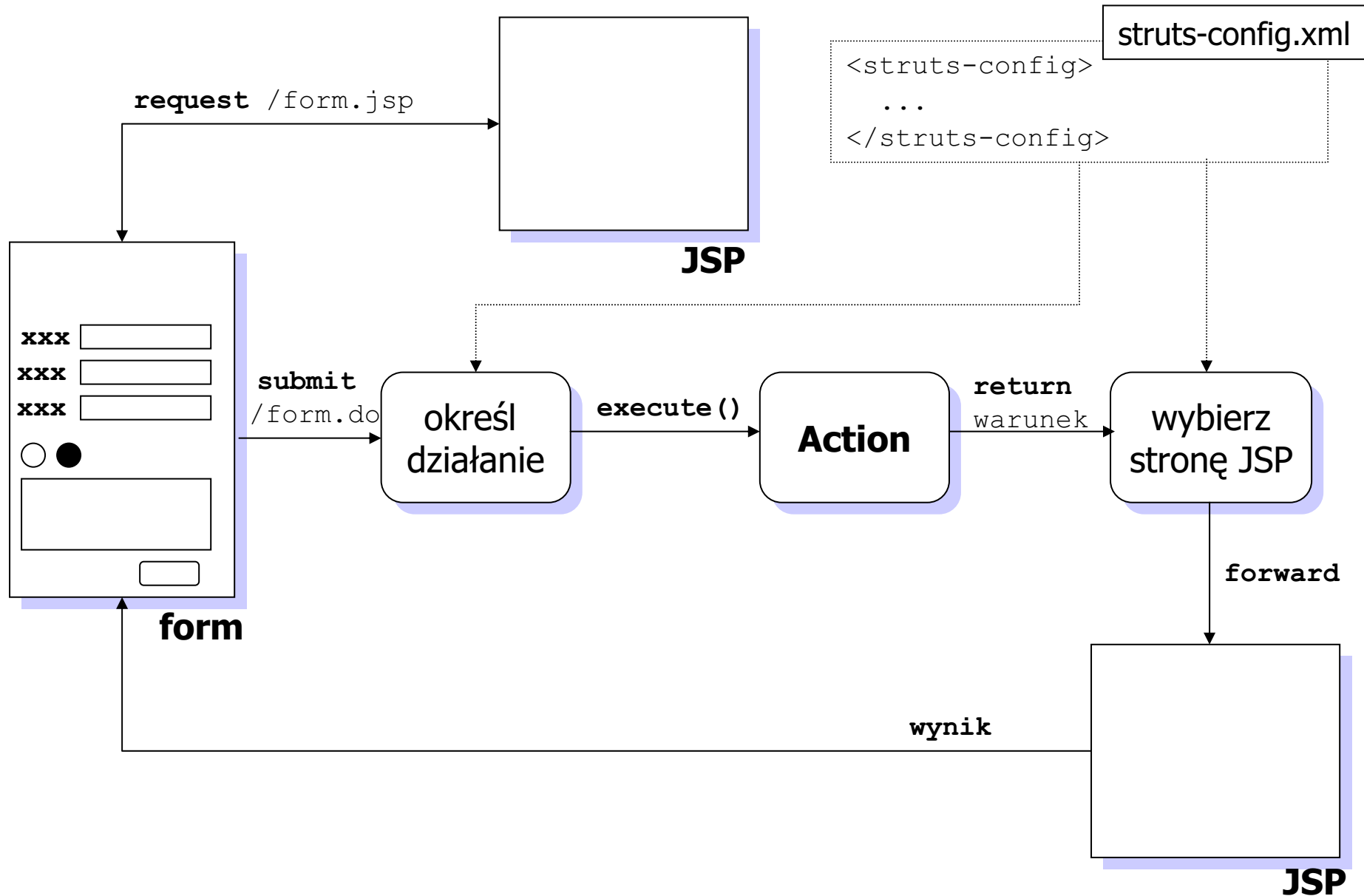
- View
 - najczęściej ta część aplikacji jest konstruowana za pomocą dokumentów JSP i standardowych znaczników umożliwiających interakcję z komponentami JavaBean przenoszącymi informacje, np. `<jsp:useBean>`. Możliwe wykorzystanie specjalizowanych bibliotek znaczników JSP, np. Struts Bean lub zaawansowanych technologii prezentacji, takich jak Cocoon, Velocity Templates, XSLT
 - komponent View jest odpowiedzialny za obsługę wielojęzyczności, automatycznej walidacji formularzy, itp.
 - Struts dostarcza gotowych rozszerzeń ułatwiających korzystanie z rozbudowanych technologii prezentacji:
 - Struts Taglibs
 - Struts Cocoon
 - Velocity Struts
 - Stxx for Struts

Składniki Struts

- Controller
 - podstawowym elementem Struts jest `org.apache.struts.action.ActionServlet` który zarządza całą aplikacją na podstawie konfiguracji zapisanej w pliku `struts-config.xml`. Kontroler jest odpowiedzialny za:
 - przetwarzanie żądań użytkowników
 - określanie zamiarów użytkowników
 - przeniesienie danych z Modelu do Widoku
 - wybór właściwego Widoku i przekazanie go do użytkownika za pomocą odwzorowania `ActionMapping`
 - parsowanie pliku konfiguracyjnego i inicjalizację aplikacji
 - pozostałe elementy komponentu Controller to klasy `org.apache.struts.action.Action` (sterujące przekazywaniem danych z formularzy do warstwy Model) i `org.apache.struts.action.ActionForm` (reprezentujące formularze HTML)

Sterowanie w Struts

Przeływ sterowania w Struts



Przeptyw sterowania w Struts

- Użytkownik żąda formularza `/form.jsp`
- Użytkownik wypełnia formularz i wysyła go pod `/form.do`
 - plik `struts-config.xml` odwzorowuje adres `/form.do` na odpowiednią klasę **Action**
- Metoda **execute ()** klasy **Action** zostaje uruchomiona
 - jednym z parametrów wywołania metody jest komponent **FormBean**, który jest automatycznie tworzony i którego składowe odpowiadają polom formularza
 - klasa **Action** odczytuje dane wejściowe, łączy się z bazą danych i uruchamia logikę biznesową w celu odczytania danych wyjściowych, dane wyjściowe są umieszczane jako zbiór komponentów **JavaBean** w zasięgu żądania, sesji, lub aplikacji
 - klasa **Action** wykorzystuje metodę `mapping.findForward()` do zwrócenia warunku, który jest odwzorowywany przez `struts-config.xml` na wynikową stronę JSP
- Sterowanie zostaje przekierowane do wybranej strony JSP

Prosty przykład aplikacji (1/16)

- Statyczny formularz HTML

form.jsp

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"></meta>
    <title>Login form</title>
  </head>
  <body>
    <h1>Login form</h1>
    <form action="/Login.do">
      <table cellspacing="1" cellpadding="5" border="0">
        <tr><td>username</td><td><input type="text" name="username"/></td></tr>
        <tr><td>password</td><td><input type="password" name="password"/></td></tr>
        <tr><td colspan="2" align="center">
          <input type="submit" value="send"/>
          <input type="reset" value="clear"/>
        </td>
      </tr>
    </table>
  </form>
</body>
</html>

```



Prosty przykład aplikacji (2/16)

- Utworzenie pliku `struts-config.xml`

struts-config.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
  Struts Configuration 1.1//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
  <action-mappings>
    <action path="/Login" type="pl.org.ploug.struts.LoginAction">
      <forward name="success" path="/success.jsp"/>
    </action>
  </action-mappings>

  <global-forwards>
    <forward name="success" path="/success.jsp"/>
  </global-forwards>
</struts-config>
```

warunek zwrócony przez metodę `execute()` klasy `LoginAction`

Prosty przykład aplikacji (3/16)

- Przygotowanie klasy `Action` do której zostanie przesłana zawartość formularza

```
package pl.org.ploug.struts; ← klasa musi być w pakiecie
import javax.servlet.http.*;
import org.apache.struts.action.*;

public class LoginAction extends Action {
    public ActionForward execute(ActionMapping mapping,
                                ActionForm form,
                                HttpServletRequest request,
                                HttpServletResponse response) throws Exception {
        return mapping.findForward("success");
    }
}
```

LoginAction.java

klasa musi rozszerzać klasę `Action`

klasa musi przesyłać metodę `execute`

klasa zwraca warunek logiczny

Prosty przykład aplikacji (4/16)

- Przygotowanie wynikowego dokumentu JSP

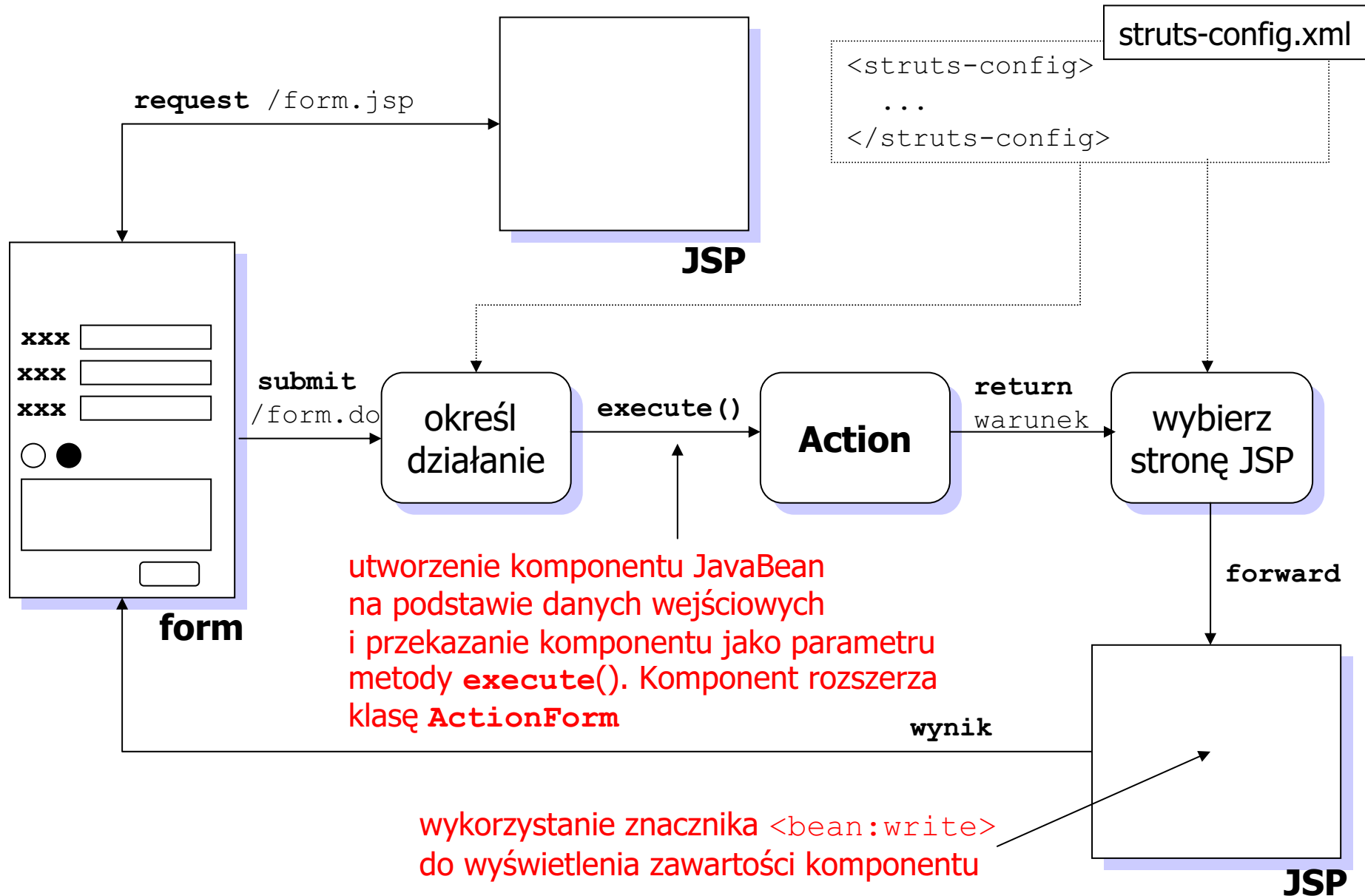
success.jsp

```
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title>Congratulations</title>
  </head>

  <body>
    <h1>Congratulations!</h1>
    You have logged on successfully.
  </body>
</html>
```



Przeływ sterowania w Struts



Prosty przykład aplikacji (5/16)

- Modyfikacja pliku `struts-config.xml`

struts-config.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
  Struts Configuration 1.1//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
  <form-beans>
    <form-bean name="LoginFormBean" type="pl.org.ploug.struts"/>
  </form-beans>

  <action-mappings>
    <action path="/Login" type="pl.org.ploug.struts.LoginAction"
      name="LoginFormBean" scope="request">
      <forward name="success" path="/success.jsp"/>
    </action>
  </action-mappings>
</struts-config>
```

nazwa komponentu

request/session

Komponenty Form Bean

Konstrukcja komponentu Form Bean

- Komponent Form Bean służy do reprezentacji danych wejściowych aplikacji. Komponent jest automatycznie wypełniany danymi wysłanymi z formularza i przekazywany do właściwej metody `execute()`
- Wymagania
 - rozszerza klasę `org.apache.struts.action.ActionForm`
 - posiada konstruktor bezargumentowy lub domyślny
 - posiada zapisywalne składowe, których nazwy odpowiadają parametrom przesłanym z formularza, dla każdej składowej istnieje odpowiadająca jej metoda *setter*
 - posiada metody *getter* dla wszystkich składowych które będą wyświetlane w wynikowych dokumentach JSP

Prosty przykład aplikacji (6/16)

- Stworzenie komponentu Form Bean

LoginFormBean.java

```
package pl.org.ploug.struts; ← klasa musi być w pakiecie

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class LoginFormBean extends ActionForm {
    private String username = "scott";
    private String password = "tiger";

    public String getUsername() { return this.username; }
    public void setUsername(String username) { this.username = username; }

    public String getPassword() { return this.password; }
    public void setPassword(String password) { this.password = password; }
}
```

klasa musi rozszerzać klasę **ActionForm**
metody *setter* i *getter*

Zasady tworzenia komponentów Form Bean

- Programista tworzący klasy dziedziczące z **ActionForm** powinien kierować się następującymi zaleceniami:
 - jeśli klasa **ActionForm** jest umieszczona w zasięgu sesji, to należy koniecznie zaimplementować metodę `reset()`,
 - klasy **ActionForm** nie muszą implementować żadnych metod, najczęściej zawierają tylko komplet metod dostępowych *getter* i *setter* dla składników formularza,
 - klasy **ActionForm** umożliwiają walidację danych wejściowych,
 - cechami formularza mogą być nie tylko pola tekstowe i listy, ale także przyciski, pola wyboru, pola radiowe, itp.,
 - klasa **ActionForm** powinna być postrzegana jako bufor między użytkownikiem i klasą **Action**, sprawdzenie poprawności danych i obecności wymaganych danych powinno się odbyć przed przekazaniem wartości do przetwarzania klasie **Action**.

Prosty przykład aplikacji (7/16)

- Stworzenie komponentu reprezentującego dane biznesowe, klasa nie jest rejestrowana w `struts-config.xml`

UserDataBean.java

```
package pl.org.ploug.struts; ← klasa musi być w pakiecie
```

```
public class UserDataBean {  
    private String username = "scott";  
    private String password = "tiger";
```

```
    public UserDataBean(String username, String password) {  
        this.username = username;  
        this.password = password;  
        // logika aplikacji, np. doczytanie informacji z bazy danych  
    }
```

```
    public String getUsername() { return this.username; }  
    public String getPassword() { return this.password; }
```

```
}
```

metody *setter* i *getter*

Prosty przykład aplikacji (8/16)

- Modyfikacja klasy `Action`
 - pobranie danych wejściowych z komponentu Form Bean
 - przygotowanie komponentu biznesowego i umieszczenie go we właściwym zasięgu

LoginAction.java

```
package pl.org.ploug.struts;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class LoginAction extends Action {
    public ActionForward execute( ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception {

        LoginFormBean lfb = (LoginFormBean)form;
        String username = lfb.getUsername();
        String password = lfb.getPassword();
        UserDataBean udb = new UserDataBean(username, password);
        request.setAttribute("UserDataBean", udb);

        return mapping.findForward("success");
    }
}
```

Prosty przykład aplikacji (9/16)

- W wynikowym dokumencie JSP dostęp do składowych komponentu JavaBean przez znacznik `<bean:write>`

```
<%@ page contentType="text/html; charset=windows-1252"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title>Congratulations</title>
  </head>
  <body>
    <h1>Congratulations!</h1>
    You have logged on successfully <br>
    as <bean:write name="UserDataBean" property="username"/>
    with password <bean:write name="UserDataBean" property="password"/>
  </body>
</html>
```

success.jsp

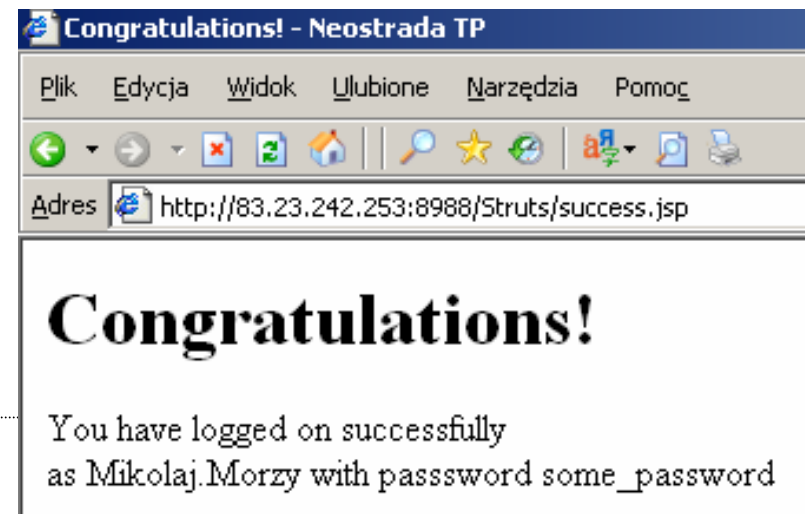


Prosty przykład aplikacji (10/16)

- W wynikowym dokumencie JSP dostęp do składowych komponentu JavaBean przez język wyrażeń EL
 - dostępny od JSP 2.0 (Tomcat 5.x, WebLogic 9)
 - konieczne użycie wersji `web.xml` odpowiadającej Servlet API 2.4
 - brak odpowiednika `<bean:message>` w EL

success.jsp

```
<%@ page contentType="text/html; charset=windows-1252"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title>Congratulations</title>
  </head>
  <body>
    <h1>Congratulations!</h1>
    You have logged on successfully <br>
    as ${UserDataBean.username}
    with password ${UserDataBean.password}
  </body>
</html>
```



Dynamiczne komponenty Form Bean

- Jeśli komponent Form Bean służy tylko i wyłącznie do wczytania danych z formularza i standardowej walidacji, zamiast tradycyjnych komponentów Form Bean można wykorzystać dynamiczne komponenty Form Bean:
 - dynamiczny komponent nie wymaga oprogramowania klasy Java, definicja komponentu jest w całości umieszczona w pliku `struts-config.xml`,
 - dynamiczne komponenty to instancje klasy `org.apache.struts.action.DynaActionForm`,
 - można tworzyć hierarchię klas w oparciu o klasę `DynaActionForm`, ale nie można w klasie umieszczać jednocześnie cech tradycyjnych i dynamicznych (czyli można specjalizować metody `validate()` i `reset()`),
 - `DynaActionForm` podlega automatycznej walidacji jeśli rozszerza `org.apache.struts.action.ValidatorActionForm`.
 - dostęp do cech klasy z wyrażeń EL: `${beanName.map.property}`

Przykład DynaActionForm

struts-config.jsp

```

...
<struts-config>
  <form-beans>
    <form-bean name="LoginFormBean" type="pl.org.ploug.struts"/>
    <form-bean name="DynaLoginFormBean"
      type="org.apache.struts.action.DynaActionForm">
      <form-property name="username" type="java.lang.String" initial="scott"/>
      <form-property name="password" type="java.lang.String" initial="tiger"/>
    </form-bean>
  </form-beans>
</struts-config>
...

```

LoginAction.java

```

...
public class LoginAction extends Action {
  public ActionForward execute( ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response) throws Exception {

    DynaActionForm lfb = (DynaActionForm) form;
    String username = (String) lfb.get("username");
    String password = (String) lfb.get("password");
    ...
  }
}

```

rzutowanie niezbędne gdyż
DynaActionForm przechowuje
mapę składowych w postaci
obiektów klasy Object

Leniwe komponenty Form Bean

- Leniwy programista może poniechać definiowania komponentu Form Bean w pliku `struts-config.xml` i wykorzystać klasę **`org.apache.commons.beanutils.LazyDynaBean`**:
 - komponent `LazyDynaBean` jest automatycznie wypełniany wszystkimi parametrami przesłanymi z formularza,
 - składowe komponentu `LazyDynaBean` mogą być dodawane i usuwane za pomocą metod `get()` `set()` i `remove()`,
 - jeśli komponent posiada indeksowane składowe (`List`, `Array`) to są one automatycznie rozszerzane wg potrzeby,
 - składowe komponentu będące listami, tablicami, mapami i zagnieżdżonymi komponentami Java Bean są automatycznie tworzone.

Zaawansowane elementy Struts

Zbiory zasobów

- Mechanizm zbiorów zasobów (resource bundles) umożliwia scentralizowane zarządzanie zasobami tekstowymi (tytuły stron, etykiety pól, łącza, komunikaty o błędach) i łatwą obsługę wielojęzyczności.
- Zbiory zasobów to pliki tekstowe zawierające mapę z kluczami i wartościami komunikatów. Pliki zasobów muszą być zarejestrowane w `struts-config.xml`

```
form.title = Success  
form.confirmation = Congratulations!  
form.message = You have logged on successfully
```

ApplicationResources.properties

```
<struts-config>  
  ...  
  <message-resources parameter="pl.org.ploug.struts.ApplicationResources"/>  
</struts-config>
```

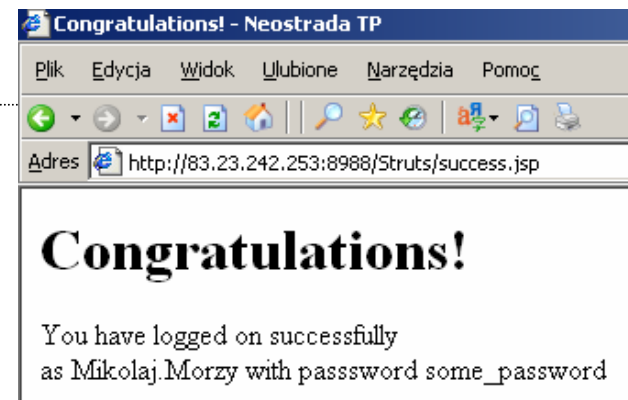
struts-config.jsp

Prosty przykład aplikacji (11/16)

- Wykorzystanie komunikatów ze zbioru zasobów za pomocą znacznika `<bean:message>`

success.jsp

```
<%@ page contentType="text/html; charset=windows-1252"%>
<%@ taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
    <title><bean:message key="form.title"/></title>
  </head>
  <body>
    <h1><bean:message key="form.confirmation"/></h1>
    <bean:message key="form.message"/><br>
    as <bean:write name="UserDataBean" property="username"/>
    with password <bean:write name="UserDataBean" property="password"/>
  </body>
</html>
```



Wielojęzyczność

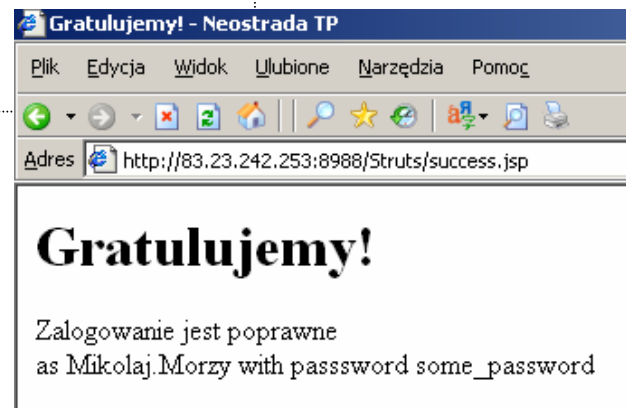
- Wykorzystanie plików ze zbiorami zasobów pozwala na łatwą obsługę wielojęzyczności
 - odwołanie `<message-resources parameter="someFile"/>` dotyczy w rzeczywistości pliku `/WEB-INF/classes/someFile.properties`
 - na podstawie ustawień przeglądarki określany jest preferowany język
 - Struts poszukuje lokalizowanych zbiorów komunikatów, np. `/WEB-INF/classes/someFile_pl.properties`
 - znalezione wpisy lokalizowane nadpisują klucze domyślne
 - zmiana lokalizacji może nastąpić programowo w klasie `Action` przez wywołanie metody `setLocale()`

Prosty przykład aplikacji (12/16)

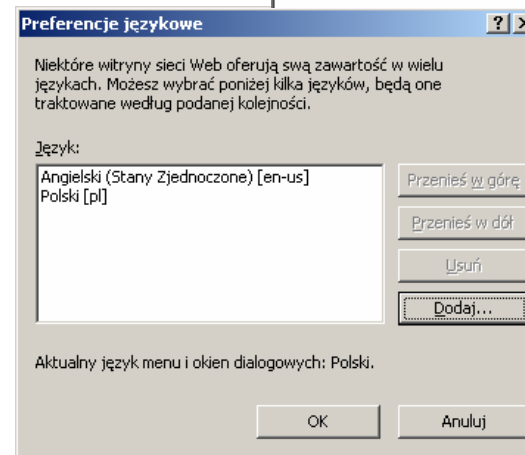
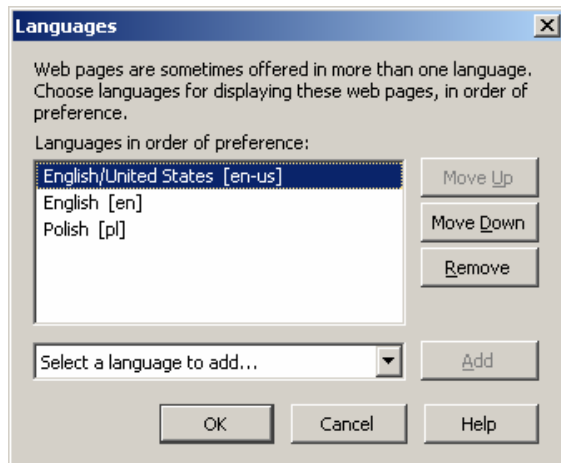
- Przygotowanie pliku z polską lokalizacją komunikatów

```
form.title = Sukces
form.confirmation = Gratulujemy!
form.message = Zalogowanie jest poprawnie
```

ApplicationResources_pl.properties



- Konfiguracja przeglądarki

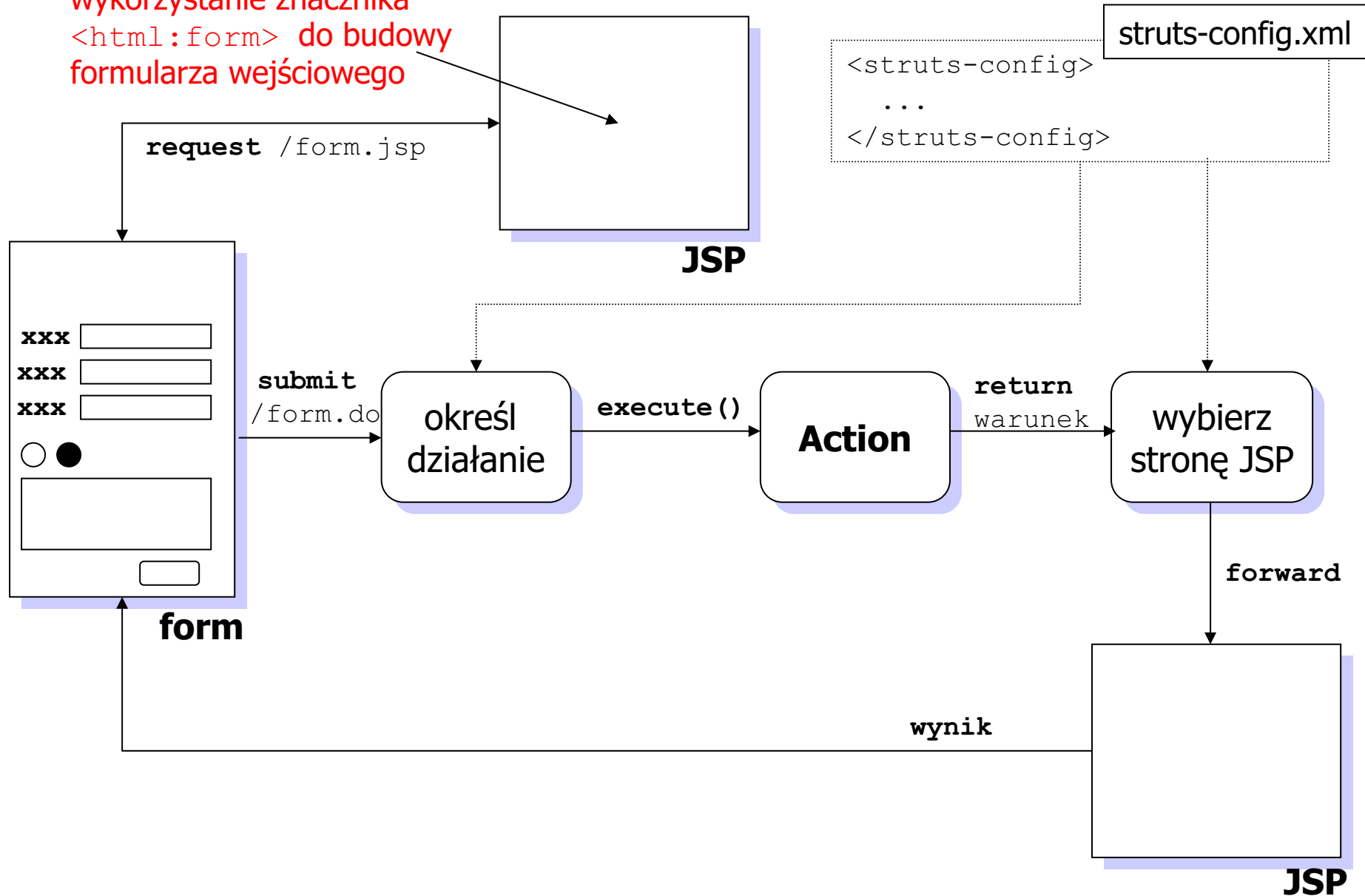


Firefox: Tools, Options, Languages

IE: Tools, Internet Options, Languages

Przeływ sterowania w Struts

wykorzystanie znacznika
`<html:form>` do budowy
 formularza wejściowego



Prosty przykład aplikacji (13/16)

- Formularz HTML zbudowany przy pomocy znacznika `<html:form>`

form.jsp

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<%@ page contentType="text/html; charset=windows-1252"%>
<%@ taglib uri="http://jakarta.apache.org/struts/tags-html" prefix="html"%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=windows-1252"></meta>
    <title>Login form</title>
  </head>
  <body>
    <h1>Login form</h1>
    <html:form action="/Login">
      <table cellspacing="1" cellpadding="5" border="0">
        <tr><td>username</td><td><html:text name="username"/></td></tr>
        <tr><td>password</td><td><html:password name="password"/></td></tr>
        <tr><td colspan="2" align="center">
          <html:submit value="send"/>
          <html:reset value="clear"/>
        </td>
      </tr>
    </table>
  </html:form>
</body>
</html>
```

przedrostek z nazwą aplikacji jest dołączany automatycznie, podobnie jak przyrostek **.do** domyślnie formularz wysyłany metodą POST

Wykorzystanie `<html:form>`

- Znacznik `<html:form>` pozwala na:
 - związanie formularza HTML z komponentem Form Bean,
 - zachowanie pełnej synchronizacji między nazwami parametrów wejściowych i składowymi komponentu Form Bean,
 - inicjalizację wartości pól formularza przez aplikację.
- Wartość atrybutu `action` znacznika `<html:form>` musi być zarejestrowana w pliku `struts-config.xml`, na tej podstawie identyfikowany jest właściwy komponent Form Bean obsługujący dany formularz.

Ponowne wyświetlenie formularza

- W przypadku wprowadzenia niekompletnych lub niepoprawnych danych formularz może być łatwo wyświetlony powtórnie z zachowaniem wcześniej wprowadzonych pól:
 - przekierowanie warunku w struts-config.xml na formularz
 - wysłanie przekierowania do przeglądarki
 - do komponentu Form Bean można dodać składowe służące do informowania użytkownika o popełnionych błędach. Zawartości składowych można odczytywać standardowo przez `<bean:write>` lub wyrażenia EL.

Prosty przykład aplikacji (14/16)

- Modyfikacja pliku `struts-config.xml`

struts-config.jsp

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE struts-config PUBLIC "-//Apache Software Foundation//DTD
  Struts Configuration 1.1//EN"
  "http://jakarta.apache.org/struts/dtds/struts-config_1_1.dtd">

<struts-config>
  <form-beans>
    <form-bean name="LoginFormBean" type="pl.org.ploug.struts"/>
  </form-beans>

  <action-mappings>
    <action path="/Login" type="pl.org.ploug.struts.LoginAction"
      name="LoginFormBean" scope="session">
      <forward name="success" path="/success.jsp"/>
      <forward name="missing" path="/form.jsp" redirect="true"/>
    </action>
  </action-mappings>
</struts-config>
```

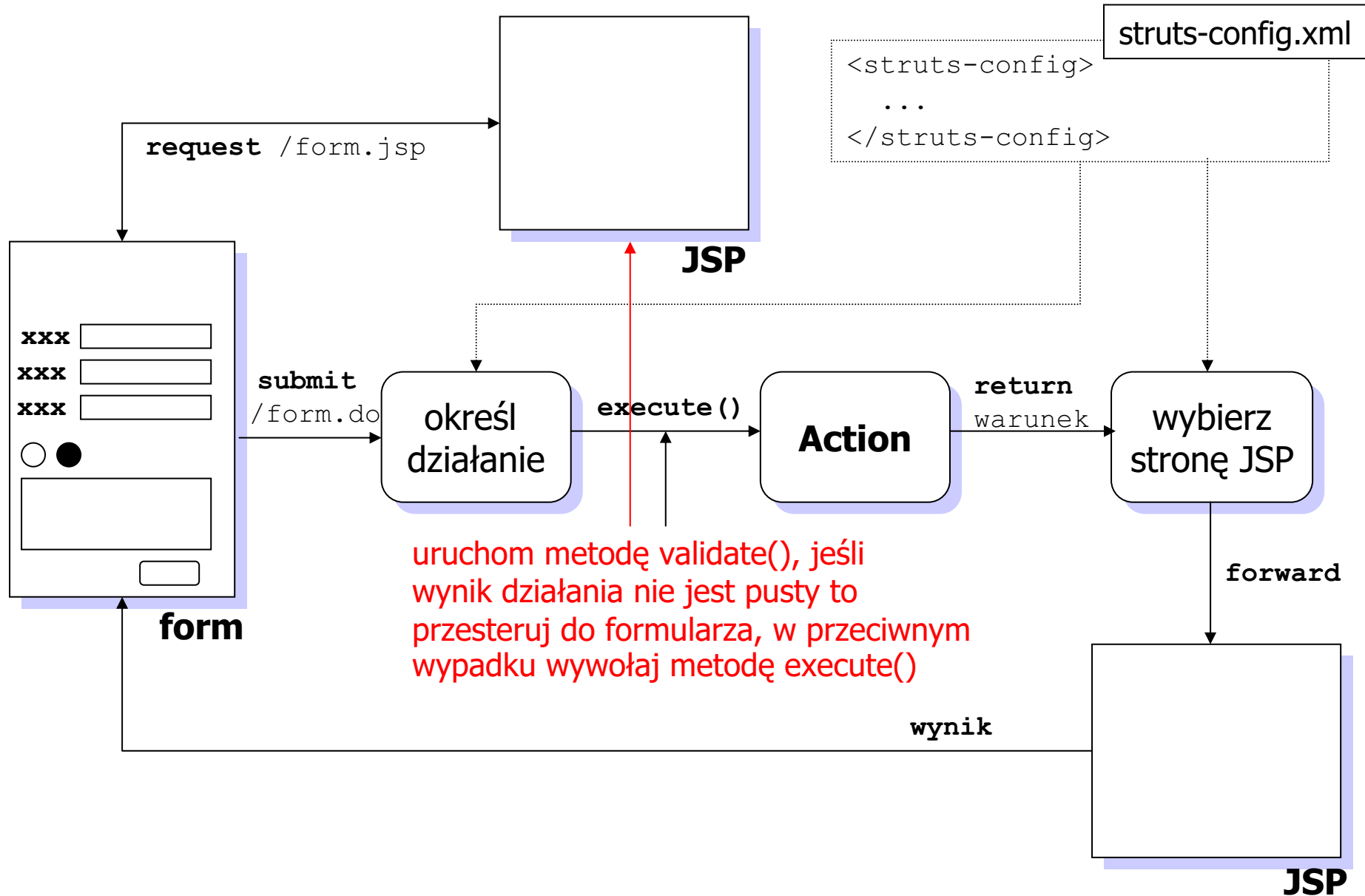
↑
warunek reprezentuje
odwzorowanie na stronę
z formularzem HTML

← zachowanie URL

Walidacja danych z formularza

- Etapy przetwarzania na których możliwa jest walidacja:
 - klasa **Action**: pełen dostęp do logiki biznesowej, źródeł danych, walidacja podobnych danych musi być powtarzana niezależnie w każdej klasie **Action**, odwzorowanie wyniku walidacji na właściwą stronę JSP odbywa się ręcznie, reguły walidacji muszą być ręcznie oprogramowane
 - klasa **ActionForm**: możliwa manipulacja wartościami składowych na poziomie poszczególnych metod *setter*, metoda `validate()` umożliwia ponowne wyświetlenie formularza i jest wspólna dla wszystkich sposobów wykorzystania komponentu Form Bean, reguły walidacji muszą być ręcznie oprogramowane
 - automatyczny walidator: Struts oferuje mechanizm automatycznej walidacji podstawowych danych przy wykorzystaniu JavaScript

Przeływ sterowania w Struts



Prosty przykład aplikacji (15/16)

- Dodanie metody `validate()` do komponentu Form Bean

LoginFormBean.java

```
package pl.org.ploug.struts;

import javax.servlet.http.*;
import org.apache.struts.action.*;

public class LoginFormBean extends ActionForm {
    ...
    public ActionErrors validate(ActionMapping mapping, HttpServletRequest req) {
        ActionErrors errors = new ActionErrors();

        if (this.getEmail() == null) {
            errors.add("email", new ActionError("email.required"));
        }
        if (this.getPassword() == null) {
            errors.add("password", new ActionError("password.required"));
        }
        return (errors);
    }
    ...
}
```

Prosty przykład aplikacji (16/16)

- Dodanie pliku z komunikatami o błędach i rejestracja pliku w `struts-config.xml`, wyświetlenie komunikatów za pomocą znacznika `<html:errors/>`

```
errors.header=<UL>
errors.prefix=<LI><B><FONT COLOR="RED">
errors.suffix=</FONT></B></LI>
errors.footer=</UL>
...
```

ApplicationResources.properties

...

struts-config.jsp

```
<action-mappings>
  <action path="/Login" type="pl.org.ploug.struts.LoginAction"
    name="LoginFormBean" scope="session" input="form.jsp">
    <forward name="success" path="/success.jsp"/>
    <forward name="missing" path="/form.jsp" redirect="true"/>
  </action>
</action-mappings>
</struts-config>
```

```
<body>
  <h1>Login form</h1>
  <html:errors/>
  <html:form action="/Login">
    ...
  </html:form>
</body>
```

form.jsp

Automatyczna walidacja formularza

- Automatyczna walidacja ujednolica kod służący do sprawdzania poprawności danych wejściowych i umożliwia korzystanie ze standardowych reguł walidacji.
- Walidacja po stronie klienta
 - kod JavaScript weryfikuje poprawność wprowadzonych danych, błędy sygnalizowane oknami dialogowymi, wysłanie formularza wstrzymane do momentu poprawienia błędów
 - walidacja nie może implementować złożonej logiki i może zostać wyłączona (celowo bądź przypadkowo)
- Walidacja po stronie serwera
 - kod Java weryfikuje poprawność wprowadzonych danych, w przypadku błędów formularz jest ponownie wyświetlony, tryb walidacji po stronie serwera jest obowiązkowy

Automatyczna walidacja (1/4)

- Zmiany w pliku konfiguracyjnym `struts-config.xml`
 - włączenie automatycznego walidatora

struts-config.jsp

```
...
<form-beans>
  <form-bean name="LoginFormBean" type="pl.org.ploug.struts"/>
</form-beans>

<action-mappings>
  <action path="/Login" type="pl.org.ploug.struts.LoginAction"
    name="LoginFormBean" scope="session" input="form.jsp">
    <forward name="success" path="/success.jsp"/>
  </action>
</action-mappings>

<message-resources parameter="MessageResources"/>

<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
</struts-config>
```

Automatyczna walidacja (2/4)

- Przygotowanie pliku z komunikatami wykorzystywanymi przez automatyczny walidator

```
# -- standardowe wyświetlanie błędów --
errors.header=<UL>
errors.prefix=<LI><B><FONT COLOR="RED">
errors.suffix=</FONT></B></LI>
errors.footer=</UL>

# -- komunikaty walidatora --
errors.invalid={0} is invalid.
errors.maxlength={0} cannot be greater than {1} characters.
errors.minlength={0} cannot be less than {1} characters.
errors.range={0} is not in the range {1} through {2}.
errors.required={0} is required.
```

MessageResources.properties

komunikaty o błędach mogą być parametryzowane

Automatyczna walidacja (3/4)

- Przygotowanie definicji reguł walidacji

validator.xml

```
<form-validation>
```

```
  <global>
```

```
    <constant>
```

```
      <constant-name>postalCode</constant-name>
```

```
      <constant-value>^\d{5}\d*$</constant-value>
```

```
    </constant>
```

```
  </global>
```

```
  <formset>
```

```
    <form name="LoginFormBean">
```

```
      <field property="username" depends="required">
```

```
        <arg0 key="LoginFormBean.username"/>
```

```
      </field>
```

```
      <field property="password" depends="required,mask">
```

```
        <arg0 key="LoginFormBean.password"/>
```

```
        <var>
```

```
          <var-name>mask</var-name>
```

```
          <var-value>^[0-9a-zA-Z]*$</var-value>
```

```
        </var>
```

```
      </field>
```

```
    </form>
```

```
  </formset>
```

```
</form-validation>
```

globalny format kodu pocztowego

nazwa reguły
z validator-rules.xml

parametr komunikatu
o błędzie

Automatyczna walidacja (4/4)

- Zmiana definicji komponentu Form Bean
 - dziedziczy z `org.apache.struts.validator.ValidatorForm`

LoginFormBean.java

```
package pl.org.ploug.struts;
```

```
import javax.servlet.http.*;
```

```
import org.apache.struts.action.*;
```

```
public class LoginFormBean extends ValidatorForm {
```

```
    private String username = "scott";
```

```
    private String password = "tiger";
```

```
    public String getUsername() { return this.username; }
```

```
    public void setUsername(String username) { this.username = username; }
```

```
    public String getPassword() { return this.password; }
```

```
    public void setPassword(String password) { this.password = password; }
```

```
}
```

klasa musi rozszerzać klasę **ValidatorForm**



Automatyczny walidator - możliwości

- Automatyczny walidator umożliwia weryfikację poprawności formatu wielu typów danych
 - mask
 - intRange, floatRange, doubleRange, maxlength, minlength
 - integer, float, double, long, short, byte
 - date
 - email
 - creditCard
 - url
 - ...
- Spis reguł można znaleźć w pliku `validator-rules.xml`

Obsługa wyjątków

- Struts umożliwia scentralizowanie obsługi typowych wyjątków występujących w aplikacji. W tym celu programista musi stworzyć klasę obsługi wyjątku która rozszerza klasę `org.apache.struts.action.ExceptionHandler`, a następnie zadeklarować klasę obsługi wyjątku w `struts-config.xml`

IOExceptionHandler.java

```
...
public class IOExceptionHandler extends ExceptionHandler {
    public ActionForward execute(
        Exception e, ExceptionConfig ec, ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) {
        // tutaj obsługa wyjątku
        return mapping.findForward("after-error");
    }
}
```

struts-config.jsp

```
...
<global-exceptions>
    <exception key="exception.any" type="java.io.IOException"
        handler="pl.org.ploug.struts.IOExceptionHandler"/>
</global-exceptions>
...
```

Struts w JDeveloper 10.1.3

Oracle JDeveloper 10g - SzkolaPLOUG2006.jws : Struts.jpr

File Edit View Search Navigate Run Debug Refactor Versigning Tools Window Help

Applications Connections Help struts-config.xml

Applications

- extensionsdk
- SRDemoSample
- SzkolaPLOUG2006
 - JDO
 - Spring
 - Struts
 - Application Sources
 - pl.org.ploug
 - szkola2006
 - struts
 - Resources
 - Web Content
 - css
 - WEB-INF
 - struts-config.xml
 - validation.xml
 - validator-rules.xml
 - web.xml
 - WEB-INF\classes\
 - .jspx
 - .tags
 - .pl
 - WEB-INF\lib
 - WEB-INF\src\pl
 - bad-address.jsp
 - bad-password.jsp
 - form.jsp
 - login.jsp
 - missing-data.jsp
 - register.jsp
 - simpleDataPage.jsp
 - success.jsp
 - summary.jsp

Diagram Source History

Struts Page Flow

- Pointer
- Page
- Data Page
- Action
- Data Action
- Forward
- Page Link
- Page Forward
- Switch Action
- Note
- Attachment

Diagram illustrating the Struts Page Flow:

```

graph TD
    RegisterData[Register Data] -- missing-data --> RegisterJSP[/register.jsp/]
    RegisterData -- success --> SuccessJSP[/success.jsp/]
    RegisterData -- bad-address --> BadAddressJSP[/bad-address.jsp/]
    RegisterData -- bad-password --> BadPasswordJSP[/bad-password.jsp/]
    RegisterData -- /old-missing-data --> MissingDataJSP[/missing-data.jsp/]
    LoginJSP[/login.jsp/] -.-> Login[Login]
    Login -- success --> SummaryJSP[/summary.jsp/]
  
```

Page Flow: 42MB of 77MB