Napredni koncepti Jave i web programiranje u Javi

Book · January 2005

DOI: 10.13140/RG.2.1.3310.5528

CITATIONS
0

READS
1,212

1 author:

Siniša Vlajić
Faculty of organisational sciences - University of Belgrade
41 PUBLICATIONS **54** CITATIONS

SEE PROFILE

UNIVERZITET U BEOGRADU

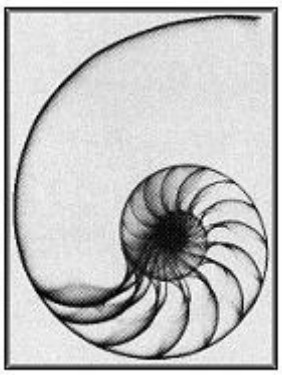
FAKULTET ORGANIZACIONIH NAUKA

(Institut za informacione sisteme)

RAZVOJ WEB APLIKACIJA KORIŠĆENJEM J2EE TEHNOLOGIJE

(SKRIPTA)

Autor: dr Siniša Vlajić



Beograd - 2005.

Autor
Dr Siniša Vlajić

Naslov skripte
RAZVOJ WEB APLIKACIJA KORIŠĆENJEM J2EE TEHNOLOGIJE

Izdavač
Dr Siniša Vlajić, Beograd

E-mail izdavača
vlajic@fon.rs

Sadržaj:

1. J2EE platforma - uvod

J2EE platforma za razvoj enterprise aplikacija može se podeliti u 4 dela koja prate odgovarajuće tehnologije i servisi:

1. Web tehnologije koje se koriste u razvoju prezentacionog nivoa J2EE ili stand-alone Web aplikacije:
    - Java servleti.
    - Java server strane (JavaServer Pages - JSP).
    - Java server strane sa standardnom bibliotekom tagova (JavaServer Pages Standard Tag Library - JSTL).
    - Java server oblici (JavaServer Faces - JSF).
    - Internacionalizacija i lokalizacija Web aplikacija.

2. Enterprise JavaBeans (EJB) tehnologije koje se koriste u razvoju poslovne logike J2EE aplikacije:
    - Session bean-ovi.
    - Entity bean-ovi.
    - Message-driven bean-ovi.
    - Enterprise JavaBeans upitni jezik

3. Java XML tehnologije za razvoj aplikacija koje procesiraju XML dokumente i implementiraju Web servise:
    - Java API za XML procesiranje (JAXP).
    - Java API za XML-RPC (JAX-RPC).
    - SOAP attachments API za Javu (SAAJ).
    - Java API za XML registrovanje (JAXR).

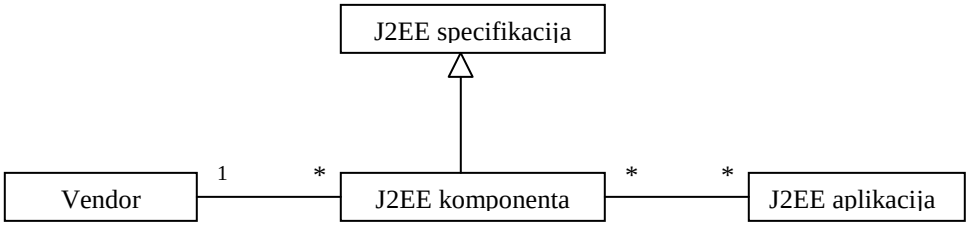
4. Servisi J2EE platforme koje koriste sve navedene tehnologije:
    - Transakcije.
    - Povezivanje resursa (Resource connections).
    - Zaštita (Security).
    - Java servis poruke (Java Message Service).

J2EE platforma podržava:

- Više nivojski distribuirani aplikacioni model (Multitiered distributed application model).
- Komponente koje se mogu ponovo koristiti (Reusable components).
- Jedinstveni model zaštite (Unified security model).
- Fleksibilnu transakcionu kontrolu (Flexible transaction control).
- Web servise koji prihvataju i razmenjuju podatke koji su zasnovani na XML (Extensible Markup Language) standardima i protokolima.

1.1. Odnos između J2EE specifikacije, komponente, aplikacije i vendora

Kada korisnik pravi J2EE aplikaciju on može da ih sastavlja od J2EE komponenti koje su napravili različiti vendori. Jedna J2EE komponenta može biti ugrađena u više različitih J2EE aplikacija. Ono što je zajedničko za svaku J2EE komponentu je to da je ona napravljena na osnovu J2EE specifikacije. Na taj način je moguće povezati J2EE komponente različitih vendora (Slika J2EESVKA).

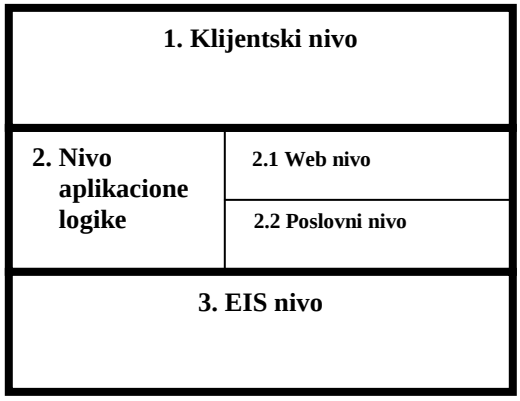


Slika J2EEsvka: Odnos između J2EE specifikacije, komponente, aplikacije i vendora.

1.2 Distribuirane multi-nivojske aplikacije

J2EE platforma koristi distribuirani više-nivojski model (obično tronivojski model – Slika TNA) za razvoj J2EE enterprise aplikacija (složenih aplikacija). J2EE aplikacija se sastoji od J2EE komponenti koje su instalirane na različitim mašinama. Komponente, odnosno mašine na kojima se izvršavaju komponente su dodeljene različitim nivoima J2EE aplikacije:

1. Klijentski nivo - komponente se izvršavaju na klijentskim mašinama.
2. Nivo aplikacione logike (Nal) – komponente se izvršavaju na J2EE serveru. Ovaj nivo je podeljen u dva podnivoa:
   2.1 Web nivo
   2.2 Poslovni nivo
3. Enterprise information system (EIS) nivo - komponente se izvršavaju na EIS serveru (Obično je to neki sistem za upravljanje bazom podataka (SUBP) – Database management system).



Slika TNA: Model tronivojske aplikacije

Na osnovu navedenog može da se zaključi da postoje tri lokacije gde se nalaze mašine na kojima se izvršavaju komponente (Slika TNAM):

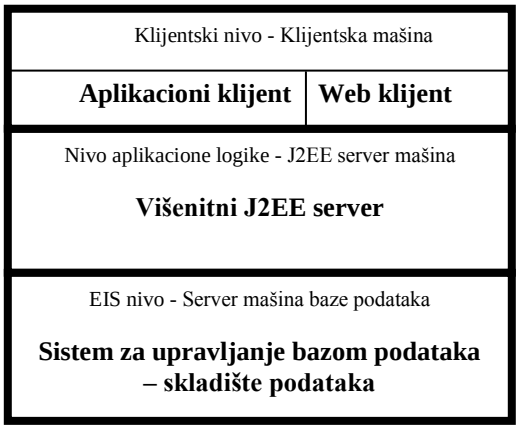
- Klijentske mašine.
- J2EE server mašine.
- Database ili legacy mašine (back and mašine).

Slika TNAM: Lokacije mašina u tronivojskoj arhitekturi

J2EE komponente su pisane u Java programskom jeziku i kompajliraju se na isti način kao i bilo koji drugi program koji je pisan u Javi. Razlika između J2EE komponenti i "standardnih" Java klasa se ogleda u tome da se J2EE komponente ugrađuju (assembled) u J2EE aplikaciju ako zadovoljavaju J2EE specifikaciju J2EE aplikacije se izvršavaju u okruženju J2EE servera. J2EE server ima ulogu da prati i upravlja izvršenjem J2EE aplikacije.

1.3 Realizacija tronivojske arhitekture preko J2EE klijenta, J2EE servera i sistema za upravljanje bazom podataka

Postoje dve moguće vrste J2EE klijenata: a) Aplikacioni klijent i b) Web klijent. Tro-nivojske aplikacije proširuju standardni klijent-server model postavljanjem višenitnog (multithread) J2EE servera između klijentske aplikacije i skladišta podataka (back-and storage) (Slika TNAPR). Skladište podataka je obično realizovano preko nekog od sistema za upravljanje bazom podataka.



Slika TNAPR: Realizacija tronivojske arhitekture preko J2EE klijenta, J2EE višenitnog servera i sistema za upravljanje bazom podataka

1.3.1 J2EE klijent

J2EE klijent (Slika J2EEKL) može biti:

a) Web klijent ili
b) Aplikacioni klijent.

Web klijent

Web klijent se sastoji od dva dela: (1) dinamičkih Web strana koje sadrže različite tipove markup jezika (HTML,XML,...), koji se generišu[1] pomoću Web komponenti koje se izvršavaju na Web nivou i (2) Web čitača (browser) koji tumači (renders) strane koje prihvata od servera.
Web klijent se ponekad naziva tanak klijent (thin client). Tanak klijent obično ne postavlja upite bazi, ne izvršava kompleksna poslovna pravila, niti se povezuje sa legacy aplikacijama.
Tanak klijent odgovornost za izvršenje složenih operacija prenosi na stranu J2EE servera koji obezbeđuje zahtevanu funkcionalnost uključujući i dopunske ne-funkcionalne osobine: zaštitu, brzinu i pouzdanost.

Apleti

Web strane koje šalje server a koje koje prihvata Web čitač mogu da sadrže aplete. Aplet je program napisan u Java programskom jeziku koji izvršava Java virtualna mašina koja je instalirana na Web čitaču. Da bi aplet mogao uspešno da se izvrši na klijentskoj strani verovatno je potrebno da postoji Java Plug-in[2] i security policy datoteka.

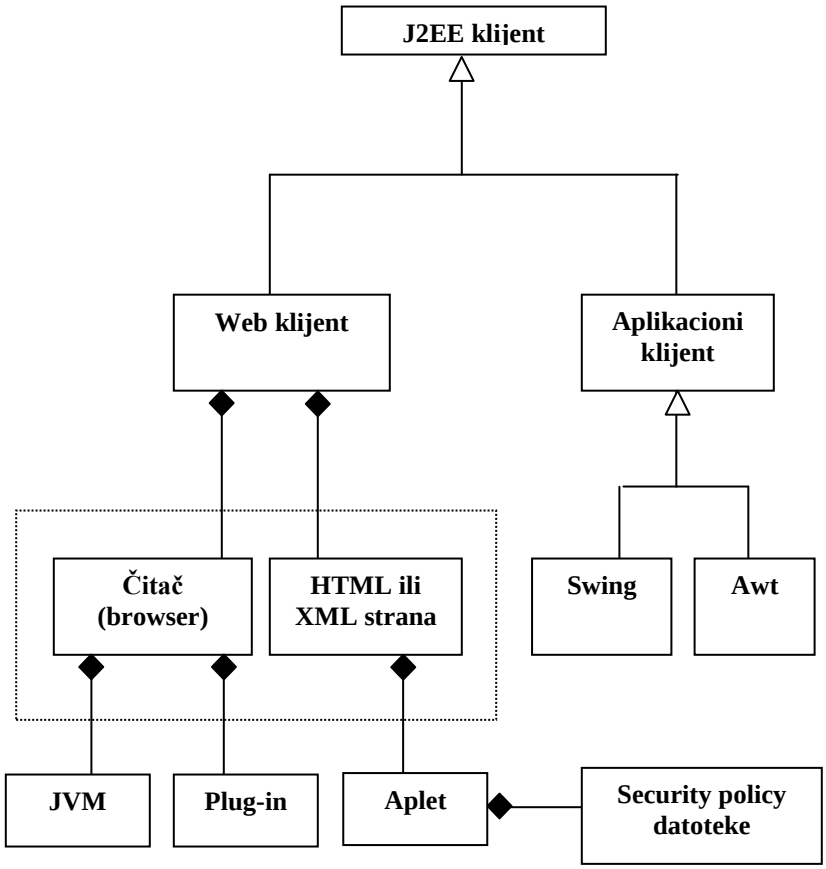
Web komponente koriste API za kreiranje Web klijentskih programa zato što oni ne traže plug-in i security policy datoteke na klijentskoj strani[3]. Takođe Web komponente omogućavaju jasnije i modularnije projektovanje aplikacije zato što one obezbeđuju način da dele aplikaciono programiranje od projektovanja.

[1] Važno: **Web komponente generišu Web strane.**
[2] Java Plug-in je mehanizam koji omogućava da se najnovije verzije Jave mogu izvršiti u run-time okruženju čitača (Internet Explorer, Netscape,...).
[3] JVM, Plug-in i Security policy datoteka su potrebni kako bi se izvršio aplet. Oni nisu neophodni kod izvršenja HTML ili XML strana.

Web strana. Na ovaj način ljudi koji se bave projektovanjem Web strana ne moraju da znaju sintaksu Java programskog jezika da bi mogli da rade njihov posao.

Aplikacioni klijenti

Aplikacioni klijenti se izvršavaju na klijentskoj mašini i oni obezbeđuju način da se naprave složeniji korisnički interfejsi nego što to može biti obezbeđeno pomoću markup jezika. Obično se grafički korisnički interfejs (graphical user interface - GUI) kreira iz Swing ili AWT klasa. Međutim moguće je napraviti aplikacionog klijenta koji će da se izvršava preko komandne linije[4]. Aplikacioni klijenti direktno pristupaju enterprise beans-ima koji se izvršavaju na nivou poslovne logike. Ukoliko postoji potreba aplikacioni klijenti mogu da naprave HTTP konekciju sa servletima koji se izvršavaju na Web nivou.



Slika J2EEKL: J2EE klijenti

JavaBeans komponentna arhitektura

Serverski i klijentski nivoi mogu takođe da uključe komponente koje su zasnovane na Java-Beans komponentnoj arhitekturi (Java Beans components) kako bi se upravljalo tokom podataka između J2EE klijenta i komponenti koje se izvršavaju na J2EE serveru, ili između J2EE server komponenti i baze podataka. Java bean komponente nisu J2EE komponente i one nisu opisane J2EE specifikacijom.

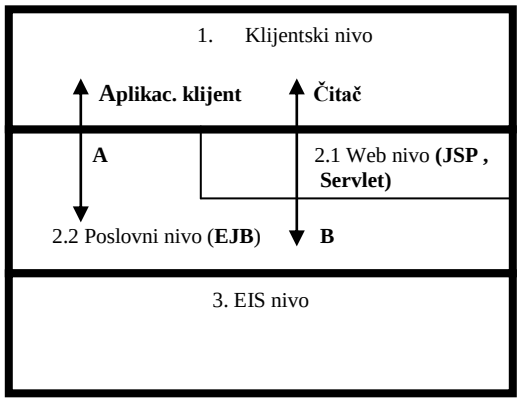
[4] To je veoma jednostavan tekstualni korisnički interfejs, koji je sličan aplikacijama koje su pisane u okruženju DOS operativnog sistema.

1.3.2 Komunikacija J2EE klijenta i J2EE servera

Klijent komunicira sa poslovnim nivoom (Slika J2EEKLAS) koji se izvršava na J2EE serveru ili A) **direktno** *(Aplikacioni klijent<->EJB komponente poslovnog nivoa)* ili B) u slučaju klijenta koji se izvršava na Web čitaču, **preko** JSP strana ili servleta koji se izvršavaju na Web nivou *(Čitač<-> JSP ili Servlet komponente Web nivoa<->EJB komponente poslovnog nivoa)*.

J2EE aplikacija može da koristi "tankog" klijenta koji se izvršava na Web čitaču (thin browser-based client) ili "debelog" aplikacionog klijenta (thick application client). Kod odlučivanja šta da se koristi treba naći balans između čuvanja funkcionalnosti na klijentskoj strani (thick client) i prenošenja odgovornosti za izvršenje funkcionalnosti na stranu servera (thin client).

Aplikacijom se lakše može upravljati, u smislu nadogradnje i održavanja, ukoliko se funkcionalnost aplikacije prenese na stranu servera.



Slika J2EEKLAS: Komunikacija J2EE klijenta i J2EE servera

1.3.3 J2EE server

J2EE server prati i upravlja izvršenjem J2EE aplikacija. J2EE server može da bude Web server (npr. *Bundled Tomcat (5.0.28)* ili *Tomcat 5*) ili može istovremeno da bude i Web i aplikacioni server (*Sun Java System Application Server 8*). Web serveri podržavaju rad Web komponenti (Servlet, JSP i Web servis) dok aplikacioni serveri podržavaju rad EJB komponenti (Session bean i Entity bean).

Na sajtu: **http://www.netcraft.com/survey** se može naći pregled najpopularnijih Web servera.

1.3.4 J2EE aplikacija

J2EE aplikacija može biti realizovana kao Web ili kao EJB aplikacija.

1.3.4.1 Web aplikacija

Web aplikacija predstavlja aplikaciju koja se izvršava u okruženju Web servera. Web aplikacija se sastoji od Web komponenti, standardnih Javinih klasa, HTML strana, JavaBeans komponenti i Message Handler komponenti.

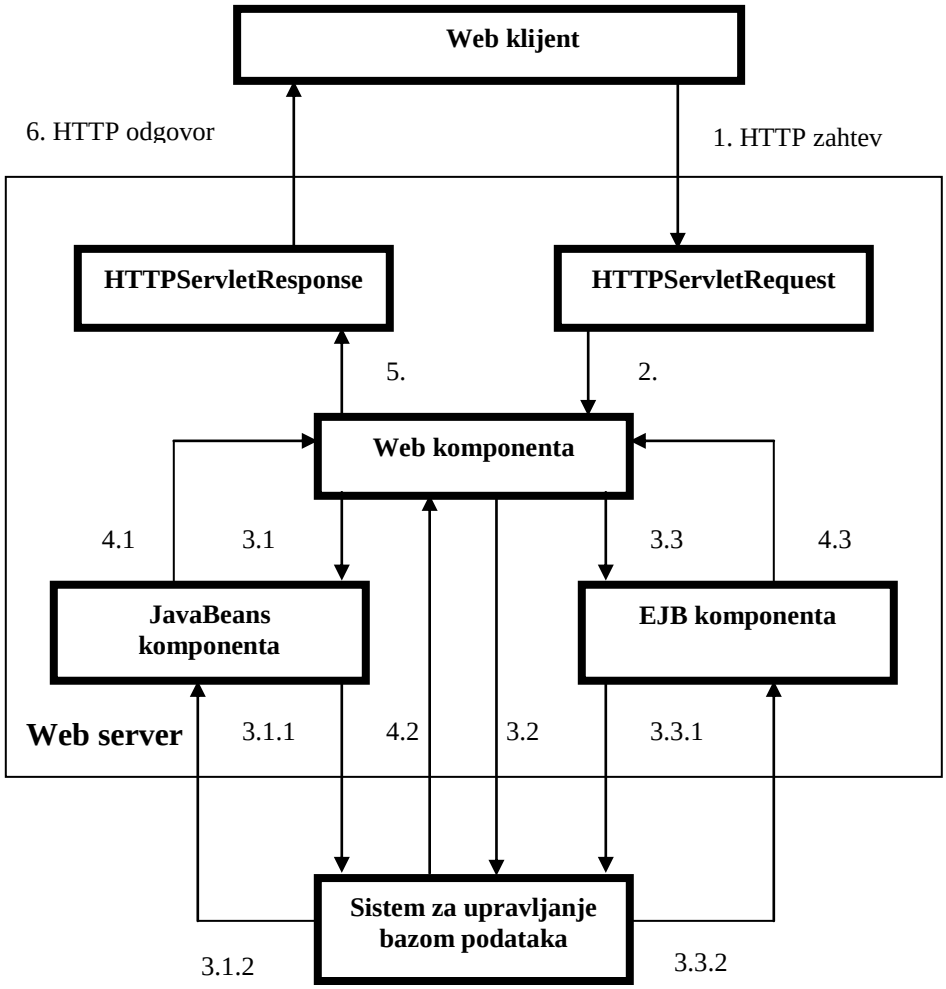
Postoje dva tipa web aplikacija:

- *Prezentaciono-orijentisana* – Prezentaciono orijentisane Web aplikacije generišu Web strane na osnovu zahteva koje im postavlja klijent. Web strana može biti realizovana preko HTML-a, XML-a ili nekog drugog markup jezika.
- *Servisno-orijentisana* – Servisno orijentisane Web aplikacije koriste Web servise da realizuju zahteve koje im postavlja klijent ili prezentaciono – orijentisana aplikacija (u tom slučaju je prezentaciono-orijentisane aplikacija klijent servis-orijentisanoj Web aplikaciji).

Interakcija između Web klijenta i Web aplikacije je sledeća (Slika IWkWa):

Klijent šalje HTTP zahtev do Web servera (1.). Web server (koji podržava Java Servlet i JSP tehnologiju), konvertuje zahtev u *HTTPServletRequest* objekat. Taj objekat se šalje do Web komponente (2), koja može da bude u interakciji sa JavaBeans komponenatama (3.1, 4.1) ili sa SUBP-a. (3.2, 4.2) ili sa EJB

komponentama (3.3, 4.3) kako bi generisala dinamički sadržaj. JavaBeans komponente (3.1.1, 3.1.2) ili EJB komponente (3.3.1, 3.3.2) mogu biti u interakciji sa SUBP-a. Web komponenta može da generiše *HTTPServletResponse* objekat (5) ili može da prosledi zahtev do druge Web komponente.Web server konvertuje *HTTPServletResponse* objekat u HTTP odgovor (6) i vraća ga nazad do klijenta.



Slika IWkWa: Interakcija između Web klijenta i Web aplikacije

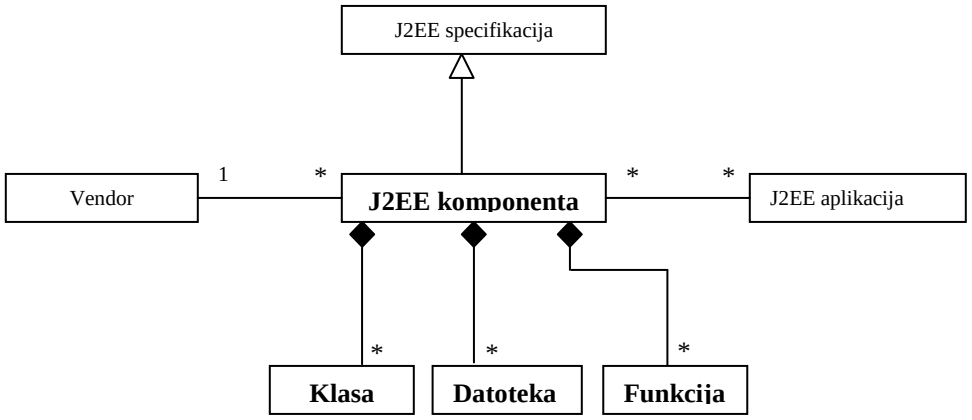
1.3.4.2 EJB aplikacija

EJB aplikacija predstavlja aplikaciju koja se izvršava u okruženju aplikacionog servera. EJP aplikacija se sastoji od EJB komponenti (Session beans, Entity beans i Message-driven beans), standardnih Javinih klasa i JavaBeans komponenti. EJB aplikacije su prvenstveno namenjeno da realizuju ponašanje i strukturu softverskog sistema[5]. Session beans služe da realizuju ponašanje softverskog sistema, dok entity beans služe da realizuju strukturu softverskog sistema. EJB server obezbeđuje EJB aplikacijama mehanizme perzistentnosti i transakcije, pomoću EJB kontejnera, tako da je programer oslobođen pisanja programskog koda koji se odnosi na perzistentnost i transakcije.

[5] Struktura i ponašanje softverskog sistema predstavljaju logiku poslovnog sistema (poslovnu logiku) koji je realizovan preko softverskog sistema.

1.3.5 J2EE Komponente

J2EE komponenta *(Slika J2EEK)* je softverska jedinica (software unit) koja ima neku funkcionalnost i koja je pridružena J2EE aplikaciji i koja je realizovana u skladu sa J2EE specifikacijom. J2EE komponenta se sastoji od skupa međusobno povezanih klasa i datoteka koje joj obezbeđuju željenu funkcionalnost. J2EE komponente međusobno komuniciraju u cilju obezbeđenja određene funkcionalnosti.

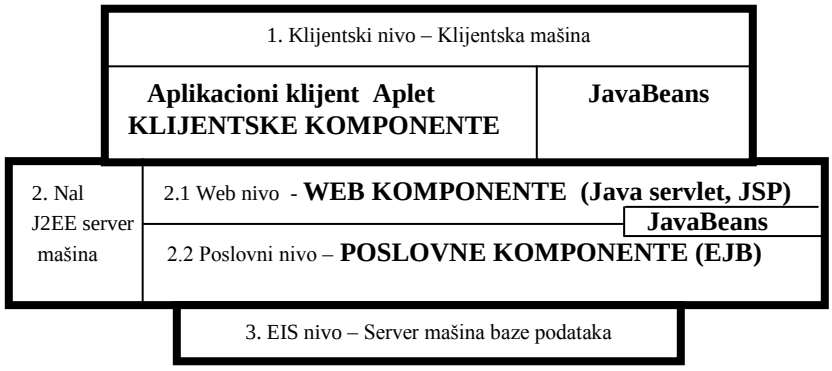


Slika J2EEK: Sastav J2EE komponente.

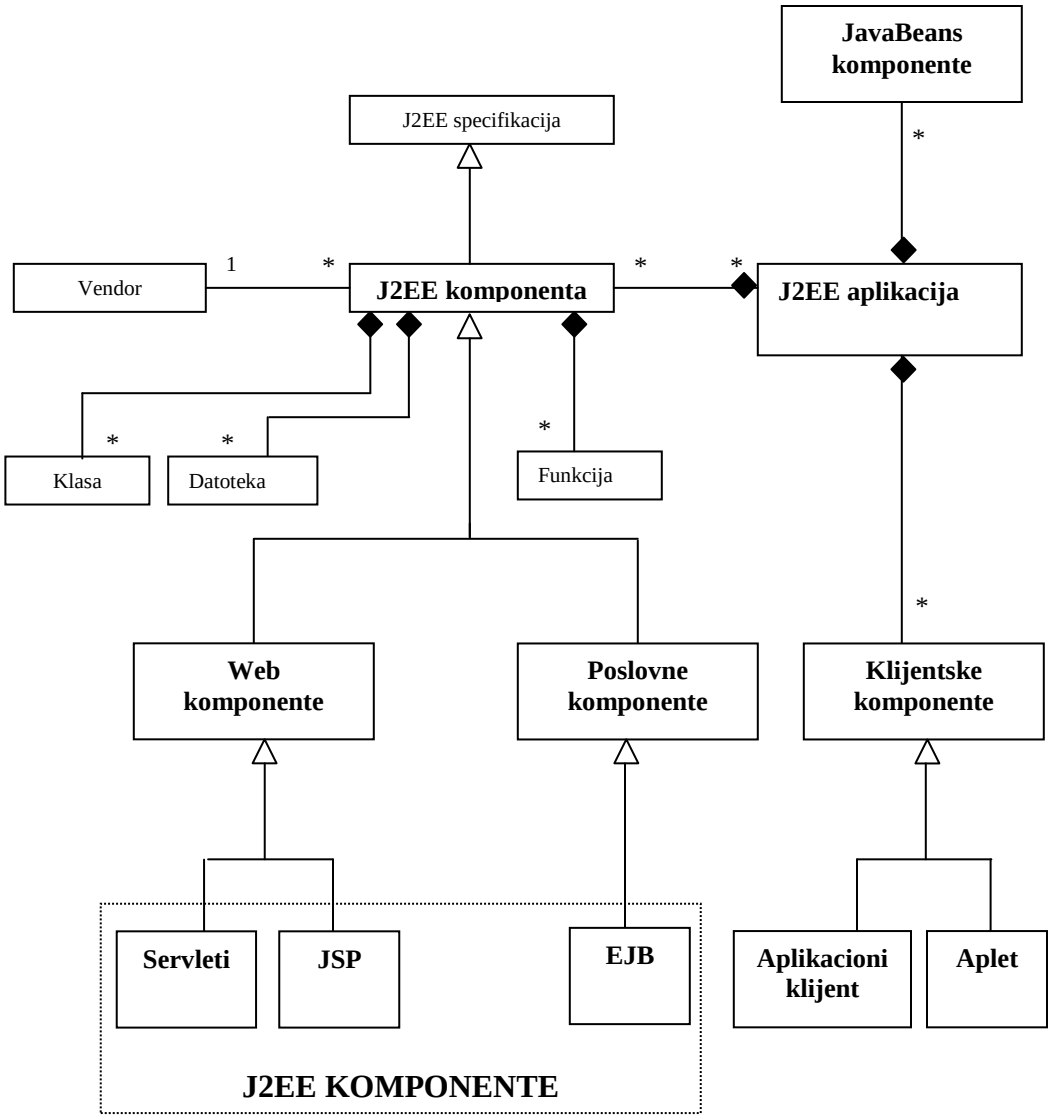
J2EE aplikacije se prave od komponenti *(Slika J2EERKNA)* . Postoje sledeće komponente:

a) Klijentske komponente.
b) Web komponente.
c) Poslovne komponente.
d) JavaBeans komponente.

Aplikacioni klijenti i apleti su klijentske komponente koje se izvršavaju na klijentskoj mašini. Java servleti i JSP su Web komponente koje se izvršavaju na J2EE serverskoj mašini. EJB komponente su poslovne komponente koje se izvršavaju takođe na J2EE serverskoj mašini.
Web komponente (Servleti i JSP-ovi) i EJB komponente (EJB-ovi) su J2EE komponente (Slika J2EEKOMP) jer su realizovane u skladu sa J2EE specifikacijom.
Klijenteske komponente (aplikacioni klijenti i apleti) nisu J2EE komponente jer nisu realizovane u skladu sa J2EE specifikacijom (klijentske komponente nisu razmatrane J2EE specifikacijom).
Na serverskoj i klijentskoj strani mogu da postoje i JavaBeans komponente koje takođe nisu realizovane u skladu sa J2EE specifikacijom (nisu razmatrane J2EE specifikacijom).



Slika J2EERKNA: Komponente J2EE aplikacije



Slika J2EEKOMP: Sastav J2EE aplikacije

1.3.5.1 Web komponente

J2EE Web komponente su ili servleti ili strane koje su kreirane korišćenjem JSP tehnologije (JSP strane). Servleti su Javine klase koje dinamički procesiraju zahteve i konstruišu odgovore. JSP strane su tekst dokumenti koji se izvršavaju kao servleti ali oni dozvoljavaju da se na neposredniji način kreira statički sadržaj tih dokumenata (moguće je direktno pisati HTML kod u JSP stranu). Servleti i JSP strane se mogu zajedno koristiti u jednoj Web aplikaciji ali treba znati da svaka od navedenih tehnologija je usmerena ka određenom tipu Web aplikacije. Servleti su najpogodniji za servisno-orijentisane aplikacije (Web servisi su implementirani kao servleti) i za kontrolne funkcije prezentaciono-orijentisane aplikacije, kao što su usmeravanje (dispatching) zahteva i obrada netekstualnih podataka. JSP strane su pogodne za generisanje markup strana (HTML, SVG-Scalable Vector Graphics, WML-Wireless Markup Language i XML), odnosno one su pogodne za prezentaciono-orijentisane aplikacije.

Najviše Web aplikacija koriste HTTP protokol pri komunikaciji sa klijentom.

Java Servlet tehnologija je osnova svih Web aplikacionih tehnologija (JSP, JavaServer Faces i JavaServerPages Standard Tag Library).

Kontejner Web komponenti
Web aplikacija može da koristi "tankog" klijenta koji se izvršava na Web čitaču (thin browser-based client) ili "debelog" aplikacionog klijenta (thick application client). Tanak klijent ne sadrži programski kod koji: a) obrađuje transakcije, b) upravlja stanjima programa, c) prati rad više niti,..., i druge kompleksne detalje Web aplikacije. Svaka Web aplikacija se sastoji od J2EE komponenti koje se pridružuju odgovarajućim kontejnerima koji obezbeđuju servise (transakcije, perzistentnost,...) koji oslobođavaju onoga ko razvija aplikaciju programiranja niskog nivoa. To znači da navedene servise ne mora da razvija onaj ko pravi aplikaciju. On treba isključivo da bude koncenrisan na shvatanje poslovne logike aplikacije.
Preporuka je da se koristi tanak klijent jer se tada Web aplikacijom može lakše upravljati, u smislu njene nadogradnje i održavanja.

Web komponente su podržane servisima koje obezbeđuje Web kontejner. Web kontejner obezbeđuje sledeće servise za Web komponente:

a) usmeravanje zahteva (request dispatching),
b) zaštita (security),
c) konkurentnost (concurrency) i
d) upravljanje životnim ciklusom (life-cycle management).

On takođe daje Web komponentama pristup do API-a koji se odnose na imenovanje (naming), transakcije i e-mail. Izvesni aspekti Web aplikacije mogu biti konfigurisani kad je aplikacija instalirana ili raspoređena (deployed) unutar Web kontejnera. Konfiguracione informacije se nalaze u XML datoteci koja se naziva *Opisivač rasporeda Web aplikacije (Web aplication deployment descriptor (DD))*. DD mora da bude u skladu sa šemom koja je opisana u Java servlet specifikaciji.

1.3.5.2 EJB (Poslovne) komponente
Postoje dve vrste EJB komponenti:
1) Session bean i
2) Entity bean
3) Message-driven bean.

Session bean je odgovoran:

a) za komunikaciju između web i poslovnog nivoa i za
b) poslovna pravila (ponašanje) aplikacije.

Entity bean je odgovoran:

a) da predstavi poslovne podatke (strukturu) aplikacije
b) i da komunicira sa bazom podataka.

Postoji sledeće poređenje između session i entity bean-a :

	Session bean	**Entity bean**
Odgovornost	Izvršava zadatak ili proces za klijenta	Predstavlja poslovne objekte
Deljeni pristup	Jedna instanca po klijentu	Deljena instanca za više klijenata
Perzistentnost	Nije perzistentan – kada klijent završi rad sa binom isti nije više na raspolaganju	Perzistentan je – po završetku rada EJB stanje objekta je sačuvano u bazi.

Message-driven bean kombinuje osobine session bean-a i java servisa poruke (Java Message Service – JMS) dozvoljavajući poslovnim komponentama da prihvate JMS poruke asinhrono.

1.3.6 Sistem za upravljanje bazom podataka
Sistem za upravljanje bazom podataka u najopštijem smislu je softverski sistem koji treba da omogući:

- čuvanje podataka,
- pouzdanost podataka u slučaju otkaza sistema,

- istovremeno korišćenje podataka od strane više korisnika i
- jednostavan način komunikacije sa bazom podataka (npr. SQL).

J2EE aplikacije komuniciraju sa SUBP, preko odgovarajućih JDBC ili JDBC-ODBC drajvera, kada žele da obezbede perzistentnost njihovih objekata i kada žele da vide (pročitaju) zapamćene objekte, shodno upitu koji postavlja J2EE aplikacija SUBP-a[6].
Web aplikacije moraju neposredno da komuniciraju sa SUBP (programer mora da implementira komunikaciju sa SUBP-a), dok EJB aplikacije to čine posredno (programer ne mora da implementira komunikaciju sa SUBP, jer je to obezbedio EJB server, odnosno njegov EJB kontejner).

1.4 Koraci u razvoju Web aplikacije

Proces kreiranja, raspoređivanja i izvršenja Web aplikacije se sastoji iz sledećih koraka:

1. Pravi se programski kod Web komponenti *(kreiranje Web aplikacije)*.
2. Pravi se opisivač rasporeda (*deployment descriptor*) za Web aplikaciju *(kreiranje Web aplikacije)*.
3. Kompajliraju se Web komponente i pomoćne klase na koje ukazuju komponente.
4. Aplikacija se pakuje (ovo je opciona mogućnost) u rasporedljivu (*deployable*) jedinicu *(raspoređivanje Web aplikacije)*.
5. Aplikacija se postavlja u Web kontejner i izvršava se *(izvršenje Web aplikacije)* .
6. Klijent pristupa URL-u koji referencira na Web aplikaciju.

1.5 J2EE Web modul

Kada se kaže Web aplikacija onda se misli na aplikaciju koja sadrži odgovarajuće elemente (Web komponente, Web strane, utility klase, JavaBeans komponente,...), koji su na *neki način organizovani* i koji međusobno sarađuju u cilju obezbeđenja određene funkcionalnosti. Kada se govori o Web aplikacijama naglasak se stavlja na njihovu funkcionalnost a ne na način njihovog organizovanja.
Koncept Web modula naglašava aspekt organizovanja Web aplikacije. Iz navedenog može da se kaže da Web modul predstavlja organizacioni okvir unutar koga se postavlja Web aplikacija. Web modul može biti organizovan (raspoređen) kao nespakovana struktura datoteka ili može biti pakovan u JAR datoteku koja je prilagođena Web aplikacijama (WAR- Web archive file). Web modul koji je spakovan preko WAR datoteke je prenosiv (portable) i on se može postaviti (prodružiti) i izvršiti preko bilo kog Web kontejnera koji zadovoljava Java Servlet specifikaciju.
Da bi se WAR datoteka postavila na Web server, datoteka mora da sadrži i runtime opisivač rasporeda (runtime deployment descriptor). Runtime opisivač rasporeda je XML datoteka koja sadrži informacije o kontekstu Web aplikacije (gde je početak-koren Web aplikacije, preko koga se pristupa Web komponentama).

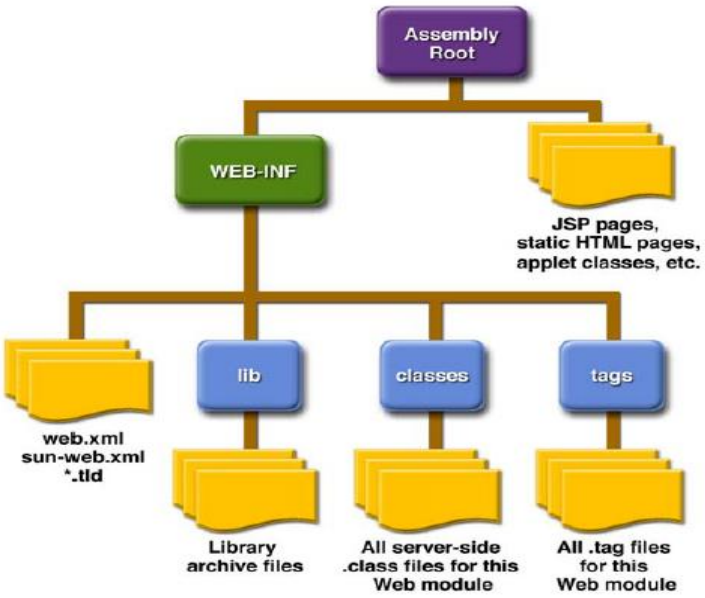
Web modul ima specifičnu strukturu (*Slika Wms*). Na vrhu Web modula je *dokument koren (document root)* aplikacije. On sadrži:

- JSP strane, klase i arhive klijentske strane i statičke Web resurse, kao što su npr. HTML strane, slike,...

Pored toga on sadrži poddirektorijum /WEB-INF/ koji sadrži sledeće datoteke i direktorijume:

- *web.xml*: Opisivač rasporeda (*deployment descriptor*) Web aplikacije.
- *.tld:* Datoteke za opis tag biblioteke (Tag library descriptor files).
- *classes*: Direktorijum koji sadrži klase serverske strane: servlete, utility klase i JavaBeans komponente.
- *tags*: Direktorijum koji sadrži tag datoteke koje su implementacije tag biblioteke.
- *lib*: Direktorijum koji sadrži biblioteke JAR arhiva koje pozivaju klase sa serverske strane.

[6] Često se kaže da upit nad bazom prestavlja jedan pogled nad bazom podataka.



Slika Wms:Web modul struktura

1.6 J2EE kontejneri

Kontejneri obezbeđuju okruženje na kome se izvršavaju komponente. To znači da komponente (aplikacioni klijent, aplet,servlet, JSP, EJB) pre izvršenja moraju biti *ugrađene (asemblirane) u J2EE modul i dodeljene odgovarajućem kontejneru.*

U process ugrađivanja - sastavljanja (asemliranja) komponenti se vrši podešavanje (setovanje) odgovarajućih XML datoteka za:

a) svaku komponentu J2EE aplikaciji i
b) za same J2EE aplikaciju.

Kontejner, na osnovu definisanog podešavanja komponenti, obezbeđuje odgovarajuću podršku koja je obezbeđena preko J2EE servera. Navodimo neke od najvažnijih servisa J2EE servera:

- J2EE model zaštite (security model) koji omogućava da se konfiguriše Web komponenta ili enterprise bean tako da sistemskim resursima može pristupiti samo autorizovani korisnici.
- J2EE transakcioni model omogućava da se specificiraju veze između metoda koje prave jednu transakciju tako da su sve metode u transakciji tretirane kao jedna jedinica.
- Servis JNDI pretraživanja obezbeđuje jedinstveni interfejs (unified interface) za višestruko imenovanje i servise direktorijuma (directory services).
- J2EE model udaljene konekcije (remote connectivity model) omogućava komunikaciju između klijenata i enterprise bean-ova. Pošto je enterprise bean kreiran klijent poziva metode i ima utisak da se metoda izvršava na istoj virtualnoj mašini gde se nalazi klijentski program. Metode se izvršavaju na serverskoj mašini.

J2EE arhitektura obezbeđuje servise koji se mogu konfigurisati(configurable services). Shodno tome komponente iste J2EE aplikacije mogu se ponašati različito u zavisnosti od toga gde su raspoređene. Kontejner takođe upravlja servisima koji se ne mogu konfigurisati, kao što su enterprise bean-ov i servletov životni ciklus, database connection resource pooling, perzistentnost podataka i pristup do API-ja J2EE platforme. Mada je perzistentnost podataka nekonfigurabilni servis, J2EE arhitektura omogućava da se prekrije perzistentnost koju obezbeđuje kontejner uključivanjem odgovarajućeg programskog koda u

enterprise bean implementaciju kada se želi veća kontrola perzistentnosti od podrazumevane perzistentnosti koju obezbeđuje kontejner.

Postoje sledeći tipovi kontejnera:

1. Web kontejner
Upravlja izvršenjem JSP strana i servlet komponenti za J2EE aplikacije. Web komponente i njihovi kontejneri se izvršavaju na J2EE serveru.

2. Enterprise JavaBeans (EJB) kontejner
Upravlja izvršenjem enterprise beans-a za J2EE aplikacije. Enterprise beans-i i njihovi kontejneri se izvršavaju na J2EE serveru.

3. Application client container
Upravlja izvršenjem komponeti aplikacionog klijenta. Aplikacioni klijenti i njihovi kontejneri se izvršavaju na klijentskoj mašini.

4. Aplet kontejneri
Upravlja izvršenjem apleta. Sadrži Web čitač i Java Plug-in koji se zajedno izvršavaju na klijentskoj mašini.

1.7 Web servisi

Web servisu su Web enterprise aplikacije koje koriste na XML-u[7] zasnovane standarde i transportne protokole za razmenu podataka sa klijentima.
Preko XML API-ja se vrši translacija podataka iz aplikacionog oblika u XML data-stream oblik koji se može preneti pomoću XML transportnog protokola. Upravo taj XML data-stream oblik omogućava interoperabilnost između Web servisa i klijenata.

Navešćemo i ukratko objasniti neke od XML tehnologija koje se koriste kao podrška Web servisima:

a) Java API za XML procesiranje (Java API for XML Processing – (JAXP))
Java API za XML procesiranje (JAXP) podržava procesiranje XML dokumenata korišćenjem Document Object Model-a (DOM), Simple API za XML (SAX) i Extensible Stylesheet Language Transformations (XSLT)[8]. JAXP je projektovan da bude fleksibilan i on omogućava korišćenje bilo kog XML parsera ili XSL procesora i on podržava W3C šemu.

b) Java API za XML RPC (Java API for XML-Based RPC – (JAX-RPC))
Java API za XML RPC koristi SOAP-HTTP protokol kod XML poziva udaljenih procedura (XML-based Remote Procedure Calls - RPCs) preko interneta. JAX-RPC takođe podržava WSDL tako da se mogu importovati i eksportovati WSDL dokumenti. Sa JAX-RPC i WSDL moguća je laka komunikacija između klijenata i servisa koji se izvršavaju na Java platformi ili na bilo kojoj drugoj platformi, kao što je npr. .NET. Na primer, na osnovu WSDL dokuemenata, Visual Basic .NET klijent može biti konfigurisan da koristi Web servise koji su implementirani u Java tehnologiji ili Web servis može biti konfigurisan da prepozna Visual Basic .NET klijenta.

[7] XML je platformski ukrštiv (cross-platform), proširljiv (extensible), na tekstu zasnovani standard (text-based standard) za reprezentovanje podataka.
XML sadrže:
- Tagove koji opisuju podatke.
- Šeme koje opisuju koji tagovi mogu biti korišćeni u XML dokumentima.
- XML stylesheet koji upravljaju prikazivanjem i obradom podataka.

[8] Pravilo: **JAXP** koristi **DOM, SAX i XSLT.**

JAX-RPC omogućava kreiranje servis aplikacija koje kombinuje HTTP sa Secury Socket Layer (SSL) Java tehnologijom i Transport Layer Security (TLS) protokolima da omoguće autentifikaciju[9]. SSL i TLS obezbeđuju integritet poruke obezbeđenjem enkripcije i autentifikaciju klijenta i servera.

SOAP transportni protokol
Klijentski zahtevi(**requests**) i Web servis odgovori (**responses**) se prenose pomoću SOAP (Simple Object Acess Protocol) protokola. SOAP je XML protokol (XML based protocol) koji je zasnovan na HTTP protokolu[10] .SOAP protokol omogućava kompletnu interoperatibilnost između klijenata i Web servisa, koji se mogu izvršavati na različitim platformama i na različitim lokacijama na internetu.
SOAP obrađuje poruku koja se prenosi na sledeći način:

- definiše se XML omotnica (envelope) koja opisuje šta je u poruci i kako se procesira poruka,
- uključuju se XML pravila šifriranja (encoding rules) da se izraze pojavljivanja podataka raznih tipova koji se nalaze u poruci,
- definiše se XML konvencija za reprezentovanje zahteva do udaljenog servisa i rezultujućeg odgovora.

WSDL standardni format (WSDL Standard Format)
Web Services Description Language (WSDL) je standardizovani XML format za opis mrežnih servisa (network services). Opis uključuje ime servisa, lokaciju servisa i način komunikacije sa servisom. WSDL opisi servisa mogu se čuvati u UDDI registrima i/ili mogu biti publikovani na Web-u.

c) SOAP sa pridruženim API-jem za Javu (SOAP with Attachments API for Java – (SAAJ))
SOAP sa pridruženim API-jem za Javu (SAAJ) je API niskog nivoa od koga JAX-RPC zavisi. SAAJ omogućava proizvođenje i korišćenje poruka koje odgovaraju SOAP 1.1 specifikaciji i SOAP-u sa pridruženim API-jem. SAAJ API se u praksi ne koristi. Umesto njega se koristi JAX-RPC API koji je višeg nivoa u odnosu na SAAJ i SOAP.

d) Java API za XML registrovanje (Java API for XML Registries – (JAXR))
Java API za XML registrovanje (JAXR) omogućava da se pristupi registrima preko Web-a i da se sačuvaju različite šeme (informacije i web servisi) u standardnom poslovnom registru. JAXR omogućava povezivanje tih šema. JAXR podržava ebXML i Universal Description, Discovery i Integration (UDDI) standarde registrovanja i publikovanja informacija na internetu. Klijenti mogu da pristupe ovim informacijama i web servisima.

1.8 Pakovanje aplikacija (Packaging Applications)

J2EE aplikacija se može spakovati u Enterprise Archive (EAR) datoteku. EAR je standardna Java Archive (JAR) datoteka koja ima .ear ekstenziju. Korišćenjem EAR datoteka moguće je spajati više različitih J2EE modula (aplikacija) u novu (složenu) J2EE aplikaciju, koji takođe može biti spakovana u EAR datoteku.
EAR datoteka sadrži J2EE module i opisivač rasporeda (deployment descriptor).
Postoje 4 tipa J2EE modula:

a) EJB modul koji sadrže class datoteke za enterprise bean-ove i EJB opisivač rasporeda. EJB moduli su pakovani u JAR datoteke sa .jar ekstenzijom.
b) Web modul koji sadrže servlet class datoteke, JSP datoteke, class datoteke koje koriste Web komponente i Web aplikacioni opisivač rasporeda. Web moduli su pakovani u JAR datoteke sa ekstenzijom .war (Web Archive).
c) Aplikaciono klijentski modul koji sadrži class datoteke i aplikaciono klijenstski opisivač rasporeda. Aplikaciono klijentski moduli su pakovani kao JAR datoteke sa .jar ekstenzijom.
d) Resurs adapter moduli koji sadrže sve Java interfejse, klase, native biblioteke i drugu dokumentaciju, zajedno sa resurs adapter opisivačem rasporeda. Zajedno oni implementiraju

[9] *Autentifikacija (Authentication)* omogućava da se ispita da li neko može da pristupi infromacijama koje su zaštićene. Informacije koje se transportuju preko interneta su naročito ranjive tako da je veoma važno da se konfiguriše JAX-RPC Web servis da zaštiti podatke u toku njihovog prenosa.
[10] HTTP je request-response standard za slanje poruka preko interneta.

Connector arhitekturu za pojedinačne EIS. Resurs adapter moduli su pakovani u JAR datoteke sa .rar (resource adapter archive) ekstenzijom.

Kod opisivača rasporeda informacije su deklarativne. To znači da one mogu biti promenjene bez potrebe da se menja izvorni kod. U runtime-u J2EE server čita opisivač rasporeda i izvodi akcije nad aplikacijom, modulom ili komponentom.

1.9 Uloge u razvoju J2EE aplikacije

Razvoj, uvođenje i održavanje J2EE aplikacije se može podeliti u nekoliko aktivnosti, pri čemu je za svaku aktivnost definisano šta treba da se radi, ko treba da radi i šta je rezultat te aktivnosti:
a) Kreiranje okruženja za razvoj J2EE aplikacija

Ko obavlja aktivnost: ***J2EE provajderi proizvoda (J2EE Product Provider)*** su kompanije koje prave okruženja za razvoj J2EE aplikacija .
Rezultat aktivnosti: Okruženja (*npr. NetBeans 4.1, Sun-ov aplikacioni server 8,...*) za razvoj J2EE aplikacija koja su u u skladu sa J2EE specifikacijom.

b) Kreiranje alata razvoj, spajanje i pakovanje J2EE aplikacije.

Ko obavlja aktivnost: ***Provajder oruđa (Tool provider)*** je kompanija ili osoba koja kreira oruđa za razvoj, spajanje i pakovanje J2EE aplikacije
Rezultat aktivnosti: Oruđa za razvoj, spajanje i pakovanje J2EE aplikacije

c) Kreiranje Web komponenti, EJB-ova, apleta ili aplikacionih klijenata za korišćenje u J2EE aplikacijama.

Ko obavlja aktivnost: ***Provajder aplikacione komponente (Application Component Provider)*** je kompanija ili osoba koja kreira Web komponente, enterprise bean-ove, aplete ili aplikacione klijente za korišćenje u J2EE aplikacijama.

Za svaki od navedenih elemenata J2EE aplikacije se obavljaju sledeće podaktivnosti:

c1) Kreiranje Web komponenti

Ko obavlja aktivnost:
Razvijač Web komponenti (Web Component Developer) izvršava sledeće zadatke kod pravljenja war datoteke :

- Piše i kompajlira servlet izvorni kod.
- Piše JSP i HTML datoteke.
- Specificira opisivač rasporeda.
- Pakuje .class, .jsp i .html datoteke i opisivač rasporeda u war datoteku.

Rezultat aktivnosti: war datoteka.

c2) Kreiranje EJB komponenti

Ko obavlja aktivnost:
Razvijač enterprise bean-a (Enterprise Bean Developer) izvršava sledeće zadatke kod pravljenja EJB JAR datoteke koja sadrži enterprise bean-ove:

- Piše i kompajlira izvorni kod
- Specificira opisivač rasporeda
- Pakuje .class datoteke i opisivač rasporeda u EJB JAR datoteku

Rezultat aktivnosti: jar datoteka.

c3) Kreiranje aplikacionog klijenta

Ko obavlja aktivnost:
Razvijač aplikacionog klijenta (Application Client Developer) izvršava sledeće zadatke kod pravljenja JAR datoteke koja sadrži aplikacionog klijenta:

- Piše i kompajlira izvorni kod.

- Specificira opisivača rasporeda.
- Pakuje .class datoteke i opisivač rasporeda u JAR datoteku.

Rezultat aktivnosti: jar datoteka.

c4) Kreiranje apleta

Ko obavlja aktivnost:
Razvijač аплета(Aplet Developer) izvršava sledeće zadatke kod pravljenja JAR datoteke koja sadrži aplet:

- Piše i kompajlira aplet.
- Piše HTML datoteku i povezuje je sa apletom.
- Specificira opisivača rasporeda.
- Pakuje aplet, HTML i opisivač rasporeda u JAR datoteku.

Rezultat aktivnosti: jar datoteka.

d) Spajanje (asembliranje) elemenata u J2EE aplikaciju.

Ko obavlja aktivnost: ***Spajač (asembler) aplikacije (Application Assembler)*** je kompanija ili osoba koja prihvata elemente od provajdera komponenti i spaja ih u J2EE aplikaciju. On izvršava sledeće zadatke:

- Spaja JAR i WAR datoteke, koje su kreirane u predhodnim fazama, u J2EE aplikaciju (EAR datoteka)
- Specificira opisivač rasporeda za J2EE aplikaciju
- Verifikuje sadržaj EAR datoteke shodno J2EE specifikaciji.

Rezultat aktivnosti: ear datoteka.

d) Instaliranje i izvršavanje J2EE aplikaciju.

Ko obavlja aktivnost: ***Aplikacioni raspoređivač i administrator (Application Deployer and Administrator)*** je kompanija ili osoba koja konfiguriše i raspoređuje J2EE aplikaciju, administrira računarima i mrežnom infrastrukturom gde se J2EE aplikacija izvršava i nadgleda runtime okruženje. Pored toga se podešava transakciona kontrola, atributi zaštite i specifiicra se konekcija ka bazama.
U toku konfiguracije, raspoređivač sledi instrukcije koje su dobijene od provajdera komponenti da reši eksterne zavisnosti, podesi zaštitu i dodeli transakcione atribute. Na kraju raspoređivač izvršava aplikaciju, odnosno komponente na serveru (za komponente se generišu odgovarajući kontejneri servera).
U suštini on izvršava sledeće zadatke:

- Dodaje J2EE aplikaciju (EAR datoteku), koja je kreirana u predhodnoj fazi, do J2EE servera (pridruživanje).
- Konfiguriše J2EE aplikaciju, shodno okruženju, modifikovanjem opisivača rasporeda J2EE aplikacije.
- Verifikuje sadržaj EAR datoteke shodno J2EE specifikaciji.
- Izvršavanje J2EE aplikacije na J2EE serveru.

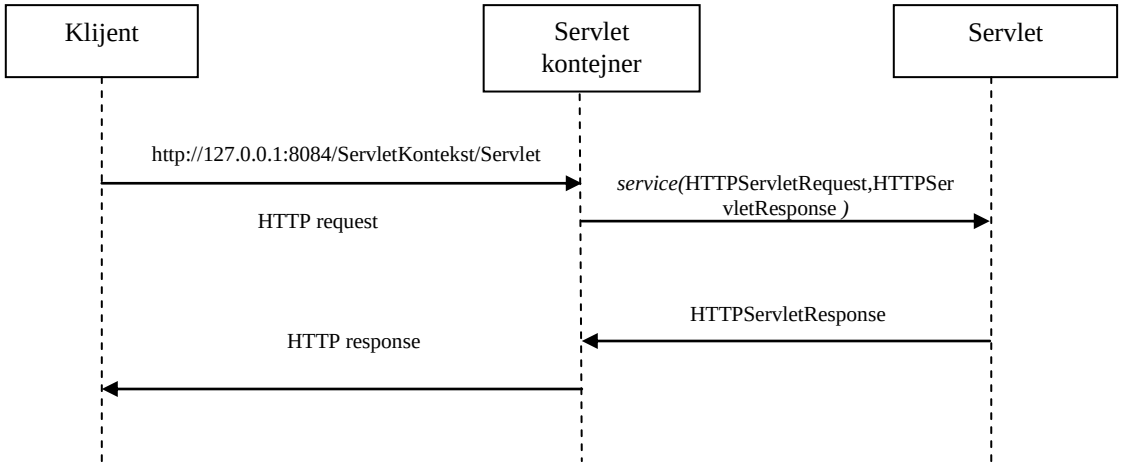
Rezultat aktivnosti: J2EE aplikacija koja se izvršava na J2EE serveru.

2. Razvoj Web aplikacije

2.1 Java servleti

J2EE Web komponente su ili servleti ili strane koje su kreirane korišćenjem JSP tehnologije (JSP strane). Servleti su Javine klase koje dinamički procesiraju zahteve i konstruišu odgovore.

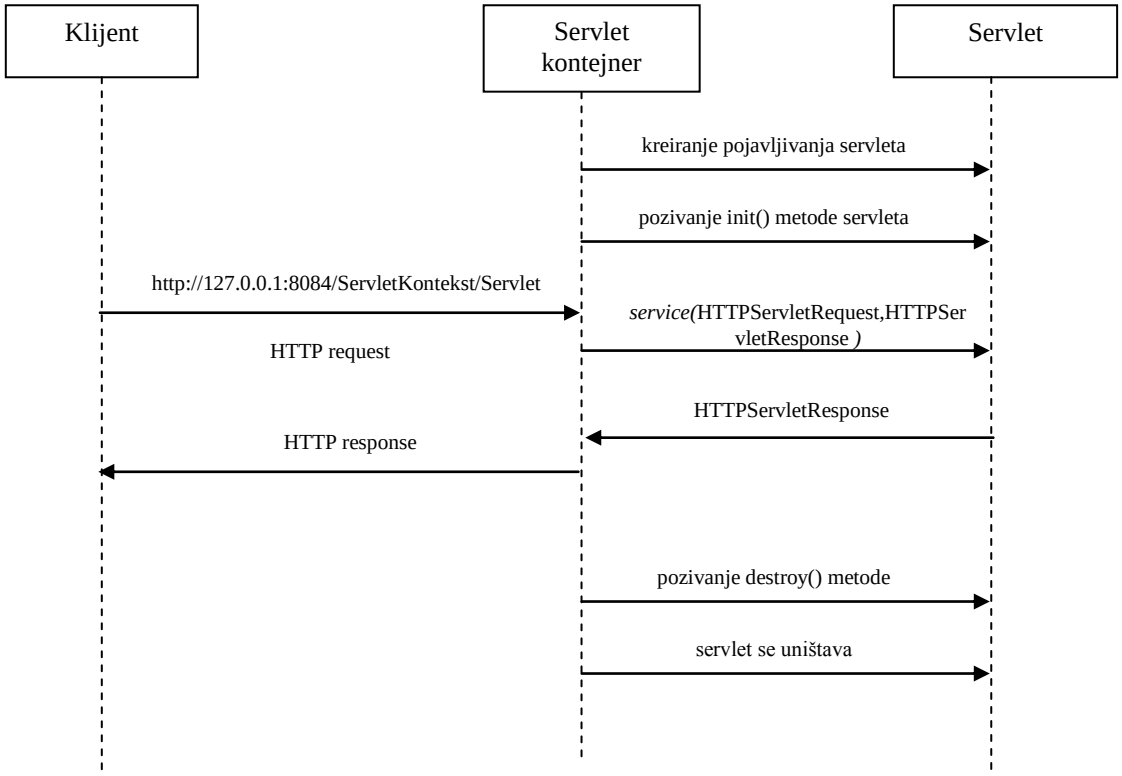
Interakcija između Web klijenta i servleta
Klijent šalje HTTP zahtev do Web servera. *Servlet kontejner* (*Servlet container*)[11] Web servera konvertuje zahtev u *HTTPServletRequest* objekat. Nakon toga kontejner poziva *service()* metodu servleta i kao argumente mu šalje *HTTPServletRequest* i *HTTPServletResponse* objekat, koji treba da se napuni sa rezultatima izvršenja servleta. Nakon izvršenja servleta *HTTPServletResponse* objekat se šalje nazad do kontejnera koji ga konvertuje u HTTP odgovor i vraća ga nazad do klijenta.



Uloga kontejnera
Servlet kontejneri obrađuju zahteve koje šalju klijenti , prosleđuju zahteve do servleta i vraćaju odgovor od servleta do klijenta. Implementacija kontejnera kod različitih J2EE okruženja se može razlikovati ali interfejs između kontejnera i servleta je isti za bilo koje J2EE okruženje i ono je određeno specifikacijom servleta. Kontejner takođe upravlja životnim ciklusom servleta koji se sastoji iz sledećih koraka:

- Kontejner servleta kreira pojavljivanje (instancu) servleta.
- Kontejner poziva *init()* metodu pojavljivanja.
- Kada kontejner dobije zahtev od klijenta prosleđuje ga do servleta pozivajući njegovu *service()* metodu.
- Pre nego što se uništi pojavljivanje, kontejner poziva *destroy()* metodu.
- Pojavljivanje se uništava i markira se za mehanizam sakupljanja smeća (garbage collection).

[11] Servlet kontejner se u literaturi često naziva mehanizam servleta (servlet engine).



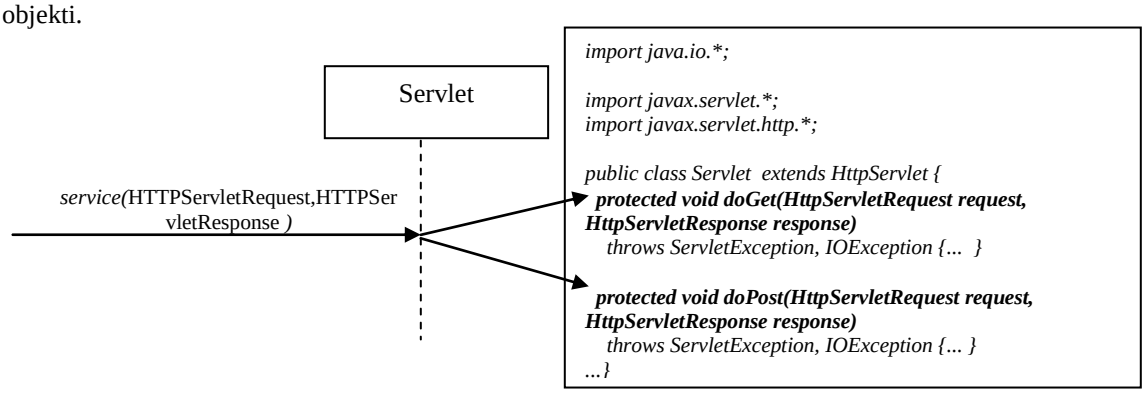
Kontejner mora da obezbedi da:

- *service()* metoda servleta neće biti pozvana pre nego što se izvrši *init() metoda* servleta.
- servlet neće biti uništen dok se pre toga ne izvrši *destroy()* metoda servleta.

Kod tipičnog servlet modela. kontejner kreira po jedno pojavljivanje za svaki servlet jedne aplikacije. Neki servlet može pozvati više klijenata, pri čemu se za svakog klijenta pravi posebna nit. U specifikaciji servleta nije određeno da izvršenje *service()* metode jedne niti isklljučuje izvršenje *service()* metode druge niti. Ukoliko postoji potreba da se obezbedi isključivi pristup jedne niti do *service()* metode servleta tada treba koristiti *synchronized* naredbu.

Uloga service() metode

Kada *service()* metoda prihvati zahtev od kontejnera ona prepoznaje tip zahteva (*GET, POST, HEAD,...*) i vrši preusmeravanje (*dispatching*) poziva do odgovarajuće metode (*doGet(), doPosto, doHead()*) shodno tipu zahteva. Navedenim metodama se takođe prenosi *HTTPServletRequest* i *HTTPServletResponse* objekti.



2.1.1 Koraci u razvoju servleta

Na primeru jednostavnog servleta koji pokazuje pozdravnu poruku biće objašnjen proces kreiranja (*creating*), raspoređivanja (*deploying*) i izvršenja(*executing*) Web aplikacije:

Primer Servlet1: Napraviti HTML dokument koji ce pozvati servlet koji treba klijentu da vrati pozdravnu poruku.Pretpostavimo da se servlet nalazi na lokalnoj mašini čiji je IP jednak 127.0.0.1: 8084[12]

1. Pravi se programski kod Web komponenti *(kreiranje Web aplikacije)*.

a) U datoteci KDobarDan.java se unosi servlet koji treba da prikaže pozdravnu poruku:

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class KDobarDan1 extends HttpServlet {

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();
       out.println("<html>");
       out.println("<head>");
       out.println("<title>Servlet</title>");
       out.println("</head>");
       out.println("<body>");
       out.println("<CENTER><h1>Pozdrav za vas prvi servlet!!!</h1><CENTER>");
       out.println("</body>");
       out.println("</html>");
       out.close();
    }

protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
       processRequest(request, response);
    }
}
```

b) U datoteci Servlet1.html nalazi se HTML kod koji poziva servlet:

```
<html><body>
<p>Primer: Napraviti HTML dokument koji ce pozvati servlet koji ce prikazati pozdravnu poruku:</p>
<! Poziv servleta>
<form action="http://127.0.0.1:8084/ServletPrimer1/DobarDan1" method="POST">
   <p align="left"><input type="submit" value="Pozdravna poruka"> </p>
</form>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>
</body></html>
```

2. Pravi se opisivač rasporeda (*deployment descriptor*) za Web aplikaciju *(kreiranje Web aplikacije)*.

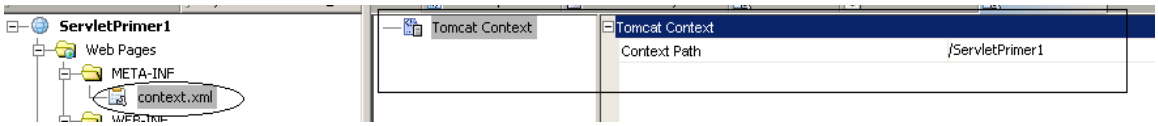
a) Datoteka Web.xml predstavlja opisivač rasporeda:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
```

[12] Port 8084 je podrazumevani port na kome se podiže Web server Net Beans-a.

```
  <servlet>
    <servlet-name>SDobarDan1</servlet-name>
    <servlet-class>KDobarDan1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SDobarDan1</servlet-name>
    <url-pattern>/DobarDan1</url-pattern>
  </servlet-mapping>
</web-app>
```

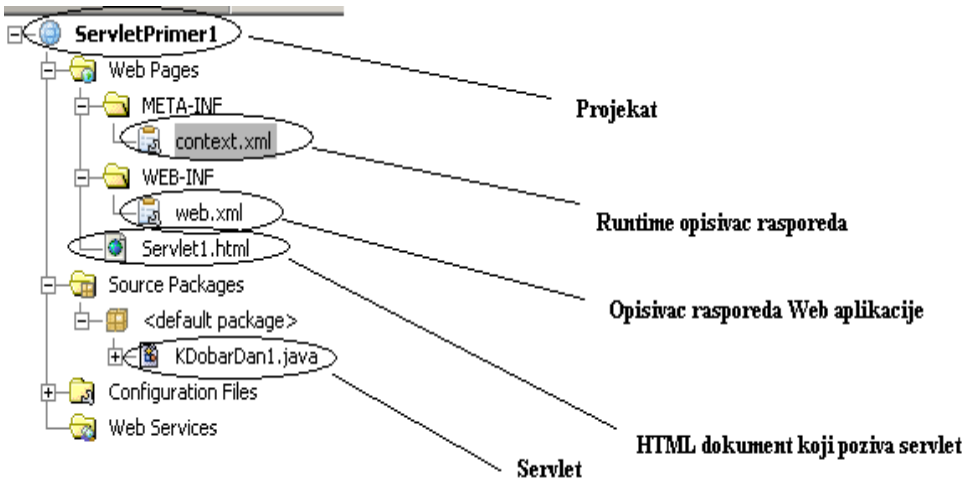
b) Pored navedenog opisivača rasporeda za Web aplikaciju postoji i Runtime opisivač rasporeda koji se nalazi u datoteci context.xml, koja je kod projekta NB4.0 predstavljena preko ekranske forme, a ne preko XML koda:



XML kod dateke context.xml ima sledeći sadržaj:

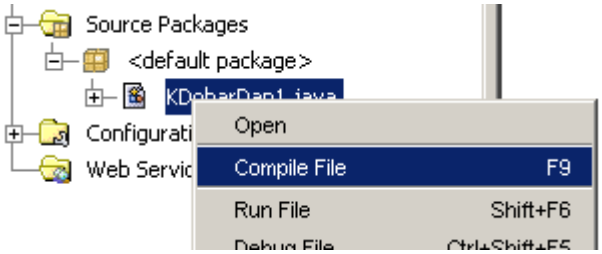
```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletPrimer1" docBase="/D:/Kursevi/Java/WebProgramiranje/Servleti/ServletPrimer1/build/web/">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletPrimer1." suffix=".log" timestamp="true"/>
</Context>
```

Navedene datoteke su kreirane u projektu *ServletPrimer1:*



3. Kompajliraju se Web komponente i pomoćne klase na koje ukazuju komponente.

   a) Kompajlira se datoteka KDobarDan1.java.



Ukoliko je datoteka korektno kompajlirana pojaviće se sledeća poruka:

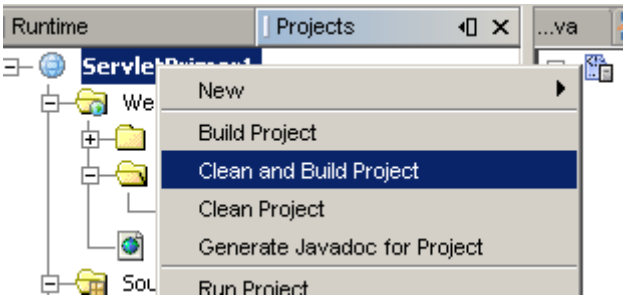
```
init:
deps-jar:
compile-single:
BUILD SUCCESSFUL (total time: 3 seconds)
```

4. Aplikacija se pakuje (ovo je opciona mogućnost) u rasporedljivu (*deployable*) jedinicu *(raspoređivanje Web aplikacije)*.

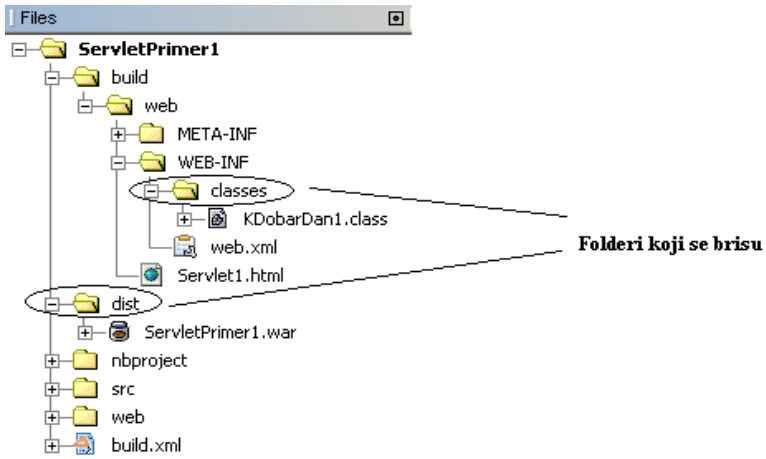
a) Rasporedljiva jedinica je war datoteka koja se krerira na 2 moguća načina:

a1) Poziva se opcija Build Project. U tom slučaju postojeće datoteke se pakuju u ServletPrimer1.war datoteku. Ukoliko ste pre toga menjali neku Web komponentu (npr. izvorni kod servleta), ta promena se neće videtu u war datoteci. Tako da ovu opciju ne preporučujemo.

a2) Poziva se opcija Clean and Build Project.



Pre pakovanja datoteka u war datoteku, se vrši brisanje foldera gde se nalaze class javine datoteke i foldera gde se čuva war datoteka.



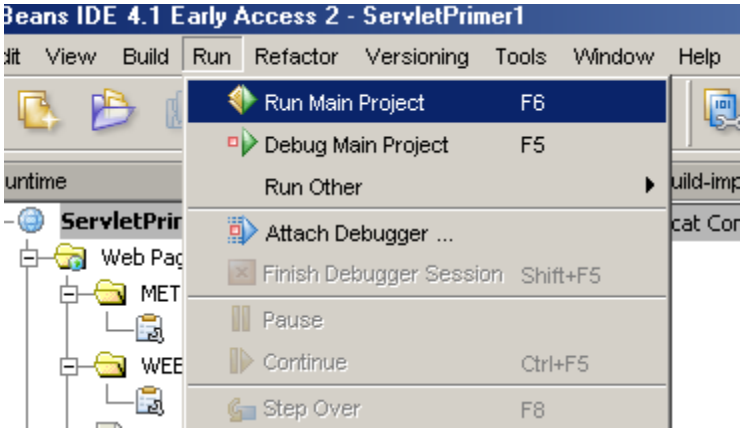
Nakon toga se kompajliraju java datoteka i prevacuju se class datoteke u folder classes koji je pre toga kreiran(folder classes se kreira nakon kreiranja foldera build\web\WEB-INF). Nakon toga se kopiraju dateoteke Context.xml, Web.xml i Servlet1.html na odgovarajuća mesta unutar build foldera.
Kreira se folder dist u kome se čuva war datoteka. Na kraju se postojeće datoteke pakuju u ServletPrimer1.war datoteku koja se prebacuje u folder dist.

Kao rezultat izvršenja navedene opcije dobija se sledeća poruka:

```
init:
deps-clean:
do-clean:
Deleting directory D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\build
Deleting directory D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\dist
post-clean:
clean:
init:
deps-jar:
Created dir: D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\build\web\WEB-INF\classes
Compiling 1 source file to D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\build\web\WEB-INF\classes
Copying 4 files to D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\build\web
compile:
pre-dist:
compile-jsps:
do-dist:
Created dir: D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\dist
Building jar: D:\Kursevi\Java\WebProgramiranje\Servleti\ServletPrimer1\dist\ServletPrimer1.war
post-dist:
dist:
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. Aplikacija se postavlja u Web kontejner i izvršava se *(izvršenje Web aplikacije)* .

Poziva se opcija Run/Run Main Project:



Tada se pokreće Tomcat Web server i on aktivira Web kontejner koji se povezuje sa Web aplikacijom. Web kontejner se ne vidi eksplicitno u NB4.1 okruženju. On u pozadini prati i kontroliše kako se izvršava Web aplikacija. Web aplikacija se podiže na lokalnom hostu na podrazumevanom portu NB4.1 (8084): http://localhost:8084/ServletPrimer1/.

Kao rezultat izvršenja navedene opcije dobija se sledeća poruka:
init:
deps-jar:
compile:
compile-jsps:
run-deploy:
Starting server Bundled Tomcat (5.0.28)
Starting Tomcat process...
Tomcat server started.
Incrementally deploying http://localhost:8084/ServletPrimer1
Completed incremental distribution of http://localhost:8084/ServletPrimer1
Incrementally redeploying http://localhost:8084/ServletPrimer1
deploy?config=file:/D:/Kursevi/Java/WebProgramiranje/Servleti/ServletPrimer1/build/web/META-INF/context.xml&war=file:/D:/Kursevi/Java/WebProgramiranje/Servleti/ServletPrimer1/build/web/
OK - Deployed application from context file file:/D:/Kursevi/Java/WebProgramiranje/Servleti/ServletPrimer1/build/web/META-INF/context.xml
run-display-browser:
Browsing: http://localhost:8084/ServletPrimer1/
run:
BUILD SUCCESSFUL (total time: 1 minute 6 seconds)

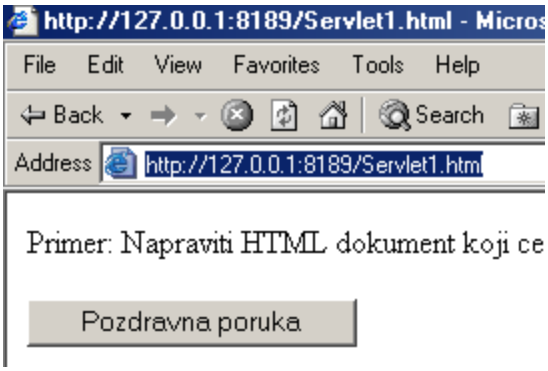
6. Klijent pristupa URL-u koji referencira na Web aplikaciju.

Navedenoj aplikaciji se može pristupiti na sledeća 2 načina ako se pristupa preko lokalne mašine:
a) http://localhost:8084/ServletPrimer1/

b) http://127.0.0.1:8084/ServletPrimer1/

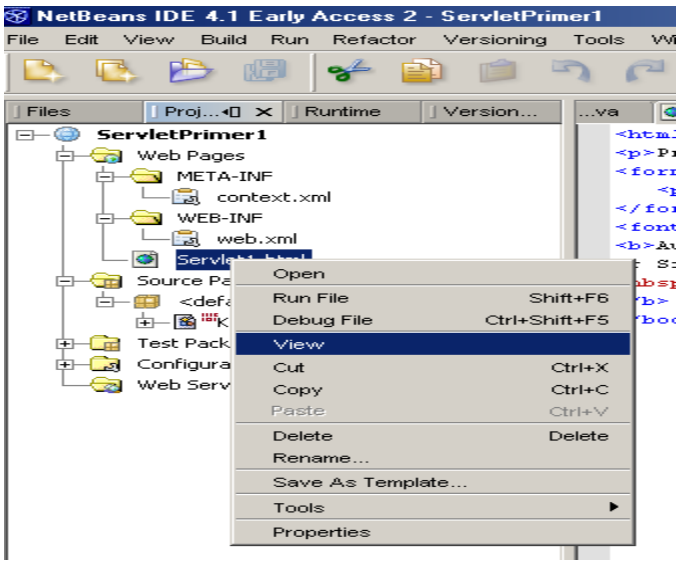
Ukoliko je tekuća mašina vezana za IP: 147.91.128.54 tada se njoj, preko mreže, može pristupiti pozivom sledeće URL adrese: http://127.0.0.1:8084/ServletPrimer1/

Međutim servletu se obično ne pristupa direktno već se to radi posredno preko HTML datoteke koja poziva servlet. U našem primeru se poziva *Servlet1.html* datoteka koja se nalazi npr. na URL-u: *http://127.0.0.1:8189/Servlet1.html*)

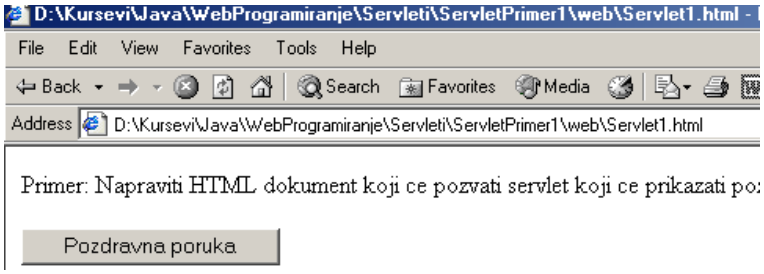


ili se poziva *Servlet1.html* preko lokalnog sistema datoteka ako želimo da testiramo aplikaciju. Kod NB4.1 to se radi na sledeći način:

Treba se postaviti na Servlet1.html datoteku, zatim se klikne desno dugme, koje pravi opadajući meni gde se bira opcija ***View:***



Nakon toga se dobija čitač ispunjen željenom stranom kojoj je pristupljeno preko lokalnog sistema direktorijuma.



a) Nakon toga se klikne dugme *Pozdravna poruka.*

Detaljno objašnjenje poziva servleta:

Unutar datoteke Servlet1.html imamo naredbu koja poziva servlet:

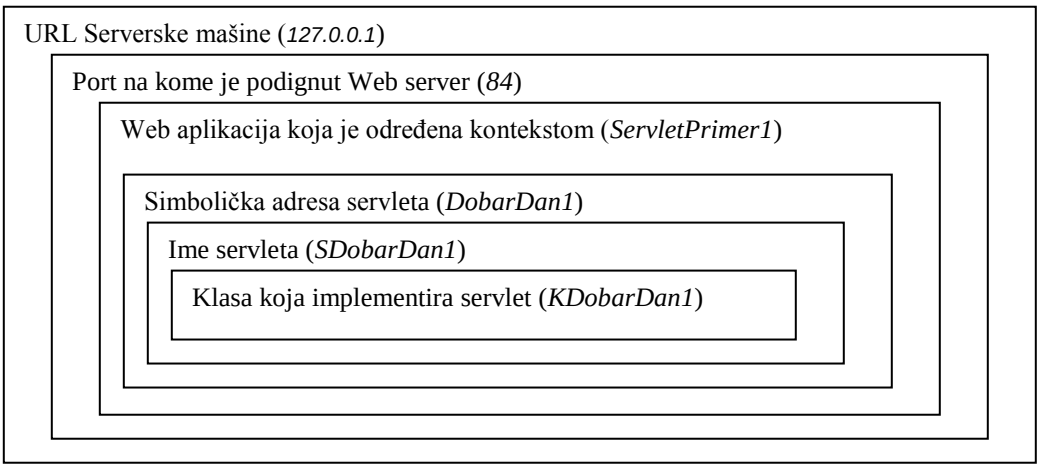
<form action="http://127.0.0.1:8084/ServletPrimer1/DobarDan1" method="POST">

URL naredba se sastoji iz sledećih elemanata:

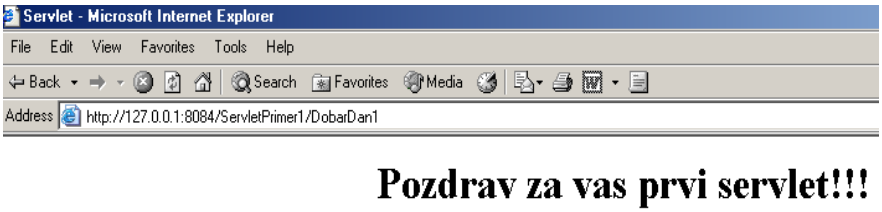
- IP adresa sa brojem porta mašine na kojoj je podignut Web server(*127.0.0.1:8084*).
- Ime konteksta (*ServletPrimer1*)[13] unutar Web servera[14].
  Ime konteksta se navodi u datoteci Context.xml (runtime opisivač rasporeda):
  <Context path="/ServletPrimer1">
- Simboličke adrese gde se nalazi servlet (*DobarDan1*)
  Simbolička adresa servleta se nalazi u datoteci Web.xml (opisivač rasporeda Web aplikacije) iza taga <url-pattern>:

```
<?xml version="1.0" encoding="UTF-8"?>

...  <servlet>
   <servlet-name>SDobarDan1</servlet-name>
   <servlet-class>KDobarDan1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>SDobarDan1</servlet-name>
    <url-pattern>/DobarDan1</url-pattern>
  </servlet-mapping>
</web-app>
```

Na simboličkoj adresi *DobarDan1* se poziva servlet *SDobarDan1* koji poziva klasu *KDobarDan1.class* koja implementira servlet. Na ovaj način je odvojen pristup do servleta od njegove implementacije. Moguće je u *web.xml* staviti drugu klasu koja implementira servlet a da poziv *<form action="http://127.0.0.1:8084/ServletPrimer1/DobarDan1" method="POST">* ostane nepromenjen.
Odnos između elemenata koji učestvuju u pozivaju izvršenja servleta se vidi na sledećoj slici:



b) Kao rezultat izvršenja servleta dobija se sledeća poruka:



[13] Ime konteksta može ali i ne mora da bude isto kao ime projekta (aplikacije).
[14] Jedan Web server može da sadrži više konteksta odnosno više aplikacija.

2.1.2 Metode slanja zahteva Web serveru

Čitač (browser) treba da spakuje podatke koji su prihvaćeni preko ekranske forme i da ih prosledi preko HTTP zahteva (request) do web servera. HTTP zahtev sadrži:

- URL (adresa) strane kojoj se želi pristupiti,
- podatke ukoliko su prihvaćeni preko forme i
- dopunske infromacije zaglavlja (header info), koje se odnose na čitač, dužinu i tip zahteva.

Pored URL strane kojoj se pristupa mora se naznačiti i metoda koja će biti korišćena pri slanju zahteva. Tri najčešće korišćene metode slanja zahteva su HEAD, GET i POST.
HEAD metoda prihvata informacije o dokumentu, ali ne prihvata sadržaj samog dokumenta. GET i POST metode se koriste kada se želi pozvati web aplikacija. Obadve metode obavljaju isti zadatak, ali to rade na različite načine.

GET metoda
Kada se koristi GET metoda, svi podaci koji su bili uneti preko ekranske forme se dodaju do stringa zahteva (*request string*). Podaci se dodaju korišćenjem ključ=vrednost parova (key=value pairs).

GET se koristi kao podrazumevana (default) metoda za sve web zahteve. Kada korisnik poziva direktno neku stranu preko URL-a čitač podrazumeva da je u pitanju GET metoda slanja zahteva. Postoje potencijalno dva problema kada se koristi GET metoda:

a) ukoliko se poziva neka strana, koja je već bila pozvana, tada se može desiti da je stara verzija sačuvana u kešu i da se ona pri pozivu prikaže na ekranu.
b) Iznos podataka koji se može poslati je ograničen.

POST metoda
Kada se koristi POST metoda, svi podaci koji su bili uneti preko ekranske forme se pakuju i unose u telo zahteva *(request body).* Telo zahteva je ulazna datoteka za server koji je čita i u njoj prepoznaje imena promenljivih i njihove vrednosti. Kod POST metode ne postoji ograničenje u veličini podataka koji će biti poslati do servera.

Primer ServletGetPost: Napraviti HTML dokument koji ce pozvati servlet (preko GET metode) koji ce prikazati ime i adresu onoga ko ga je pozvao.
Pravi se HTML datoteku, npr. ServletGetPost.html koja ima sledeći izgled:

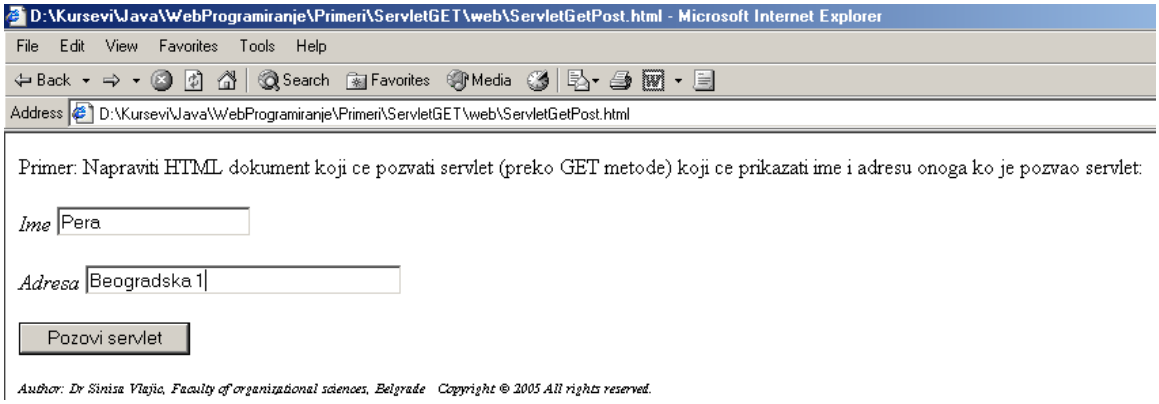
```
<html><body>
<p>Primer: Napraviti HTML dokument koji ce pozvati servlet (preko GET metode) koji ce prikazati ime i adresu onoga ko je pozvao servlet:</p>
<form action="http://127.0.0.1:8084/ServletGetPost/ServletGetPost" method="GET">
        <i>Ime</i>
        <input type="text" size="20" name="Ime">
        <BR><BR>
        <i>Adresa</i>
        <input type="text" size="35" name="Adresa">


<p align="left"><input type="submit" value="Pozovi servlet"> </p>
</form>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade&amp;nbsp;
&amp;nbsp;Copyright © 2005 All rights reserved.
</b>
</body></html>
```

Servlet koji se poziva nalazi se u datoteci KServletGetPost.java:

```
import java.io.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;

public class KServletGetPost extends HttpServlet {

  protected void processRequest(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

       String Ime = request.getParameter("Ime");
       String Adresa = request.getParameter("Adresa");

       out.println("<html>");
       out.println("<head>");
       out.println("<title>ServletGetPost</title>");
       out.println("</head>");
       out.println("<body>");
       if (request.getMethod().equals("POST")==true)
       { out.println("<b> Ovo je post metoda</b><br>");
       }

       if (request.getMethod().equals("GET") ==true)
       { out.println("<b> Ovo je get metoda</b><br>");
       }
       out.println("<Left>");
       out.println("<b> Do servleta su stigli sledeci podaci:</b> " +
       "<p>Ime: " + Ime + "</p> <p>Adresa: " + Adresa);
       out.println("</Left>");
       out.println("</body>");
       out.println("</html>");
       out.close();
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
       processRequest(request, response);
    }
      protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
       processRequest(request, response);
    }

}
```

Kada se poziva sevlet preko HTML strane (nakon punjenja polja *Ime* i *Adresa* se pritiska dugme *Pozovi servlet*):

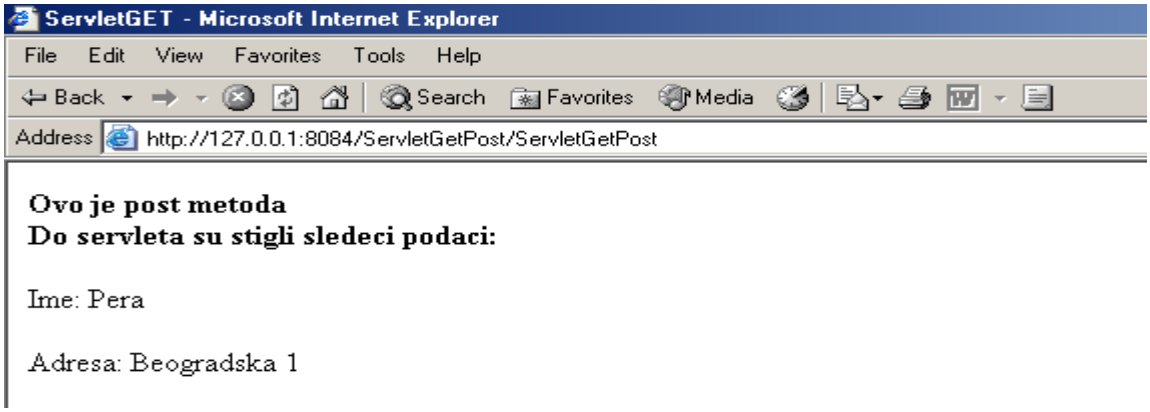


Tada se može primetiti kako se vrednosti unetih podataka dodaju do stringa zahteva:

Ukoliko bi promenili datoteku *ServletGetPost.html* tako da umesto GET metode izaberemo POST metodu:

<html><body>
* Primer: Napraviti HTML dokument koji ce pozvati servlet (preko GET metode) koji ce prikazati ime i adresu onoga ko je pozvao servlet: *
<form action="http://127.0.0.1:8084/ServletGetPost/ServletGetPost" method="POST">
...

tada bi servlet kao rezultat dao:



Iz navedenog može da se vidi da string zahteva nema vrednosti podataka koje šalje. To je sakriveno u datoteci koja se šalje do servera.
Još jednom ponavljamo, rezultat izvršenja servleta je potpuno isti nezavisno od toga koja je metoda prenosa zahteva korištišćena.

Važna napomena: *Servleti isto kao i JSP strane kao rezultat njihovog izvršenja daju HTML dokument koji prosleđuju do korisnika.*

Zadatak ZServletGetPost*: Napraviti HTML dokument koji ce pozvati servlet (preko POST metode) kome će proslediti ime. Servlet treba da prikaže ime i poruku o dužini imena. Pretpostavimo da se servlet nalazi na lokalnoj mašini čiji je IP jednak 127.0.0.1: 8084.*

Primer servleta koji obradjuje informacije o poslovnom partneru

Korisnički zahtev: Napraviti HTML dokument preko koga će biti prihvaćeni lični podaci poslovnog partnera: Ime i prezime, Adresa, E-mail, Telefon, Napomena, Zanimanje, Predznanje Jave. Navedeni podaci treba da budu poslati do servleta koji će ih prihvatiti i prikazati odgovarajući izveštaj.

Prave se sledeće datoteke:

a) *Servlet2.html* koja će da prihvati podatke i da pozove servlet.

```
<!pocetak HTML datoteke>
<html>

<!pocetak zaglavlja datoteke>
<head>
<!meta podaci o datoteci - u pitanju je html tekst datoteka>
<meta http-equiv="Content-Type" content="text/html"></meta>
<!kraj zaglavlja datoteke>
</head>

<!odredjuje se telo html datoteke sa odredjenom bojom pozadine (bg-background) i teksta.>
<body bgcolor="#C0C0C0" text="#000080">
<body>

<!br je skracenica od break-prelomiti. Sluzi da uvede novi red.>
Primer: Napraviti HTML dokument preko koga ce biti prihvaceni licni
podaci poslovnog partnera:<br> Ime i prezime, Adresa, E-mail, Telefon, Napomena, Zanimanje, Predznanje
Jave.

<!otvara formu u kojoj je biti pozvana akcija (poziv servleta)>
<form action="http://127.0.0.1:8084/ServletPrimer2/Servlet2" method="POST">

<!dd pomera tekst u desno (tabulator)>
   <dd><b>OSNOVNI PODACI O POSLOVNOM PARTNERU:</b>
   <br><br>

 <!pocetak uokvirene tabele (border = 1, da je border = 0 tabela ne bi bila uokvirena) cija je duzina 396 pisela.>
  <dd><table width="396" border =1>
         <!td oznacava pocetak sadrzaja celije. Sadrzaj celije ima vrednost 'Ime'>
          <td><i>Ime</i>
         <!td oznacava pocetak sadrzaja celije. Sadrzaj celije ima vrednost 'Ime'>
          <td>
          <!definisanje polja za prihvat teksta (podataka) cija je duzina 50 znakova i cije je ime 'Adresa'>
          <input type="text" size = 50 name="Ime">
        <!tr oznacava pocetak novog reda. >
        <tr>
          <td><i>Adresa</i>
          <td> <input type="text" size = 75 name="Adresa">
        <!kraj zavrsetka reda>
        </tr>
        <tr>
          <td><i>E-mail</i>
          <td> <input type="text" size = 20  name="E-mail">
               </tr>
        <tr>
           <td><i>Telefon</i>
           <td> <input type="text" size = 20 name="Telefon">
        </tr>

       <tr>
          <td><i>Zanimanje</i>
          <!definisanje kombo boksa koji moze da prihvati jednu od ponudenih vrednosti
          koje su definisane preko taga option.>
          <td> <select name="Zanimanje" size="1">
                 <option selected>Student</option>
                 <option>Student</option>
                 <option>Profesor</option>
                 <option>Programer</option>
                 <option>Sluzbenik</option>
```

```
            <option>(Drugo)</option>
          </select>
      </tr>
    <!kraj tabele>
    </table>


<!p je skracenica od paragraf-pasus. Sluzi da razdvoji pasuse u tekstu.Uvodi jedan prazan red izmedju pasusa>
     <dd><p><i><b>Predznanje Jave </b></i>
<!definisanje radio dugmadi preko kojih se bira jedna od ponudjenih vrednosti(Nema, Malo, Srednje, Visoko).>
<!podrazumevana vrednost je Srednje jer je ime boksa checked name (oznaceno ime).>
          <br><br>
          <dd><input type="radio" name="Predznanje" value="Nema">  <i>Nema</i>
             <input type="radio" name="Predznanje" value="Malo"><i>Malo</i>
             <input type="radio" checked name="Predznanje" value="Srednje"><i>Srednje</i>
             <input type="radio" name="Predznanje" value="Visoko"><i>Visoko</i>
          <br><br>
              <dd><i>Napomena</i>
        <dd><p><textarea name="Napomena" rows="5" cols="42"></textarea>

       <dd> <p><input type="checkbox" name="CuvaUBazi" value="false">
       <i>Da li želite da budete sacuvani u bazi</i><b>?</b>
       <p align="center"><input type="submit" value="Obradi partnera"> <input type="reset" value="Obrisi formu"></p>

<!kraj forme se iz koje se poziva akcija>
</form>

<!hr uvodi novi red i podvlaci ga.>
<hr>

<!odredjuje velicinu fonta. i je oznaka za italic slova.   je oznaka za prazno mesto koje se ne vidi ali koje ne dozvoljava da dve reci koje su spojene preko toga budu razdvojene pri prelamanju reda.>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>

<!kraj tela dokumanta>
</body>
<!kraj HTML dokumenta>
</html>
```

Navedena datoteke je detaljno objašnjena preko komentara koji su podvučeni i označeni sa <!...>.

b) *KServlet2.java* koja će da implementira servlet.

```
import java.io.*;
import java.net.*;

// Paketi koji sadrze klase potrebne za rad sa servletima.
import javax.servlet.*;
import javax.servlet.http.*;

// Klasa koja implementira servlet.
public class KServlet2 extends HttpServlet {

   // Metoda koja obradjuje request objekat (cita vrednosti parametara), obradjuje parametre
   // i puni response objekat sa vrednostima koje ce biti poslate klijentu.
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      // Setovanje response objekta na text/html. To znaci da ce response objekat da
      // bude ispunjen sa HTML kodom.
      response.setContentType("text/html");
      // Kreiranje izlaznog objekta out, koji ce da prima podatke i njegovo povezivanje
      // sa response objektom. Sve sto bude poslato na out objekat ce biti preusmereno u
      // response objekat.
      PrintWriter out = response.getWriter();
      // Citanje vrednosti parametara iz request objekta i punjenje odgovarajucih string
```

```
        // objekata sa tom vrednoscu.
        String Ime = request.getParameter("Ime");
        String Adresa = request.getParameter("Adresa");
        String Email = request.getParameter("E-mail");
        String Telefon = request.getParameter("Telefon");
        String Zanimanje = request.getParameter("Zanimanje");
        String Predznanje = request.getParameter("Predznanje");
        String Napomena = request.getParameter("Napomena");
        String CuvaUBazi = request.getParameter("CuvaUBazi");

        // slanje html koda u izlazni objekat.
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Servlet2</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<Left>");
        out.println("<b> Do servleta su stigli sledeci podaci:</b> " +
        "<p>Ime: " + Ime + "</p> <p>Adresa: " + Adresa + "</p> <p> E-mail: " + Email +
        "</p> <p>Predznanje: " + Predznanje + "</p> <p>Telefon: " + Telefon + "</p> <p>Zanimanje: " + Zanimanje + " </p>
<p>Napomena: " + Napomena + "</p> <p>Cuva u bazi: " + CuvaUBazi + "</p>");
        out.println("</Left>");
        out.println("</body>");
        out.println("</html>");
        // zatvaranje izlaznog objekta.
        out.close();
    }
}
```

c) web.xml koja je opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>Servlet2</servlet-name>
    <servlet-class>KServlet2</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Servlet2</servlet-name>
    <url-pattern>/Servlet2</url-pattern>
  </servlet-mapping>
 </web-app>
```

d) Context.xml koji je runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletPrimer2">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletPrimer2." suffix=".log" timestamp="true"/>
</Context>
```

2.1.3 Obrada izuzetaka kod servleta

Izuzetak predstavlja slučaj koji se dešava u toku izvršenja programa koji može da poremeti normalan rad programa.
Izuzetke mogu generisati:

a) metode postojećih Javinih klasa ili
b) metode korisničkih klasa.

Izuzeci se uglavnom javljaju onda kada se desi neka greška u programu. U tom smislu postoje 4 tipa mogućih grešaka:

- Greške pri unosu podataka (User input errors)
- Greške koje su vezane za rad hardvera računara (Device errors) – npr. flopi disk ne čita dobro podatke, štampač nije uključen i spreman za rad.

- Greške koje su vezane za razna fizička ograničenja – disk je pun i ne može da pamti nove podatke, raspoloživa memorija je nedovoljna za izvršenje programa (Physical limitations)
- Greške kod izvršenja metoda programa (Code errors).

Kada se desi neka greška u programu, metoda u kojoj se desila greška pravi (baca) izuzetak koji se odnosi na tu grešku. Izuzetak se zatim hvata (catch) i obrađuje. Mesta u programu gde se hvata greška, nazivaju se tačke presretanja grešaka (error-trapping).

Ukoliko se u programu desi izuzetak koji ne može da se obradi korisničkim programom, pokreće se standardni Javin obrađivač koji prikazuje:

- tekst sa opisom izuzetka i
- stak pozvanih metoda u trenutku dešavanja izuzetka.

Na kraju standardni obrađivač prekida izvršenje programa.

Opšti oblik generisanja i obrade greške (try-catch par) je:

```
try
{ // deo programa u kome se očekuje pojava izuzetaka
  …
} catch (Exception e)
{ // naredbe koje obradjuju izuzetak
  …
}
```

U primeru koji sledi:

```
try {  broj1 = stringToFloat(request.getParameter("Broj1"));
       broj2 = stringToFloat(request.getParameter("Broj2"));
    }  catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD UNOSA BROJEVA!!!",true);
                          out.println(h.prikaziStranu());  out.close(); return 0;}
```

generisaće se izuzeci ukoliko podaci koji su prihvaćeni preko ekranske forme nisu brojevi.

Nakon obrade greške program nastavlja normalno da se izvršava (u suprotnom ako se ne obradi, kao što je malopre rečeno, prekida se izvršenje programa). Ukoliko se zna da će u okviru metode neke klase da bude napravljen izuzetak, koji neće biti obrađen u toj metodi, potpis metode se proširuje sa naredbom throws. Iza naredbe throws se navode imena klasa očekivanih izuzetaka koji neće biti obrađeni u toj metodi. Na primer klasa *doPost(...)* objavljuje preko throws da je moguće da će se desiti izuzeci *ServletException ili IOException,* koji u metodi neće biti obrađeni*:*

```
protected void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
```

Ukoliko postoji potreba u programu se mogu praviti (bacati - throw) izuzeci pomoću naredbe **throw.**

Moguće je praviti sopstvene klase izuzetaka. Ono što predstavlja osnovu svake takve klase je klasa Exception. To znači da se pri pravljenju korisničke klase izuzetka nasleđuje klasa Exception. Svaki izuzetak koji se napravi u programu mora se obraditi inače će program prekinuti da se izvršava.
U svom standardnom paketu *java.lang*, Java definiše više klasa izuzetaka.

Pitanja:

1. Da li se generisani (bačeni) izuzetak mora obraditi?
2. Šta se događa ukoliko se ne obradi izuzetak?
3. Kako se oglašava izuzetak u metodi?

Primer obrade izuzetaka kod servleta

Korisnički zahtev Servlet3: Napraviti HTML dokument preko koga će biti prihvaćena dva broja. Navedeni brojevi treba da budu poslati do servleta koji će ih prihvatiti, izračunati njihov zbir i prikazati odgovarajući izveštaj. Obezbediti obradu izuzetaka koji se mogu desiti ukoliko format brojeva koji su poslati nije odgovarajući.

Prave se sledeće datoteke:

a) *Servlet3.html* koja će da prihvati podatke i da pozove servlet.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1252">
<meta http-equiv="Content-Language" content="en-us">
</head>

<body bgcolor="#C0C0C0" text="#000080">

<p>Primer: Napraviti HTML dokument preko koga &#263;e biti prihva&#263;ena dva broja koji treba da se saberu.</p>
<form action="http://127.0.0.1:8084/ServletPrimer3/Servlet3" method="POST">

     <dd> <b>ZBIR DVA BROJA:</b> <br> <br>
     <i>Broj1</i>    <input type="text" size="15" name="Broj1"> <br><br>

     <i>Broj2</i>   <input type="text" size="15" name="Broj2"><br>

    <dd>   <p align="left"><input type="submit" value="Racunaj zbir"></p>
</form>
<hr>
<font size="1"><i>
<b>Author: Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade   Copyright © 2005 All rights reserved.
</b>

</body>
</html>
```

b) *Servlet3.java* koja će da implementira servlet.

```
import java.net.*;
import java.text.*;
import java.io.*;
import WebKlase.*;

import javax.servlet.*;
import javax.servlet.http.*;

public class Servlet3 extends HttpServlet {

   protected int doPost(HttpServletRequest request, HttpServletResponse response)
    // Objavljivanje izuzetaka koji neće biti obrađeni.
   throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      double broj1=0;
      double broj2=0;
      HTML h = new HTML("Zbir brojeva:");
      // Generisanje izuzetka ukoliko format parametara Broj1 i Broj2 nije odgovarajući.
       try {
            broj1 = stringToFloat(request.getParameter("Broj1"));
            broj2 = stringToFloat(request.getParameter("Broj2"));
         }  catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD UNOSA BROJEVA!!!",true);
                        out.println(h.prikaziStranu());  out.close(); return 0;}
```

```
        double zbir = broj1 + broj2;
        DecimalFormat df = new DecimalFormat(".###");

       h.add(HTML.HEADING,"ZBIR DVA BROJA:",false);
       h.add(HTML.LINE,"",false);
       h.add(HTML.LINE,"",true);
          try {
               h.add(HTML.NORMAL, "Broj 1: " + df.format(broj1),true);
               h.add(HTML.NORMAL, "Broj 2: " + df.format(broj2),true);
               h.add(HTML.NORMAL, "Zbir brojeva: " + df.format(zbir),true);
              }
         catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD FORMATIRANJA BROJEVA!!!",true);
                        out.println(h.prikaziStranu());  out.close(); return 0;}
       h.add(HTML.LINE,"",true);
       h.add(HTML.AUTOR,"",false);
       out.println(h.prikaziStranu());
       out.close();
       return 1;
    }

    public static float stringToFloat(String ulazniString) throws NumberFormatException
    { Float f = new Float(ulazniString);
      return f.floatValue();
    }


}
```

c) web.xml koja je opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>Servlet3</servlet-name>
    <servlet-class>Servlet3</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Servlet3</servlet-name>
    <url-pattern>/Servlet3</url-pattern>
  </servlet-mapping>
</web-app>
```

d) Context.xml koji je runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletPrimer3">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletPrimer3." suffix=".log" timestamp="true"/>
</Context>
```

d) Pored navedenih datoteka napravljena je još jedna datoteka HTML.java u kojoj se čuva klasa HTML koja pomaže da se pojednostavi pisanje HTML koda. Ona je izdvojena kao nezavisna komponenta i kao takva je uključena u ovu aplikaciju. Ona ima sledeći sadržaj:

```
// Klasa koja olaksava pisanje HTML koda. Pojedine naredbe HTML koda su svedene
// na metode klase HTML.
public class HTML
{
// Definisanje konstanti,koje ce odrediti HTML naredbe koje ce biti izvrsene.
    public static final int NORMAL = 0;
    public static final int HEADING = 1;
    public static final int LINE = 2;
    public static final int AUTOR = 3;
    public static final int POZIVJSP = 4;

// Bafer koji se puni sa HTMl kodom.
    public StringBuffer buf;

// Konstruktor metoda koja puni bafere sa HTML zaglavljem i naslovom.
```

```
   public HTML(String naslov)
   {
    buf = new StringBuffer(4096);
    this.buf.append("<HTML><HEAD><TITLE>");
    this.buf.append(naslov);
    this.buf.append("</TITLE></HEAD><BODY>");
   }

// Metoda koja u zavisnosti od parametra stil puni bafer sa odredjenim HTNL kodom.
   public void add(int stil, String tekst, boolean sledecired)
   {
      switch(stil)
      {

       // Puni se bafer sa tekstom koji je poslat.
       case NORMAL: this.buf.append(tekst);
                    break;
        // Puni se bafer sa tekstom koji se odnosi na zaglavlje.
        case HEADING: this.buf.append("<H0><b>");
                    this.buf.append(tekst);
                    this.buf.append("</H0></b>");
                    break;
         // Puni se bafer sa znakom koji ukazuje na novu podvucenu liniju.
         case LINE:   this.buf.append("<HR>");
                    break;
         // Puni se bafer sa informacijama o autoru programa.
          case AUTOR:
                    this.buf.append("<b><font size='1'><i>");
                    this.buf.append("Author: Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade; Copyright © 2005
All rights reserved</b>");
                    break;
           // Puni se bafer sa pozivom zeljene jsp strane.
           case POZIVJSP:this.buf.append("<jsp:include page=" + tekst + "/>");
                    break;

          default: break;
       }
     // <BR> dodaje prazan red
     if (sledecired)
     { this.buf.append("<BR>");
     }
   }

// Genericka metoda za poziv servleta.
 public void add(String IP, String MetodaPrenosa, String poravnjanje, String tipDugmeta, String tekstDugmeta,String
nevidljivoPolje)
   {     this.buf.append("<form action ='" + IP +"' method='" + MetodaPrenosa + "'<p align='" + poravnjanje + "'><input type
         ='" + tipDugmeta +  "' value = '" + tekstDugmeta + "'>" + nevidljivoPolje + "</p></form>");
   }

// Puni bafer sa znacima koji oznacavaju kraj HTML datoteke i vraca kompletan sadraj bafera.
     public String prikaziStranu()
     {
       StringBuffer buf1 = this.buf.append("</BODY></HTML>");
       buf = new StringBuffer(4096);
       return buf1.toString();
     }
 // Prazni se postojeci sadrzaj bafera.
     public void Isprazni()
     {buf = new StringBuffer(4096);}

}
```

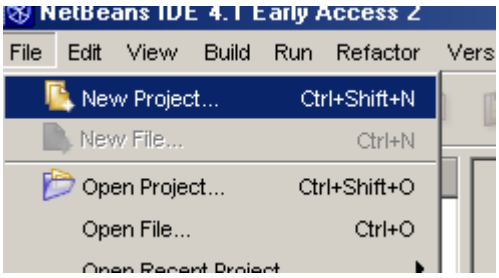
Zadatak ZServletOI1: Napisati program koji ce da generise izuzetak pri deljenju broja sa 0, koji ce nakon toga biti obradjen. Koristiti klasu *ArithmeticException* da se uhvati izuzetak.

2.1.4 Kreiranje biblioteke Javinih klasa i njeno povezivanje sa Web aplikacijom

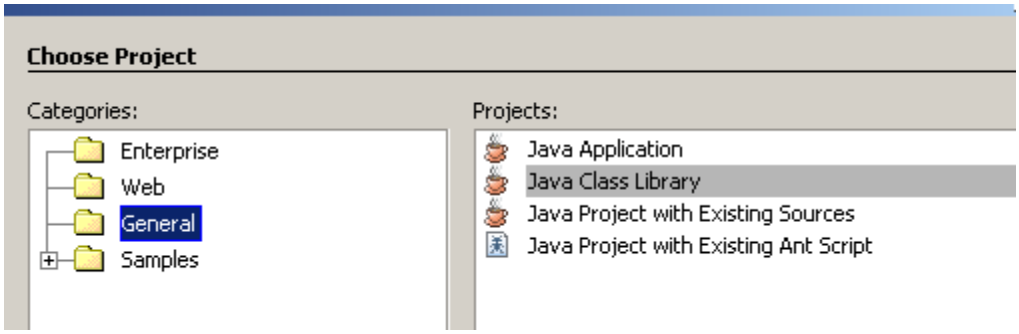
Biblioteka klasa sadrži klase koje se mogu uključiti u bilo koju aplikaciju. Objasnićemo preciznu proceduru kreiranja biblioteke Javinih klasa i proceduru povezivanja biblioteke klasa sa Web aplikacijom.

Kreiranje biblioteke Javinih klasa
Kod NB4.1 biblioteka klasa se pravi na sledeći način:

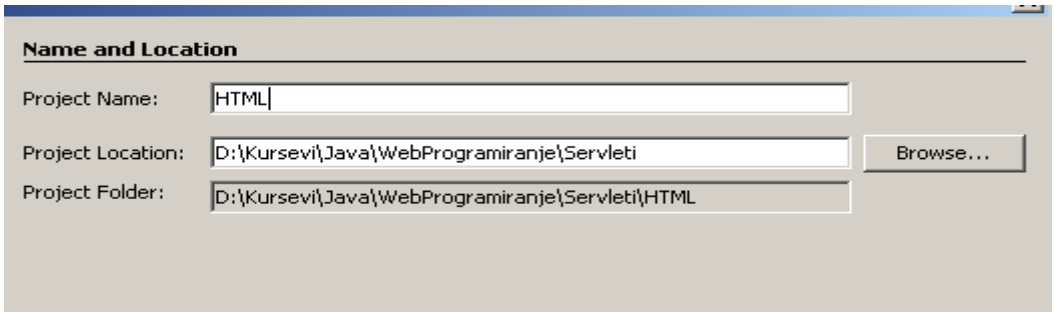
a) Otvora se novi projekat.



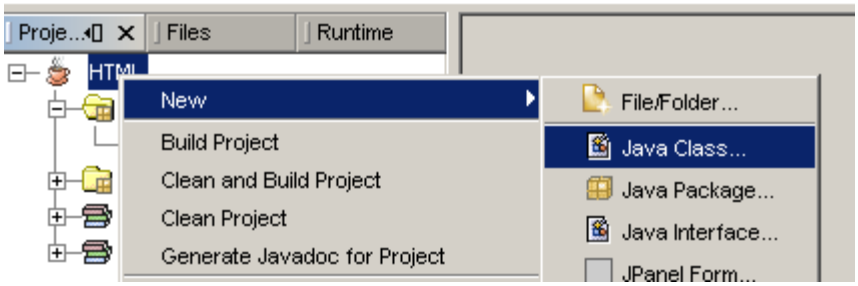
b) Bira se Java Class Library projekat. Pritiska se dugme *Next* koje je na dnu otvorene forme.



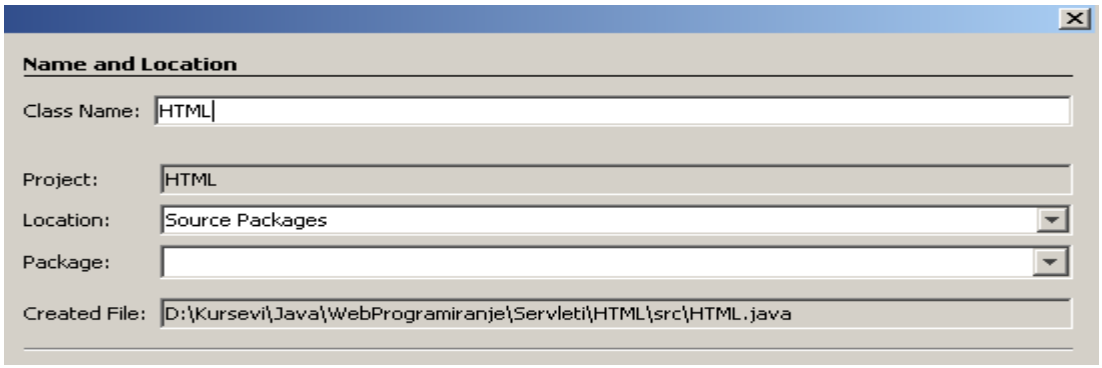
c) Unosi se ime projekta (*HTML*). Pritiska se dugme *Finish* koje je na dnu otvorene forme.



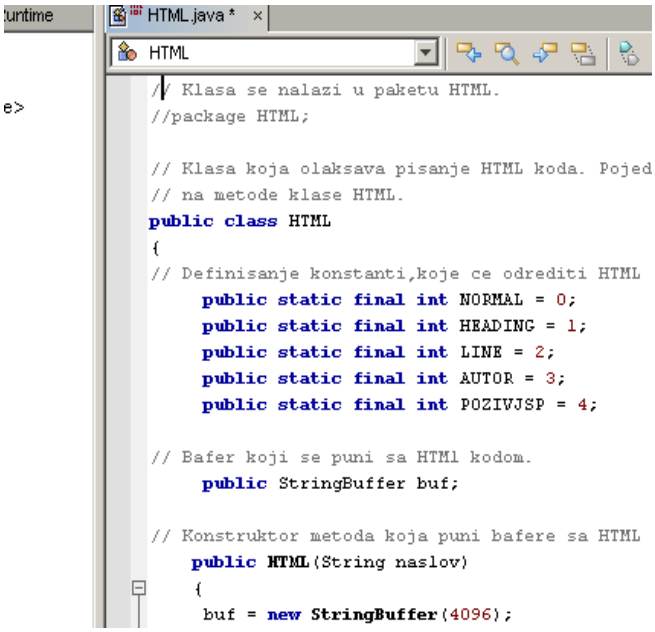
d) Kreira se nova klasa.



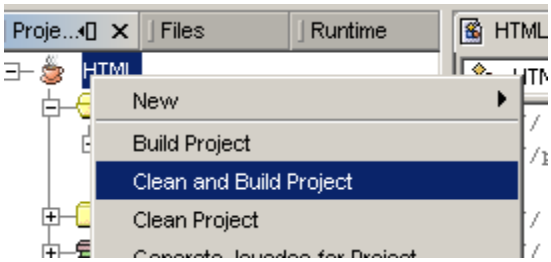
e) Unosi se ime nove klase(*HTML*). Pritiska se dugme *Finish* koje je na dnu otvorene forme.



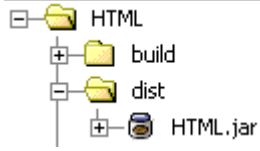
f) Programira se klasa HTML.



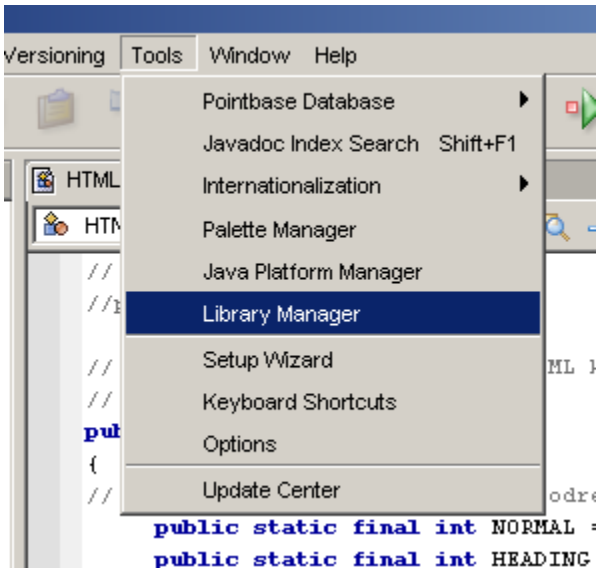
g) Izborom opcije *Clean and Build Project* se kompajlira klasa i pravi se *HTML.jar* datoteka.



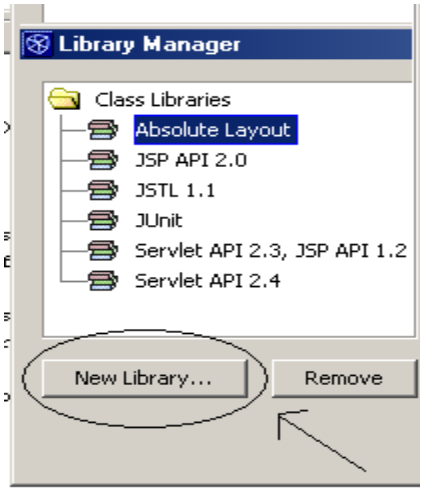
Datoteka *HTML.jar* se nalazi u folderu *dist* u odnosu na osnovni folder *HTML*.



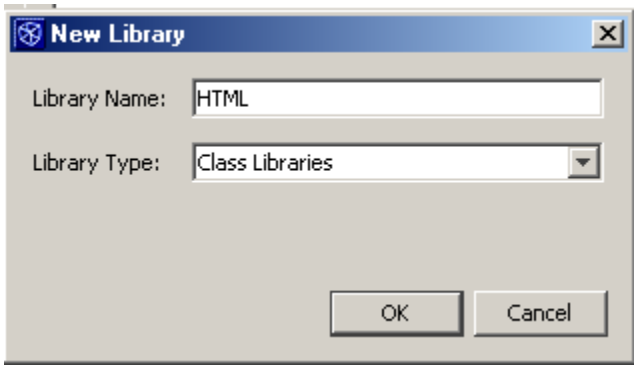
h) Bira se *Tools/Library Manager*



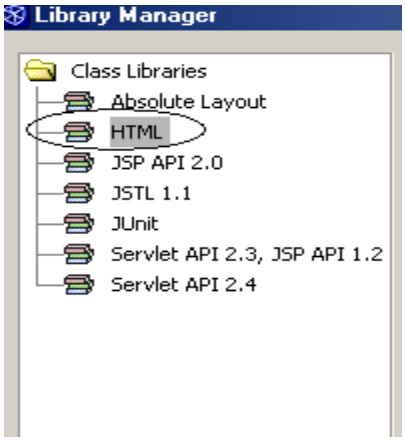
i) Pritiska se dugme *New Library*



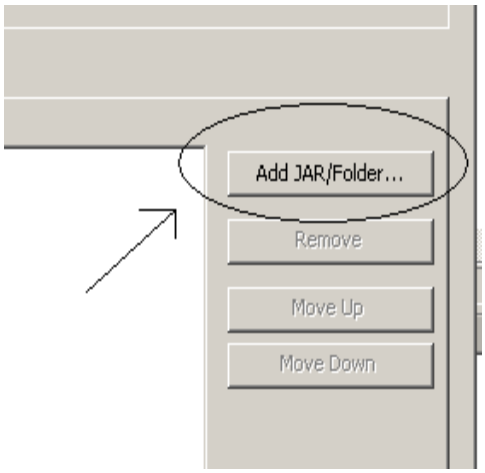
j) Unosi se ime nove biblioteke (*HTML*).



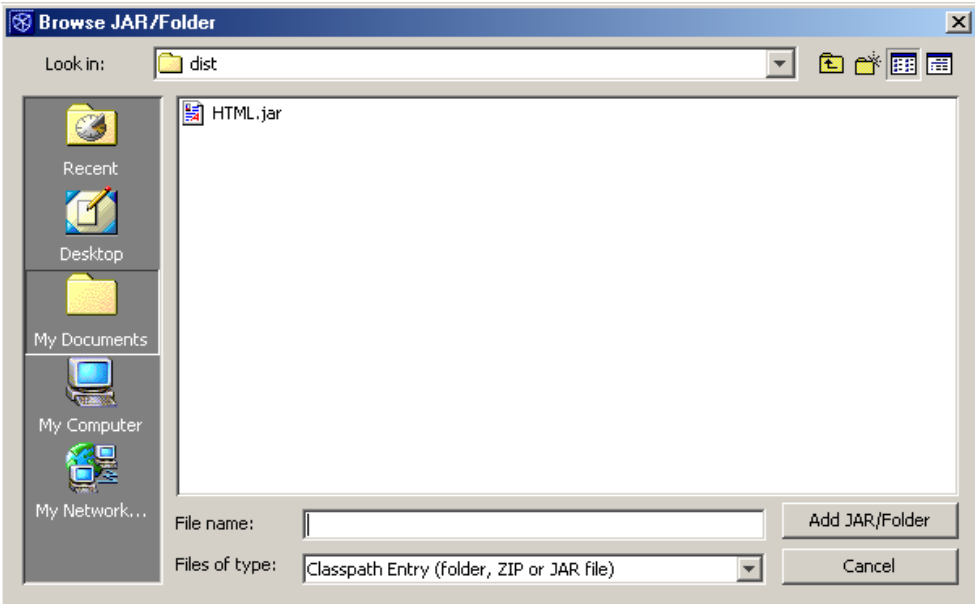
Pritiska se dugme *OK* i tada se dobija:



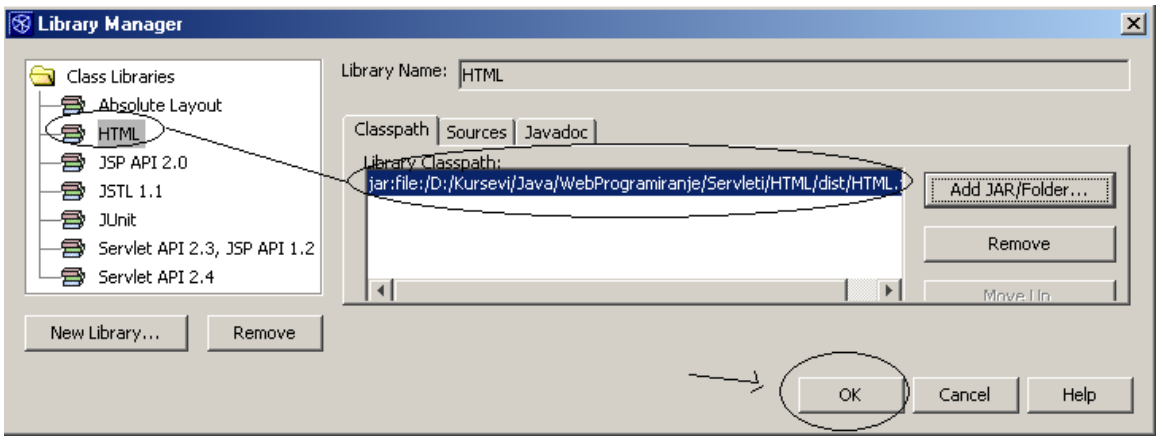
k) Pritiska se dugme Add JAR/Folder.



l) Bira se folder gde se nalazi HTML.jar datoteka. Pritiska se dugme *Add JAR/Folder*.



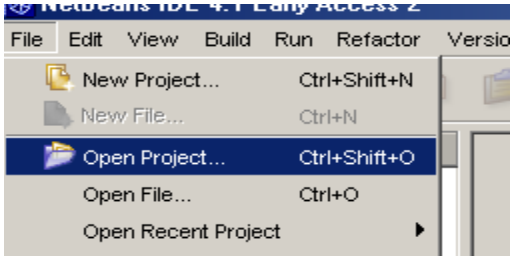
Dobija se forma gde se vidi da je uneta nova biblioteka (*HTML*) sa putanjom do HTML.jar datoteke u kojoj se nalazi klasa HTML.class. Na kraju se pritiska dugme *OK* kojim se potvrđuje da je kreirana nova biblioteka.



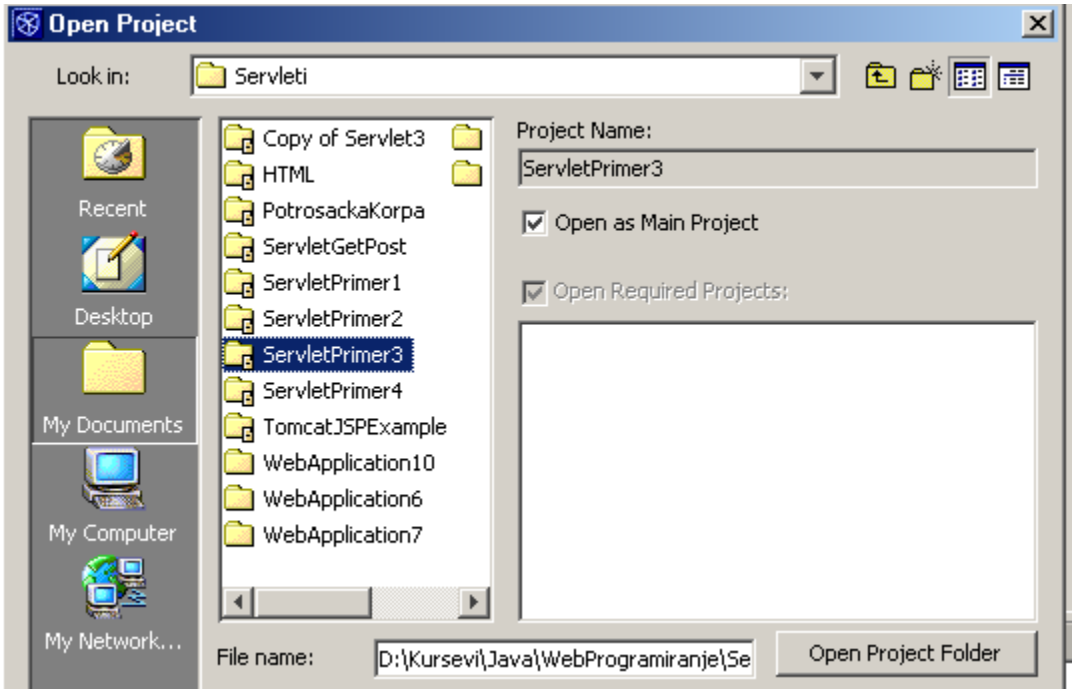
Povezivanje aplikacije sa bibliotekom klasa

Kod NB4.1 biblioteka klasa se povezuje sa Web aplikacijom na sledeći način:

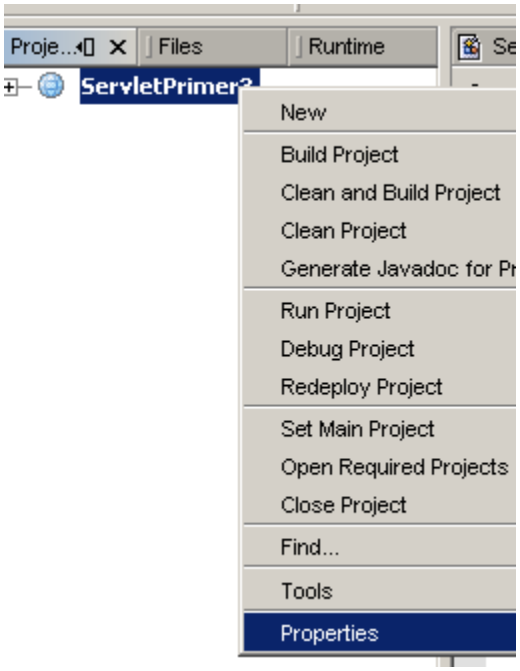
a) Bira se opcija *File/Open Project.*



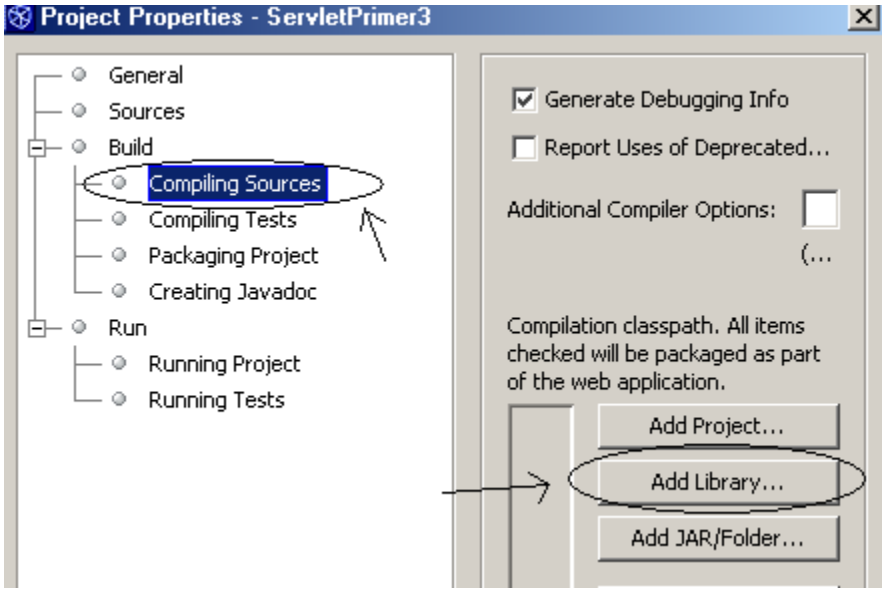
b) Otvara se postojeći projekat (*ServletPrimer3*). Prvo se selektuje a zatim se klikće dugme *Open Project Folder.*



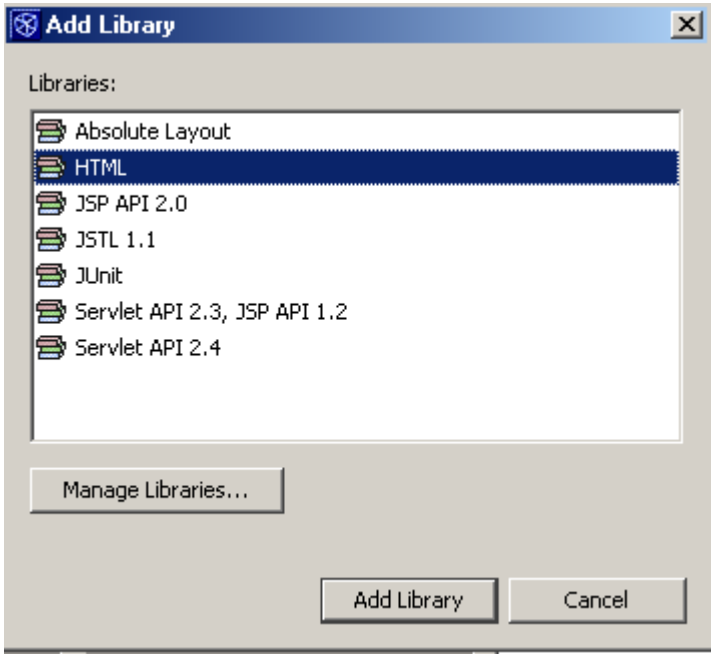
c) Bira se projekat (*ServletPrimer3*) i pritiska se desni klik. Bira se opcija *Properties.*



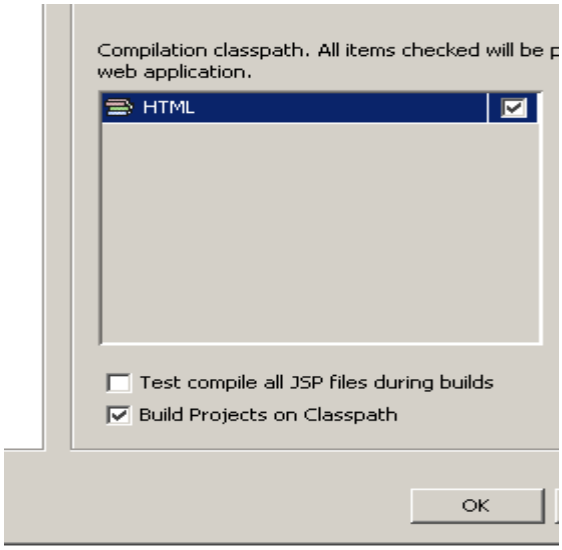
d) Bira se opcija *Compiling Sources.* Nakon toga se pritiska dugme *Add Library.*



e) Bira se željena bibiloteka. Pritiska se dugme *Add Library.*



f) Pritiska se dugme *OK* da bi se potvrdilo povezivanje aplikacije sa *HTML* bibliotekom klasa.



Zadatak ZPovWASKom1: Napraviti sopstveni primer koji će povezati postojeću Web aplikaciju sa novom bibliotekom klasa.

2.1.5 Sesije

Postoje dva tipa internet komunikacionih protokola: ***Stateless*** i ***Stateful***. Oni se razlikuju po načinu uspostavljanja konekcije između klijenta i servera. *Telnet* i *FTP* , npr. koriste *stateful* tip protokola. Scenario *stateful* tipa protokola je sledeći:

a) Klijent se konektuje na server.
b) Klijent izvršava seriju operacija preko konekcije.
c) Klijent prekida konekciju (diskonekcija) sa serverom.

Dokle god postoji konekcija između klijenta i servera, za *stateful* tip protokola, server može da prihvata i grupiše sve zahteve jednog klijenta[15]. U internet žargonu, za navedeni slučaj, se kaže da konekcija ima ***stanje.*** To znači da server pamti *ko je klijent i šta je od uradio od trenutka uspostavljanja konekcije.*

Kod HTTP-a je *stateless* protokol koji je zasnovan na *zahtev-odgovor(request-response model)* modelu[16]. To znači da je svaki par *zahtev-odgovor* izolovana transakcija. Kod navedenog tipa protokola server ne zna da li serija zahteva dolazi od istog ili od različitih klijenata. Server takođe ne zna da li su zahtevi na neki način međusobno povezani ili ne. Kada se pravi Web aplikacija o navedenom nedostatku *zahtev-odgovor* modela mora da se vodi računa. To znači da Web aplikacija treba da održi ***stanje*** klijenta na serveru, ukoliko postoji objektivna potreba za tim.

Scenario *stateless* tipa protokola je sledeći:

a) Klijent se konektuje na server i šalje mu jedan zahtev.
b) Server izvršava zahtev i šalje odgovor.
c) Klijent prekida konekciju sa serverom.

Sesija se može definisati kao serija povezanih interakcija, koja se dešava u nekom periodu vremena, između jednog klijenta i web servera. Sesija može da sadrži više zahteva koji su upućeni do istog servleta ili se zahtevi mogu poslati do različitih web komponenti iste web aplikacije.
Pošto je HTTP stateless protokol, web server ne zna sa kojom sesijom komunicira. Postoje dva načina da on sazna koja je to sesija:

1) Svaki put kada klijent šalje zahtev on treba da se identifikuje. Tada se obrađuju podaci (pamte i čitaju) koji su povezani sa tim klijentom. Podaci se čuvaju u nekom skladištu podataka.
2) Server šalje klijentu podatke za identifikaciju. Klijent svaki put kada postavlja zahtev serveru šalje te podatke koji ga identifikuju.

Održavanje stanja klijenta na serveru je poznato po terminu ***praćenje sesije (session tracking).*** Postoji nekoliko pristupa koji se koriste kod praćenja sesije:

a) Kreiranje sakrivenih polja forme koji sadrže podatke.
b) URL-a uključuje dopunske parametre da identifikuje sesiju.
c) Korišćenje kolačića (*cookies*).
d) Korišćenje određenih metoda Servlet API-a.

[15] Analogija sa programiranjem: Kada program (konekcija) počne da se izvršava on ima pristup do globalnih promenljivih (stanje konekcije), kojima može u vremenu da menja vrednost (više operacija može da se izvrši). Kada se završi program (prekid konekcije), gubi se informacija o vrednostima globalnih promenljivih(gubi se stanje konekcije).

[16] Analogija sa programiranjem: Kada program počne da se izvršava on može da pozove neku metodu (konekcija). Kada metoda počne da se izvršava program ima pristup do lokalnih promenljivih (stanje konekcije). Kada se metoda izvrši (prekid konekcije), program gubi informaciju o vrednostima lokalnih promenljivih. To znači da kada program opet pozove metodu (nova konekcija) vrednost lokalnih promenljivih neće biti sačuvana.
U programskom jeziku C stavljamo *static* ključnu reč isped imena lokalne promenljive, kada želimo da obezbedimo da lokalna promenljiva sačuva svoju vrednosti pri svakom sledećem pozivu funkcije u kojoj se ona nalazi. Navedeni efekat treba i kod Web aplikacije da se obezbedi ukoliko se želi pratiti stanje sesije.

Praćenje sesija preko servleta

Sesija se pokreće identifikovanjem klijenta koji je pozvao servlet (*jedna sesija je vezana za jednog klijenta*). Početno stanje sesije je određeno identifikacionim podacima klijenta. Nakon toga klijent šalje zahtev koji može da promeni početno stanje u neko novo (tekuće) stanje sesije. Svaki sledeći poziv može da promeni tekuće stanje u neko novo stanje sesije. **Trajanje sesije je određeno stanjem sesije**. **Kada sesija izgubi stanje tada ona prestaje da se izvršava (**prestaje da postoji**).**
Server identifkuje sesiju i prati njeno stanje na sledeći način:
Klijent šalje podatke za identifikaciju do servleta. Podaci za identifikaciju klijenta se mogu poslati na 2 načina:

a. Klijent preko *stringa zahteva* šalje podatke za identifikaciju, koji su fiksirani u URL-u. (*Primer ServletSesURL1*).
b. Klijent unosi identifikacione podatke preko forme za logovanje a zatim šalje podatke za identifikaciju do servera preko post metode(*Primer ServletSesLog1*).

Kada server prihvati zahtev, on preko podataka identifikacije zna ko je klijent, odnosno zna koja je sesija aktivna. Stanje sesije je vezano za identifikacione podatke klijenta koji se povezuje sa servletom. Pored toga stanje sesije može biti određeno i dopunskim podacima koje generiše klijent i/ili servlet.
Servlet nakon identifikovanja klijenta, u tekućoj sesiji, može da mu pošalje nove identifikacione podatke, koji će biti validni, dok bude aktivna tekuća sesija. Nove identifikacione podatke[17] servlet šalje klijentu na nekoliko načina:

- ***Prepisuje (rewriting)*** URL sa HTML strane koja ga poziva, dodajući na njegov kraj nove identifikacione podatke klijenta(*Primer ServletSesURL2*).
- Puni HTML stranu koja ga poziva sa sakrivenim poljima za identifikaciju(*Primer ServletSesSakrPolja1*).
- Šalje kolačiće (cookies) do klijenta koji služe za novu identifikaciju klijenta(*Primer ServletSesKolacic1*).
- Servlet kreira sesiju (daje joj broj) i šalje ga do klijenta za novu identifikaciju klijenta*Primer (Primer: ServletSesObjekat1)*.

Stanje sesije se može čuvati transijentno i perzistentno. Transijentnost stanja sesije se obezbeđuje na nekoliko načina:

- Servlet salje nevidljiva polja, koja čuvaju stanje sesije, do klijenta(*Primer ServletSesSakrPolja2*).
- Kada servlet kreira sesiju (objekat sesije) on može preko nje da čuva stanje sesije*(Primer: ServletSesObjekat1)*.

Perzistentnost stanja sesije se obezbeđuje preko:

- Datoteke ili baze podataka(*Primer ServletSesDat1, ServletSesBP1*).
- Servleta koji generišu i šalju kolačiće, koja čuvaju stanje sesije, do klijenta(*Primer ServletSesKolacic2*).

Kada se stanje čuva transijentno trajanje sesije je ograničeno vremenom trajanja identifikacionih podataka klijenta i izvršavanjem čitača koji je pozvao servlet. Kada se čitač zatvori stanje sesije se gubi. Kada se stanje čuva perzistentno trajanje sesije je ograničeno vremenom trajanja identifikacionih podataka klijenta, vremenom trajanja kolačića koji su poslati do klijenta i dostupnošću skladišta podataka koja čuvaju stanje sesije.

[17] Novi identifikacioni podaci mogu da odrede oblast delovanja klijenta.

Primer ServletSesLog1

Korisnički zahtev: Napraviti HTML dokument preko koga će klijent da pošalje svoj identifikacione podatke (korisnicko ime i lozinku) do servera.

Prave se sledeće datoteke:

e) *ServletSesLog1.html* koja će da prihvati podatke i da pozove servlet.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<p>Korisnicki zahtev:Napraviti HTML dokument preko koga ce klijent da pošalje svoj identifikacione podatke (korisnicko
ime i lozinku) do servera. Identifikacione podatke klijent treba da unosi preko forme za logovanje.</p>

<form action="http://127.0.0.1:8084/ServletSesLog1/ServletSesLog1" method="POST">
<dd> <b>IDENTIFIKACIONI PODACI</b> <br> <br>
<dd><table border =1>
        <td> Korisnicko ime:
        <td>  <input type="text" name="ImeKorisnika" size="16">
        <tr>
        <td> Lozinka:
        <td> <input type="password" name="Lozinka" size="16">
        <tr>
      </table>
  <br><br>
 <dd><dd><dd> <p align="left"><input type="submit" value="Login"></p>
</form>

<hr>
<font size="1"><i>
<b>Author: Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

f) *ServletSesLog1.java* koja će da implementira servlet.

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class ServletSesLog1 extends HttpServlet {
   // Atributi servleta
   HTML h = new HTML("Unos identifikacionih podataka novog korisnika:");
   PrintWriter out;
   Korisnik kor[];

   // Metode setvleta
   protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
    { // Kreiranje klijenata koji se mogu povezati (ulogovati) sa servletom.
      kor = new Korisnik[3];
      kor[0]=new Korisnik("pera","121");
      kor[1]=new Korisnik("mile","122");
      kor[2]=new Korisnik("laza","123");


      response.setContentType("text/html");
      out = response.getWriter();
      // Prihvatanje parametara koje je poslao klijent.
      String ImeKorisnika = request.getParameter("ImeKorisnika");
      String Lozinka = request.getParameter("Lozinka");



      h.add(HTML.HEADING,"IDENTIFIKOVANJE KORISNIKA:",false);
      h.add(HTML.LINE,"",false);
```

```
        // Ukoliko klijent posalje prazna polja tada ce ga servlet odbiti sa odgovarajucim upozorenjem.
       if ((ImeKorisnika.equals("")) || (Lozinka.equals("")))
       { h.add(HTML.NORMAL,"Unesite ime korisnika ili lozinku!!!",true); out.println(h.prikaziStranu()); return 0;}

       // Polazi se od pretpostavke da klijent koji se loguje ne postoji.
       boolean signal = false;
       for(int i =0; i<3; i++)
         { if ((ImeKorisnika.equals(kor[i].ImeKorisnika)) && (Lozinka.equals(kor[i].Lozinka)))
             {signal = true; break;} // Klijent koji se loguje postoji medju autorizovanim  klijentima.
         }

       // Servlet salje odgovarajucu poruku klijentu u zavisnosti od toga da li je on autorizovan ili ne.
       if(signal==true)
         { h.add(HTML.NORMAL,"Korisnik je autorizovan",true);}
       else
         { h.add(HTML.NORMAL,"Korisnik nije autorizovan",true);}

       h.add(HTML.LINE,"",true);
       h.add(HTML.AUTOR,"",false);
       out.println(h.prikaziStranu());
       out.close();
       return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
       obradi(request, response);
    }

}

// Klasa Korisnik
class Korisnik implements Serializable
{ String ImeKorisnika;
  String Lozinka;

  public Korisnik(String ImeKorisnika1,String Lozinka1)
       { ImeKorisnika = new String(ImeKorisnika1);
         Lozinka = new String(Lozinka1);
        }
}
```

g) web.xml - opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesLog1</servlet-name>
    <servlet-class>ServletSesLog1</servlet-class>
  </servlet>
    <servlet-mapping>
     <servlet-name>ServletSesLog1</servlet-name>
     <url-pattern>/ServletSesLog1</url-pattern>
  </servlet-mapping>
  </web-app>
```

d) *Context.xml* - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesURL1">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesURL1." suffix=".log" timestamp="true"/>
</Context>
```

Primer ServletSesURL2

Korisnički zahtev*: Napraviti HTML dokument preko koga će klijent da se poveže sa servletom koji treba da prepiše (rewriting) URL sa HTML strane koja ga poziva, dodajući na njegov kraj poziv drugog servleta i nove identifikacione podatke klijenta.*

Prave se sledeće datoteke:

a) ServletSesURL2.html koja će da pozove servlet.

```
<html>
<body>
<p>Napraviti HTML dokument preko koga ce klijent da se poveže sa servletom koji treba da prepiše (rewriting) URL sa
HTML strane koja ga poziva,
dodajuci na njegov kraj poziv novog servleta i nove identifikacione podatke klijenta.</p>

<form action="http://127.0.0.1:8084/ServletSesURL2/ServletSesURL2?ImeKorisnika=pera" method="POST">
<p align="left"><input type="submit" value="Pozovi servlet"></p>
</form>

</body>
</html>
```

b) ServletSesURL2.java koja će da implementira servlet koji se poziva preko HTML strane.

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;.

public class ServletSesURL2 extends HttpServlet
  {
      HTML h = new HTML("Prepisivanje URL-a sa HTML strane koja poziva servlet");
      PrintWriter out;

     protected int obradi(HttpServletRequest request, HttpServletResponse response) throws ServletException,
                                                                                    IOException {
      String ImeKorisnika;
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      ImeKorisnika = request.getParameter("ImeKorisnika");

       if (ImeKorisnika.equals("pera") == true)
          { h.add(HTML.NORMAL, "Korisnik ce biti autorizovan", true);
            // Prepisivanje URL- sa HTML strane koja poziva servlet.
            h.add("http://127.0.0.1:8084/ServletSesURL2/ServletSesURL21?ImeKorisnika=laza","POST",
                  "left","submit","Pozovi servlet","");
          }
         else
          { h.add(HTML.NORMAL, "Korisnik nije autorizovan", true);   }

       h.add(HTML.LINE,"",true);   h.add(HTML.AUTOR,"",false);
       out.println(h.prikaziStranu());  out.close();
       return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    obradi(request, response);
   }

}
```

c) ServletSesURL21.java koja će da implementira servlet koji se poziva preko HTML strane koju je generisao servlet *ServletSesURL2*.

```
import java.io.*;
import java.net.*;
```

```
import javax.servlet.*;
import javax.servlet.http.*;


public class ServletSesURL21 extends HttpServlet {

   HTML h = new HTML("Prepisivanje URL-a sa HTML strane koja poziva servlet:");
   PrintWriter out;


   protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {

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

      ImeKorisnika = request.getParameter("ImeKorisnika");

       if (ImeKorisnika.equals("laza") == true)
           { h.add(HTML.NORMAL, "Korisnik je autorizovan", true);
           }
          else
           { h.add(HTML.NORMAL, "Korisnik nije autorizovan", true);
           }
    // Ponovno generisanje URL-a koji ce da pozove tekuci servlet.
      h.add("http://127.0.0.1:8084/ServletSesURL2/ServletSesURL21?ImeKorisnika=laza", "POST",
              "left","submit","Pozovi servlet","");

       h.add(HTML.LINE,"",true);
       h.add(HTML.AUTOR,"",false);
       out.println(h.prikaziStranu());
       out.close();
       return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    obradi(request, response);
   }

}
```

h) web.xml - opisivač rasporeda aplikacije

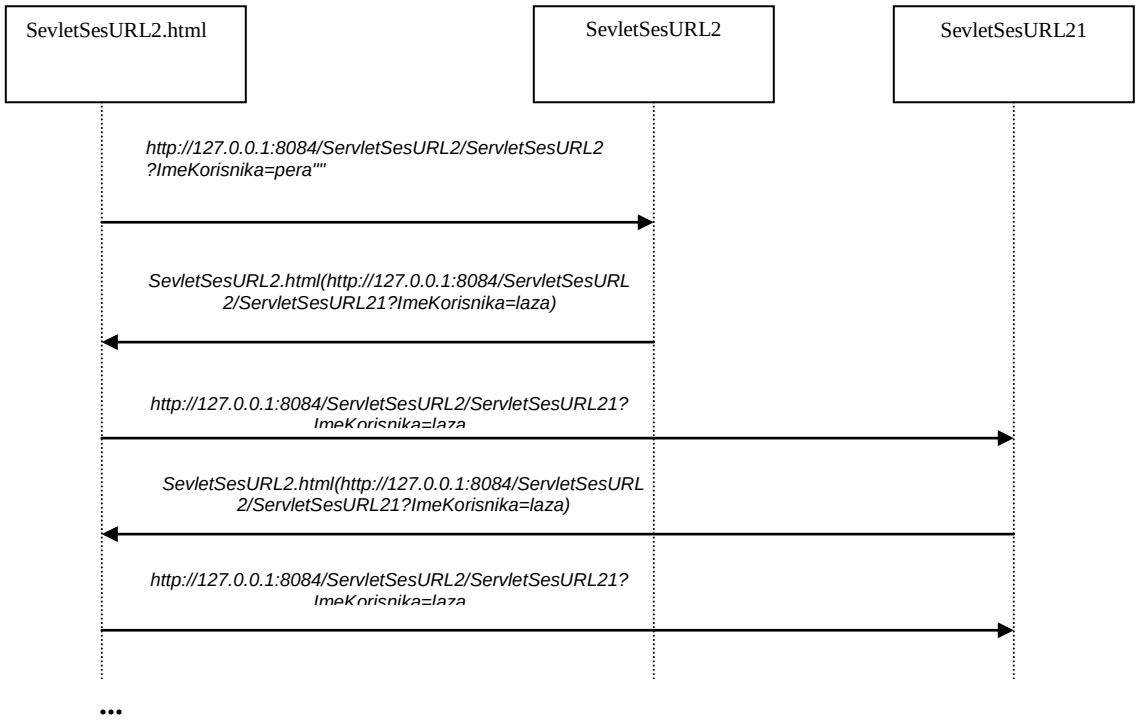
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesURL2</servlet-name>
    <servlet-class>ServletSesURL2</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>ServletSesURL21</servlet-name>
    <servlet-class>ServletSesURL21</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletSesURL2</servlet-name>
    <url-pattern>/ServletSesURL2</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ServletSesURL21</servlet-name>
    <url-pattern>/ServletSesURL21</url-pattern>
  </servlet-mapping>
</web-app>
```

e) Context.xml - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesURL2">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesURL2." suffix=".log" timestamp="true"/>
</Context>
```

Scenarijo komunikacije HTML strane i servleta za primer ServletSesURL2

ServletSesURL2.html preko čitača poziva servlet *ServletSesURL2* (*Identifikacioni podatak – ImeKorisnika=pera*) koji generiše novu *ServletSesURL2.html* datoteka kod koje je ***prepisan (rewriting) URL*** koji poziva servlet i dodat je novi identifikacioni broj klijenta (*ImeKorisnika = laza*). Sada *ServletSesURL2.html* poziva *ServletSesURL21* koji generiše *ServletSesURL2.html* koja je identična predhodno generisanoj *ServletSesURL2.html* datoteci. Poslednji korak se rekurzivno ponavlja dokle god je aktivan servlet *ServletSesURL21* i čitač.



...

Primer ServletSesSakrPolja1.
Korisnički zahtev*: Napraviti HTML dokument preko koga će klijent da se poveže sa servletom koji će mu vratiti novi identifikacioni podatak preko nevidljivog polja.*

Prave se sledeće datoteke:
a) ServletSesSakrPolja1.html koja će da pozove servlet.

```
<html>
<body>
<p>Korisnicki zahtev:Napraviti HTML dokument preko koga ce klijent da se poveze sa servletom koji ce mu vratiti
novi identifikacioni podatak preko nevidljivog polja </p>
<form action="http://127.0.0.1:8084/ServletSesSakrPolja1/ServletSesSakrPolja1" method="POST">
<p align="left"><input type="submit" value="Pozovi servlet"></p>
</form>
</body>
</html>
```

b) ServletSesSakrPolja1.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletSesSakrPolja1 extends HttpServlet {

   HTML h = new HTML("Identifikacija klijenta preko sakrivenih polja:");
   PrintWriter out;

    protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {

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

      String ImeKorisnika = request.getParameter("ImeKorisnika");

      //Kada se servlet prvi put pozove parametar ImeKorisnika imace null vrednost.
      if (ImeKorisnika == null)
         {h.add(HTML.NORMAL, "Korisnik ce biti autorizovan.", true);}
      else
        // Ovo ce se desiti kod drugog poziva servleta kada klijent dobije sakriven identifikacioni podatak,
         if (ImeKorisnika.equals("pera") == true)
           { h.add(HTML.NORMAL, "Korisnik je autorizovan.", true);
           }
      // Klijent dobija od servleta sakriveni identifikacioni podatak (ImeKorisnika=pera)
      h.add("http://127.0.0.1:8084/ServletSesSakrPolja1/ServletSesSakrPolja1", "POST", "left","submit","Pozovi
            servlet","<Input type = 'hidden' Name = 'ImeKorisnika' Value='pera'>");

       h.add(HTML.LINE,"",true);
       h.add(HTML.AUTOR,"",false);
       out.println(h.prikaziStranu());
       out.close();
       return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    obradi(request, response);
   }
}
```

c) web.xml - opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesSakrPolja1</servlet-name>
```

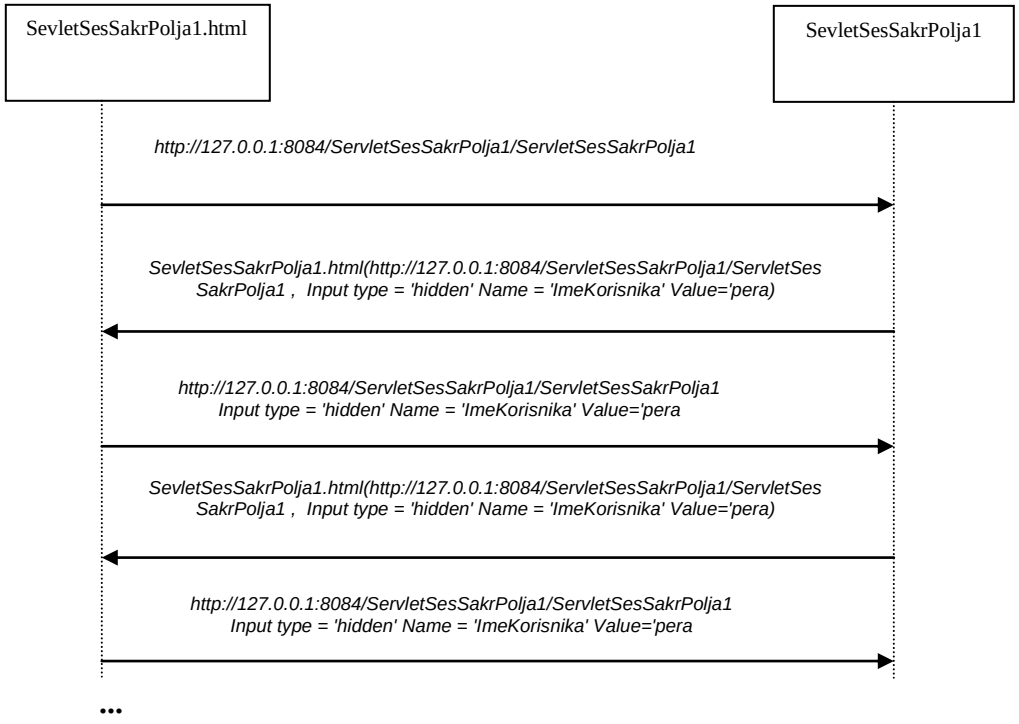
```
  <servlet-class>ServletSesSakrPolja1</servlet-class>
 </servlet>
  <servlet-mapping>
   <servlet-name>ServletSesSakrPolja1</servlet-name>
   <url-pattern>/ServletSesSakrPolja1</url-pattern>
 </servlet-mapping>
 </web-app>
```

d) Context.xml - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesSakrPolja1">
  <Logger  className="org.apache.catalina.logger.FileLogger"  prefix="ServletSesSakrPolja1."  suffix=".log"
timestamp="true"/>
</Context>
```

Scenarijo komunikacije HTML strane i servleta za primer ServletSesSakrPolja1

ServletSesSakrPolja1.html preko čitača poziva servlet *ServletSesSakrPolja1* koji generiše *ServletSesSakrPolja1.html* datoteku koja dobija **sakrivena** polja za identifikaciju klijenta. Pri sledećem pozivu servleta *ServletSesSakrPolja1* od strane *ServletSesSakrPolja1.html* prosleđuje se sakriveni identifikacioni podatak na osnovu koga servlet vrši autorizaciju klijenta. Dokle god traje sesija (dokle god se izvršavaju servlet i čitač) ponavljaće se koraci poziva servleta, slanja sakrivenih identifikacionih podataka do servleta i generisanja HTML strane sa sakrivenim identifikacionim podatkom od strane servleta.



Primer ServletSesKolacic1.

Korisnički zahtev*: Napraviti HTML dokument preko koga će klijent da se poveže sa servletom koji će mu vratiti novi identifikacioni podatak preko kolacica (cookie).*

Prave se sledeće datoteke:

a) ServletSesKolacic1.html koja će da pozove servlet.

```
<html>
<body>
<p>Korisnicki zahtev: Napraviti HTML dokument preko koga ce klijent da se poveže sa servletom koji ce mu vratiti
novi identifikacioni podatak preko kolacica (cookie). </p>
<form action="http://127.0.0.1:8084/ServletSesKolacic1/ServletSesKolacic1" method="POST">
<p align="left"><input type="submit" value="Pozovi servlet"></p>
</form>

</body>
</html>
```

b) ServletSesKolacic1.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ServletSesKolacic1 extends HttpServlet {

   HTML h = new HTML("Identifikacija klijenta preko kolacica:");
   PrintWriter out;

   protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {

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

     boolean signal = false;
     int brojcok=0;
     // Pri prvom pozivu servleta co ce da dobije null jer kolacic jos nije stvoren i vezan za novu HTML stranu.
     Cookie co[]= request.getCookies();
     // Pri drugom i sledecim pozivima co je razlicito od 0 jer je napravljen kolacic i poslat je do korisnika.
     if (co !=null)
        brojcok = co.length; // odredjuje se broj kolacica.
      // Prolazi se kroz sve kolacice i proverava se da li neki od njih ima ime i vrednost jednaku autorizovanom klijentu.
     for(int i=0; i< brojcok; i++)
       { if(co[i].getName().equals("ImeKorisnika"))
         { ImeKorisnika = co[i].getValue();
           if(ImeKorisnika.equals("pera"))
             { h.add(HTML.NORMAL, "Korisnik je autorizovan.", true);
               signal = true;
             }
         }
       }

     // Pri prvom pozivu servleta pravi se novi kolacic.
     if (signal == false)
       {    h.add(HTML.NORMAL, "Korisnik nije autorizovan.", true);
            Cookie c = new Cookie("ImeKorisnika","pera"); // Pravi se kolacic.
            c.setMaxAge(30); // Odredjuje se posle koliko sekundi kolacic zastareva
            response.addCookie(c); // Kolacic se vezuje za odgovor koji pravi servlet.
       }

      h.add("http://127.0.0.1:8084/ServletSesKolacic1/ServletSesKolacic1", "POST", "left","submit","Pozovi
              servlet","");

      h.add(HTML.LINE,"",true);
      h.add(HTML.AUTOR,"",false);
      out.println(h.prikaziStranu());
      out.close();
      return 1;
   }
```

```
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    obradi(request, response);
   }

}
```

c) web.xml - opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesKolacic1</servlet-name>
    <servlet-class>ServletSesKolacic1</servlet-class>
  </servlet>
    <servlet-mapping>
     <servlet-name>ServletSesKolacic1</servlet-name>
     <url-pattern>/ServletSesKolacic1</url-pattern>
  </servlet-mapping>
  </web-app>
```

d) Context.xml - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesKolacic1">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesKolacic1." suffix=".log"
timestamp="true"/>
</Context>
```

Scenarijo komunikacije HTML strane i servleta za primer ServletSesKolacic1

ServletSesKolacic1.html preko čitača poziva servlet *ServletSesSakrKolacic1* koji generiše *ServletSesKolacic1.html* datoteku koja dobija **kolačić** za identifikaciju klijenta. Pri sledećem pozivu servleta *ServletSesKolacic1* od strane *ServletSesKolacic1.html* prosleđuje se kolačić (koji ima ulogu identifikacionog podatka) na osnovu koga servlet vrši autorizaciju klijenta. Dokle god traje sesija (trajanje sesije je odredjeno vremenom trajanja kolačića) ponavljaće se koraci poziva servleta, slanja kolačića do servleta i generisanja HTML strane sa kolačićem(identifikacionim podatkom).

Primer ServletSesURL1

Korisnički zahtev: *Napraviti HTML dokument preko koga će klijent da se poveže sa servletom šaljući mu lozinku preko stringa zahteva.*

Prave se sledeće datoteke:

a) *ServletSesURL1.html* koja će da prihvati podatke i da pozove servlet.

```
<html>
<body>
<p>Korisnicki zahtev:Napraviti HTML dokument preko koga ce klijent da se poveze sa servletom saljuci mu lozinku
preko stringa zahteva</p>

<form action="http://127.0.0.1:8084/ServletSesURL1/ServletSesURL1?ImeKorisnika=pera" method="POST">
<p align="left"><input type="submit" value="Pozovi servlet"></p>
</form>

</body>
</html>
```

b) *ServletSesURl1.java* koja će da implementira servlet.

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class ServletSesURL1 extends HttpServlet {

   HTML h = new HTML("Pracenje sesija prihvatanjem stringa zahteva:");
   PrintWriter out;


   protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {


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

     String ImeKorisnika = request.getParameter("ImeKorisnika");

     // Ukoliko klijent direktno pozove servlet bez slanja parametra ImeKorisnika, tada ce servlet da napuni ImeKorisnika
     // sa null.
      if (ImeKorisnika == null)
       { h.add(HTML.NORMAL, "Pristup nije dozvoljen neautorizovanim  korisnicima", true);
       }
      else
       if (ImeKorisnika.equals("pera") == true)
          { h.add(HTML.NORMAL, "Ovaj korisnik je autorizovan.", true);
          }
         else
          { h.add(HTML.NORMAL, "Pristup nije dozvoljen neautorizovanim  korisnicima", true);
          }


      h.add(HTML.LINE,"",true);
      h.add(HTML.AUTOR,"",false);
      out.println(h.prikaziStranu());
      out.close();
      return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    obradi(request, response);
   }

}
```

c) *web.xml* - opisivač rasporeda aplikacije.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesURL1</servlet-name>
    <servlet-class>ServletSesURL1</servlet-class>
  </servlet>
    <servlet-mapping>
     <servlet-name>ServletSesURL1</servlet-name>
     <url-pattern>/ServletSesURL1</url-pattern>
  </servlet-mapping>
  </web-app>
```

d) *Context.xml* - runtime opisivač rasporeda.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesURL1">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesURL1." suffix=".log" timestamp="true"/>
</Context>
```

Primer ServletSesObjekat1

***Korisnički zahtev**: Napraviti HTML dokument preko koga će biti prihvaćeni podaci o naručenim knjigama. Naručene knjige treba proslediti do servleta tako da se pri svakom sledećem pozivu servleta pamte sve one knjige koje su bile naručene.*

Prave se sledeće datoteke:

a) *ServletSesObjekat1.html* koja će da prihvati podatke i da pozove servlet.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<p>Primer: Napraviti HTML dokument preko koga će biti prihvaćeni podaci o naručenim knjigama. Naručene knjige treba proslediti do servleta tako da se pri svakom sledećem pozivu servleta pamte sve one knjige koje su bile naručene. </p>
<form action="http://127.0.0.1:8084/ServletSesObjekat1/ServletSesObjekat1" method="POST">

<b>NARUCIVANJE KNJIGA-POTROŠACKA KORPA</b>
<br><br>

<p><i><b>PROIZVODI</b></i>
<p><input type="checkbox" name="proizvod" value="Applying UML and patterns"> Applying UML and patterns
<p><input type="checkbox" name="proizvod" value="Design patterns"> Design patterns
 <p><input type="checkbox" name="proizvod" value="Unified software devel. process"> Unified software devel. process
 <p><input type="checkbox" name="proizvod" value="Database"> Database
 <p><input type="checkbox" name="proizvod" value="J2EE 1.4 Tutorial"> J2EE 1.4 Tutorial
 <p align="center"><input type="submit" value="Naruci robu"> </p>

</form>
 <br><br>

<hr>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade; Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

b) *ServletSesObjekat1.java* koja će da implementira servlet.

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class ServletSesObjekat1 extends HttpServlet {

   protected void processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      // Pri prvom pozivu servleta sesija se kreira. Kod ostalih poziva servleta (drugi,treci,…) vraca kreiranu sesiju.
      HttpSession sesija = request.getSession(true);
      // Pri prvom pozivu servleta sesija parameter brojProizvoda vraca vrednost null, jer taj parameter jos nije bio kreiran.
      Integer brojProizvoda = (Integer) sesija.getValue("brojProizvoda");

String statusSesije;

// Pri prvom pozivu servleta metoda isNew() vraca true. To znaci da je sesija kreirana ali da jos nije vracena do čitača.
// Kada sledeći put čitač pozove servlet metoda isNew() vraća false. Sesija je aktivna sve dok se ne izađe iz čitača
// (kada čitač prestane da se izvršava) ili dok servlet ne prestane da se izvršava.

     if (sesija.isNew())
       statusSesije = "Ovo je nova sesija";
     else
       statusSesije = "Ovo je stara sesija";

// Pri prvom pozivu servleta (kada je broj proizvoda jednak null)  kreira se objekat brojProizvoda i inicijalizuje  se na 0.
      if(brojProizvoda == null)
      { brojProizvoda = new Integer(0);  }
```

```
      // Deklarisanje niza stringova koji treba da prihvate naručene proizvode.
       String [] selektovaniProizvodi;
     // Deklarisanje promenljive koja će da prihvati tekući proizvod.
       String imeProizvoda;

       // Prohvatanje proizvoda koji su naručeni.
       selektovaniProizvodi = request.getParameterValues("proizvod");

       // Ukoliko postoje proizvodi koji su naručeni,
       if (selektovaniProizvodi !=null)
       {  // tada se prolazi kroz sve naručene proizvode, kojima se dodeljuje šifra (BrojProizvoda) i
            for(int i=0; i<selektovaniProizvodi.length; i++)
            {  imeProizvoda = selektovaniProizvodi[i];
              brojProizvoda = new Integer(brojProizvoda.intValue()+1);
            // koji se pamte u sesiji kao par ključ-vrednost. Pri svakom pozivu servleta se pamte
            // proizvodi koji su bili selektovani preko HTML strane. Sesija akumulira (pamti) proizvode
            // koji su bili naruceni (koji su nastali kao rezultat više poziva servleta od jednog klijenta).
              sesija.putValue("Proizvod" + brojProizvoda, imeProizvoda);
              sesija.putValue("brojProizvoda",brojProizvoda);
            }
       }

       // VAŽNA NAPOMENA: Za svakog klijenta se pravi posebna sesija. Svaka sesija ima svoje stanje.
      // Kasnije ćemo videti da klijenti mogu da dele stanje promenljivih preko konteksta.


       HTML h = new HTML("SADRZAJ POTROSACKE KORPE");
       h.add(HTML.HEADING,"SADRZAJ POTROSACKE KORPE",false);
       h.add(HTML.LINE,"",false);
       for(int i = 1; i<= brojProizvoda.intValue();i++)
       { String proizvod = (String) sesija.getValue("Proizvod" + i);
         h.add(HTML.NORMAL,proizvod,true);
       }

     h.add(HTML.LINE,"",true);

     h.add(HTML.NORMAL, "Sesija broj: " + sesija.getId(),true);
     h.add(HTML.NORMAL,statusSesije,true);
// Kada se na ovaj način pozove HTML strana vrednosti parametara strane će biti inicijalne (neće da se pamte vrednosti
// parametara pri zadnjem pozivu servleta )
h.add(HTML.NORMAL,"<A HREF = http://127.0.0.1:8084/ServletSesObjekat1/ServletSesObjekat1.html> Nazad do
potrosacke korpe </A>",true);

     h.add(HTML.LINE,"",true);

     h.add(HTML.AUTOR,"",false);


        out.println(h.prikaziStranu());
        out.close();
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }

}
```

c) *web.xml* - opisivač rasporeda aplikacije.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesObjekat1</servlet-name>
    <servlet-class>ServletSesObjekat1</servlet-class>
  </servlet>
  <servlet-mapping>
```

```
    <servlet-name>ServletSesObjekat1</servlet-name>
    <url-pattern>/ServletSesObjekat1</url-pattern>
  </servlet-mapping>
</web-app>
```

d) *Context.xml* - runtime opisivač rasporeda.

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesObjekat1">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesObjekat1." suffix=".log"
timestamp="true"/>
</Context>
```

Zadatak ZServlet2: *Napraviti HTML dokument preko koga će biti prihvaćeni podaci o Proizvodu: Sifra, Naziv, JedinicaMere, Kolicina i Cena. Navedeni podaci treba da budu poslati do servleta koji će ih prihvatiti, izracunati vrednost proizvoda (Kolicina*Cena) i prikazati odgovarajući izveštaj.*
HTML dokument treba napraviti ručno bez korišćenja bilo kakvog razvojnog okruženja za HTML strane.

Zadatak ZHTML1: *Dopuniti klasu HTML sa metodama koje na jednostavan način kreiraju proizvoljnu tabelu i njene elemente. Napisati servlet koji će pozivati navedene metode kako bi se dobila HTML strana koja sadrži željenu tabelu.*

Zadatak ZServletSesLog1: *Napraviti HTMl dokument preko koga će klijent da pošalje svoje identifikacione podatke (korisničko ime i lozinku) do servera. Servlet treba da omogući:*

a) Unos šifara novih korisnika.
b) Autorizaciju korisnika.

Šifre korisnika treba da budu sačuvane u serijalizovanim datotekama.

Primer ServletSesNoviPret.
Korisnički zahtev*: Napraviti dva servleta:*

a) prvi servlet treba da pamti novog korisnika
b) drugi servlet treba da pronađe klijenta i da mu ažurira atribut Zanimanje.

Pored Zanimanja klijent ima Ime i Lozinku kao atribute.

Prave se sledeće datoteke:
a) ServletSesNovi.html koja će da pozove servlet.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<p>Korisnicki zahtev:Napraviti HTML dokument preko koga ce novi klijent da pošalje identifikacione podatke do servleta kako bi isti bili zapamćeni. </p>

<form action="http://127.0.0.1:8084/ServletSesNoviPret/ServletSesNovi" method="POST">
  <dd> <b>IDENTIFIKACIONI PODACI</b> <br> <br>
  <dd><table border =1>
        <td>Korisnicko ime:
        <td>  <input type="text" name="ImeKorisnika" size="16">
        <tr>
        <td>Lozinka:
        <td> <input type="password" name="NovaLozinka" size="16">
        <tr>
        <td>Ponovi lozinku:
        <td> <input type="password" name="PonovljenaLozinka" size="16">
        <tr>
      </table>
  <br><br>
      <p align="left"><input type="submit" value="Zapamti"></p>
</form>
<hr>
<font size="1"><i>
<b>Author: Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

b) ServletSesPret.html koja će da pozove servlet.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<p>Korisnicki zahtev:Napraviti HTML dokument preko koga ce klijent da pošalje svoj identifikacione podatke (korisnicko ime i lozinku) do servleta  kako bi server prepoznao klijenta i ažurirao mu atribut Zanimanje. </p>
<form action="http://127.0.0.1:8084/ServletSesNoviPret/ServletSesPret" method="POST">
  <dd> <b>IDENTIFIKACIONI PODACI</b> <br> <br>
  <dd><table border =1>
        <td>Korisnicko ime:
        <td>  <input type="text" name="ImeKorisnika" size="16">
        <tr>
        <td>Lozinka:
        <td> <input type="password" name="Lozinka" size="16">
        <tr>
        <td>Zanimanje:
        <td>  <input type="text" name="Zanimanje" size="26">
      </table>
  <br><br>
      <p align="left"><input type="submit" value="Login"></p>
</form>

<hr>

<font size="1"><i>
<b>Author: Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

c) ServletSesNovi.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class ServletSesNovi extends HttpServlet {

   // Deklaracija atributa.
   HTML h = new HTML("Unos identifikacionih podataka novog korisnika:");
   PrintWriter out;
   Korisnici k= new Korisnici(h);

   protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException
    {
      response.setContentType("text/html");
      out = response.getWriter();

       // Prihvatanje vrednosti parametara koje je uneo klijent.
      String ImeKorisnika = request.getParameter("ImeKorisnika");
      String NovaLozinka = request.getParameter("NovaLozinka");
      String PonovljenaLozinka = request.getParameter("PonovljenaLozinka");


       // Ukoliko jedan od parametara nije unet odbija se operacija unosa nove lozinke.
      if(ImeKorisnika.equals("") || NovaLozinka.equals("") || PonovljenaLozinka.equals(""))
        {  h.add(HTML.NORMAL, "Unesite ime i lozinku.",true); out.println(h.prikaziStranu());return 0; }
       // Ukoliko ponovljeni unos lozinke nije jednak novoj lozinci unet odbija se operacija unosa nove lozinke.
      if(NovaLozinka.equals(PonovljenaLozinka) == false)
        {  h.add(HTML.NORMAL, "Nije dobro uneta ponovljena lozinka",true); out.println(h.prikaziStranu());return 0; }

       h.add(HTML.HEADING,"Unos novog korisnika:",false);
       h.add(99,"",true);
        // Pre unosa novog korisnika se proverava da li korisnik postoji i da li je popunjen niz korisnika.
       if(k.proveraDaliKorisnikPostoji(ImeKorisnika) == false)
       {  if(k.proveraDaliJePopunjeNizKorisnika() == false)
           { k.dodajNovogKorisnika(ImeKorisnika,NovaLozinka); // Dodaje se novi korisnik u niz.
             if (k.zapamtiKorisnike()==true)  // Pamte se korisnici u datoteci.
                h.add(HTML.NORMAL, "Zapamcen je novi korisnik.",true);
             else
                h.add(HTML.NORMAL, "Korisnik nije uspeo da se unese u datoteku.",true);
           }
         else
            h.add(HTML.NORMAL, "Sva mesta su zauzeta.",true);
        }
       else
        {h.add(HTML.NORMAL, "Korisnik sa datim imenom postoji.",true);}

       h.add(HTML.LINE,"",true);
       h.add(HTML.AUTOR,"",false);
       out.println(h.prikaziStranu());
       out.close();
       return 1;
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      obradi(request, response);
   }

}
```

```
class Korisnici
{
   HTML h;    Korisnik[] kor = new Korisnik[5];    int tekuciKorisnik=0;
   public Korisnici(HTML h1) { h=h1;}  // Atribut h dobija referencu na h koje je kreirano u servletu. Tako da servlet
objekat i korisnici objekat koriste isti objekat h.

// Opis metode: Provera da li u datoteci kor1.txt postoji definisani korisnik.
   boolean proveraDaliKorisnikPostoji(String ImeKorisnika)
    {  boolean signal = false;
      // Otvaranje ulazne datoteke.
      try {FileInputStream fis = new FileInputStream("kor1.txt");
          // Povezivanje datoteke sa izlaznim objektom.
          ObjectInputStream ois = new ObjectInputStream(fis);
         // Čitaju se svi objekti iz datoteke ukoliko postoje.
          kor = (Korisnik[]) ois.readObject();
         // Ispituje se da li postoji korisnik sa definisanim imenom.
           for(int i=0; i< kor.length;i++)
             {
               if (kor[i]!=null)
               { if (kor[i].ImeKorisnika.equals(ImeKorisnika))
                   { signal = true;
                     break;
                   }
               }
             }

          } catch(Exception e) { h.add(HTML.NORMAL, "Izuzetak kod citanja datoteke: " + e,true);
                          return signal;}
      return signal;
   }

   // Opis metode: Provera da li je niz korisnika popunje. Ukoliko jeste metoda vraca true, inace vraca false. Metoda
  // takodje pronalazi prvo slobodno mesto u nizu i pamti ga preko atributa tekuciKorisnik.
   boolean proveraDaliJePopunjeNizKorisnika()
   {  boolean signal = true;

      try {FileInputStream fis = new FileInputStream("kor1.txt");
          ObjectInputStream ois = new ObjectInputStream(fis);
          kor = (Korisnik[]) ois.readObject();

           for(int i=0; i< kor.length;i++)
             {  // Pronalazi prvo slobodno mesto u nizu.
                 if (kor[i] == null)
                  {signal = false;
                   tekuciKorisnik = i;
                   break;}

             }

          }  catch(EOFException e) {signal = false;}
             catch(FileNotFoundException e) { signal = false;}
             catch(Exception e) { h.add(HTML.NORMAL, "Izuzetak kod citanja datoteke: " + e,true);}
      return signal;
    }

  // Opis metode: Pamti sve korisnike u datoteci kor1.txt.
   boolean zapamtiKorisnike()
    {  boolean signal = false;
       try { // Otvaranje izlazne datoteke.
             FileOutputStream fos = new FileOutputStream("kor1.txt");
            // Povezivanje datoteke sa izlaznim objektom.
             ObjectOutputStream oos = new ObjectOutputStream(fos);
             //Pamćenje objekata u datoteku.
             oos.writeObject(kor);
            // Zatvaranje datoteke.
             oos.close();
         } catch(FileNotFoundException e){return signal;}
           catch(Exception e) { h.add(HTML.NORMAL, "Izuzetak kod upisivanja u datoteku: " + e,true);
                          return signal;}
      return true;   }
```

```
// Opis metode: Proverava da li su korisnicko ime i lozinka postoje. Ukoliko postoje vraća true, inače vraća false. Pored
// toga metoda pamti poziciju korisnika u nizu (ukoliko ga je pronašao).
   boolean proveraDaliSuKorisnickoImeILozinkaDobri(String ImeKorisnika,String Lozinka)
   {   boolean signal = false;
       try { FileInputStream fis = new FileInputStream("kor1.txt");
             ObjectInputStream ois = new ObjectInputStream(fis);
             kor = (Korisnik[]) ois.readObject();

             for(int i=0; i< kor.length;i++)
              {
                if (kor[i].ImeKorisnika.equals(ImeKorisnika) && kor[i].Lozinka.equals(Lozinka))
                 { signal = true;
                   tekuciKorisnik = i;
                   break;
                 }
              }

           } catch(Exception e) { h.add(HTML.NORMAL, "Izuzetak kod citanja datoteke: " + e,true);
                                  return signal;}
       return signal;

   }

    // Opis metode: Pamćenje zanimanja korisnika u nizu.
   void zapamtiZanimanjeKorisnika(String Zanimanje)
   {kor[tekuciKorisnik].Zanimanje = new String(Zanimanje);}

    // Opis metode: Pamćenje novog korisnika u nizu.
   void dodajNovogKorisnika(String ImeKorisnika, String NovaLozinka)
   {Korisnik novikor= new Korisnik(ImeKorisnika,NovaLozinka);
    kor[tekuciKorisnik] = novikor;
   }

   void prikaziKorisnika()
   { h.add(HTML.NORMAL, "Ime korisnika: " + kor[tekuciKorisnik].ImeKorisnika + " Lozinka: " + kor[tekuciKorisnik].Lozinka
+ " Zanimanje: " + kor[tekuciKorisnik].Zanimanje,true); }


}

 // Definicija klase Korisnik. Klasa je serijalizovana što joj omogućava da na jednostavan način pamti i čita podatke iz
// datoteke.
class Korisnik implements Serializable
{ String ImeKorisnika;
  String Lozinka;
  String Zanimanje;
  public Korisnik(String ImeKorisnika1,String Lozinka1)
       { ImeKorisnika = new String(ImeKorisnika1);
         Lozinka = new String(Lozinka1);
         Zanimanje = new String("nema");
       }
}
```

d) ServletSesPret.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;

import javax.servlet.*;
import javax.servlet.http.*;


public class ServletSesPret extends HttpServlet {

   HTML h = new HTML("Unos zanimanja korisnika:");
   PrintWriter out;
   Korisnici k= new Korisnici(h);


   protected int obradi(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {


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

     String ImeKorisnika = request.getParameter("ImeKorisnika");
     String Lozinka = request.getParameter("Lozinka");
     String Zanimanje = request.getParameter("Zanimanje");

      // Provera da li postoji klijent sa definisanim imenom i lozinkom.
     if(k.proveraDaliSuKorisnickoImeILozinkaDobri(ImeKorisnika,Lozinka) == true)
       {   // Ukoliko postoji proverava se da li uneta neka vrednost u polje Zanimanje.
         if ( Zanimanje.equals("")==false)
           // Ukoliko je uneta vrednost u polje Zanimanje ono se pamti u nizu,
           { k.zapamtiZanimanjeKorisnika(Zanimanje);
             // a nakon toga i u datoteci.
            if (k.zapamtiKorisnike())
            { h.add(HTML.NORMAL, "Zapamceno je zanimanje korisnika.",true);
              k.prikaziKorisnika();
            }
            else
              h.add(HTML.NORMAL, "Zanimanje korisnika nije uspelo da se zapamti u datoteci.",true);
           }
         else
            k.prikaziKorisnika();
      }
      else
       { h.add(HTML.NORMAL, "Ime i lozinka korisnika nisu dobri!!!" + Zanimanje,true);}

      h.add(HTML.LINE,"",true);
      h.add(HTML.AUTOR,"",false);
      out.println(h.prikaziStranu());
      out.close();
      return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
    obradi(request, response);
   }

}
```

e) *web.xml - opisivač rasporeda aplikacije*

```
<html>

<body bgcolor="#C0C0C0" text="#000080">

<p>Korisnicki zahtev:Napraviti HTML dokument preko koga ce novi klijent da pošalje identifikacione podatke do servera.
</p>

<form action="http://127.0.0.1:8084/ServletSesNoviPret/ServletSesNovi" method="POST">

  <dd> <b>IDENTIFIKACIONI PODACI</b> <br> <br>
  <dd><table border =1>
            <td>Korisnicko ime:
        <td>  <input type="text" name="ImeKorisnika" size="16">
        <tr>
        <td>Lozinka:
        <td> <input type="password" name="NovaLozinka" size="16">
        <tr>
        <td>Ponovi lozinku:
        <td> <input type="password" name="PonovljenaLozinka" size="16">
        <tr>

      </table>
  <br><br>
      <p align="left"><input type="submit" value="Zapamti"></p>
</form>

<hr>

<font size="1"><i>
<b>Author: Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>

</body>


</html>
```

f) *Context.xml* - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesNoviPret">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesNoviPret." suffix=".log"
timestamp="true"/>
</Context>
```

Zadatak ZServletSesObjekat1: *Napravite HTML stranu preko koje će biti prihvaćene vrednosti sledećih polja: ŠifraProizvoda, Naziv, Kolicina, TipDokumenta (prijem ili otprema). Poslati servletu navedene podatke. Zapamtiti u servletu šifru proizvoda, naziv i količinu. Pri svakom pozivu servleta proizvod podiže odnosno spušta stanje u zavisnosti od vrednosti parametra TipDokumenta (prijem podiže stanje, otprema spušta stanje). Stanje sesije obezbediti preko HTTPSesion objekta.*

Primer ServletSesBP1.

Korisnički zahtev*: Napisati servlet pomoću koga će moći da se:*

a) unose podaci o novom poslovnom partneru
b) brišu podaci o poslovnom partneru
c) menjaju podaci o poslovnom partneru
d) pretražuje partner po imenu
e) prikažu imena svih partnera.

Podaci o partneru treba da se čuvaju u bazi podataka. Poslovni partner ima sledeće atribute: Ime, Adresa, *Email, Telefon, Starost. Za svaku od navedenih opeacija napraviti HTML stranu koja će je pozivati.*

Prave se sledeće datoteke:

a) ServletSesBP1Ubaci.html koja će da pozove metodu servleta koja unosi podatke partnera u bazu.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<body>
Primer: Napraviti HTML dokument preko koga ce biti prihvaceni licni
podaci poslovnog partnera I pozvan servlet koji ce ih zapamtiti u bazi:<br> Ime i prezime, Adresa, E-mail, Telefon,
BrojGodina.
<br><br>
   <dd><b>OSNOVNI PODACI O POSLOVNOM PARTNERU:</b>
   <br><br>
<form action="http://127.0.0.1:8084/ServletSesBP1/ServletSesBP1?Oper=U" method="POST">
 <dd><table width="396" border =1>
        <td><i>Ime</i>
        <td>   <input type="text" size = 50 name="Ime">
        <tr>   <td><i>Adresa</i>    <td> <input type="text" size = 75 name="Adresa">      </tr>
        <tr>    <td><i>E-mail</i>     <td> <input type="text" size = 20  name="EMail">      </tr>
        <tr>    <td><i>Telefon</i>   <td> <input type="text" size = 20 name="Telefon">    </tr>
        <tr>    <td><i>Starost</i>    <td> <input type="text" size = 3 name="Starost">       </tr>
    </table>
 <p align="center"><input type="submit" value="Zapamti partnera"></p>
</form>
<hr>

<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade&amp;nbsp;
&amp;nbsp;Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

b) ServletSesBP1Promeni.html koja će da pozove metodu servleta koja menja podatke željenog partnera..

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<body>
Primer: Napraviti HTML dokument preko koga ce biti promenjeni licni podaci poslovnog partnera:
<br><br>
   <dd><b>OSNOVNI PODACI O POSLOVNOM PARTNERU:</b>
   <br><br>
<form action="http://127.0.0.1:8084/ServletSesBP1/ServletSesBP1?Oper=P " method="POST">
```

... Ostali deo koda je isti kao kod ServletSesBP1Ubaci.html. Podvučeno je mesto gde se razlikuju datoteke ServletSesBP1Ubaci.html I ServletSesBP1Promeni.html

c) ServletSesBP1Promeni.html koja će da pozove metodu servleta koja briše podatke željenog partnera.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<body>

Primer: Napraviti HTML dokument preko koga ce biti obrisani podaci poslovnog partnera na osnovu imena partnera:<br>

<br><br>
   <dd><b>BRISANJE PARTNERA:</b>
   <br><br>
<form action="http://127.0.0.1:8084/ServletSesBP1/ServletSesBP1?Oper=B" method="POST">
  <dd><table width="396" border =1>
         <td><i>Ime</i>  <td>
         <input type="text" size = 50 name="Ime">
        </table>
 <p align="center"><input type="submit" value="Obrisi partnera"></p>

</form>
... Ostali deo koda (nakon zatvaranja forme)je isti kao kod ServletSesBP1Ubaci.html.
```

d) ServletSesBP1Prikazi.html koja će da pozove metodu servleta koja prikazuje imena svih partnera.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<body>
Primer: Napraviti HTML dokument preko koga ce biti prikazana imena svih poslovnih partnera.<br>

<br><br>
   <dd><b>PRIKAZI IMENA POSLOVNIH PARTNERA</b>
   <br><br>
   <form action="http://127.0.0.1:8084/ServletSesBP1/ServletSesBP1?Oper=Pr" method="POST">
    <p align="center"><input type="submit" value="Prikazi"></p>
    </form>

... Ostali deo koda (nakon zatvaranja forme)je isti kao kod ServletSesBP1Ubaci.html.
```

e) ServletSesBP1Nadji.html

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<body>
Primer: Napraviti HTML dokument preko koga ce biti pretrazeni poslovni partneri po zadatom imenu:<br>
<br><br>
   <dd><b>OSNOVNI PODACI O POSLOVNOM PARTNERU:</b>
   <br><br>
    <form action="http://127.0.0.1:8084/ServletSesBP1/ServletSesBP1?Oper=T" method="POST">
     <dd><table size = 25 border =1>
         <td><i>Ime</i>
         <td>  <input type="text" size = 50 name="Ime">
          </table>
      <p align="center"><input type="submit" value="Pretrazi partnera"></p>
   </form>

... Ostali deo koda (nakon zatvaranja forme)je isti kao kod ServletSesBP1Ubaci.html.
```

f) *ServletSesBP1.java* koja će da implementira servlet.

```
import java.io.*;
import java.net.*;
import java.sql.*;

import javax.servlet.*;
import javax.servlet.http.*;



public class ServletSesBP1 extends HttpServlet {

   HTML h = new HTML("Azuriranje i pretrazivanje partnera u bazi: ");

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

     {

      response.setContentType("text/html");


      PrintWriter out = response.getWriter();

      // Prihvatanje podataka o partneru.
      String Ime = request.getParameter("Ime");
      String Adresa = request.getParameter("Adresa");
      String EMail = request.getParameter("EMail");
      String Telefon = request.getParameter("Telefon");
      String Starost = request.getParameter("Starost");
      String Oper = request.getParameter("Oper");


      h.add(HTML.HEADING,"Azuriranje i pretrazivanje partnera u bazi: ",false);
       // Definisanje database brokera.
      DBBrokerSp db = new DBBrokerSp()
      // Otvaranje baze preko database brokera..
      int signal = db.otvoriBazu("Baza");

      // Ukoliko je izabrana opcija  prikazivanja podataka o zeljenom partneru,
      if(Oper.equals("T"))
      {       // tada se izvrsava upit. Rezultat upita je trazeni slog ukoliko isti postoji.
             ResultSet rs = db.izvrsiUpit("Select * From PoslovniPartner Where Ime='" + Ime + "'");
              try
               {if (rs.next())
                 { // Prikazivanje podataka o pronadjenom poslovnom partneru.
                  h.add(HTML.NORMAL, "<>br<br>PODACI O POSLOVNOM PARTNERU: <br>",true);
                  h.add(HTML.NORMAL,"Ime:" + Ime + " <br>",true);
                  h.add(HTML.NORMAL,"Adresa:" + rs.getString("Adresa") + " <br>",true);
                  h.add(HTML.NORMAL,"Telefon:" + rs.getString("Telefon") + " <br>",true);
                  h.add(HTML.NORMAL,"Email:" + rs.getString("EMail") + " <br>",true);
                  h.add(HTML.NORMAL,"Starost:" + rs.getInt("Starost") + " <br>",true);
                  }
                 else
                  //  Ukoliko ne postoji poslovni partner prikazuje se odgovarajuca poruka.
                   h.add(HTML.NORMAL, "Ne postoji trazeni partner.",true);
                 } catch(SQLException esql){h.add(HTML.NORMAL, "Izuzetak kod citanja rezultata pretrazivanja.",true);}
      }

      else
      {    // Ukoliko je izabrana opcija  prikazivanja imena svih poslovnih partnera,
          if (Oper.equals("Pr"))
                {   // tada poziva se metoda koja ce da vrati imena svih poslovnih partnera.
                  String kolone[] = db.nadjiSlogove("PoslovniPartner","Ime");
                  // Ukoliko ne postoje imena prikazace se odgovarajuca poruka,
                 if (kolone == null)
                    { h.add(HTML.NORMAL, "Ne postoje partneri.",true);}
                  else
                    // inace ce biti prikazana imena poslovnih partnera.
```

```
                { h.add(HTML.NORMAL, "Imena poslovnih partnera:",true);
                  for(int i=0;i<kolone.length;i++)
                     h.add(HTML.NORMAL, kolone[i],true);
                }
              }
        else
            // Ukoliko je izabrana jedna od opcija  azuriranja partnera, proverava se da li je baza dobro otvorena I da li je
            //uneto Ime. Ukoliko jedan od uslova nije zadovoljen prekida se izvrsenje servleta.
            if (signal!=41 || Ime.equals(""))
               {h.add(HTML.NORMAL, "Baza nije mogla da se otvori ili nije uneto ime partnera",true);}
            else
             { // Ukoliko je izabrana opcija  unosa novog partnera.
               if (Oper.equals("U"))
                { //novi partner se pamti I prikazuje  se odgovarajuca porukau yavisnosti od uspesnosti izvrsenja pamcenja.
                  signal = db.pamtiSlog("PoslovniPartner", "'" + Ime + "', '" + Adresa + "', '" + EMail + "', '" + Telefon +
                                                  "', '"  +   Starost + "'");
                   // Pamcenje sloga se mora zavrsiti sa commit operacijom, kako bi se podaci iz kes memorije prebacili u
                   // bazu.
                   if (signal == 32)
                      { db.rollbackTransakcije();h.add(HTML.NORMAL, "Partner nije zapamcen u bazi.",true);}
                   else
                      { db.commitTransakcije(); h.add(HTML.NORMAL, "Zapamcen je novi partner.",true);}
                 }
                // Na slican nacin se izvrsavaju operacije brisanja i promene podataka o poslovnom partneru.
               if (Oper.equals("B"))
                {
                 signal = db.brisiSlog("PoslovniPartner", " Ime = '" + Ime + "'");
                 if (signal == 34)
                      { db.rollbackTransakcije();h.add(HTML.NORMAL, "Partner nije obrisan u bazi.",true);}
                  else
                      { db.commitTransakcije(); h.add(HTML.NORMAL, "Obrisan je partner u bazi.",true);}
                 }

               if (Oper.equals("P"))
                 {
                  signal = db.promeniSlog("PoslovniPartner", "Ime = '" + Ime + "', Adresa = '" + Adresa + "', EMail = '"
                          +  EMail + "', Telefon = '" + Telefon + "' " + ", Starost = '" + Starost + "'",  " Ime = '" + Ime + "');

                  if (signal == 36)
                      { db.rollbackTransakcije();h.add(HTML.NORMAL, "Partner nije promenjen u bazi.",true);}
                   else
                      { db.commitTransakcije(); h.add(HTML.NORMAL, "Promenjen je partner u bazi.",true);}
                 }

              }
          }
        db.zatvoriBazu();
        h.add(HTML.LINE,"",true);
        h.add(HTML.AUTOR,"",false);
        out.println(h.prikaziStranu());
        out.close();
        }

}
```

g) DBBrokerSP.java datoteka služi da uspostavi komunikaciju sa bazom podataka.

```
import java.io.*;
import java.sql.*;

public class DBBrokerSp
{
  static Connection con;
  static Statement st;
  boolean otvoreno =false;

  public DBBrokerSp(){}
  public int otvoriBazu(String imeBaze)
              {   if (otvoreno == true)
            { return 41;}
                    String Urlbaze;
                    try {
                    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
      Urlbaze = "jdbc:odbc:" + imeBaze;
      con = DriverManager.getConnection(Urlbaze);
      con.setAutoCommit(false); // Ako se ovo ne uradi nece moci da se radi roolback.
       st = con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONCUR_UPDATABLE);
                } catch(ClassNotFoundException e) { System.out.println("Drajver nije ucitan:" + e);return 42;}
                  catch(SQLException esql) { System.out.println("Greska kod konekcije:" + esql);return 43;}
                        catch(SecurityException ese) {System.out.println("Greska zastite:" + ese);return 44;}
          otvoreno = true;
                    return 41;
            }

 public int zatvoriBazu()
{ try {  otvoreno = false; st.close();con.close();} catch(Exception e) {System.out.println(e);return 62;} return 61; }


 synchronized public String[] nadjiSlogove(String imeTabele, String imeAtributa)
 {
      String upit;
      ResultSet rs;
      int brojSlogova;
      int i=0;
      String spom;
      int ipom;
      double dpom;
      float fpom;
      short shpom;
      boolean bpom;
      String str[];
      str = null;

       try {

       upit ="SELECT " + imeAtributa + " FROM " + imeTabele;

       rs = st.executeQuery(upit);
       rs.last();
       brojSlogova = rs.getRow();
       if (brojSlogova == 0)
        { return str; // Nema slogova.
                }

       str = new String[brojSlogova];
       ResultSetMetaData rsm = rs.getMetaData();

       int brojKolone = rs.findColumn(imeAtributa);
       String imeTipaAtributa = rsm.getColumnTypeName(brojKolone);
```

```
      rs.beforeFirst();
      while(rs.next())
       {
        if (imeTipaAtributa.equals("SMALLINT") == true)
           { shpom = rs.getShort(imeAtributa); str[i] = String.valueOf(shpom); }
        if (imeTipaAtributa.equals("INTEGER") == true)
           { ipom = rs.getInt(imeAtributa); str[i] = String.valueOf(ipom); }
         if (imeTipaAtributa.equals("VARCHAR") == true)
           { spom = rs.getString(imeAtributa); str[i] = String.valueOf(spom);}
         if (imeTipaAtributa.equals("FLOAT") == true)
           { fpom = rs.getFloat(imeAtributa); str[i] = String.valueOf(fpom);}
         if (imeTipaAtributa.equals("DOUBLE") == true)
           { dpom = rs.getFloat(imeAtributa); str[i] = String.valueOf(dpom);}
         if (imeTipaAtributa.equals("BOOLEAN") == true)
           { bpom = rs.getBoolean(imeAtributa); str[i] = String.valueOf(bpom);}
         i++;
                }
       rs.close();
        }
              catch(Exception e) { System.out.println("Greska kod citanja slogova iz baze" + e); }

     return str;

}

 synchronized public ResultSet izvrsiUpit(String upit)
{ ResultSet rs=null;
  try {    rs = st.executeQuery(upit);
      } catch(Exception e) { System.out.println("Greska kod izvrsenja upita" + e); }
  return rs;
}

 synchronized public int pamtiSlog(String imeObjekta, String vrednostiAtributa)
{   try{
     String upit ="INSERT INTO " + imeObjekta  + " VALUES (" + vrednostiAtributa + ")";
     st.executeUpdate(upit);
      }   catch(Exception esql)  {  System.out.println("Nije uspesno zapamcen slog u bazi: " + esql);
                return 32;   }
     return 31;
   }

 synchronized public int promeniSlog(String imeObjekta, String vrednostiAtributa,String uslovZaNadjiSlog)
   {  try {
         String upit ="UPDATE " + imeObjekta +
              " SET " + vrednostiAtributa +
              " WHERE " + uslovZaNadjiSlog;
         st.executeUpdate(upit);
         }   catch(SQLException esql) {System.out.println("Nije uspesno promenjen slog u bazu: " + esql);
                                       return 36;}
     return 35;
   }


synchronized public int brisiSlog(String imeObjekta,String uslovZaNadjiSlog)
   {   try {
          String upit ="DELETE *  FROM " + imeObjekta + " WHERE " + uslovZaNadjiSlog;
          st.executeUpdate(upit);
         }   catch(SQLException esql)
             { System.out.println("Nije uspesno obrisan slog u bazi: " + esql); return 34; }
      return 33;
   }

 synchronized public int commitTransakcije()
  {   try{ con.commit();}
       catch(SQLException esql)
         {  System.out.println("Nije uspesno uradjen commit. " + esql);
              return 52;
                                  }
     return 51;
  }
```

```
   synchronized public int rollbackTransakcije()
    {   try{ con.rollback();}
          catch(SQLException esql)
            {  System.out.println("Nije uspesno uradjen rollback." +
              "Ako nije uradjen rollback sistem prelazi u nestabilno stanje. " + esql);
                    return 54;
      return 53;
    }

}
```

c) *web.xml - opisivač rasporeda aplikacije*

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletSesBP1</servlet-name>
    <servlet-class>ServletSesBP1</servlet-class>
  </servlet>

  <servlet-mapping>
     <servlet-name>ServletSesBP1</servlet-name>
     <url-pattern>/ServletSesBP1</url-pattern>
  </servlet-mapping>
 </web-app>
```

d) *Context.xml* - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletSesBP1">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletSesBP1." suffix=".log" timestamp="true"/>
</Context>
```

Zadatak ZServletSesDat1: *Napravite servlet pomoću koga će biti prihvaćeni podaci o Studentu (BrojIndeksa, Ime, Pol, Semestar). Primarni ključ je BrojIndeksa. Napraviti bazu podataka u Access-u, koja će imati tabelu Student sa navedenim poljima. Povezati servlet sa bazom preko database brokera. Kada servlet prihvati podatke o studentu, on ih pamti u operativnoj memoriji. Servlet ne treba odmah da ih pamti studente u bazu. To radi posle nekog vremena (periodično osvežavanje baze). Onemogućiti da dva klijenta istovremeno pristupaju metodi koja pamti studente u operativnoj memoriji. Omogućiti sledeće metode servleta:*

a) Prikaz svih studenata ženskog pola.

b) Prikazati sve studente koji su upisali treću godinu studija (broj semestra je veći od četiri). Preko SQL upita koji šalje servlet obrisati sve studente koji su uneli semestar koji je izvan dozvoljenih granica (između 1 i 8).

Primer ServletKontekst1.
Korisnički zahtev*: Napraviti HTML dokument preko koga će biti prihvaćeno polje Količina. Omogućiti da se pritiskom dugmeta Obrada na serverskoj straini ažurira atribut Kolicina u zavisnosti od izabrane vrste dokumenta (Prijemnica, Otpremnica).*
Prave se sledeće datoteke:
a) ServletKontekst1.html koja će da pozove servlet.

```
<html>

<body bgcolor="#C0C0C0" text="#000080">

<p align="left">Primer: Napraviti HTML dokument preko koga ce biti prihvaceno polje Kolicina. Omoguciti da se pritiskom
dugmeta Obrada na serverskoj strani azurira atribut Kolicina u zavisnosti od izabrane vrste dokumenta (Prijemnica,
Otpremnica).</p>

<form action="http://127.0.0.1:8084/ServletKontekst/ServletKontekst" method="POST">

  <table width="581" height="354" >

    <tr> <td width="581" height="24">AZURIRANJE KOLICINE: </tr>

     <tr> <td width="581" height="20">Vrsta dokumenta:
            <select name="VrstaDokumenta" size="1">
            <option selected>Prijemnica</option>
            <option>Otpremnica</option>
            </select>   
     </tr>

     <tr> <td width="581" height="20"> Kolicina
     <input type="text" size="9" name="Kolicina" value = "0">
     </tr>

   <tr> <td width="581" height="17">
        <p align="center"><input type="submit" value="Obrada"></p>
   </tr>
  </table>
```

... Preostali deo HTML strane se odnosi na autora koda..

b) ServletKontekst2.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;


// Klasa koja implementira servlet.
public class ServletKontekst1 extends HttpServlet {

 // Deklarisanje atributa servleta.
HTML h = new HTML("AZURIRANJE MAGACINA:");

// Metoda servleta koja obradjuje zahtev klijenta.
synchronized protected int processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      // Deklaracija promenljivih servleta.
      String VrstaDokumenta;
      int Kolicina = 0;
      int Stanje = 0;

      // Prihvatanje parametara koje salje klijent.
      VrstaDokumenta = request.getParameter("VrstaDokumenta");
```

```
    try {   Kolicina = Integer.parseInt(request.getParameter("Kolicina"));
          }  catch(Exception e) {h.add(HTML.NORMAL, "Greska kod unosa kolicines!!!",true);
                                 out.println(h.prikaziStranu());  out.close(); return 0;}

     // Kod prvog poziva servleta  kontekst atribut Stanje ima vrednost null jer jos nije kreiran.
    if (getServletContext().getAttribute("Stanje") == null)
        getServletContext().setAttribute("Stanje", new Integer(0));
    else // Kod drugog i svakog sledeceg poziva servleta uzima se njegovo zapamceno stanje.
        Stanje = ((Integer)getServletContext().getAttribute("Stanje")).intValue();

   // U zavisnosti od vrste dokumenta bice podignuto odnosno skinuto stanje za iznos definisane kolicine.
   if (VrstaDokumenta.equals("Prijemnica") == true)
    {  Stanje = Stanje + Kolicina;
    }

    if (VrstaDokumenta.equals("Otpremnica"))
    {  Stanje = Stanje - Kolicina;
    }

    // Kada se zeli testirati da li dva ili vise klijenta sinhronizovano pristupaju metodi. Ne bi smelo da se desi da dva
    //  klijenta u istom trenutku pristupaju metodi jer bi Stanje moglo biti nekonzistentno.
    //try { Thread.currentThread().sleep(10000);} catch(InterruptedException ie){h.add(HTML.NORMAL,"Greska
      //      kod  pauze od 10 sekundi!!!" ,true);return 0;}

    h.add(HTML.NORMAL,"Uspesno azurirano!!!" ,true);

     // U kontekstu se pamti novo stanje.
    getServletContext().setAttribute("Stanje", new Integer(Stanje));

     // Prikaz stanja magacina.
    h.add(HTML.HEADING,"Stanje: " + Stanje,true);
    h.add(HTML.LINE,"",true);
    h.add(HTML.AUTOR,"",false);
    out.println(h.prikaziStranu());
   out.close();
   return 1;
  }

  protected void doPost(HttpServletRequest request, HttpServletResponse response)
  throws ServletException, IOException {
     processRequest(request, response);
  }
}
```

c) web.xml - opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletKontekst1</servlet-name>
    <servlet-class>ServletKontekst1</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletKontekst1</servlet-name>
    <url-pattern>/ServletKontekst1</url-pattern>
  </servlet-mapping>
</web-app>
```

d) Context.xml - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletKontekst1">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletKontekst1." suffix=".log" timestamp="true"/>
</Context>
```

Zadatak ZKontekst1: *Napravite servlet koji će da prihvati niz znakova (niz realizovati preko kontekstnih promenljiivih). Pri svakom sledećem pozivu servlet vraća jedan znak iz niza i prazni ga. Kada se niz na strani servleta isprazni on može opet da prihvati niz znakova od klijenta. Ne sme da se desi da isti znak prihvate dva ili više klijenata u isto vreme (Jednom izvršenju metode koja prazni niz pristupa samo jedan klijent).*

Primer ServletKontekst2.
Korisnički zahtev*: Napraviti HTML dokument preko koga će biti prihvaćeni podaci o Sifri proizvoda, Magacinu i Kolicini. Omogućiti da se pritiskom dugmeta Obrada na serverskoj strani ažurira atribut Kolicina u zavisnosti od izabrane vrste dokumenta (Prijemnica, Otpremnica).*
Prave se sledeće datoteke:
a) ServletKontekst2.html koja će da pozove servlet.

```
<html>

<body bgcolor="#C0C0C0" text="#000080">

<p align="left">Primer: Napraviti HTML dokument preko koga ce biti prihvaceni podaci o Sifri proizvoda, Magacinu i
Kolicini. Omoguciti da se pritiskom dugmeta Obrada na serverskoj strani azurira atribut Kolicina u
zavisnosti od izabrane vrste dokumenta (Prijemnica, Otpremnica):</p>

<form action="http://127.0.0.1:8084/ServletKontekst/ServletKontekst" method="POST">

  <table width="581" height="354" >

   <tr> <td width="581" height="24">AZURIRANJE PROIZVODA: </tr>

    <tr> <td width="581" height="20">Vrsta dokumenta:
          <select name="VrstaDokumenta" size="1">
          <option selected>Prijemnica</option>  <option>Otpremnica</option>
          </select>   
    </tr>
    <tr> <td width="581" height="20"> Magacin:
        <select name="Magacin" size="1">
        <option selected>mag1</option>    <option>mag2</option>
        </select>
     </tr>

    <tr> <td width="581" height="20">Sifra proizvoda:
        <select name="SifraProizvoda" size="1">
        <option selected>p1</option>  <option>p2</option>  <option>p3</option>  <option>p4</option>
        </select>
    </tr>
    <tr> <td width="581" height="20"> Kolicina    <input type="text" size="9" name="Kolicina" value = "0">
    </tr>

  <tr> <td width="581" height="17">
      <p align="center"><input type="submit" value="Obrada"></p>
  </tr>
 </table>

<hr>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>

</font>

</form>
</body>
```

b) ServletKontekst2.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;
import java.util.*;

import javax.servlet.*;
import javax.servlet.http.*;

// Klasa koja implementira servlet.
public class ServletKontekst2 extends HttpServlet {
      // Atribut servleta.
      HTML h = new HTML("AZURIRANJE MAGACINA:");

    // Metoda servleta koja obradjuje zahtev klijenta.
   synchronized protected int processRequest(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

       // Deklaracija promenljivih servleta.
      String VrstaDokumenta;
      String Magacin;
      String SifraProizvoda;
      int Kolicina = 0;
      int Stanje = 0;

     // Prihvatanje parametara koje salje klijent.
      VrstaDokumenta = request.getParameter("VrstaDokumenta");
      Magacin = request.getParameter("Magacin");
      SifraProizvoda = request.getParameter("SifraProizvoda");

       // Kreiranje kljuca koji ce jedinstveno da identifikuje par SifraProizvoda-Magacin.
      String Kljuc = SifraProizvoda + Magacin;

       // Citanje parametra Kolicina i njena konverzija u celobrojni tip.
       try {   Kolicina = Integer.parseInt(request.getParameter("Kolicina"));
          }  catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD UNOSA KOLICINE!!!",true);
                        out.println(h.prikaziStranu());  out.close(); return 0;}

       // Pri prvom pozivu za poslati Kljuc (par Magacin-SifraProizvoda) kontekst atribut Kljuc ima null vrednost jer on jos
       // nije kreiran.
      if (getServletContext().getAttribute(Kljuc) == null)
          // Postavljanje Kljuca na vrednost 0. Ime Kljuca (para Magacin-SifraProizvoda) jednoznacno odredjuje kolicinu.
          getServletContext().setAttribute(Kljuc, new Integer(0));
      else
          // Uzima se zapamcena kolicina za dati Kljuc.
          Stanje = ((Integer)getServletContext().getAttribute(SifraProizvoda+Magacin)).intValue();

    // U zavisnosti od vrste dokumenta bice podignuto odnosno skinuto stanje za iznos definisane kolicine.
     if (VrstaDokumenta.equals("Prijemnica") == true)
     {  Stanje = Stanje + Kolicina;
     }

     if (VrstaDokumenta.equals("Otpremnica"))
     {  Stanje = Stanje - Kolicina;
     }

       // Kada se zeli testirati da li dva ili vise klijenta sinhronizovano pristupaju metodi. Ne bi smelo da se desi da dva
     //  klijenta u istom trenutku pristupaju metodi jer bi Stanje moglo biti nekonzistentno.
     //try { Thread.currentThread().sleep(10000);} catch(InterruptedException ie){h.add(HTML.NORMAL,"Greska
     // kod  pauze od 10 sekundi!!!" ,true);return 0;}
     h.add(HTML.NORMAL,"Uspesno azurirano!!!" ,true);

     // Postavljanje kontekstne promenljive Kljuc na novu kolicinu.
     getServletContext().setAttribute(SifraProizvoda+Magacin, new Integer(Stanje));

     h.add(HTML.HEADING,"Stanje magacina: " + Magacin,true);
     h.add(HTML.NORMAL,"za proizvod:" + SifraProizvoda,true);
     h.add(HTML.NORMAL,"je: " + Stanje,true);
```

```
        h.add(HTML.LINE,"",true);
        h.add(HTML.AUTOR,"",false);
        out.println(h.prikaziStranu());
       out.close();
       return 1;
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
        processRequest(request, response);
    }
}
```

c) web.xml - opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletKontekst2</servlet-name>
    <servlet-class>ServletKontekst2</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletKontekst2</servlet-name>
    <url-pattern>/ServletKontekst2</url-pattern>
  </servlet-mapping>
</web-app>
```

d) Context.xml - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletKontekst2">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletKontekst2." suffix=".log" timestamp="true"/>
</Context>
```

Zadatak ZKontekst2: *Napraviti servlet koji će da omogući izdavanje autobuskih karata. U jednom autobusu ima 52 mesta. Jedan kupac (klijent) nemože da kupi više od 4 karte. Omogućiti da se karte mogu vratiti pre nego što autobus krene. Kada autobus krene prodaja karata po tom autobusu je završena. Poželjno je da se napravi timer koji će servletu javiti da je autobus krenuo. Takođe omogućiti da dispečer može da javi da je autobus krenuo.*

Primer ServletApplet.
Korisnički zahtev*: Napraviti aplet koji će da prihvati naziv poslovnog partnera i broj godina saradnje sa njim. Poslati podatke do servleta koji treba da vrati iznos popusta za partnera u zavisnosti od broja godine saradnje.*

Prave se sledeće datoteke:
a) OA.java koja će da implementira aplet.

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
import java.io.*;
import java.net.*;

// Klasa koja implementira aplet.
public class OA extends Applet implements ActionListener, ItemListener{

// Polja za prohvat podataka.
TextField tfIme;
TextField tfbrojGodinaSaradnje;
// U ovo polje ce biti vracen rezultat od servleta.
TextField tfPopust;

// Dugme preko koga ce biti pozvan servlet.
Button bPartner;

// Niz labela koji se koristi radi preloma strane.
Label umetak[] = new Label[10];
int bu =0;

// Poruka koja se ispisuje nakon poyiva servleta o uspesnosti izvrsenja.
String poruka;

// Metoda koja inicijalizuje aplet.
public void init()
{
   String str;
   int duzina;
// Odredjivanje boje crtanja i pozadine.
   setBackground(Color.yellow);    setForeground(Color.blue);

 // Pokretanje menadzera rasporeda.
setLayout(new FlowLayout(FlowLayout.LEFT));

// Kreiranje grafickih objekata za ispisivanje labela i za prijem podataka.
str = "OBRADA POSLOVNIH PARTNERA";
Label lpartner = new Label(str);add(lpartner);
Umetak(str.length());  Umetak(str.length());

str = "Ime";duzina =30; Label lIme = new Label(str);add(lIme);
tfIme = new TextField(duzina);add(tfIme);Umetak(str.length()+duzina);

str = "Broj godina saradnje";duzina =3;    Label lbrojGodinaSaradnje = new Label(str);add(lbrojGodinaSaradnje);
tfbrojGodinaSaradnje = new TextField(duzina);add(tfbrojGodinaSaradnje);Umetak(str.length()+duzina);

str = "Popust"; duzina =8; Label lPopust = new Label(str);add(lPopust);
tfPopust = new TextField(duzina);add(tfPopust);Umetak(str.length()+duzina);

// Kreiranje dugmeta.
str= "Pozovi servlet";
bPartner = new Button(str);add(bPartner);  bPartner.addActionListener(this);

}

// Podesavanje novog reda.
void Umetak (int brojpraznih)
{  int duzinanaj = 42;
   char n[] = new char [duzinanaj];
```

```
    for(int i=0;i<duzinanaj-brojpraznih;i++)
        n[i]=' ';
    String s = new String(n,0,duzinanaj-brojpraznih);
    umetak[bu] = new Label(s);add(umetak[bu]);
    bu++;
}

// Poruka o uspesnosti izvrsenja servleta.
public void paint(Graphics g)
    {    int j=0;
         g.drawString("Poruka o uspesnosti poziva servleta:" + poruka, 6,300);
   }

// Na klig dugmeta poziva se servlet. Nakon toga se osvezava aplet.
public void actionPerformed(ActionEvent ae)
 { String strdugme = ae.getActionCommand();
   if (strdugme.equals("Zapamti partnera"))
      {   poruka = pozoviServlet();    }
    repaint();
 }


public void itemStateChanged(ItemEvent ie)  { repaint();}

// Poziv servleta.
String pozoviServlet()
{
  try {
   // Definisanje parametara koji ce biti prosledjeni do servleta.
  String pIme = "Ime" + "=" + URLEncoder.encode(tfIme.getText()) + "&";
  String pBGS = "brojGodinaSaradnje" + "=" + URLEncoder.encode(tfbrojGodinaSaradnje.getText());
  String par = pIme+pBGS;

  // Kreiranje objekta koji ce se povezati sa servletom.
  URL url = new URL("http://127.0.0.1:8084/ServletApplet/ServletApplet?" + par);

  // Poyiv servleta.
  URLConnection urlcon = url.openConnection();

  // Prijem odgovora od servleta.

  InputStreamReader in = new InputStreamReader(urlcon.getInputStream());
  int znak = in.read();
  String pom="";
  while(znak!=-1)
    {
              pom = pom + (char)znak;
              znak = in.read();
    }
    System.out.println("odgovor:" + pom);
    tfPopust.setText(pom);
      } catch(Exception e) {return "Iz" + e;}

  return "Servlet je korektno izvrsen";
 }
}
```

b) ServletApplet.java koja će da implementira servlet.

```
import java.io.*;
import java.net.*;
import java.text.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.net.*;
```

```
// Klasa koja implmentira servlet.
  public class ServletApplet extends HttpServlet {

    HTML h = new HTML("Racunanje popusta:");
    PrintWriter out;


    protected int obradi(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

       PrintWriter out = response.getWriter();
       response.setContentType("text/html");
        // Deklaracija promenljivih servleta.
       String ImeKorisnika = request.getParameter("Ime");
       String brojGodinaSaradnje = request.getParameter("brojGodinaSaradnje");
        // Kreiranje promenljive popust.
       double popust;

        // U zavisnosti od broja godina saradnje servlet odredjuje iznos popusta.
       if (Integer.parseInt(brojGodinaSaradnje) > 10)
           popust = 12.5;
       else
          popust = 5.25;
        // Vracanje vrednosti popusta do klijenta.
       out.println(popust);
       out.close();
        return 1;
     }

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

}
```

c) web.xml - opisivač rasporeda aplikacije

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd" version="2.4">
  <servlet>
    <servlet-name>ServletApplet</servlet-name>
    <servlet-class>ServletApplet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>ServletApplet</servlet-name>
    <url-pattern>/ServletApplet</url-pattern>
  </servlet-mapping>
</web-app>
```

d) Context.xml - runtime opisivač rasporeda

```
<?xml version="1.0" encoding="UTF-8"?>
<Context path="/ServletApplet">
  <Logger className="org.apache.catalina.logger.FileLogger" prefix="ServletApplet." suffix=".log"
timestamp="true"/>
</Context>
```

Zadatak ZservletApplet1: *Napraviti aplet koji će da prihvati dva broja. Brojeve proslediti do servleta koji treba da izracuna njihov zbir. Prikazati u okviru apleta izvestaj o brojevima i njihovom zbiru.*

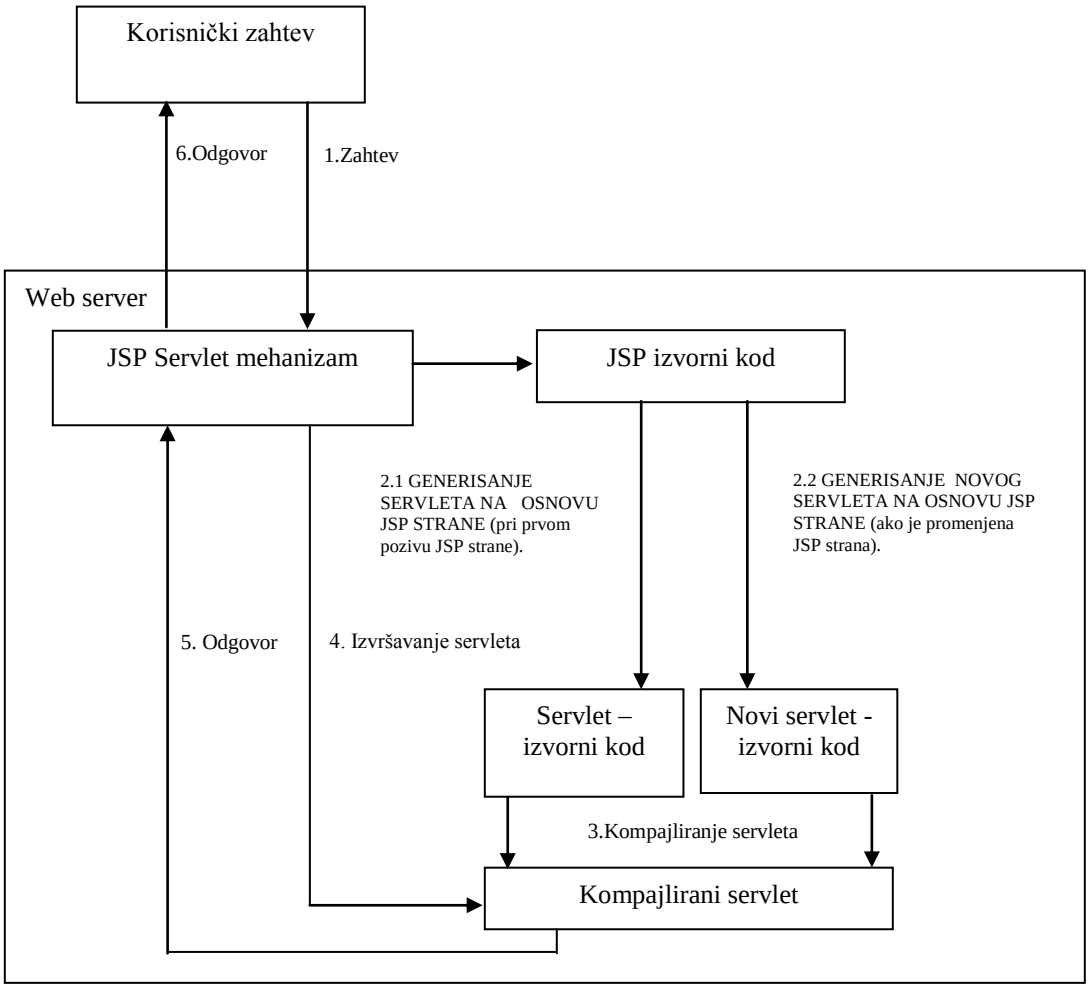
2.1 JSP (JavaServer Pages) – Java serverske strane

Web aplikacija se sastoji iz Web komponenti. Web komponente su servleti, JSP strane i Web servisi. JSP predstavlja kombinaciju markup (HTML ili XML) i Java programskog jezika koji kao rezultat daju dinamičke web strane. Svaka web strana se automatski kompajlira u servlet pomoću JSP mehanizma. Preko JSP-a je moguće ostvariti vezu sa Javinim klasama, servletima, apletima i sa Web serverom.
Da bi Web server mogao da koristi JSP komponente on mora da implementira JSP specifikaciju[1].

2.2.1 JSP arhitektura

JSP je tekst datoteka koja se sastoji iz HTML ili XML sadržaja i JSP elemenata. Postoje tri scenarija izvršenja JSP strane:
1. Klijent prvi put poziva JSP stranu. Klijent šalje zahtev do Web servera (1). Web server pokreće JSP mehanizam koji na osnovu JSP strane generiše servlet (2.1), koga nakon toga kompajlira (3). JSP mehanizam izvršava servlet (4), koji vraća neki rezultat (5). Rezultat se vraća nazad do klijenta (6).
2. Klijent zove JSP stanu za koju je već formiran servlet. Klijent šalje zahtev do Web servera (1). JSP mehanizam izvršava servlet (4), koji vraća neki rezultat (5). Rezultat se vraća nazad do klijenta (6).
3. Promenjena je JSP strana za koju je bio formiran servlet (servlet nije validan). Klijent šalje zahtev do Web servera (1). Web server pokreće JSP mehanizam koji na osnovu JSP strane generiše novi servlet (2.2), koga nakon toga kompajlira (3). JSP mehanizam izvršava servlet (4), koji vraća neki rezultat (5). Rezultat se vraća nazad do klijenta (6).

Na osnovu navedenog može da se zaključi da se za svaku JSP stranu generiše po jedan servlet.



[1] JSP 2.0 specifikacija je u ovom trenutku aktuelna.

Primer JSPPrimer1:
Korisnički zahtev: Napraviti HTML dokument preko koga ce biti pozvan odgovarajuca JSP strana

U datoteci JSPPrimer1.html nalazi se HTML kod koji poziva JSP stranu:

```
<html>
   <body>
      <p>JSP Primer 1 : Napraviti HTML dokument preko koga ce biti pozvana JSP strana</p>
      <form action="http://127.0.0.1:8084/JSPPrimer1/JSPPrimer1.jsp" method="POST">
         <p align="center"><input type="submit" value="Pozovi JSP stranu"></p>
      </form>
      <hr>
      <font size="1"><i>
      <b>Author:
         Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
          Copyright © 2005 All rights reserved.
      </b>
   </body>
</html>
```

Pozivanje JSP strana se radi na sličan način kao što je to rađeno kod servleta. Suštinska razlika se ogleda u tome što se JSP strane direktno pozivaju preko konteksta, dok se servlet poziva preko konteksta i simboličke adrese.

Datoteka *JSPPrimer1.jsp* ima sledeći izgled:

```
<html>
   <body>
      <h1>JSP - Dobar dan</h1>
   </body>
</html>
```

Navedena JSP strana treba da prikaže pozdravnu poruku. U ovom jednostavnom primeru je dat samo HTML kod u JSP strani. U daljem tekstu ćemo objasniti kako se uvodi Java programski kod u JSP strane.

2.2.2 Elementi JSP datoteka

Postoje sledeći elementi JSP-a:

- *Direktive (directives)* – obezbeđuju globalne informacije o strani. Npr. koje se klase uključuju (*import*) u JSP stranu, koje će se kodne strane koristiti(*contentType*), informacije o tome koji će se jezik koristiti u JSP strani(*language=”java”*),...
- *Deklaracije(declaratives)* - deklarišu se promenljive i metode u JSP stranama.
- *Izrazi (expressions)* – formatiraju se u stringove koji se uključuju u rezultat izvršenja JSP strane.
- *Skriptleti (sciptleti)* – java programski kod koji se ugrađuje u JSP stranu.

2.2.2.1 Direktive

JSP direktiva je izraz koji daje JSP mehanizmu informacije o JSP strani koja treba da se obradi. Generalna sintaksa JSP direktive je:

<%@ directive {attribute=”value”} %>

gde direktive mogu da imaju više atributa.
Moguće direktive su:

a) ***page*** – informacije za stranu.
b) ***include –*** datoteke koje trebaju da se uključe.
c) ***taglib*** – URI[2] za biblioteku tagova koja će se koristiti u strani..

[2] **U**niform **R**esource **I**dentifier je deo od URL-a koji ukazuje na internet resurs. Kod servleta URI ukazuje na deo URL-a koji se odnosi na adresu konteksta aplikacije i adresu servleta.

Direktiva **page** može da ima sledeće atribute: *language, extends, import, session, buffer, autoFlush, isThreadSafe, info, errorPage, isErrorPage* i *contentType.*

U nastavku ćemo objasniti svaki od navedenih atributa kroz nekoliko konkretnih primera.
****

- **language –** govori serveru koji će programski jezik biti korišćen. Java je za sada jedini programski jezik koji se može koristiti kod JSP strana, po JSP specifikaciji 2.0.

  Sintaksa: language=”java”

Primer JSPLanguage.
Programski zahtev: Napisati JSP stranu koja govori serveru da će se Java programski jezik koristiti kod JSP strana.

U datoteci *JSPLanguage.jsp* se unosi sledeći kod:

```
<%-- Server se obaveštava da će se koristiti Java programski jezik--%>[3]
<%@ page language="Java" %>

<%-- Html kod --%>
<html>
   <body>
      <p><b>HTML poruka</b></p>
   </body>
<%-- Java kod  se ugradjuje u JSP strane --%>
<% out.write("Java poruka"); %>
</html>
```

****

- **extends –** definiše roditeljsku klasu generisanog servleta. Treba biti oprezan kada se koristi ovaj atribut jer već postojeća (podrazumevana) roditeljska klasa (***HttpJspBase***) obezbeđuje potrebnu osnovu za funkcionisanje servleta. *Ne preporučuje se korišćenje ovog atributa.*

Sintaksa: extends=”package.class”

****

- **import –** uključuje pakete i klase u JSP stranu.

Sintaksa: import=” package., package.class”*

Primer JSPImport.
Programski zahtev: Napisati JSP stranu koja će da uključi HTML klasu, koja se nalazi u HTML paketu. Deklarisati promenljivu objekta tipa klase HTML. Prikazati preko tog objekta odgovarajuću poruku koja će biti poslata do klijenta.

U datoteci *JSPImport.jsp* se unosi sledeći kod:

```
<%@ page language="Java" %>
<%-- U JSP stranu se uključuje HTML paket koji sadrži HTML klasu. --%>
<%@ page import="HTML.*" %>

<%! HTML h = new HTML("");%>
<% h.add(HTML.NORMAL,"Koriscenje import atributa page direktive.",true);
out.println(h.prikaziStranu());
%>
```

Zadatak ZJSPImport: *Napraviti klasu Presek koja ima atribute rec1 i rec2 (navedeni atributi su String tipa) i metodu NadjiPresek koja kao rezultat vraća presek (zajedničke znakove) navedena 2 stringa. Vezati klasu presek za biblioteku klasa preko Library managera-a. Napraviti JSP stranu koja će uključiti klasu*

[3] Kod JSP-a komentari se obeležavaju na sledeći način: **<%-- *Komentar*--%>**

Presek preko import atributa page direktive. Kreirati objekat klase Presek i pozvati metodu NadjiPresek kojoj treba proslediti 2 proizvoljna stringa. Rezultat metode prikazati klijentu.

****
****

- **session –** podrazumevano je sesion atribut postavljen na true vrednost što znači da se session objekat servleta može koristiti i mogu se koristiti Java bean objekti. Ukoliko se postavi false session objekat se mora eksplicitno zadati ukoliko se želi koristiti. Java bean objekti u tom slučaju se ne mogu koristiti.

Sintaksa: session=”true | false”

Primer JSPSession.
Programski zahtev: Napisati JSP stranu koja će da testira atribut session u sledećim situacijama:

a) kada je atribut session postavljen na true a želi se koristiti Java bean.
b) kada je atribut session postavljen na false a želi se koristiti Java bean.
c) kada je atribut session postavljen na true a želi se koristiti sesion objekat.
d) kada je atribut session postavljen na false a želi se koristiti sesion objekat.

Sukcesivno u datotekama (*JSPSession1.jsp, JSPSession2.jsp, JSPSession3.jsp i JSPSession4.jsp*) daćemo rezultate navedenih zahteva.

a) kada je session postavljen na true a želi se koristiti Java bean.

Datoteka JSPSession1.jsb:

```
<%@ page language="Java" %>
<%-- U JSP stranu se uključuje session atribut page direktive i postavlja se na true. --%>
<%@ page session="true" %>

<%-- Deklarisanje promenljivih JSP strane --%>
<%! double zbir=0;
%>

<%-- Uključivanje Java bean-a. --%>
<jsp:useBean id ="z" scope = "session" class="AO.AritmetickeOperacije"/>

<%-- Izvršenje metode Java bean-a. --%>
<%  zbir = z.Saberi(zbir,1);
out.println("Zbir je: " + zbir);
%>
```

Izgled onih delova programa servleta Servleta JSPSession1_jsp koji ukazuju na sesiju:

```
...
// Paket javax.servlet.http  je potreban da bi se izvršila sesija.
import javax.servlet.http.*;

// Servlet klasa koja se generiše na osnovu JSP strane.
public final class JSPSession1_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

// Metoda koju poziva JSP mehanizam kada želi izvršiti servlet.
public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {
   ...
// Deklarisanje i inicijalizacija sesije.
     HttpSession session = null;
  ...

// Ukoliko je sesija nova ona se kreira inače se dobija referenca na već postojeću sesiju.
 session = pageContext.getSession();
```

```
// Sinhronizacija sesije u kojoj se Java bean objekat proglašava kontekstnom promenljivom koja se kreira i kojoj se pamte i dodaju vrednosti.
    synchronized (session) {
      z = (AO.AritmetickeOperacije) _jspx_page_context.getAttribute("z", PageContext.SESSION_SCOPE);
      if (z == null){     z = new AO.AritmetickeOperacije();
                          _jspx_page_context.setAttribute("z", z, PageContext.SESSION_SCOPE);
                  }
   ...
}
```

b) kada je session postavljen na false a želi se koristiti Java bean.

Datoteka JSPSession2.jsp:

```
<%@ page language="Java" %>
<%-- U JSP stranu se uključuje session atribut page direktive i postavlja se na true. --%>
<%@ page session="false" %>
...
Preostali kod je isti kao u primeru JSPSession1.jsp
```

Napomena: U navedenom slučaju nije moguće koristiti Java bean.

c) kada je atribut session postavljen na true a želi se koristiti sesion objekat.

Datoteka JSPSession3.jsb:

```
<%@ page language="Java" %>
<%@ page session="true" %>

<%! Integer broj;
%>

<%-- Koristi se postojeci session objekat jer je atribut session direktive page postavljen na true --%>
<%
broj = (Integer)session.getValue("Broj");
if (session.isNew())
 { broj = new Integer(0);
   out.println("Ovo je nova sesija. Broj: " + broj);

 }
else
{ broj = new Integer(broj.intValue()+1);
  out.println("Ovo je stara sesija. Broj: " + broj);
}%>
<%--Pamcenje broja u session objektu.--%>
<% session.putValue("Broj",broj);
%>
```

d) kada je atribut session postavljen na false a želi se koristiti sesion objekat.

Datoteka JSPSession4.jsb:

```
<%@ page language="Java" %>
<%@ page session="false" %>

<%--Mora da se deklarise objekat sesije jer je session atribut page direktive postavljen na false--%>
<%!
HttpSession sesija;
Integer broj;
%>

<%
sesija = request.getSession(true);
broj = (Integer)sesija.getValue("Broj");
if (sesija.isNew())
 {  broj=new Integer(0);
    out.println("Ovo je nova sesija. Broj: " + broj);
 }
```

```
else
{  broj = new Integer(broj.intValue()+1);
   out.println("Ovo je stara sesija. Broj: " + broj);
}
sesija.putValue("Broj",broj);
%>
```

Zadatak ZJSPSession: *Napraviti JSP stranu (Banka) koja će preko session objekta da prati u banci stanje klijenta. Omogućiti da JSP strana prihvata ImeKlijenta, VrstuTransakcije(Uplata,Isplata) i Iznos. U zavisnosti od vrste transakcije će se podizati(Uplata) odnosno spuštati (Isplata) stanje klijenta. Ne sme se dozvoliti da stanje klijenta uđe u minus.*

****

- **buffer –** određuje se veličina izlaznog toka ukoliko postoji. Podrazumevano je veličina 8kb. Ovaj atribut se koristi zajedno sa *autoFlush* atributom.

Sintaksa: buffer=”none | 8kb | sizekb”

Primer JSPBuffer.
Programski zahtev: Napisati JSP stranu kod koje će buffer parametar biti postavljen na 1kb. Atribut autoFlush postaviti na *false.*

U datoteci JSPBuffer.jsp se unosi sledeći kod:

```
<%@ page language="Java" %>

<%-- Definisanje veličine bafera izlaznog toka--%>
<%@ page buffer = "1kb" %>

<%-- Ukoliko se prepuni bafer desice se izuzetak i prekinuce se program.--%>
<%@ page autoFlush = "false" %>
<%  for(int i=0;i<1020;i++)
      out.print('A');
%>
```

Izgled onih delova programa servleta *JSPBuffer_jsp* koji ukazuju na baferovanje izlaznog toka:

```
...
public final class JSPBuffer_jsp extends org.apache.jasper.runtime.HttpJspBase
   implements org.apache.jasper.runtime.JspSourceDependent {

  public void _jspService(HttpServletRequest request, HttpServletResponse response)
      throws java.io.IOException, ServletException {


// Vidi se broj bajtova (1024) koje moze da prihvati bafer izlaznog toka.
pageContext = _jspxFactory.getPageContext(this, request, response,
                                  null, true, 1024, false);
...
     out.write('\n');
     out.write('\n');
     out.write('\n');
  for(int i=0;i<1020;i++)
      out.print('A');

...
```

Iz navedenog programa može da se zaključi da je izlazni bafer prihvatio 1023 znaka. Ukoliko bi pokušali da stavimo da indeks ciklusa ide do 1021:

```
 for(int i=0;i<1021;i++)
     out.print('A');
```

tada bi se desio izuzetak. To znači da se na 1024 bajtu koji bafer primi dešava izuzetak.

Navedeni problem se rešava tako što se *atribut autoFlush* postavi na *true:*

```
<%-- Ukoliko se prepuni bafer isti će biti ispražnjen (slanjem podataka do korisnika). Nakon toga se bafer opet iznova puni. --%>
<%@ page autoFlush = "true" %>
<%  for(int i=0;i<1021;i++)
      out.print('A');
%>
```

Zadatak ZJSPBuffer1: *Napisati JSP stranu kod koje će buffer parametar biti postavljen na 1kb. Atribut autoFlush* postaviti na *false. Napraviti program koji će da naruši postavljeno ograničenje bafera, što će dovesti do generisnja izuzetka. Izuzetak treba da se uhvati a podaci sa kojima je napunjen bafer treba da se pošalju do klijenta.*

Zadatak ZJSPBuffer2: *Šta radi navedeni program?*

```
<%@ page language="Java" %>
<%@ page buffer = "1kb" %>
<%@ page autoFlush = "false" %>

<%
  try {
       for(int i=0;i<100;i++)
       { out.println("AAAAAAAAAAAAA");
         out.flush();
         Thread.sleep(1000);
       }
     } catch(Exception e) {out.println("IZUZETAK");}
%>
```

Da li će se u ovom zadatku desiti izuzetak kao rezultat prekoračenja bafera?

****

- **autoFlush –** ukoliko je postavljen na true tada se izlazni bafer prazni kada se napuni. Ukoliko je postavljen na false desiće se izuzetak kada se napuni izlazni bafer.

Sintaksa: autoFlush=" true | false"

Primer JSPBuffer (urađen je kao primer koji je objašnjavao buffer atribut page direktive).

****

- **isThreadSafe –** podrazumevano je postavljen na *true*, što znači da više korisnika u isto vreme može da pristupi do JSP strane. U tom slučaju se mora voditi o eventualnoj sinhronizaciji deljenih atributa JSP strane. Ukoliko je postavljen na *false* tada samo jedan korisnik, u jednom trenutku, može da pristupi do JSP strane. Međutim i u tom slučaju treba voditi računa o sinhronizaciji jer se može desiti da zahtevi istog klijenta ne budu prosleđeni do iste instance JSP strane.

Sintaksa: isThreadSafe=" true | false"

Primer JSPisThreadSafe .

Programski zahtev: Napisti JSP stranu koja će demonstrirati kako više klijenata pristupa istoj JSP strani. Omogućiti da jedan od atributa JSP strane bude sinhronizovan.

U datoteci se unosi sledeći kod:

```
<%@ page language="Java" %>
<%@ page isThreadSafe="true" %>

<%! int deljeni=0;

synchronized void povecajDeljeni() {deljeni++;} %>
```

```
<%for(int i =0; i<10;i++)
{ povecajDeljeni();
  out.println("Sesija ID:" + session.getId()+ " broj:" + deljeni);
  Thread.sleep(1000);
}
%>
```

Zadatak ZJSPThread: *Napraviti preko JSP strane čet program koji će da omogući razmenu poruka između 2 ili više klijenata. Buffer parametar postaviti na 1kb. Atribut autoFlush postaviti na false. Kada se buffer napuni treba javiti poruku svim klijentima da je dalja razmena poruka onemogućena.*

****

- **info –** informacije o strani kojima se može pristupiti preko *Servlet.getServletInfo()* metode.

Primer JSPInfo.
Programski zahtev: Napisati JSP stranu za koju će biti vezana sledeća informacija: *JSP strana koja objasnjava info atribut page direktive.* Prikazati informaciju klijentu.

U datoteci *JSPInfo.jsp* se unosi sledeći kod:

```
<%@ page language="Java" %>
<%@ page info=" JSP strana  koja objasnjava info atribut page direktive." %>

Informacije o servletu:
<% out.println(this.getServletInfo());%>
```

****

- **errorPage –** relativna putanja do JSP strane gde će se obraditi neobrađeni izuzeci.

Sintaksa: errorPage=" pathToErrorPage"

****

- **isErrorPage –** markira se strana kao error strana.

Sintaksa: isErrorPage=" true | false"

****

- **contentType –** određuje se tip sadržaja strane i kod znakova (*podrazumevani je ISO-8859-1*) koji će se vratiti klijentu.

Sintaksa: contentType ="text/html;charset=ISO-8859-1"

****

- **isElIgnored –** Određuje se da li će se EL izrazi koji se nalaze na JSP strani obraditi ili ne.

Sintaksa: isElIgnored =" true | false"

****

2.2.2.2 Deklaracija (Declaration)

Deklarišu se promenljive i metode u JSP stranama.

Sintaksa: <%! declaration; [declaration;] ... %>

Primeri

```
<%! int broj = 5; %>
<%! char z1, z2; %>
<%! Klasa o = new Klasa(); %>
<%! int saberi(int a, int b) {return a+b;} %>
```

Deklariše se jedna ili više promenljivih i/ili metoda koje se koriste u JSP strani. Promenljive i metode se moraju deklarisati pre nego što će se koristiti.

Kod deklaracije se mora voditi računa o sledećim pravilima:

a) Kraj deklaracije se mora završiti sa tačka-zarez (;) znakom.

b) Mogu se koristiti promenljive ili metode koji su deklarisane u paketima koji su importovani u JSP stranu preko *page* direktive. Takve promenljive i metode se ne moraju deklarisati unutar JSP strana.

2.2.2.3 Izrazi (Expression)

Izrazi se definišu se unutar logike Java naredbi kako bi se prikazale određene vrednosti na JSP strani.

Sintaksa: <%= *expression* %>

Primeri:

```
<% for(int i=0;i<5;i++)
    { %>
  <BR>
      Vrednost indeksa: <%= i %>
<% } %>
```

JSP izrazi su veoma dobar alat za ugrađivanje vrednosti u HTML kod. Bilo šta između <%= i %> tagova će biti obrađeno (evaluirano), tako što će biti konvertovano u string i ugrađeno u response objekat. Konverzija iz primitivnog tipa u string se radi automatski.

2.2.2.4 Skriptlet (Scriptlet)

Skriptlet je validan Java programski kod koji se ugrađuje u JSP stranu.

Sintaksa: <% *Java programski kod* %>

Primer:

```
<%
sesija = request.getSession(true);
broj = (Integer)sesija.getValue("Broj");
if (sesija.isNew())
 {  broj=new Integer(0);
    out.println("Ovo je nova sesija. Broj: " + broj);
 }
else
{  broj = new Integer(broj.intValue()+1);
   out.println("Ovo je stara sesija. Broj: " + broj);
}
sesija.putValue("Broj",broj);
%>
```

Unutar skriptleta se mogu izvoditi sledeće akcije:

- Deklarišu se promenljive i metode koje će se kasnije koristiti u JSP strani.
- Pišu se validne Jave naredbe.
- Koriste se implicitni objekti (deklarisani izvan skriptleta) ili drugi objekti deklarisani kao <jsp:*useBean*> objekti.

Zadatak ZSkriptlet1: *Napraviti JSP stranu – skriptlet, koji će da prihvati i da sortira niz strigova. Sortirani niz treba prikazati klijentu.*

Implicitni objekti kojima može pristupiti skriptlet

Iz skriptleta se može pristupiti sledećim implicitnim objektima koji će biti generisani u servletu:

- **a)** ***request*** - Klijentski zahtev (HTTPServletRequest objekat).
- ***b)*** ***response*** – Odgovor klijentu (HTTPServletResponse objekat).
- ***c)*** ***pageContext*** – Kontekst strane. Iz konteksta se dobijaju brojne informacije o aplikaciji, servletu, sesiji, objektu izlaznog toka,...npr:

  application = pageContext.getServletContext();
  session = pageContext.getSession();
  out = pageContext.getOut();

- ***d)*** ***session –*** HTTP session objekat koji omogućava da se unutar njega kreiraju promenljive sesije.
- ***e)*** ***application –*** na osnovu *pageContext.getServletContext()* se dobija objekat same aplikacije unutar koje se koristi JSP strana.
- ***f)*** ***out –*** izlazni tok, koji se povezuje sa response objektom i pomoću koga se šalju podaci do klijenta.
- ***g)*** ***config –*** konfiguracioni objekat za stranu.
- ***h)*** ***page*** – objekat koji sadrži referencu na samoga sebe (slično *this* objektu u Javi).
- ***i)*** ***exception*** – objekat koji se prosleđuje do strane koja obrađuje greške koje se dešavaju u toku izvršenja programa.

Primer generisanog servleta

Na osnovu *JSPSession3.jsp*:

```
<%@ page language="Java" %>
<%@ page session="true" %>

<%!
Integer broj;
%>

<%
broj = (Integer)session.getValue("Broj");
if (session.isNew())
 { broj = new Integer(0);
   out.println("Ovo je nova sesija. Broj: " + broj);

 }
else
{ broj = new Integer(broj.intValue()+1);
  out.println("Ovo je stara sesija. Broj: " + broj);
}
 session.putValue("Broj",broj);
%>
```

generiše se servlet *JSPSession3_jsp:*

```
// Uključivanje paketa i klasa koje podržavaju rad jsp servleta.
package org.apache.jsp;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.servlet.jsp.*;
```

```
// Uključivanje klasa koje podržavaju rad servleta.
public final class JSPSession3_jsp extends org.apache.jasper.runtime.HttpJspBase
    implements org.apache.jasper.runtime.JspSourceDependent {

// Prebačena promenljiva iz JSP strane.
Integer broj;
  private static java.util.Vector _jspx_dependants;

  public java.util.List getDependants() {
    return _jspx_dependants;
  }

// Metoda koju poziva JSP servlet mehanizam kada klijent pošalje zahtev.
  public void _jspService(HttpServletRequest request, HttpServletResponse response)
        throws java.io.IOException, ServletException {

// Deklarisanje implicitnih objekata kojima može pristupiti skriptlet.
    JspFactory _jspxFactory = null;
    PageContext pageContext = null;
    HttpSession session = null;
    ServletContext application = null;
    ServletConfig config = null;
    JspWriter out = null;
    Object page = this;
    JspWriter _jspx_out = null;
    PageContext _jspx_page_context = null;
// Kreiranje implicitnih objekata.
    try {
      _jspxFactory = JspFactory.getDefaultFactory();
      response.setContentType("text/html");
      pageContext = _jspxFactory.getPageContext(this, request, response,
                                  null, true, 8192, true);
      _jspx_page_context = pageContext;
      application = pageContext.getServletContext();
      config = pageContext.getServletConfig();
      session = pageContext.getSession();
      out = pageContext.getOut();
      _jspx_out = out;

      out.write('\n');
      out.write('\n');
      out.write('\n');
      out.write('\n');
      out.write('\n');

// Prebačeni programski kod iz JSP strane.

broj = (Integer)session.getValue("Broj");
if (session.isNew())
 { broj = new Integer(0);
   out.println("Ovo je nova sesija. Broj: " + broj);
 }
else
{ broj = new Integer(broj.intValue()+1);
  out.println("Ovo je stara sesija. Broj: " + broj);
}
 session.putValue("Broj",broj);

      out.write('\n');
    } catch (Throwable t) {
      if (!(t instanceof SkipPageException)){
        out = _jspx_out;
        if (out != null && out.getBufferSize() != 0)
          out.clearBuffer();
        if (_jspx_page_context != null) _jspx_page_context.handlePageException(t);
      }
    } finally {
      if (_jspxFactory != null) _jspxFactory.releasePageContext(_jspx_page_context);
    }
  }
}
```

Primer JSPPrimer2.
Korisnički zahtev: Napraviti HTML dokument preko koga će biti pozvane odgovarajuće JSP strane koje će demonstrirati komentare, deklaracije, izraze i skriptlete kod JSP strana.

Prave se sledeće datoteke:
a) JSPPrimer2.html poziva JSP strane.

```
<html>
<body bgcolor="#C0C0C0" text="#000080">

<body>
<p>JSP Primer 1 : Napraviti HTML dokument preko koga ce biti pozvane odgovarajuce JSP strane
koje ce demonstrirati komentare, deklaracije, izraze i skriptlete kod JSP strana.</p>

<!poziv JSP strane>
<form action="http://127.0.0.1:8084/JSPPrimer2/Komentar.jsp" method="POST">
    <p align="center"><input type="submit" value="Primer komentara"></p>
</form>
<form action="http://127.0.0.1:8084/JSPPrimer2/Deklaracija.jsp" method="POST">
    <p align="center"><input type="submit" value="Primer deklaracije"></p>
</form>
<form action="http://127.0.0.1:8084/JSPPrimer2/Izrazi.jsp" method="POST">
    <p align="center"><input type="submit" value="Primer izraza"></p>
</form>

<form action="http://127.0.0.1:8084/JSPPrimer2/Skriptleti.jsp" method="POST">
      <i>Broj1</i>  <input type="text" size="15" name="Broj1"> <br><br>
      <i>Broj2</i>  <input type="text" size="15" name="Broj2"><br>
     <p align="center"><input type="submit" value="Primer skriptleta"></p>
</form>
<hr>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

b) Komentar.jsp objašnjava komentare kod JSP strana:

```
<%@ page language="java" %>
<html>
<head><title>Test za komentar</title></head>
<body>
<h2>Test za komentar</h2>
<%-- Komentar nece biti ugradjen u response objekat --%>
<!-- Komentar ce biti ugradjen u response objekat, ali ga nece videti klijent -->

</body>
</html>
```

Postoje dva načina prikazivanja komentara u JSP strani:

- ***<%-- komentar --%>***
- ***<!-- komentar -->***

c) Deklaracija.jsp prikazuje deklaracije kod JSP strana:

```
<%@ page language="java" %>
<html>
<head><title>Test za komentar</title></head>
<body>
<h2>Deklaracija</h2>
<%-- Deklaracija promenljivih --%>
<%! int i = 0; %>
<%! char b = 'a'; %>

Promenljiva i: <%= i %> <br>
Promenljiva b: <%= b %>
</body></html>
```

c) Izrazi.jsp prikazuje izraze kod JSP strana:

```
<%@ page language="java" %>
<html>
<head><title>Test za komentar</title></head>
<body>
<h2>Izrazi</h2>
<%-- Deklaracija --%>
<%! int i = 0; %>
<%! char b = 'a'; %>

<%-- Izrazi se stavljaju izmedju <%= i %> tagova--%>
Promenljiva i: <%= i %> <br>
Promenljiva b: <%= b %>

</body>
</html>
```

d) Skriptlet.jsp prikazuje skriptlet koji se ugrađuje u JSP stranu:

```
<%@ page language="java" %>
<%@ page import="HTML.*" %>

<html>
   <head><title>Test za skriptlet</title></head>
   <body>
      <h2>Skriptleti</h2>
      <%-- Deklaracija --%>

<%! double broj1=0, broj2=0,zbir=0;
HTML h = new HTML("Zbir dva broja:");

  public static float stringToFloat(String ulazniString) throws NumberFormatException
   { Float f = new Float(ulazniString);
     return f.floatValue();
   }
 %>

<%-- SKRIPTLET --%>
<%
 try {
      broj1 = stringToFloat(request.getParameter("Broj1"));
      broj2 = stringToFloat(request.getParameter("Broj2"));
    }  catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD UNOSA BROJEVA!!!" + broj1 + " " + broj2,true);
       out.println(h.prikaziStranu());  out.close(); }

    zbir = broj1 + broj2;

%>
      Zbir brojeva: <%= broj1 %> i <%= broj2 %> <br>
      je : <%= zbir %>
   </body>
</html>
```

2.2.3 Korišćenje Java Bean-ova kod JSP strana

Java bean-ovi su komponente koje se mogu koristiti u razlčitim aplikacijama. To su klase koje zadovoljavaju sledeće kriterijume:

a) Klasa im je javna (public).
b) Imaju javni konstruktor bez argumenata.
c) Metode kojima se pristupa iz aplikacija moraju biti javne.

Na primer:

```
package AO;
public class AritmetickeOperacije // a
{  double rezultat;
   public AritmetickeOperacije(){rezultat=0;} // b
  public double Saberi(double broj1, double broj2)    { rezultat = broj1 + broj2;      return rezultat;}  // c
  public double Oduzmi(double broj1, double broj2)    { rezultat = broj1 - broj2;     return rezultat;} // c
}
```

Java bean-ovi se čuvaju u jar datoteci. U navedenom primeru to je *AO.jar*.

U JSP strane se bean ubacuje na sledeći način:
<jsp:useBean id ="z" scope = "application" class="AO.AritmetickeOperacije"/>
Unutar JSP strane, u okviru skriptleta, objekat *z* poziva metode klase *ArtimetickeOperacije* na standardan način:

```
<%
...     zbir = z.Saberi(broj1,broj2);
  %>
```

JSP Primer 3.
Korisnički zahtev: *Napraviti HTML dokument preko koga ce biti pozvana JSP stana koja treba da izracuna zbir 2 broja preko Java bean-a.*
Prave se sledeće datoteke:
a) JSPPrimer3.html prihvata podatke i poziva JSP stranu.

```
<html>
<p>JSP Primer 3 : Napraviti HTML dokument preko koga ce biti pozvana JSP stana koja treba da izracuna
zbir 2 broja preko Java bean-a.
</p>

<form action="http://127.0.0.1:8084/JSPPrimer3/useBean.jsp" method="POST">
        <i>Broj1</i>
        <input type="text" size="15" name="Broj1"> <br><br>

        <i>Broj2</i>
        <input type="text" size="15" name="Broj2"><br>
        <p align="center"><input type="submit" value="Primer koriscenja Bean-a"></p>
</form>

<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade 
 Copyright © 2005 All rights reserved.
</b>

</body>
</html>
```

b) useBean.jsp treba da implementira JSP stranu koja preko Java bean-a treba da izračuna zbir brojeva.

```
<%@ page language="java" %>
<%@ page import="HTML.*" %>
<%@ page session="true" %>

<html>
<head><title>Test za koriscenje bean-ova</title></head>
<body>
<h2>Koriscenje bean-a</h2>
<%-- Deklaracija --%>

<%! double broj1=0, broj2=0,zbir=0;
 HTML h = new HTML("Zbir dva broja:");
  public static float stringToFloat(String ulazniString) throws NumberFormatException
    { Float f = new Float(ulazniString);
      return f.floatValue();
    }
 %>

<%-- Uključivanje Java bean-a --%>
<jsp:useBean id ="z" scope = "session" class="AO.AritmetickeOperacije"/>
```

```
<%

 try {
      broj1 = stringToFloat(request.getParameter("Broj1"));
      broj2 = stringToFloat(request.getParameter("Broj2"));
    }  catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD UNOSA BROJEVA!!!" + broj1 + " " + broj2,true);
       out.println(h.prikaziStranu());  out.close(); }
   <%-- Pozivanje metode Java bean-a --%>
    zbir = z.Saberi(broj1,broj2);

%>

Zbir brojeva: <%= broj1 %> i <%= broj2 %> <br>
je : <%= zbir %>

</body>
</html>
```

c) *AritmetičkeOperacije.java* treba da implementira metodu Saberi():

```
package AO;

import java.text.*;
public class AritmetickeOperacije
{ double rezultat;
  double broj1;
  double broj2;
  DecimalFormat df = new DecimalFormat(".###");
  public AritmetickeOperacije(){rezultat=0.0;broj1=0.0;broj2=0.0;}
  public double Saberi(double broj1, double broj2)
    { this.broj1=broj1;  this.broj2=broj2;
      rezultat = broj1 + broj2;
      return rezultat;}
  public double Oduzmi(double broj1, double broj2)
    {this.broj1=broj1;  this.broj2=broj2;
     rezultat = broj1 - broj2;
      return rezultat;}
  public String vratiBroj1() {return df.format(broj1);}
  public String vratiBroj2() {return df.format(broj2);}
  public String vratiRezultat() {return df.format(rezultat);}
 }
```

Zadatak ZJSPBean: *Napraviti HTML dokument preko koga će biti pozvana JSP stana, kojoj se šalje proizvoljna reč. Treba izračunati koliko se puta pojavljuju znakovi u reči preko klase ObradiRec, koja je uključena u JSP stranu kao Java bean objekat.*

JSP Primer 4.
Korisnički zahtev: Napraviti JSP stranu (tekuća) preko koje ce biti prihvacena dva broja. Poslati ta dva broja do druge JSP strane (sabira ta dva broja) koja ce biti ukljucena u tekucu JSP stranu.

Prave se sledeće datoteke:
a) JSPPrimer4.jsp prihvata podatke i uključuje JSP stranu *SaberiBrojeve.jsp*.

```
<html>
<head><title>Ukljucenje JSP strane u JSP stranu</title></head>
<body>

<body bgcolor="#C0C0C0" text="#000080">

<p>JSP Primer 4 : Napraviti JSP stranu preko koje ce biti prihvacena dva broja.
Poslati ta dva broja do druge JSP strane (sabira ta dva broja ) koja ce biti ukljucena u tekucu JSP stranu.
</p>

<%-- Uključivanje jsp datoteke. --%>
```

```
<%@ include file="SaberiBrojeve.jsp" %>

<%-- Prihvat brojeva i prikaz rezultata sabiranja. --%>
<form action="http://127.0.0.1:8084/JSPPrimer4/JSPPrimer4.jsp" method="POST">
        <i>Broj1</i>  <input type="text" size="15" name="Broj1" value= <%= z.vratiBroj1() %> > <br><br>

        <i>Broj2</i>  <input type="text" size="15" name="Broj2" value= <%= z.vratiBroj2() %> ><br><br>

        <i>Zbir</i>   <input type="text" size="15" name="Zbir" value= <%= z.vratiRezultat() %>  ><br>
<%-- Samopoziv (rekurzivni poziv) JSPPrimer4.jsp datoteke kako bi se izracunao zbir brojeva. Jsp strana
šalje sama sebi parametre Broj1 i Broj2.--%>
        <p align="center"><input type="submit" value="Saberi"></p>

</form>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade&amp;nbsp;
&amp;nbsp;Copyright © 2005 All rights reserved.
</b>
</body>
</html>
```

b) SaberiBrojeve.jsp računa zbir brojeva:

```
<%@ page language="java" %>
<%-- Uključivanje paketa HTML. --%>
<%@ page import="HTML.*" %>


<%-- Deklaracija promenljivih, objekta h i metode stringToFloat() --%>
<%! double broj1=0, broj2=0;
 HTML h = new HTML("Zbir dva broja:");
  public static float stringToFloat(String ulazniString) throws NumberFormatException
    { Float f = new Float(ulazniString);
      return f.floatValue();
    }
 %>

<%-- Definisanje Java bean objekta z klase AritmetickeOperacije koji se nalazi u paketu AO.--%>
<jsp:useBean id ="z" scope = "page" class="AO.AritmetickeOperacije"/>

<%
if (request.getParameter("Broj1") == null || request.getParameter("Broj2") ==null)
{}
else
{%>
 <%-- Konvertovanje prihvaćenih brojeva. --%>
 <%try {
      broj1 = stringToFloat(request.getParameter("Broj1"));
      broj2 = stringToFloat(request.getParameter("Broj2"));
    }  catch(Exception e) {h.add(HTML.NORMAL, "GRESKA KOD UNOSA BROJEVA!!!" + broj1 + " " + broj2,true);
       out.println(h.prikaziStranu());  out.close(); }
}%>
<%-- Poziv metode Java bean objekta koji računa zbir dva broja. --%>
 <%z.Saberi(broj1,broj2);
%>
```

JSPPrimer4.jsp

```
<html>
<head><title>Ukljucenje JSP strane u JSP stranu</title></head>
<body>
<body bgcolor="#C0C0C0" text="#000080">
<p>JSP Primer 4 : Napraviti JSP stranu preko koje ce biti prihvacena dva
broja. Poslati ta dva broja do druge JSP strane koja ce biti ukljucena u tekucu
JSP stranu.
</p>
            <%@ include file="SaberiBrojeve.jsp" %>
```

SaberiBrojeve.jsp

```
<%@ page language="java" %>
<%-- Uključivanje paketa HTML. --%>
<%@ page import="HTML.*" %>
...

 z.Saberi(broj1,broj2);
 %>
```

```
<form action="http://127.0.0.1:8084/JSPPrimer4/JSPPrimer4.jsp"
method="POST">
 <i>Broj1</i>  <input type="text" size="15" name="Broj1" value= <%=
z.vratiBroj1() %> > <br><br>
...
</b>
</body>
</html>
```

Zadatak ZJSP1: Napraviti JSP stranu (tekuća) preko koje ce biti prihvacen string. Poslati string do druge JSP strane (računa kolika je dušina stringa) koja ce biti ukljucena u tekucu JSP stranu.

JSP Primer5.
Korisnički zahtev: *Napraviti JSP stranu preko koje ce biti napunjen kombo boks Zanimanje.*

Pravi se datoteka JSPPrimer.jsp:

```
<html>
<body bgcolor="#C0C0C0" text="#000080">
<%-- Deklarisanje niza stringova, koji predstavljaju zanimanja. --%>
<%!String[] st = {"Student","Profesor","Programer","Sluzbenik","Drugo"}; %>

<p>JSP Primer5: Napraviti JSP stranu preko koje ce biti napunjen kombo boks Zanimanje.</p>
<table>
        <%-- Definisanje kombo boksa i njegovo punjenje. --%>
        <td width="127">
        <p><i>Zanimanje</i>
        <p><select name="Zanimanje" size="1">
        <option selected><%=st[0]%></option>
        <% for(int i=1; i<5; i++)
        {%>
        <%-- primer korišćenja izraza --%>
        <option><%=st[i]%></option>
        <%}%>
</table>
<hr>
<font size="1"><i>
<b>Author:
Dr Sinisa Vlajic, Faculty of organizational sciences, Belgrade&amp;nbsp;
&amp;nbsp;Copyright © 2005 All rights reserved.
</b>
</body>
```

Zadatak ZJSP2: *Za niže navedeni program odgovorite na pitanja.*

```
<%-- Ukljucivanje paketa HTML. --%>
<%@ page import="HTML.*" %>

<%!String[] st = {"Student","Profesor","Programer","Sluzbenik","Drugo"};
 HTML h = new HTML("Punjenje kombo boksa:");
%>

<%
h.add(HTML.HEADING,"Punjenje kombo boksa",true);
h.add(HTML.NORMAL,"JSP Primer51: Napraviti JSP stranu preko koje ce biti napunjen kombo boks Zanimanje uz
koriscenje klase HTML",true);
out.println(h.prikaziStranu());
h.Isprazni();
%>

<table>
   <td width="127">
   <p><i>Zanimanje</i>
   <p><select name="Zanimanje" size="1">
   <option selected><%=st[0]%></option>
         <% for(int i=1; i<5; i++)
         {%>
   <option><%=st[i]%></option>
         <%}%>
</table>
<%
h.add(HTML.LINE,"",true);
h.add(HTML.AUTOR,"",false);
out.println(h.prikaziStranu());
%>
```

1. Šta bi se desilo kada ne bi uključili narebdu `<%@ page import="HTML.*" %>`

2. Šta bi se desilo kada bi umesto:

```
<%
h.add(HTML.HEADING,"Punjenje kombo boksa",true);
...
h.Isprazni();
%>
```

stavili:

```
<% !
h.add(HTML.HEADING,"Punjenje kombo boksa",true);
...
h.Isprazni();
%>
```

3. Koju naredbu možemo koristiti umesto(koji implicitni objekat skriptleta):

```
h.add(HTML.NORMAL,"JSP Primer51: Napraviti JSP stranu preko koje ce biti napunjen kombo boks Zanimanje uz
koriscenje klase HTML",true);
```

JSP Primer6.
Korisnički zahtev: Napraviti JSP stranu koja će parametarski pozvati drugu JSP stranu (puni kombo boks vrednostima iz baze podataka) i proslediti joj parametre o zaglavlju i komenatar o programu.

Prave se sledeće datoteke:

a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavljeDirektnoJSP.jsp)

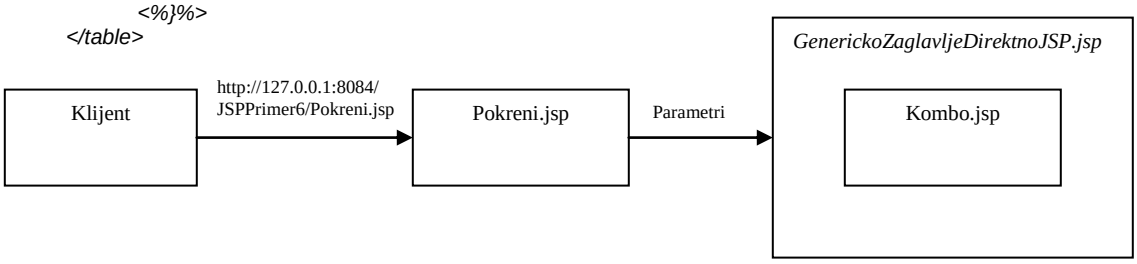
```
<%-- Poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavljeDirektnoJSP.jsp">

<%-- Definisanje parametara koji se prosleđuju drugoj strani. --%>
<jsp:param name="Zaglavlje" value="Punjenje kombo boksa" />
<jsp:param name="KomentarOProgramu" value="JSP Primer6: Napraviti JSP stranu preko koje ce biti napunjen kombo boks Zanimanje " />
<jsp:param name="NazivJSP" value="Kombo.jsp" />
</jsp:forward>
```

b) GenerickoZaglavljeDirektnoJSP.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje jsp stranu (Kombo.jsp) i prikazuje informacije o autoru aplikacije.

```
<%-- Uključivanje paketa HTML. --%>
<%@ page import="HTML.*" %>

<%-- Deklaracija objekta HTML klase. --%>
<%! HTML h = new HTML("");%>

<%-- Prikaz informacija o zaglavlju. --%>
<%
h.add(HTML.HEADING,request.getParameter("Zaglavlje"),true);
h.add(HTML.NORMAL,request.getParameter("KomentarOProgramu"),true);
out.println(h.prikaziStranu());
%>

<%-- Uključivanje jsp datoteke. --%>
<jsp:include page="Kombo.jsp" />

<%-- Prikaz komentara o autoru --%>
<%
h.add(HTML.LINE,"",true);
h.add(HTML.AUTOR,"",false);
out.println(h.prikaziStranu());
%>
```

c) Kombo.jsp prikazuje kombo boksa sa vrednostima zanimanja.

```
<%!String[] st = {"Student","Profesor","Programer","Sluzbenik","Drugo"};%>
<table>
        <td width="127">
        <p><i>Zanimanje</i>
        <p><select name="Zanimanje" size="1">
        <option selected><%=st[0]%></option>

        <% for(int i=1; i<5; i++)
        {%>
        <option><%=st[i]%></option>
        <%}%>
</table>
```



Zadatak ZJSP3: *Napraviti JSP stranu koja će parametarski pozvati drugu JSP stranu (analizira komentar o programu – traži da li se u komentaru programa nalazi termin: "baza podataka") i proslediti joj parametre o zaglavlju i komenatar o programu.*

JSP Primer71.
Korisnički zahtev: Napraviti JSP stranu koja će parametarski pozvati drugu JSP stranu i proslediti joj parametre o zaglavlju i komenatar o programu. Preko druge JSP strane ce biti napunjen kombo boks Zanimanje preko baze podataka.

Prave se sledeće datoteke:
a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavlje1.jsp)

```
<%-- poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavlje1.jsp">

<%-- definisanje parametara koji se prosleđuju drugoj strani. --%>
<jsp:param name="Zaglavlje" value="Punjenje kombo boksa" />
 <jsp:param name="KomentarOPrograma" value="JSP Primer6: Napraviti JSP stranu koja će parametarski
pozvati drugu JSP stranu (puni kombo boks vrednostima iz baze podataka) i proslediti joj parametre o zaglavlju i
komenatar o programu." />
<jsp:param name="NazivJSP" value="Kombo.jsp" />
<jsp:param name="ImeBaze" value="Baza" />
<jsp:param name="ImeTabele" value="Zanimanje" />
<jsp:param name="ImeAtributa" value="Sifra" />

</jsp:forward>
```

b) GenerickoZaglavlje1.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje jsp stranu (Kombo.jsp) i prikazuje informacije o autoru aplikacije.

```
<%-- Uključivanje paketa HTML. --%>
<%@ page import="HTML.*" %>

<%-- Deklarisanje objekta HTML klase. --%>
<%! HTML h = new HTML("");%>

<%-- Prikaz informacija o zaglavlju. --%>
<%
h.add(HTML.HEADING,request.getParameter("Zaglavlje"),true);
h.add(HTML.NORMAL,request.getParameter("KomentarOPrograma"),true);
out.println(h.prikaziStranu());
%>

<%-- Preko navedene naredbe postižemo punu generalnost strane. Uključuje se strana čiji je naziv parametar.--%>
<%
 org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, request.getParameter("NazivJSP"),
out, false);
 %>

<%-- Prikaz komentara o autoru --%>
<%
h.add(HTML.LINE,"",true);
h.add(HTML.AUTOR,"",false);
out.println(h.prikaziStranu());
%>
```

c) Kombo.jsp prikazuje kombo boksa sa vrednostima zanimanja preko baze podataka.

```
<%-- Uključivanje paketa DB. --%>
 <%@ page import="DB.*" %>

<%-- deklarisanje promenljivih i objekata. --%>
<%!String[] st;
String st1[];
DBBrokerSp db = new DBBrokerSp();
int signal;
%>
```

```
<%-- otvaranje baze i čitanje slogova. --%>
<%

signal = db.otvoriBazu(request.getParameter("ImeBaze"));
if (signal!=41)
  {}
else
{ st1 = db.nadjiSlogove(request.getParameter("ImeTabele"), request.getParameter("ImeAtributa"));
  if (st1 == null)
     {}
  else
     st=st1;
  db.zatvoriBazu();
}
%>

<%-- punjenje kombo boksa sa vrednostima zanimanja --%>
<table>
        <td width="127">
        <p><i>Zanimanje</i>
        <p><select name="Zanimanje" size="1">


        <% for(int i=0; i<st.length; i++)
          if (i==0)
            {%> <option selected><%=st[0]%></option>
          <%}
          else
           {%>
              <option><%=st[i]%></option>
         <%}%>
</table>
```

Zadatak ZJSPBP1.

Objasniti šta se dešava sa programom:

```
signal = db.otvoriBazu(request.getParameter("ImeBaze"));
if (signal!=41)
   {}
else
{ st1 = db.nadjiSlogove(request.getParameter("ImeTabele"), request.getParameter("ImeAtributa"));
  if (st1 == null)
     {}
  else
     st=st1;
  db.zatvoriBazu();
}
```

kada ga poziva prvi a posle toga drugi klijent?

Zadatak ZJSPBP2: *Napraviti JSP stranu koja će parametarski pozvati drugu JSP stranu i proslediti joj parametre o zaglavlju i komenatar o programu. Preko druge JSP strane napuniti kombo boks Imena iz baze podataka sa imenima (tabela PoslovniPartner) čiji e-mail se zavrsava sa fon.bg.ac.yu. Drugim rečima prikazati imena partnera koji imaju otvoren nalog e-mail-a preko Fon-a.*

Zadatak ZJSPBP3: *Napraviti JSP stranu koja će parametarski pozvati drugu JSP stranu i proslediti joj parametre o zaglavlju i komenatar o programu. Preko druge JSP strane napuniti kombo boks Imena iz baze podataka (tabela Covek) sa Imenima ljudi koji su radnici, stariji od 25 godina i muskog su pola.*

JSP Primer8.

Korisnički zahtev: *Napraviti JSP stranu preko koje će biti prihvaćen i izvršen proizvoljni SQL upit koji vraća neki rezultat (tip upita:Select-From-Where).*

Prave se sledeće datoteke:

a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavlje1.jsp)

```
<%-- poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavlje1.jsp">

<%-- definisanje parametara koji se prosleđuju drugoj strani. --%>

<jsp:param name="Zaglavlje" value="Unos i izvrsenje SQL upit" />
<jsp:param name="KomentarOPrograma" value="JSP Primer8: Napraviti JSP stranu preko koje ce biti prihvacen
                i izvrsen proizvoljni SQL upit koji vraca neki rezultat (tip upita:Select-From-Where)" />
<jsp:param name="NazivJSP" value="UnIzvSQL.jsp" />
<jsp:param name="ImeBaze" value="Baza" />
</jsp:forward>
```

b) GenerickoZaglavlje1.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje html i jsp stranu (Unos.html i UnIzvSQL.jsp(implicitno)) i prikazuje informacije o autoru aplikacije.

```
<%-- Uključivanje paketa HTML. --%>
<%@ page import="HTML.*" %>

<%-- Deklarisanje objekta HTML klase. --%>
<%! HTML h = new HTML("");%>

<%-- Prikaz informacija o zaglavlju. --%>
<%
h.add(HTML.HEADING,request.getParameter("Zaglavlje"),true);
h.add(HTML.NORMAL,request.getParameter("KomentarOPrograma"),true);
out.println(h.prikaziStranu());
%>

<%-- Uključuje se html datoteka. --%>
<jsp:include page="Unos.html" />

<%-- Preko navedene naredbe postižemo punu generalnost strane. Uključuje se strana čiji je naziv parametar.--%>
<%
 org.apache.jasper.runtime.JspRuntimeLibrary.include(request, response, request.getParameter("NazivJSP"),
out, false);
 %>

<%-- Prikaz komentara o autoru --%>
<%
h.add(HTML.LINE,"",true);
h.add(HTML.AUTOR,"",false);
out.println(h.prikaziStranu());
%>
```

c) UnIzvSQL.jsp izvršava i prikazuje rezultat upita.

```
<%-- Uključivanje paketa koji su potrebni za rad sa bazom podataka. --%>
<%@ page import="DB.*" %>
<%@ page import = "java.sql.*" %>

<%-- deklarisanje promenljivih i objekata. --%>
<%!String[] st;
String st1[];
DBBrokerSp db = new DBBrokerSp();
ResultSet rs;
ResultSetMetaData rsmd;
int signal;
int brojKolona;
%>
```

```
<%-- otvaranje baze i izvršavanje upita i prikazivanje rezultata upita. --%>

<%
signal = db.otvoriBazu(request.getParameter("ImeBaze"));
if (signal!=41 || request.getParameter("SQLUpit") == null)
  {}
else
{ %>

 <tr> Izvrsen je upit: <b><i><%=request.getParameter("SQLUpit")%> </b></i><hr></tr>
  <%
  try {
       rs = db.izvrsiUpit(request.getParameter("SQLUpit"));
       if (rs==null)
            {}
       else
          {
               rsmd = rs.getMetaData();
               brojKolona = rsmd.getColumnCount();%>
              <table border = 1>
              <tr>
              <%for (int i = 1; i<=brojKolona; i++)
                          { %>
                           <th align="LEFT"> <%= rsmd.getColumnName(i) %> </th>
                           <%
                           }
               %>
              </tr>
              <%
              while(rs.next())
              {
              %>
                          <tr>
                          <%
                           for (int i = 1; i<=brojKolona; i++)
                              {
                           %>
                                   <td> <%=rs.getString(i)     %> </td>
                           <%
                             }
                           %>
                           </tr>
          <%
              } // zatvara while
         rs.close();
       }   // zatvara else
  } catch(SQLException e) {}
 %>
 </table>
 <%}%>
```

d) Unos.html prihvata upit i prosleđuje ga do jsp strana da ga obrade.

```
<html>
<body>
<form action="http://127.0.0.1:8084/JSPPrimer8/Pokreni.jsp" method="POST">
<p><strong>Unesite SQL upit:</strong></p>
<dl>
  <dd><textarea name="SQLUpit" rows="5" cols="42"></textarea></dd>
</dl>
<p><input type="submit" value="Izvrsi SQL upit"></p>
</form>

</body>
</html>
```

Zadatak ZJSPBP4: *Napraviti JSP stranu (Banka) koja će da prati u banci stanje klijenta. Stanje klijenta pratiti preko baze podataka. Omogućiti da JSP strana prihvata ImeKlijenta, VrstuTransakcije (Uplata,Isplata) i Iznos. U zavisnosti od vrste transakcije će se podizati(Uplata) odnosno spuštati (Isplata) stanje klijenta. Ne sme se dozvoliti da stanje klijenta uđe u minus. Pored stanja klijenta u bazi podataka treba pratiti i istoriju njegovih uplata i isplata. Omogućiti klijentu da može videti navedenu istoriju.*

JSP Primer91.
Korisnički zahtev: *Napraviti JSP stranu preko koje će biti prihvaćen i zapamćen slog Čovek u bazi.*

Prave se sledeće datoteke:
a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavlje1.jsp)

```
<%-- poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavlje1.jsp">

<%-- definisanje parametara koji se prosleđuju drugoj strani. --%>
<jsp:param name="Zaglavlje" value="Pamcenje sloga Covek u bazi" />
<jsp:param name="KomentarOPrimeru" value="JSP Primer91: Napraviti JSP stranu preko koje ce biti
                    prihvacen i  zapamcen slog Covek u bazi." />
<jsp:param name="NazivJSP" value="Zapamti.jsp" />
<jsp:param name="ImeBaze" value="Baza" />
</jsp:forward>
```

b) GenerickoZaglavlje1.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje html i jsp stranu (Unos.html i Zapamti.jsp(implicitno)) i prikazuje informacije o autoru aplikacije.

Isto kao u primeru JSPPrimer8.

c) Zapamti.jsp pamti novog čoveka u bazi.

```
<%-- Uključivanje paketa koji su potrebni za rad sa bazom podataka. --%>
<%@ page import="DB.*" %>
<%@ page import = "java.sql.*" %>

<%-- deklarisanje promenljivih i objekata. --%>
<%!String[] st;
String st1[];
DBBrokerSp db = new DBBrokerSp();
int signal;
%>

<%-- otvaranje baze i pamcenje sloga u bazu. --%>
<%
signal = db.otvoriBazu(request.getParameter("ImeBaze"));
if (signal!=41 || request.getParameter("Ime") == null || request.getParameter("Ime").equals(""))
  {%> <p>Nije uneto ime ili baza nije uspesno otvorena!</p>
<%}
else
{ signal = db.pamtiSlog("Covek", "'" + request.getParameter("Ime") + "', '" + request.getParameter("BrojGodina")
+ "', '" + request.getParameter("Pol") + "', '" + request.getParameter("Zanimanje") + "' ");
 if (signal == 32)
    { db.rollbackTransakcije();%> <p> Neuspesno zapamcen slog u bazi!</p>
    <%}
 else
    { db.commitTransakcije();%> <p> Uspesno zapamcen slog u bazi! </p>
  <%}
 //db.zatvoriBazu();

 }%>
```

d) Unos.html preko koje se prihvataju podaci o Čoveku koji treba da bude zapamćen.

```
<html>
<body>

<form action="http://127.0.0.1:8084/JSPPrimer91/Pokreni.jsp" method="POST">
<p><strong>Unesite i zapamtite podatke o coveku:</strong></p>
```

```
<table border = 1>
        <tr> <td> <i>Ime</i>  <td width=50 ><input type="text"size =50 name= Ime >   </tr>
        <tr> <td> <i>BrojGodina</i> <td width=11 ><input type="text"size =11 name= BrojGodina >  </tr>
        <tr>  <td> <i>Pol</i>   <td width=1 ><input type="text"size =1 name= Pol >  </tr>
        <tr>  <td> <i>Zanimanje</i>  <td width=50 ><input type="text"size =50 name= Zanimanje >   </tr>
 </table>
  <p><input type="submit" value="Zapamti"></p>
</form>
</body>
</html>
```

JSP Primer92.

Korisnički zahtev: *Napraviti JSP stranu preko koje će biti prihvaćen i promenjen slog Čovek u bazi.*

Prave se sledeće datoteke:

a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavlje1.jsp)

```
<%-- poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavlje1.jsp">

<%-- definisanje parametara koji se prosleđuju drugoj strani. --%>
   <jsp:param name="Zaglavlje" value="Promena sloga Covek u bazi" />
   <jsp:param name="KomentarOProgramu" value="JSP Primer92: Napraviti JSP stranu preko koje ce biti
                        prihvacen i promenjen slog Covek u bazi" />
   <jsp:param name="NazivJSP" value="Promeni.jsp" />
   <jsp:param name="ImeBaze" value="Baza" />
</jsp:forward>
```

b) GenerickoZaglavlje1.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje html i jsp stranu (Unos.html i Promeni.jsp(implicitno)) i prikazuje informacije o autoru aplikacije.

Isto kao u primeru JSPPrimer8.

c) Promeni.jsp menja vrednosti izabranog čoveka u bazi.

```
<%-- Uključivanje paketa koji su potrebni za rad sa bazom podataka. --%>
<%@ page import="DB.*" %>
<%@ page import = "java.sql.*" %>

<%-- deklarisanje promenljivih i objekata. --%>
<%!String[] st;
String st1[];
DBBrokerSp db = new DBBrokerSp();
int signal;
%>

<%-- otvaranje baze i promena sloga u bazu. --%>
 signal = db.promeniSlog("Covek", "Ime = '" + request.getParameter("Ime") + "', BrojGodina = '" +
request.getParameter("BrojGodina") + "', Pol = '" + request.getParameter("Pol") + "', Zanimanje = '" +
request.getParameter("Zanimanje") + "' ", " Ime = '" + request.getParameter("Ime") + "'");
  if (signal == 36)
     { db.rollbackTransakcije();%> <p> Neuspesno promenjen slog u bazi!</p>
     <%}
  else
     { db.commitTransakcije();%> <p> Uspesno promenjen slog u bazi! </p>
    <%}
  }%>
```

d) Promeni.html preko koje se prihvataju podaci o Čoveku koji treba da bude promenjen.

```
<html>
<body>

<form action="http://127.0.0.1:8084/JSPPrimer92/Pokreni.jsp" method="POST">
<p><strong>Unesite i promenite podatke o coveku:</strong></p>
```

```
<table border = 1>
        <tr>  <td> <i>Ime</i>     <td width=50 ><input type="text"size =50 name= Ime >       </tr>
        <tr>  <td> <i>BrojGodina</i>  <td width=11 ><input type="text"size =11 name= BrojGodina >      </tr>
        <tr>  <td> <i>Pol</i>  <td width=1 ><input type="text"size =1 name= Pol >   </tr>
        <tr>  <td> <i>Zanimanje</i>  <td width=50 ><input type="text"size =50 name= Zanimanje >  </tr>
  </table>
  <p><input type="submit" value="Promeni"></p>
</form>
</body>
</html>
```

JSP Primer93.

Korisnički zahtev: *Napraviti JSP stranu preko koje će biti obrisan slog Čovek u bazi.*

Prave se sledeće datoteke:

a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavlje1.jsp)

```
<%-- poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavlje1.jsp">

<%-- definisanje parametara koji se prosleđuju drugoj strani. --%>
   <jsp:param name="Zaglavlje" value="Brisanje sloga Covek u bazi" />
   <jsp:param name="KomentarOProgramu" value="JSP Primer93: Napraviti JSP stranu preko koje ce biti
                        obrisan slog Covek u bazi" />
   <jsp:param name="NazivJSP" value="Obrisi.jsp" />
   <jsp:param name="ImeBaze" value="Baza" />
</jsp:forward>
```

b) GenerickoZaglavlje1.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje html i jsp stranu (Unos.html i Obrisi.jsp(implicitno)) i prikazuje informacije o autoru aplikacije.

Isto kao u primeru JSPPrimer8.

c) Obrisi.jsp brise izabranog čoveka iz baze.

```
<%-- Uključivanje paketa koji su potrebni za rad sa bazom podataka. --%>
<%@ page import="DB.*" %>
<%@ page import = "java.sql.*" %>

<%-- deklarisanje promenljivih i objekata. --%>
<%!String[] st;
String st1[];
DBBrokerSp db = new DBBrokerSp();
int signal;
%>

<%-- otvaranje baze i brisanje sloga u bazu. --%>
  signal = db.brisiSlog("Covek", " Ime = '" + request.getParameter("Ime") + "'");
  if (signal == 34)
     { db.rollbackTransakcije();%> <p> Neuspesno obrisan slog u bazi!</p>
     <%}
  else
     { db.commitTransakcije();%> <p> Uspesno obrisan slog u bazi! </p>
    <%}
  }%>
```

d) Unos.html preko koje se prihvataju podaci o Čoveku koji treba da bude obrisan.

```
<html>
<body>
<form action="http://127.0.0.1:8084/JSPPrimer93/Pokreni.jsp" method="POST">
<p><strong>Brisanje podataka o coveku:</strong></p>
<table border = 1>
        <tr>   <td> <i>Ime</i>  <td width=50 ><input type="text"size =50 name= Ime >     </tr>
 </table>
  <p><input type="submit" value="Obrisi"></p>
</form>
</body>
</html>
```

Zadatak ZJSPBP5: *Napraviti JSP stranu preko koje će moći da se izvrše operacije ubaci, izbaci, promeni nad vašim novokreiranim objektom. Poželjno je da se sve operacije mogu izvršiti nad jednom ekranskom formom.*

JSP Primer10.

Korisnički zahtev: *Napraviti JSP stranu preko koje će biti prihvaćen naziv objekta za koji će biti generisana ekranska forma*

Prave se sledeće datoteke:

a) Pokreni.jsp pokreće drugu jsp stranu (GenerickoZaglavlje1.jsp)

```
<%-- poziv druge jsp strane (preusmeravanje poziva). --%>
<jsp:forward page="GenerickoZaglavlje1.jsp">

<%-- definisanje parametara koji se prosleđuju drugoj strani. --%>
 <jsp:param name="Zaglavlje" value="Generisanje ekranske forme na osnovu sloga" />
 <jsp:param name="KomentarOPrograma" value="JSP Primer10: Napraviti JSP stranu preko koje ce biti
                    prihvacen naziv objekta za koji ce biti generisana ekranska forma" />
<jsp:param name="NazivJSP" value="GenPrihFormu.jsp" />
<jsp:param name="ImeBaze" value="Baza" />

</jsp:forward>
```

b) GenerickoZaglavlje1.jsp prikazuje generičko zaglavlje, komentar o programu, uključuje html i jsp stranu (Unos.html i GenPrihFormu.jsp(implicitno)) i prikazuje informacije o autoru aplikacije.

Isto kao u primeru JSPPrimer8.

c) GenProhFormu.jsp preko koje se generiše ekranska forma za izabrani objekat.

```
<%-- Uključivanje paketa koji su potrebni za rad sa bazom podataka. --%>
<%@ page import="DB.*" %>
<%@ page import = "java.sql.*" %>

<%-- deklarisanje promenljivih i objekata. --%>
%>
<%!String[] st;
String st1[];
DBBrokerSp db = new DBBrokerSp();
ResultSet rs;
ResultSetMetaData rsmd;
int signal;
int brojKolona;
%>

<%-- otvaranje baze i generisanje forme na osnovu izabranog objekta. --%>
<%

signal = db.otvoriBazu(request.getParameter("ImeBaze"));
if (signal!=41 || request.getParameter("NazivObjekta") == null)
  {}
else
{  try {
   rs = db.izvrsiUpit("Select * From " + request.getParameter("NazivObjekta") + ";");
   rsmd = rs.getMetaData();
   brojKolona = rsmd.getColumnCount();%>
  <table border = 1>
    <%for (int i = 1; i<=brojKolona; i++)
      { %>
         <tr>
         <td> <i><%= rsmd.getColumnName(i) %></i>
          <td width=<%= rsmd.getColumnDisplaySize(i)%> ><input type="text"size =<%= rsmd.getColumnDisplaySize(i)%>
                    name= <%= rsmd.getColumnName(i) %>
         </tr>
  <%
  }

 rs.close();
```

```
}catch(SQLException e) {}
%>
 </table>
<%}%>
```

d) Unos.html preko koje se prihvata naziv objekta za koji treba da bude generisana ekranska forma.

```
<html>
<body>
<form action="http://127.0.0.1:8084/JSPPrimer10/Pokreni.jsp" method="POST">
<p><strong>Unesite naziv objekta:</strong></p>
<dl>
  <dd><textarea name="NazivObjekta" rows="5" cols="42"></textarea></dd>
</dl>
<p><input type="submit" value="Generisi formu za objekat"></p>
</form>
</body>
</html>
```

Zadatak ZJSPBP6: *Proširiti primer JSPPrimer10 tako da se omogući operacija pamćenja objekta (u bazi podataka) za koji je generisana ekranska forma.*

JSP Primer72: Napraviti JSP stranu koja će parametarski pozvati drugu JSP stranu i proslediti joj parametre o zaglavlju i komenatar o programu. Preko druge JSP strane ce biti napunjen kombo boks Zanimanje preko baze podataka. Ubaciti kontekst promenljivu i vezati je za database brokera kako bi se strana povezala sa bazom pri prvom pozivu. Pri svakom sledećem pozivu (dokle god je podignuta JSP strana) veza sa bazom bi bila zadržana.