

# JSF

Internet programiranje  
IV godina, ETF Banjaluka

# JSF

- ▶ JSF predstavlja Java–bazirano aplikativno okruženje (*framework*) razvijeno 2004. godine od strane kompanije Sun Microsystems
- ▶ JSF – *component based framework*
- ▶ od razvoja inicijalne verzije, JSF je postao jedno od najpouzdanijih i najpopularnijih okruženja za razvoj naprednih web aplikacija, što dokazuje i konstantan porast broja JSF–baziranih aplikacija kao i njihove zastupljenosti u segmentu poslovnih aplikacija
- ▶ u prilog popularnosti okruženja ide i ogroman broj nadogradnji i eksternih biblioteka koje značajno podižu nivo funkcionalnosti aplikacije, bilo da se radi o bibliotekama koje doprinose razvoju bržih korisničkih interfejsa ili unapređenju poslovne logike
- ▶ mogućnosti razvoja biblioteka nisu ograničene arhitekturom sistema, pa je njihov broj u stalnom porastu

# JSF

## ▶ Java Server Faces

- skup Web-baziranih GUI kontrola i povezanih obrada
  - JSF obezbjeđuje veliki broj HTML-orijentisanih GUI kontrola, zajedno sa kodom koji obrađuje njihove događaje.
- nezavisan GUI kontrolni framework
  - JSF ima mogućnost da generiše i grafičke kontrole koje nisu HTML, koje koriste protokole koji nisu HTTP
- može se koristiti kao MVC framework za generisanje HTML formi, validaciju njihovih vrijednosti, realizaciju poslovne logike i prikazivanje rezultata

# JSF

- ▶ JSF:
  - skup ugrađenih ulazno–izlaznih komponenti
  - događajima vođen programski model, sa opcijama za obradu i reagovanje na događaje
  - model komponenti koji omogućava programerima serverske strane razvoj dodatnih komponenti
- ▶ širok spektar komponenti – od jednostavnih (npr., ulazna polja ili dugmad), do sofisticiranih (npr., tabele podataka i stabla)
- ▶ JSF sadrži sav potreban kod za obradu događaja i organizaciju komponenti
- ▶ aplikativni programeri mogu da zanemare sve nepotrebne detalje i da se usredsrede na sam razvoj aplikativne logike
- ▶ JSF je dio Java EE standarda – uključen je u svaki Java EE aplikativni server, i može jednostavno da se doda i na jednostavnije serverske implementacije, kao što je Tomcat

# JSF

## ▶ dobre strane

- ugrađene GUI kontrole – JSF sadrži skup API-ja i povezanih ugrađenih tagova koji omogućavaju kreiranje HTML formi sa kompleksnijim interfejsom
- obradu događaja – JSF omogućava jednostavno pisanje Java koda koji se poziva kada se forma submit-uje – Kod može da odgovara određenom dugmetu, promjeni određene vrijednosti, selekciji korisnika, ...
- managed bean-ove – omogućava automatsko popunjavanje bean-a i jednostavno procesiranje request parametara
- Expression Language – JSF sadrži koncizan i “moćan” jezik za pristup bean-ovima i kolekcijama

# JSF

- ▶ dobre strane
  - konverziju i validaciju polja forme – JSF ima ugrađene opcije za provjeru unesenih vrijednosti forme (u odnosu na traženi format), kao i za konverziju iz stringa u željeni tip podataka – u slučaju da postoji problem (nisu unesene tražene vrijednosti ili vrijednost nije u traženom formatu), forma se automatski ponovo prikazuje, sa porukom(ma) o grešci
  - centralizovanu konfiguraciju baziranu na datotekama – mnoge JSF vrijednosti se čuvaju u okviru XML ili property datoteka – ovakav pristup omogućava vršenje promjena, bez modifikovanja ili ponovnog kompajliranja Java koda – dodatno, prednost ovakvog pristupa je mogućnost da se Java i Web programeri usredsrede na specifične zadatke
  - konzistentan pristup – JSF ohrabruje konzistentnu upotrebu MVC pristupa u okviru aplikacije

# JSF

## ▶ loše strane

- potrebno više vremena za učenje – da bi se realizovao MVC pristup pomoću RequestDispatcher-a, potrebno je znati samo standardni JSP i servlet API, a da bi se realizovao MVC pristup pomoću JSF frameworka, potrebno je znati standardni JSP i servlet API i veliki i složeni framework koji je otprilike jednak po veličini sa osnovnim sistemom. Ovaj nedostatak dolazi do izražaja posebno kod manjih projekata, kratkih rokova, i manjeg iskustva razvojnog tima.

# JSF

## ▶ loše strane

- manje transparentno – sa JSF aplikacijama, dosta više stvari se dešava u pozadini, nego kod uobičajenih Java–baziranih Web aplikacija – kao krajni rezultat, JSF aplikacije su:
  - teže za razumijevanje
  - teže za testiranje i optimizaciju
- rigidni pristup
  - iz razloga baziranja na MVC pristupu u okviru JSF je teško koristiti druge pristupe
- razvoji alati
  - podrška postoji, ali u manjem obimu nego kod uobičajenih Java–baziranih Web aplikacija



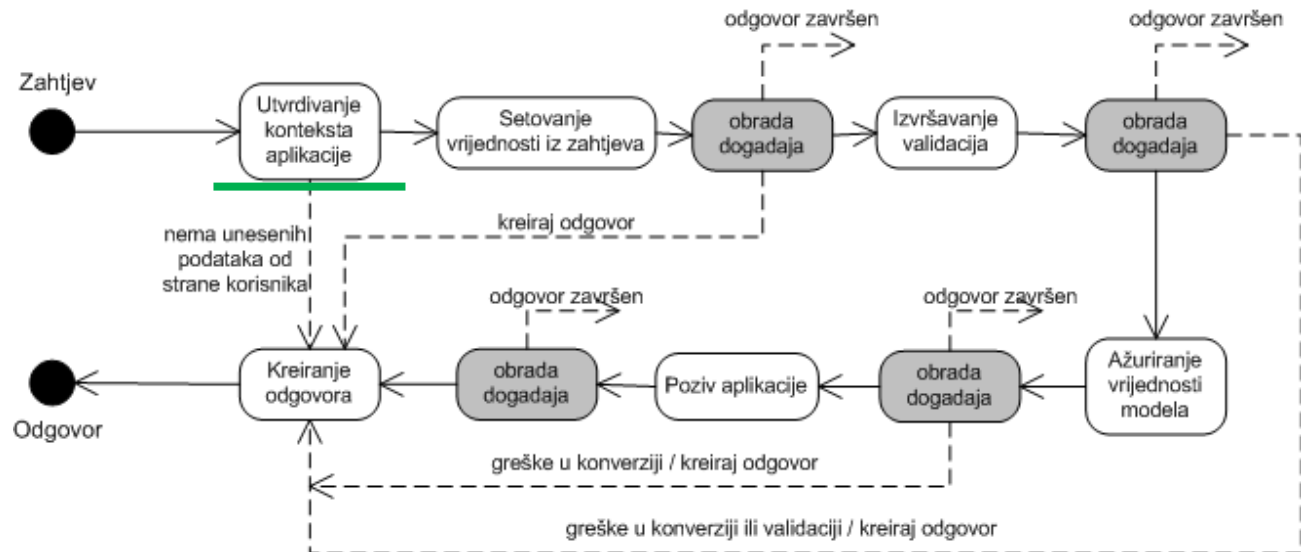
# JSF

- ▶ najznačajnije unapređenje kompletnog JSF okruženja dolazi sa pojavom verzije 2.0
- ▶ u ovoj verziji ispravljani su nedostaci otkriveni praktičnom primjenom i uvedene su značajne promjene u logici razvoja web aplikacija
- ▶ specifična priroda unesenih promjena arhitekture za posljedicu ima paralelnu egzistenciju dvije verzije okruženja i nakon objavljivanja konačne specifikacije nove verzije
- ▶ određene osobine onemogućavaju jednostavno unapređenje postojećih aplikacija, pa i dalje postoje primjene u kojima se starija verzija JSF okruženja zadržala kao dominantna

# JSF

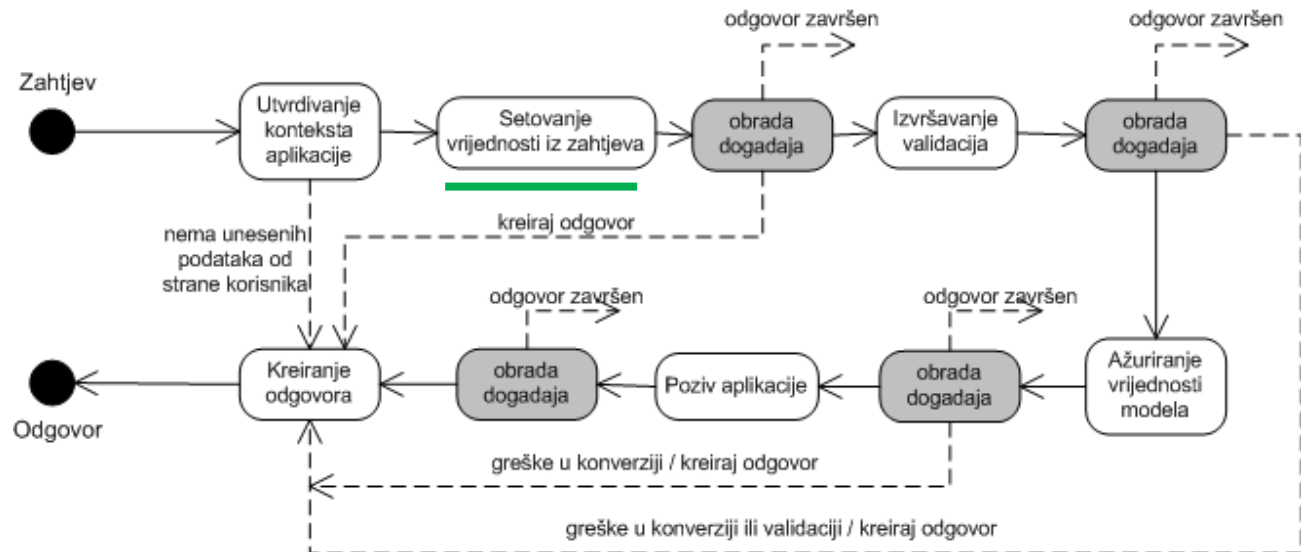
- ▶ polazna tačka svih funkcionalnosti JSF aplikacija (bez obzira na verziju specifikacije) leži u njenom ciklusu (*application life cycle*)
- ▶ JSF specifikacija definiše šest odvojenih faza koje čine ciklus svake aplikacije
- ▶ ciklus aplikacije predstavlja niz koraka kroz koje se prolazi od prijema, validacije i obrade korisničkih podataka, pa do kreiranja odgovora i njegovog slanja prema klijentu
- ▶ ciklus aplikacije je zadržan od inicijalne verzije okruženja
- ▶ unesene izmjene odnose se na olakšanu upotrebu okruženja i pojednostavljenje procesa razvoja JSF-baziranih aplikacija

# JSF



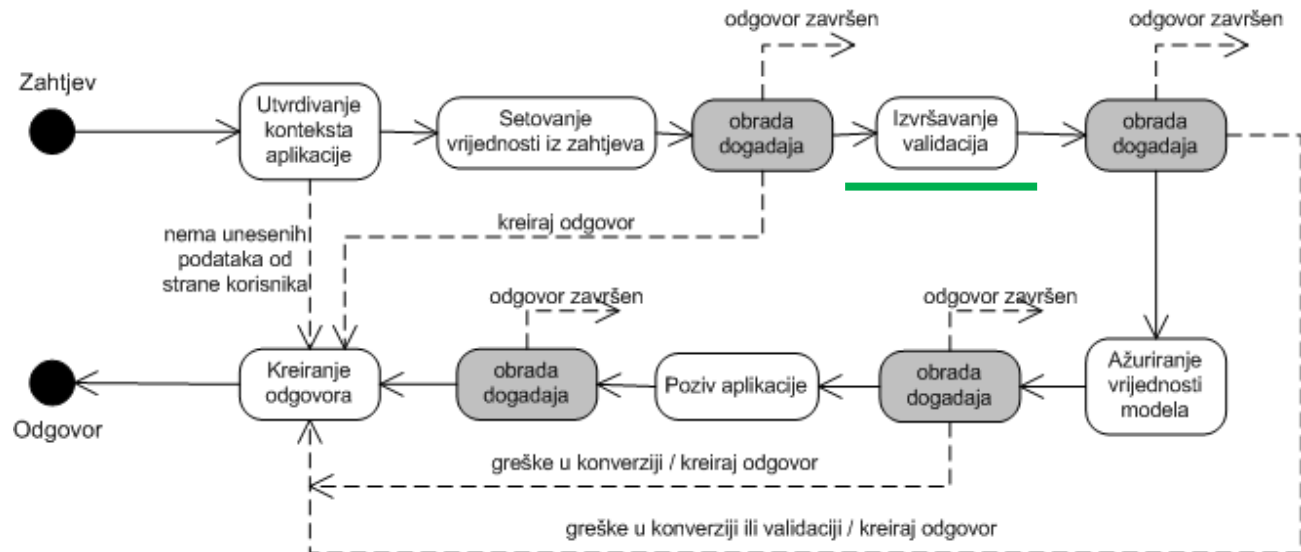
- ▶ JSF framework kontroler koristi view ID (uzet iz zahtjeva) kako bi pretražio komponente – ako view ne postoji JSF kontroler će kreirati novi, a ako postoji, koristiće ga – view sadrži sve GUI komponente, tj. stablo komponenti
- ▶ potrebno je razlikovati 3 view instance: new view, initial view i postback
- ▶ new view – JSF gradi view Faces strane i povezuje event handler-e i validatore sa komponentama – view se čuva u FacesContext objektu – ne postoji od JSF 1.2
- ▶ FacesContext objekat sadrži sve informacije koje JSF treba da bi upravljao stanjem GUI komponenti, u tekućem zahtjevu u tekućoj sesiji
- ▶ initial view – kada se stranica prvi put učitava – JSF kreira prazan view – sa inicijalnog view-a, JSF direktno prelazi na fazu kreiranja odgovora
- ▶ postback – korisnik se vraća na stranicu koju je ranije posjetio – view koji odgovara datoj strani već postoji – radi se restore – JSF radi rekonstrukciju – sljedeća faza nakon postback-a je faza "setovanje vrijednosti iz zahtjeva"

# JSF



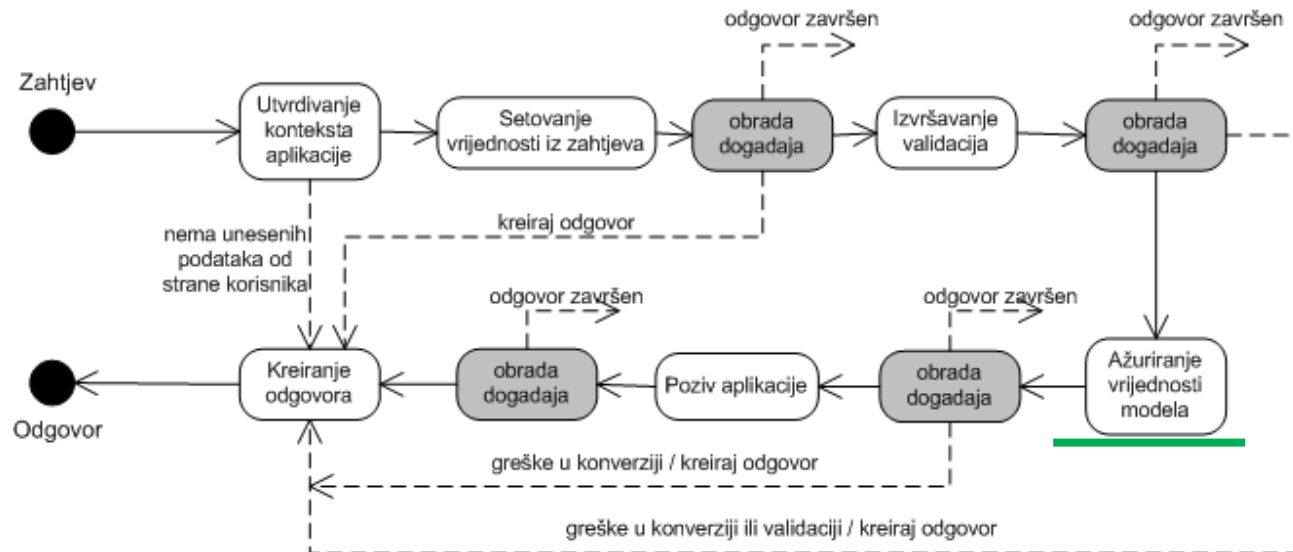
- ▶ u fazi setovanja vrijednosti iz zahtjeva prolazi se kroz sve komponente u okviru stabla komponenti – za svaki objekat komponenti se provjerava koja mu vrijednost pripada i ona mu se dodjeljuje
- ▶ vrijednosti komponenti se uzimaju iz request parametara, ili iz cookie-a ili header-a
- ▶ ako immediate event handling property komponente nije postavljan na true – vrijednosti se samo konvertuju (npr., ako je vrijednost polja Integer, vrijednost se konvertuje u Integer) – ako konverzija nije uspjela, generiše se poruka o grešci koja će biti prikazana za vrijeme faze “kreiranje odgovora”, zajedno sa svim validacionim greškama (smješta se u FacesContext) – koristi se kada nije potrebno izvršiti validaciju kompletne forme
- ▶ ako je immediate event handling property komponente postavljan na true, vrijednosti se konvertuju u odgovarajući tip i vrši se validacija – konvertovana vrijednost se smješta u komponentu – ako konverzija ili validacija nije uspješna, generiše se poruka o grešci...

# JSF



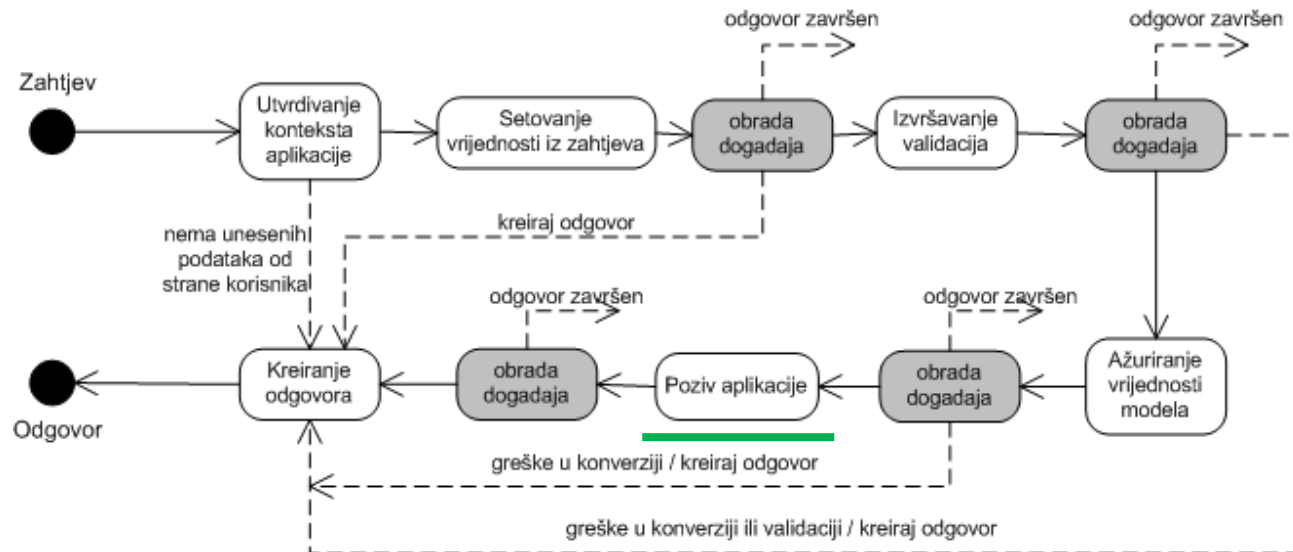
- ▶ u fazi provjere validnosti unesenih podataka okruženje pruža mogućnost prekida dalje obrade i generisanja odgovora koji informiše korisnika o neispravnom unosu
- ▶ poslati stringovi se prvo prebacuju u „lokalne“ vrijednosti, koje mogu biti objekti bilo kog tipa
- ▶ u ovoj fazi biće izvršena validacija vrijednosti svih komponenti, prema definisanim validacionim pravilima koja mogu biti predefinisana (ugrađena JSF validaciona pravila) ili definisana od strane programera
- ▶ unesene vrijednosti se provjeravaju prema validacionim pravilima – ako unesena vrijednost nije validna poruka o grešci se dodaje u FacesContext, a komponenta se označava kao nevalidna
- ▶ ako postoji nevalidna komponenta JSF prelazi u fazu “kreiranje odgovora” gdje će biti prikazan trenutni view sa porukama o validacionim greškama
- ▶ ako ne postoje validacione greške, JSF prelazi u fazu “ažuriranje vrijednosti modela”

# JSF



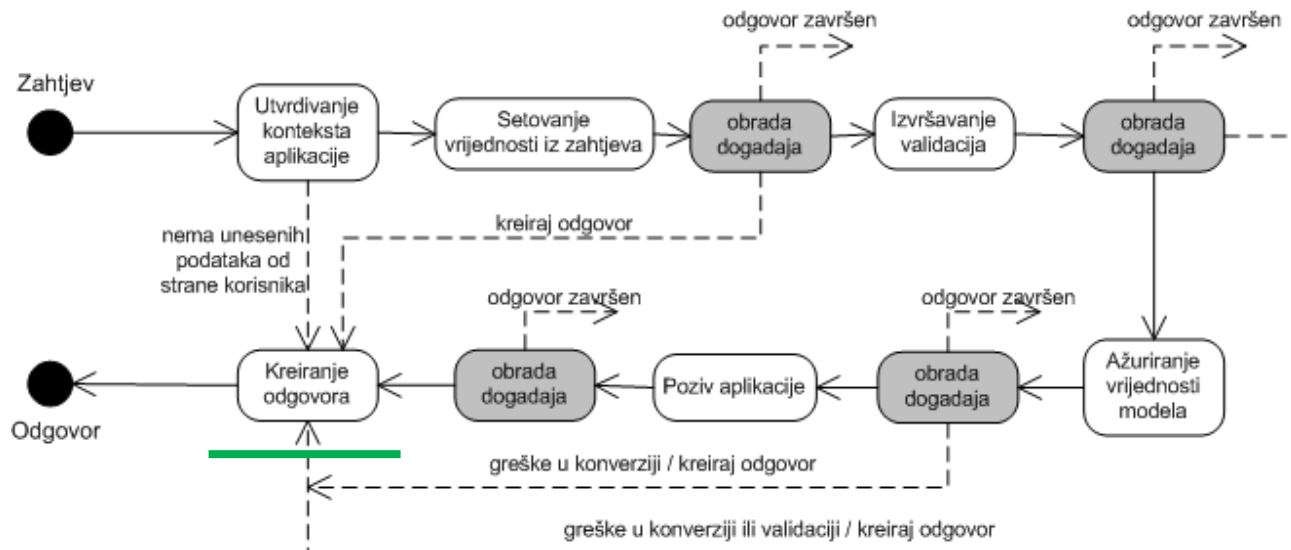
- ▶ u ovoj fazi se dobijene vrijednosti iz prezentacionog sloja prenose prema modelu podataka, kako bi se u sljedećoj fazi mogla izvršiti obrada uz pomoć procedura koje čine dio poslovne logike aplikacije
- ▶ ažuriraju se vrijednosti server-side modela, tako što ažuriraju property-je managed bean-ova
- ▶ samo property-ji bean-ova koji su vezani za vrijednost komponente će biti promijenjeni (ažurirani)
- ▶ kako se ova faza izvršava nakon validacije, može se sa sigurnošću znati da su vrijednosti validne, bar na nivou polja forme

# JSF



- ▶ prije generisanja odgovora koji će biti prezentovan korisniku, vrši se konačna obrada podataka proslijeđenih modelu, kao i odluka o navigacionom pravilu koje će biti upotrebjeno u daljem toku aplikacije
- ▶ JSF kontroler poziva metodu koja obrađuje podatke unesene sa forme, koji su konvertovani, nad kojim je izvršena validacija i koji su smješteni u managed bean-ove, tako da se sada mogu koristiti za izvršavanje business logike
- ▶ u ovoj fazi se generiše izlazni string, koji se šalje dijelovima odgovornim za navigaciju, gdje se izvršava poziv odgovarajuće stranice

# JSF



- ▶ u ovoj fazi kreira se odgovor i šalje klijentu.
- ▶ prikazuje se odgovarajući view, sa svim svojim komponentama u trenutnom stanju
- ▶ kada korisnik sa nove stranice pošalje novu formu, klikne na link ili na drugi način generiše novi zahtjev, startuje se novi ciklus

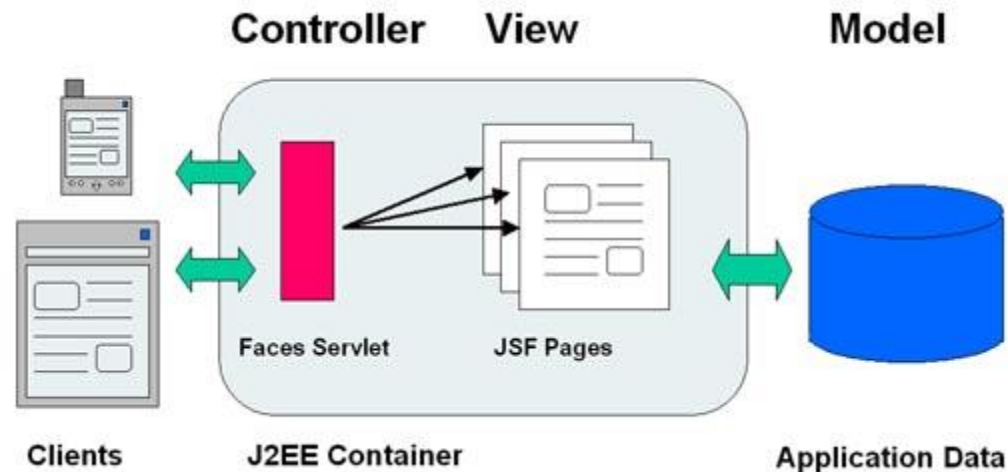


# JSF

- ▶ JSF pojednostavljuje razvojni proces web aplikacije obezbeđujući komponent–centričan pristup razvoju Java web user interface–a
- ▶ osnovne komponente JSF–a su:
  - API koji omogućava predstavljanje UI komponenti i manipulisanje njihovim stanjima, upravljanje događajima, validaciju na strani servera, konverziju podataka, definisanje navigacije između stranica, internacionalizaciju.
  - dvije tag biblioteke za predstavljanje UI komponenti u okviru JSP strane i za povezivanje komponenti sa serverskim objektima (JSF 1.2)

# JSF

- ▶ JSF framework prati Model-View-Controller (MVC) patern što JSF aplikacije čini dosta prilagodljivim i pogodnim za rad – user-interface kod (View) je razdvojen od aplikacionih podataka i logike (Model)



- ▶ JSF servlet (Controller) upravlja svom korisničkom interakcijom sa aplikacijom. On priprema JSF kontekst koji omogućava stranicama pristup aplikacionim podacima kao i zaštitu od neovlašćenog i neadekvatnog pristupa stranicama.

# JSF tagovi

- ▶ da bi mogli da se koriste JSF tagovi, svaka JSP strana na vrhu mora imati sljedeće dvije taglib direktive:

```
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
```

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
```

- ▶ prva uključuje html\_basic a druga jsf\_core tagove.
- ▶ html\_basic tag-ovi predstavljaju HTML form kontrole i druge osnovne HTML elemente i prikazuju podatke ili prihvataju podatke od korisnika (column, commandButton, commandLink, form, message, outputText, itd.)
- ▶ jsf\_core tagovi se koriste za ključne akcije koje su nezavisne od određenog render-a (na primer tagovi za rad sa događajima (actionListener), tagovi za konverziju podataka (converter, convertDateTime, convertNumber), tagovi za validaciju (validator, validateLength), loadBundle, param, subview, view itd.).

# Navigacija

- ▶ statička i dinamička
- ▶ definiše se u faces-config.xml konfiguracionoj datoteci

```
<navigation-rule>  
<from-view-id>/pages/first.jsp</from-viewid>  
<navigation-case>  
<from-outcome>second</from-outcome>  
<to-view-id>/pages/second.jsp</to-view-id>  
</navigation-case>  
</navigation-rule>
```

# Navigacija

- ▶ definisano je kako će se sa strane first.jsp (definisane u okviru from-view-id elementa) preći na stranu second.jsp (definisane u okviru to-view-id elementa).
- ▶ navigation-rule može imati proizvoljan broj navigation-case-ova od kojih svaki definiše koja se strana sljedeća učitava bazirano na logičkom ishodu (definisanom u okviru from-outcome)
- ▶ ishod može biti definisan od strane action atributa UICommand komponente koja vrši submit forme

```
<h:commandButton action="second" value="#{msg.button_text}" />
```

- ▶ ishod se takođe može dobiti kao povratna vrijednost akcione metode managed bean-a – ova metoda vrši neki proces da bi utvrdila ishod (npr. metoda može da provjeri da li je par korisničko ime – lozinka koju je korisnik unio ispravna) – ako jeste metoda vraća success, a u suprotnom slučaju failure – u prvom slučaju korisnik bi bio prebačen na, npr., pozdravnu stranicu u okviru sajta, a u drugom ponovo bi bio vraćen na login stranicu, uz odgovarajuću poruku o grešci

# Navigacija

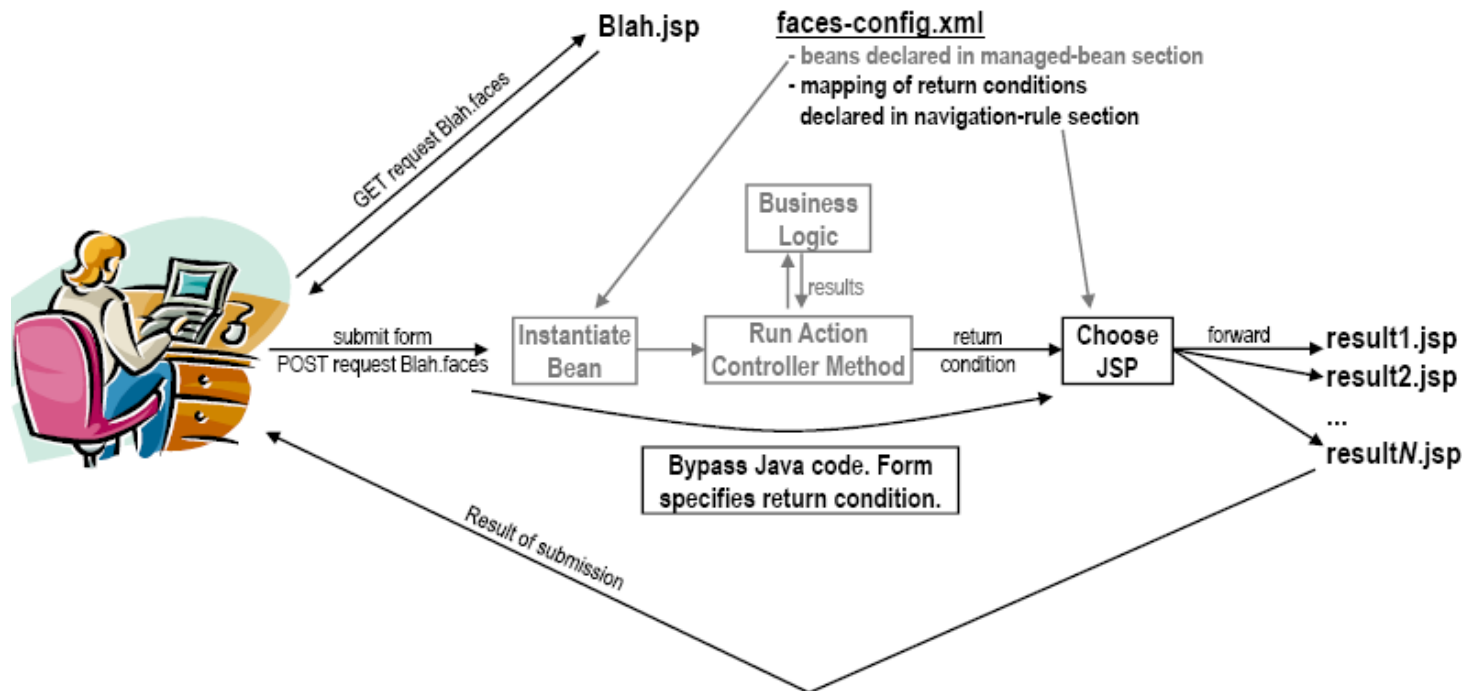
- ▶ odgovarajućim izborom stringova, moguće je skupiti višestruka pravila navigacije na jedno mjesto. Ako želimo da se akcija *logout* nalazi u više različitih JSF stranica i da iz svake poziva stranicu *logout.jsp*, tada definišemo sljedeće pravilo navigacije:

```
<navigation-rule>  
<navigation-case>  
<from-outcome>logout</from-outcome>  
<to-view-id>/logout.jsp</to-view-id>  
</navigation-case>  
</navigation-rule>
```

- ▶ ovo pravilo se odnosi na sve stranice zato što nijedan `from-view-id` element nije definisan

# Navigacija

▶ statička navigacija



# Navigacija

- ▶ statička navigacija – tipičan tok
- ▶ prikaz forme
  - f:view i h:form
- ▶ submit forme
  - originalni URL i ACTION URL su identični
- ▶ action atribut sadrži string koji odgovara from–outcome u navigacionim pravilima definisanim u faces–config.xml datoteci
- ▶ prikaz rezultujuće stranice
  - ova stranica je specificirana pomoću to–view–id u navigacionim pravilima definisanim u faces–config.xml datoteci

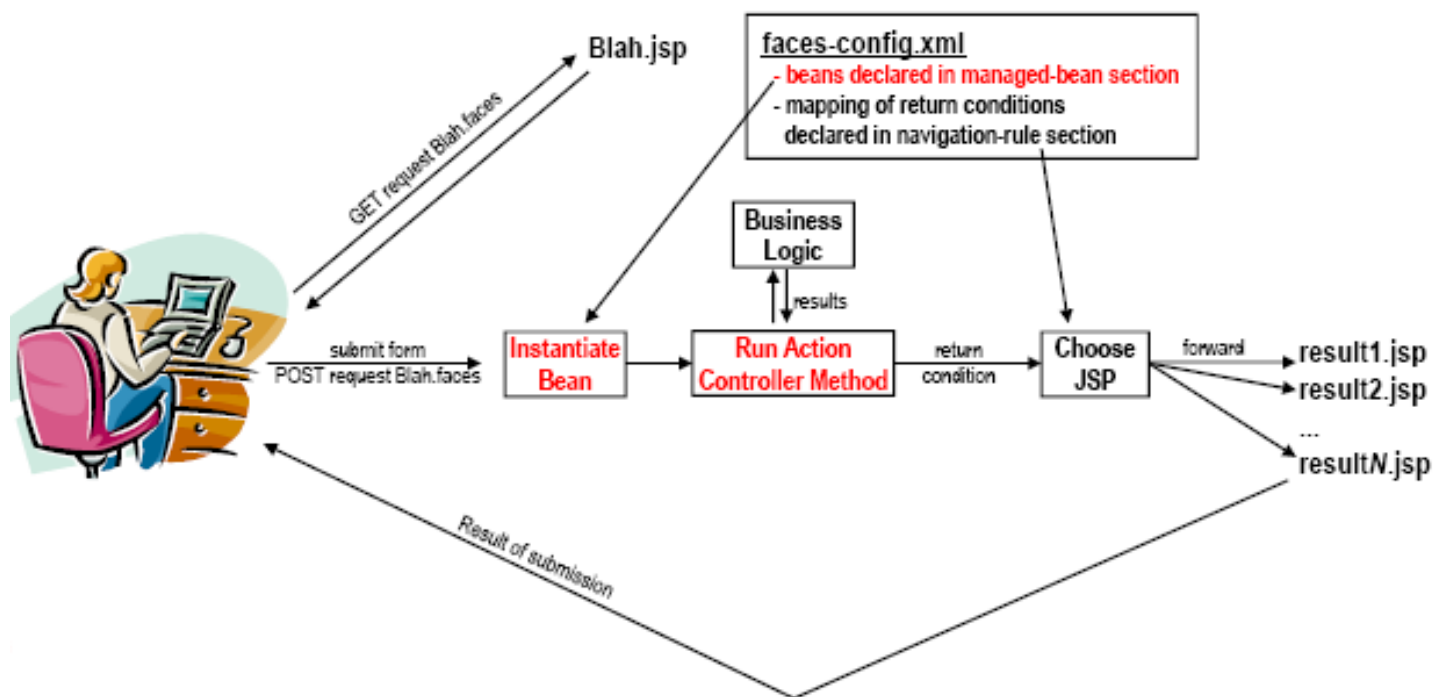


# Navigacija

- ▶ statička navigacija – kreiranje JSF stranica
- ▶ kreirati izvorišnu JSF stranu
- ▶ kreirati ulaznu formu i definisati return uslov (string koji odgovara from–outcome u navigacionim pravilima definisanim u faces–config.xml datoteci)
- ▶ editovati faces–config.xml
  - specificirati navigaciono pravilo
- ▶ kreirati odredišnu JSF stranu
  - prikazati podatke, npr. pomoću h:outputText
- ▶ spriječiti direktan pristup JSP stanama
  - korišćenjem filtera koji vrši redirekciju sa npr. a.jsp u a.jsf

# Navigacija

## ► dinamička navigacija



# Navigacija

- ▶ dinamička navigacija – tipičan tok
- ▶ prikaz forme
  - f:view i h:form
- ▶ submit forme
  - originalni URL i ACTION URL su identični
- ▶ instanciranje bean-a
  - iz managed-bean sekcije faces-config.xml konfiguracione datoteke
- ▶ poziv action metode
  - specificirana u action atributu h:commandButton komponente
- ▶ action metoda vraća uslov
  - string koji odgovara from-outcome u navigacionim pravilima definisanim u faces-config.xml datoteci
- ▶ prikaz rezultujuće stranice
  - ova stranica je specificirana pomoću to-view-id u navigacionim pravilima definisanim u faces-config.xml datoteci

# Navigacija

- ▶ dinamička navigacija – kreiranje JSF stranica
- ▶ kreiranje managed bean-a
  - kreiranje action metode
- ▶ kreirati izvorišnu JSF stranu
- ▶ kreirati ulaznu formu i referencirati action metodu odgovarajućeg managed bean-a
- ▶ editovati faces-config.xml
  - deklarirati managed bean
  - specificirati navigaciono pravilo
- ▶ kreirati odredišnu JSF stranu
  - prikazati podatke, npr. pomoću h:outputText
- ▶ spriječiti direktan pristup JSP stanama
  - korišćenjem filtera koji vrši redirekciju sa npr. a.jsp u a.jsf

# Navigacija – wildcards

- ▶ \* u from-view-id odnosi se na bilo koju početnu stranicu
  - koristi se kada različite izvorišne stranice vode na istu odredišnu
- ▶ **primjer**

```
<navigation-rule>
<from-view-id>*</from-view-id>
<navigation-case>
  <from-outcome>logout</from-outcome>
  <to-view-id>logout.jsp</to-view-id>
</navigation-case>
</navigation-rule>
```
- ▶ **bez \*** ovo navigaciono pravilo bi bilo mnogo komplikovanije

# Navigacija – wildcards

- ▶ ako se ne navede from-outcome – to znači da će svi rezultujući uslovi biti okidači (match)
  - osim null – označava da se forma ponovo prikaže

- ▶ **primjer**

```
<navigation-rule>  
<from-view-id>/some-page.jsp</from-view-id>  
  <navigation-case>  
    <from-outcome>uslov1</from-outcome>  
    <to-view-id>result1.jsp</to-view-id>  
  </navigation-case>  
  <navigation-case>  
    <to-view-id>default.jsp</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

- ▶ navođenjem svih drugih uslova ovo navigaciono pravilo bi bilo mnogo komplikovanije

# Navigacija – from-action

- ▶ određuje metodu sa koje se dolazi
- ▶ primjer, dva dugmeta koja pozivaju različite metode, pri čemu obe vraćaju “error” – ako se želi korisiti “error” string koji će voditi na različite odredišne strane, neophodno je koristiti from-action

- ▶ **primjer**

```
<navigation-rule>
<from-view-id>/somepage.jsp</from-view-id>
<navigation-case>
  <from-action>#{beanName.method1}</from-action>
  <from-outcome>error</from-outcome>
  <to-view-id>error1.jsp</to-view-id>
</navigation-case>
<navigation-case>
  <from-action>#{beanName.method2}</from-action>
  <from-outcome>error</from-outcome>
  <to-view-id>error2.jsp</to-view-id>
</navigation-case>
</navigation-rule>
```

- ▶ rijetko se koristi
- ▶ jednostavno se izbjegava – različit from-outcome za različite odredišne strane

# Navigacija

## ▶ pristup request i response objektima

```
ExternalContext ctxt =  
FacesContext.getCurrentInstance().getExternalContext();  
HttpServletRequest req =  
(HttpServletRequest) ctxt.getRequest();  
HttpServletResponse res =  
(HttpServletResponse) ctxt.getResponse();
```

## ▶ korisno za mnoge request property-je:

- eksplicitna manipulacija sesijama – npr. mijenjanje perioda neaktivnosti ili invalidacija sesije
- eksplicitna manipulacija cookie-ima
- čitanje request header-a
- određivanje imena hosta sa kojeg dolazi zahtjev
- nije potreban pristup request objektu radi populacije bean-a – automatski

## ▶ korisno za nekoliko response property-ja:

- postavljanje statusnih kodova
- postavljanje response header-a
- postavljanje cookie-ja



# Navigacija

- ▶ preklapanje managed-bean i navigation-rule tagova
  - dozvoljeno je
  - može biti korisno ako je u pitanju mali broj managed bean-ova i navigacionih pravila
  - uobičajeno:
    - managed-bean tagovi, pa ispod njih
    - navigation-rule tagovi
  - ispravno je oboje – izbor na programeru

# Managed beans

- ▶ Java beans
- ▶ Java klase koje poštuju sljedeće konvencije:
  - imaju default–ni (zero–argument) konstruktor
  - nemaju public polja
  - imaju setere i getere – setXxx i getXxx metode za pristup i promjenu vrijednosti polja
- ▶ često se, u JSF terminologiji, nazivaju pozadinskim bean–ovima – “backend beans”

# Managed beans

- ▶ Java beans – public polja ne koristiti !!!

umjesto

```
public double speed;
```

koristiti

```
private double speed;
```

```
public double getSpeed() {  
    return speed;  
}
```

```
public void setSpeed(double newSpeed) {  
    speed = newSpeed;  
}
```

- ▶ moguće je definisati ograničenja vrijednosti
- ▶ moguće je promijeniti internu reprezentaciju, bez mijenjanja interfejsa (potpisa metoda)
- ▶ moguće je proizvesti bočne efekte

# Managed beans

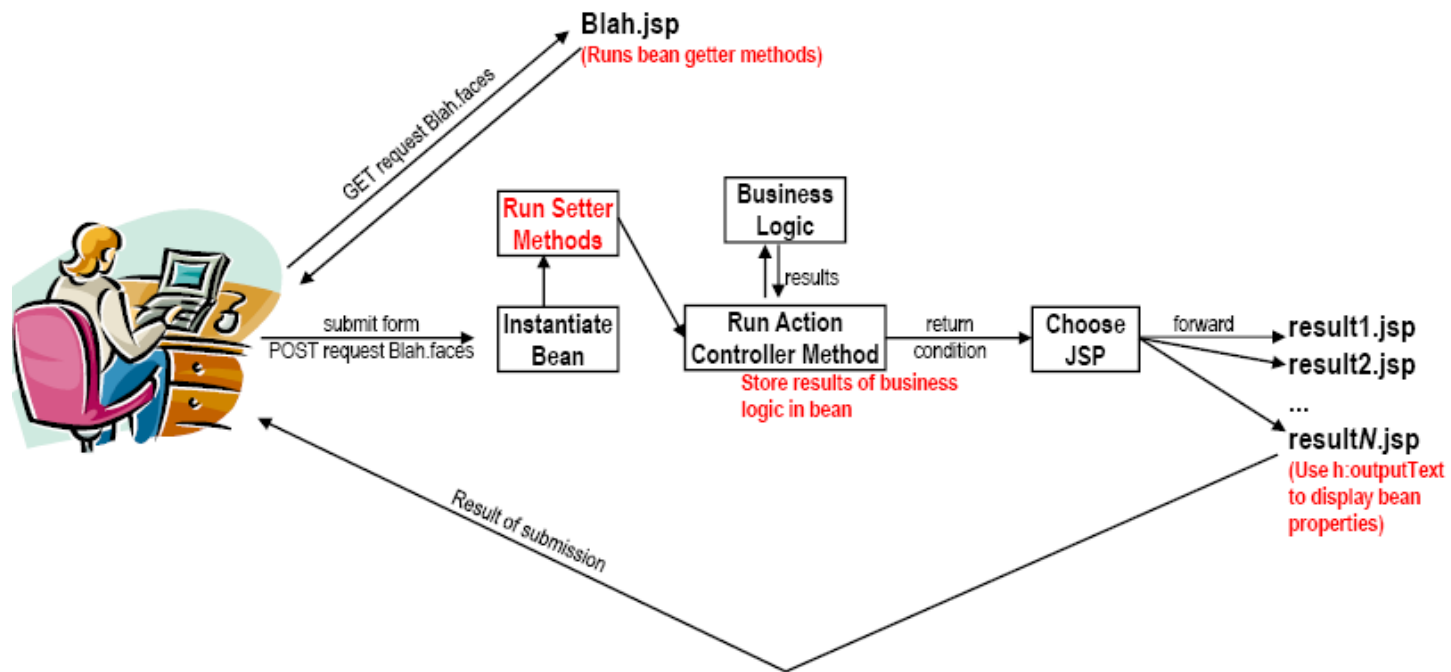
- ▶ Java beans – trebali bi biti serijalizabilni

```
public class TestBean implements Serializable
```

- ▶ neki serveri podržavaju distribuirane Web aplikacije
  - pomoću load balancing mehanizma različiti zahtjevi se mogu poslati različitim serverima – sesije bi trebale raditi, bez obzira koji server obrađuje zahtjev
- ▶ neki serveri podržavaju perzistentne sesije
  - podaci o sesiji se pamte na disku i ponovo se učitavaju u slučaju da je server restartovan – ovo treba da radi dok god je web čitač korisnika aktivan
  - Tomcat 5+ podržava perzistentne sesije

# Managed beans

- managed beans – control flow



# Managed beans

- ▶ managed beans – kontrola toka
- ▶ prikaz forme
  - f:view i h:form
- ▶ submit forme
  - originalni URL i ACTION URL su identični
- ▶ instanciranje bean-a
  - iz managed-bean sekcije faces-config.xml konfiguracione datoteke
  - setter metode navedene u h:inputText (...) se izvršavaju
- ▶ poziv action metode
  - specificirana u action atributu h:commandButton komponente
- ▶ action metoda vraća uslov
  - string koji odgovara from-outcome u navigacionim pravilima definisanim u faces-config.xml datoteci
- ▶ prikaz rezultujuće stranice
  - ova stranica je specificirana pomoću to-view-id u navigacionim pravilima definisanim u faces-config.xml datoteci
  - stranica koristi h:outputText da bi prikazala propertie-ije bean-a

# Managed beans

- ▶ managed beans – kreiranje JSF stranica
- ▶ kreiranje managed bean-a
  - kreiranje property-ja za svako polje forme
  - kreiranje setter i getter metoda za svaki property
  - kreiranje action metode
- ▶ kreirati izvorišnu JSF stranu
- ▶ kreirati ulaznu formu i referencirati action metodu odgovarajućeg managed bean-a
  - koristiti f:view, h:form, h:blah, i h:commandButton
  - koristiti action atribut h:commandButton
- ▶ editovati faces-config.xml
  - deklarirati managed bean
  - specificirati navigaciono pravilo
- ▶ kreirati odredišnu JSF stranu
  - prikazati podatke, npr. pomoću h:outputText
- ▶ spriječiti direktan pristup JSP stanama
  - korišćenjem filtera koji vrši redirekciju sa npr. a.jsp u a.jsf

# EL

- ▶ Expression Language
- ▶ dobre osobine:
  - kraća notacija za pristup property-ima bean-a
    - npr. `#{company.companyName}` – `companyName` property, getter `getPropertyName()` scoped variable (objekat smješten u request, session ili application scope) ili managed bean-a
    - `#{company.president.firstName}` – `firstName` property objekta `president`, koji je property scoped variable ili managed bean-a `Company`
  - jednostavan pristup elementima kolekcije
    - za referenciranje elementa niza, List-e ili Map-e koristi se `#{variable[indexOrKey]}`



# EL

## ► dobre osobine:

- jednostavan pristup parametrima zahtjeva, cookie-ima, i drugim podacima zahtjeva
- za pristup standardnim tipovima podataka zahtjeva, može se koristiti jedan od nekoliko predefinisanih implicitnih objekata
- mali, ali koristan skup jednostavnih operatora
  - za obradu objekata pomoću EL izraza, može se koristiti neki od nekoliko aritmetičkih, relacionih, logičkih, ili drugih operatora
- uslovni izlaz
  - za izbor između više izlaznih opcija, ne moraju se koristiti Java skripteti, već `{test ? option1 : option2}`.
- automatska konverzija tipova
  - EL uklanja potrebu za konverzijom tipova
- prazna vrijednost umjesto poruke o grešci
  - u većini slučajeva, greška ili `NullPointerExceptions` daju prazan string, a ne izuzetak

# EL

## ▶ JSF EL

- Može se koristiti samo u okviru atributa JSF tagova
- zahtjevaju taglib deklaraciju
- koriste se na serverima koji podržavaju JSP 1.2+ (Tomcat 4)
- koriste `{abc}`
- mogu da prikažu poslate podatke i izlazne vrijednosti
- pristupaju bean-ovima definisanim u okviru zahtjeva, sesije ili aplikacije ili managed bean-ovima

## ▶ JSP 2.0 EL

- Može se koristiti bilo gdje u okviru JSP stranice
- ne zahtjevaju taglib deklaraciju
- koriste se na serverima koji podržavaju JSP 2.0 (Tomcat 5)
- koriste `${abc}`
- mogu da prikažu izlazne vrijednosti
- pristupaju bean-ovima definisanim u okviru zahtjeva, sesije ili aplikacije

# EL

- ▶ čitanje property-ja bean-a
- ▶ `#{varName.propertyName}`
  - pretražuje se `HttpServletRequest`, `HttpSession`, `ServletContext` i managed bean-ovi, u ovom poretku, i vraća vrijednost property-ja
  - mora se koristiti u atributu JSF taga
- ▶ ekvivalentne forme

```
<h:outputText value="#{customer.firstName}"/>
```

- radi u svim JSF verzijama – **scoped varijable** i managed bean-ovi

```
${customer.firstName}
```

- radi samo u JSP 2.0 i kasnijim verzijama – **samo scoped varijable**

```
<%@ page import="net.etfbl.NameBean" %>
```

```
<% NameBean person =
```

```
(NameBean)pageContext.findAttribute("customer"); %>
```

```
<%= person.getFirstName() %>
```

- **pre-EL verzija**

# EL

- ▶ čitanje property-ja bean-a

```
# { varName.property1.property2 }
```

- ▶ prvo se izvršava pretraga za odgovarajuću definiciju beana po imenu varName
- ▶ nakon toga se pristupa property-iju property1 (poziva se metoda `getProperty1()`)
- ▶ nakon toga se pristupa property-iju property2 dobijenog rezultata – poziva se metoda `getProperty2()` objekta koji je dobijen sa `getProperty1()`

# EL

- ▶ tri značenja #
- ▶ označavanje izlaznih vrijednosti:
  - `#{varName.propertyName}`
    - prikazuje vrijednost property-ija date promenljive
  - `<h:outputText value="#{employee.address}"/>`
    - bilo kada da se pristupa, podrazumijeva izlazni tekst
  - `<h:inputText value="#{employee.address}"/>`
    - kada se forma inicijalno prikazuje, prikazuje se predefinisana vrijednost
- ▶ označavanje submit-ovane vrijednosti
  - `<h:inputText value="#{employee.address}"/>`
    - kada se forma šalje, definiše gdje se smješta vrijednost
- ▶ dizajn poziva metoda kod slanja
  - `<h:commandButton value="Button Label" action="#{employee.processEmployee}"/>`
    - kada se forma šalje, definiše se određena akcija

# EL

- ▶ pristup kolekcijama
- ▶ radi za
  - nizove – ekvivalentno sa  
`theArray[index]`
  - liste – ekvivalentno sa  
`theList.get(index)` ili `theList.set(index, submitted-val)`
  - mape – ekvivalentno sa  
`theMap.get(key)` ili `theMap.put(key, submitted-val)`
- ▶ ekvivalentne forme (za HashMap-e)
  - `#{name.property}`
  - `#{name["property"]}`

# EL

- ▶ predefinisane varijable
- ▶ facesContext – FacesContext objekat
  - `#{facesContext.externalContext.session.id}`
- ▶ param i paramValues – request parametri
  - `#{param.custID}`
- ▶ header i headerValues – request header-i
  - `#{header.Accept}` or `#{header["Accept"]}`
  - `#{header["Accept-Encoding"]}`
- ▶ cookie – Cookie objekat
  - `#{cookie.userCookie.value}` ili `#{cookie["userCookie"].value}`
- ▶ initParam – Context initialization parametar
- ▶ requestScope, sessionScope, applicationScope
  
- ▶ potencijalni problem
  - implicitni objekti (predefinisane varijable) obično lošije rade sa MVC modelom

# EL

- ▶ operatori
- ▶ aritmetički
  - + - \* / div % mod
- ▶ relacioni
  - == eq != ne < lt > gt <= le >= ge
- ▶ logički
  - && and || or ! Not
- ▶ empty
  - empty
  - vraća true za null, prazan string, prazan niz, praznu listu, praznu mapu
- ▶ napomena
  - koristiti ograničeno kako bi MVC model bio očuvan



# EL

- ▶ `${ test ? expression1 : expression2 }`
  - izračunava test i vraća ili `expression1` ili `expression2`
- ▶ problemi
  - podrazumijeva stavljanje dijela business logike u JSP stranicu
  - trebalo bi ga koristiti samo za prezentacionu logiku
- ▶ napomena
  - koristiti ograničeno kako bi MVC model bio očuvan

# properties datoteke

- ▶ .properties datoteka
  - sadrži parove ključ – vrijednost
  - mora biti smještena u WEB-INF/classes
- ▶ učitavanje datoteke – pomoću f:loadBundle
  - basename atribut – ime datoteke, bez ekstenzije .properties
  - var – vraća scoped varijablu – Map-u
    - relativno u odnosu na WEB-INF/classes, .properties se podrazumijeva
    - primjer 1, WEB-INF/classes/messages.properties  
`<f:loadBundle basename="messages" var="msgs"/>`
    - primjer 2., WEB-INF/classes/package1/test.properties  
`<f:loadBundle basename="package1.test" var="msgs"/>`
- ▶ ispis poruka – korišćenjem EL  
`{msgs.keyName}`

# properties datoteke

- ▶ parametrizovani stringovi
- ▶ kreirati .properties datoteku unutar WEB-INF/classes
  - vrijednosti sadrže {0}, {1}, {2}, itd.
  - npr. poruka=Unesena vrijednost se nalazi u intervalu između {0} i {1} !!!
- ▶ učitavanje datoteke – pomoću f:loadBundle
  - basename atribut – ime datoteke, bez ekstenzije .properties
  - var – vraća scoped varijablu – Map-u
- ▶ ispis poruka – korišćenjem h:outputFormat
  - vrijednost vraća osnovnu poruku
  - ugnježdeni f:param vraćaju vrijednosti koje se mijenjaju

```
<h:outputFormat value="#{msgs.poruka}">  
  <f:param value="vrijednost za ulaz 0"/>  
  <f:param value="vrijednost za ulaz 1"/>  
</h:outputFormat>
```

# properties datoteke

- ▶ internacionalizacija
- ▶ kreirati višestruke .properties datoteke
  - messages\_en.properties, messages\_sr.properties, messages\_hr.properties
- ▶ proslijediti locale argument u f:view, putem locale atributa

```
<f:view
  locale="#{facesContext.externalContext.request.locale}">
```

  - pročitati locale iz podešavanja Web čitača
- ili
  - postaviti izabrani locale

```
locale="#{settings.selectedLocale}"
```
- ▶ učitavanje datoteke – pomoću f:loadBundle
  - basename atribut – ime datoteke, bez ekstenzije .properties
    - verzija koja odgovara izabranom locale-u će biti automatski korištena
  - var – vraća scoped varijablu – Map-u
- ▶ ispis poruka – korišćenjem h:outputForma

# Event handling

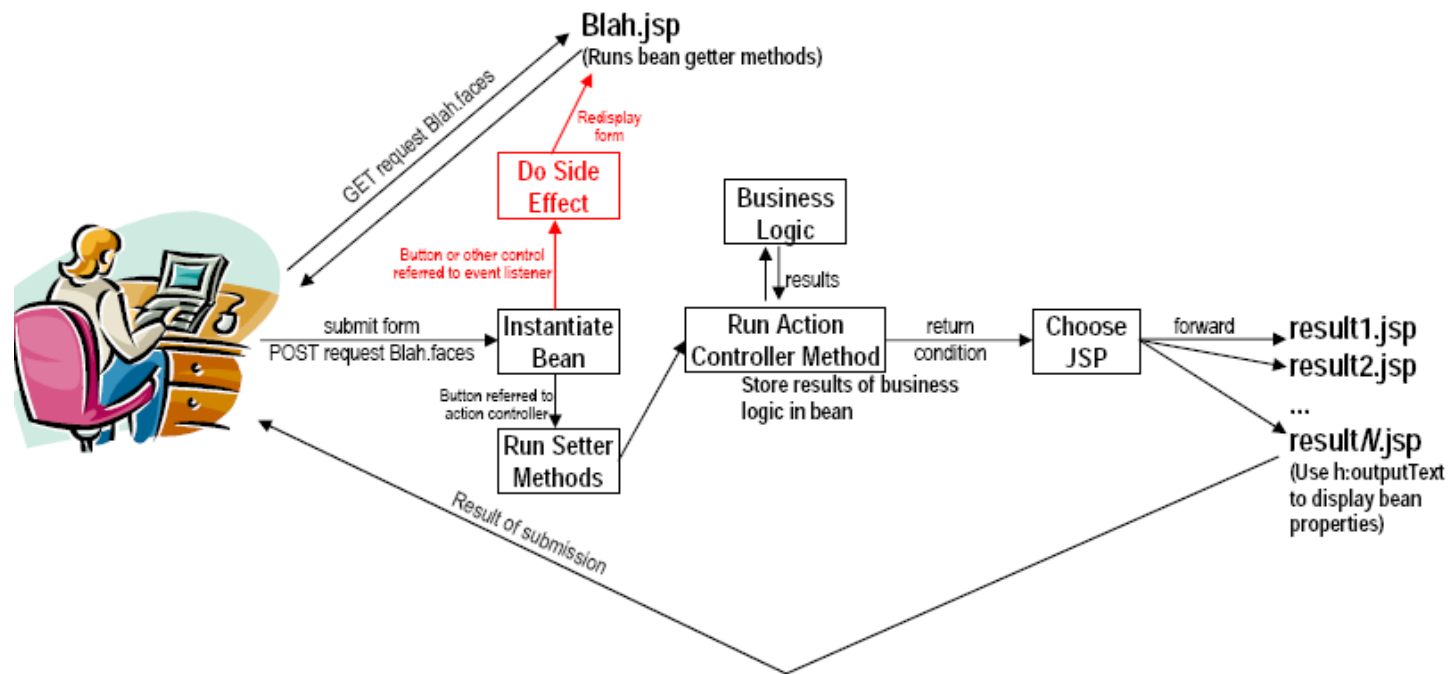
- ▶ dvije vrste događaja koje generiše korisnik:
  - događaji koji startuju back-end procesiranje
  - događaji koji utiču samo na format korisničkog interfejsa
- ▶ JSF kod koji upravlja ovim akcijama dijeli na:
  - action kontroler-e
  - event listener-e
- ▶ action kontroleri obrađuju submit forme
  - okidaju se poslije popunjavanja bean-ova
  - okidaju se poslije validacione logike
  - vraćaju stringove koji utiču na dalju navigaciju
- ▶ event listener-i obrađuju UI događaje:
  - obično se okidaju prije popunjavanja bean-ova
  - obično izbjegavaju validacionu logiku
  - nikad direktno ne utiču na navigaciju

# Event handling

- ▶ event handler-i su jedna prepoznatljiva karakteristika JSF-a
- ▶ Ajax je često mnogo efikasniji
  - u mnogim situacijama samo mali broj elemenata se mijenja – u tom slučaju ajax će obezbijediti veći stepen interakcije

# Event handling

- ▶ event handling – control flow



# Event handling

- ▶ tipovi event listener-a

- ▶ **ActionListener**

- poziva se pomoću submit dugmeta, slika, i linkova sa odgovarajućim JavaScript kodom

```
<h:commandButton value="..." .../>
```

```
<h:commandButton image="..." .../>
```

```
<h:commandLink .../>
```

- automatski submit-uju formu

- ▶ **ValueChangeListener**

- poziva se pomoću combo box-ova, checkbox-ova, radio dugmadi, tekst polja, ...

```
<h:selectOneMenu .../>
```

```
<h:selectBooleanCheckbox.../>
```

```
<h:selectOneRadio .../>
```

```
<h:inputText .../>
```

- ne submit-uju formu automatski



# ActionListener

- ▶ korišćenje ActionListener-a
- ▶ neka dugmad samo submit-uju formu i pokreću backend procese
  - koristi se `<h:commandButton action="..." ...>`
- ▶ druga dugmad utiče samo na UI
  - koristi se `<h:commandButton ActionListener="..." .../>`
  - obično se želi da se ovaj proces izvršava prije nego što se popune bean-ovi i posebno prije validacije
  - zato se koristi atribut "immediate" da bi se naznačilo izvršavanje prije navedenih operacija
  - `<h:commandButton ActionListener="..." immediate="true" .../>`

# ActionListener

- ▶ listeneri su obično u form bean klasi
  - mogu biti i u odvojenoj klasi ako se koristi FacesContext da bi se dobio request ili session objekat i potražio form bean eksplicitno
- ▶ ActionEvent kao argument metode
  - nema vraćanja rezultata (nije String kao kod action kontrolera)
  - ActionEvent je u javax.faces.event
  - ActionEvent ima getComponent metod koji definiše UIComponent reference
    - iz UIComponent, može se dobiti component ID, renderer, i druge informacije niskog nivoa

```
public void someMethod(ActionEvent event) {  
    doSomeSideEffects();  
}
```

# ActionListener

- ▶ ActionListener – kreiranje JSF stranica
- ▶ kreiranje managed bean-a
  - kreiranje property-ja za svako polje forme
  - kreiranje setter i getter metoda za svaki property
  - kreiranje action metode i **event listener** metode
- ▶ kreirati izvorišnu JSF stranu
- ▶ kreirati ulaznu formu i referencirati action metodu i **event listener** odgovarajućeg managed bean-a
  - koristiti f:view, h:form, h:blah, i h:commandButton
  - koristiti action atribut h:commandButton
- ▶ editovati faces-config.xml
  - deklarirati managed bean
  - specificirati navigaciono pravilo
- ▶ kreirati odredišnu JSF stranu
  - prikazati podatke, npr. pomoću h:outputText
- ▶ spriječiti direktan pristup JSP stranama
  - korišćenjem filtera koji vrši redirekciju sa npr. a.jsp u a.jsf

# ValueChangeListener

## ▶ ValueChangeListener

- nije vezan za dugme
- veže se za combobox, listbox, radio button, checkbox, textfield, itd.
- forme se ne submit-uju automatski
- potrebno je dodati JavaScript za automatski submit `onclick="submit()"` ili `onchange="submit()"`
- problem nekompatibilnosti Web čitača
  - Firefox, Netscape, i Opera okidaju onchange događaje kada se promijeni selekcija combobox-a, kad je radio dugme selektovano ili kad je checkbox selektovan, tj. deselektovan
  - Internet Explorer okida događaj kada se selekcija promijeni, ali tek kada druga GUI kontrola dobije fokus

# ValueChangeListener

- ▶ obično su u form bean klasi
  - mogu biti i u odvojenoj klasi ako se koristi FacesContext da bi se dobio request ili session objekat i potražio form bean eksplicitno
- ▶ uzimaju ValueChangeEvent kao argument
  - korisne ValueChangeEvent metode
    - getComponent
    - getOldValue – prethodna vrijednost GUI elementa
    - getNewValue – trenutna vrijednost GUI elementa
      - potrebna jer bean još uvijek, vjerovatno, nije popunjen

```
public void someMethod(ValueChangeEvent event) {  
    boolean flag =  
    ((Boolean) event.getNewValue()).booleanValue();  
    takeActionBasedOn(flag);  
}
```

# ValueChangeListener

- ▶ ActionListener – kreiranje JSF stranica
- ▶ kreiranje managed bean-a
  - kreiranje property-ja za svako polje forme
  - kreiranje setter i getter metoda za svaki property
  - kreiranje action metode i **event listener** metode
- ▶ kreirati izvorišnu JSF stranu
- ▶ kreirati ulaznu formu i referencirati action metodu i **event listener** odgovarajućeg managed bean-a
  - koristiti f:view, h:form, h:blah, i h:commandButton
  - koristiti action atribut h:commandButton
- ▶ editovati faces-config.xml
  - deklarirati managed bean
  - specificirati navigaciono pravilo
- ▶ kreirati odredišnu JSF stranu
  - prikazati podatke, npr. pomoću h:outputText
- ▶ spriječiti direktan pristup JSP stranama
  - korišćenjem filtera koji vrši redirekciju sa npr. a.jsp u a.jsf

# “h” biblioteka

- ▶ **h:form**
  - ne specificira se ACTION atribut
  - mora se koristiti POST metoda
- ▶ **h:inputText**
  - NAME se generiše automatski
  - VALUE je u formi #{beanName.propertyName}
- ▶ **h:inputSecret**
  - NAME se generiše automatski
  - VALUE se primjenjuje samo na izlaz, ne na ulaz
- ▶ **h:commandButton**
  - ACTION odgovara atributu ACTION taga forme
- ▶ **h:outputText**
  - prikazuju se property-iji bean-a

# “h” biblioteka

- ▶ dijeljeni atributi
  - ove attribute je moguće koristiti u skoro svim h:blah elementima
- ▶ najviše korišćeni dijeljeni atributi
  - id – identifikator; koristi se da bi se dodijelilo ime elementu, tako da može da mu se pristupi kasnije (npr., za validaciju)
  - onclick, ondblclick, ... JavaScript obrada događaja
    - obratiti pažnju na velika i mala slova, onClick nije ispravno
  - styleClass – CSS ime stila
- ▶ očekivani atributi
  - ovi atributi su oni koji odgovaraju određenom HTML elementu, npr. maxlength za tekst polje



# h:form

- ▶ odgovarajući HTML element
  - `<FORM ...>`
- ▶ mogućnosti
  - ne specificira se akcija
- ▶ atribut
  - `enctype` – tip kodiranja
    - `application/x-www-form-urlencoded` (default)
    - `multipart/form-data`
    - `text/plain`
  - `target` – frame dio u kojem se prikazuje rezultat
  - `onsubmit`. JavaScript kod koji se izvršava prije slanja forme
    - može da se izvrši validacija polja na klijentskoj strani

# h:inputText

- ▶ odgovarajući HTML element
  - `<INPUT TYPE="TEXT" ...>`
- ▶ atributi
  - value – odgovarajuća vrijednost bean-a
    - `value="#{beanName.propertyName}"`
    - koristi se i za inicijalnu vrijednost i pri slanju serverskoj komponenti
  - valueChangeListener – definiše koji se metod izvršava kada se šalje forma i kada je prethodna vrijednost promijenjena
  - onchange – JavaScript koji se izvršava pri promjeni vrijednosti
  - immediate – koristi se da označi da listener zaobilazi validaciju
  - required – definiše da li korisnik mora da unese vrijednost
    - ako je ne unese, forma se ponovo prikazuje i prikazuje se poruka o grešci
  - validator – metod koji izvršava validaciju

# h:inputSecret

- ▶ odgovarajući HTML element
  - `<INPUT TYPE="PASSWORD" ...>`
- ▶ atributi
  - isti kao za h:inputText, s tim što se value atribut koristi samo za submit

# h:commandButton

- ▶ odgovarajući HTML element
  - `<INPUT TYPE= "SUBMIT" ...>`
- ▶ atributi
  - value – natpis na dugmetu – često je statički string, ali se može i dinamički mijenjati
  - action – metod action controller-a koji se izvršava kada se šalje forma – izvršava se kada svi listener-i završe svoj posao
  - ActionListener – metod listener-a koji se izvršava kada se šalje forma – izvršava se prije action controller-a

# h:commandLink

- ▶ odgovarajući HTML element
  - `<A HREF="...">` sa povezanim JavaScript kodom koji šalje formu kada se link aktivira
  - JavaScript se ubacuje automatski pomoću JSF
- ▶ atributi
  - `value` – tekst linka – često je statički string, ali se može i dinamički mijenjati
  - `action` – metoda action controller-a koja se izvršava kada se šalje forma – izvršava se kada svi listener-i završe svoj posao
  - `actionListener` – metod listener-a koji se izvršava kada se šalje forma – izvršava se prije action controller-a

# Elementi sa valueChangeListener atributom

- ▶ Checkbox–ovi
  - h:selectBooleanCheckbox
- ▶ Combobox–ovi
  - h:selectOneMenu
  - h:selectManyMenu
- ▶ List box–ovi
  - h:selectOneListbox
  - h:selectManyListbox
- ▶ Radio Button
  - h:selectOneRadio
- ▶ Textfields
  - h:inputText

# Combobox-ovi i List box-ovi

- ▶ navođenjem opcije po opcije

```
<h:selectOneMenu ...>  
<f:selectItem itemValue="..." itemLabel="..." />  
<f:selectItem itemValue="..." itemLabel="..." />  
<f:selectItem itemValue="..." itemLabel="..." />  
</h:selectOneMenu>
```

- ▶ navođenjem svih opcija u jednom redu

```
<h:selectOneMenu ...>  
  <f:selectItems value="..." />  
</h:selectOneMenu>
```

- ▶ value atribut h:selectOneMenu određuje selected vrijednost

# Combobox-ovi i List box-ovi

- ▶ vrijednost za `f:selectItems` mora biti List ili niz elemenata tipa `java.faces.model.SelectItem`
- ▶ `SelectItem` ima dva osnovna konstruktora
  - jedan samo specificira String
    - vrijednost koja se prikazuje i vrijednost koja se šalje setter metodi bean-a su isti
  - drugi specificira Java objekat i String
    - string se prikazuje, a Java objekat se šalje kao vrijednost setter metodi bean-a (mapiranje se izvršava automatski pomoću JSF-a)

```
SelectItem[] listBoxItems =  
{  
    new SelectItem(realValue1, "Choice 1"),  
    new SelectItem(realValue2, "Choice 2"),  
    ...  
}
```



# h:outputText i h:outputFormat

## ▶ h:outputText

- prikazuje se bean property ili element kolekcije
  - vrijednosti iz properties fajla se upisuju u kolekciju pomoću f:loadBundle
- Atributi
  - styleClass ili style: rezultat je SPAN element
  - escape: vrijednost false označava da se ne koristi filtera za < i >

## ▶ h:outputFormat

- prikazuje poruke o grešci sa parametrima
- parametri se prosljeđuju sa f:param

# Validacija

- ▶ dva zadatka koja skoro svaka Web aplikacija mora da izvrši:
  - provjera da su sva polja forme popunjena i da su unesene vrijednosti u odgovarajućem formatu
  - ponovno prikazivanje forme u slučaju kada nisu unesene sve vrijednosti ili kada je unos pogrešan, zajedno sa porukama koje ukazuju na greške i sa vrijednostima koje imaju pravilan format

# Validacija

- ▶ ručna validacija
  - koriste se string property-iji za bean
  - sprovodi se validacija u setter metodama i/ili action controller-ima
  - vraća se null da bi se ponovo prikazala forma
  - kreiraju se odgovarajuće poruke o greškama
- ▶ implicitna automatska validacija
  - koriste se int, double, ... bean property-iji, ili se dodaje required
  - sistem ponovo prikazuje formu, ako postoji greška prilikom konverzije
  - koristi se h:message da se prikaže poruka za određeno polje
- ▶ eksplicitna validacija
  - koriste se f:convertNumber, f:convertDateTime, f:validateLength, f:validateDoubleRange, ili f:validateLongRange
  - sistem ponovo prikazuje forma u slučaju greške; opet h:message
- ▶ kreiranje sopstvenih metoda validacija

# Ručna validacija

- ▶ setter metode konvertuju iz stringova
  - koriste se try/catch blokovi
  - koristi se logika specifična za aplikaciju
- ▶ Action controller provjerava vrijednosti
  - ako su vrijednosti u redu, vraća se uobičajeni izlaz
  - ako vrijednosti nedostaju ili su nelegalne, smješta se poruka o grešci u bean i vraća se null
- ▶ ulazna forma
  - prikazuje poruke o grešci
    - poruke su prazni stringovi po defaultu
    - u okviru h:outputText, treba koristiti escape="false" ako poruka sadrži HTML tagove

# Ručna validacija

- ▶ alternativa kreiranju vlastitih poruka o grešci
  - kreirati FacesMessage
  - pohraniti je u globalnu listu poruka korišćenjem
    - `facesContext.addMessage`
  - vratiti null za ponovni prikaz početne forme
  - ispisati poruke o grešci pomoću
    - `h:messages` ili `h:message`
- ▶ prednosti:
  - nije potrebno čuvati poruke “na svoj način”
  - uklapa se u standardnu JSF validaciju
- ▶ nedostaci
  - manja kontrola nad formatom poruka o grešci

# Implicitna automatska validacija

- ▶ definišu se bean property-iji na standardne tipove podataka
  - int, long, double, boolean, char, ...
- ▶ sistem pokušava da izvrši automatsku konverziju, kao kod `jsp:setProperty`
  - metode `Integer.parseInt`, `Double.parseDouble`...
- ▶ ako postoji greška, forma se ponovo prikazuje
  - i prikazuje se poruka o grešci
- ▶ može se dodati atribut `required` za svaki ulazni element, da bi se specificiralo da prazne vrijednosti predstavljaju grešku
  - koristi se `h:message` da bi se prikazale poruke o greškama
  - `h:message` vraća prazan string ako nema poruke
  - `h:message` prihvata `styleClass` za CSS ime stila
- ▶ dodaje se atribut `immediate` da bi se preskočila validacija
  - na primjer, za `h:commandButton` sa `logout` ili `cancel` operacijama

# EksPLICITNA automatska validacija

- ▶ definisati bean property-ije na proste tipove podataka
  - int, long, double, boolean, char, ...
- ▶ dodaje se f:validateBlah ili f:convertBlah elementima
- ▶ sistem provjerava da li polja odgovaraju pravilima
  - f:validateBlah atributi dozvoljavaju da vrši kontrola formata
- ▶ ako postoji greška prilikom validacije, forma se ponovo prikazuje
  - i smješta se poruka o grešci
- ▶ ostali pristupi su i dalje mogući
  - i dalje se može dodati atribut required za bilo koji ulazni element
  - i dalje se sa h:message prikazuju poruke
    - h:message vraća prazan string ako nema poruke
  - i dalje se dodaje atribut immediate da bi se izbjegla validacija

# Konverzija vs. validacija

- ▶ i `f:convertBlah` i `f:validateBlah` provjeravaju format i prikazuju ponovo formu u slučaju greške
  - `f:convertBlah` mijenja i format u kome se polje prikazuje
  - `f:validateBlah` ima smisla samo sa `h:inputText`
  - `f:convertBlah` ima smisla koristiti i sa `h:inputText` i sa `h:outputText`
- ▶ **primjer**

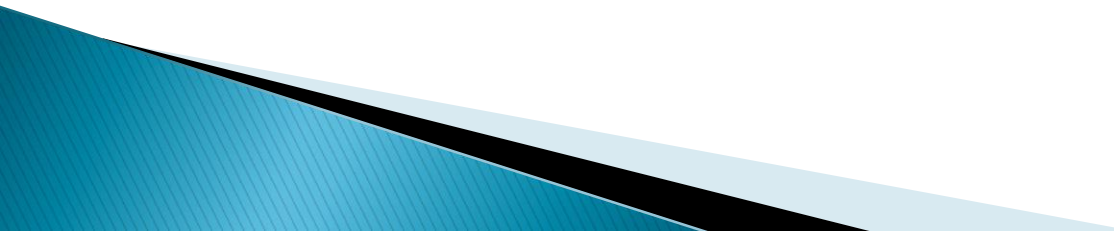
```
<h:inputText value="#{itemBean.price}">
<f:convertNumber maxFractionDigits="2"/>
</h:inputText>
```
- ▶ **prikazuje 0.75, a ne 0.749**



# Atributi konvertora i validatora

- ▶ f:convertNumber
  - currencyCode, currencySymbol
  - groupingUsed
  - integerOnly
  - locale
  - max(min)FractionDigits
  - max(min)IntegerDigits
  - pattern
  - type
    - number, currency, percentage
- ▶ f:convertDateTime
  - type
    - date, time, both
  - dateStyle, timeStyle
    - default, short, medium, long, full
  - pattern (ala SimpleDateFormat)
  - locale
  - timeZone

# Atributi konvertora i validatora

- ▶ f:validateLength
    - minimum
    - maximum
  - ▶ f:validateLongRange
    - minimum
    - maximum
  - ▶ f:validateDoubleRange
    - minimum
    - maximum
- 

# Korisnički konvertori

- ▶ implementiraju Converter interfejs
  - javax.faces.convert.Converter
  - getObject
    - uzima String; vraća Object (konvertuje unos korisnika)
  - getString
    - uzima Object; vraća String
- ▶ greške pri konverziji
  - baca se ConverterException sa FacesMessage
- ▶ registracija u faces-config.xml
  - converter, converter-id, converter-class
- ▶ koristi se converter atribut

```
<h:inputText  
value="#{...}" converter="someID"/>
```

# Korisnički validatori

- ▶ implementiraju Validator interfejs
- ▶ implementiranje validate metode
- ▶ greške pri konverziji
  - baca se ValidationException sa FacesMessage
- ▶ registracija u faces-config.xml
  - validator, validator-id, validator-class
- ▶ koristiti f:validator tag u JSF stranici

```
<h:inputText value="#{...}">
```

```
<f:validator validatorId="someID"/>
```

```
</h:inputText>
```



# Korisničke poruke o greškama

- ▶ kreira se i učitava globalna .properties datoteka

```
<application>
```

```
<message-bundle>...</message-bundle>
```

```
</application>
```

- ▶ koriste se eksplicitna imena property-ija u .properties datoteci

```
javax.faces.component.UIInput.CONVERSION
```

```
javax.faces.component.UIInput.REQUIRED
```

- ▶ **primjer**

```
javax.faces.component.UIInput.CONVERSION=
```

```
Pogresan format!
```

```
javax.faces.component.UIInput.REQUIRED=
```

```
Unesite vrijednost!!!
```

# Korisničke metode validacije

## ▶ JSP

- za ulaznu komponentu eksplicitno specificirati ime metode

```
<h:inputText id="someID"  
  validator="#{someBean.someMethod}"/>
```

- koristi se `h:message` kao i ranije

```
<h:message for="someID"/>
```

## ▶ Java

- pomoću `ValidatorException` sa `FacesMessage` ako validacija nije uspjela – ništa ne raditi ako je validacija uspjela

## ▶ argumenti metoda:

- `FacesContext`
  - kontekst
- `UIComponent`
  - komponenta nad kojom se vrši validacija
- `Object`
  - submit-ovana vrijednost (primitivni tipovi koriste wrapper-e)

# Tabele

## ▶ upotreba

- kada se na serverskoj strani kreira rezultat sa nepoznatim brojem elemenata

## ▶ alternative

- bez petlji

```
<h:outputText  
  value="#{bankCustomer.depositTable}"/>  
<mytags:showMytTable custID="..." month="..."  
  styleClasses="..."/>
```

- sa petljama

```
<% for(...) { ... %>
```

HTML kod

```
<% } %>
```

JSTL

# Tabele

## ▶ h:dataTable

- definicija za jednu vrstu – JSF je ponavlja onoliko puta koliko je potrebno
- value: kolekcija podataka (obično lista bean-ova) – dozvoljeni tipovi kolekcija:
  - Array
  - List (e.g., ArrayList, LinkedList)
  - ResultSet (moraju biti scroll-insensitive)
  - Result (wrapped ResultSet from JSTL)
  - DataModel (in javax.faces.model)
- var: vezan za svaki ulaz kolekcije
  - ovaj ulaz treba da bude nešto što JSF EL može da prikaže
    - Bean, array, List, Map
- drugi atributi
  - standardni TABLE atributi
    - border, bgcolor, width, cellpadding, cellspacing, frame, ...
  - style sheet
    - rowClasses, headerClass, footerClass



# Tabele

## ▶ h:column

- obično okružuju h:outputText elemente

```
<h:column>  
    <h:outputText value="#{rowVar.colData}"/>  
</h:column>
```

- mogu okruživati druge h:Xxxx elemente
  - h:inputText, ili bilo koje druge elemente

## ▶ regularni HTML sadržaj mora biti u okviru f:verbatim

```
<h:column>  
<f:verbatim>First Name: </f:verbatim>  
<h:outputText value="#{rowVar.firstName}"/>  
</h:column>
```


## ▶ header i footer tabele se specificira pomoću f:facet

# Tabele

- ▶ header i footer tabele
- ▶ header
  - koristi se f:facet sa name="header"
  - vrijednost može da bude f:verbatim ili h:outputText
  - i dalje je potreban h:outputText za ne-heading vrijednosti
- ▶ footer
  - koristi se f:facet sa name="footer"
  - vrijednost može da bude f:verbatim ili h:outputText
- ▶ primjer

```
<h:column>  
<f:facet name="header">  
<f:verbatim>...</f:verbatim>  
</f:facet>  
<h:outputText value="#{rowVar.colVal}"/>  
</h:column>
```

# Tabele

- ▶ CSS
  - ▶ eksplicitno formatiranje u JSP
    - zahtjeva f:verbatim prije i poslije svakog entry-ja
  - ▶ formatiranje ugrađeno u bean
    - nije dostupno Web dizajneru
    - nekonzistentnost sa style sheet-ovima
    - bean-ovi su dio modela, ne view-a
  - ▶ koristiti rowClasses, headerClass, footerClass
    - rowClasses: zarezima razdvojena lista CSS stilova – primjenjuje se na svaku vrstu do kraja liste, a onda se ponavlja
    - headerClass: CSS stil za header tabele
    - footerClass: CSS stil za footer
- 

# Tabele

- ▶ prikaz sadržaja iz baze podataka
- ▶ uzeti podatke iz result set-a i smjestiti ih u niz ili List-u
  - sporo
  - zahtjeva bean koji će reprezentovati podatke koji čine vrstu tabele
- ▶ moguće je koristiti ResultSet kao h:dataTable value
  - podržano je, prema specifikaciji
- ▶ moguće je koristiti i JSTL ResultSupport klasu za pretvaranje ResultSet-a u Result
  - samo jedna dodatna linija koda
  - robusno i pouzdano
  - konekciju je moguće zatvoriti ili vratiti u pool konekcija, prije nego što se počne koristiti Result

# Resursi

- ▶ JSF – Sun

<http://java.sun.com/javaee/javaxserverfaces/>

- ▶ JSF tutorial

<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSFIntro.html>

- ▶ JSF community

<https://javaxserverfaces.dev.java.net/>

- ▶ JSF technology extensions

<https://jsf-extensions.dev.java.net/>

- ▶ facelets

<https://facelets.dev.java.net/>



# JSF 2 vs JSF 1.2

- ▶ najznačajniju izmjenu u novoj verziji specifikacije predstavlja prenos kompletne arhitekture sistema u domen *Java Enterprise* web aplikacija (u skladu sa J2EE specifikacijom)
- ▶ ovom promjenom su JSF aplikacijama stavljene na raspolaganje funkcionalnosti koje nisu dio same JSF specifikacije, već postoje u okviru J2EE okruženja

# JSF 2 vs JSF 1.2

- ▶ **podrška za anotacije unutar *bean*-ova**
- ▶ uvođenjem podrške za anotacije unutar *bean*-ova omogućava se deklarisanje opsega važenja *Managed bean*-ova i izvan *faces-config* datoteke
  - *Managed bean*-ove je i dalje moguće registrovati u *faces-config* datoteci

```
// JSF 2.0
@ManagedBean(name="testBean")
@SessionScoped
public class TestBean {
    ...
}
```

```
// JSF 1.X
<managed-bean>
    <managed-bean-name>testBean</managed-bean-name>
    <managed-bean-class>TestBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

- ▶ ovakav vid „decentralizacije“ programske logike olakšava dalje unapređenje aplikacije i smanjuje njenu kompleksnost

# JSF 2 vs JSF 1.2

- ▶ **podrška za anotacije unutar *bean*-ova**
- ▶ osim postojećih opsega dostupnosti *bean*-ova (opseg aplikacije, opseg sesije i opseg zahtjeva), uvedeni su i:
  - *View Scope*,
  - *Flash Scope* i
  - *Custom Scope*.
- ▶ opseg dostupnosti *View* i *Flash Scope Managed bean*-ova veći je od opsega zahtjeva, a manji od opsega sesije
- ▶ *Custom Scope* se specificira korištenjem EL ( *Expression Language*) izraza, a ne ključne riječi

```
<managed-bean>  
  <managed-bean-name>testBean</managed-bean-name>  
  <managed-bean-class>TestBean</managed-bean-class>  
  <managed-bean-scope>#{someCustomScope}</managed-bean-scope>  
</managed-bean>
```



# JSF 2 vs JSF 1.2

- ▶ **navigacija – implicitna navigaciona pravila**
- ▶ navigaciona pravila u tom slučaju ne moraju biti eksplicitno navedena, već se na bazi rezultata *action controller* metoda implicitno mapira odgovarajući pogled (*view*) u formi fajla sa odgovarajućim imenom na fajl sistemu
- ▶ implicitno navigaciono pravilo se koristi u slučaju kada ne postoji eksplicitno navigaciono pravilo
- ▶ mogućnosti iz prethodne verzije ovim dodacima nisu izbačene, pa je odluka o vrsti navigacije koja se koristi ostavljena programeru

```
public String test() {  
    if (Math.random() < 0.5) {  
        return("first");           => first.xhtmll  
    } else {  
        return("second");          => second.xhtmll  
    }  
}
```

# JSF 2 vs JSF 1.2

- ▶ **navigacija – uslovna navigaciona pravila**
- ▶ drugo poboljšanje navigacionog podsistema ogleda se u uslovnim navigacionim pravilima
- ▶ uslovna navigacija se implementira kao EL izraz korištenjem novog `<if>` konfiguracionog elementa

```
<navigation-rule>  
  <display-name>page1.xhtml</display-name>  
  <from-view-id>/page1.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/page2.xhtml</to-view-id>  
    <if>#{testBean.someCondition}</if>  
  </navigation-case>  
</navigation-rule>
```

# JSF 2 vs JSF 1.2

- ▶ navigacija – mehanizam navigacije sa izvorišne na odredišnu stranu
- ▶ JSF 1.x specifikacija definiše *forward* serverske strane kao mehanizam navigacije sa izvorišne na odredišnu stranu
- ▶ pored ovog mehanizma navigacije, JSF 2.0 specifikacija definiše i redirekciju strane kao mehanizam navigacije sa izvorišne na odredišnu stranu
- ▶ kao podrazumijevani mehanizam navigacije koristi se *forward* serverske strane, dok se redirekcija aktivira dodavanjem “faces-redirect=true” stringa na kraj izlaznog stringa
- ▶ drugi način aktiviranja redirekcije jeste korištenje `<redirect/>` elementa u okviru `<navigation-case>` elementa navigacionog pravila

# JSF 2 vs JSF 1.2

- ▶ navigacija – mehanizam navigacije sa izvorišne na odredišnu stranu
- ▶ drugi način aktiviranja redirekcije jeste korišćenje `<redirect/>` elementa u okviru `<navigation-case>` elementa navigacionog pravila

```
<navigation-rule>  
  <display-name>page1.xhtml</display-name>  
  <from-view-id>/page1.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/page2.xhtml</to-view-id>  
    <redirect />  
  </navigation-case>  
</navigation-rule>
```

# JSF 2 vs JSF 1.2

- ▶ **template-ing**
- ▶ *Facelets* biblioteka je zbog svojih dobrih osobina uvedena kao osnovni okvir za implementaciju interfejsa prema krajnjim korisnicima, čime su uvedene značajne izmjene u načinu projektovanja prezentacionog sloja web aplikacije
- ▶ *Facelets* biblioteka je razvijena kako bi se JSF aplikacijama omogućio *templating* mehanizam, tj. mehanizam boljeg i efikasnijeg iskorištenja postojećeg programskog koda “razbijanjem” kompletne strane na dijelove koji se mogu iskoristiti u identičnoj formi na više strana u okviru iste aplikacije (zaglavlja strane, podnožja strane, meniji itd.).

# JSF 2 vs JSF 1.2

- ▶ **template-ing**
- ▶ u JSF verziji 1, postojalo je više sličnih biblioteka (*Tapestry*, *Tiles*) od kojih nijedna nije bila standardizovana u okviru JSF specifikacije
- ▶ zbog uočenih prednosti *Facelets* biblioteke nad ostalim bibliotekama iste namjene, ona je uvrštena u samu specifikaciju donoseći mogućnosti *templating*-a u okviru standardne specifikacije
- ▶ JSP podrška za izgradnju korisničkog interfejsa i dalje je zadržana u JSF 2.0 specifikaciji, s tim da novouvedene osobine u formi novih tagova uvedenih u JSF 2.0 nisu podržane
  - primjer: nije moguće koristiti ugrađenu AJAX podršku ili korisnički definisane komponente

# JSF 2 vs JSF 1.2

- ▶ **ugrađena AJAX podrška**
- ▶ podrška za AJAX komponente predstavlja značajno unapređenje u novoj verziji JSF specifikacije
- ▶ JSF 1.x aplikacije su AJAX funkcionalnosti preuzimale iz velikog broja postojećih biblioteka koje su predstavljale nadogradnju osnovne arhitekture
- ▶ JSF 2.0 donosi integrisanu podršku za AJAX funkcionalnosti, najviše u domenu parcijalnog osvježavanja strane i sličnih aspekata asinhronne komunikacije

```
<h:inputText value="#{testBean.text}">  
    <f:ajax execute="@form" event="keyup" render="result"/>  
</h:inputText>
```

# JSF 2 vs JSF 1.2

- ▶ **ugrađena AJAX podrška**
- ▶ pored ugrađene AJAX podrške, zadržana je i mogućnost integracije vanjskih komponenata iz bilo koje od biblioteka koje podržavaju novu specifikaciju
- ▶ bitno je napomenuti da, zbog prethodno navedenih razlika arhitektura sistema, postoje problemi u kompatibilnosti AJAX biblioteka razvijenih za starije aplikacije u odnosu na zahtjeve koje postavlja nova specifikacija
- ▶ broj komponenata koje se mogu iskoristiti u novim aplikacijama je u konstantnom porastu i već uveliko prelazi minimum koji je potreban za implementaciju čak i naprednijih aplikacija, iako se radi o relativno novoj specifikaciji



# JSF 2 vs JSF 1.2

- ▶ **podrška za debug-ovanje**
- ▶ razvoj moderne web aplikacije nije jednostavan, pa je podrška za *debug*-ovanje u fazi razvoja aplikacije je veoma značajan aspekt aplikativnog okruženja
- ▶ JSF 2.0 specifikacija uvodi PROJECT\_STAGE parametar u *web.xml* konfiguracionoj datoteci aplikacije koji može imati vrijednosti: *Production, Development, UnitTest, SystemTest* i *Extension*

```
<context-param>  
    <param-name>javax.faces.PROJECT_STAGE</param-name>  
    <param-value>Development</param-value>  
</context-param>
```

- ▶ korištenjem ovog parametra mnoge greške koje bi kod JSF 1.x aplikacija ostale neprimjećene, sada mogu biti jednostavno detektovane

# JSF 2 vs JSF 1.2

- ▶ korištenje JSF EL za ispisivanje vrijednosti *property*-ja *bean*-ova
- ▶ prema ranijoj specifikaciji ovo nije bio slučaj
- ▶ primjer korištenja *outputText* komponente u JSF 1.x i korištenje JSF EL za ispis vrijednosti *property*-ja *bean*-a

```
// JSF 2.0
```

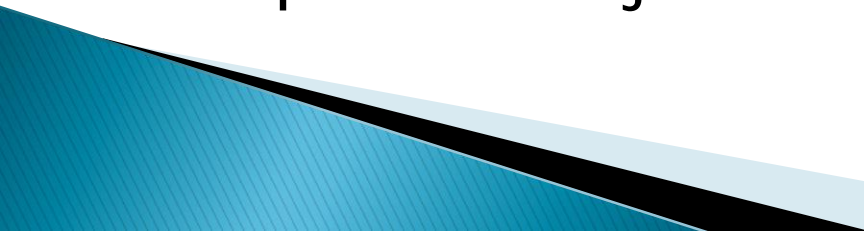
```
{ userBean.usernameProperty }
```

```
// JSF 1.x
```

```
<h:outputText value="#{userBean.usernameProperty}"/>
```

- ▶ korištenje *outputText* komponente u JSF 2.0 aplikacijama je i dalje omogućeno, s tim što se ona obavezno mora koristiti u slučaju kada ju je potrebno opcionalno prikazivati ili kada je potrebno ispisivati HTML kod u JSF stranicu.

# JSF 2 vs JSF 1.2

- ▶ **jednostavnije kreiranje korisničkih komponenti**
  - ▶ **veoma bitno poboljšanje koje donosi JSF 2.0 specifikacija**
  - ▶ **podrška za implementaciju korisničkih komponenti koja je postojala u JSF 1.x aplikacijama bila je veoma značajna jer je dovela do kreiranja velikog broja biblioteka, poput *RichFaces*, *IceFaces*, *Tomahawk*, *WebGalileo* i ADF**
  - ▶ **ovaj API u verziji JSF 1.x bio je veoma komplikovan**
  - ▶ **JSF 2.0 specifikacija uvodi novi pristup u implementaciji korisničkih komponenti**
- 

# JSF 2 vs JSF 1.2

- ▶ **jednostavnije kreiranje korisničkih komponenti**
- ▶ novi pristup baziran je na *face/ets*-ima i omogućava relativno jednostavno kreiranje novih jednostavnih i srednje komplikovanih komponenti

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:composite="http://java.sun.com/jsf/composite">
<head>
<title>Test Composite Component</title>
</head>
<body>
<composite:interface>
  <composite:attribute name="test"/>
</composite:interface>
<composite:implementation>
  <h:outputText value="Hello, #{cc.attrs.test}!"/>
</composite:implementation>
</body>
</html>
```

# JSF 2 vs JSF 1.2

- ▶ **podrška za GET zahtjeve**
- ▶ JSF 2.0 specifikacija uvodi i značajniju podršku za GET zahtjeve
- ▶ ova podrška uključuje i nove komponente `<h:link>` i `<h:button>`

```
<h:link outcome="success">  
  <f:param name="param1" value="abc"/>  
</h:link>
```

# Anotacije

- ▶ **@ManagedBean anotacija**

```
@ManagedBean  
public class TestBean{  
    private String abc;  
    ...  
}
```

- ▶ navođenje: #{testBean.abc}, gdje naziv bean-a odgovara nazivu klase, s tim što je prvo slovo naziva malo slovo
- ▶ request scope je podrazumijevani scope
- ▶ abc je naziv property-ja (koji ima odgovarajuće getere i setere) ili naziv metode
- ▶ ako action controller metoda vraća string “xyz”, i ako ne postoje eksplicitno definisana navigaciona pravila, onda je rezultujuća strana xyz.xhtml

# Anotacije

- ▶ **name atribut @ManagedBean-a**

```
@ManagedBean(name="testBean2")  
public class TestBean{  
    private String abc;  
    ...  
}
```

- ▶ navođenje: #{testBean2.abc}, gdje je testBean2 vrijednost name atributa
- ▶ request scope je i dalje podrazumijevani scope

# Anotacije

- ▶ **eager atribut @ManagedBean-a**

```
@ManagedBean(eager=true)
@ApplicationScoped
public class TestBean{
    ...
    ...
}
```

- ▶ ako je “eager=true” i scope je application, onda bean mora biti kreiran u trenutku učitavanja aplikacije, a ne u trenutku prvog referenciranja bean-a, tj. prije opsluživanja bilo kojeg zahtjeva

```
<managed-bean eager="true">
<managed-bean-name>cbbhBean</managed-bean-name>
<managed-bean-class>net.etfbl.s_cube.external.beans.CBBHBean</managed-bean-class>
<managed-bean-scope>application</managed-bean-scope>
</managed-bean>
```



# Anotacije

- ▶ **scope**
- ▶ **@RequestScoped**
  - podrazumijevani
  - nova instanca se kreira pri svakom HTTP zahtjevu
- ▶ **@SessionScoped**
  - scope sesije
  - korisnik sa istim cookie-jem, ako sesija nije istekla, uvijek pristupa istom sesijskom bean-u
  - bean bi trebao biti serijalizabilan
- ▶ **@ApplicationScoped**
  - scope aplikacije
  - dijele ga svi korisnici aplikacije
  - ovaj bean ili nema stanje ili je potrebno izvršiti sinhronizaciju pristupa

# Anotacije

- ▶ **scope**
- ▶ **@ViewScoped**
  - ista instanca bean-a se koristi dok god je korisnik na istoj stranici, npr. u slučaju AJAX zahtjeva
  - trebao bi biti serijalizabilan
- ▶ **@CustomScoped(value="#{someMap}")**
  - bean je smješten u Map objekat, i programer upravlja njegovim životnim vijekom
- ▶ **@NoneScoped**
  - bean nije smješten u scope
  - korisno u slučajevima kada bean treba biti referenciran od strane drugih bean-ova koji su u nekom od scope-ova

# Anotacije

- ▶ **scope**
- ▶ obično se smješta iza @ManagedBean

```
@ManagedBean  
@SessionScoped  
public class TestBean {  
    ...  
}
```

# Anotacije

- ▶ **@ManagedProperty**
- ▶ **JSF podržava dependency injection**
  - moguće je dodijeliti vrijednosti property-ju managed bean-a, bez hard kodovanja u definiciji klase
- ▶ **način I:**  
`@ManagedProperty(value="#{someBean}")`  
`private SomeType someField;`
- ▶ **način II:**
  - `<managed-property>` u `faces-config.xml` datoteci
  - isto kao i u verziji JSF 1.x
- ▶ **setter metoda za property je obavezna**  
`setSomeField`
- ▶ **ako ime setter metode ne odgovara imenu property-ja, moguće je koristiti "name" atribut @ManagedProperty-ja**

# AJAX

## ▶ f:ajax

```
<h:inputText value="#{testBean.text}">  
<f:ajax execute="@form" event="keyup" render="result"/>  
</h:inputText>  
  
...  
<h:outputText value="#{testBean.text}" id="result"/>
```

# AJAX

- ▶ f:ajax atributi
- ▶ render
  - id elemenata ili id-evi liste elemenata koji se trebaju osvježiti (izmijeniti u DOM-u) nakon izvršavanja AJAX zahtjeva
  - često je to h:outputText
  - specijalne vrijednosti @this, @form, @none i @all – obično se ne koriste kao vrijednosti render atributa
- ▶ execute
  - elementi koji se šalju na serversku stranu radi procesiranja
  - često input elementi kao što je h:inputText ili h:selectOneMenu
  - @this – element koji okružuje f:ajax – podrazumijevano
  - @form – h:form koji okružuje f:ajax – kada je potrebno da se pošalju vrijednosti svih parametara forme
  - @none – ništa se ne šalje
  - @all – svi JSF UI elementi sa stranice
- ▶ event
  - DOM event na koji se aktivira AJAX zahtjev (npr., keyup, blur)
- ▶ onevent
  - JavaScript funkcija koja se pokreće kada se “okida” event

# Looping

- ▶ rad sa sadržajem promjenljive veličine
- ▶ mogućnosti:
  - bean metoda koja vraća string ili HTML kod
  - h:dataTable
  - korisnička kompozitna komponenta
  - korišćenje ui:repeat

# Looping

- ▶ rad sa sadržajem promjenljive veličine
- ▶ bean metoda koja vraća string ili HTML kod
  - kolekcija se pretvara u string ili HTML
  - korisno
    - kada je izlaz jednostavan tekst ili veoma jednostavan HTML
    - kada izlaz ne koristi CSS
  - nije korisno
    - kada je potrebna veća kontrola nad izlazom



# Looping

- ▶ rad sa sadržajem promjenljive veličine
- h:dataTable
  - ugrađena komponenta pretvara kolekciju u HTML tabelu
  - korisno
    - kada je potrebno kreirati HTML tabelu na osnovu podataka sadržanih u kolekciji
  - nije korisno
    - kada je potrebno kreirati nešto što nije HTML tabela
    - kada različiti dijelovi tabele dolaze iz različitih izvora

# Looping

- ▶ rad sa sadržajem promjenljive veličine
- korisnička kompozitna komponenta
  - korisnička komponenta pretvara kolekciju u HTML izlaz
  - korisno
    - kada je potrebna kreirati nešto što nije HTML tabela na osnovu podataka sadržanih u kolekciji
  - nije korisno
    - kada su podaci u formatu drugačijem od onog koji je očekivan
    - kada je potrebno napraviti izmjenu u grafičkom dizajnu (makar i najmanju)

# Looping

- ▶ rad sa sadržajem promjenljive veličine
- korišćenje `ui:repeat`
  - facelets looping za kreiranje HTML-a unutar stranice
  - korisno
    - kada je potrebna veća kontrola nad izlazom
    - kada jedna od prethodnih opcija nije odgovarajuća
  - nije korisno
    - kada HTML kod stranice postaje suviše kompleksan

# Looping

- ▶ rad sa sadržajem promjenljive veličine
- korišćenje ui:repeat
  - uključiti JSTL 1.2 u CLASSPATH
  - uključiti facelets namespace  
xmlns:ui="http://java.sun.com/jsf/facelets"
  - koristiti ui:repeat isto kao i JSTL c:forEach
    - razlika: c:forEach se izvršava kada se stablo komponenti gradi, a ui:repeat kada se stablo renderuje

```
<ui:repeat var="x" value="#{testBean.collection}">  
...<HTML kod>...  
#{x.someProperty}  
...</HTML kod>...  
</ui:repeat>
```

# Looping

- ▶ rad sa sadržajem promjenljive veličine
- atributi ui:repeat
  - var – ime kojim će biti referenciran element kolekcije
  - value – EL izraz koji specificira kolekciju
  - varStatus – ime kojim će biti referenciran status objekt (korisne osobine ovog objekta: index, first/last, even/odd)
  - offset – specificira početni element kolekcije – podrazumijevana vrijednost je 0
  - size – specificira posljednji element kolekcije – podrazumijevana vrijednost je posljednji element kolekcije
  - step – specificira korak pri prolasku kroz kolekciju – podrazumijevana vrijednost je 1

# Korisničke kompozitne komponente

- ▶ definisanje komponente

- abc.xhtml smješta se u “resources/xyz” folder
- dostupni atributi se definišu u composite:interface
- izlaz se specificira u composite:implementation
- u fajlu komponente koristi se composite namespace  
`xmlns:composite="http://java.sun.com/jsf/composite"`

- ▶ u facelets stranici koristi se component namespace

`xmlns:scube=http://java.sun.com/jsf/composite/scubecomponents`

- ▶ nakon toga, moguće je koristiti komponentu u facelets stranici

`<scube:cbbhcurrencyexchange panelStyleClass="exchange" />`

# Korisničke kompozitne komponente

```
<composite:interface>
<composite:attribute name="panelStyleClass"/>
</composite:interface>

<composite:implementation>
<h:panelGrid styleClass="#{cc.attrs.panelStyleClass}">
<rich:panel header="#{msgs.cbbh_currency_exchange}">
<h:form>
<h:panelGrid columns="3" styleClass="#{cc.attrs.panelStyleClass}" frame="border">
<h:outputText value="#{msgs.cbbh_amount}"/>
<h:inputText id="currencyKM" size="10" value="#{cbbhBean.currencyKM}">
<f:validateLongRange minimum="0" maximum="100000"/>
    <f:ajax execute="@this" render="conversionValue" event="blur"/>
</h:inputText>
<h:outputText value="KM"/>
<h:outputText value="#{msgs.cbbh_currency}"/>
<h:selectOneMenu value="#{cbbhBean.selectedCurrency}">
<f:selectItem itemValue="" itemLabel="#{msgs.cbbh_choose}" />
    <f:selectItems value="#{cbbhBean.selectItems}" />
    <f:ajax execute="@this" render="conversionValue" event="valueChange"/>
</h:selectOneMenu>
<h:outputText/>
<h:outputText value="#{msgs.cbbh_conversion}"/>
<h:outputText id="conversionValue" value="#{cbbhBean.conversionValue}">
<f:convertNumber maxFractionDigits="2"/>
</h:outputText>
<h:outputText/>
<h:outputText/>
<h:outputLink value="http://www.cbbh.ba" target="_blank" title="CBBH"><h:outputText value="#{msgs.yahoo_source}: CBBH"/></h:outputLink>
</h:panelGrid>
</h:form>
</rich:panel>
</h:panelGrid>
</composite:implementation>
```

# f:viewParam, GET i bookmarking

- ▶ f:viewParam omogućava povezivanje property-ja bean-a sa request parametrima
- ▶ iz ovog nastaju nove mogućnosti:
  - tagovi koji koriste GET, a ne POST metodu i šalju parametre kroz URL
  - slanje podataka iz ne-JSF forme ka JSF stranama
  - bookmark stranica
- ▶ ovo je nova osobina JSF 2.0



# f:viewParam, GET i bookmarking

- ▶ prihvatanje request parametara

```
<!DOCTYPE ...>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:f="http://java.sun.com/jsf/core"
xmlns:h="http://java.sun.com/jsf/html">
<f:metadata>
<f:viewParam name="param1" value="#{bean.prop1}"/>
<f:viewParam name="param2" value="#{bean.prop2}"/>
</f:metadata>
<h:head>...</h:head>
<h:body>
Blah, blah, #{bean.prop1}
Blah, blah, #{bean.prop2}
Blah, blah, #{bean.derivedProp}
</h:body>
</html>
```

# f:viewParam, GET i bookmarking

- ▶ slanje request parametara
- ▶ komponente
  - h:link
  - h:button

```
<h:link outcome="abc?param1=v1&param2=v2"  
value="Click"/>
```

```
<h:button outcome="abc?param1=v1&param2=v2"  
value="Click"/>
```

- u ovom primjeru odredište je abc.xhtml

# f:viewParam, GET i bookmarking

- ▶ slanje podataka JSF aplikaciji iz ne-JSF aplikacije
- ▶ kreira se “klasična” HTML forma

```
<form action="abc.jsf">
```

Param 1:

```
<input type="text" name="param1"/><br/>
```

Param 2:

```
<input type="text" name="param2"/><br/>
```

```
<input type="submit" value="Go"/>
```

```
</form>
```

# f:viewParam, GET i bookmarking

- ▶ bookmarking
- ▶ redirekcija sa viewParam
- ▶ za implicitnu navigaciju dodaje se  
“includeViewParams=true” na kraj outcome stringa
- ▶ za eksplicitnu navigaciju korišćenjem faces-  
config.xml datoteke dodaje se  
`<redirect include-view-params="true"/>` u navigaciona pravila

# JSF 2

- ▶ JSF 2.3 / 2.2
  - <https://javaee.github.io/javaxserverfaces-spec/>
- ▶ JSF specifikacija / download
  - <https://projects.eclipse.org/projects/ee4j.mojarra>
  - <https://javaxserverfaces.github.io/>
  - <http://myfaces.apache.org/>