Tutorial Spring Framework – moduł WebFlow

Autor: Andrzej Klusiewicz

klusiewicz@jsystems.pl

Spis treści

Tutorial Spring Framework – moduł WebFlow....1
Podstawy Spring WebFlow....3
Proste formularze z przejściami....3
Zaawansowane elementy Spring WebFlow....16
Stany decyzyjne....17
Wyświetlanie danych....21
Wieloetapowe uzupełnianie obiektu podczas przepływu....23
Przejście globalne....33
Podprzepływy....35

Podstawy Spring WebFlow

Proste formularze z przejściami

Kod źródłowy z przykładami do tego rozdziału możesz pobrać pod adresem:

http://jsystems.pl/storage/spring/springflow1.zip

Do czego może nam się przydać Spring WebFlow? Ten moduł umożliwia robienie wieloetapowych przepływów – coś na kształt wizardów instalacyjnych. Możemy dzięki niemu w stosunkowo prosty sposób stworzyć np. kilkustronnicowy formularz rejestracyjny.

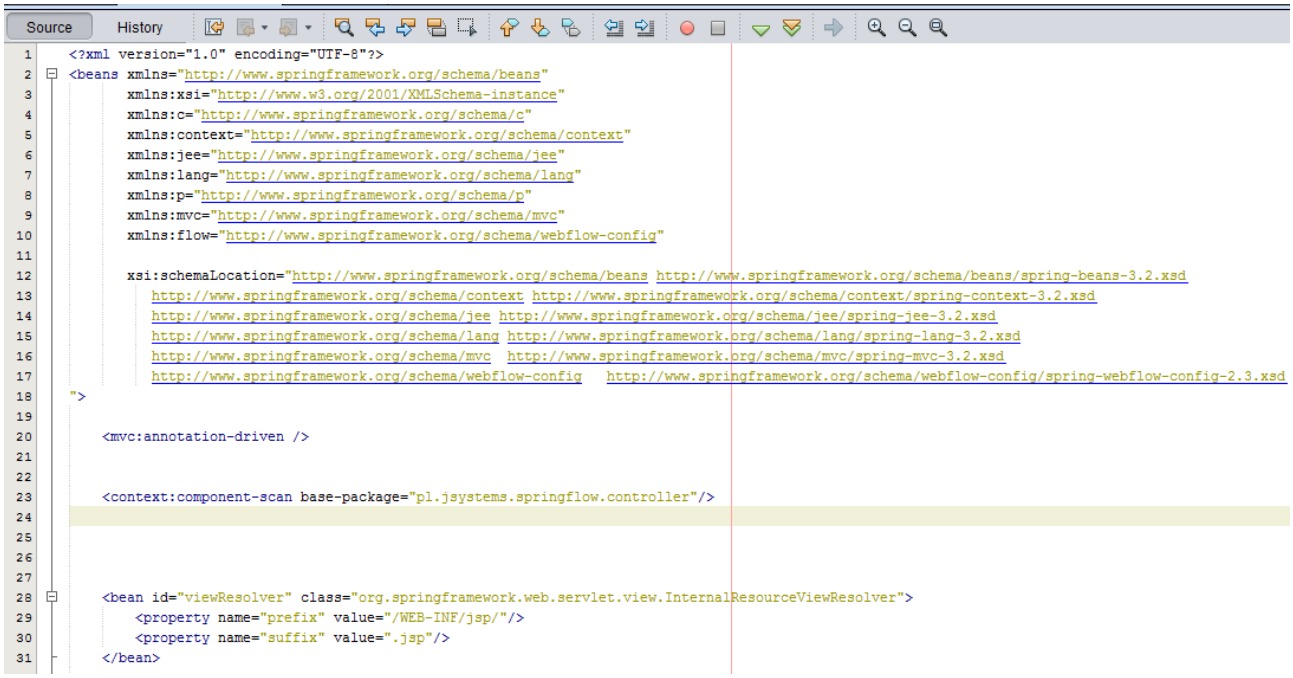
W tym rozdziale rozpoczniemy od dodania dwóch bardzo prostych funkcjonalności na potrzeby sklepu internetowego. Jedna będzie pozwalała zapisać się do newslettera, druga założyć konto w sklepie. Zaczynamy od stworzenia zwyczajnej aplikacji opartej o Spring MVC. Nie pojawia się tutaj nic nowego ani nadzwyczajnego, niemniej zaznaczę kilka kluczowych elementów:

Wpis w web.xml:

```
5
6
7       <servlet>
8           <servlet-name>sklep</servlet-name>
9           <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
10      </servlet>
11      <servlet-mapping>
12          <servlet-name>sklep</servlet-name>
13          <url-pattern>*.do</url-pattern>
14      </servlet-mapping>
15
```

Wszystkie podstrony tej aplikacji z rozszerzeniem .do będą obsługiwane przez Springa.

Do pliku *****-servlet.xml trzeba będzie dodać przestrzenie nazw związane ze Spring WebFlow, więc zrobimy to od razu:



a konkretniej chodzi mi o dodanie przestrzeni nazw:

xmlns:flow="http://www.springframework.org/schema/webflow-config"

oraz jej położenia:

http://www.springframework.org/schema/webflow-config
http://www.springframework.org/schema/webflow-config/spring-webflow-config-2.3.xsd

WebFlow wymaga dodatkowych bibliotek, więc poza tym co zwykle dodać trzeba jeszcze będzie biblioteki spring-webflow,spring-binding i springframework.js:

Wszystkie trzy pliki jar znajdują się z katalogu libs projektu dołączonego do niniejszego artykułu.

Dodaję też najzwyklejszy w świecie kontroler z jedną metodą mapującą adres „/start.do” :

```
12
13  /**
14   *
15   * @author andrzej
16   */
17  @Controller
18  public class Startowa {
19
20      @RequestMapping(value="/start.do")
    public String startowa(Model model){
22          System.out.println("Ktoś wszedł na stronę startową...");
23          return "startowa";
24      }
25  }
26
```

Do pliku JSP obsługującego to żądanie dodałem dwa linki:

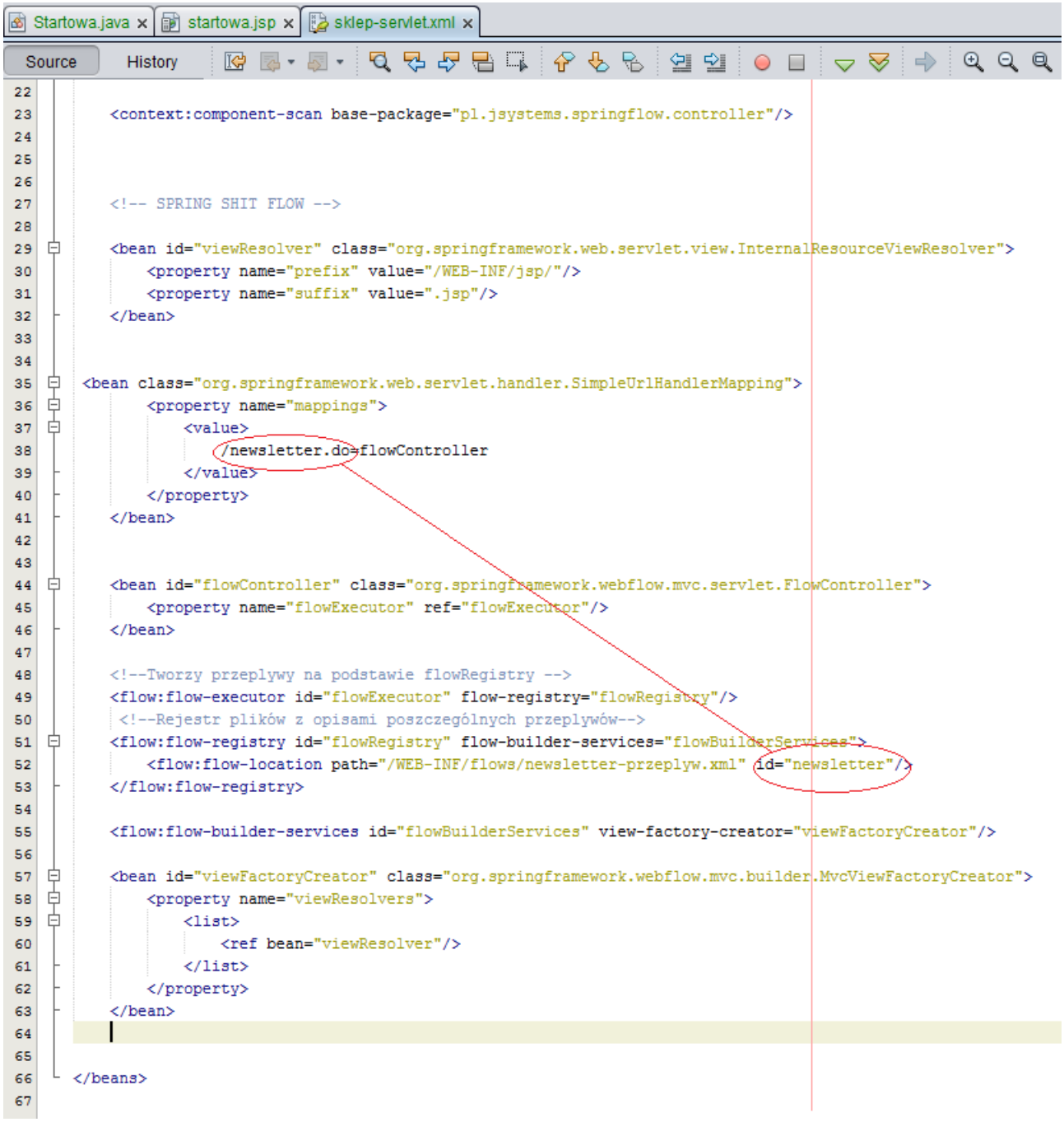
```
15
16          <h3>Witamy w naszym sklepie </h3>
17          <br>
18          <a href="newsletter.do">Zapisz się do newslettera!</a>
19          <br>
20          <a href="rejestracja.do">Rejestracja konta</a>
21
```

I teraz się zacznie... W pliku ****-servlet.xml musimy dodać kilka niezbędnych elementów. Na potrzeby choćby jednego przepływu musimy mieć całą taką konfigurację jaką wprowadziłem w swoim pliku w liniach 29-63... Omówimy elementy zmienne, które będą się różnić w różnych aplikacjach.

Dodamy na początek przepływ który będzie się sprowadzał do dodania swojego adresu email, do newslettera. Nieco później rozbudujemy nasz projekt.

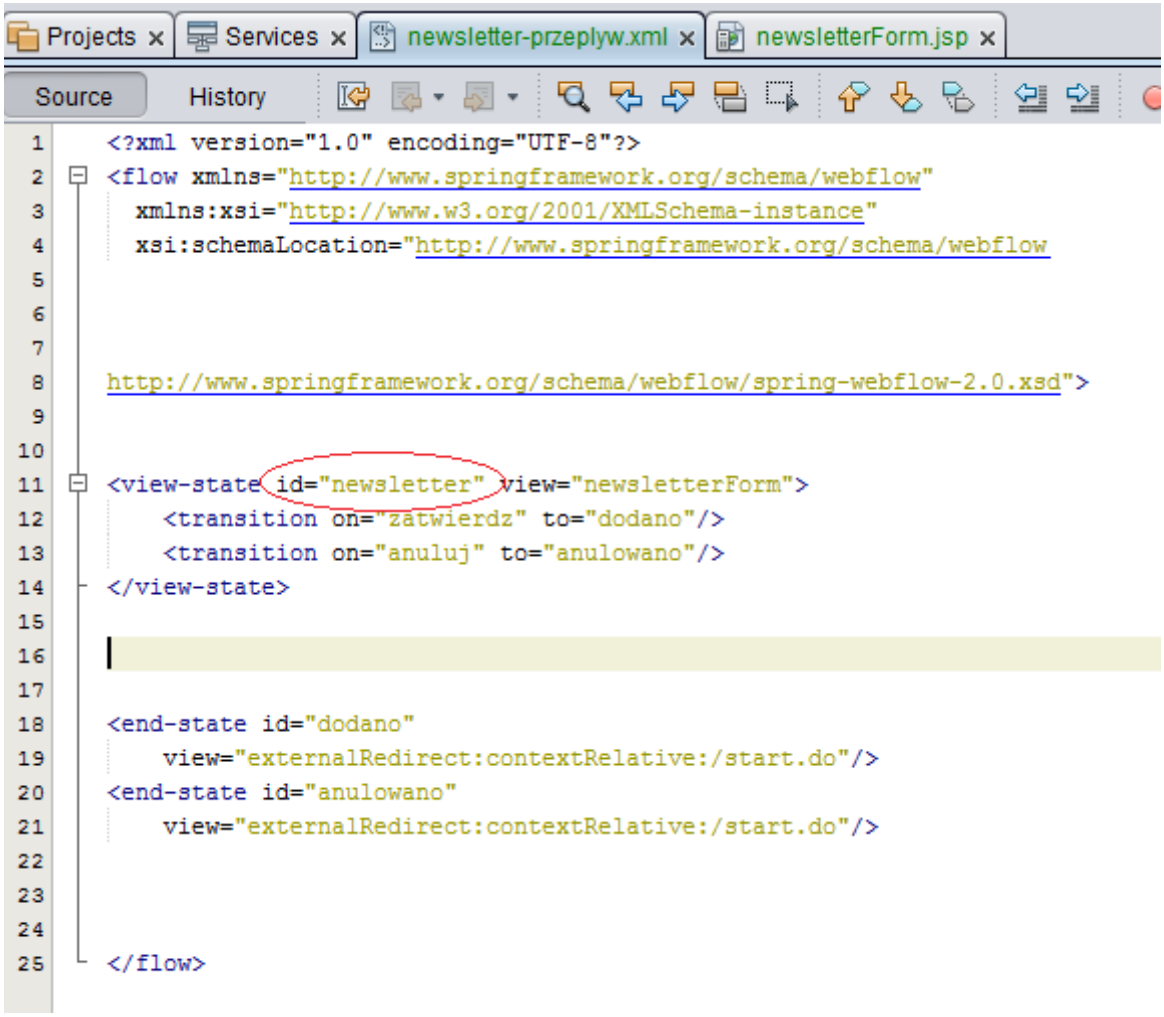
Każdy przepływ będzie miał osobny plik konfiguracyjny w którym opiszemy co po czym ma się uruchamiać i przy spełnieniu jakich warunków. Rejestrujemy te pliki konfiguracyjne w sekcji flow-registry (linie 51-53). Bardzo ważny jest tutaj parametr ID, ponieważ jest on związany z tak zwanym adresem wyzwalającym przepływ. Jeśli adres naszego formularza jest /newsletter.do, to WebFlow będzie poszukiwał w rejestrze wpisu o id newsletter, jeśli adres rejestracji konta w sklepie będzie miał postać /rejestracja.do to będzie szukał wpisu o id rejestracja. Liczy się tylko „końcówka”, same adresy wyzwalające mogą mieć postać np. /modul1/newsletter.do.

Ten adres wyzwalający oznacza rozpoczęcie przepływu – w tym przypadku rejestracji do newslettera.



Jak widać w rejestrze przepływów dla przepływu o id „newsletter” wskazałem plik newsletter-przeplyw.xml znajdujący się w katalogu /WEB-INF/flows. Tworzymy więc podkatalog flows w WEB-INF i umieszczamy w nim na razie pusty plik o nazwie newsletter-przeplyw.xml

Wewnątrz tego pliku umieszczamy taką konfigurację:

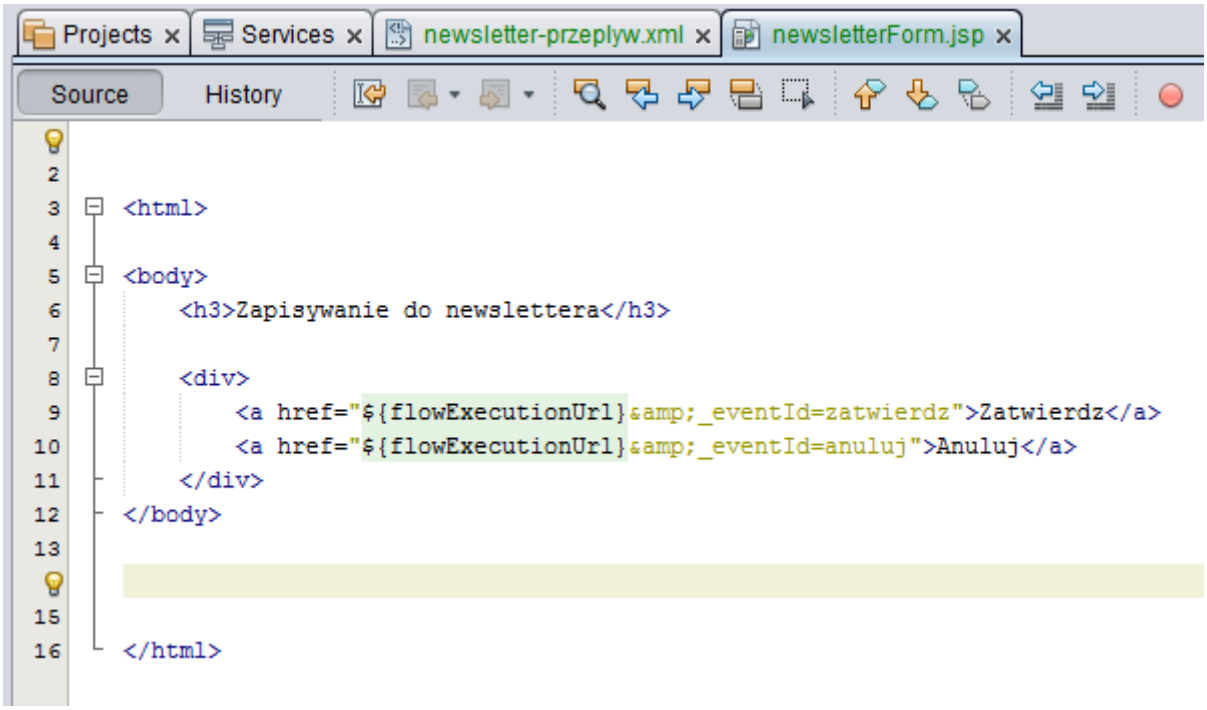


Każdy element zamieszczony tutaj to jeden „krok” w naszym przepływie. Takim krokiem może być wyświetlenie strony, wykonanie jakiejś akcji, podjęcie decyzji warunkowej. Każdy przepływ musi mieć dokładnie jeden stan początkowy i przynajmniej jeden końcowy. Oznaczają one pierwszy i ostatni krok w przepływie. Pomiędzy tymi krokami możemy dodać kolejne, nic nie stoi na przeszkodzie. W naszym mini projekcie w tym przepływie będą tylko dwa kroki. Jeden to wyświetlenie formularza do którego wprowadzimy nasz adres email, drugi to stan końcowy który po prostu przekieruje nas do strony początkowej. Nieco później zadbamy o to, by w zależności od tego czy zatwierdzimy czy anulujemy formularz wyświetlił się stosowny komunikat.

Przyjrzyjmy sięteraz elementowi w liniach 11-14. Oznacza on wyświetlenie strony jsp o nazwie newsletterForm (która powinna znaleźć się w WEB-INF/jsp – jak mamy skonfigurowane w beanie o ID viewResolver w pliku ****-servlet.xml). Zauważ, że element ten ma również id „newsletter”. To jest stan początkowy, co oznacza że jako pierwszy krok w tym przepływie zostanie wyświetlona strona newsletterForm. Pojawiają nam się tutaj też znaczniki <transition on =”x” to=”y”/> Co to oznacza? Jeśli pojawi się (na razie bliżej nie określone) „COŚ” o nazwie „zatwierdz”, sterowanie ma zostać przekierowane do elementu „dodano”. Jeśli pojawi się takie samo „COŚ” o nazwie „anuluj”, sterowanie ma zostać przekierowane do elementu o nazwie „anulowano”.

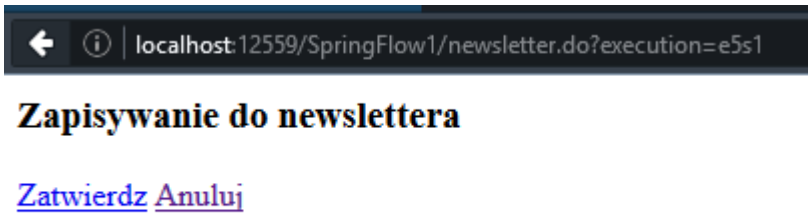
To „COŚ” to są komunikaty wysyłane z poziomu widoku i za chwilę się tym zajmiemy. W tej chwili przyjrzyjmy się jeszcze elementom z linii 18-21. Są to dwa możliwe stany końcowe. Oba w tej chwili robią to samo – przekierowują do strony początkowej. Który ze stanów końcowych nastąpi zależy od tego, czy z widoku zostanie przesłany komunikat „zatwiedz” czy „anuluj”.

Przejdźmy teraz do naszego pliku widoku „newsletterForm” który jest wyświetlany w pierwszym kroku naszego przepływu. Niedługo pojawią się tutaj też pola do których wprowadzać będziemy dane, teraz interesuje nas tylko przepływ jako taki.



Może zastanawiać dziwny adres linka który podałem. Zacznijmy od lewej. Pojawia się tutaj

${flowExecutionUrl} - dzięki temu fragmentowi powracamy do przepływu w którego trakcie jesteśmy. Ten znacznik widoczny jest zresztą w pasku adresu:



Dalej mamy tajemniczy ciąg zawierający **_eventId=zatwierdz ,** oraz drugi podobny **_eventId=anuluj** . Myślę że kiedy przyjrzysz się zaznaczonemu poniżej fragmentowi pliku konfiguracyjnego tego przepływu to wszystko stanie się jasne:

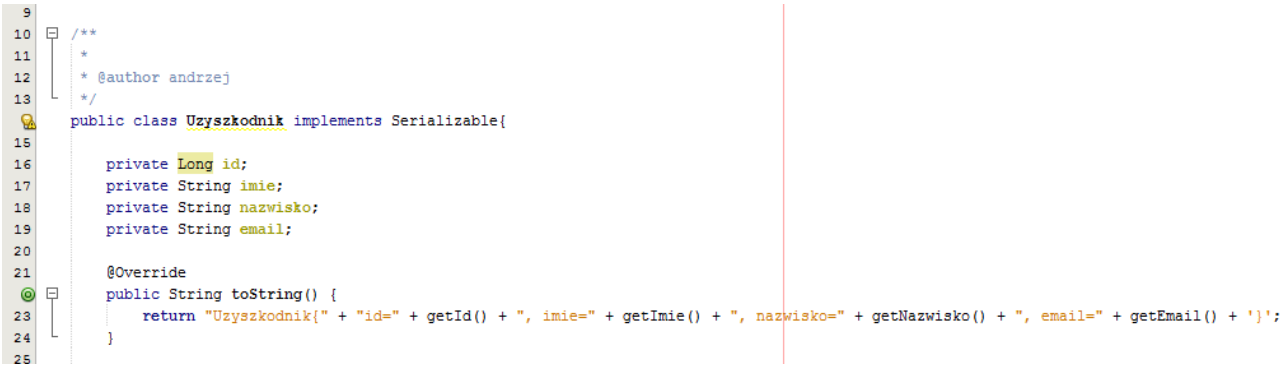
```
10
11  <view-state id="newsletter" view="newsletterForm">
12      <transition on="zatwierdz" to="dodano"/>
13      <transition on="anuluj" to="anulowano"/>
14  </view-state>
15
```

Gdybyś jednak nie wypił porannej mocnej kawy to wyjaśniam – to jest to nasze „COŚ" które wywołane powoduje przejście do kolejnego stanu. Ale jakiego stanu? W pierwszym przypadku do stanu o nazwie „dodano", w drugim do stanu o nazwie „anulowano". A teraz przyjrzyj się znajdującemu się nieco niżej fragmentowi pliku newsletter-przeplyw.xml:

```
16
17
18  <end-state id="dodano"
19      view="externalRedirect:contextRelative:/start.do"/>
20  <end-state id="anulowano"
21      view="externalRedirect:contextRelative:/start.do"/>
22
23
```

Kliknięcie tych linków spowoduje przejście do stanów końcowych przepływu. Oczywiście można by tutaj było dać jakiś inny stan – choćby kolejny widok. Wszystko fajno, ale trochę słabe takie zapisywanie się do newslettera, jeśli nigdzie nie podajemy swojego adresu email ;)

Wzbogaćmy teraz naszą aplikację o obiekt w którym będziemy gromadzili dane na temat użytkownika. Uzyszkownik będzie obiektem klasy którą musimy stworzyć:



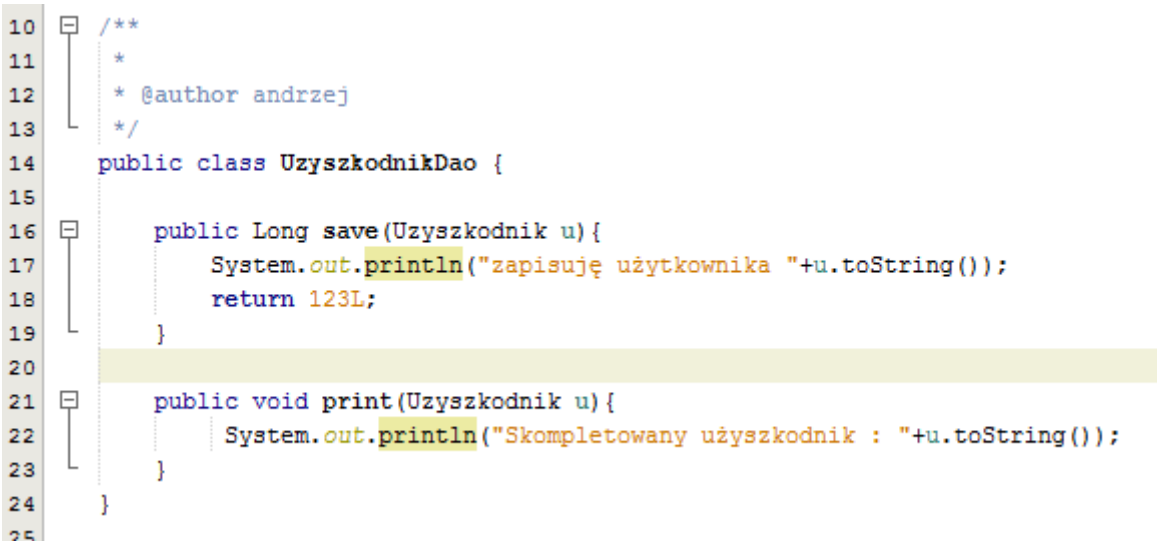
```
/**
 *
 * @author andrzej
 */
public class Uzyszkodnik implements Serializable{

    private Long id;
    private String imie;
    private String nazwisko;
    private String email;

    @Override
    public String toString() {
        return "Uzyszkodnik{" + "id=" + getId() + ", imie=" + getImie() + ", nazwisko=" + getNazwisko() + ", email=" + getEmail() + '}';
    }
```

Moja klasa zawiera cztery proste pola . Nie są ujęte na screenie, ale klasa posiada też gettery i settery do tych pól. Klasa musi być serializowalna.

Dodaję też jakieś fake'owe dao do obiektów klasy Uzyszkodnik. Metoda save ma przyjmować obiekt użytkownika po wypełnieniu formularza i zwracać ID pod jakim obiekt ten wylądował w bazie. Tutaj jest to jakaś wartość „na sztywno”, abyśmy teraz nie komplikowali sobie życia sprawami związanymi z bazami danych. Jest też metoda print która przyjmuje przez parametr obiekt użytkownika i wypisuje jego zawartość na konsolę. Oczywiście w żadnym normalnym DAO taka metoda by się nie pojawiała, tutaj jest jedynie do celów prezentacyjnych działania przepływów.



```
/**
 *
 * @author andrzej
 */
public class UzyszkodnikDao {

    public Long save(Uzyszkodnik u){
        System.out.println("zapisuję użytkownika "+u.toString());
        return 123L;
    }

    public void print(Uzyszkodnik u){
         System.out.println("Skompletowany użyszkodnik : "+u.toString());
    }
}
```

Ponieważ do obiektu klasy UzyszkodnikDao zamierzam się odwoływać w ramach mojego przepływu na zasadzie „uzyszkodnikDao.jakasMetoda()” w pliku konfiguracji przepływu, Spring musi wiedzieć co to takiego jest ten uzyszkodnikDao. W związku z tym deklarujemy tę klasę jako bean w pliku ***-servlet.xml:

```
<bean id="uzytkownikDao" class="pl.jsystems.springflow.dao.UzyszkodnikDao"/>
```

Przejdźmy teraz do pliku konfiguracji przepływu. Pojawiło się tutaj kilka nowych rzeczy:

sklep-servlet.xml | newsletter-przeplyw.xml | newsletterForm.jsp | Uzyszkodnik.java | Uzyszkodni

Source | History

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <flow xmlns="http://www.springframework.org/schema/webflow"
3    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4    xsi:schemaLocation="http://www.springframework.org/schema/webflow
5
6
7
8  http://www.springframework.org/schema/webflow/spring-webflow-2.0.xsd">
9
10 <var name="uzyszkodnik" class="pl.jsystems.springflow.model.Uzyszkodnik"/>
11
12 <view-state id="newsletter" view="newsletterForm" model="uzyszkodnik">
13     <transition on="zatwierdz" to="zapiszUzytkownika"/>
14     <transition on="anuluj" to="anulowano"/>
15 </view-state>
16
17 <action-state id="zapiszUzytkownika">
18     <evaluate expression="uzytkownikDao.save(uzyszkodnik)" result="uzyszkodnik.id"   />
19     <transition to="wydrukujUzytkownika" />
20 </action-state>
21
22 <action-state id="wydrukujUzytkownika">
23     <evaluate expression="uzytkownikDao.print(uzyszkodnik)" />
24     <transition to="dodano" />
25 </action-state>
26
27 <end-state id="dodano"
28     view="externalRedirect:contextRelative:/start.do"/>
29 <end-state id="anulowano"
30     view="externalRedirect:contextRelative:/start.do"/>
31
32
33
34 </flow>
```

1.
2.
3.
4.

Element oznaczony jako nr 1 to zmienna która widoczna będzie w całym przepływie. Spring stworzy obiekt o nazwie podanej w parametrze „name” klasy podanej w parametrze „class” i umieści ją w przepływie. Nasz widoczek newsletterForm zyskał nowego kolegę pod postacią parametru model. Zauważ że jest tutaj podana nazwa naszej nowej zmiennej – uzyszkodnik. W formularzu będziemy uzupełniać pola obiektu – więc wypadałoby wskazać jakiego :). Bez tego dodatku w prawdzie przepływ będzie działał (w sensie będą działały przejścia i nie będzie sypało żadnymi błędami), ale wypełniane pola pójdą „w kosmos” a obiekt na koniec przepływu pozostanie pusty (sprawdzone organoleptycznie). W linii 13 też jest mała zmiana – przejście następuje nie do końcowego stanu a do stanu „zapiszUzytkownika” który to się właśnie pojawił. Element action-state oznacza wykonanie jakiejś akcji – wywołania metody , a nie jak view-state wyświetlenia widoku. Przyjrzyjmy się więc teraz temu elementowi. „Evaluate expression” wskazuje metodę która ma zostać wykonana. Jest to metoda „save” z obiektu klasy uzytkownikDao któgo bean przed momentem dodawaliśmy do pliku ****-servlet.xml . Do tej metody jako parametr przekazuje obiekt uzyszkodnik który.... też przed momentem definiowaliśmy ;). Chodzi więc o obiekt który sobie w ramach naszego przepływu uzupełniamy. W naszym formularzu który jest uzupełniany stan temu uzupełnialiśmy pola właśnie tego obiektu. W tym kroku przekazujemy go do metody by zapisać go w bazie. Ten obiekt jest już uzupełniony wartościami wprowadzonymi przez formularz. Pojawia się tutaj tez parametr result. Obiekt który zapisujemy wyląduje w bazie pod jakimś identyfikatorem, a identyfikator ten zwracany jest przez metodę save. Przyda się on do późniejszego wyświetlenia, lub np. podpinania kolejnych obiektów pod ten. Result spowoduje podstawienie pod pole id naszego przepływowego obiektu wartości zwracanej przez metodę wywoływaną w evaluate expression. Action-state po wykonaniu oczekiwanych operacji przekazuje sterowanie do stanu „wydrukujUzytkownika”. Stan wydrukujUzytkownika to stan akcji który wywołuje stworzoną w dao metodę print i przekazuje jej uzupełniony obiekt do wyswietlenia. Dalej przepływ przekazywany jest do stanu końcowego dodano.

Podsumowując – pierwszy stan to stan widoku wyświetlający formularz z pliku newsletterForm.jsp, przy użyciu którego uzupełniamy obiekt uzyszkodnik który jest widoczny w całym przepływie. Po zatwierdzeniu formularza następuje stan akcji „zapiszUzytkownika” który zapisuje obiekt „uzyszkodnik” do bazy i uzupełnia w nim pole id identyfikatorem pod jakim wylądował w bazie. Sterowanie jest przekazywane do stanu „wydrukujUzytkownika” który na konsoli drukuje zawartość obiektu i przekazuje sterowanie do stanu końcowego, który przekierowuje do strony startowej.

Troszkę zmian nastąpiło też w samym formularzu do wypełniania danych.

```
1
2   <%@page pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>
3   <%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
4   <%@taglib prefix="spring" uri="http://www.springframework.org/tags" %>
5   <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
6   <html>
7
8       <body>
9           <h3>Zapisywanie do newslettera</h3>
10
11          <div>
12              <form:form commandName="uzyszkodnik">
13                  Imię: <form:input path="imie"/><br>
14                  Nazwisko: <form:input path="nazwisko"/><br>
15                  Email: <form:input path="email"/><br>
16
17                  <input type="submit" name="_eventId_zatwierdz" value="Zatwierdź"/>
18                  <input type="submit" name="_eventId_anuluj" value="Anuluj"/>
19
20
21                  <a href="${flowExecutionUrl}&amp;_eventId=zatwierdz">Zatwierdz</a>
22                  <a href="${flowExecutionUrl}&amp;_eventId=anuluj">Anuluj</a>
23
24
25              </form:form>
26          </div>
27      </body>
28
29
30
31  </html>
```

W liniach 2-5 pojawiły się importy do bibliotek tagów których używamy. Musisz je mieć, inaczej formularz nie zadziała. Elementy z linii 21 i 22 już znasz, nie będą nam więcej potrzebne ale zostawiłem je do porównania z elementami z linii 17 i 18 które je zastąpią. Zacznijmy jednak od początku – linia 12. <form:form.... z zasady oznacza początek formularza, </form:form> jego koniec. Wszystkie pola służące do wprowadzania danych muszą się znajdować pomiędzy nimi.

W tagu otwierającym formularz pojawia się parametr commandName=”uzyszkodnik”. Pewne podejrzenie odnośnie tego co to takiego, powinno już Ci się nasunąć samo. To jest wskazanie obiektu który będziemy uzupełniać, a ściślej obiektu klasy Uzytkownik który zdeklarowaliśmy w pliku konfiguracyjnym przepływu. Możemy tam przecież mieć kilka deklaracji <var..../>

<form:input path=”nazwaPola”/> odnosi się do pól tego obiektu. Krótko mówiąc są to pola tekstowe do uzupełnienia pól imię, nazwisko, email w obiekcie uzyszkodnik.

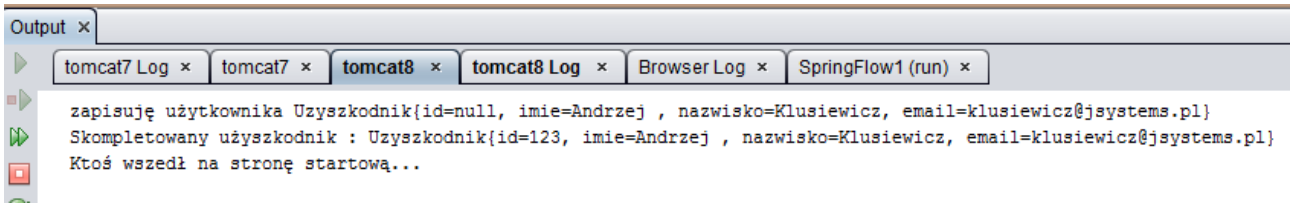
Linie 17 i 18 to przyciski. Jeden będzie zatwierdzał formularz, drugi anulował. Ściślej – oba będą wywoływały akcję którą zdeklarowaliśmy w pliku konfiguracji przepływu. Zwróć uwagę na ich nazwy. Aby spełniały założoną rolę, muszą składać się z elementów _eventId_ oraz naszego uruchamiacza przejścia do następnego stanu.

Jeśli przyjrzysz się teraz elementom „transition on …. to …” to myślę że nie trzeba będzie nic więcej wyjaśniać. Linki zrobione wcześniej (linia 21 i 22) można już usunąć.

Skoro już wszystkie elementy mamy, zobaczmy jak to działa. Rozpoczynamy przepływ wchodząc na stronę newsletter.do:

Jesteśmy na pierwszym stanie. Jest to stan widoku „newsletter”. Uzupełniam wszystkie pola:

Po zatwierdzeniu zostaje wywołane zdarzenie „zatwierdź” które powoduje przejście do stanu akcji „zapiszUzytkownika”. Zajrzyjmy teraz do konsoli:



Pierwsza linia pochodzi z dao wywołanego w stanie akcji „zapiszUzytkownika”. Jak widać wyświetlają się dane wprowadzone przez formularz, ale pole id pozostaje puste. Jest ono uzupełniane w stanie akcji „zapiszUzytkownika” . Następnie do akcji wkracza „wydrukujUzytkownika” który prezentuje nam na konsoli już uzupełnione razem z ID dane. Ostatnia linia pochodzi z naszego kontrollera od strony głównej gdzie przepływ został przekierowany przez stan końcowy.

Zaawansowane elementy Spring WebFlow

Kod źródłowy z przykładami do tego rozdziału możesz pobrać pod adresem:

http://jsystems.pl/storage/spring/springflow2.zip

W poprzedniej części stworzyliśmy banalnie prosty przepływ z przejściami. W niniejszym będziemy rozwijać kod z poprzedniej części wzbogacając go o kilka elementów które pomogą nam ulepszyć program. Co chcemy osiągnąć:

- Dodać element który będzie sprawdzał czy użytkownik o takim emailu już jest zapisany do newslettera. Jeśli okaże się że owszem, to przekierowanie widoku informującego o tym.
- Dodać stronę podsumowującą z wyświetleniem wszystkich danych i podziękowaniem za zapisanie się do newslettera.
- Dodanie drugiego przepływu który będzie pozwalał rejestrować konta w sklepie, ale tym razem pojawi się tutaj więcej widoków.
- Dorobić jakiś jeden globalny stan końcowy oznaczający poprawne zakończenie przepływu bez potrzeby definiowania przejść do niego w każdym po kolei stanie na wypadek naciśnięcia przycisku „anuluj” na którymś stanie.
- Dodanie podprzepływu który będzie uruchamiany kiedy użytkownik wprowadzi kod promocyjny.

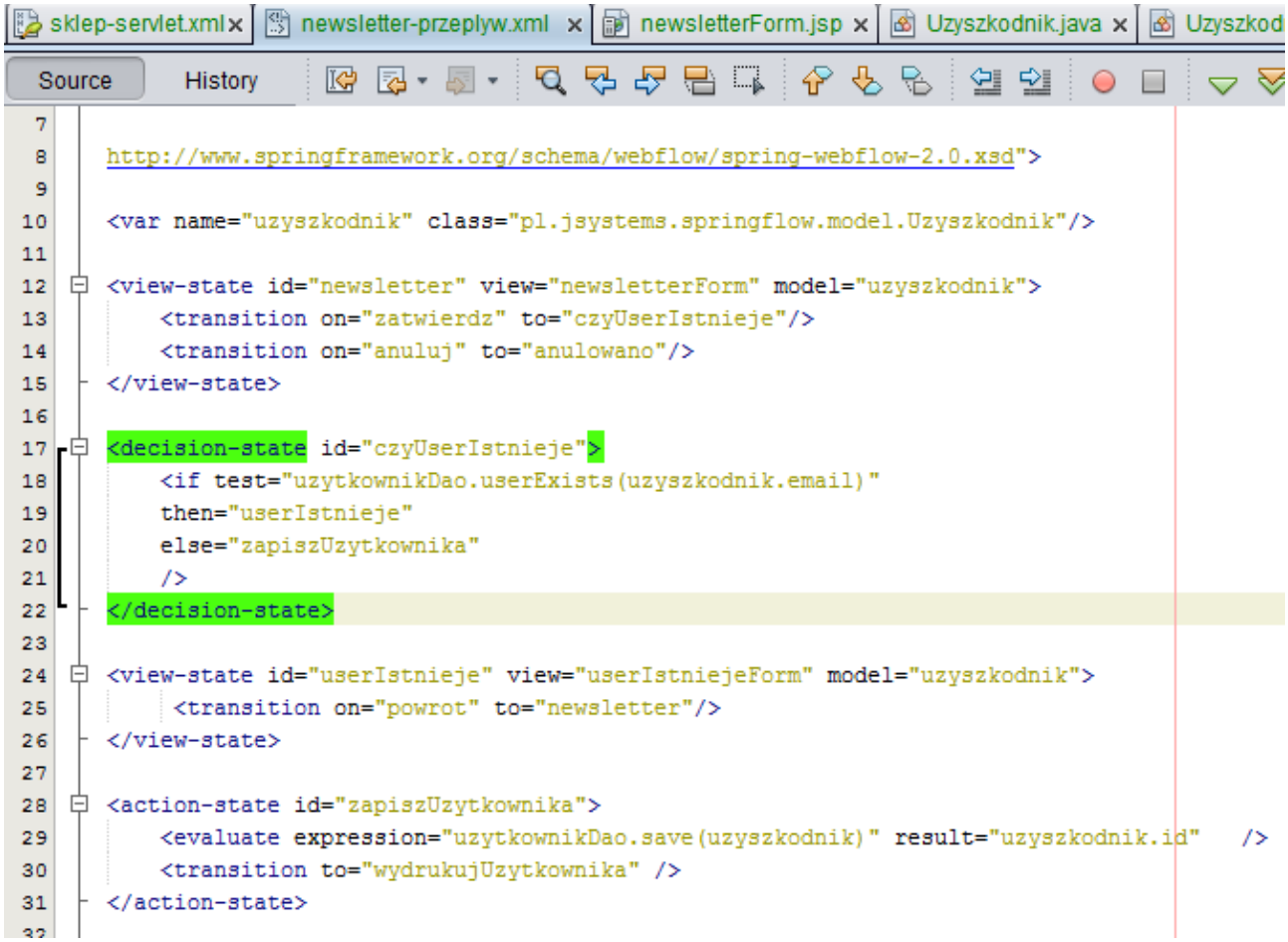
Stany decyzyjne

Zaczynamy od zadania: „**Dodać element który będzie sprawdzał czy użytkownik o takim emailu już jest zapisany do newslettera. Jeśli okaże się że owszem, to przekierowanie widoku informującego o tym.”**

Do naszego wcześniej stworzonego DAO dodaję nową metodę, która z założenia ma sprawdzać czy użytkownik o podanym adresie email już istnieje czy jeszcze nie. Nie bawimy się tutaj w kurs baz danych, zrobiłem po prostu sprawdzanie czy podano mój email czy jakiś inny. Jeśli ktoś poda adres email klusiewicz@Jsystems.pl, metoda zwróci „true” informując o istnieniu takiego użytkownika.

```
10  /**
11   *
12   * @author andrzej
13   */
14  public class UzyszkodnikDao {
15
16      public boolean userExists(String email) {
17          System.out.println("Sprawdzam czy użytkownik "+email+" już istnieje");
            if (email.equalsIgnoreCase("klusiewicz@jsystems.pl")) {
19              return true;
20          } else {
21              return false;
22          }
23      }
```

Przejdźmy teraz do pliku konfiguracji przepływu. Przyjrzyjmy się linii 13. Wcześniej w przypadku zatwierdzenia formularza było tutaj przekierowanie do stanu „zapiszUzytkownika”, teraz jest przekierowanie do nowego stanu – „czyUserIstnienie”. Jest to stan decyzyjny. W zależności od sprawdzanych warunków, przepływ może pójść w całkiem różnych kierunkach. Nasz stan decyzyjny (linie 17-22) sprawdza czy użytkownik o podanym adresie email już istnieje czy jeszcze nie. Robi to z użyciem metody userExists naszego DAO (dao to zostało zdeklarowane jako bean w poprzednim rozdziale). Do metody userExists przekazujemy zawartość pola email z uzupełnianego w trakcie tego przepływu obiektu uzyszkodnik. W zależności od tego czy metoda zwróci true czy false, zostaniemy przekierowani albo do stanu „userIstnieje”, albo do stanu „zapiszUzytkownika”. W przypadku gdyby okazało się że takiego użytkownika jeszcze nie było, to sprawa jest o tyle prosta że wystarczy przejść do zapisu obiektu. Jeśli jednak okazałoby się że mamy już użytkownika o takim adresie email, należy wyświetlić jakąś stronę informacyjną. Przyjrzymy się więc teraz liniom 24-26. Mamy tutaj kolejny stan widoku.



Na ten moment jako widok podałem „userIstniejeForm”. Wcześniej go nie tworzyliśmy, tymczasowo dodałem sobie taki wpis by cokolwiek tutaj było. Zaraz stworzymy warstwę widoku. Zauważ że tutaj też przekazuję do modelu obiekt uzyszkodnik (ten który uzupełniamy podczas przeplywu). Po co? Ponieważ będę chciał sięgnąć do jego zawartości w tym widoku – konkretnie do adresu email. W linii 25 pojawia nam się nowe zdarzenie – powrot. Zostanie ono wywołane znaną już konstrukcją kiedy ktoś kliknie na nowej podstronie informacyjnej link „powrót”. W takim przypadku zostanie przekierowany do stanu „newsletter” czyli naszego formularza.

Przejdźmy teraz do warstwy widoku. Stworzyłem wspomniany wcześniej plik „userIstniejeForm.jsp” i umieściłem w nim taką oto treść:

```
1
2   <%@page pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>
3
4   <html>
5
6       <body>
7           <h3>Zapisywanie do newslettera</h3>
8
9           <div>
10              <font color="red">Użytkownik ${uzyszkodnik.email} już istnieje</font>
11              <br><br>
12              <a href="${flowExecutionUrl}&amp;_eventId=powrot">Powrót</a>
13          </div>
14      </body>
15
16
17
18  </html>
```

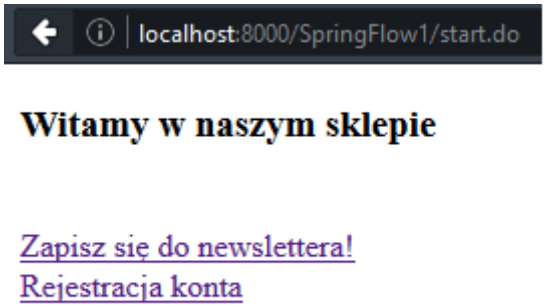
W zasadzie interesujące są tylko linie 10 i 12, bo tylko tam coś się dzieje ;)

W linii 10 wyświetlam email uzupelnianego podczas przepływu użytkownika, czyli bezpośrednio ten email który wstukaliśmy w formularzu (wyjaśnia się podawanie parametru model w nowym view-state). Linia 12 to znana już konstrukcja linka wywołującego akcję. Tym razem jest to akcja o nazwie „powrot” która wg zapisów w naszym pliku konfiguracji przepływu spowoduje powrót do formularza. Sprawdźmy więc działanie. Uzupełniam formularz i zatwierdzam.

Wywoływana w naszym stanie decyzyjnym metoda weryfikuje że użytkownik o podanym emailu już istnieje i powoduje przejście do stanu widoku "userIstnieje”. Przy okazji zwróć uwagę na pasek adresu. Cały czas widnieje tam adres „newsletter.do”!

Kliknięcie linka „Powrót” powoduje wywolanie akcji „powrot” i co za tym idzie powrót do formularza wypelniania danych:

Teraz działa to tak, że po wypełnieniu formularza wracamy do strony startowej:



Wyświetlanie danych

Chcemy dodać jakąś stronę podsumowującą z podziękowaniem za rejestrację w newsletterze. Zacznijmy więc od dodania kolejnego stanu – stanu widoku i zadbania o właściwe przekierowanie. Przechodzę do edycji pliku newsletter-przeplyw.xml i wprowadzam nieco modyfikacji:

```
32
33  <action-state id="wydrukujUzytkownika">
34      <evaluate expression="uzytkownikDao.print(uzyszkodnik)" />
35      <transition to="podziekowanie" />
36  </action-state>
37
38
39  <view-state id="podziekowanie" view="podziekowanieView" model="uzyszkodnik">
40      <transition on="zakoncz" to="dodano" />
41  </view-state>
42
```

Stan akcji „wydrukujUzytkownika” nie przekazuje już przepływu do stanu dodano który po prostu przekierowywał do strony startowej, tylko do nowego stanu „podziekowanie” który jest stanem widoku mającym wyświetlać podziękowanie. Dopiero ten nowy stan przekaże przepływ do strony startowej kiedy nastąpi zdarzenie „zakoncz”. W linii 39 widzimy, że za wyświetlenie widoku odpowiedzialny jest plik podziekowanieView.jsp. Toteż go tworzymy i wprowadzam poniższy kod:

```
4       Author     : andrzej
5   --%>
6   <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
7   <%@page contentType="text/html" pageEncoding="UTF-8"%>
8   <!DOCTYPE html>
9   <html>
10      <head>
11          <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
12          <title>JSP Page</title>
13      </head>
14      <body>
15
16          <h3>Dziękujemy ${uzyszkodnik.imie} ${uzyszkodnik.nazwisko} za rejestrację w naszym newsletterze.</h3>
17          <h4>Będziesz otrzymywał na podany adres email (${uzyszkodnik.email}) nowości z naszej oferty!</h4>
18
19          <a href="${flowExecutionUrl}&amp;_eventId=zakoncz">Powrót do strony startowej</a>
20      </body>
21  </html>
22
```

Nie ma tutaj nic nadzwyczajnego, to jest zwykła strona podziękowania. Warte chwili jest za to przyjrzenie się liniom 16 i 17, oraz linii 19. W liniach 16 i 17 widzimy odwołanie do obiektu uzyszkodnik – a właściwie do jego pól. Zauważ, że jest to obiekt który uzupełniamy w ramach tego przepływu. Aby mieć dostęp do danych w nim zawartych, musimy dodać parametr „model”, który widzimy w linii 39 pliku newsletter-przeplyw.xml. Bez tej deklaracji nie będziemy mieli dostępu do tych danych! W linii 19 jest wywołanie przez link akcji „zakoncz” która powoduje przekazanie przepływu do stanu „dodano” (co widać w linii 40 pliku newsletter-przeplyw.xml).

Istnieje taka możliwość, że ktoś w formularzu wprowadzi polskie litery. Choć nie jest to akurat element Springa, warto dodać poniższy wpis do swojego pliku web.xml, aby wprowadzone polskie znaki nie zmieniły się w krzaczki.

```
6        <filter>
7        <filter-name>encoding-filter</filter-name>
8        <filter-class>
9            org.springframework.web.filter.CharacterEncodingFilter
10       </filter-class>
11       <init-param>
12           <param-name>encoding</param-name>
13           <param-value>UTF-8</param-value>
14       </init-param>
15       <init-param>
16           <param-name>forceEncoding</param-name>
17           <param-value>true</param-value>
18       </init-param>
19   </filter>
20
21   <filter-mapping>
22       <filter-name>encoding-filter</filter-name>
23       <url-pattern>/*</url-pattern>
24   </filter-mapping>
25
```

Po wszystkim uzupełniam formularz i sprawdzam działanie całości:

Dziękujemy Grześ Brzęczyszczykiewicz za rejestrację w naszym newsletterze.

Będziesz otrzymywal na podany adres email (grzes@lubiespringa.pl) nowości z naszej oferty!

Powrót do strony startowej

Wieloetapowe uzupełnianie obiektu podczas przepływu

Czas dodać drugi przepływ, tym razem umożliwiający założenie konta w naszym sklepie internetowym. Trzeba będzie podać imię i nazwisko, dane kontaktowe, adres, dane do faktur etc. Mało przejrzyste i wygodne to będzie jeśli zrobimy to na jednej stronie. Lepiej rozbić to na kilka etapów. Osobna podstrona do imienia i nazwiska, osobna do danych kontaktowych, do adresu i osobna do danych do faktur. Przepływ będzie jeden, ale podczas jego trwania będziemy uzupełniali jeden obiekt.
Zaczynamy od edycji naszego pliku ****-servlet.xml, aby zarejestrować nowy przepływ. Nowe wpisy zaznaczylem na czerwono. Dodałem adres strony początkowej przepływu – konto.do, oraz nazwę i położenie pliku konfiguracyjnego kolejnego przepływu – konto-przeplyw.xml.

```
37
38      <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
39          <property name="mappings">
40              <value>
41                  /newsletter.do=flowController
42                  /konto.do=flowController
43              </value>
44
45          </property>
46      </bean>
47
48
49      <bean id="flowController" class="org.springframework.webflow.mvc.servlet.FlowController">
50          <property name="flowExecutor" ref="flowExecutor"/>
51      </bean>
52
53      <!--Tworzy przeplywy na podstawie flowRegistry -->
54      <flow:flow-executor id="flowExecutor" flow-registry="flowRegistry"/>
55      <!--Rejestr plików z opisami poszczególnych przeplywów-->
56      <flow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices">
57          <flow:flow-location path="/WEB-INF/flows/newsletter-przeplyw.xml" id="newsletter"/>
58          <flow:flow-location path="/WEB-INF/flows/konto-przeplyw.xml" id="konto"/>
59      </flow:flow-registry>
```

Przyjrzyjmy się naszemu plikowi konfiguracji przepływu.

Przepływ jest stosunkowo prosty, ponieważ ogranicza się do czterech następujących po sobie stanów widoku i na końcu jednego stanu akcji zapisującego nowe konto do bazy.

```
9
10  <var name="konto" class="pl.jsystems.springflow.model.Konto"/>
11
12  <view-state id="step1" view="daneKontaktowe" model="konto">
13      <transition on="dalej" to="step2"/>
14      <transition on="wstecz" to="start"/>
15      <transition on="anuluj" to="start"/>
16  </view-state>
17
18  <view-state id="step2" view="adresDostawy" model="konto">
19      <transition on="dalej" to="step3"/>
20      <transition on="wstecz" to="step1"/>
21      <transition on="anuluj" to="start"/>
22  </view-state>
23
24  <view-state id="step3" view="daneDoFaktury" model="konto">
25      <transition on="dalej" to="podsumowanie"/>
26      <transition on="wstecz" to="step2"/>
27      <transition on="anuluj" to="start"/>
28  </view-state>
29
30  <view-state id="podsumowanie" view="podsumowanie" model="konto">
31      <transition on="dalej" to="zapisz"/>
32      <transition on="wstecz" to="step3"/>
33      <transition on="anuluj" to="start"/>
34  </view-state>
35
36  <action-state id="zapisz">
37      <evaluate expression="kontoDao.zapisz(konto)" />
38      <transition to="start" />
39  </action-state>
40
41  <end-state id="start"
42      view="externalRedirect:contextRelative:/start.do"/>
```

Podczas tego przepływu będziemy uzupełniać obiekt klasy Konto (co widać w deklaracji w linii 10). Za chwilę do niego wrócimy, teraz przyjrzyjmy się kolejnym przejściom tego przepływu. Na każdym z widoków będą trzy przyciski – dalej, wstecz, anuluj. Pierwszy powoduje przejście do następnego kroku, drugi do poprzedniego a trzeci anuluje cały przepływ i powraca do strony startowej. Wszystkie stany widoku mają zadeklarowany parametr model – będziemy uzupełniać obiekt, więc stany te muszą mieć do niego dostęp. Zauważ że w każdym kolejnym stanie mamy deklarację taką :

<transition on="anuluj" to="start"/>

Deklarowanie przejścia anulującego w każdym kroku jest co najmniej męczące. Co gdybyśmy np. chcieli zmienić stan do którego przechodzi przepływ po kliknięciu „anuluj”? W dalszych krokach zadeklarujemy przejście globalne i nie będziemy musieli deklarować tego przejścia w każdym stanie.

Przepływ składa się z trzech formularzy i jednego widoku podsumowującego. W pierwszym formularzu uzupełniamy dane kontaktowe i informacje o koncie, w drugim adres dostawy, w trzecim dane do faktury. W widoku podsumowującym widzimy wszystkie dane które uzupełnialiśmy i możemy ewentualnie wrócić i coś poprawić.

Przez wszystkie etapy uzupełniamy obiekt klasy konto, więc przyjrzyjmy się tejże klasie.

```
9
10  /**
11   *
12   * @author andrzej
13   */
    public class Konto implements Serializable{
15
16      private String login;
17      private String haslo;
18      private String imie;
19      private String nazwisko;
20      private String email;
21      private String telefon;
22      private Adres adresDostawy=new Adres();
23      private DaneDoFaktury daneDoFaktury=new DaneDoFaktury();
24
```

Oczywiście w klasie tej są również gettery i settery do tych pól. Widzimy tutaj obiekty klas Adres i DaneDoFaktury, więc zobaczmy również ich zawartość. Klasa Adres:

```
10  /**
11   *
12   * @author andrzej
13   */
    public class Adres  implements Serializable{
15      private Long Id;
16      private String miasto;
17      private String kodPocztowy;
18      private String ulica;
19      private String numerBudynku;
20      private String numerLokalu;
```

Klasa DaneDoFaktury:

```
 9
10  /**
11   *
12   * @author andrzej
13   */
     public class DaneDoFaktury  implements Serializable{
15
16       private Long id;
17       private String nazwaFirmy;
18       private String NIP;
19       private Adres adresFirmy=new Adres();
20
```

Zauważ że przy polach będących obiektami mamy użycie domyslnego konstruktora. Przy typach prostych tego nie robię. Musi tak być, ponieważ Spring tworzy nowy obiekt klasy którą zadeklarowaliśmy jako zmienną przepływu, ale już nie robi tego dla „podobiektów". Jeśli sami nie zaincjalizujemy ich, dostaniemy błąd podczas próby dostępu do ich pól. Przy okazji zobaczmy też deklarację metody zapisz którą wywołuję w stanie „zapisz" naszego przepływu. Jak widać jest to zaślepka – nie zajmujemy się tutaj integracją Springa z bazami danych.

```
10  /**
11   *
12   * @author andrzej
13   */
14  public class KontoDao {
15      public void zapisz(Konto konto){
16          System.out.println("zapisuję do bazy nowe konto : "+konto.toString());
17      }
18  }
```

W pliku konfiguracji przepływu do tego DAO odwołuję się tak:

```
35
36  <action-state id="zapisz">
37      <evaluate expression="kontoDao.zapisz(konto)" />
38      <transition to="start" />
39  </action-state>
40
```

a to oznacza że musimy mieć jeszcze zadeklarowany bean do tej klasy w pliku *****-servlet.xml:

```
25
26      <bean id="uzytkownikDao" class="pl.jsystems.springflow.dao.UzyszkodnikDao"/>
27      <bean id="kontoDao" class="pl.jsystems.springflow.dao.KontoDao"/>
28
```

Zobaczmy teraz warstwę widoku, czyli nasze formularze i stronę podsumowującą. Formularz numer 1 (czyli stan step1) służy do uzupełniania danych o koncie i informacji kontaktowych. Tradycyjne pola do wprowadzania danych służące uzupełnianiu bezpośrednich pól obiektu klasy Konto i guziki o których wspominałem wcześniej:

```
<%@page pageEncoding="UTF-8" contentType="text/html; charset=UTF-8"%>
<%@taglib prefix="form" uri="http://www.springframework.org/tags/form" %>
<%@taglib prefix="spring" uri="http://www.springframework.org/tags" %>
<%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

    </head>
    <body>
        <h3>Rejestracja konta - krok 1 - dane kontaktowe i informacje o koncie</h3>

        <div>
            <form:form commandName="konto">
                <table>
                    <tr><td>Login: </td><td><form:input path="login"/></td></tr>
                    <tr><td>Haslo:</td><td><form:input path="haslo"/></td></tr>
                    <tr><td>Imię: </td><td><form:input path="imie"/></td></tr>
                    <tr><td>Nazwisko: </td><td><form:input path="nazwisko"/></td></tr>
                    <tr><td>Email: </td><td><form:input path="email"/></td></tr>
                    <tr><td>Telefon: </td><td><form:input path="telefon"/></td></tr>
                </table>
                <input type="submit" name="_eventId_dalej" value="Dalej"/>
                <input type="submit" name="_eventId_wstecz" value="Wstecz"/>
                <input type="submit" name="_eventId_anuluj" value="Anuluj"/>

            </form:form>
        </div>
    </body>


</html>
```

Kolejny krok i drugi formularz to uzupełnianie danych do dostawy zakupów:

```
<body>
    <h3>Rejestracja konta - krok 2 - adres dostaw</h3>

    <div>
        <form:form commandName="konto">
            <table>
                <tr><td>Miasto: </td><td><form:input path="adresDostawy.miasto"/></td></tr>
                <tr><td>Kod pocztowy: </td><td><form:input path="adresDostawy.kodPocztowy"/></td></tr>
                <tr><td>Ulica: </td><td><form:input path="adresDostawy.ulica"/></td></tr>
                <tr><td>Numer budynku: </td><td><form:input path="adresDostawy.numerBudynku"/></td></tr>
                <tr><td>Numer lokalu: </td><td><form:input path="adresDostawy.numerLokalu"/></td></tr>
            </table>
            <input type="submit" name="_eventId_dalej" value="Dalej"/>
            <input type="submit" name="_eventId_wstecz" value="Wstecz"/>
            <input type="submit" name="_eventId_anuluj" value="Anuluj"/>

        </form:form>
    </div>
</body>
```

Przyjrzyj się dobrze parametrom „path”... W taki właśnie sposób odwołujemy się do podobiektów i ich pól. Krok trzeci to uzupełnianie danych do faktury. Widzimy tutaj odwołania do jeszcze bardziej zagnieżdżonych podobiektów.

```
<h3>Rejestracja konta - krok 3 - dane do faktury</h3>

<div>
    <form:form commandName="konto">
        <table>

            <tr><td>Nazwa firmy: </td><td><form:input path="daneDoFaktury.nazwaFirmy"/></td></tr>
            <tr><td>NIP: </td><td><form:input path="daneDoFaktury.NIP"/></td></tr>

            <tr><td>Miasto: </td><td><form:input path="daneDoFaktury.adresFirmy.miasto"/></td></tr>
            <tr><td>Kod pocztowy: </td><td><form:input path="daneDoFaktury.adresFirmy.kodPocztowy"/></td></tr>
            <tr><td>Ulica: </td><td><form:input path="daneDoFaktury.adresFirmy.ulica"/></td></tr>
            <tr><td>Numer budynku: </td><td><form:input path="daneDoFaktury.adresFirmy.numerBudynku"/></td></tr>
            <tr><td>Numer lokalu: </td><td><form:input path="daneDoFaktury.adresFirmy.numerLokalu"/></td></tr>
        </table>
        <input type="submit" name="_eventId_dalej" value="Dalej"/>
        <input type="submit" name="_eventId_wstecz" value="Wstecz"/>
        <input type="submit" name="_eventId_anuluj" value="Anuluj"/>

    </form:form>
```

No i na koniec nasza strona podsumowująca:

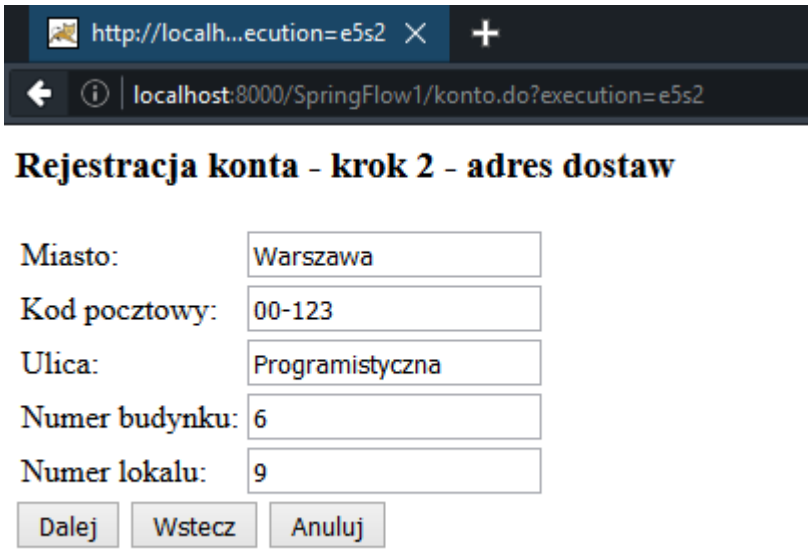
```
11      <body>
12          <h3>Podsumowanie rejestracji</h3>
13
14          <div>
15              <h4>Dane kontaktowe i informacje o koncie</h4>
16              <table>
17                  <tr><td>Login: </td><td>${konto.login}</td></tr>
18                  <tr><td>Haslo:</td><td>${konto.haslo}</td></tr>
19                  <tr><td>Imię: </td><td>${konto.imie}</td></tr>
20                  <tr><td>Nazwisko: </td><td>${konto.nazwisko}</td></tr>
21                  <tr><td>Email: </td><td>${konto.email}</td></tr>
22                  <tr><td>Telefon: </td><td>${konto.telefon}</td></tr>
23              </table>
24
25              <h4>Adres dostawy</h4>
26
27               <table>
28                      <tr><td>Miasto: </td><td>${konto.adresDostawy.miasto}</td></tr>
29                      <tr><td>Kod pocztowy: </td><td>${konto.adresDostawy.kodPocztowy}</td></tr>
30                      <tr><td>Ulica: </td><td>${konto.adresDostawy.ulica}</td></tr>
31                      <tr><td>Numer budynku: </td><td>${konto.adresDostawy.numerBudynku}</td></tr>
32                      <tr><td>Numer lokalu: </td><td>${konto.adresDostawy.numerLokalu}</td></tr>
33                  </table>
34
35              <h4>Dane do faktury</h4>
36
37                 <table>
38                      <tr><td>Nazwa firmy: </td><td>${konto.daneDoFaktury.nazwaFirmy}</td></tr>
39                      <tr><td>NIP: </td><td>${konto.daneDoFaktury.NIP}</td></tr>
40                      <tr><td>Miasto: </td><td>${konto.daneDoFaktury.adresFirmy.miasto}</td></tr>
41                      <tr><td>Kod pocztowy: </td><td>${konto.daneDoFaktury.adresFirmy.kodPocztowy}</td></tr>
42                      <tr><td>Ulica: </td><td>${konto.daneDoFaktury.adresFirmy.ulica}</td></tr>
43                      <tr><td>Numer budynku: </td><td>${konto.daneDoFaktury.adresFirmy.numerBudynku}</td></tr>
44                      <tr><td>Numer lokalu: </td><td>${konto.daneDoFaktury.adresFirmy.numerLokalu}</td></tr>
45                  </table>
46
47
48              <form:form commandName="konto">
49                  <input type="submit" name="_eventId_dalej" value="Dalej"/>
50                  <input type="submit" name="_eventId_wstecz" value="Wstecz"/>
51                  <input type="submit" name="_eventId_anuluj" value="Anuluj"/>
52              </form:form>
53          </div>
54      </body>
```

Tutaj standardowy sposób wyświetlania danych z obiektu na zasadzie „${obiekt.pole}”. Wyświetlamy dane uzupełniane we wszystkich etapach. Możemy tak zrobić ponieważ cały czas pracujemy na jednym obiekcie klasy Konto zainicjalizowanym na początku przepływu. Odwołujemy się do pól, podobiektów, pól podobiektów dokładnie tak jak przy np. servletach – w końcu robimy to z użyciem tagów JSTL. A teraz przyjrzyj się liniom 48-52. Guziki jak wczesniej, ale przecież wiemy że to jest strona wyświetlająca podsumowanie, nie ma tu formularza. Co więc robią tutaj tagi „<form:form>” ? Otóż bez nich nie działałyby nasze przyciski, ponieważ zatwierdzają one formularze! Inaczej musielibyśmy je zamienić na linki. Zobaczmy teraz jak to wszystko razem działa ;)

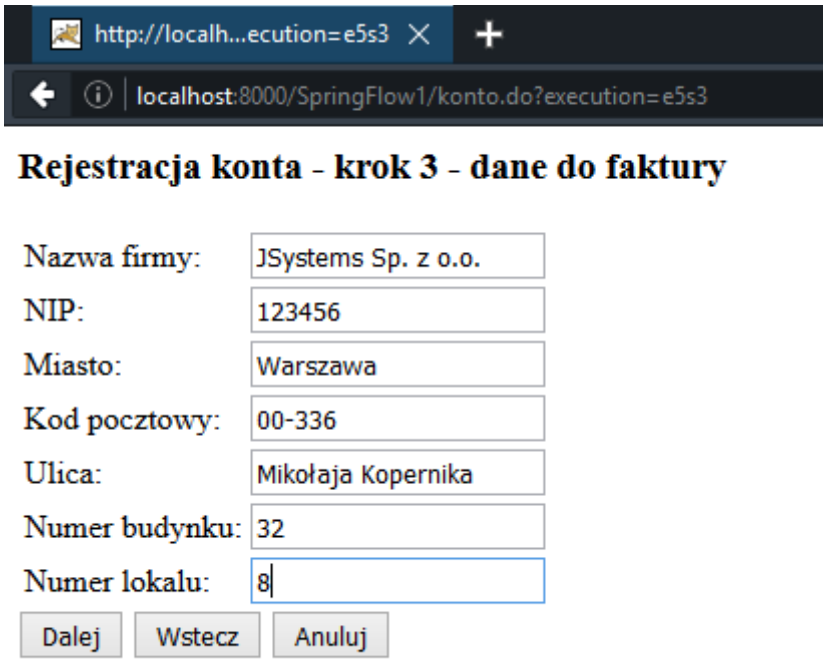
Krok 1:


localhost:8000/SpringFlow1/konto.do?execution=e5s1
Rejestracja konta - krok 1 - dane kontaktowe i informacje o koncie
Login: mapet
Haslo: mapettubyl
Imię: Andrzej
Nazwisko: Klusiewicz
Email: klusiewicz@jsystems.pl
Telefon: 112
Dalej
Wstecz
Anuluj

Krok 2:

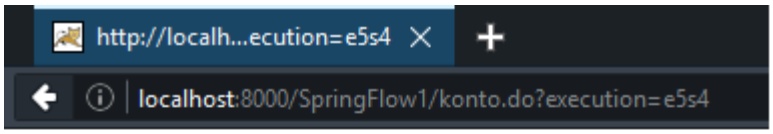

http://localh...ecution=e5s2
localhost:8000/SpringFlow1/konto.do?execution=e5s2
Rejestracja konta - krok 2 - adres dostaw
Miasto: Warszawa
Kod pocztowy: 00-123
Ulica: Programistyczna
Numer budynku: 6
Numer lokalu: 9
Dalej
Wstecz
Anuluj

Krok 3:



Taka mała ciekawostka przy okazji. Przyjrzyj się paskowi adresu, a właściwie to wartości w parametrze execution w poszczególnych krokach przepływu.

Czas na stronę podsumowującą:



Podsumowanie rejestracji

Dane kontaktowe i informacje o koncie

Login:	mapet
Haslo:	mapettuby1
Imię:	Andrzej
Nazwisko:	Klusiewicz
Email:	klusiewicz@jsystems.pl
Telefon:	112

Adres dostawy

Miasto:	Warszawa
Kod pocztowy:	00-123
Ulica:	Programistyczna
Numer budynku:	6
Numer lokalu:	9

Dane do faktury

Nazwa firmy:	JSystems Sp. z o.o.
NIP:	123456
Miasto:	Warszawa
Kod pocztowy:	00-336
Ulica:	Mikołaja Kopernika
Numer budynku:	32
Numer lokalu:	8

Dalej Wstecz Anuluj

Przejście globalne

Pojawił nam się wcześniej problem z ciągłym deklarowaniem przejścia do strony startowej w przypadku naciśnięcia anuluj na którymkolwiek z kroków przepływu:

```
12  <view-state id="step1" view="daneKontaktowe" model="konto">
13      <transition on="dalej" to="step2"/>
14      <transition on="wstecz" to="start"/>
15      <transition on="anuluj" to="start"/>
16  </view-state>
17
18  <view-state id="step2" view="adresDostawy" model="konto">
19      <transition on="dalej" to="step3"/>
20      <transition on="wstecz" to="step1"/>
21      <transition on="anuluj" to="start"/>
22  </view-state>
23
24  <view-state id="step3" view="daneDoFaktury" model="konto">
25      <transition on="dalej" to="podsumowanie"/>
26      <transition on="wstecz" to="step2"/>
27      <transition on="anuluj" to="start"/>
28  </view-state>
29
30  <view-state id="podsumowanie" view="podsumowanie" model="konto">
31      <transition on="dalej" to="zapisz"/>
32      <transition on="wstecz" to="step3"/>
33      <transition on="anuluj" to="start"/>
34  </view-state>
35
36  <action-state id="zapisz">
37      <evaluate expression="kontoDao.zapisz(konto)" />
38      <transition to="start" />
39  </action-state>
40
41  <end-state id="start"
42      view="externalRedirect:contextRelative:/start.do"/>
43
```

Można i tak, ale wygodniej będzie zdeklarować tak zwane przejście globalne. Ustalasz w jednym miejscu że np. skądkowiek wywołana akcja „powrot" przekierowuje do jakiegos poczatkowego stanu widoku i nie musisz tego deklarować w każdym stanie. W podobny sposób możesz np. zrobić przekierowanie na stronę główną czy do formularza logowania.

Dodajmy więc globalne przejście do naszego pliku przepływu:

```
10    <var name="konto" class="pl.jsystems.springflow.model.Konto"/>
11
12    <view-state id="step1" view="daneKontaktowe" model="konto">
13        <transition on="dalej" to="step2"/>
14        <transition on="wstecz" to="start"/>
15    </view-state>
16
17    <view-state id="step2" view="adresDostawy" model="konto">
18        <transition on="dalej" to="step3"/>
19        <transition on="wstecz" to="step1"/>
20    </view-state>
21
22    <view-state id="step3" view="daneDoFaktury" model="konto">
23        <transition on="dalej" to="podsumowanie"/>
24        <transition on="wstecz" to="step2"/>
25    </view-state>
26
27    <view-state id="podsumowanie" view="podsumowanie" model="konto">
28        <transition on="dalej" to="zapisz"/>
29        <transition on="wstecz" to="step3"/>
30    </view-state>
31
32    <action-state id="zapisz">
33        <evaluate expression="kontoDao.zapisz(konto)" />
34        <transition to="start" />
35    </action-state>
36
37
38    <end-state id="start"
39        view="externalRedirect:contextRelative:/start.do"/>
40
41
42    <global-transitions>
43        <transition on="anuluj" to="start"/>
44    </global-transitions>
45
46    </flow>
```

Pozbyłem się przejść kierujących do stanu start z poszczególnych stanów, a w ich miejsce wstawiłem element z linii 42-44. Przy okazji mała uwaga – z jakiegoś powodu sekcja globa-transitions musi być na końcu pliku. Ilekroć znajdowała się gdzieś indziej, dostawałem dziwne błędy i zwykle dotyczyły one sekcji następującej po niej.

Podprzepływy

Przypuśćmy teraz, że w zależności od jakichś warunków chcemy przeprowadzić jakąś dodatkową sekwencję kroków. Przykładowo użytkownik dysponuje kodem promocyjnym. Jeśli nim dysponuje, uruchomimy dodatkowy przepływ który sprawdzi poprawność kodu i powróci do pierwotnego przepływu. Chcemy by pytanie o posiadanie kodu promocyjnego pojawiało się od razu na początku. Będą do wyboru dwa przyciski - „Posiadam" i „Nie posiadam". Jeśli użytkownik wybierze „Posiadam" to główny przepływ zostanie na chwilę wstrzymany, przejdzie przez dodatkowy podprzepływ i wróci do przepływu głównego. Jeśli ktoś wybierze „Nie posiadam", to po prostu główny przepływ będzie kontynuowany.

Zaczynamy od poinformowania Springa o nowym przepływie w pliku *****-servlet.xml:

```
37
38      <bean class="org.springframework.web.servlet.handler.SimpleUrlHandlerMapping">
39          <property name="mappings">
40              <value>
41                  /newsletter.do=flowController
42                  /konto.do=flowController
43                  /kodPromocyjny.do=flowController
44              </value>
45
46          </property>
47      </bean>
48
49
50      <bean id="flowController" class="org.springframework.webflow.mvc.servlet.FlowController">
51          <property name="flowExecutor" ref="flowExecutor"/>
52      </bean>
53
54      <!--Tworzy przeplywy na podstawie flowRegistry -->
55      <flow:flow-executor id="flowExecutor" flow-registry="flowRegistry"/>
56      <!--Rejestr plików z opisami poszczególnych przeplywów-->
57      <flow:flow-registry id="flowRegistry" flow-builder-services="flowBuilderServices">
58          <flow:flow-location path="/WEB-INF/flows/newsletter-przeplyw.xml" id="newsletter"/>
59          <flow:flow-location path="/WEB-INF/flows/konto-przeplyw.xml" id="konto"/>
60          <flow:flow-location path="/WEB-INF/flows/kodpromocyjny-przeplyw.xml" id="kodpromocyjny"/>
61      </flow:flow-registry>
```

Tworzymy plik konfiguracji podprzepływu który przed momentem obiecaliśmy: kodpromocyjny-przeplyw.xml. Poniżej tylko jego „mięsista zawartość.

```
<var name="konto" class="pl.jsystems.springflow.model.Konto"/>

<view-state id="czymaszkod" view="czyMaszKod" model="konto">
    <transition on="mam" to="kodpromocyjny"/>
    <transition on="niemam" to="anulowano"/>
</view-state>


<view-state id="kodpromocyjny" view="kodPromocyjny" model="konto">
    <transition on="zatwierdz" to="sprawdzKod"/>
    <transition on="anuluj" to="anulowano"/>
</view-state>

<action-state id="sprawdzKod">
    <evaluate expression="kontoDao.sprawdzKod(konto.kodPromocyjny)" />
    <transition to="zakonczono" />
</action-state>

<end-state id="zakonczono">
    <output name="konto"/>
</end-state>

<end-state id="anulowano"/>
```

Przepływ jest dosyć prosty, natomiast w związku z tym że jest to podprzepływ, muszę wyjaśnić kilka rzeczy. Posługuję się tutaj cały czas obiektem klasy Konto. Zostanie on przekazany do przepływu i z niego zwrócony. Sam kod promocyjny, jako że jest zwykłym tekstem umieścimy w nowym polu klasy Konto które też dopisałem:

```
/**
 *
 * @author andrzej
 */
public class Konto implements Serializable{

    private String login;
    private String haslo;
    private String imie;
    private String nazwisko;
    private String email;
    private String telefon;
    private Adres adresDostawy=new Adres();
    private DaneDoFaktury daneDoFaktury=new DaneDoFaktury();

    private String kodPromocyjny;
```

Pierwszy stan to stan widoku który jedyne co robi to pyta czy posiadasz kod promocyjny czy nie. Zawartość pliku czyMaszKod.jsp:

```
11      <body>
12          <h3>Kod promocyjny</h3>
13
14          <div>
15              <form:form commandName="konto">
16                  Czy posiadasz kod promocyjny?
17                  <br><br>
18                  <input type="submit" name="_eventId_mam" value="Posiadam"/>
19                  <input type="submit" name="_eventId_niemam" value="Nie posiadam"/>
20
21              </form:form>
```

W zależności od wyboru albo wróci do głównego przepływu (w wyniku wywołania akcji „niemam” jest przejście do stanu „anulowano” który jest stanem końcowym), albo przejdzie do stanu „kodpromocyjny” w którym wprowadzamy owy kod. Zawartość pliku kodPromocyjny.jsp:

```
11      <body>
12          <h3>Kod promocyjny</h3>
13
14          <div>
15              <form:form commandName="konto">
16                  Kod promocyjny: <form:input path="kodPromocyjny"/><br>
17
18                  <input type="submit" name="_eventId_zatwierdz" value="Zatwierdź"/>
19                  <input type="submit" name="_eventId_anuluj" value="Anuluj"/>
20
21              </form:form>
```

Przy wyborze „anuluj” jest przejście do stanu końcowego anulowano, ale jeśli ktoś wprowadzi kod i zatwierdzi wybierając „zatwierdź”, podprzepływ przejdzie do stanu akcji „sprawdzKod”. Pole do wprowadzania kodu promocyjnego jest podpięte do naszego nowego pola „kodPromocyjny” w klasie „Konto”. Stan sprawdxKod wywołuje metodę „sprawdzKod” z kontoDao która jest tylko zaślepką:

```
14  public class KontoDao {
15      public void zapisz(Konto konto){
16          System.out.println("zapisuję do bazy nowe konto : "+konto.toString());
17      }
18
19      public boolean sprawdzKod(String kod){
20          System.out.println("sprawdzam kod "+kod);
21          return true;
22      }
23  }
24
```

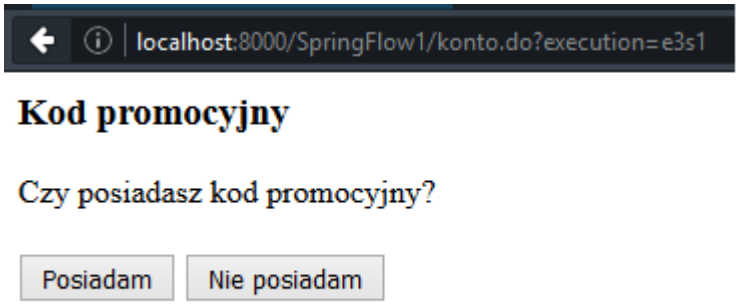
Oczywiście można byłoby tutaj podpiąć faktyczne sprawdzanie i jakiś stan decyzyjny / walidację. Taki stan jaki jest na razie całkiem nam wystarczy, ponieważ zajmujemy się w tej chwili podprzepływami jako takimi. Efekt jest taki, że jaki byś kod nie wprowadził to zawsze będzie ok ;)

Po zakończeniu tego stanu akcji następuje przejście do stanu „zakonczono” który jest stanem końcowym tego podprzepływu.Pojawia się tutaj za to nowa rzecz - <output name=”konto”/>. Co to? To jest element który przekazuje do głównego przepływu uzupełniany obiekt. Zaraz się to wyjaśni, jak tylko przejdziemy do wpinania podprzepływu do głównego przepływu. Przejdźmy teraz do pliku konfiguracji głównego przepływu – konto-przeplyw.xml:

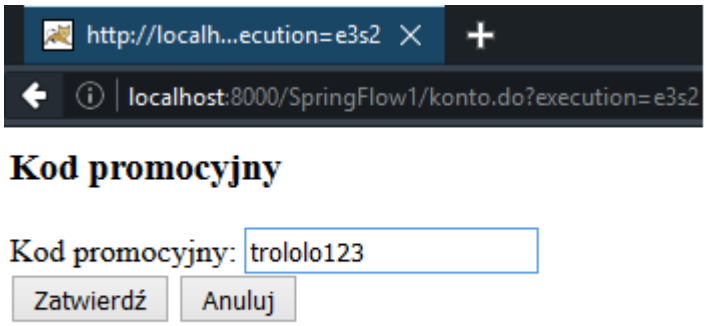
```
10      <var name="konto" class="pl.jsystems.springflow.model.Konto"/>
11
12      <subflow-state id="kodPromocyjny" subflow="kodpromocyjny">
13          <output name="konto" value="konto"/>
14          <transition to="step1"/>
15      </subflow-state>
16
17      <view-state id="step1" view="daneKontaktowe" model="konto">
18          <transition on="dalej" to="step2"/>
19          <transition on="wstecz" to="start"/>
20      </view-state>
21
```

Jako pierwszy stan na liście pojawił się teraz „subflow-state” - czyli stan podprzepływu. Zostanie on uruchomiony jako pierwszy, ponieważ jako pierwszy znajduje sięna liście. Parametr subflow określa nazwę podprzepływu który ma zostać uruchomiony. Parametr output dotyczy tego co nam „wypadnie” z podprzepływu. Będziemy uzupełniać cały czas obiekt klasy „Konto” - ten sam co w głównym przepływie. Transition to z linii 14 mówi o tym gdzie mamy przejść po zakończeniu podprzepływu – a wrócimy do kroku pierwszego – tj. formularza danych kontaktowych.

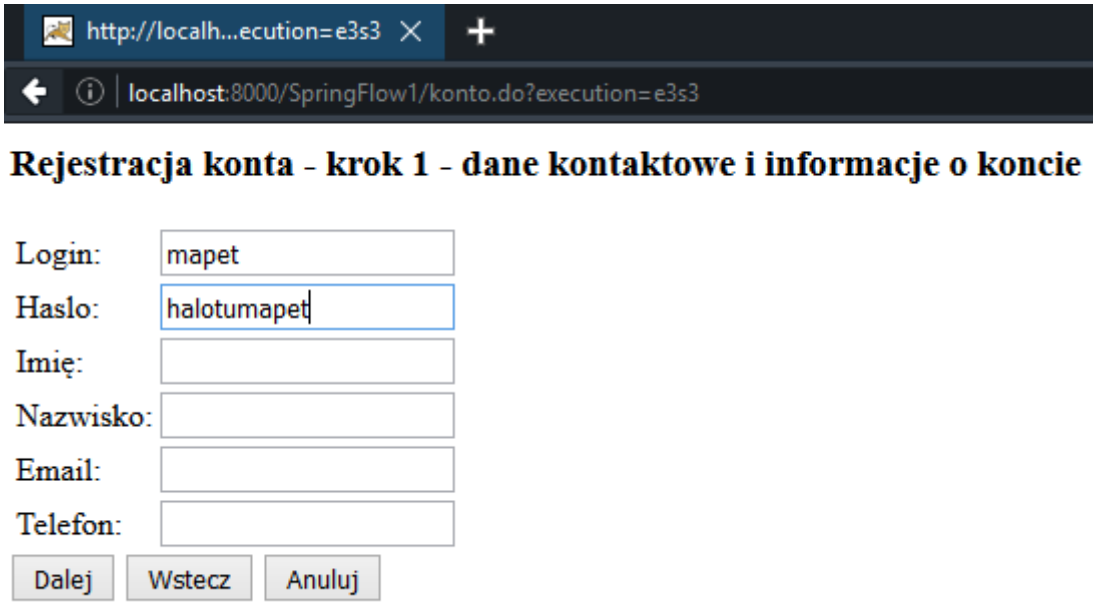
Uruchamiamy całość i sprawdzamy:



Wybrałem że posiadam kod promocyjny, który w następnym kroku wprowadzam:



Zatwierdzam i przepływ wraca do głównego przebiegu:



Uzupełniam tylko login i hasło, resztę we wszystkich krokach pozostawiam puste.

W pliku podsumowanie.jsp dodałem też wyświetlanie ewentualnie uzupełnionego kodu promocyjnego:

```
14      <div>
15          <h4>Dane kontaktowe i informacje o koncie</h4>
16          <table>
17              <tr><td>Login: </td><td>${konto.login}</td></tr>
18              <tr><td>Haslo:</td><td>${konto.haslo}</td></tr>
19              <tr><td>Imię: </td><td>${konto.imie}</td></tr>
20              <tr><td>Nazwisko: </td><td>${konto.nazwisko}</td></tr>
21              <tr><td>Email: </td><td>${konto.email}</td></tr>
22              <tr><td>Telefon: </td><td>${konto.telefon}</td></tr>
23              <tr><td>Kod promocyjny: </td><td>${konto.kodPromocyjny}</td></tr>
            </table>
```

Strona z podsumowaniem po zakończeniu przebiegu: