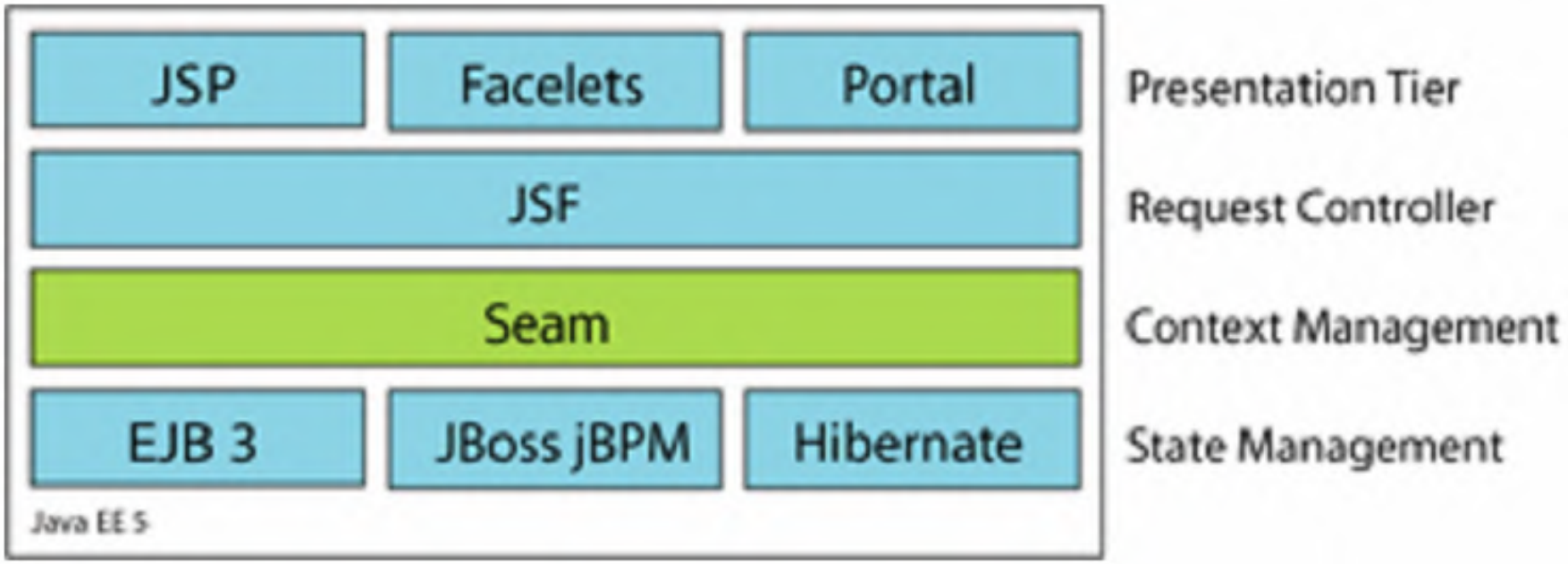
JBoss Seam

Framework Java EE 5

JBoss Seam

- Framework dla aplikacji Java EE 5
- Wykorzystywana technologia
  - EJB 3.0
  - JavaServer Faces 1.2
  - Java Persistence API
- Integracja z wieloma mniejszymi projektami

Czym jest Seam ?



http://docs.jboss.com/seam/

Czym jest Seam ?

- Integracja JSF z EJB 3.0

  Jednymi z najlepszych cech Java EE 5 jest właśnie JSF oraz EJB 3.0

  EJB 3.0 – logika biznesowa, obiekty persistence

  JSF – warstwa prezentacyjna

  Oba te modele współpracują najlepiej razem, jednakże specyfikacja JEE 5 nie daje nam standardu do zintegrowania tych modeli.

Seam łączy te dwa modele (JSF i EJB3), eliminując kod sklejający te modele.

Dzięki czemu programista może się skupić na logice biznesowej aplikacji.

Integracja z AJAX

Seam wykorzystuję technologie AJAX

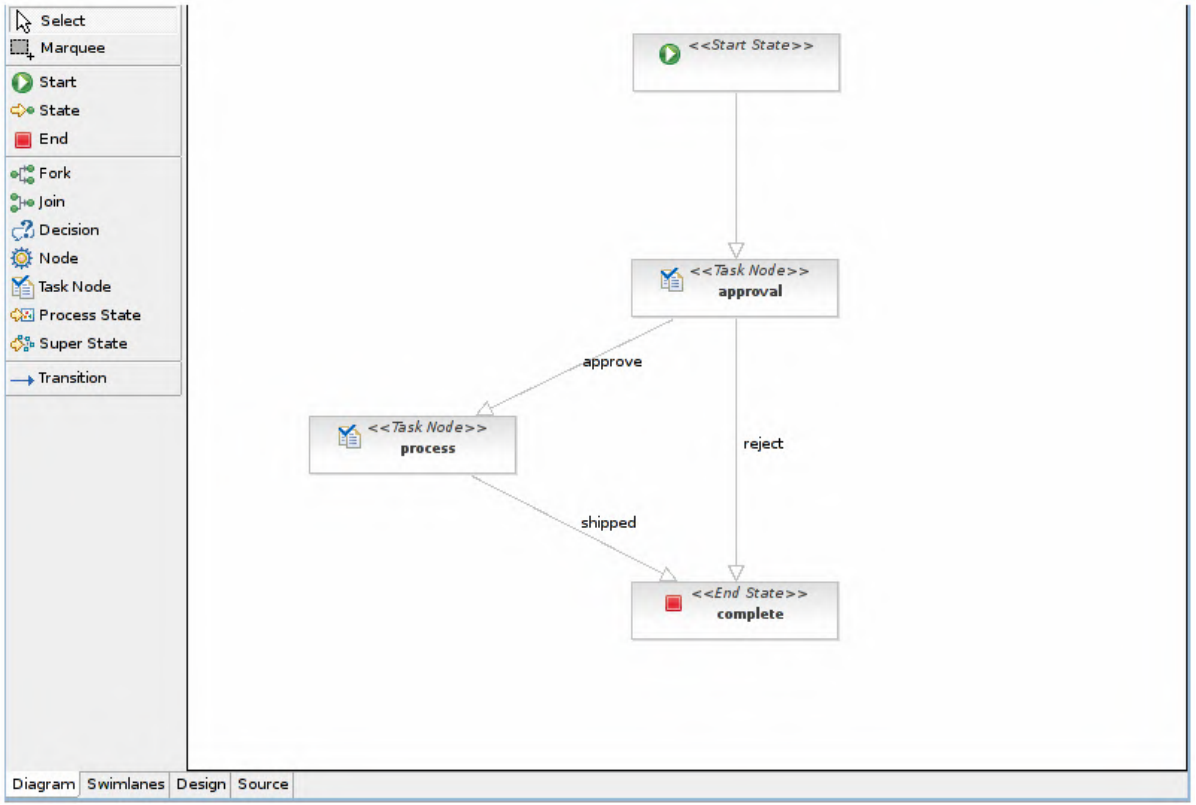
- ICEFaces
- Ajax4JSF

Możemy skorzystać z funkcjonalności AJAX'a w w ogóle nie dotykając kodu JavaScript.

Wykorzystanie tej technologii zwiększa wydajność serwera.

Integracja z jBPM

- jBPM - służy do definiowania procesów biznesowych



Bijection

- *Dependency injection* lub *Inversion of Control*
- „*Dependency outjection*” – (Seam)

  Komponenty seam’a mogą wchodzić w interakcje dwu-stronną razem ze sobą, „*dependency injection* w obie strony”.

Architektura

http://docs.jboss.com/seam/

Wymagania techniczne

Dla wersji Seam 1.2.1.GA

- JDK 5.0 i Ant 1.6+ (*z wyjątkiem* Ant 1.7 beta)
- JBoss AS 4.0.5+ (profil EJB 3.0), (możliwa współpraca z dowolnym serwerem aplikacji Java EE 5)
- Możliwa współpraca z serweremTomcat

Instalacja:

W głównym katalogu seam'a w pliku `build.properties` należy ustawić ścieżke do JBoss'a (lub Tomcata)

Czas na kawałek kodu.

User.java

```
@Entity
@Name("user")
@Scope(SESSION)
@Table(name="users")
public class User implements Serializable
{
   private static final long serialVersionUID = 18814134643L;

   private String username;
   private String password;
   private String name;

   public User(String name, String password, String username)
   {
      this.name = name;
      this.password = password;
      this.username = username;
   }
```

User.java cd.

```
public User() {}

@NotNull
public String getName()
{
   return name;
}

public void setName(String name)
{
   this.name = name;
}

@NotNull @Length(min=5, max=15)
public String getPassword()
{
   return password;
}
```

RegisterAction.java

```
@Stateless
@Name("register")
public class RegisterAction implements Register
{

   @In
   private User user;

   @PersistenceContext
   private EntityManager em;

   @Logger
   private static Log log;

   public String register()
   {
      List existing = em.createQuery(
      "select u.username from User u where u.username=#{user.username}")
         .getResultList();
```

RegisterAction.java cd.

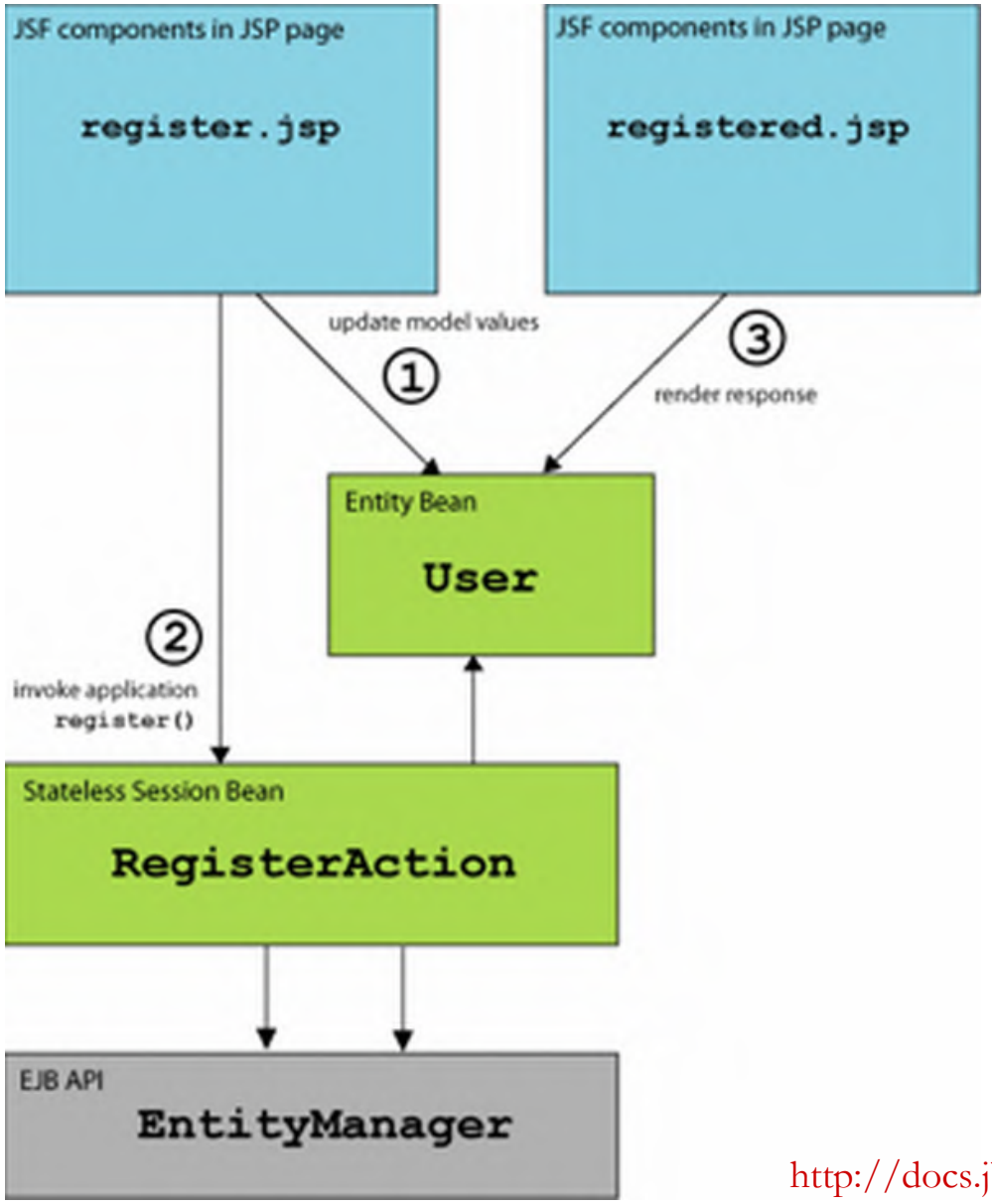
```
      if ( existing.size()==0 )
      {
         em.persist(user);
         log.info("Registered new user #{user.username}");
         return "/registered.jspx";
      }
      else
      {
         FacesMessages.instance().add("User #{user.username} already exists");
         return null;
      }
   }
}
```

register.jsp

```
</head>
<body>
  <f:view>
    <h:form>
      <table border="0">
        <s:validateAll>
          <tr>
            <td>Username</td>
            <td><h:inputText value="#{user.username}" required="true"/></td>
          </tr>
          <tr>
            <td>Real Name</td>
            <td><h:inputText value="#{user.name}" required="true"/></td>
          </tr>
          <tr>
            <td>Password</td>
            <td><h:inputSecret value="#{user.password}" required="true"/></td>
          </tr>
        </s:validateAll>
      </table>
      <h:messages/>
      <h:commandButton value="Register" action="#{register.register}"/>
    </h:form>
  </f:view>
</body>
</html>
</jsp:root>
```

registered.jsp

```
<jsp:directive.page contentType="text/html"/>
<html>
<head>
  <title>Successfully Registered New User</title>
</head>
<body>
  <f:view>
    Welcome, <h:outputText value="#{user.name}"/>,
    you are successfully registered as <h:outputText value="#{user.username}"/>.
  </f:view>
</body>
</html>
</jsp:root>
```

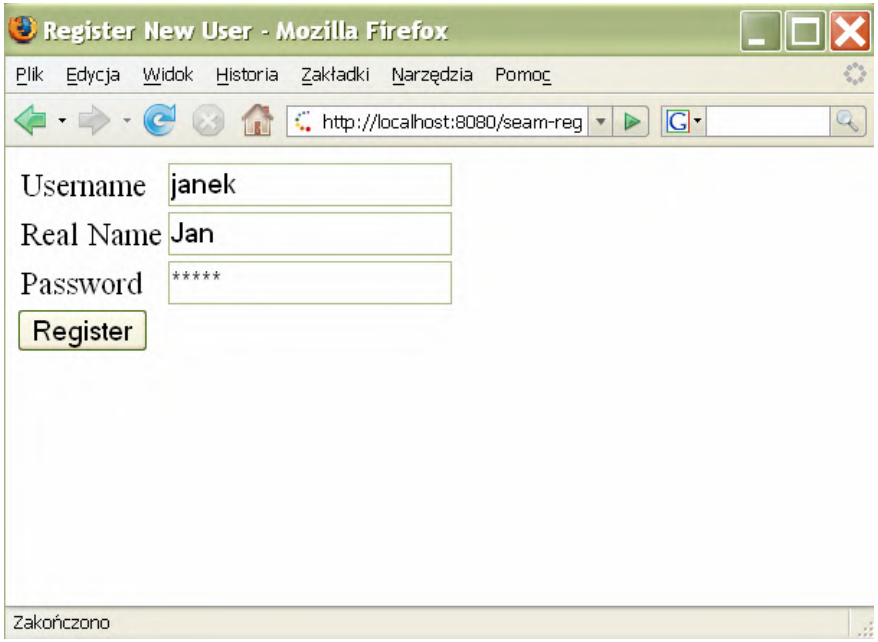


http://docs.jboss.com/seam/

1. Wysyłamy dane z formularza, JSF próbuje Uzyskać od Seam'a obiekt **user.**

2. Seam tworzy obiekt (User), i umieszcza ten obiekt kontekście sesji.

3. Wszystkie wartości z formularza są sprawdzane pod kątem poprawności, min. Hibernate Validator (@NotNull, @Length).




```
<tr>
  <td>Username</td>
  <td><h:inputText value="#{user.username}"
</tr>
<tr>
```

4a. Jeżeli którakolwiek anotacja nie jest spełniona, następuję ponowne wyświetlenie strony z komunikatem.

4b. Jeżeli wszystkie dane przechodzą proces walidacji następuję wypełnienie encji user danymi z formularza.

```
<h:messages/>
<h:commandButton value="Register" action="#{register.register}"/>
```

5. Następnie JSF pyta Seam'a o zmienną **register** , Seam znajduję w dostępnym kontekście RegisterAction – bean Bezstanowy, zwraca go. JSF wywołuje na tym komponencie Metodę register().
6. Seam przechwytuję wywołanie tej metody i wstrzeliwuję encję user z kontekstu do obiektu RegisterAction.
7. Następnie metoda register() sprawdza w bazie czy istnieje w bazie obiekt o podanej nazwie, jeżeli istnieje to funkcja tworzy komunikat FacesMessages z właściwym komunikatem i zwraca null, w przeciwnym przypadku zwraca strone registered.jsp.

Anotacje Seam'a (komponent)

- @Name – definiuje nazwę komponentu pod którą będzie widoczny komponent w kontekście
- @Scope – definiuje zasięg kontekstu, możliwe: (EVENT, PAGE, CONVERSATION, SESSION, BUSINESS_PROCESS, APPLICATION, STATELESS)
- @Role @Roles – dany obiekt może występować pod różnymi nazwami i w róznych kontekstach

Anotacje Seam'a (injection)

- @In – jeżeli Seam napotka obiekt oznaczony ta annotacją, zacznie poszukiwania tego obiektu od najwęższego kontekstu
- @Out – oznaczenie obiektu to wyeksportowania do kontekstu, opcjonalnie (value - nazwa komponentu, scope – kontekst docelowy), wspomniane Injection dependency.
- @Unwrap – anotacja obecna przy metodach getter, gdy komponent ma jakąś metodę tak oznaczoną, to zamiast komponentu zwracany jest wynik metody oznaczonej tą anotacją.
- @Factory – występuje wraz z nazwą komponentu do którego jest przyporządkowana. Wywołanie metody oznaczonej tą annotacją następuję gdy komponent do którego ona się odnosi jest pusty, metoda ta domyślnie powinna tworzyć ten obiekt.

Anotacje Seam'a (cykl życia)

- @create – metoda oznaczona tą anotacją jest wywołowana gdy komponent jest tworzony, dotyczy tylko JavaBeans i SFSB
- @destroy – jak wyżej, tylko wtedy gdy komponent jest usuwany

Anotacje Seam'a (JSF)

- @DataModel – anotacja występowała już w JSF, gdy seam napotka w kodzie liste oznaczoną tą anotacja, to otacza ją wrapperem dataModel i umieszcza w kontekście, dzięki czemu można się do niej odwołać na stronie jsf w tagu <h:dataModel value="#{nazwa}"> i wyświetlić całą listę
- @DataModelSelection – na liście wyświetlonej w/w tagiem, można podlinkować każdy obiekt, wtedy po kliknięciu na nim, seam będzie automatycznie kopiował obiekt do zmiennej oznaczonej tą anotacją.
- DataModelSelectionIndex – działanie jak anotacji wyżej, z tą różnicą że zwracany jest numer obiektu z listy.

Konwersacje

W Java EE 5 (JSF) mogliśmy „przechowywać stan” aplikacji w następujących zasięgach:

- *request* (aktualne żadanie, mała funkcjonalność)
- *session* (dotyczy sesji przeglądarki, większa ale i tak nieduża funkcjonalność)
- *application* (dotyczy danych globalnie dostępnych, akurat w tym przypadku nas nie interesuje)

Wobec zapotrzebowania na zasięg pomiędzy request a session (znacznie węższym niż session i krótszym) w Seamie zostały stworzone *konwersacje (conversation)*.

Anotacje Seam'a (conversation)

- @Begin – anotacja może się pojawić przy metedzie, gdy zostanie wywołana metoda, to od momentu kiedy ta metoda zwróci cokolwiek różnego on null'a i nie podniesie wyjątku, to zaczyna się kontekst konwersacji
- @End – analogicznie jak wyżej, gdy zostanie wywołana metoda oznaczona tą anotacją, to gdy ta metoda zwróci cokolwiek różnego od null'a i nie podniesie wyjątku to oznacza, że konwersacja wcześniej zaczęta się kończy.

Użytkownik może mieć aktywnych kilka konwersacji, dzięki czemu w jednej przeglądarce można w kilku oknach pracować bez ryzyka na kolizje danych. Konwersacje mogą być też zagnieżdżane.

To nie koniec anotacji…

Jeszcze są anotacje typu:

- Zarządzanie prosesem jBPM (@StartTask, @BeginTask, @EndTask, @CreateProcess, @ResumeProcess)
- I wszystkie pozostałe dostępne w Java EE 5

Deskryptory

- components.xml – używane komponenty, JNDI
- web.xml – konfiguracja Sema i JSF (podobny plik w większości projektów)
- faces-config.xml - brak managed beans + SeamPhaseListener
- ejb-jar.xml – jak w EJB3 + integracja z SeamInterceptor
- persistence.xml – znaczenie jak w EJB
- application.xml – jak w EJB, konfiguracja głownych komponentów aplikacji

seam-gen

- Narzędzie seam-gen służy do generowania szkieletu aplikacji
- seam setup – konfiguracja projektu (osobno dla każdego projektu)
- seam new-project – tworzy nowy projekt wcześniej skonfigurowany
- seam new-action – „nowa strona z bezstanową akcją” – bean bezstanowy
- seam new-form – jest tworzony bean stanowy z formularzem
- seam restart – rekompilacja

Aby wystarczyło pamięci…

- Należy ustawić opcje JBossa dla JVM:

```
-Xms512m -Xmx1024m -XX:PermSize=256m -
  XX:MaxPermSize=512
```

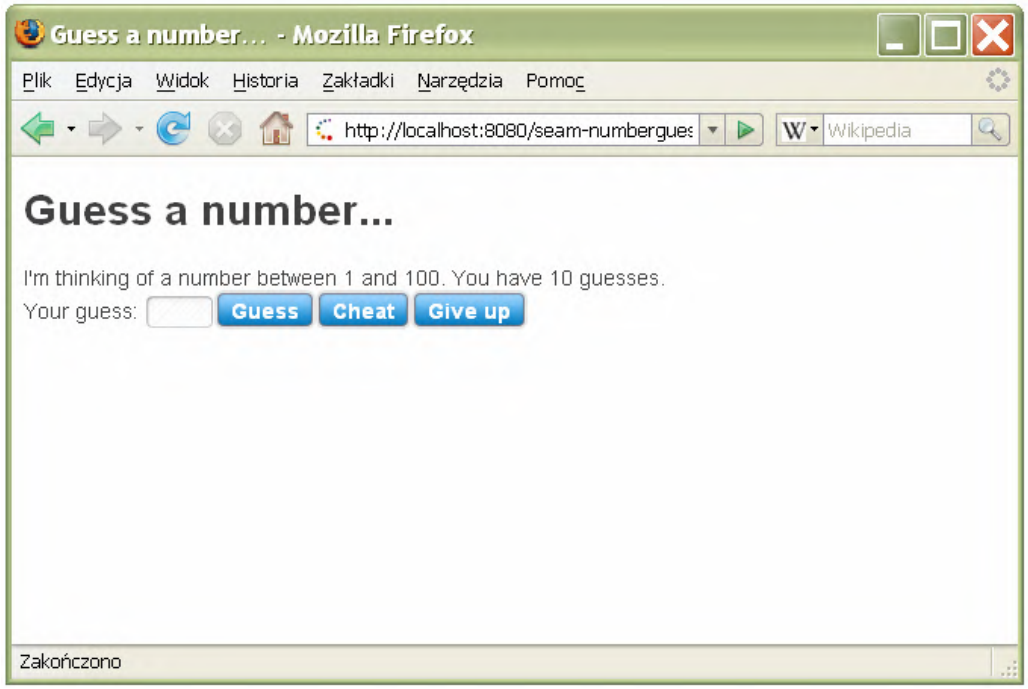
- A minimum to:

```
-Xms256m -Xmx512m -XX:PermSize=128m -
  XX:MaxPermSize=256
```

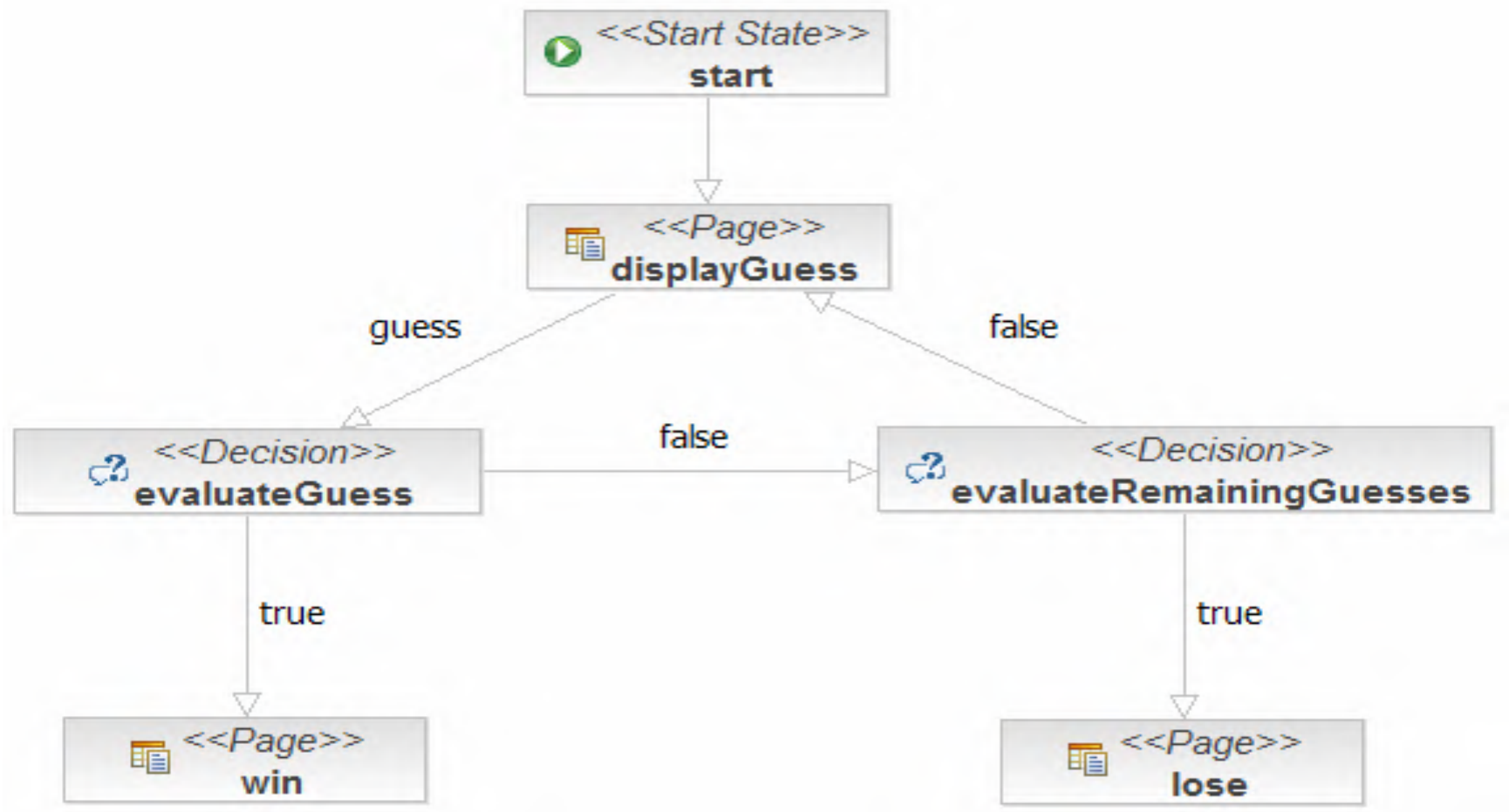
W przeciwnym przypadku po kilku akcjach spotka nas wyjątek `OutOfMemoryException`

Przykład z wykorzystaniem jBPM

- Przykład numberguess z dokumentacji Seam'a



numberguess - pageflow



jBPM

- jBPM to potężne narzędzie ułatwiające budowe aplikacji
- służy do definiowania procesów biznesowych

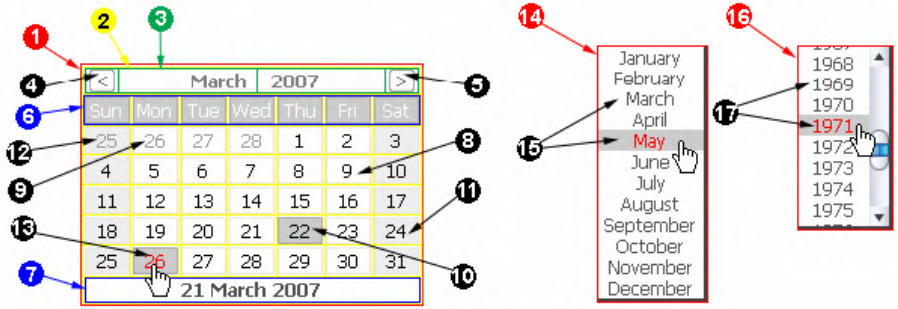
Testowanie

- TestNG
- JUnit

Seam ułatwia tworzenie testów, dostępna specjalna klasa `SeamTest` z której się dziedziczy przy pisaniu testów np. TestNG.

Świetna dokumentacja

- `seam-date-inMonth` — This class is applied to the table cell (`td`) elements that contain a date within the month currently displayed. (8)
- `seam-date-outMonth` — This class is applied to the table cell (`td`) elements that contain a date outside of the month currently displayed. (9)
- `seam-date-selected` — This class is applied to the table cell (`td`) element that contains the currently selected date. (10)
- `seam-date-dayOff-inMonth` — This class is applied to the table cell (`td`) elements that contain a "day off" date (i.e. weekend days, Saturday and Sunday) within the currently selected month. (11)
- `seam-date-dayOff-outMonth` — This class is applied to the table cell (`td`) elements that contain a "day off" date (i.e. weekend days, Saturday and Sunday) outside of the currently selected month. (12)
- `seam-date-hover` — This class is applied to the table cell (`td`) element over which the cursor is hovering. (13)
- `seam-date-monthNames` — This class is applied to the `div` control that contains the popup month selector. (14)
- `seam-date-monthNameLink` — This class is applied to the anchor (`a`) controls that contain the popup month names. (15)
- `seam-date-years` — This class is applied to the `div` control that contains the popup year selector. (16)
- `seam-date-yearLink` — This class is applied to the anchor (`a`) controls that contain the popup years. (15)



- http://docs.jboss.com/seam/1.2.1.GA/reference/en/

Materiały

- www.jboss.org
- Każdy projekt JBoss'a posiada bardzo dobrą dokumentacje.

Dziękuje. ☺

Pytania?