



Funded by the  
Erasmus+ Programme  
of the European Union



---

This project has been funded with support from the European Commission. This publication [communication] reflects the views only of the author, and the Commission cannot be held responsible for any use which may be made of the information contained therein.

---



# KI204 - JAVA 7: JAVA ENTERPRISE EDITION

Servleti

Lekcija 01

PRIRUČNIK ZA STUDENTE

# KI204 - JAVA 7: JAVA ENTERPRISE EDITION

## Lekcija 01

### *SERVLETI*

- ✓ Servleti
- ✓ Poglavlje 1: Uvod u servlete
- ✓ Poglavlje 2: Kreiranje i postavljanje servleta
- ✓ Poglavlje 3: Tokovi podataka
- ✓ Poglavlje 4: Servleti i sesije
- ✓ Poglavlje 5: GlassFish server
- ✓ Poglavlje 6: Domaći zadatak 1
- ✓ Zaključak

Copyright © 2017 – UNIVERZITET METROPOLITAN, Beograd. Sva prava zadržana. Bez prethodne pismene dozvole od strane Univerziteta METROPOLITAN zabranjena je reprodukcija, transfer, distribucija ili memorisanje nekog dela ili čitavih sadržaja ovog dokumenta., kopiranjem, snimanjem, elektronskim putem, skeniranjem ili na bilo koji drugi način.

Copyright © 2017 BELGRADE METROPOLITAN UNIVERSITY. All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning or otherwise, without the prior written permission of Belgrade Metropolitan University.

## ▼ Uvod

### UVOD

*Lekcija će se fokusirati na obradu tema vezanih za kreiranje Java EE servleta.*

U ovoj lekciji će posebna pažnja biti obrađena na analizu i demonstraciju mehanizama za kreiranje i upotrebu klasa servleta u Java EE 7 veb aplikacijama. Posebno će biti neophodno napraviti dobar uvod u ovu problematiku:

- definisati servlete;
- pokazati kako se kreiraju;
- pokazati njihovu organizaciju;
- pokazati skup izvesnih pravila koji se poštuju prilikom rada sa servletima.

U nastavku lekcija se bavi kreiranjem i postavljanjem servleta. Posebno, u ovom delu lekcije, biće diskutovano o fazama životnog ciklusa servleta.

Problematici praćenja sesija biće posvećeno najviše vremena u okviru ove lekcije. Posebno će biti posmatrani slučajevi primene kolačića i modifikacije URL adrese za realizovanje proces praćenja sesija. Ovaj deo lekcije se fokusira takođe i na primenu klasa filtera i asinhronih servleta.

Lekcija završava izlaganje diskusijom o korišćenom aplikacionom serveru **GlassFish**.

## ▼ Poglavlje 1

# Uvod u servlete

## UVODNA RAZMATRANJA

*Servlet je server - side komponenta koja se izvršava isključivo unutar Java virtualne mašine.*

Servlet je server - side komponenta koja se izvršava isključivo unutar Java virtualne mašine. Pošto se servlet izvršava na serverskoj strani ne proverava se kompatibilnost sa browser - om. Servlet može pristupiti čitavoj familiji Java API-ja, uključujući JDBC API za pristup bazama podataka. Servlet, takođe, može pristupiti biblioteci HTTP specifičnih poziva, poprimiti sve povoljnosti Java jezika uključujući prenosivost, performanse, ponovno korišćenje i zaštitu. Servleti predstavljaju popularan način izgradnje interaktivnih web aplikacija. Trebalo bi napomenuti da su servletski kontejneri su uglavnom komponente web ili aplikacionog servera, kao što su BEA WebLogic Application Server, IBMWebSphere i Sun Java System Web Server.

Servleti nisu projektovani za specifične protokole. Oni najčešće koriste HTTP protokol i klase koje su smeštene u `javax.servlet` i `javax.servlet.http` Javinim paketima. Servleti obezbeđuju sofisticirani način kreiranja serverske strane prateći standardno J2EE okruženje i koristeći visoko prenosiv Java programski jezik. HTTP servlet se obično koristi da:

- Obezbedi dinamički sadržaj kao što je uzimanje rezultata upita i vraćanje istih do klijenta.
- Obrada i čuvanje podataka koji se nalaze na HTML strani.
- Upravljanje informacijama koja se odnose na stanje HTTP-a.

Web aplikacije mogu da pružaju statički ili dinamički sadržaj. Primer statičkog sadržaja su tekstualne datoteke koje sadrže HTML, slike i video. Dinamički sadržaj se generiše u toku izvršavanja.

Servlet je komponenta koja se koristi za proširenje funkcionalnosti web servera. Obezbeđuje objektno-orijentisanu apstrakciju za izgradnju dinamičkih web aplikacija. Servlet pripada serverskoj strani i može dinamički da generiše HTML kao rezultat HTTP zahteva. On prima zahtev sa klijentskog hosta (web čitač) i šalje odgovor ka istom.

Servlet je Java klasa napisana na osnovu određenih pravila i biće isporučena i Java EE kontejner koji programer odabere. Klijent program može biti jednostavan HTML / JavaScript kod, applet, Swing ili JavaFX programa. Za komunikaciju sa servletima, koriste se web pregledači.

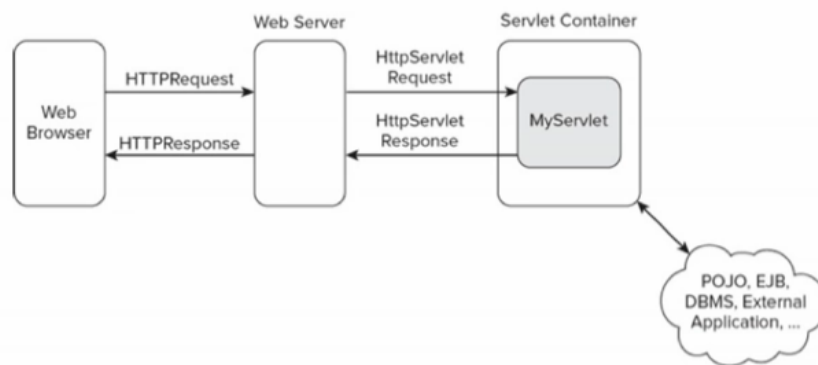
Upravo u narednim izlaganjima će biti akcenat na analizi i demonstraciji tehnologija i alata za kreiranje i implementaciju servleta.

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

## SERVLETI - OPŠTI PRIKAZ

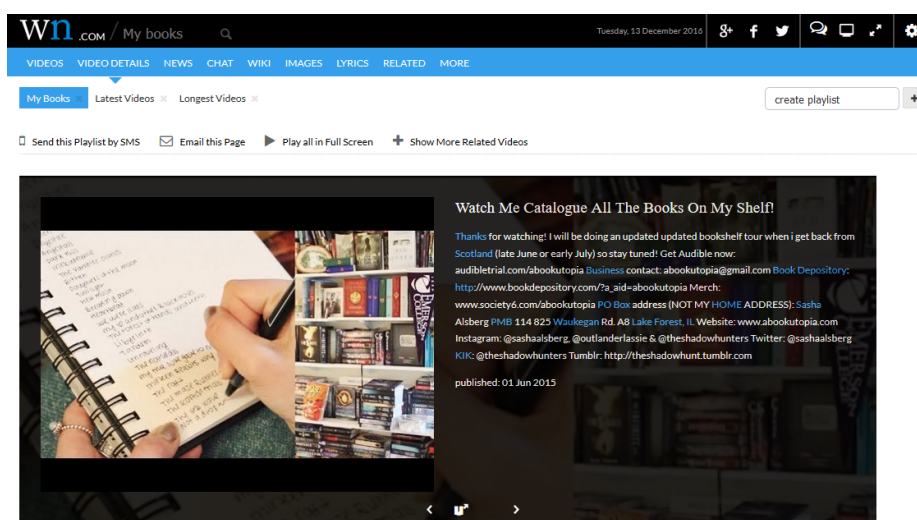
*Neophodno je razumeti komponente i tokove podataka.*

Sljedećom slikom je prikazan web pregledač (Mozilla, IE, Google Chrome, Safari ili neki drugi) čiji je zadatak slanje HTTP zahteva do servleta čiji je naziv **MyServlet** i prijem HTTP odgovora koje mu servlet prosleđuje.



Slika 1.1 Tok podataka između klijenta i servera

Pre nego što studenti savladaju mehanizme pokretanja i upotrebe servera, neophodno im je pružiti ilustraciju komponenata i tokova podataka u simulaciji web prodavnice, dostupne na stranici [www.MyBooks.com](http://www.MyBooks.com), koja je realizovana pomoću servleta (izvor: Yakov Fain, Java 8 programiranje, Kombib - 2015). Sljedećom slikom je prikazan izgled navedene veb prodavnice koja će poslužiti kao dobar osnov za narednu analizu i demonstraciju.



Slika 1.2 Veb prodavnica realizovana pomoću servleta

U narednom izlaganju biće postavljeni temelji nad kojima se kreira servlet aplikacija.

# SERVLETI - PRAVILA KREIRANJA I IZVRŠAVANJA

*Servlet aplikacija mora da poštuje izvesna pravila da bi mogla da bude korišćena.*

Kao što je napomenuto u uvodnim razmatranjima, servlet aplikacija mora da poštuje izvesna pravila. U narednom izlaganju će biti predložena pravila koja mora da zadovolji servlet aplikacija, a imajući u vidu priloženi primer veb prodavnice.

- **Klijent računar kojim se obraća servletu mora da ima instaliran veb pregledač.** Aplikacija koja je priložena kao primer, sastoji se od više HTML stranica koje služe za interakciju aplikacije sa korisnicima.
- **Računar koji hostuje stranicu kreiranu pomoću servletamora da pokrene izvesti veb server** - obično na portu 80;
- **Veb server ima zadatak da osluškuje zahteve korisnika** - Zahtevi mogu biti prosti i složeni. U slučaju prijema prostog zahteva (statički HTML sadržaj - datoteke ili slike), obrada zahteva se realizuje jednostavno bez primene dodatnog softvera i vraća nazad `HttpServletResponse` paket sa traženim statičkim sadržajem;
- **Stranica koja je kreirana pomoću servleta (primer MyBooks.com) izvršava kontejner servleta zajedno sa isporučenim servletom** - Ako server prihvati izvesni korisnički zahtev, na primer pronalaženje knjiga na osnovu kreiranog kriterijuma pretrage, on će kreirati i proslediti `HttpServletRequest` do odgovarajućeg servleta. Ovaj servlet može biti, na primer `FindBookServlet`, koji će nakon isporučivanja biti izvršen u kontejneru servleta;
- **Servlet angažuje Java kod čiji je zadatak realizovanje pretrage** (u ovom slučaju knjiga) čiji će rezultati biti prikazani u odgovarajućoj HTML stranici generisnoj tokom izvršavanja ovog Java koda. Rezultat se šalje veb serveru u formi `HttpServletResponse` objekta;
- **Veb server izoluje sadržaj iz HttpServletResponse objekta**, pakuje ga u `HttpServletResponse` objekat i šalje ga u izabrani veb pregledač;
- **Veb pregledač prikazuje korisniku primljenu stranicu** - veb pregledač ne poseduje informacije da li je stranica statička ili dinamička.

Veb pregledač ne mora da bude upućen u primenjenu serversku tehnologiju za kreiranje dinamičkog sadržaja. Njegov zadatak je slanje podataka, u formi `HttpRequest` objekta, primenom HTTP protokola. Veb pregledač mora da ima sposobnost prikazivanja sadržaja dobijenog od servera u formi `HttpServletResponse` objekata. Ostala obrada se ne tiče veb pregledača.

## PRIMER 1 - TANAK KLIJENT

*Biće kreirana jednostavna HTML stranica koja će da simulira klijent za Java servlet.*

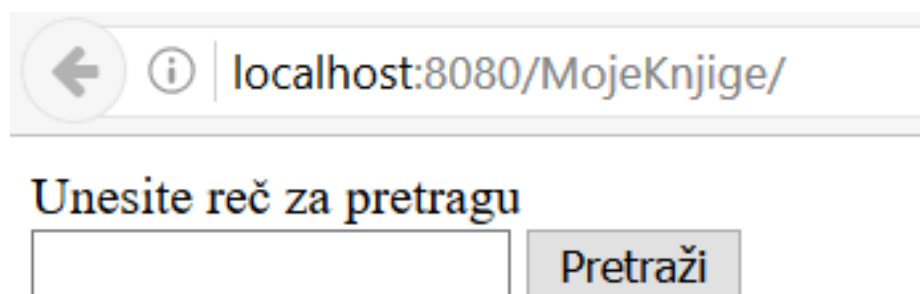
Za demonstraciju naveden problematike biće kreirana jednostavna HTML stranica koja će da simulira klijent za Java servlet. Stranica je po organizaciji i funkcionalnosti veoma

jednostavna. Sadrži polje za unos reči po kojoj će biti vršena pretraga, kao i dugme za slanje zahteva za pronalaženje knjige na osnovu unete ključne reči za pretragu. Sledećim listingom je priložen kod index.html datoteke koja odgovara navedenoj stranici.

```
<html>
  <head>
    <title>Pronađi knjigu</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div>Unesite reč za pretragu</div>
    <form action=http://www.MyBooks.com/servlet/FindBooksServlet method=Get>
      <input type=Text name=booktitle>
      <input type=Submit value="Pretraži">
    </form>
  </body>
</html>
```

Ova datoteka je mogla da bude kreirana u bilo kojem tekst editoru, međutim, ona je u ovom slučaju kreirana kao index.html datoteka u razvojnom okruženju NetBeans. Slikom 3 je prikazan izgled stranice koji odgovara uspešnom učitavanju ove datoteke.

Ukoliko se u ovoj fazi razvoja, pokrene ova datoteka, u bilo kojem veb pregledaču, javiće se poznata greška 404 . Razlog je veoma jednostavan. Na ovoj adresi ne postoje server sa odgovarajućom URL adresom , kao i servlet čiji bi naziv trebalo da bude FindBookServlet.



Slika 1.3 Uspešno učitavanja index.html datoteka

Posebno je neophodno analizirati pojavu grešaka. Ukoliko server ne uzvрати odgovorom biće prikazan odgovarajući kod greške. Međutim, ako klijent uspešno primi resurs prosleđen sa tražene URL adrese, HTTP statusni kod će imati vrednost na intervalu 200 - 300. Lista svih mogućih statusnih kodova je raspoloživa na veb adresi: [www.w3.org/Protocols/rfc2616/rfc2616-sec10.html](http://www.w3.org/Protocols/rfc2616/rfc2616-sec10.html).

## ZADATAK 1

*Samostalno kreiranje tankog klijenta*



- Po uzoru na prethodni zadatak, napravite vlastite modifikacije na prethodnoj stranici.
- slobodno koristite Internet prilikom rešavanja problema.

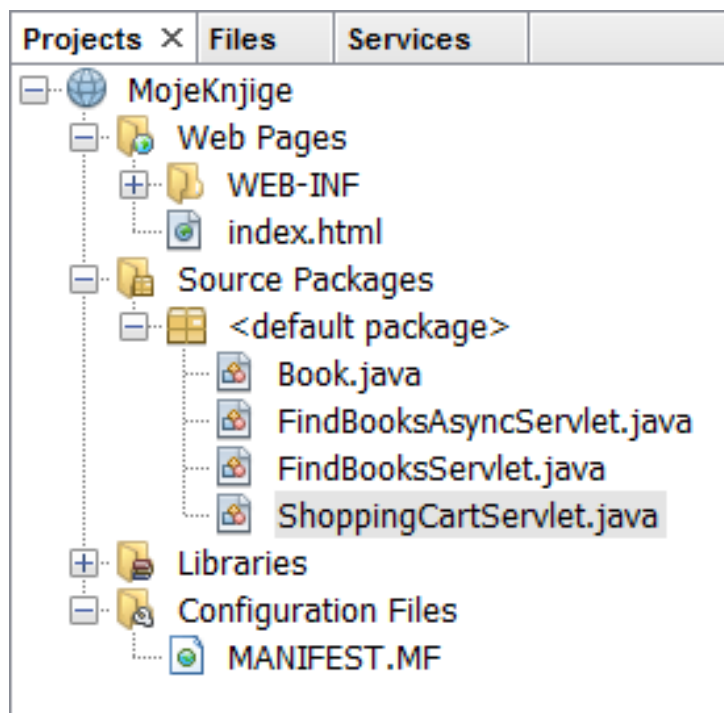
## ▼ Poglavlje 2

# Kreiranje i postavljanje servleta

## KREIRANJE SERVLETA

*Servlet predstavlja Java klasu koja nasleđuje osnovnu klasu `HttpServlet`.*

Servlet predstavlja Java klasu koja nasleđuje osnovnu klasu `HttpServlet` i koja je obeležena anotacijom `@WebServlet`. Primenom razvojnog okruženja NetBeans, servlet klasa se veoma jednostavno kreira. Ona se u hijerarhiji projekta nalazi zajedno sa ostalim klasama u okviru foldera `Source Packages` (sledeća slika).



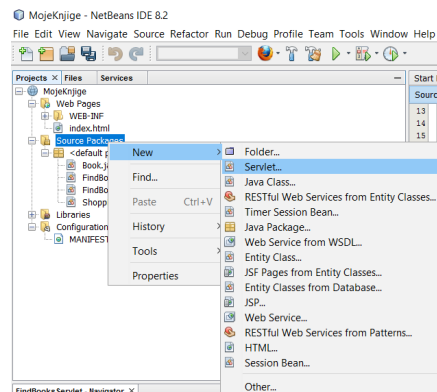
Slika 2.1 Obične i servlet Java klase u hijerarhiji projekta

Kao prvi korak, koji je prethodio i kreiranju pomenute `index.html` datoteke (koja se takođe vidi na slici, je bio kreiranje samog projekta. Projekat je kreiran kao java Web aplikacija pod nazivom `MojeKnjige`.

Kreiranje servlet klase je moguće obaviti veoma jednostavno u nekoliko koraka:

- desnim klikom na folder `Source Packages` i izborom opcije `New / Servlet` (sledeća slika) otvara se čarobnjak za unos podataka o servletu koji se kreira;
- unose se podaci relevantni za servlet poput naziva klase i paketa;

- Klikom na **Finish** kreirana je servlet klasa čiju programsku logiku bi trebalo definisati u narednim koracima.



Slika 2.2 Kreiranje servlet klase

## PRIMER 2 - KREIRANJE SERVLETA

*HttpServlet je naslednica klase GenericServlet.*

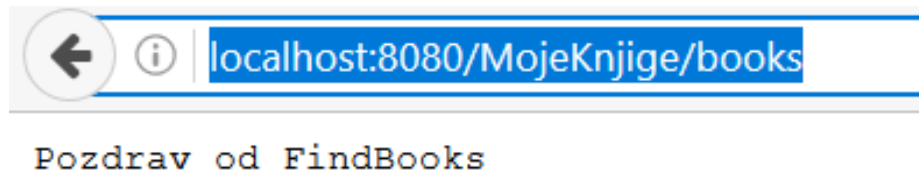
Klasa, koju nasleđuje kreirana servlet klasa, HttpServlet je naslednica klase GenericServlet u kojoj je definisana metoda service(). Ova metoda prima zahtev klijenta i usmerava ga ka metodama kreirane servlet klase.

Kreirana servlet klasa mora da, nakon kreiranja, dobije konkretnu implementaciju sa ciljem obezbeđivanja konkretne funkcionalnosti. Obično se vrši redefinisane neke od metoda **doGet()** ili **doPost()**. Koja će od metoda biti redefinisana i konkretizovana zavisi od klijentskog zahteva:

- Kada klijent prosleđuje HTTP zahtev pomoću metode Post, biće redefinisana **doPost()** metoda servleta;
- Ukoliko klijent koristi Get metodu (kao u slučaju priloženog koda datoteke index.html), biće redefinisana metoda **doGet()**.

Upravo, vodeći se navedenim, moguće je pristupiti definisanju programske logike servleta pod nazivom FindBooksServlet.java koji je prethodno kreiran na način prikazan prethodnim izlaganjem.

Za početak ovoj klasi će biti dodeljena anotacija **@WebServlet("/books")**. Anotacija preuzima jedan argument kojim je omogućeno povezivanje rezultata izvršavanja kreiranog servleta sa stranicom na kojoj će rezultati biti prikazani ako je servlet uspešno postavljen. Navedeno se realizuje navođenjem stringa `http://localhost:8080/MojeKnjige/books` u veb pregledaču (slika 6).



Slika 2.3 Izvršavanje servleta FindBooksServlet

Sledećim listingom je priložen kod kreiranog servleta sa redefinisanim metodom `doGet()`.

```
/**
 *
 * @author Vladimir Milicevic
 */
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class FindBooksServlet
 */
@WebServlet("/books", name="FindBooksServlet" )
public class FindBooksServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public FindBooksServlet() {
        super();
        // TODO Auto-generated constructor stub
    }

    /**
     * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse
    response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        // TODO Auto-generated method stub
        PrintWriter out = response.getWriter();
        out.println("Pozdrav od FindBooks");
    }
}
```

```

/**
 * @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse
response)
 */
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // TODO Auto-generated method stub
}
}

```

## PRIMER 3 - POSTAVLJANJE SERVLETA

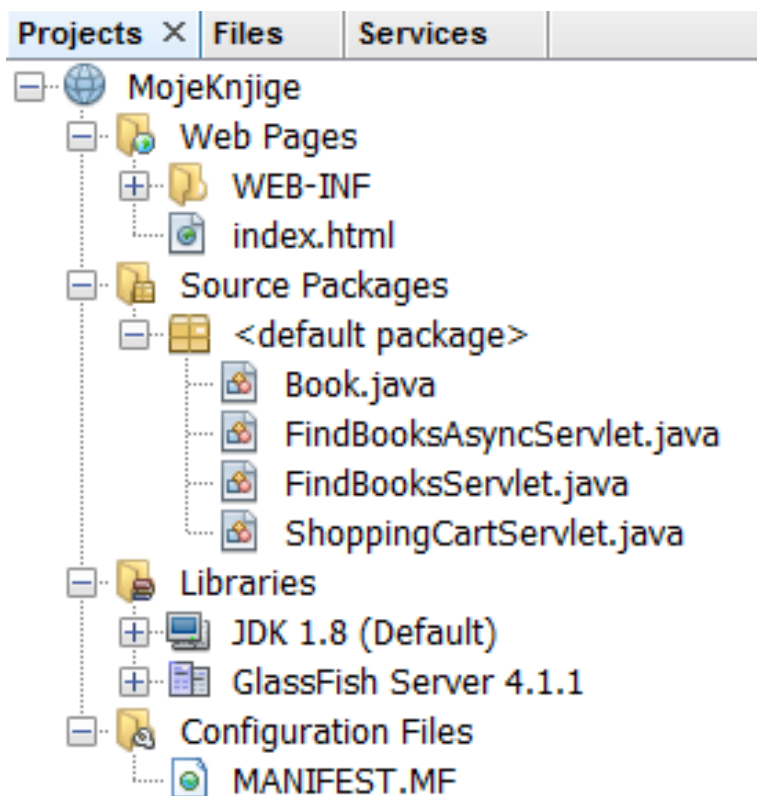
*Veoma važnu ulogu u novim verzijama Jave obavlja anotacija `@WebServlet`.*

Verzija Jave Enterprise Edition 7 donosi jednu novinu koja u velikoj meri olakšava rad sa servletima. **Primenom anotacije `@WebServlet` omogućena je direktna definicija i upotreba razvojnih parametara umesto potrebe za njihovim deklarisanjem u deskriptoru veb angažovanja `web.xml`.** Ako se obrati pažnja na kreirani servlet `FindBooksServlet`, moguće je primetiti da on koristi razvojne parametre `urlPatterns` i `name`. Prvi od njih se koristi za identifikovanje servleta na serveru pomoću URL adrese. Drugi parametar, čija vrednost predstavlja `/books`, ukazuje da uvek kada klijent šalje zahtev koji sadrži šablon `books` u URL adresi, biće izvršeno prosleđivanje zahteva ka servletu `FindBooksServlet`. U konkretnom slučaju to znači da kontejner servleta prosleđuje zahtev `http://localhost:8080/MojeKnjige/books` do servleta `FindBooksServlet`.

Projekat koji sadrži servlet klase je veoma precizno organizovan. Koren projekta je određen folderom koji nosi naziv samog projekta (MojeKnjige). Zatim je moguće uočiti podfoldere: **Web Pages, Source Packages, Libraries i Configuration Files.**

Folder Web Pages sadrži datoteke sa veb stranicama i podrazumevani podfolder WEB-INF u kojem se nalazi pominjana datoteka `index.html`.

U Folderu Source Packages nalazi se prostor rezervisan za čuvanje Java klase koje učestvuju u kreiranom projektu.



Slika 2.4 Kompletna struktura projekta MojeKnjige

U folderu Libraries čuvaju se datoteke platforme, servera i druge .jar datoteke značajne za kreiranje veb aplikacije.

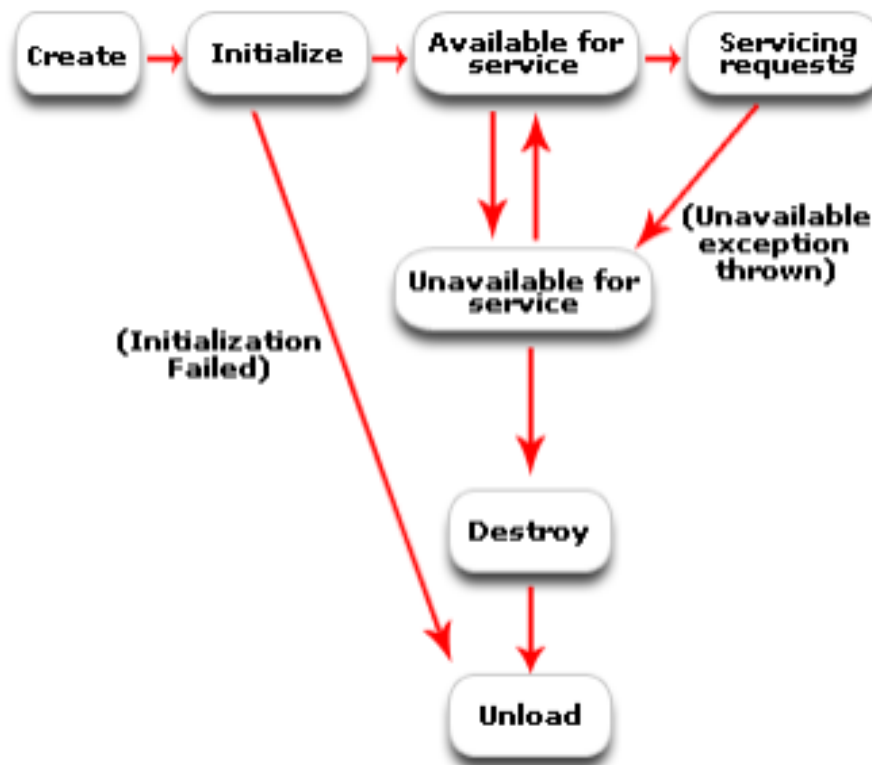
Podšavanja se čuvaju u folderu projekta Configuration Files.

Kada je aplikacij akreirana, sastojaće se od nekoliko datoteka, a celokupna sruktura projekta će biti komprimovana u jedinstveneu datoteku sa ekstenzijom .war (web archive). Datoteka se čuva u folderu MojeKnjige/build/dist (pod Files) nakon prevođenja.

## ŽIVOTNI CIKLUS SERVLETA

*Neophodno je demonstrirati korake životnog ciklusa servleta.*

Životni ciklus servleta može biti prikazan sledećom slikom.



Slika 2.5 Životni ciklus servleta

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

- **Učitavanje:** Servletski kontejner učitava servlet za vreme pokretanja ili kada se pošalje prvi zahtev. Nakon učitavanja servleta, kontejner kreira instancu servleta;
- **Inicijalizacija:** Nakon kreiranja instance, servletski kontejner poziva `init()` metodu i prosleđuje inicijalne parametre servleta istoj. Metoda `init()` mora biti pozvana od strane kontejnera pre nego što se omogući servletu da obrađuje zahteve. Inicijalizovani parametri traju do uništenja (`destroy`) servleta. Metoda `init()` se poziva samo jednom u toku životnog ciklusa servleta. Servlet će biti sposoban za rad ako je uspešno učitao. Ukoliko nije uspešno učitao servletski kontejner "prazni" (`unload`) servlet.
- **Obrada zahteva:** Nakon uspešnog završetka procesa inicijalizacije, servlet je spreman za rad. Servlet kreira posebne niti za svaki zahtev (`request`). Servletski kontejner poziva `service()` metodu za obradu zahteva. Metoda `service()` određuje vrstu zahteva i poziva odgovarajuću metodu (`doGet()` ili `doPost()`) za obradu zahteva i slanje odgovora klijentu;
- **Uništavanje servleta:** Ukoliko servlet duže vreme nije potreban za obradu zahteva, kontejner poziva metodu `destroy()`. Kao i `init()` metoda i ova se metoda poziva samo jednom u toku životnog ciklusa servleta.

## ZADATAK 2

### *Kreiranje i postavljanje vlastitog servleta*

Po uzoru na prethodni zadatak:

- kreirajte i
- postavite vlastiti servlet



## ▼ Poglavlje 3

# Tokovi podataka

## TOK PODATAKA VEB PREGLEDAČ - SERVLET

*Kontejner servleta kontroliše da li je servlet učitán ,a zatim se izvršavaju `init ()`, `service ()` i `destroy ()` metode.*

U narednom izlaganju, vezanom za problematiku servleta, neophodno je obratiti pažnju na tokove podataka, konkretno na tok podataka između veb pregledača i kreiranog servleta. Prvo će biti govora o potencijalnom sadržaju veb stranice. Veb stranica može da integriše HTML formu, link ili JavaScript kod koji može da pošalje HTTP zahtev ka veb serveru. Kada prvi zahtev korisnika dođe do servleta, u posmatranom slučaju do `FindBooksServletservleta`, kontejner proverava, u prvom koraku, da li je ovaj servlet aktivan i da li se izvršava. Ukoliko servlet nije aktivan kontejner ga učitava i, nakon toga, kreira njegovu instancu, a neposredno zatim izvršava metodu `init()` servleta.

Kontejner, ubrzo nakon toga, izvršava `service()` metodu superklase servleta, koja preusmerava zahtev `doGet()`, `doPost()` i / ili sličnu `doXXX()` metodu, prosleđujući kao argumente `HttpServletRequest` i `HttpServletResponse`. Nakon što se očita parametar , on se obrađuje u sloju poslovne logike koji se može implementirati pomoću POJO objekta ili pomoću EJB - a, dolazi do vraćanja rezultata klijentu dobijanjem reference na `PrintWriter` objekat (videti priloženi kod OU2 / Kreiranje servleta - dopunska razmatranja) . Ukoliko rezultati nisu tekstualni, neophodno će bitiprimeniti klasu `OutputStream` umesto klase `PrintWriter`. Potrebno je definisati itip sadržaja (MIME tip) izvršavanjem `setContentType ()` metoda.

Kao primer moguće je navesti sledeće - ukoliko se šalje objekat koji sadrži PDF dokument i ukoliko je neophodno dati dozvolu da veb pregledač otvori aplikaciju koja je podrazumevana za čitanje PDF dokumenata, potrebno je izvršiti sledeći poziv: `response.setContentType("application/pdf")`.

Kontejner servleta kontroliše da li je servlet učitán, a zatim se izvršavaju `init ()`, `service ()` i `destroy ()` metode. Metoda `destroy ()` se izvršava kada administrator servera odluči da ukloni servlet, da isključi server ili da oslobodi memoriju servera.

## PRIMER 4 - DOGET METODA

*Sledi primer `doGet()` metode servleta*

Ukoliko HTTP klijent šalje određene podatke do servleta, oni mogu da se preuzmu primenom `getParameter ()` metode nad `HttpServletRequest` objektom. Sledećim listingom je

omogućeno učitavanje knjige po nazivu zadatom u polju booktitle, uz dopunsku informaciju o ceni.

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
    // TODO Auto-generated method stub
    String title = request.getParameter("booktitle");
    PrintWriter out = response.getWriter();
    response.setContentType("text/html");
    out.println("<html><body>");
    out.println("<h2>Knjiga " + title + " ima cenu 2000 dinara");
    out.println("</html></body>");
}
```

## HTTP GET I POST ZAHTEVI

*Get i Post su njačešće korišćene meode za razmenu podataka na vebu.*

U ovom delu lekcije će akcenat biti na analizi i demonstraciji primene dve najčešće korišćene metode za razmenu podataka na vebu. **Ono što je posebno potrebno napomenuti da se za razmenu podataka na vebu najčešće koriste, pored pomenutih metoda Get i Post, i servleti.** Upravo će u ovom pravcu i teći izlaganje u ovom delu lekcije.

Ukoliko se ne navede eksplicitno metoda, na osnovu inicijalnih podešavanja podrazumevana metoda je Get. Upravo je primer ove metode pokazan kodom datoteke **indeks.xml** čiji je poziv bio smešten u tag `<form>` kao što je pokazano sledećim izolovanim delom koda:

```
<form action=http://www.MyBooks.com/servlet/FindBooksServlet method=Get>
```

**Koristeći Get metod, web pregledač dodaje vrednosti unete u formi na kraj URL adrese, nakon znaka pitanja. Ovo je moguće uraditi i ručno u web pregledaču, unoseći isti taj string u polje za unos veb adrese.**

Primer: Ukoliko korisnik unese reč Java8 kao naziv knjige, URL adresa može da bude sledećeg oblika:

`http://www.myboks.com:?booktitle=Java8 .`

Ukoliko forma ili skript u upitu koristi više vrednosti, kriterijuma pretrage u ovom slučaju, iza znaka pitanja, u okviru adrese, navode se ovi kriterijumi, u formi ključ / vrednost, razdvojeni znakom `&`, na sledeći način:

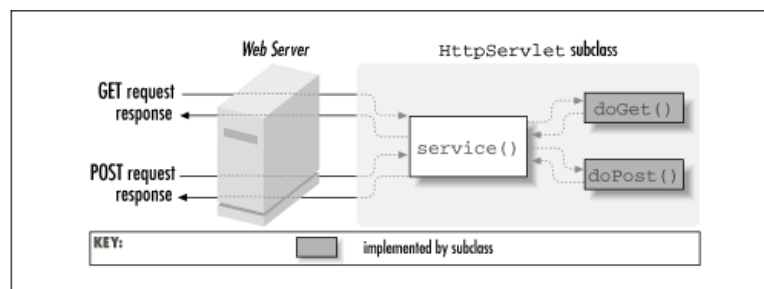
`http://www.myboks.com:?booktitle=Java8&authot=Fain .`

Upotreba Get metode je veoma jednostavna. Ona kopira ili označava URL adresu koja sadrži parametre.

Primenom Get metode podaci nisu zaštićeni, vidljivi su i prikazani u tekstualnom obliku.

Metoda Post ima, ipak, malo ozbiljniji zadatak. Ovom metodom je moguće slanje binarnih (na primer slike i muzički fajlovi) i tekstualnih podataka do servera. Da bi servlet obradio Post zahtev, neophodno je da redefiniše metodu `doPost()`. U praksi, a to je važno napomenuti u ovom delu razmatranja, Get metoda se koriste za čitanje podataka, dok se Post metoda koristi za slanje podataka ka serveru.

Forme za prijavljivanje ne bi trebalo da budu realizovane pomoću Get metode da bi se izbeglo prikazivanje korisničkog imena i lozinke u URL adresi.



Slika 3.1 Get i Post zahtevi - ilustracija

## ZADATAK 3

### *Get i Post implemnatacija*

Pokušajte sami da implementirate Get i Post zahteve - dozvoljeno je korišćenje Interneta.

## ▼ Poglavlje 4

# Servleti i sesije

## PRAĆENJE SESIJA

*Sesija je logički zadatak, koji korisnik pokušava da izvrši pristupanjem web prezentaciji.*

Sesije su veoma bitna stavka analize i diskusije u polju razvoja veb aplikacija. Neophodno je razumeti šta sesija predstavlja, kako se koristi i koja joj je namena. Tako će, u ovom izlaganju, korak - po - korak i teći diskusija.

Prvo je neophodno primetiti da Http predstavlja protokol bez stanja. Ukoliko korisnik očita jednu web stranicu sa određenim sadržajem, u konkretnom i posmatranom slučaju listom knjiga na osnovu `FindBooksServlet` servleta koji se izvršava na strani servera, a zatim pristupi drugoj veb stranici, druga veb stranica ne može da ima saznanja o tome šta je prikazano ili selektovano na prvoj. U ovom pravcu je neophodno tražiti mehanizme koji će omogućiti čuvanje navedenih podataka. Da bi navedeni podaci mogli da budu sačuvani na više web stranica, neophodno je implementirati mehanizam pozna pod nazivom praćenje sesija.

*Sesija je logički zadatak, koji korisnik pokušava da izvrši pristupanjem web prezentaciji.*

Primer sesije je lako navesti. Neka to bude, iz konkretnog i aktuelnog primera sledeće:

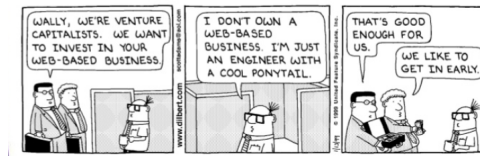
- Proces kupovine knjige može da uključi nekoliko koraka;
- Koraci mogu biti: izbor knjige, izbor načina plaćanja , unos podataka o isporuci, i tako dalje;
- Kombinacija ovih koraka predstavlja primer sesije;
- Kada se završi naručivanje sesija se završava.

Informacije o sesiji mogu biti čuvane na dva načina. Konkretno, *informacije o sesiji se mogu čuvati ili na strani klijenta ili na strani servera*. Na strani klijenta se korisnički podaci o sesiji beleže smeštanjem u „kolačićima“ ili modifikovanjem URL adrese - ta informacija se šalje klijentu ili serveru kao deo URL adrese.

Sa druge strane, informacije o sesijama mogu da se čuvaju i na serverskoj strani primenom aplikacionog programskog interfejsa (API) za praćenje sesija. Ovaj interfejs sadrži veliki broj metoda definisanih u `HttpSession` interfejsu. U ovom slučaju, podaci o sesijama se isključivo čuvaju na serveru, a klijent dobija samo identifikacioni broj (ID) sesije da bi bila identifikovana serija HTTP zahteva koje šalje isti korisnik. U nastavku izlaganja je neophodno još istaći i sledeće. Da bi bilo moguće dobiti referencu na postojeću sesiju ili kreirati nove sesije na serveru , neophodno je izvršiti metodu `getSession (true)` kojoj se, u tu svrhu, obraća `HttpServletRequest` objekat.

• Why session tracking?

- When clients at on-line store add item to their shopping cart, how does server know what's already in cart?
- When clients decide to proceed to checkout, how can server determine which previously created cart is theirs?



Slika 4.1 Praćenje sesija - ilustracija

## PRAĆENJE SESIJA - PRIMENA "KOLAČIĆA"

*Kolačić je podatak koji servlet može da šalje web klijentu da bi bio snimljen u datoteku na računaru korisnika.*

U ovom delu izlaganja je neophodno predložiti koncept "kolačića" (eng. **cookies**), kako se koriste i koja ima je namena. Kolačić je podatak koji servlet može da šalje web klijentu da bi bio snimljen u datoteku na računaru korisnika. Pri svakom narednom zahtevu koji šalje korišćeni klijent, web pregledač proverava lokalne aktivne „kolačiće“ i šalje ih server, povezujući , na jednostavan način , zahtev sa odgovarajućom sesijom. „Kolačići“ su perzistentni ali korisnik može da onemogući njihovo korišćenje izborom odgovarajućeg podešavanja u vlastitom izabranom web pregledaču.

Posebno, u ovom delu izlaganja, bilo bi značajno pokazati način na koji servlet može da šalje klijentu objekat tipa **Cookie**. Upravo je navedeno realizovano programskim kodom koji je prikazan sledećim listingom.

```
Cookie myCookie= new Cookie ( „bookName“ .
„Java Programming –hour trainer „);

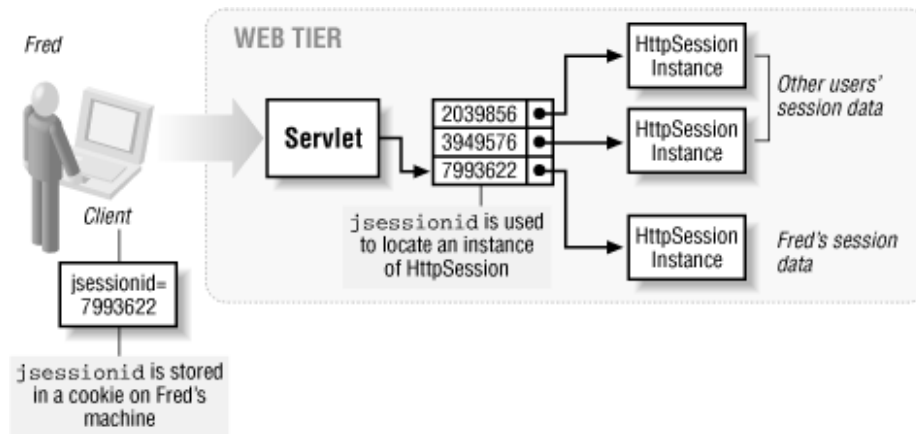
// Definisane „životnog veka“ jednog „kolačića“ tako da bude 24časa
myCookie. setMaxAge (60*60*24);
response.addCookie (myCookie);

// Ovako servlet može da primi klijentove kolačiće koji dolaze sa HttpServlet
Request:
Cookie () cookies= request.getCookies ();
If (cookies !=null) (
// Očitavanje „kolačića“ (par naziv vrednost)
For ( i=0; < cookies.length; i++) (
Cookie currentCookie= cookie (i);
String name= currentCookie.getName ();
String value= currentCookie.getValue ();
}
}
```

Važno je napomenuti da iako je moguće sačuvati veći broj kolačića na klijentu, na načinurađen kao u priloženom prethodnom kodu, ne predstavlja dobru ideju i praksu

slanje podataka aplikacije dvosmerno kroz mrežu. Zbog toga se, u velikom broju slučajeva, podaci koji su u direktnoj vezi sa sesijom čuvaju u HttpSession objektu, o čemu će biti više govora u jednom od narednih izlaganja.

Sledeća slika ilustruje primenu kolačića tokom procesa praćenja sesija.



Slika 4.2 Čuvanje kolačića na klijent računaru

## MODIFIKACIJA URL ADRESE

*Modifikovanjem URL adrese je omogućeno praćenje sesija kada su kolačići onemogućeni.*

Ukoliko je korisnik u vlastitom veb pregledaču onemogućio korišćenje kolačića, praćenje sesije je omogućeno drugim mehanizmom koji podrazumeva modifikaciju URL adrese. U ovom slučaju vrši se dodavanje tokena ili identifikatora na URL adresu narednog servleta ili resursa. Moguće je poslati par u formi naziv / vrednost na sledeći način:

*url?name1=value1&name2=value2&?? .*

Iz ovog zahteva je moguće primetiti da je par naziv / vrednost razdvojen primenom znaka za relaciju jednakosti, a različiti parovi su razdvojeni znakom ampersand(&). Kada se desi klik na odgovarajući hiperlink, par naziv / vrednost će biti prosleđen ka serveru. Iz servleta je moguće koristiti metodu `getParameter()` za prihvatanje vrednosti parametra.

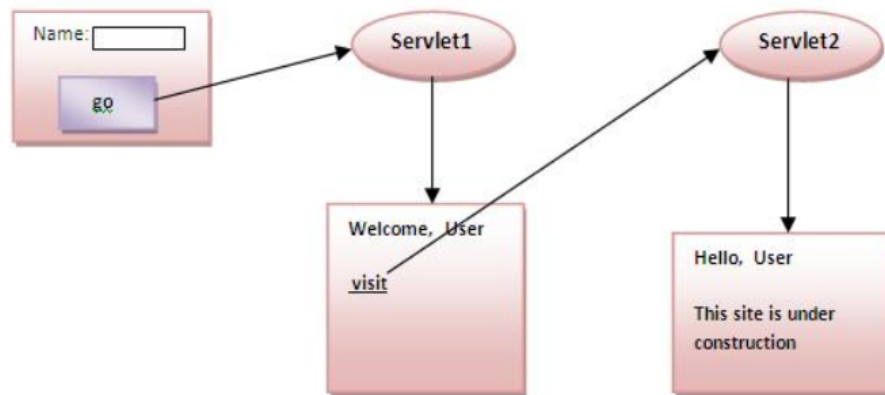
Ovakav pristup ima vlastite prednosti i nedostatke koje je neophodno istaći u narednom izlaganju. Prednosti modifikacije URL adrese, u procesu praćenja sesija, su sledeće:

- Ne zavise od tipa veb pregledača - radiće i kada su kolačići onemogućeni;
- Ne zahteva postojanje dodatne forme za potvrđivanje na svakoj stranici.

Ovaj pristup ima sledeće nedostatke:

- Funkcioniše isključivo sa linkovima;
- Prosleđuje samo tekstualne informacije.

Sledećom slikom je moguće ilustrovati mehanizam modifikacije URL adrese za proces praćenja sesija.



Slika 4.3 Modifikacija URL adrese za proces praćenja sesija.

## PRIMER 5 - MODIFIKACIJA URL ADRESE - STUDIJA SLUČAJA

*Kreira se primer za ilustraciju modifikacije URL adrese.*

Za ilustraciju primene koncepta modifikacije URL adrese za praćenje sesija, moguće je razviti sledeći primer:

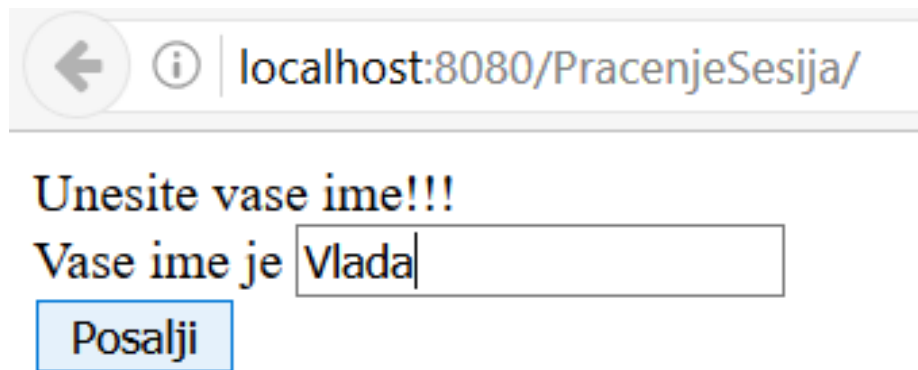
1. Kreira se veb aplikacija koja sadrži datoteke `index.html`, `FirstServlet.java` i `SecondServlet.java`.
2. `index.html` definiše polje za unos teksta i dugme. Klikom na dugme uneti podatak se prosleđuje u servlet `FirstServlet`;
3. Na ovom servletu se nalazi poruka dobrodošlice za osobu unetu na početnoj formi i link za prelazak na servlet `SecondServlet`.
4. Klikom na navedeni link vrši se prosleđivanje podatka, unetog u polaznoj formi, sa servleta `FirstServlet` na servlet `SecondServlet`.
5. Takođe, klikom na ovaj link dolazi do modifikacije URL adrese servleta `SecondServlet` na način definisan u kodu klase `FirstServlet`.

U prvom koraku biće predstavljen listing datoteke `index.html`, koja predstavlja naslovnu stranu ove veb aplikacije.

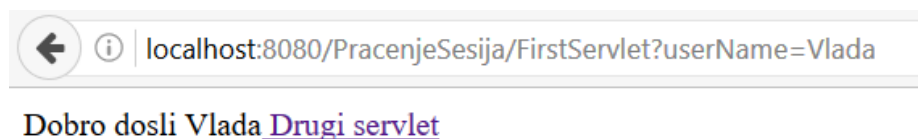
```
<html>
  <head>
    <title>Unos imena</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div>Unesite vase ime!!!</div>
    <form action="FirstServlet" method=Get>
Vase ime je <input type="text" name="userName"/><br/>
```

```
<input type="submit" value="Posalji"/>
</form>
</body>
</html>
```

Unosom željenog podatka, odnosno vašeg imena, i klikom na dugme "Posalji" (slika 4) vrši se prosleđivanje unetog podatka u servlet pod nazivom FirstServlet (slika 5). Ovaj servlet je preuzeo navedeni podatak instrukcijom: `String n=request.getParameter("userName");`



Slika 4.4 Stranica index.html



Slika 4.5 Servlet FirstServlet je preuzeo prosleđeni podatak

## PRIMER 5 - MODIFIKACIJA URL ADRESE - STUDIJA SLUČAJA - NASTAVAK

*U nastavku je neophodno kreirati i priložiti klase servleta.*

Da bi bila moguća funkcionalnost koja je prikazana prethodnom slikom, bilo je neophodno kreirati klasu FirstServlet.java. Navedena klasa je priložena sledećim listingom.

```
/**
 *
 * @author Vladimir Milicevic
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet(urlPatterns = {"/FirstServlet"})
public class FirstServlet extends HttpServlet {
```



```
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response){
    try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

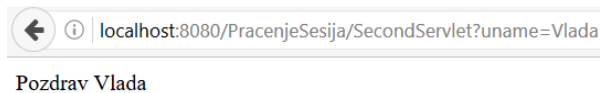
        String n=request.getParameter("userName");
        out.print("Dobro dosli " + n);

        //prosleđuje username u string upita
        out.print("<a href='SecondServlet?uname="+ n +" '> Drugi servlet</a>");

        out.close();

        }catch(Exception e){System.out.println(e);}
    }
}
```

U ovoj klasi je definisan i kreiran link pod nazivom "Drugi servlet". Klikom na ovaj link prosleđuje se uneto ime u servlet SecondServlet - posebno pogledati instrukciju servleta FirstServlet: `out.print("<a href= 'SecondServlet?uname=" + n + "'> Drugi servlet</a>");`. Servlet SecondServlet koristi izraz: `String n=request.getParameter("uname");` za preuzimanje unetog imena iz stringa upita. Sledećom slikom je prikazana stranica koja odgovara servletu SecondServlet.



Slika 4.6 Servlet SecondServlet je preuzeo prosleđeni podatak

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

Konačno, navedena funkcionalnost, koja je dovedena u vezu sa servletom SecondServlet, ugrađena je u programski kod koji je priložen sledećim listingom.

```
/**
 *
 * @author Vladimir Milicevic
 */

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.annotation.WebServlet;

@WebServlet(urlPatterns = {"/SecondServlet"})
public class SecondServlet extends HttpServlet {
```

```
@Override
public void doGet(HttpServletRequest request, HttpServletResponse response) {
    try{

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        //uzima vrednost iz stringa upita
        String n=request.getParameter("uname");
        out.print("Pozdrav "+n);

        out.close();

        }catch(Exception e){System.out.println(e);}
    }
}
```

## SESIJA NA STRANI SERVERA

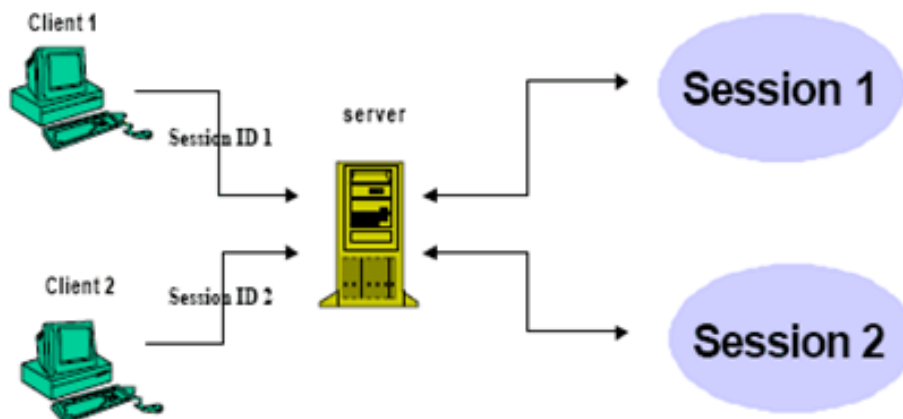
*Servlet može da sačuva bilo koji Serializable objekat.*

U narednom izlaganju, neophodno je pokazati kako se podaci koji se odnose na korisničku sesiju mogu čuvati u objektu tipa HttpSession u okviru kontejnera servleta. Za svakog pojedinačnog klijenta, kontejner servleta kreira po jedan takav objekat. **Servlet može da sačuva bilo koji Serializable objekat.** Sledećim kodom je pokazano kreiranje objekta sesije ili pronalaženje prethodno kreiranog objekta:

```
HttpSession mySession = request.getSession (true) ;
```

Poziv metoda **getSession(true)** pronalazi objekat sesije klijenta ili kreira novi objekat ukoliko nije prethodno kreiran. Posebno je bitno naglasiti da poziv metode getSession(true) trebalo bi da se koristi prilikom prvog zahteva da se startuje poslovni proces koji omogućava uspostavljanje nove sesije. U tom slučaju se dešava da aplikacioni server generiše jedinstveni ID sesije i prosleđuje ga dalje korisničkom web pregledaču pomoću posebnog kolačića **JSESSIONID** ili **modifikovanjem URL adrese**. Kada pregledač pošalje HTTP zahtev serveru, ID sesije se locira u zaglavlju zahteva, tako da kontejner servleta može da pronađe odgovarajući objekat sesije.

U suprotnom slučaju, kada se izvršava **getSession(false)**, dešava se pronalaženje objekta sesije uz pretpostavku da je objekat sesije kreiran u njoj u prethodnim operacijama. Ukoliko se tokom ovog poziva vrati vrednost **null**, to praktično znači da je objekat sesije uklonjen ili više nije aktivan.



Slika 4.7 Sesija na strani servera

Za demonstraciju i lakše razumevanje ovog problema biće proširena veb aplikacija koja je do sada služila kao primer, a naziva se **MojeKnjige**.

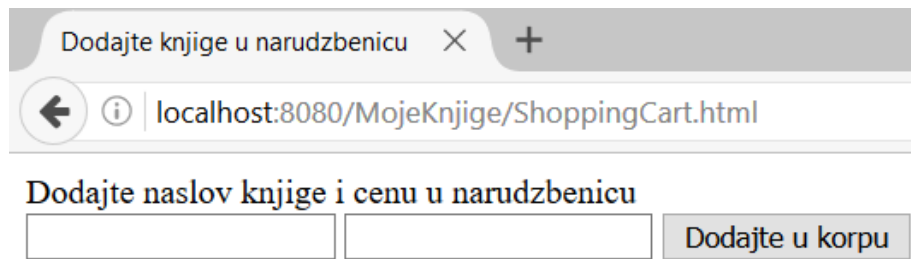
**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

## PRIMER 6- SESIJA NA STRANI SERVERA - DEMONSTRACIJA

*Proširuje se postojeći primer za ilustrovanje problematike sesija na strani servera.*

Za početak biće kreirana nova HTML stranica čiji je zadatak simuliranje narudžbenice. Stranica prima podatke poput naziva knjige i cene. Sa dodavanjem nove knjige servlet će vratiti novu HTML stranicu koja prikazuje aktuelna narudžbenica sa opcijom unosa dodatnih knjiga. Ova HTML stranica će biti realizovana pod nazivom **ShoppingCart.html** i biće čuvana u podfolderu projekta Web Pages, sa kodom koji je priložen sledećim listingom.

```
<html>
  <head>
    <title>Dodajte knjige u narudzbenicu</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div>Dodajte naslov knjige i cenu u narudzbenicu</div>
    <form action=shoppingcart method=Get>
      <input type = Text name = booktitle>
      <input type = Text name = price>
      <input type = Submit value = "Dodajte u narudzbenicu">
    </form>
  </body>
</html>
```



Slika 4.8 Stranica ShoppingCart.html

Ova HTML datoteka je deo iste web aplikacije kao servlet. Iz navedenog razloga nije potrebno navoditi punu URL adresu servleta. Sve što je neophodno jeste šablon `/ShoppingCart` i servlet obeležen anotacijom `@WebServlet("/ShoppingCart")`.

U nastavku je potrebno još dodati i sledeće - svaka stavka narudžbenice je određena objektom klase `Book` čiji je kod priložen sledećim listingom.

```
/**
 *
 * @author Vladimir Milicevic
 */
import java.io.Serializable;

public class Book implements Serializable {

    String title;
    double price;
}
```

Namera je da se klasa `Book` čuva u `HttpSession` objektu i zbog toga nasleđuje interfejs `Serializable`.

## SESIJA NA STRANI SERVERA - DEMONSTRACIJA - NASTAVAK

### *Sledi implementacija klase servleta.*

Nakon definisane HTML stranice, kao i klase `Book`, neophodno je implementirati odgovarajuću servlet klasu koja će, u konkretnom slučaju, biti nazvana `ShoppingCartServlet.java`. U `NetBeans IDE` razvojnom okruženju, desnim klikom na podfolder `Source Packages` i izborom opcije `New / Servlet` kreira se klasa sa navedenim imenom čiji je kod priložen sledećim listingom.

```
/**
 *
 * @author Vladimir Milicevic
 */
```

```
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

/**
 * Servlet implementacija narudzbenice sa sesijom
 */
@WebServlet("/shoppingcart")
public class ShoppingCartServlet extends HttpServlet {

    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException,
        IOException {

        Cookie[] cookies = request.getCookies();

        if (cookies != null){

            for (int i=0; i < cookies.length; i++){
                Cookie currentCookie = cookies[i];
                String name = currentCookie.getName();
                String value = currentCookie.getValue();

                System.out.println("Primio kolacic" + name + "=" + value);
            }
        }

        // Prima ili kreira objekat sesije
        HttpSession session = request.getSession(true);

        // Pokusava da prihvati narudzbenicu
        ArrayList<Book> myShoppingCart=(ArrayList<Book>)
session.getAttribute("shoppingCart");

        if (myShoppingCart == null){
            // Prvi poziv - kreiranje instance narudzbenice
            myShoppingCart = new ArrayList<>();
        }

        // kreiranje objekta knjige na osnovu unetih parametara
        Book selectedBook = new Book();
        selectedBook.title=request.getParameter("booktitle");
        selectedBook.price = Double.parseDouble(request.getParameter("price"));

        // Dodavanje knjige u narudzbenicu
```

```
myShoppingCart.add(selectedBook);

// Vracanje narudzbenice u objekat sesije
session.setAttribute("shoppingCart", myShoppingCart);

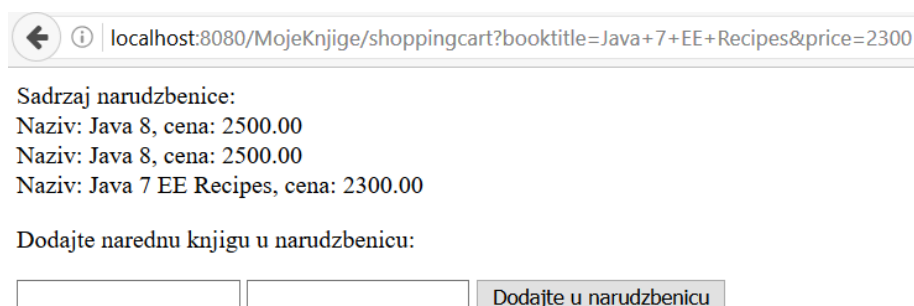
// Priprema veb stranice i njeno prosledjivanje u veb pregledac
PrintWriter out = response.getWriter();

// Dodavanje sadrzaja narudzbenice na veb stranicu
out.println("<body>Sadrzaj narudzbenice:");
myShoppingCart.forEach(book ->
    out.printf("<br>Naziv: %s, cena: %.2f", book.title, book.price)
);

//dodavanje HTML forme na veb stranicu
out.println("<p>Dodajte narednu knjigu u narudzbenicu:");
out.println("<form action=shoppingcart method=Get>");
out.println("<input type=Text name=booktitle>");
out.println("<input type=Text name=price>");
out.println("<input type=Submit value='Add to shopping cart'>");
out.println("</form>");
out.println("</body>");
}
}
```

Kreirana klasa realizuje URL mapiranje `/ShoppingCart`, a to predstavlja zahtev upućen iz veb pregledača koji pokušava da pronađe objekat na server strani koji je povezan sa nazivom `shoppingcart`. Metoda `doGet()` izlistava sadržaje kolačića iz aktuelnog veb pregledača u sistemskoj konzoli. Na ova način se vrši provera da li je vrednost kolačića `JSESSIONID` za svaki zahtev koji je prosleđen iz posmatranog veb pregledača.

Prilikom prvog pristupa servleta narudžbenici, iz objekta sesije, servlet neće pronaći narudžbenicu. Tada će biti kreiran `ArrayList<>` objekat. Iz HTML forme se učitavaju podaci sa nazivom knjige i cenom i kreira se nov objekat klase `Book` koji se dodaje u kreiranu listu i postavlja objekat `HttpSession` gde će biti čuvan. Petljom i lambda izrazom, sadržaj liste se šalje HTML kodom na veb stranicu, zajedno sa postojećom HTML formom, tako da korisnik može da nastavi unos u narudžbenicu (slika 9).



← ⓘ | localhost:8080/MojeKnjige/shoppingcart?booktitle=Java+7+EE+Recipes&price=2300

Sadrzaj narudzbenice:

Naziv: Java 8, cena: 2500.00

Naziv: Java 8, cena: 2500.00

Naziv: Java 7 EE Recipes, cena: 2300.00

Dodajte narednu knjigu u narudzbenicu:

Slika 4.9 Kreiranje narudžbenice

## SESIJA NA STRANI SERVERA - DEMONSTRACIJA - DODATNA RAZMATRANJA

*Moguće je dodati još neke korisne funkcionalnosti.*

Ako se pažljivo pogleda priloženi kod za klasu servleta ShoppingCart.java, moguće je primetiti da on ne implementira kod kojim se zatvara sesija. Ovu funkcionalno je moguće veoma lako realizovati ukoliko se na veb stranici kreira **Place Order** taster, a to implicira i kreiranje odgovarajuće **placeorder()** metode u servlet. Metoda može da zatvori sesiju ugradnjom i izvršavanjem sledećeg koda:

```
session.invalidate();
```

U nastavku će biti pokazan veoma jednostavan način dodavanja navedenog tastera na stranicu koja odgovara **ShoppingCartServlet** servletu. Nophodno je u formu dodati taster sledećim kodom:

```
<input type = Submit name = placeorder value = "Place Order">
```

U metodi **doGet()** servleta, biće izvršena provera da li je korisnik kliknuo na ovaj taster. Ako jeste, sesija se poništava. Ovo se realizuje tako što se u pomenutu metodu ugrađuje sledeći kod:

```
if (request.getParameter("placeorder") != null){  
    session.invalidate();  
}
```

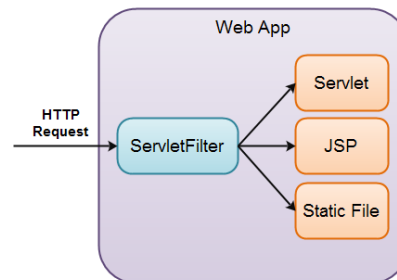
Ako se ne izvrši eksplicitno uklanjanje sesije, aplikacioni server će je automatski ukloniti nakon isteka izvesnog perioda (timeout). Ova vrednost može biti definisana programskim kodom izvršavanjem metode **setMaxInactiveInterval()** HttpSession objekta. Zatvaranjem veb pregledača, kolačić sesije se uklanja i sesija se gasi.

## KLASE FILTERI

*Moguće je promeniti način obrade zahteva i odgovora čak i kada je servlet isporučen na server.*

U nastavku, obrade i demonstracije rada sa servletima, neophodno je pokazati mehanizam pomoću kojeg može da se menja oblik obrade zahteva i odgovora, čak i kada je servlet isporučen na server, bez potrebe za modifikovanjem koda servleta. Mehanizam koji omogućava navedenu funkcionalnost naziva se filter. **Filter predstavlja predstavljaju Java klasu koja ima mogućnost podešavanja da obrađuje HTTP zahteve pre nego što se proslede do servleta ili kada servlet treba da ih vrati kao odgovor klijentu.** Filteri mogu da se

promenjuju u različitim scenarijima: provera autentičnosti, logovanje, enkripcija, kompresija podataka, konverzacija slika i sl. **Filteri mogu i da blokiraju objekte zahteva i odgovora, tako da se ne prosleđuju dalje.** Posebno je neophodno obraditi način kreiranja klase filtera. Da bi filter bio kreiran, **potrebno je napisati klasu koja implementira Filter interfejs označen anotacijom @WebFilter.** Posebno je vazno istaći tri metode ovog interfejsa: **doFilter ()**, **init ()** i **destroy ()**. Filtere je moguće i lančano povezati implementiranjem FilterChain interfejsa.



Slika 4.10 0 0 0 0 Primena servlet filtera

Klasa filtera koja se koristi za FindBooksServlet i ShoppingCartServlet.

Metod destroy se izvršava jednom, pre nego što kontejner ukloni filter, ukoliko je filter zauzeo određene resurse, kao što su konekcije sa DBMS sistemom, koje možemo osloboditi izvršavanjem destroy () metoda.

Metod init () se poziva na objekat filtera samo jednom u toku instanciranja. Kontejner servleta prosleđuje init () metodu instancu FilterConfig objekta, koja omogućava pristup sadržaju servleta i inicijalizacijom parametrima ukoliko su oni specifikirani pomoću @WebFilter anotacije

```

import java.io.IOException;
import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.annotation.WebFilter;

/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */

/**
 *
 * @author Vladimir Milicevic
 */

@WebFilter(servletNames = {"/FindBookServlet", "/ShoppingCartServlet"})
public class MyAuthenticationFilter implements Filter{
    FilterConfig config;

```



```
@Override
public void doFilter (ServletRequest request, ServletResponse response,
FilterChain filter)
    throws IOException, ServletException{

    //provera autentičnosti korisnika
    //Primena sledećeg filtera ukoliko je neophodan
    filter.doFilter(request, response);
}

@Override
public void init(FilterConfig filterConfig) throws ServletException{
    this.config = filterConfig;
}

@Override
public void destroy(){
    //ova metoda oslobađa zauzete sistemske resurse
}
}
```

## ASINHRONI SERVLETI

*Za svaki zahtev korisnika servleti kreiraju posebnu nit.*

Pre zaključivanja analize i diskusije koja se odnosi na rad sa servletima, neophodno je napomenuti da **za svaki zahtev korisnika servleti kreiraju posebnu nit**, a dobro je poznato da svaka nit zauzima dodatne sistemske resurse. Može doći do situacije da nakon prisustva određenog broja konkurentnih zahteva, server prestane da šalje odgovore. Ako bi, u posmatranom primeru, veliki broj korisnika (stotine ili hiljade) istovremeno pristupio servletu **FindBooksServlet**. Neka je potrebno određeno vreme (par sekundi) da se izvrši pretraživanje baze podataka po zahtevu. Tokom ovog vremena, dok se čeka rezultat izvršavanja upita nad bazom podataka (koja se izvršava na drugom serveru), kontejner se ne menja i zaključava konkretnu nit.

Asinhroni servleti služe za minimiziranje vremena zaključavanja niti višestrukim korišćenjem niti u kontejneru servleta. Ukoliko korisnik A pošalje zahtev koji se izvršava par sekundi na DBMS serveru, njegova nit u kontejneru servleta se prosleđuje zahtevu korisnika B, a kada dođe odgovor DBMS sistema za korisnika A, kontejner alocira nit tako da se rezultat prosleđuje korisniku A.

Java EE 7 uključuje Servlets 3.1 specifikaciju koja podržava asinhronu obradu. U doGet () ili doPost () metodu moguće je instancirati **AsyncContext** objekat, koji kreira asinhronu radnu nit i ne zaključava klijentsku nit u toku realizovanja klijentskog zahteva i dobijanja odgovora.

```
/**
 *
 * @author Vladimir Milicevic
 */
```

```
import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.AsyncContext;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet(urlPatterns = {"/booksasync"}, asyncSupported=true)
public class FindBooksAsyncServlet extends HttpServlet {
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException{

        // Ne slati odgovor kada se završi metoda doGet()
        AsyncContext aContext = request.startAsync();

        //Obezbeđuje Runnable implementaciju start() metodu
        aContext.start(() ->{

            // Ovde se postavlja blokirajuća operacija
            try{
                String title = aContext.getRequest().getParameter("booktitle");

                PrintWriter out;
                try {
                    Thread.currentThread().sleep(3000); // Simulacija procesa od 3
sekunde

                    HttpServletResponse resp = (HttpServletResponse)
aContext.getResponse();
                    out = resp.getWriter();
                    out.println("Pozdrav, ovo je Async FindBooks");
                } catch (IOException e) {
                    e.printStackTrace();
                }
            }catch( InterruptedException e){
                e.printStackTrace();
            }finally{
                aContext.complete(); // zatvori response objekat
            }

        });
    }
}
```

## ZADATAK 4

### *Praćenje sesija - primena "kolačića"*

Pokušajte samostalno da implementirate praćenje sesija primenom "kolačića".

## ▼ Poglavlje 5

# GlassFish server

## OSNOVNA RAZMATRANJA O GLASSFISH SERVERU

*GlassFish je open - source aplikativni server.*

Oracle GlassFish aplikacioni server je baziran na Java EE implementaciji i predstavlja prvi aplikacioni server koji u potpunosti podržava Java EE platformu, primenu Java EE veb profila i napisan je i dizajniran kao specifična podrška razvoju Java veb aplikacija. Nove verzije Jave su unazad kompatibilne, a to znači da sve postojeće veb aplikacije mogu bez ikakvih problema da se izvršavaju na najnovijoj platformi. Kreiranjem veb aplikacija primenom Java EE platforme i GlassFish servera insistira se na: pouzdanosti, skalabilnosti, kvalitetnom upravljanju greškama, kao i visokom stepenu performansi koje veb aplikacije pružaju. Podrška se proteže od instalacije do administratorskih funkcija, od nativnih veb server dodataka (plug - in) do naprednih veb servisa, JMS i EJB. Ove karakteristike omogućavaju razvoj različitih delova veb aplikacije na dinamičan način, veoma brzo, sa visokim stepenom ponovne upotrebljivosti i moguće nadogradnje pri čemu je akcenat na zadovoljavanju korisničkih zahteva. Jedan domenski GlassFish server može da upravlja većim brojem instanci koje su simultano pokrenute od strane više softverskih komponentata.

GlassFish server je nabrazi open - source aplikativni server. On pokazuje izuzetne performanse zadržavajući, pritom, jednostavnost upotrebe, brzo pokretanje i pojednostavljeno administriranje.

Takođe, GlassFish server, a to je od posebnog značaja za ovu lekciju, pruža višestruku razvojnu podršku uključujući razvojna okruženja poput Eclipse i NetBeans IDE.

GlassFish daje podršku za omiljene alate brojnih programera: GUI i CLI / (Command Line Interface), Maven, Ant, RESTFul API i brojni drugi. U sinergiji sa GlassFish serverom, Java EE 7, koja sada funkcioniše na JDK 8, obezbeđuje širok spektar naprednih setova alata za: unapređenje razvojne produktivnosti, boljih performansi i održivosti.

## GLASSFISH SERVER I RAZVOJNA OKRUŽENJA

*GlassFish značajno unapređuje proces razvoja kombinovan sa aktuelnim razvojnim okruženjem.*

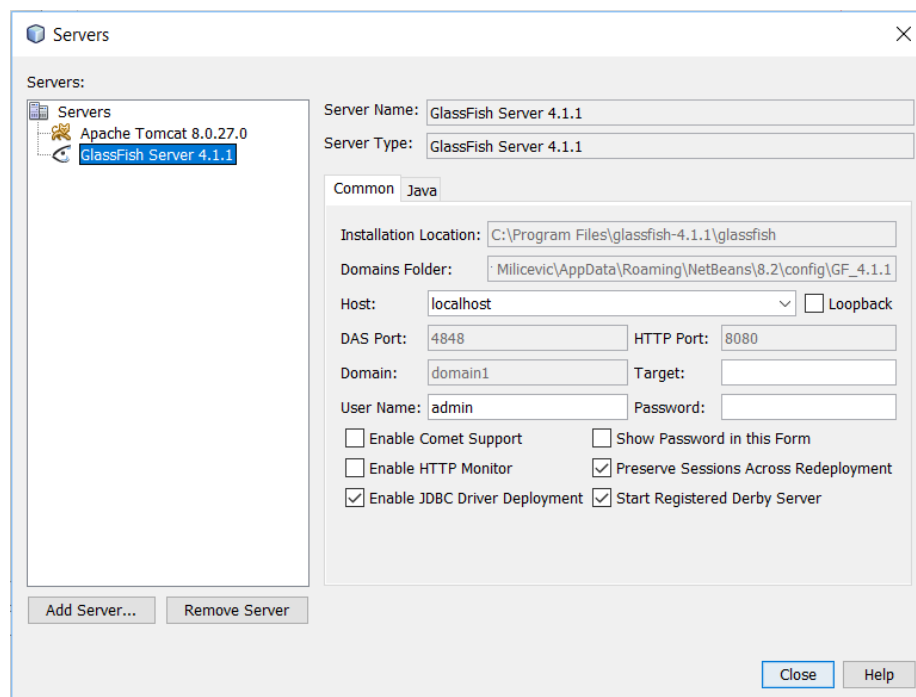
Kada se GlassFish server kombinuje sa savremenim razvojnim okruženjima, poput NetBeans (Slika 1) ili Eclipse, dolazi do značajnog unapređenja iterativnog razvoja veb aplikacija.

Umesto razvoja u šest koraka: editovanje, čuvanje, kompajliranje, pakovanje, angažovanje i re- populacija podataka sesije, ceo proces je redukovano na sledeća tri: editovanje, snimanje i osvežavanje veb pregledača. Primenom GlassFish servera, omogućeno je da podaci sesije, iz HTTP sesije i EJB sa stanjem, budu sačuvani tokom angažovanja aplikacije, eliminišući tako potrebu za ponovnim pozivanjem podataka sesije kada se testira nov kod u aplikaciji. Programeri imaju mogućnost ograničavanja skupa resursa za angažovanu aplikaciju.

Posebno je značajna snažna podrška primeni anotacija, na račun XML konfiguracija, od strane Java EE 7. Na ovaj način je značajno pojednostavljeno pakovanje poslovnih komponenta i konstruisanje više POJO (Plain Old Java Objects) objekata. Na ovaj način je obezbeđeno da se uradi više posla, za kraće vreme, tokom procesa razvoja.

U nastavku je neophodno napomenuti da GlassFish server pruža višezjezičnu podršku i dostupan je u sledećim jezicima: engleski, nemački, francuski, španski, pojednostavljeni kineski, tradicionalni kineski, japanski, korejski i brazilski portugalski jezik.

Sledećom slikom je prikazana integracija GlassFish servera 4.1.1 u razvojnom okruženju NetBeans 8.2.



Slika 5.1 GlassFish server i NetBeans IDE

## GLASSFISH SERVER - OSNOVNE KARAKTERISTIKE

*U ovom delu lekcije, akcenat je na osnovnim elementima servera.*

Na samom početku izlaganja biće prvo reči o GlassFish server kontrolama kao skupu alata kojim je unapređeno upravljanje procesom angažovanja softverskih proizvoda:

- **Monitoring Scripting Client** - omogućava prilagođeno nadgledanje skripti primenom fino - granuliranih testova;

- **Domain Backup and Recovery** - zakazivanje i automatsko čuvanje rezervnih kopija pokrenutog domenskog servera;
- **Performance Tuner** - podešavanje performansi servera;
- **Active Cache for GlassFish** - omogućava robusniju i fleksibilniju primenu servera;
- **Oracle Access Manager Integration** - omogućava pojednostavljeno prijavljivanje aplikacija i servisa;
- **Load Balancer Web Server Plug-in & Installer** - balansirano učitavanje preko komponenta aplikacije i lakše prevazilaženje grešaka u slučaju pojavljivanja grešaka kod komponenta.

**Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.**

GlassFish server je fleksibilan, proširiv i prilagodljiv.

Imajući u vidu poslednje tvrđenje moguće je istaći brojna umapređenja koja donosi primena Oracle GlassFish Servera:

- **Ugrađen API za čvrsto integrisano rešenje sa punom EJB podrškom;**
- **JMX i RESTFul administrativni API za nove, ili integraciju sa postojećim, veb servise;**
- **Razvoj prilagođenih administrativnih alata i njihova dostupnost kroz veb konzolu, alate komandne linije i RESTFul API;**
- **Mogućnost unapređenja korisničkog interfejsa novim funkcionalnostima;**
- **64 - bitna plug - in podrška za bolje performanse i iskorišćenost memorije;**
- **Veoma proširiva platforma.**

GlassFish server veoma kvalitetno funkcioniše u sinergiji sa proizvodima koji pripadaju segmentu Oracle Fusion Middleware Product, kao što su: Oracle Internet Directory, Oracle Virtual Directory, Oracle JRockit JVM, Oracle Coherence, Oracle Web Services Manager i Oracle Access Manager.

## PRIMER 7

### *Kreiranje jednostavne veb aplikacije primenom servleta.*

1. Otvoriti NetBeans IDE razvojno okruženje i kreirati projekat veb aplikacije pod nazivom Vezba1;
2. Projekat sadrži html stranicu sa formom koja simulira trgovinu akcijama;
3. Na osnovu šifre akcije, varijablu simbol, pomoću get metode, preuzima servlet pod nazivom StockServerServlet.java;
4. Servlet dinamički kreira HTML izlaz za prikazivanje primljenog podatka (akcije) zajedno sa slučajno generisanom njenom cenom;
5. Za generisanje cene zadužen je objekat Java klase StockQuoteGenerator.java čiji je zadatak, takođe, da proveriti da li je kod akcije podržan ili ne.

Sledi kod uvodne HTML stranice pod nazivom index.html:

```
<html>
  <head>
    <title>Vezba broj 1</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
  </head>
  <body>
    <div>Vezba broj 1 - Kvote akcija</div>
    <form action="http://localhost:8080/Vezba2/cena" method="get">
      <input type="text" name="simbol" placeholder="Unesite simbol za akciju">
      <input type="submit" value="Preuzmite cenu akcije">
    </form>
  </body>
</html>
```

Servlet klasa je priložene sledećim listingom:

```
package com.vezba1;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/cena")
public class StockServerServlet extends HttpServlet {
    private StockQuoteGenerator stockQuoteGenerator =
        new StockQuoteGenerator();

    @Override
    protected void doGet(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException {

        String simbol = request.getParameter("simbol");
        String cena = stockQuoteGenerator.getCena(simbol);

        PrintWriter out = response.getWriter();

        response.setContentType("text/html");
        out.println("<html><body>");
        out.println("<h3>Trazeni simbol: " + simbol + "</h3>");
        out.println("<h3>Cena: " + cena + "</h3>");
        out.println("</body></html>");
    }
}
```

## PRIMER 7- NASTAVAK

*Aplikacija je zaokružena jednom pomoćnom klasom.*

Za generisanje cene zadužen je objekat Java klase StockQuoteGenerator.java čiji je zadatak, takođe, da proveriti da li je kod akcije podržan ili ne. Sledi kod ove pomoćne Java klase.

```
package com.vezba2;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class StockQuoteGenerator {

    private List<String> akcije = new ArrayList<>();

    private Random randomGenerator = new Random();

    public StockQuoteGenerator() {
        akcije.add("AAPL");
        akcije.add("MSFT");
        akcije.add("YHOO");
        akcije.add("AMZN");
    }

    public String getCena(String simbol) {
        double cena;

        if (akcije.indexOf(simbol.toUpperCase()) != -1) {
            cena = randomGenerator.nextDouble();
            return "" + cena;
        } else {
            return "Simbol akcije nije podrzan";
        }
    }

    public List<String> getSimboli() {
        return akcije;
    }
}
```

## ZADATAK 5

*Probajte sami*

Pokušajte da kreirate servlet aplikaciju:

1. Na početnoj stranici unosite broj indeksa, naziv fakulteta u vaše ime;



2. Na stranici za prikazivanje rezultata dobijate string: "Dobrodošli (vaše ime), studente (naziv fakulteta) fakulteta sa indeksom broj (broj indeksa)".

## ▼ Poglavlje 6

# Domaći zadatak 1

## ZADATAK 1

*Domaći zadatak za proveru stečenog znanja o servletima.*

Kreirati Java EE 7 veb aplikaciju po sledećim zahtevima:

1. index.html stranica sadrži polja za logovanje;
2. Klasa servleta Servlet1 preuzima podatke unete u ova polja.
3. Nakon obrade preuzetih podataka otvara se nova stranica dobrodošlice (osobi čiji su podaci uneti na početnoj formi);
4. Na ovoj stranici postoji još jedan link koji preusmerava na stranicu koja odgovara servletu Servlet2;
5. Klikom na link otvara se nova stranica na kojoj se obaveštava korisnik (na osnovu podataka sa početne forme) da je uspešno savladao osnove rada sa servletima.

## ▼ Poglavlje 7

# Zaključak

## ZAKLJUČAK

*Lekcija se bavila analizom i demonstracijom primene klasa servleta u veb aplikacijama.*

Lekcija 2 je za zadatak imala analizu i demonstraciju mehanizama za kreiranje i upotrebu klasa servleta u Java EE 7 veb aplikacijama. Zbog važnosti tema koje su obrađivane, napravljen je prvo detaljan uvod koji se bavio sledećim tezama:

- definisanje servleta;
- pokazivanjem organizacije klase servleta;
- isticanjem izvesnih pravila koji se poštuju prilikom rada sa servletima.

U nastavku lekcija se bavila kreiranjem i postavljanjem servleta. Posebno, u ovom delu lekcije, diskutovano je detaljno o fazama životnog ciklusa servleta.

Problematici praćenja sesija posvećeno je najviše vremena u okviru ove lekcije. Detaljno su posmatrani slučajevi primene kolačića i modifikacije URL adrese za realizovanje proces praćenja sesija. Ovaj deo lekcije se fokusirao, takođe, i na primenu klasa filtera i asinhronih servleta.

Lekcija je završila izlaganje diskusijom o korišćenom aplikacionom serveru GlassFish.

Savladavanjem ove lekcije studenti će biti osposobljeni da koriste servlete u Java EE 7 veb aplikacijama.

## LITERATURA

*Za pripremu materijala L01 korišćena je savremena pisana i elektronska literatura.*

1. <http://www.oracle.com/us/products/middleware/application-server/050870.pdf>
2. <http://www.slideshare.net/martyhall/jsp-and-servlet-tutorial-session-tracking>
3. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
4. David R. Heffelfinger. 2015. Java EE7 Developomnet With NetBeans 8, PACK Publishing
5. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
6. Josh JUneau. 2015. Java EE7 Recipes, Apress