PŘÍLOHA č. 3

k bakalářské práci

Informační systém ozdravných pobytů zdravotní pojišťovny

Framework Simpleton – uživatelský manuál

Framework SIMPLETON

Uživatelský manuál

1. Úvod

Framework Simpleton je jednoduchý javovský framework pro vytváření webových UI. Na rozdíl od standardních frameworků jako je např. JSF či Struts, které jsou primárně určeny pro webové designéry, je Simpleton zamýšlen především pro javovské vývojáře. Proto definuje jenom minimální počet webových elementů a většina funkcionality se provádí doprovodným javovským kódem. Na rozdíl od jiných frameworků nepoužívá Simpleton žádné xml konfigurační soubory a nedefinuje žádné anotace ani speciální skriptovací jazyk.

Značkový jazyk Simpletonu je standardní XHTML rozšířený o jmenný prostor www.smp.org, ve kterém je definováno pouze několik nezbytně nutných elementů.

Simpleton je navržen tak, aby maximum možných chyb bylo nalezeno při sestavení aplikace a aby zbytek byl jasně detekován při běhu.

2. Hello world

Každá xhtml stránka s dynamickým obsahem musí být doprovázena deklarací stejnojmenné třídy (tzv. pagebean třídy), která musí být podtřídou ***org.smp.pages.Page***. Pagebean musí obsahovat definice vlastností (properties), jejichž hodnoty se vkládají do dynamicky generované xhtml stránky. Hodnoty vlastností jsou ve stránce reprezentovány závorkami **${ … }** (resp **%{ … }**) obsahujícími jméno vlastnosti. Před tím, než se začne výsledná stránka generovat, vytvoří Simpleton instanci pagebean třídy (tzv. pagebean) a odkaz na vlastnost je ve stránce nahrazena její aktuální hodnotou z pagebean.

Přílklad xhtml stránky a její odpovídající pagebean třídy, která zobrazuje aktuální datum:

Stránka ***HelloWorld.xhtml***:

```
<html xmlns="http://www.w3.org/1999/xhtml">
 <body>
   <div style="${style}"> Hello world at: ${date} </div>
 </body>
</html>
```

Odpovídající pagebean třída:

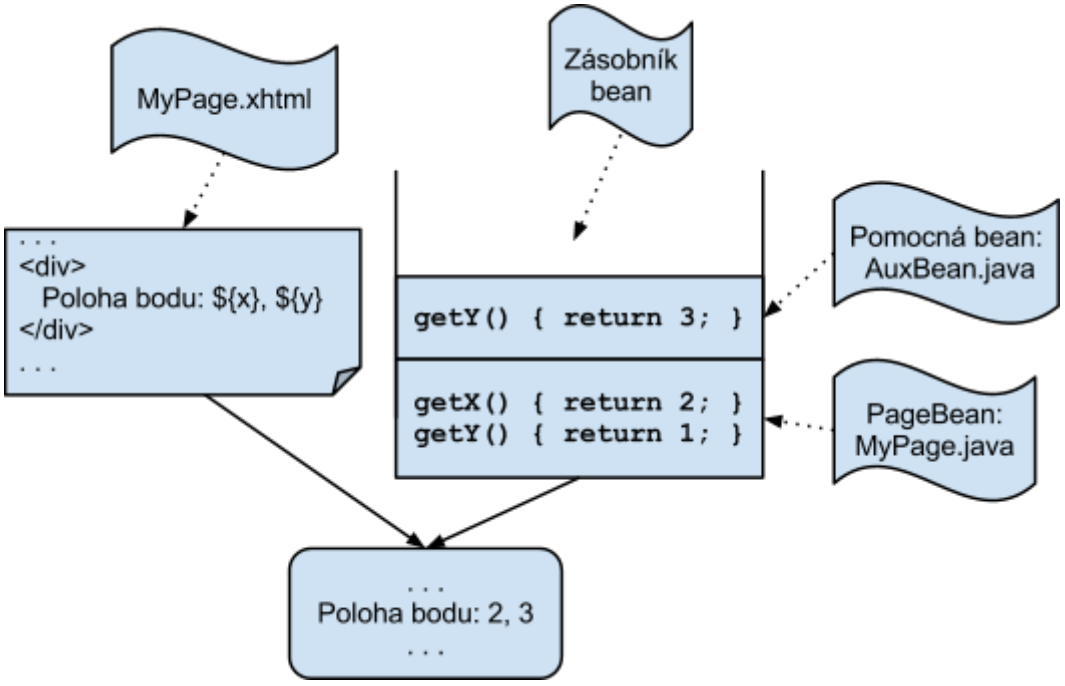
```
package pages;
import java.util.Date;
import org.smp.pages.Page;

public class HelloWorld extends Page {
 public Date getDate() {
   return new Date();
 }
```

```
 public String getStyle() {
   return "color:red";
 }
}
```

3. Zásobník bean

Pagebean je jenom jedna z bean, které se mohou účastnit generování stránky. Kromě pagebean třídy mohou být vlastnosti definovány i v dalších pomocných beanách, což mohou být libovolné javovské objekty (POJO). Všechny beany, které se účastní generování stránky, jsou uloženy na zásobníku, na jehož dně je umístěna právě pagebean. Při vyhodnocování vlastnosti se zásobník prohledává od vrcholu, takže vlastnosti se mohou zastiňovat:



Zásobník bean se v pagebean ovládá operacemi **Page.pushbean** a **Page.popBean**. Tyto metody je možno volat v metodě **Page.createContent**, kterou Simpleton volá před vlastním generováním stránky. Předcházející obrázek odpovídá následujícímu programu:

```
package pages;

public class AuxBean {
 public int getY() {
   return 3;
 }
}
. . .
```

```
package pages;
import org.smp.pages.Page;

public class MyPage extends Page {
 public int getX() { return 2; }
 public int getY() { return 1; }

 @Override
 public void createContent() {
   pushBean(new AuxBean());
 }
}
```

Hodnoty vlastností nemusí být definovány pouze pomocí getrů. Pokud je pomocná beana deklarována jako mapa (implementuje rozhraní **Map**), potom jsou jako hodnoty vlastností chápány položky uložené v této mapě. Pro pagebean je navíc možné definovat hodnoty vlastnosti voláním metody **Page.putItem** v metodě **Page.createContent**.

Příklad: variantní forma pagebean třídy pro stránku **HelloWorld.xhtml**:

```
package pages;
import org.smp.pages.Page;

public class HelloWorld extends Page {

 @Overide
 public void createContent() {
   Map<String, String> styleMap = new HashMap<>();
   styleMap.put("style", "color.red");
   putItem("date", new Date());
   pushBean(styleMap);
 }
}
```

Pokud jsou v xhtml stránce použity závorky **${ … }**, musí odpovídající vlastnost povinně existovat a její hodnota nesmí být null, zatímco pro závorky **%{ … }** je její existence nepovinná a v tom případě je ve výsledné stránce nahrazena prázdným řetězem.

Je definována jedna standardní vlastnost **string**, jejíž hodnotou je návratová hodnota metody **toString** té beany, která je na vrcholu zásobníku bean.

4. Elementy

Ve jmenném prostoru **www.smp.org** jsou definovány následující standardní elementy:

- **<smp:if>**
- **<smp:else>**
- **<smp:loop>**
- **<smp:option>**
- **<smp:include>**

5. If a Else

```
<smp:if condition = "..."> ... </smp:if>,
<smp:else condition = "..."> ... </smp:else>
```

jsou elementy pro podmíněné vkládání obsahu do stránky. Hodnotou atributu **condition** musí být jméno vlastnosti, která musí být typu **Boolean**, **Iterable**, **pole** nebo **String**. Tělo elementu se vkládá do generované stránky, pokud je hodnota vlastnosti **true** resp. je tvořena neprázdnou kolekcí, polem nebo stringem. Logika elementu **else** je inverzní.

Příklad: vložení hodnoty vlastnosti pouze tehdy, když je definovaná:

```
<smp:if condition = "user">
  User: ${user}
</smp:if>
```

6. Loop

<smp:loop iterable = "..."> ... </smp:loop> je element, který cyklicky generuje svoje tělo a typicky se používá pro generování tabulek. Atribut **iterable** určuje jméno povinné vlastnosti, jejíž hodnota musí být buď:

- pole nebo
- kolekce implementující interface **Iterable**.

Loop využívá pro generování zásobník bean. Při každém kroku iterace se na vrchol zásobníku uloží aktuální prvek kolekce, přes kterou se iteruje. Hodnoty vlastností této beany se tak použijí při generování těla elementu **loop**.

Příklad: fragment stránky **Customers.xhtml:**

```
<smp:if condition = "customers">
 <table>
   <smp:loop iterable="customers">
     <tr>
       <td>
         ${id}
       </td>
       <td>
         ${name}
       </td>
     </tr>
   </smp:loop>
 </table>
</smp:if>
```

Odpovídající třídy:

```
package model;
public class Customer {
```

```
  . . .
  public int getId() { . . . }
  public String getName() { . . . }
}

package pages;
import org.smp.pages.Page;

public class Customers extends Page {
  public Collection<Customer> getCustomers() {
     return  ...;
  }
}
```

7. Include

Element **<smp:include/>** je povolen pouze v šablonách (viz. dále) a na jeho místo se vloží vlastní obsah stránky.

8. Option

Element **<smp:option/>** se chová stejně jako standardní element **<xhtml:option/>** (tj. kopíruje se na výstup) s výjimkou atributu **selected**, který se při hodnotě **false** vynechává.

9. Combobox

Dynamický combobox se vytváří iterací elementů **<smp:option/>**:

```
<select name="sex">
 <smp:loop iterable="sexValues">
    <option value="${id}">${string}</option>
 </smp:loop>
</select>

package pages;
import org.smp.pages.Page;

class SexChooser extends Page {
 enum Sex {
   MALE,
   FEMALE;
   public int getId() {
     return ordinal();
   }
 }

 public Sex[] getSexValues() {
   return Sex.values();
```

```
  }
}
```

10. Šablony

Šablona je speciální xhtml stránka, která tvoří společný design pro skupinu obyčejných stránek. Šablona musí být doprovázená třídou, která je podtřídou třídy **org.smp.pages.Template**. Protože třída **Template** je podtřída třídy **Page**, je možno dynamický obsah šablon vytvářet stejně jako obsah stránek. Obyčejná stránka se při generování umístí na to místo, kde je v šabloně použit element <**smp: include**>.

Příklad **CustomersTemplate.xhtml:**

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:smp="http://www.smp.org">
. . .

<body>
 <table>
   <tr>
     <td>
       ${head}
     </td>
   </tr>
   <tr>
     <td>
       <smp:include>Template Content</smp:include>
     </td>
   </tr>
 </table>
<body>
</html>
```

Obyčejná xhtml stránka, která má být obklopena šablonou, musí mít ve své pagebean třídě předefinovanou metodu **getTemplatePath,** která vrací absolutní cestu stránky šablony:

```
package pages;
import org.smp.pages.Page;

public class CustomersTemplate extends Template {
 public String getHead() { return "..."; }
}

public class Customers extends Page {
   @Override
   public String getTemplatePath() {
       return "/CustomersTemplate.xhtml";
   }
}
```

Z vkládané stránky se na místo elementu <**smp:include**> vloží obsah jejího elementu <**body**>.

11. Individualizace stránek

Pokud je potřeba, aby stránka zobrazovala vlastnosti nějaké beany, která je určená dynamicky např. parametrem requestu, je možno příslušnou beanu uložit na zásobník bean:

```
package pages;
import org.smp.pages.Page;

public class EditCustomer extends Page {

   @Override
   public void createContent() throws SmpException {
      // customer id
       String id = getParameter("id"));
       Customer cust = findCustomer(id);
       pushBean(customer);
   }
}
```

Vlastnosti je možné použít pro vytváření parametrů requestu:

Příklad: fragment stránky, zobrazující tabulku zákazníků a u každého zákazníka odkaz na stránku umožňující editaci:

```
<table>
  <smp:loop iterable="customers">
     <tr>
        <td>
          ${id}
        </td>
        <td>
          ${name}
        </td>
        <td>
          <a href="EditCustomer.xhtml?id=${id}">Edit</a>
        </td>
     </tr>
  </smp:loop>
</table>
```

V odpovídající pagebean je využita hodnota parametru **id** pro individualizaci stránky.

12. Akce

Akce jsou třídy, které definují aplikační logiku aplikace. Třída akce musí být následníkem třídy **org.smp.actions.AbstractWebAction**, která má definovanou abstraktní metodu

execute. Jestliže server přijme požadavek na provedení akce, vytvoří se objekt akce a zavolá se nad ním metoda **execute**. Obvyklou činností metody **execute** je validace parametrů obsažených v requestu a podle jejího výsledku buď:

- redirect na novou stránku, pokud se validace zdařila, nebo
- uložení chybovými hlášek jako vlastností s a forward na původní pagebean.

Příklad formuláře a odpovídající akce pro přidávání nějaké položky:

```
. . .
<form>
 <input type="text" name="itemCode"/>
 %{errorItemCode}
 <input type="submit" formaction="items.AddItem.do"
                                          method="POST"/>
</form>

package items;
import org.smp.actions.AbstractWebAction;

public class AddItem extends AbstractWebAction {

 @Override
 public void execute() throws
              ServletException, IOException, SmpException {
   String itemCode = getParameter("itemCode");
   if (isOk(itemCode)) {  // validace
      addItem(itemCode); // business logic
      redirect("/Index.xhtml");
   } else {
     Page pageBean = createPageBean("/AddItem.xhtml");
     pageBean.putItem("errorItemCode", "Invalid item code");
     forward(pageBean);
   }
}
```

13. Repopulace

Pokud se nezdaří validace hodnot zadaných ve formuláři, je vhodné, aby se při opakovaném zobrazení formuláře vstupní elementy naplnily původními nezvalidovanými hodnotami (tzv. repopulace). To je možné dosáhnout použitím použitím závorek **%{. . .}** v hodnotě atributu **value** elementu **input.** Předcházející případ s repopulací:

```
<form>
 <input type="text" name="itemCode" value=%{itemCode}/>
 %{errorItemCode}
 <input type="submit" formaction="items.AddItem.do"
                                          method="POST"/>
</form>
```

Pro vytvoření beany obsahující repopulované položky je možno použít metodu **AbstractWebAction.requestParamsToMap**, která vytvoří mapu a naplní ji parametry z requestu:

```
package items;
import org.smp.actions.AbstractWebAction;

public class AddItem extends AbstractWebAction {

 @Override
 public void execute() throws
              ServletException, IOException, SmpException {
   String itemCode = getParameter("itemCode");
   if (isOk(itemCode)) {  // validace
      addItem(itemCode); // business logic
      redirect("/Index.xhtml");
   } else {
     Page pageBean = createPageBean("/AddItem.xhtml");
     Map<String, Object> mapBean = new HashMap<>();
     requestParamsToMap(mapBean); // repopulation
     mapBean.put("errorItemCode", "Invalid item code");
     forward(pageBean);
   }
}
```

Pro repopulaci nastavení combo boxu se používá standardní atribut **selected** elemetu **option**. Hodnotou atributu musí být vlastnost, která bude mít hodnotu **true** pro vybraný prvek a **false** pro ostatní prvky:

```
<select name="sex">
 <smp:loop iterable="sexValues">
   <smp:option value="${id}" selected="${selectedSex}">
     ${string}
   </smp:option>
 </smp:loop>
</select>
```

Vlastnost **selectedSex** může být definována např. v pagebean třídě:

```
public boolean getSelectedSex() {
 return getItem("id") == ...
}
```

Metoda **Page.getItem** vrací hodnotu vlastnosti uložené na zásobníku bean. Protože se vlastnost **selectedSex** vyhodnocuje v každém kroku iterace, je hodnotou metody **getItem("id")** **id** toho elementu, který se právě nachází na vrcholu zásobníku.

14. Podpora validace

Třída **org.smp.actions.AbstractWebAction** obsahuje generickou metodu **<T>void validateItem(T type, Map<String, Objects> errors)**, která usnadňuje implementaci validací vstupních hodnot. Metoda předpokládá, že pro vstupní hodnotu, kterou chceme validovat, existuje reprezentační třída obsahující statickou validační a konverzní metodu **validate**:

```
package model;
import org.smp.ValidationException;

public class Age {
 private int age;
 public Age(int age) {
   this.age = age;
 }

 public static Age validate(String inp) {
    try {
      int age = Integer.parseInt(inp);
      if (age <= 0)
          throw new ValidationException("Negative age");
      return new Age(age);
    } catch (NumberFormatException ex) {
      throw new ValidationException("Invalid age format");
   }
}
```

Pokud je taková třída definovaná, tak příslušná akce může vyvolat validaci použitím metody **validateItem**:

```
import org.smp.actions.AbstractWebAction;
public class SetAge extends AbstractWebAction {

   @Override
   public void execute() throws
              ServletException, IOException, SmpException {
       Map<String, Object> errorBean = new HashMap<>();
       Age age = validateItem(Age.class, errorBean);
       if (errorBean.isEmpty()) { // validation OK
           setAge(age);           // business logic
           redirect("/Index.xhtml");
       } else {
           Page pageBean =
                       createPageBean("/AddCustomer.xhtml");
           requestParamsToMap(errorBean);   // repopulation
           pageBean.pushBean(errorBean);
           forward(pageBean);
       }
   }
}
```

Metoda **validateItem** zajišťuje automatické zavolání metody **Age.validate** s request parametrem **age** a v případě neúspěchu validace uloží chybovou hlášku z výjimky **ValidationException** do **errorBean** pod klíčem **errorAge**. Příslušný formulář má potom následující tvar:

```
<form>
  <input type="text" name="age" value="%{age}"/>
  %{errorAge}
  <input type="submit" formaction="SetAge.do" method="POST"/>
</form>
```

15. Konfigurace

Simpleton neobsahuje žádný explicitní konfigurační soubor. Konfigurace se chápe jako služba, kterou může aplikace poskytnout frameworku. Konfigurační služba musí implemetovat rozhraní **org.smp.conf.Configuration**:

```
package org.smp.conf;

public abstract class Configuration {
 public abstract String charSet();           // default:"utf-8"

 public abstract String pagesPackage();      // default:"pages"

 public abstract ElementRenderer elementRenderer(
                 String elementName,
                 String elementNamespaceURI);

 public abstract String getResourceBundleName();
                              // default: "org/smp/Bundle"

 public abstract boolean isDebug();          // default: false

 public abstract void isAllowed(String resource,
     HttpServletRequest request) throws SmpException;
                              // default: empty
}
```

Aplikace, která potřebuje změnit implicitní nastavení, musí implementovat rozhraní **Configuration** (např. tak, že deklaruje podtřídu třídy **DefaultConfiguration)**, a zaregistrovat je jako službu pomocí standardního javovského mechanismu registrace služeb. Framework vyhledává zaregistrovanou službu pomocí třídy **java.util.ServiceLoader**, a pokud ji nenalezne, použije implicitní konfiguraci.

16. Bezpečnost

Pro kontrolu přístupu ke chráněným zdrojům je možno využít konfigurační metodu **isAllowed**. Ta se volá před přístupem ke každému zdroji, a pokud uživatel není autorizován,

musí hodit výjimku **AccessViolationException**. Ta se potom projeví jako standardní http chyba 404.

Pokud se v Simpletonu vytváří webová aplikace Java EE, tak se pro zabezpečený přenos dat, autentizaci uživatelů a autorizaci mohou využít bezpečnostní domény definované konfiguračním souborem web.xml.

17. Zákaznické elementy

Aplikace může rozšířit standardní sadu elementů o zákaznický element. Zákaznický element je tvořen singletonem, který je následníkem třídy **org.smp.elements.ElementRenderer**. Třída musí implementovat metodu **printElement**, která definuje výstupní tvar elementu. Příklad zákaznického elementu <**mns:currentTime**>, který zobrazuje aktuální čas a je definován v namespace **org.mynamespace** s předponou **mns**:

```
public class CurrentTime extends ElementRenderer {

 public static CurrentTime instance = new CurrentTime();

 @Override
 public void printElement(Page page, Element element,
           Appendable out) throws IOException, SmpException {
   out.append(new Date().toString());
 }
}
```

Registrace zákaznického elementu se provede v konfigurační službě.

Příklad: registrace elementu <**mns:currentTime**/>:

```
public class Konfigurace extends DefaultConfiguration {

 @Override
 public ElementRenderer elementRenderer(String elementName,
                                 String elementNamespaceURI) {
    if ("currentTime".equals(elementName) &&
           "org.mynamespace".equals(elementNamespaceURI) {
         return CurrentTime.instance;
       }
       return super.elementRenderer(elementName,
                                    elementNamespaceURI);
   }
}
```

Použití:

```
<html xmlns="http://www.w3.org/1999/xhtml"
            xmlns:smp="http://www.smp.org"
            xmlns:mns="org.mynamespace>"
```

```
  <body>
    <mns:currentTime/>
  </body>
</html>
```

18. Architektura

Simpleton používá pro parsování xhtml stránek knihovnu ***JDom***. Pro každou stránku je vytvořen DOM model a tento model je kešován jako položka mapy slabých referencí. Celá mapa potom tvoří jediný atribut aplikačního kontextu s klíčem: **SimpletonXHTMLPages**. Příchozí požadavky (requests) jsou zpracovávány dvěma servlety, které slouží jako frontkontroléry:

- první z nich je standardně namapován na příponu **.xhtml** a zpracovává požadavky na stránky,
- druhý front kontrolér je namapován na příponu **.do** a provádí zpracování požadavků na provedení akcí.

Každá pagebean obsahuje zásobník, jehož elementy jsou jednotlivé beany. Při hledání hodnoty vlastnosti se postupuje od vrcholu směrem ke dnu zásobníku, takže vlastnosti se mohou zastiňovat. Do zásobníku se ukládá:

- implicitně:
    - pokud je stránka vnořená do šablony, tak pagebean šablony
    - pagebean stránky,
    - prvek kolekce elementu **<loop/>**.
- explicitně: metodou **pushBean**.