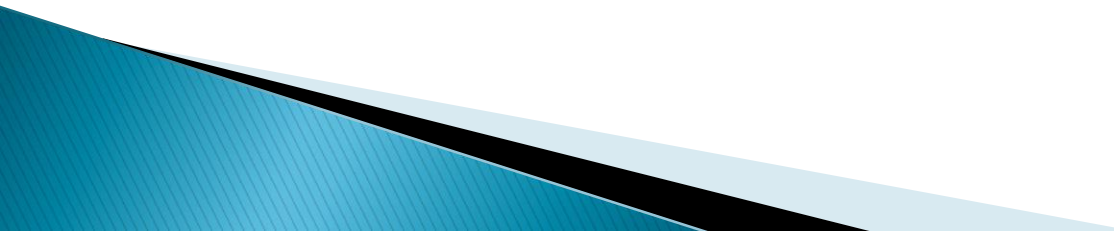


# Servleti

Internet programiranje  
IV godina, ETF Banjaluka

# Servleti

- ▶ HTTP
  - ▶ Web serveri
  - ▶ Web čitači
  - ▶ HTML
  - ▶ statički sadržaj
  - ▶ dinamički sadržaj
- 

# Servleti

## ► specifikacije

- 2.0
- 2.1
- 2.2
- 2.3
- 2.4
- 2.5
- 3.0
- 3.1
- 4.0 – 05.09.2017.g.

# Servlet

- ▶ jedna od tehnologija za generisanje dinamičkog sadržaja
- ▶ WWW server mora da ima podršku za servlete
- ▶ pisanje servleta isključivo u Javi
- ▶ za njihovo izvršavanje potrebna je i JVM (obezbjeđuje je Web server)
- ▶ servlet obavlja:
  - primanje i čitanje eksplicitnih podataka poslatih od strane klijenta (podaci sa forme)
  - primanje i čitanje implicitnih podataka poslatih od strane klijenta (header polja zahtjeva)
  - generisanje odgovora
  - slanje eksplicitnih podataka klijentu (HTML)
  - slanje implicitnih podataka klijentu (kodovi statusa i header polja odgovora)

# Servlet

- ▶ Java klasa
- ▶ paketi javax.servlet i javax.servlet.http
- ▶ implementira interfejs Servlet (javax.servlet.Servlet) – metode koje definišu životni ciklus servleta
- ▶ nasljeđuje klase GenericServlet ili HttpServlet (javax.servlet.GenericServlet, javax.servlet.http.HttpServlet )
- ▶ paketi javax.servlet i javax.servlet.http
- ▶ može da redefiniše sljedeće metode:
  - init() – izvršava se samo jednom
  - service() – izvršava se za svaki zahtjev
  - doGet()
  - doPost()
  - destroy() – poziva se kada server obriše instancu servleta
- ▶ ove metode se nikad ne pozivaju direktno, već ih poziva Web server u odgovarajućim trenucima
- ▶ rezultat izvršavanja servleta je dinamički kreiran html kod, ali to mogu biti i podaci u nekom drugom formatu

```
java.lang.Object
|
+--javax.servlet.GenericServlet
|
+--javax.servlet.http.HttpServlet
```

# Servlet

- ▶ instance servleta se kreiraju automatski, prilikom njihovog zahtijevanja
- ▶ kreira ih servlet kontejner
- ▶ nema potrebe za definisanjem konstruktora
- ▶ “entry point” nije main metoda

# Životni ciklus servleta

- ▶ kontrolisan od strane servlet kontejnera
- ▶ kad se zahtjev mapira u odgovarajući servlet, servlet kontejner radi sljedeće:
  - ako ne postoji instanca servleta:
    - učitava servlet klasu
    - kreira instancu servlet klase
    - vrši inicijalizaciju – metoda `init()` – izvršava se samo jednom kada se servlet prvi put učitava
  - poziva service metodu proslijeđujući joj *request* i *response* objekte
    - poziva se kod novog thread-a od strane servera za svaki zahtjev
    - ne treba preklapati ovu metodu
  - `doMethod`
    - obrađuju GET, POST i druge zahtjeve (DELETE, OPTIONS, PUT, HEAD, TRACE)
    - preklapanje ovih metoda omogućava željeno ponašanje servleta
  - `destroy`
    - poziva se kada server uništava instancu servleta
    - ne poziva se posle svakog zahtjeva
- ▶ upravljanje greškama:
  - izuzeci
  - podrazumijevana stranica: “*A Servlet Exception Has Occurred*”
  - definisanje error strane

# Inicijalizacija servleta

- ▶ `HttpServlet.init()` – namijenjena za inicijalizaciju servleta, prije njegove prve upotrebe
- ▶ poziva se tačno jednom
- ▶ nema prepreke za postojanje konstruktora u servlet klasi u kome će se odvijati dio inicijalizacije, ali je na raspolaganju i `init` metoda
- ▶ `init()` se koristi za čitanje konfiguracionih podataka, inicijalizaciju resursa i drugih *one-time* aktivnosti



# Servisne metode servleta

- ▶ servisi koje obezbjeđuje servlet implementirani su u:
  - `service()` metodi klase `GenericServlet`,
  - *doMethod* metodama klase `HttpServlet` – `doGet`, `doDelete`, `doOptions`, `doPost`, `doPut`, `doTrace`) klase `HttpServlet`,
  - ili bilo kojoj drugoj protokol-specifičnoj metodi definisanoj u klasi koja implementira `Servlet` interfejs
- ▶ opšti obrazac rada servisne metode:
  - uzeti informacije iz zahtjeva
  - pristupiti eksternim resursima
  - kreirati odgovor na bazi prethodnog
- ▶ važno:
  - pravilna procedura za kreiranje odgovora podrazumijeva da popunjavanje *response header*-a prethodi uzimanju izlaznog *stream*-a od *response* objekta, i nakon toga upis sadržaja u izlazni *stream*

# Uzimanje informacija iz zahtjeva

- ▶ zahtjev sadrži podatke koji se prosljeđuju od klijenta ka servletu
- ▶ svi zahtjevi implementiraju ServletRequest interfejs koji definiše metode za pristup sljedećim informacijama:
  - paramteri – koji se obično koriste za prenos informacija između klijenta i servleta
  - atributi scope objekata – koji se koriste za prenos informacija između servlet kontejnera i servleta ili između servleta koji sarađuju
  - informacije o protokolu kojim komuniciraju klijent i server
  - informacije o lokalizaciji
- ▶ primjeri HTML koda:

```
<FORM name="postForma" action="TestServlet"
method=POST>
</FORM>
```

```
<FORM name="getForma"
action="TestServlet?id=12" method=GET>
</FORM>
```

# Uzimanje informacija iz zahtjeva

- ▶ primjeri:

```
String bookId = request.getParameter("bookId");
```

- ▶ HTTP servletima se proslijeđuje HTTP request objekat – `HttpServletRequest`, koji sadrži URL, HTTP *header*-e, *query* string...

- ▶ URL HTTP zahtjeva sadrži sljedeće:

```
http://<host>:<port><request path>?<query string>
```

- ▶ *request path* se sastoji od:

- **context path** – konkatencija kose crte (/) sa *context root*-om servlet aplikacije
- **servlet path** – *path* sekcija – konkatencija kose crte (/) i putanje (alijasa) koja je aktivirala ovaj zahtjev
- **path info** – dio zahtjeva koji nije dio *context path*-a ili *servlet path*

# Uzimanje informacija iz zahtjeva

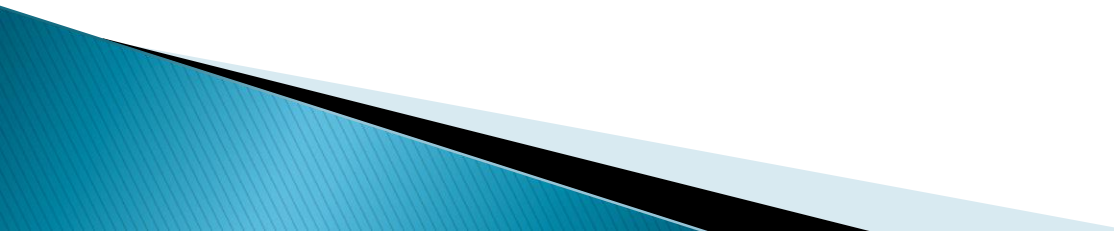
- ▶ `HttpServletRequest` objekat omogućava:
  - pristup HTTP header podacima
  - prihvatanje argumenata koje klijent šalje kao dio svog zahtjeva
- ▶ metode `ServletRequest` klase:
  - `getParameter` metoda – vraća vrijednost imenovanog parametra – za pristup podacima koje šalje klijent
  - `getParameterValues` – ako parametar ima više od jedne vrijednosti – vraća niz vrijednosti za imenovani parametar
  - `getParameterNames` – vraća imena svih parametara
  - `getParameterMap()` – vraća `java.util.Map` objekat sa svim parametrima
  - ...

# Konstrukcija odgovora

- ▶ odgovor sadrži podatke koji se prosleđuju od servleta ka klijentu
- ▶ svi odgovori implementiraju `ServletResponse` interfejs koji definiše metode za:
  - dobijanje izlaznog stream-a putem kojeg se šalju podaci klijentu:
    - za slanje karakter podataka koristi se `PrintWriter` kojeg vraća `getWriter()` metoda
    - za slanje binarnih podataka koristi se `ServletOutputStream` kojeg vraća `getOutputStream()` metoda
    - poziv `close()` metode ovih objekata, nakon slanja odgovora, omogućava serveru da zna da je odgovor kompletiran
  - definisanje content type-a (npr., `text/html`) podataka koji se vraćaju – registar content tipova održava Internet Assigned Numbers Authority (IANA): <ftp://ftp.isi.edu/in-notes/iana/assignments/media-types>
  - utvrditi da li se odgovor buffer-uje – po default-u, sav sadržaj koji se šalje u izlazni stream se trenutno šalje klijentu
  - postavljanje informacija o lokalizaciji
- ▶ HTTP response objekt (`HttpServletResponse`) ima polja koja predstavljaju HTTP header-e, npr.:
  - statusni kodovi
  - cookies – čuvaju specifične informacije o aplikaciji na strani klijenta – mogu se koristiti i za praćenje sesije korisnika

# Primjer servleta

```
public class HelloServlet extends HttpServlet {  
    public void doGet(HttpServletRequest request,  
        HttpServletResponse response)  
        throws ServletException, IOException {  
        response.setContentType("text/html");  
        PrintWriter out = response.getWriter();  
        out.println("<HTML>\n" +  
            "<HEAD><TITLE>Naslov</TITLE></HEAD>\n"+  
            "<BODY BGCOLOR=\"blue\">\n" +  
            "<H1>Tekst</H1>\n" +  
            "</BODY></HTML>");  
        out.close();  
    }  
}
```



# HttpServlet.doGet()

- ▶ namijenjena za obradu GET zahtjeva
- ▶ poziva se za svaki GET zahtjev klijenta koji je tražio datoteku za čije generisanje je zadužen dati servlet

```
public void doGet(HttpServletRequest req,
    HttpServletResponse res) {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Test</TITLE></HEAD>");
    out.println("<BODY>");
    ...
    out.println("</BODY>");
    out.close();
}
```

# HttpServlet.doPost()

- ▶ namijenjena za obradu POST zahtjeva
- ▶ poziva se za svaki POST zahtjev klijenta

```
public void doPost(HttpServletRequest req,
    HttpServletResponse res) {
    String parameter1 = req.getParameter("p1");
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
    out.println("<HEAD><TITLE>Test</TITLE></HEAD>");
    out.println("<BODY>");
    ...
    out.println(parameter1);
    out.println("</BODY>");
    out.close();
}
```

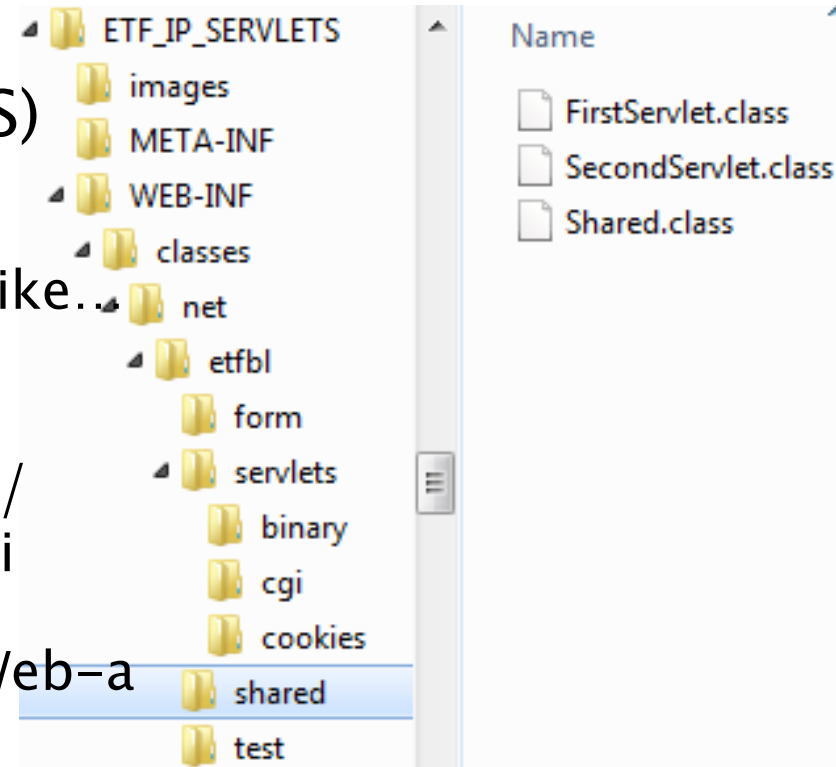


# HttpServlet.destroy()

- ▶ poziva se prilikom uništavanja servleta
- ▶ koristi se za *clean-up* zadatke neposredno prije uništenja servleta – oslobađanje resursa koje je servlet zauzimao:
  - otvorene datoteke, konekcija sa bazom podataka
- ▶ obično prilikom *undeploy-a* aplikacije

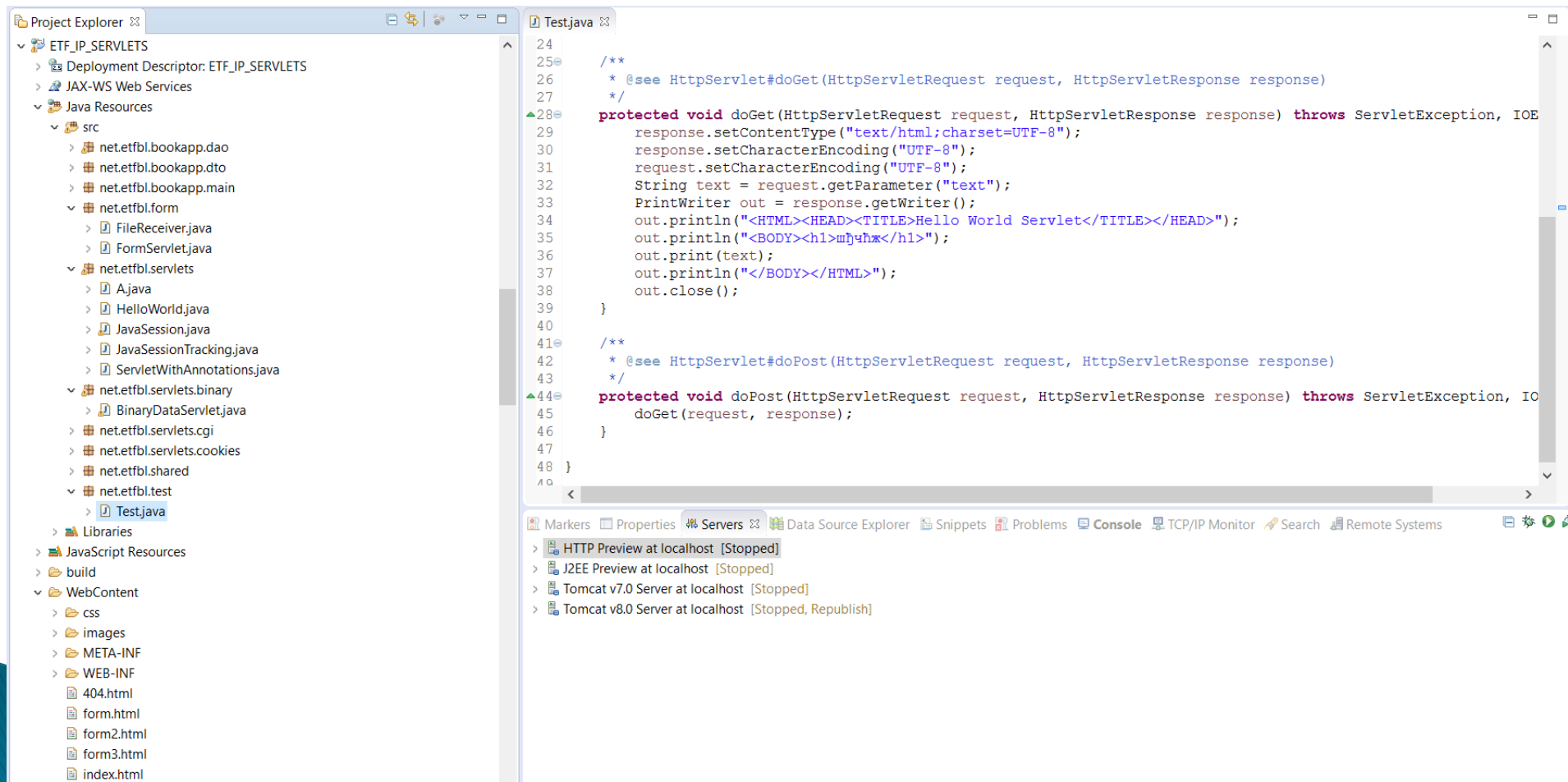
# Servlet deployment

- ▶ root folder (ETF\_IP\_SERVLETS)
  - “polazna tačka” Web aplikacije
  - sve datoteke i poddirektorijumi su unutar ovog foldera: html, slike...
- ▶ / ETF\_IP\_SERVLETS/WEB-INF/
  - sadrži konfiguracione datoteke i kompajlirane klase
  - nije direktno dostupan putem Web-a
- ▶ / ETF\_IP\_SERVLETS/WEB-INF/classes/
  - sve kompajlirane klase (servlet klase i druge klase) se nalaze u ovom folderu



# Servlet deployment

## ► Projektat



The screenshot displays an IDE interface with two main panels. The left panel, titled 'Project Explorer', shows a project named 'ETF\_IP\_SERVLETS'. The project structure includes a 'Deployment Descriptor: ETF\_IP\_SERVLETS', 'JAX-WS Web Services', and 'Java Resources'. Under 'Java Resources', there is a 'src' directory containing several packages and files, including 'net.etfbl.bookapp', 'net.etfbl.form', 'net.etfbl.servlets', and 'net.etfbl.test'. The 'Test.java' file is selected. The right panel shows the code for 'Test.java', which implements the 'HttpServlet' interface. It includes annotations for 'doGet' and 'doPost' methods, and the implementation logic for both. The 'doGet' method sets the content type to 'text/html; charset=UTF-8', sets character encoding to 'UTF-8', and prints a 'Hello World Servlet' message. The 'doPost' method calls 'doGet' and returns the response. The bottom panel shows the 'Servers' tab, listing several servers: 'HTTP Preview at localhost [Stopped]', 'J2EE Preview at localhost [Stopped]', 'Tomcat v7.0 Server at localhost [Stopped]', and 'Tomcat v8.0 Server at localhost [Stopped, Republish]'. The 'Tomcat v8.0 Server at localhost [Stopped, Republish]' server is highlighted.

**Project Explorer:**

- ETF\_IP\_SERVLETS
  - Deployment Descriptor: ETF\_IP\_SERVLETS
  - JAX-WS Web Services
  - Java Resources
    - src
      - net.etfbl.bookapp.dao
      - net.etfbl.bookapp.dto
      - net.etfbl.bookapp.main
      - net.etfbl.form
        - FileReceiver.java
        - FormServlet.java
      - net.etfbl.servlets
        - A.java
        - HelloWorld.java
        - JavaSession.java
        - JavaSessionTracking.java
        - ServletWithAnnotations.java
      - net.etfbl.servlets.binary
        - BinaryDataServlet.java
      - net.etfbl.servlets.cgi
      - net.etfbl.servlets.cookies
      - net.etfbl.shared
      - net.etfbl.test
        - Test.java
  - Libraries
  - JavaScript Resources
  - build
  - WebContent
    - css
    - images
    - META-INF
    - WEB-INF
      - 404.html
      - form.html
      - form2.html
      - form3.html
      - index.html

# Mapiranje servleta

- ▶ servlet klasa mora biti mapirana – URI
- ▶ pristup servletu – general pattern
  - `http://[domain:port]/[context]/servlet/[ServletClassName]`
  - `http://localhost:8080/servletintro/servlet/SimpleServlet`
- ▶ mapiranje korištenjem konfiguracione datoteke *web.xml*
  - Servlet se mapira u URL koji je definisan u *web.xml* datoteci
- ▶ mapiranje korištenjem anotacija
  - `@WebServlet("/swa")`

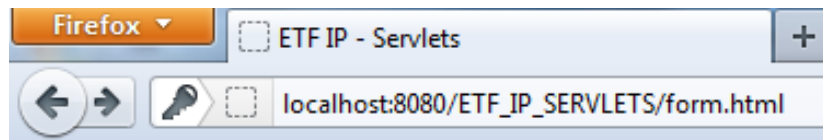
# web.xml

- ▶ web.xml se nalazi u “WEB-INF” folderu
- ▶ **Primjer**
  - Servlet klasa
    - HelloWorld.class
  - Kontekst aplikacije:
    - <http://localhost:8080/servletintro/>
  - Invoker class mapiranje
    - <http://localhost:8080/servletintro/servlet/HelloWorld>
  - Mapiranje putem web.xml datoteke
    - <http://localhost:8080/servletintro/hello>

```
<servlet>
    <servlet-name>HelloW</servlet-name>
    <servlet-class>HelloWorld</servlet-class>
</servlet>
```

```
<servlet-mapping>
    <servlet-name>HelloW</servlet-name>
    <url-pattern>hello</url-pattern>
</servlet-mapping>
```

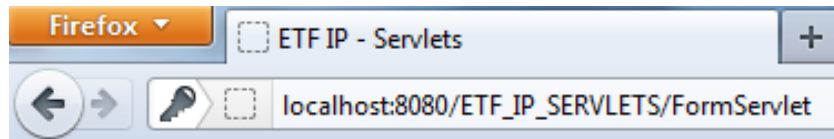
# Preuzimanje podataka sa formi



## Form

username:

password:



**Hello marko!**

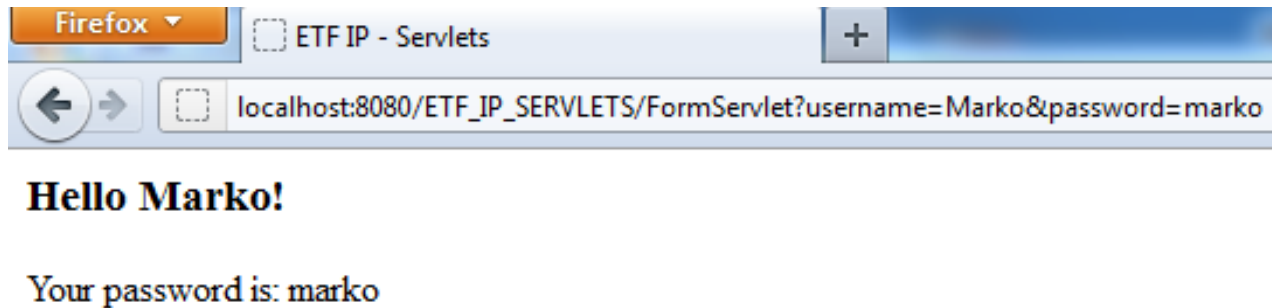
Your password is: markovalozinka@1234

# Preuzimanje podataka sa formi

- ▶ `request.getParameter("name")`
  - dobija se URL-dekodirana vrijednost prvog elementa koji se zove *name*
  - ponaša se isto i za GET i za POST zahtjeve
  - rezultat je null ako ne postoji takav parametar u elementima forme
- ▶ `request.getParameterValues("name")`
  - dobija se niz URL-dekodiranih vrijednosti za sve elemente koji se zovu *name*
  - dobija se niz sa jednim elementom, ako se ime pojavljuje samo jednom
  - rezultat je null ako ne postoji takav parametar u elementima forme
- ▶ `request.getParameterNames()` i `request.getParameterMap()`
  - dobija se Enumeration ili Map objekti od poslatih elemenata

# GET i POST zahtjevi

GET /ETF\_IP\_SERVLETS/FormServlet?username=Marko&password=marko HTTP/1.1

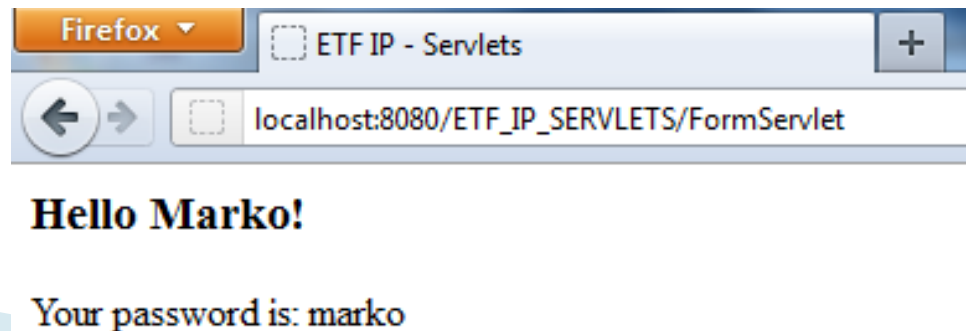


POST /ETF\_IP\_SERVLETS/FormServlet HTTP/1.1

...

username=Marko&password=marko

...





# Dijeljenje informacija

- ▶ Web komponente dijele informacije putem objekata koji su atributi 4 *scope* objekta – ovim atributima se pristupa pomoću [get|set]Attribute metoda klase koja predstavlja odgovarajući opseg (*scope*)
- ▶ *scope* objekti i odgovarajuće klase:
  - Web context – javax.servlet.ServletContext
  - session – javax.servlet.http.HttpSession
  - request – podtip javax.servlet.HttpServletRequest
  - page – javax.servlet.jsp.PageContext

# Dijeljenje informacija

- ▶ imenovani objekti se mogu dijeliti između svih servleta u Servlet context-u (i drugim kontekstima) vezivanjem objekata za ServletContext objekat koji se dijeli između više servleta
- ▶ ServletContext klasa ima nekoliko metoda za pristup dijeljenim objektima:
  - `public void setAttribute(String name, Object object)` – dodaje novi ili zamjenjuje stari objekat sa datim imenom. ime atributa bi trebalo da poštuje konvenciju davanja imena Java paketima.
  - `public Object getAttribute(String name)` – vraća imenovani objekat ili null ako atribut ne postoji
  - `public Enumeration getAttributeNames()` – vraća enumeraciju imena svih dostupnih atributa
  - `public void removeAttribute(String name)` – uklanja atribut sa specificiranim imenom, ako takav postoji

# Cookies

- ▶ Ideja
  - servlet generiše jedinstveno ime i vrijednost klijentu
  - klijent vraća isto ime i vrijednost kada se ponovo konektuje na isti sajt ( ili isti domen u zavisnosti od podešavanja vrijednosti cookie-ija)
- ▶ tipična upotreba cookie-ija
  - identifikacija korisnika tokom e-commerce sesije
  - izbjegavanje korisničkog imena i lozinke
  - prilagođavanje sajta korisniku
  - Ciljane reklame
- ▶ RFC 6265

# Cookies

## ▶ problemi:

- serveri mogu da pamte korisnikove ranije akcije
- ako korisnik daje personalne informacije, serveri mogu da povezuju te informacije sa prethodnim akcijama
- serveri mogu dijeliti cookie informacije sa trećom stranom
- loše implementirane aplikacije smještaju povjerljive informacije, kao što su brojevi kreditnih kartica, direktno u cookie

## ▶ preporuke:

- ako cookie-iji nisu kritični za izvršavanje zadatka, treba izbjegavati servlete koji u potpunosti prestaju sa radom ako se cookie-iji zabrane
- ne smještati povjerljive informacije u cookie-ije

# Cookies

## ▶ slanje cookie-a:

- kreira se Cookie objekat – poziva se Cookie konstruktor sa imenom cookie-ija i željenom vrijednosti, oba argumenta su tipa String

```
Cookie c = new Cookie("studentID", "123");
```

- postavlja se maksimalno vreme života cookie-ija

```
c.setMaxAge(60*60*24); // Jedan dan
```

- Cookie se smješta u objekat HTTP response – koristi se response.addCookie() – bez nje cookie neće biti poslat klijentu

```
response.addCookie(c);
```

# Cookies

- ▶ čitanje cookie-a:
  - metoda `request.getCookies()` – vraća niz objekata tipa `Cookie`
  - prolazak kroz elemente niza i pozivanje metode `getName()` za svaki element niza sve dok se ne pronađe željeni cookie
  - čitanje vrijednost upisane u `Cookie` – pomoću `getValue()` metode

# Cookies

- ▶ **getDomain/setDomain**
  - specificira se domen koji obrađuje cookie – trenutni host mora biti dio specificiranog domena
- ▶ **getMaxAge/setMaxAge**
  - čita/postavlja vrijeme života cookie-ija (u sekundama) – ako se ova vrijednost ne postavi, podrazumijeva se da je vrijeme života cookie-ija samo trenutna sesija.
- ▶ **getName**
  - dobija se ime cookie-ija – ne postoji setName metoda, jer se ime cookie-ja definiše u okviru konstruktora

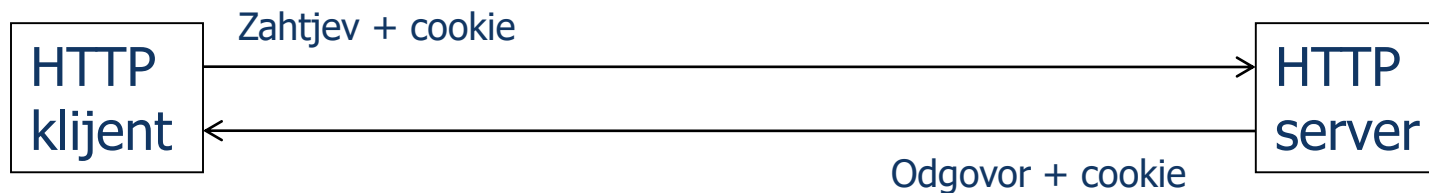
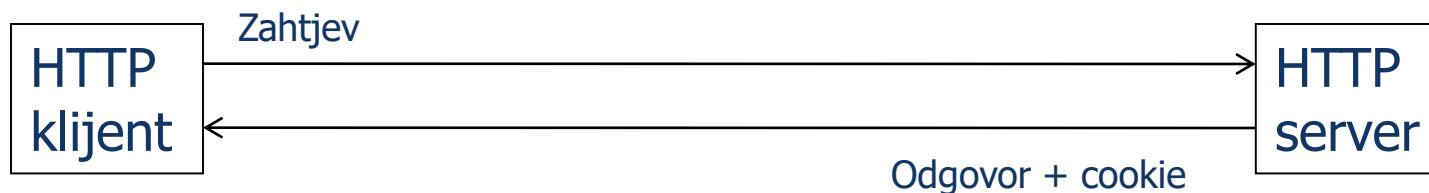
# Cookies

- ▶ **getPath/setPath**
  - čita/postavlja putanju koja obrađuje cookie – ako se ne navede, cookie pripada URL-u koji je u okviru ili direktorijum iznad trenutne stranice
- ▶ **getSecure/setSecure**
  - čita/postavlja flag koji definiše da li se cookie izvršava samo putem SSL konekcije
- ▶ **getValue/setValue**
  - čita/postavlja vrijednost koja se želi pamtiti u cookie-u – za nove cookie-ije, ova vrijednost se postavlja u konstruktoru, a ne pomoću setValue – getValue se koristi da bi se dobila upisana vrijednost. Ako se postojećem cookie-iju mijenja vrijednost, potrebno je tu novu vrijednost poslati pomoću metode `response.addCookie`.



# Praćenje sesije korisnika

- ▶ cookie mehanizam – povezivanje cookie-a sa podacima na serveru



# Praćenje sesije korisnika

- ▶ URL rewriting
- ▶ na klijentskoj strani dodaje se novi sadržaj koji identifikuje datu sesiju na kraj svakog URLa
- ▶ server povezuje poslati podatak sa sesijom koja se izvršava

`http://localhost:8080/path/file.jsp;jsessionid=1234`

- ▶ prednosti:
  - korektno se izvršava i u slučajevima kada su cookie-iji zabranjeni ili nisu podržani
- ▶ nedostaci:
  - moraju se mijenjati svi URL-ovi koji se pozivaju sa date stranice
  - sve stranice se moraju dinamički generisati
  - ne radi korektno za linkove sa drugih sajtova

# Praćenje sesije korisnika

## ▶ hidden polja u formama

```
<INPUT TYPE="HIDDEN" NAME="sessionid"  
VALUE="...">
```

## ▶ prednosti:

- korektno se izvršava i u slučajevima kada su cookie-iji zabranjeni ili nisu podržani

## ▶ nedostaci:

- dosta napornog procesiranja
- sve stranice moraju biti rezultat slanja forme

# Praćenje sesije korisnika

- ▶ praćenje sesije korisnika pomoću Java
- ▶ objekti sesije se čuvaju na serveru
- ▶ sesije su automatski povezane sa klijentom pomoću cookie-ija ili promjene URL-a
  - koristi se `request.getSession` da bi se dobio objekat sesije
  - u pozadini, sistem pregleda cookie ili URL dodatne informacije i pretražuje da li se ključ poklapa sa nekim od prethodno smještenih objekata sesije
  - ako pronađe poklapanje, kao rezultat vraća pronađeni objekat
  - ako nema poklapanja, kreira novi objekat sesije, povezuje cookie ili URL informacije sa njegovim ključem i kao rezultat vraća novi objekat sesije
- ▶ mehanizmi poput `Hashtable` dozvoljavaju da se unutar sesije smještaju neophodni objekti
  - `setAttribute` smješta vrijednost
  - `getAttribute` vraća vrijednost

# Praćenje sesije korisnika – Java

- ▶ pristup objektu sesije
  - pozivom `request.getSession` dobija se `HttpSession` objekat
  - to je objekat tipa `Hashtable` povezan sa korisnikom
- ▶ pretraga informacija povezanih sa sesijom
  - poziv `getAttribute` u okviru `HttpSession` objekta, prebacuje vrijednost u odgovarajući tip i provjerava da li je rezultat `null`
- ▶ smještanje informacije u okviru sesije.
  - koristi se `setAttribute` sa parom ključ, vrijednost kao argumentom
- ▶ uništavanje podataka sesije
  - poziva se `removeAttribute` da bi se uništila specifična vrijednost
  - poziva se `invalidate` da bi se uništila trenutna sesija

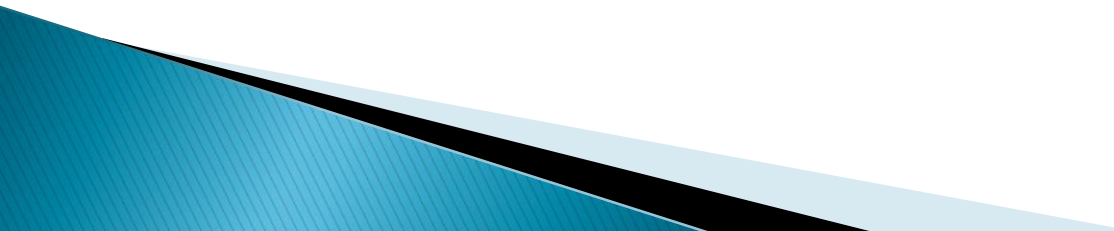
# HttpSession metode

- ▶ **getAttribute**
  - vraća prethodno smještenu vrijednost iz objekta sesije
  - rezultat je null ako nijedna vrijednost nije povezana sa datim imenom
- ▶ **setAttribute**
  - povezuje vrijednost sa navedenim imenom
  - vrši se monitoring promjena: vrijednosti implementiraju HttpSessionBindingListener
- ▶ **removeAttribute**
  - uklanja vrijednosti koje su povezane sa navedenim imenom
- ▶ **getAttributeNames**
  - vraća imena svih atributa u okviru sesije
- ▶ **getId**
  - vraća jedinstveni identifikator

# HttpSession metode

- ▶ **isNew**
  - vraća true ako klijent ne zna za sesiju, ili ako ne može da pristupi sesiji. Npr., ako server radi samo sa cookie-baziranim sesijama, i ako je korisnik disable-ovao korištenje cookie-ja, onda će sesija biti nova pri svakom zahtjevu
- ▶ **getCreationTime**
  - vraća vrijeme kada je sesija prvi put kreirana
- ▶ **getLastAccessedTime**
  - vraća kao rezultat vrijeme kada je sesija posljednji put poslata klijentu
- ▶ **getMaxInactiveInterval, setMaxInactiveInterval**
  - pročitati ili postaviti vremenski interval kada je sesije bez pristupa i postavlja se kao nevalidna
- ▶ **invalidate**
  - postaviti trenutnu sesiju kao nevalidnu

# Primjer: elementarni servlet

- ▶ tipična sekvenca poziva metoda servleta:
    - inicijalizacija
    - doGet
    - destroy
- 

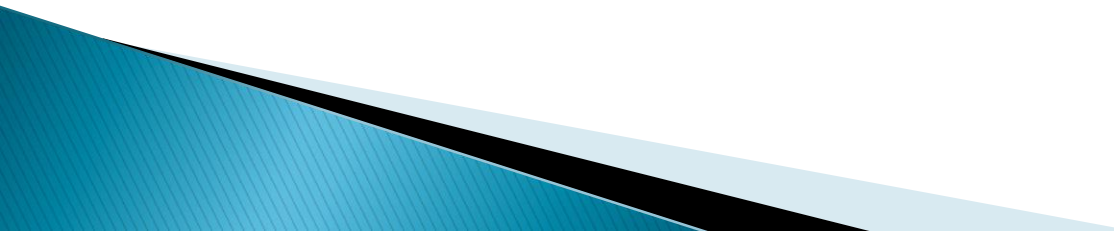


# Primjer: elementarni servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SimpleCounter extends HttpServlet {

    int count = 0;

    public void doGet(HttpServletRequest req, HttpServletResponse
res) throws ServletException, IOException {
        res.setContentType("text/plain");
        PrintWriter out = res.getWriter();
        count++;
        out.println("Ovom servletu je pristupljeno " + count + "
puta.");
    }
}
```



# Servleti i JDBC

- ▶ **init()**: otvaranje konekcije sa bazom
- ▶ **doGet()**: postavljanje upita, formiranje rezultata
- ▶ **destroy()**: zatvaranje konekcije sa bazom

# Konkurentni pristup servletu

- ▶ za svaku servlet klasu instancira se tačno jedan objekat koji opslužuje sve klijente
- ▶ njegove **doGet()** i **doPost()** metode mogu biti istovremeno pozvane iz više programskih niti Web servera
- ▶ konkurentni pristup se može pojaviti u sljedećim situacijama:
  - kada višestruke Web komponente pristupaju objektima smještenim u Web context
  - kada višestruke Web komponente pristupaju objektima smještenim u sesiji
  - višestruke niti unutar Web komponente pristupaju varijablama instanci
- ▶ pristup SUBP – Web komponente koriste JDBC 2.0 API za pristup relacionim bazama podataka

# Sinhronizacija

- ▶ varijabla `count` je dijeljena od strane više niti

```
count++ // Nit 1
```

```
count++ // Nit 2
```

```
out.println // Nit 1
```

```
out.println // Nit 2
```

- ▶ dvije niti
- ▶ nekonzistentnost

# Sinhronizacija

## ► rješenje

```
PrintWriter out = res.getWriter();  
synchronized(this) {  
    count++;  
    out.println("Ovom servletu je pristupljeno " + count  
+ " puta.");  
}
```