



CS230 - DISTRIBUIRANI SISTEMI

Java API za JSON procesiranje

Lekcija 11

PRIRUČNIK ZA STUDENTE

# CS230 - DISTRIBUIRANI SISTEMI

## Lekcija 11

### *JAVA API ZA JSON PROCESIRANJE*

- ✓ Java API za JSON procesiranje
- ✓ Poglavlje 1: JavaScript Object Notation - JSON
- ✓ Poglavlje 2: JSON-P API objektnog modela
- ✓ Poglavlje 3: JSON-P API tokova
- ✓ Poglavlje 4: Java API za JSON procesiranje - pokazni primer 1
- ✓ Poglavlje 5: Java API za JSON procesiranje - pokazni primer 2
- ✓ Poglavlje 6: Individualna vežba 11
- ✓ Poglavlje 7: Domaći zadatak 11
- ✓ Zaključak

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

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

## ▼ Uvod

### UVOD

*Lekcija će se baviti mehanizmima JSON procesiranja u distribuiranim Java EE aplikacijama.*

U narednoj lekciji sledi detaljna analiza i diskusija u vezi sa mehanizmima JSON procesiranja u distribuiranim Java EE veb aplikacijama. Studentima je verovatno poznat *JSON* pojam međutim, za one koji nisu upućeni u njega sledi nekoliko uvodnih rečenica. *JSON* (skraćenica za *JavaScript Object Notation*) predstavlja lak (*lightweight*) format razmene poruka. Vremenom ovaj format je stekao veliku popularnost i široku primenu. Posebno je istaći njegove prednosti u odnosu na *XML* baziranu razmenu podataka koja se ogleda u sledećem:

- *JSON je lakši za upotrebu programerima jer se lakše čita i piše;*
- *JSON je lakši za upotrebu računarima jer se lakše generiše i parsira.*

Posebno je važno napomenuti da je JSON vremenom stekao široku primenu u brojnom savremenim veb aplikacijama.

*Java EE 7* uvodi *Java API* za *JSON procesiranje (JSON-P)* koji je postao standardan *Java EE API* za generisanje i parsiranje *JSON* podataka. *JSON-P* obezbeđuje dva načina za parsiranje i generisanje *JSON* podataka. To su: API objektnog modela (*the object model API*) i API tokova (*the streaming API*).

Na osnovu navedenog lekcija će se posebno baviti sledećim temama:

- *JSON-P API objektnog modela* - Poseban akcenat će biti na analizi, diskusiji i demonstraciji generisanja i parsiranja JSON podataka pomoću JSON-P API objektnog modela;
- *JSON-P API tokova* - Poseban akcenat će biti na analizi, diskusiji i demonstraciji generisanja i parsiranja JSON podataka pomoću JSON-P API modela tokova;

Savladavanjem ove lekcije, student će u potpunosti moći da razume i koristi *Java EE* mehanizme za *JSON* procesiranje. Takođe, student će ovladati dodatnim naprednim alatom za razvoj distribuiranih veb sistema.

## ▼ Poglavlje 1

# JavaScript Object Notation - JSON

## OPŠTE NAPOMENE

*JSON je jedan od lakših tekstualnih otvorenih standarda dizajniran za čitljivu razmenu podataka*

**JSON** (JavaScript Object Notation) je jedan od lakših tekstualnih otvorenih standarda dizajniran za čitljivu razmenu podataka. Ekstenzija datoteke s podacima u **JSON**-ovom formatu je **.json**, dok je meta oznaka (**MIME** format) **application/json**.

JSON je format koji u velikoj meri zamenjuje XML (o tome će biti puno govora i u nastavku) jer ima nekolicinu prednosti u odnosu na **XML**. **JSON** ne koristi tagove pa stoga ima kraći kod koji je lakši za pisanje i razumljiviji za čitanje. Mada je najbitnija prednost JSON-a u odnosu na XML je ta što se **JSON** parsira kroz standardnu API funkciju dok se XML parsira kroz XML parser.

**JSON ne zavisi od programskog jezika i kao takav je idealna notacija za razmenu podataka.**

Detaljnije o JSON notaciji:

1. <https://www.json.org/json-sr.html>
2. [https://www.w3schools.com/js/js\\_json\\_intro.asp](https://www.w3schools.com/js/js_json_intro.asp)
3. <https://www.tutorialspoint.com/json/index.htm>

Video materijal: **JavaScript Object Notation (JSON) Format** - trajanje - 5:46.

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

## JSON - TIPOVI I STRUKTURA PODATAKA

*U okviru JSON-ove strukture mogu da se koriste tipovi podataka: broj, string, niz, boolean i objekat.*

U okviru **JSON**-ove strukture mogu da se koriste sledeći tipovi podataka:

1. **Broj** (JavaScript format u pokretnom zarezu sa dvostrukom preciznošću, zavisi od implementacije)
2. **String** (Unicode format, sa dvostrukim navodnicima, kao izlazna sekvenca se koristi backslash )
3. **Boolean** (true ili false)

4. *Niz* (uredjena sekvenca vrednosti, odvojena zarezima i uokvirena kockastim zagradama; vrednosti ne moraju biti istog tipa)
5. *Objekat* (neuredjena kolekcija ključ:vrednost parova sa ':' karakterom koji razdvaja ključ i vrednost, razdvojeni zarezima i uokvireni vitičastim zagradama. Ključevi moraju biti niske i različiti od ostalih ključeva)
6. *null* (prazno)

Beline se mogu slobodno dodati izmedju strukturalnih karaktera (zagrada "{ } [ ]", dve tačke ":", i zarez " , ").

Struktura može biti organizovana u vidu:

- *Zbirke parova (ime / vrednost)* - poznato još i kao *mapa*. - Na raznim jezicima, to je realizovano kao objekat, zapis, struktura, rečnik, heš tabela, lista sa ključevima ili asocijativni niz. Ova struktura se obeležava sa { } vitičastim zagradama, a podaci u obliku ime/vrednost su odvojeni zarezom. Ime je niz slova pod dvostrukim navodnicima, a vrednost veličina je dodeljena imenu.
- *Niza (uređena lista vrednosti)* - Niz se obeležava [ ] uglastim zagradama, a članovi niza u odvojeni zarezom.

Na sledećem primeru je *JSON*-ova reprezentacija objekta koji opisuje osobu. Objekat ima polja ime i prezime predstavljena pomoću string-a, broj za godinu, objekat koji prikazuje adresu te osobe i niz objekata sa brojevima telefona.

```
{
  "firstName": "Vladimir",
  "lastName": "Milicevic",
  "age": 45,
  "address": {
    "streetAddress": "Tadeusa Kocscuska 63",
    "city": "Beograd",
    "state": "SR",
    "postalCode": 11000
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "555 555-1234"
    },
    {
      "type": "fax",
      "number": "646 555-4567"
    }
  ]
}
```

## ▼ Poglavlje 2

# JSON-P API objektnog modela

## UVOD U JSON-P API OBJEKTNOG MODELA

*Ovaj deo lekcije se bavi generisanjem i parsiranjem JSON podataka pomoću JSON-P API objektnog modela*

Ovaj deo lekcije se fokusira na izučavanje problematike koja je u direktnoj vezi sa JSON-P API objektnog modela (JSON-P object model API).

Za *JSON-P API* objektnog modela, ovaj deo lekcije će analizu, diskusiju i demonstraciju podeliti u dve grane:

- analiza, diskusija i demonstracija generisanja *JSON* podataka pomoću *JSON-P API objektnog modela*;
- analiza, diskusija i demonstracija parsiranja *JSON* podataka pomoću *JSON-P API objektnog modela*;

Pre nego što se lekcija upusti u detaljno izlaganje navedene problematike, važno je napomenuti da *JSON-P API objektnog modela* dozvoljava programerima da generišu reprezentaciju JSON objekata u formi strukturiranog stabla. *JSON-P API objektnog modela* koristi šablon (*builder pattern*) koji omogućava programerima Java EE aplikacije da lak i elegantan način kreiraju JSON reprezentaciju Java objekata.

Posebno bi trebalo napomenuti da će izlaganje, koje prati ovaj deo lekcije biti praćeno odgovarajućim praktičnim primerima, veoma pažljivo biranim i testiranim, kako bi teorijska razmatranja bila razumljivija.

## ▼ 2.1 Generisanje JSON podataka

### KREIRANJE ZRNA JSON-P API OBJEKTNOG MODELA

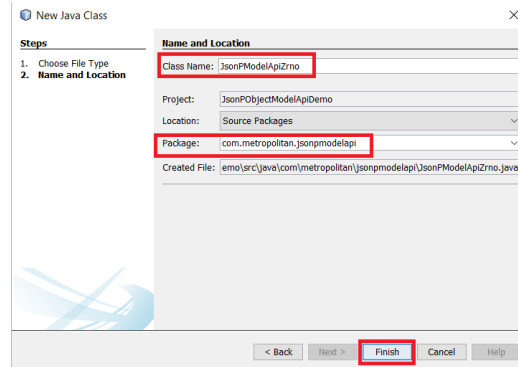
*U nastavku izlaganja tema će biti zrno JSON-P API objektnog modela.*

U nastavku izlaganja tema će biti zrno *JSON-P API objektnog modela*. Kada se koristi JSON-P API objektnog modela, početak primene je obično vezan za poziv metode **add()** u implementaciji interfejsa **JsonObjectBuilder**. Navedena metoda vraća instancu druge implementacije **JsonObjectBuilder** interfejsa. Ovde je moguće izvršiti ulančavanje

*JsonObject.add()* poziva i na taj način lako i elegantno kreirati *JSON* reprezentaciju iz Java objekta.

Da bi primena *JSON-P API objektnog modela* bila kvalitetnije analizirana i diskutovana biće kreiran odgovarajući primer u formi Java EE veb projekta pod nazivom *JsonPObjectModelApiDemo*.

Za sam početak, u ovom projektu biće kreirana klasa koja odgovara zrnju *JSON-P API objektnog modela*.



Slika 2.1.1 Kreiranje klase zrna JSON-P API objektnog modela. [izvor: autor]

Kod klase pod nazivom *JsonPModelApiZrno*, iz paketa *com.metropolitan.jsonpmodelapi*, priložen je u potpunosti sledećim listingom.

```
package com.metropolitan.jsonpmodelapi;

import java.io.StringReader;
import java.io.StringWriter;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;
import javax.json.Json;
import javax.json.JsonObject;
import javax.json.JsonObjectBuilder;
import javax.json.JsonReader;
import javax.json.JsonWriter;
/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JsonPModelApiZrno {
    @Inject
    private Osoba osoba;
    private String jsonStr;

    public String generateJson() {
        JsonObjectBuilder jsonObjectBuilder = Json.createObjectBuilder();

        JsonObject jsonObject = jsonObjectBuilder.
```

```

        add("ime", osoba.getIme()).
        add("prezime", osoba.getPrezime()).
        add("pol", osoba.getPol()).
        add("starost", osoba.getStarost()).
        build();

        StringWriter stringWriter = new StringWriter();

        try (JsonWriter jsonWriter = Json.createWriter(stringWriter)) {
            jsonWriter.writeObject(jsonObject);
        }
        setJsonStr(stringWriter.toString());

        return "prikazi_popunjen_obj";
    }

    public String parseJson() {
        JsonObject jsonObject;

        try (JsonReader jsonReader = Json.createReader(new StringReader(jsonStr))) {
            jsonObject = jsonReader.readObject();
        }

        osoba.setIme(jsonObject.getString("ime"));
        osoba.setPrezime(jsonObject.getString("prezime"));
        osoba.setPol(jsonObject.getString("pol"));
        osoba.setStarost(jsonObject.getInt("starost"));

        return "generisan_json";
    }

    public String getJsonStr() {
        return jsonStr;
    }

    public void setJsonStr(String jsonStr) {
        this.jsonStr = jsonStr;
    }
}

```

## ANALIZA KODA ZRNA JSON-P API OBJEKTNOG MODELA

*U sledećem izlaganju će biti obavljena detaljna analiza priloženog listinga.*

Sada je neophodno fokusirati se na priloženi kosa zrna *JsonPModelApiZrno.java* i njegovu metodu **generateJson()**. U navedenoj datoteci je realizovana *JSON* reprezentacija jednostavne klase *Osoba.java* koja sadrži jednostavne osobine *ime*, *prezime*, *pol* i *starost*, zajedno sa odgovarajućim **setter** i **getter** metodama.



Prvo što je urađeno u primeru jeste obezbeđivanje instance klase koja implementira interfejs *JsonObjectBuilder* pozivom statičke metode *createObjectBuilder()* klase *Json*. Ova metoda vraća instancu klase koja implementira interfejs *JsonObjectBuilder*, a to može predstavljati početnu tačku za generisanje *JSON* reprezentacije Java objekta.

Kada je obezbeđena instanca koja implementira interfejs *JsonObjectBuilder* neophodno je izvršiti pozivanje redefinisanih metoda *add()* od kojih svaka prihvata dva argumenta: string i vrednost, tim redom. Kao što je moguće videti u priloženom kodu metoda vraća drugu instancu za *JsonObjectBuilder*. U primeru je pokazano kako se vrši ulančavanje poziva metoda *add()* i na taj način je brzo i lako generisana tražena *JSON reprezentacija*. Ovde je na delu pokazan šablona za generisanje JSON reprezenatcije.

U primeru je moguće primetiti da su korišćene dve verzije metode *JsonObjectBuilder.add()*. Prva prihvata *String* kao drugi parametar, a druga za ovaj parametar prihvata vrednost tipa *Integer*. Ovde je prosleđen Integer objekat metodi.

Java raspakivanje (unboxing) vodi računa o konverziji ovog parametra u primitivni int tip.

Postoji podrška za nekoliko različitih načina redefinisanja (*overloading*) metode *JsonObjectBuilder.add()*. Na ovaj način je omogućen visok stepen fleksibilnosti građena *JSON* reprezentacija Java objekata pomoću *JSON-P API objektnog modela*. U narednom izlaganju će biti navedeni i opisani načini redefinisanja metode *JsonObjectBuilder.add()*. U svim slučajevima prvi parametar odgovara nazivu *JSON* osobine za generisani *JSON objekat*, dok drugi parametar predstavlja odgovarajuću vrednost u generisanom *JSON*.

O načinima redefinisanja ove metode biće detaljno govora u izlaganju koje sledi neposredno posle ovoga.

## RAZLIČITI OBLICI METODE ADD()

*Postoji podrška za nekoliko različitih načina potpisivanja (overloading) metode add().*

Prethodno izlaganje je napravilo odličan uvod u ovaj deo lekcije, Kao što je posebno napomenuto Java EE platforma obezbeđuje značajnu podršku u formi nekoliko različitih načina potpisivanja (*overloading*) metode *JsonObjectBuilder.add()*. Na ovaj način je omogućen visok stepen fleksibilnosti građena *JSON* reprezentacija Java objekata pomoću *JSON-P API objektnog modela*. Upravo u ovom delu lekcije će biti navedeni i detaljno opisani načini redefinisanja metode *JsonObjectBuilder.add()*.

U svim slučajevima prvi parametar metode *JsonObjectBuilder.add()* odgovara nazivu *JSON* osobine za generisani *JSON objekat*, dok drugi parametar predstavlja odgovarajuću vrednost u generisanom *JSON*.

U narednom izlaganju će biti priložena lista različitih definicija metode *add()*, uključujući i preteći opis:

- `add(String name, BigDecimal value)` - Poziv ove metode dodaje **JsonNumber** reprezentaciju **BigDecimal** vrednosti za generisani **JSON** objekat;
- `add(String name, BigInteger value)` - Poziv ove metode dodaje **JsonNumber** reprezentaciju **BigInteger** vrednosti za generisani **JSON** objekat;
- `add(String name, boolean value)` - Poziv ove metode dodaje vrednost **JsonValue.TRUE** ili **JsonValue.FALSE** za generisani **JSON** objekat u zavisnosti od vrednosti tipa **Boolean** koja je prosledjena kao parametar;
- `add(String name, double value)` - Poziv ove metode dodaje **JsonNumber** reprezentaciju **double** vrednosti za generisani **JSON** objekat;
- `add(String name, int value)` - Poziv ove metode dodaje **JsonNumber** reprezentaciju **int** vrednosti za generisani **JSON** objekat;
- `add(String name, JsonArrayBuilder builder)` - Poziv ove metode dodaje niz **JSON** objekata za generisani **JSON** objekat;
- `add(String name, JsonObjectBuilder builder)` - Poziv ove metode dodaje drugi **JSON** objekat za generisani **JSON** objekat;
- `add(String name, JsonValue value)` - Dodaje implementaciju **JsonValue** interfejsa za generisani **JSON** objekat;
- `add(String name, long value)` - Poziv ove metode dodaje **JsonNumber** reprezentaciju **long** vrednosti za generisani **JSON** objekat;
- `add(String name, String value)` - Poziv ove metode dodaje **String** reprezentaciju **long** vrednosti za generisani **JSON** objekat;

## KOD ZRNA JSON-P API OBJEKTNOG MODELA - DODATNA RAZMATRANJA

*Neophodno je dati dodatna objašnjenja u vezi sa kodom zrna JSON-P API objektnog modela.*

Ako se pogleda kod koji je priložen u sekciji "Kreiranje zrna JSON-P API objektnog modela" moguće je primetiti da je nakon poziva lanca metoda `add()` izvršen poziv metode `build()` za **JsonObjectBuilder** implementaciju. Pozivom ove metode vraća se instanca klase koja implementira **JsonObject**.

Kada je dobijena implementacija za **JsonObject** neophodno je izvršiti njeno konvertovanje u odgovarajuću **String** reprezentaciju. Navedeno je omogućeno pozivom statičke metode `createWriter()` klase **Json** koja prosleđuje novu instancu **StringWriter** kao parametar. Navedena metoda vraća instancu klase koja implementira interfejs **JsonWriter**.

Konačno, moguće je pozvati metodu `writeObject()` interfejsa **JsonWriter**, prosleđujući **JsonObject** instancu prethodno kreiranu kao parametar. Pozivom ove metode popunjava se objekat **StringWriter** koji je upotrebljen za kreiranje interfejsa **JsonWriter**. Sada je moguće dobiti **JSON String** reprezentaciju konkretnog objekta jednostavnim pozivom metode `toString()` za **StringWriter**.

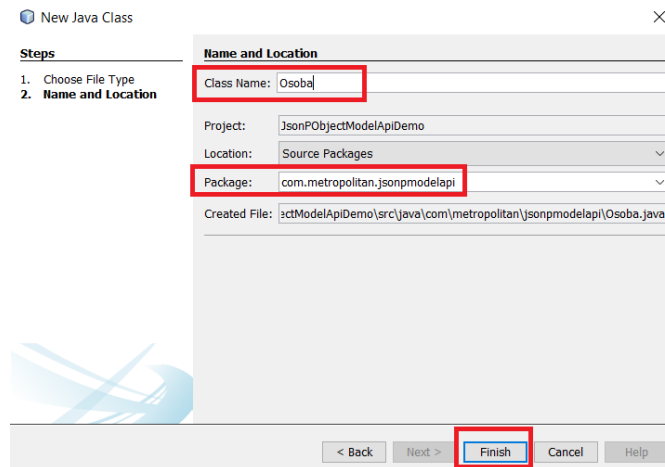
## KREIRANJE CDI ZRNA

*Neophodno je kreirati klasu iz koje se generiše JSON String.*

U nastavku je neophodno pozabaviti se klasom *Osoba.java* na osnovu čijeg sadržaja se i generiše traženi *JSON String*. Ova klasa predstavlja CDI zrna, a to znači da će biti obeležena anotacijama *@Named* i *@SessionScoped*. Takođe, ova klasa će morati da implementira interfejs *Serializable*.

Veoma lako je naslutiti da će ova klasa sadržati privatna polja: *ime*, *prezime*, *pol* i *starost*, zajedno sa pripadajućim *setter* i *getter* metodama.

Sledi listing klase *Osoba.java*, koja je kreirana na dobro poznat način primenom *NetBeans IDE* čarobnjaka (sledeća slika).



Slika 2.1.2 Kreiranje CDI zrna *Osoba.java* [izvor: autor]

```
package com.metropolitan.jsonmodelapi;
import java.io.Serializable;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@SessionScoped
public class Osoba implements Serializable{
    private String ime;
    private String prezime;
    private String pol;
    private Integer starost;

    public String getIme() {
        return ime;
    }
}
```

```

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getPrezime() {
        return prezime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }

    public String getPol() {
        return pol;
    }

    public void setPol(String pol) {
        this.pol = pol;
    }

    public Integer getStarost() {
        return starost;
    }

    public void setStarost(Integer starost) {
        this.starost = starost;
    }
}

```

## JSF STRANICA SA FORMOM ZA GENERISANJE JSON STRING - A

*U nastavku je neophodno obezbediti unos podataka za JSON reprezentaciju.*

Primer, koji je podrška analizi i diskusiji u ovom delu lekcije, zamišljen je kao jednostavna Java veb aplikacija koja koristi JSF stranicu za popunjavanje *CDI zrna* i generisanje *JSON String* - a iz sadržaja zrna. Stranica će da sadrži formu koja sadrži nekoliko polja za unos teksta koja su u direktnoj vezi sa osobinama *CDI zrna Osoba.java*.

Važna kontrola na ovoj formi jeste komandno dugme *Kreiraj JSON* koje prenosi kontrolu metodi *generateJson()* klase *JsonPModelApiZrno.java* koja predstavlja zrno *JSON-P API objektnog modela* o kojem je bilo detaljno govora u prethodnom izlaganju.

Sledi listing koda koji se čuva u datoteci *index.xhtml*, a koji odgovara navedenoj *JSF* stranici.

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"

```

```

xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
  <title>Primena JSON - P API objektnog modela</title>
</h:head>
<h:body>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputLabel for="ime" value="Ime"/>
      <h:inputText id="ime" value="#{osoba.ime}"/>
      <h:outputLabel for="prezime" value="Prezime"/>
      <h:inputText id="prezimee" value="#{osoba.prezime}"/>
      <h:outputLabel for="pol" value="Pol"/>
      <h:inputText id="pol" value="#{osoba.pol}"/>
      <h:outputLabel for="starost" value="Starost"/>
      <h:inputText id="starost" value="#{osoba.starost}"/>
      <h:panelGroup/>
      <h:commandButton value="Kreiraj JSON"
        action="#{jsonPModelApiZrno.generateJson()}" />
    </h:panelGrid>
  </h:form>
</h:body>
</html>

```

Iz atributa **action** taga **<h:commandButton>** se jasno vidi kako se nakon klika na ovo dugme kontrolra prenosi ka klasi kontrolera koji poziva metodu za generisanje **JSON String** - a. Vrednost atributa **action**, kao što je moguće videti iz listinga, odgovara izrazu **`#{jsonPModelApiZrno.generateJson()}`**.

## JSF STRANICA ZA PRIKAZIVANJE KREIRANOG JSON STRING - A

*Neophodno je kreirati stranicu koja će prikazati generisanu JSON reprezentaciju.*

Da bi posao, koji je do sada urađen na konkretnom primeru, bilo moguće testirati, neophodno je kreirati još jednu **JSF** stranicu čiji će zadatak biti prikazivanje kreiranog **JSON stringa**. Ako se pogleda ponovo kod kontrolera **JsonPModelApiZrno.java**, moguće je primetiti da je za prikazivanje **JSON Stringa** moguće kreirati **JSF** stranicu **prikazi\_popunjen\_obj.xhtml**. Na ovoj stranici, pored prikazanog generisanog JSON stringa (videti Sliku 4) nalazi se i komandno dugme **Potvrdi** koji se kontrolra prenosi na kontroler **JsonPModelApiZrno.java**. Tačnije rečeno, biće angažovana metoda **parseJson()** o kojoj će više biti reči u narednom izlaganju. Sledi listing navedene JSF stranice.

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Generisan JSON pomoću JSON-P API objektnog modela</title>
  </h:head>

```

```
<h:body>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputLabel for="parsiranJson" value="Parsiran JSON"/>
      <h:inputTextarea
        value="#{jsonPModelApiZrno.jsonStr}" rows="4"/>
      <h:panelGroup/>
      <h:commandButton value="Potvrđi"
        action="#{jsonPModelApiZrno.parseJson()}" />
    </h:panelGrid>
  </h:form>
</h:body>
</html>
```

Sada je moguće testirati urađeni posao. Popunjavanjem forme na stranici [index.xhtml](#) i klikom na dugme *Kreiraj JSON* (sledeća slika) kreira se JSON String.

Slika 2.1.3 Forma za kreiranje JSON Stringa [izvor: autor]

Slika 2.1.4 Kreirani JSON String [izvor: autor]

Ukoliko se u polju za unos teksta promene vrednosti u JSON String - u za odgovarajuće osobine, nakon klika na dugme Potvrđi originalne vrednosti objekta tipa Osoba biće ažurirane izmenjenim vrednostima.

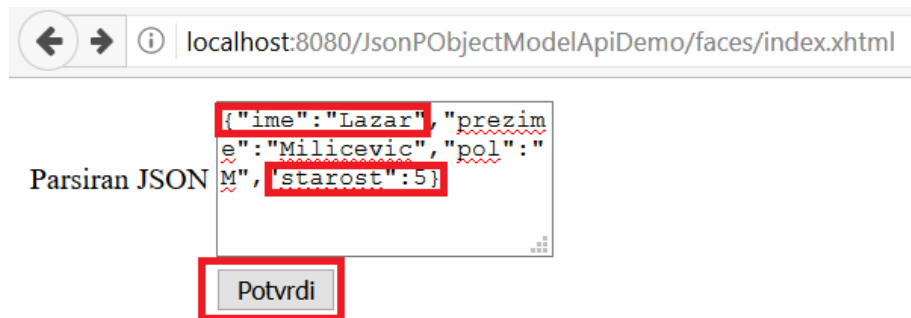
## ▼ 2.2 Parsiranje JSON podataka

### AŽURIRANJE CDI ZRNA IZ JSON STRINGA

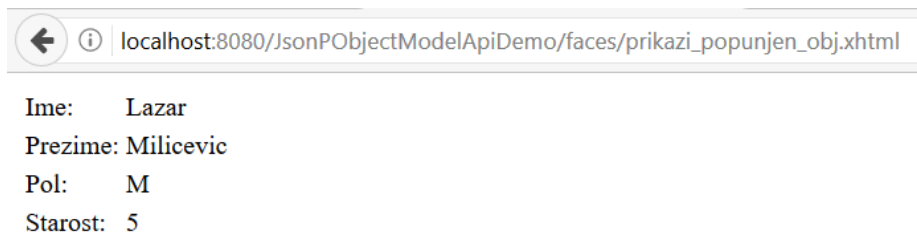
*Neophodno je prikazati obrnutu funkcionalnost iz prethodnog izlaganja.*

Nakon fokusiranja na kreiranje *JSON* reprezentacije Java objekata, sada je moguće posmatrati problem sa druge strane. Moguće je iz *JSON String* - a izvršiti popunjavanje Java objekata odgovarajućim sadržajem. Ako se pogleda Slika 4, prethodnog objekta učenja, moguće je primetiti da se na njoj nalazi polje za unos teksta sa generisanim *JSON String* - om i jedno kontrolno dugme. Ideja je da se klikom na to dugme vrši ažuriranje sadržaja Java objekta iz kojeg je, u prethodnoj analizi i diskusiji i kreiran *JSON String*.

Neka je tako i urađeno, pojedini podaci u *JSON String* - u su izmenjeni, kao na sledećoj slici, nakon čega je kliknuto na dugme *Potvrdi*. Nakon toga se otvara stranica *generisan\_json.xhtml* koja pokazuje ažurirane podatke objekta klase *Osoba.java* (Videti Sliku 2).



Slika 2.2.1 Izmenjen JSON String [izvor: autor]



Slika 2.2.2 Ažuriran sadržaj objekta klase Osoba.java [izvor: autor]

Sledećim listingom je priložena sadržaj JSF datoteke *generisan\_json.xhtml*.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Osobine Java objekta popunjenog iz JSON</title>
  </h:head>
  <h:body>
    <table>
      <tr>
        <td>
          Ime:
        </td>
        <td>
          #{osoba.ime}
        </td>
      </tr>
    </table>
  </h:body>
</html>
```

```

        </tr>
        <tr>
            <td>
                Prezime:
            </td>
            <td>
                #{osoba.prezime}
            </td>
        </tr>
        <tr>
            <td>
                Pol:
            </td>
            <td>
                #{osoba.pol}
            </td>
        </tr>
        <tr>
            <td>
                Starost:
            </td>
            <td>
                #{osoba.starost}
            </td>
        </tr>
    </table>
</h:body>
</html>

```

## METODA ZA PARSIRANJE ZRNA JSON-P API OBJEKTNOG MODELA

*Neophodno je analizirati kod za parsiranje priložene klase zrna JSON-P API objektnog modela.*

Dakle, u prethodnoj sekciji je pokazana obrnuta funkcionalnost u odnosu na demonstraciju iz objekta učenja "*Generisanje JSON podataka*". Dakle, iz *JSON String* - a su popunjene osobine objekta tipa *Osoba*. U ovom delu lekcije će biti izolovan kod klase *JsonPModelApiZrno.java* zadužen za omogućavanje navedene funkcionalnosti.

```

package com.metropolitan.jsonpmodelapi;

// import instrukcije su izostavljene jer su već prikazane
/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JsonPModelApiZrno {
    @Inject

```



```
private Osoba osoba;
private String jsonStr;

// kod za generisanje JSON je izostavljen jer je vec prikazan

public String parseJson() {
    JsonObject jsonObject;

    try (JsonReader jsonReader = Json.createReader(new StringReader(jsonStr))) {
        jsonObject = jsonReader.readObject();
    }

    osoba.setIme(jsonObject.getString("ime"));
    osoba.setPrezime(jsonObject.getString("prezime"));
    osoba.setPol(jsonObject.getString("pol"));
    osoba.setStarost(jsonObject.getInt("starost"));

    return "generisan_json";
}

public String getJsonStr() {
    return jsonStr;
}

public void setJsonStr(String jsonStr) {
    this.jsonStr = jsonStr;
}
}
```

Kao što je moguće naslutiti, glavni akcenat u narednom izlaganju će biti na metodi ***parseJson()*** klase *JsonPModelApiZrno.java*.

Iz priloženog koda je moguće primetiti da je prvo kreirana instanca ***java.io.StringReader*** iz ***JSON String*** - a. Navedeno je obavljeno prosleđivanjem ***String*** - a koji sadrži ***JSON*** podatak konstruktoru klase ***StringReader***. Nakon toga, neophodno je izvršiti prosleđivanje rezultujuće ***StringReader*** instance statičkoj metodi ***createReader()*** ***JSON-P*** klase ***javax.json.Json***. Poziv ove metode vraća implementaciju interfejsa ***javax.json.JsonReader*** koja se, nakon toga koristi, za dobijanje implementacije za ***javax.json.JsonObject***.

Napokon, priloženim **setter** metodama promenjen su vrednosti polja objekta osoba klase *Osoba*. U ovim pozivima moguće je uočiti i **getter** metode JSON objekta o kojima će detaljno biti govora u narednom izlaganju.

## GET METODE JSON OBJEKTA

*JSON objekat sadrži nekoliko GET metoda koje nisu analizirane u prethodnom izlaganju.*

Kao što je istaknuto u prethodnom izlaganju sledi posebna diskusija koja je u direktnoj vezi sa **getter** metodama *JSON* objekta.

*JSON* objekat sadrži nekoliko *GET* metoda koje je obavezno potrebno izložiti i opisati u narednom izlaganju. Sledi lista ovih metoda:

- *getBoolean(String name)* - Metoda vraća vrednost osobine specificirane kao tip **Boolean**;
- *getInt(String name)* - Metoda vraća vrednost osobine specificirane kao tip **int**;
- *getJSONArray(String name)* - Metoda vraća vrednost osobine specificirane kao niz oblika **JSONArray** implementacije;
- *getJsonNumber(String name)* - Metoda vraća vrednost osobine specificirane kao numerički tip. Vrednost može biti naknadno konvertovana u **int**, **long** ili **double**, pozivom metoda *intValue()*, *longValue()* ili *doubleValue()*, respektivno;
- *getJsonObject(String name)* - Metoda vraća vrednost osobine specificirane kao **JsonObject** implementacija;
- *getString(String name)* - Metoda vraća vrednost osobine specificirane kao **JsonString** implementacija;
- *getString(String name)* - Metoda vraća vrednost osobine specificirane kao tip **String**.

Ovim izlaganjem završena je analiza i demonstracija problematike vezane za oblast *JSON-P object model API*. Lekcija dalje nastavlja da se bavi specifičnom problematikom JSON procesiranja primenom *JSON-P API tokova*.

## JAVA JSON-P API NOVITETI KROZ VIDEO MATERIJAL

*Updates to the Java API for JSON Processing for Java EE 8.*

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

## ▼ Poglavlje 3

# JSON-P API tokova

## UVOD U JSON-P API TOKOVA

*Ovaj deo lekcije se bavi generisanjem i parsiranjem JSON podataka pomoću JSON-P API tokova.*

Ovaj deo lekcije se fokusira na izučavanje problematike koja je u direktnoj vezi sa JSON-P API tokova (JSON-P streaming API).

Za *JSON-P API tokova*, ovaj deo lekcije će analizu, diskusiju i demonstraciju podeliti u dve grane:

- analiza, diskusija i demonstracija generisanja *JSON* podataka pomoću *JSON-P API tokova*;
- analiza, diskusija i demonstracija parsiranja *JSON* podataka pomoću *JSON-P API tokova*.

*JSON-P API tokova* dozvoljava čitanje i pisanje *JSON* podataka iz toka (*stream*). *Tok* predstavlja potklasu neke od klasa *java.io.OutputStream* ili *java.io.Writer*. Ako se uporede *API* - ji, *JSON-P API tokova* pokazuje bolje rezultate po pitanju performansi i iskorišćenja memorije u odnosu na *JSON-P API objektnog modela*. Međutim, prednosti u performansama i iskorišćenju memorije ipak dolaze sa izvesnim ograničenjima. *JSON-P API tokova* dozvoljava sekvencijalno čitanje *JSON* podataka. To praktično znači da nije moguće, na ovaj način, direktno pristupiti *JSON* osobinama kao što je to bio slučaj kada je korišćen *JSON-P API objektnog modela*. U osnovi, poželjno je koristiti *JSON-P API tokova* kada se rukuje ogromnom količinom *JSON* podataka, u suprotnom mnogo je jednostavnije i elegantnije koristiti *JSON-P API objektnog modela*.

Posebno bi trebalo napomenuti da će izlaganje, koje prati ovaj deo lekcije biti praćeno odgovarajućim praktičnim primerima, veoma pažljivo biranim i testiranim, kako bi teorijska razmatranja bila razumljivija.

## ▼ 3.1 Generisanje JSON podataka za JSON-P API tokova

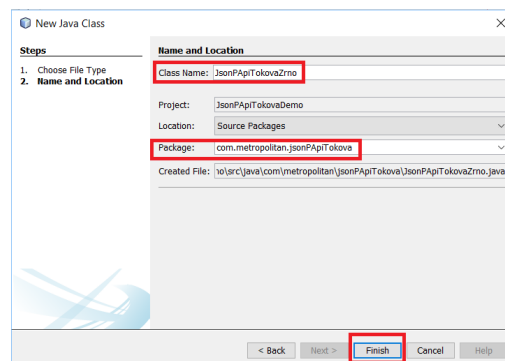
### KREIRANJE ZRNA JSON-P API TOKOVA

*U nastavku izlaganja tema će biti zrno JSON-P API tokova.*

U nastavku izlaganja tema će biti zrno *JSON-P API tokova*. Kada se koristi *JSON-P API tokova*, neophodno je izvršiti generisanje *JSON* podataka preko klase *JsonGenerator*, za koju se vrši pozivanje jedne ili više redefinisanih (*overload*) metoda *write()* za dodavanje *JSON* osobina i njima odgovarajućih vrednosti u *JSON* podatke.

Da bi primena *JSON-P API tokova* bila kvalitetnije analizirana i diskutovana biće kreiran odgovarajući primer, u formi Java EE veb projekta pod nazivom *JsonPApiTokovaDemo*.

Za sam početak, u ovom projektu biće kreirana klasa koja odgovara zrnu *JsonPApiTokovaZrno*. Sledećom slikom je prikazan prozor, *NetBeans IDE* čarobnjaka, u kojem se definiše naziv zrna i paketa kojem zrno pripada.



Slika 3.1.1 Kreiranje zrna JSON-P API tokova [izvor: autor]

Kod klase pod nazivom *JsonPApiTokovaZrno.java*, iz paketa *com.metropolitan.jsonPApiTokova*, priložen je u potpunosti sledećim listingom.

```
package com.metropolitan.jsonPApiTokova;
import java.io.StringReader;
import java.io.StringWriter;
import java.util.HashMap;
import java.util.Map;
import javax.enterprise.context.RequestScoped;
import javax.inject.Inject;
import javax.inject.Named;
import javax.json.Json;
import javax.json.stream.JsonGenerator;
import javax.json.stream.JsonParser;
import javax.json.stream.JsonParser.Event;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JsonPApiTokovaZrno {
    @Inject
    private Osoba osoba;
    private String jsonStr;

    public String generateJson() {
```

```
StringWriter stringWriter = new StringWriter();
try (JsonGenerator jsonGenerator
    = Json.createGenerator(stringWriter)) {
    jsonGenerator.writeStartObject().
        write("ime", osoba.getIme()).
        write("prezime", osoba.getPrezime()).
        write("pol", osoba.getPol()).
        write("starost", osoba.getStarost()).
        writeEnd();
}

setJsonStr(stringWriter.toString());
return "prikazi_popunjen_obj";
}

public String parseJson() {
    StringReader stringReader = new StringReader(jsonStr);

    JsonParser jsonParser = Json.createParser(stringReader);

    Map<String, Object> jsonMap = new HashMap<>();
    String jsonKeyNm = null;
    Object jsonVal = null;

    while (jsonParser.hasNext()) {
        JsonParser.Event event = jsonParser.next();

        if (event.equals(Event.KEY_NAME)) {
            jsonKeyNm = jsonParser.getString();
        } else if (event.equals(Event.VALUE_STRING)) {
            jsonVal = jsonParser.getString();
        } else if (event.equals(Event.VALUE_NUMBER)) {
            jsonVal = jsonParser.getInt();
        }

        jsonMap.put(jsonKeyNm, jsonVal);
    }

    osoba.setIme((String) jsonMap.get("ime"));
    osoba.setPrezime((String) jsonMap.get("prezime"));
    osoba.setPol((String) jsonMap.get("pol"));
    osoba.setStarost((Integer) jsonMap.get("starost"));

    return "generisan_json";
}

public String getJsonStr() {
    return jsonStr;
}

public void setJsonStr(String jsonStr) {
    this.jsonStr = jsonStr;
}
```

```
}
```

## ANALIZA KODA ZRNA JSON-P API TOKOVA

*U nastavku će biti obavljena detaljna analiza priloženog listinga zrna JSON-P API tokova.*

Za generisanje *JSON String* - a iz *JSON-P API tokova*, ako se pogleda priloženi listing klase *JsonPApiTokovaZrna*, od posebnog je značaja metoda *generateJson()*.

Za generisanje *JSON* podataka primenom *JSON-P API tokova*, prvo je neophodno pozvati statičku metodu *Json.createGenerator()*. Ova metoda vraća instancu klase koja implementira interfejs *javax.json.stream.JsonGenerator*. Postoje dve različite verzije (*overload*) metode *Json.createGenerator()*. U prvoj verziji metoda preuzima instancu klase *java.io.OutputStream* (ili neke od njenih potklasa) kao parametar. U drugom slučaju metoda preuzima instancu klase *java.io.Writer* (ili neke od njenih potklasa) kao parametar. U posmatranom slučaju vrši se prosleđivanje instance *java.io.StringWriter* ka pozivi *Json.createGenerator()*.

Kada se obezbedi instanca *JsonGenerator* neophodno je za nju pozvati metodu *writeStartObject()*. Ova metoda vrši upisivanje početnog karaktera objekta *JSON* - a (otvara veliku zagradu "{") u *OutputStream* ili *Writer* koji je prosleđen pozivu *Json.createGenerator()*. Metoda *writeStartObject()* vraća drugu instancu za *JsonGenerator* i pri tome omogućava direktno pozivanje metode *write()* rezultujućeg objekta tipa *JsonGenerator*.

Za lakše sagledavanje prethodne diskusije sledi izolovan listing metode *generateJson()* klase zrna *JSON-P API tokova JsonPApiTokovaZrna.java*.

```
public String generateJson() {
    StringWriter stringWriter = new StringWriter();
    try (JsonGenerator jsonGenerator
        = Json.createGenerator(stringWriter)) {
        jsonGenerator.writeStartObject().
            write("ime", osoba.getIme()).
            write("prezime", osoba.getPrezime()).
            write("pol", osoba.getPol()).
            write("starost", osoba.getStarost()).
            writeEnd();
    }

    setJsonStr(stringWriter.toString());
    return "prikazi_popunjen_obj";
}
```

U nastavku predstoji diskusija o različitim oblicima u kojima može da se pojavi pomenuta metoda *write()*.

## RAZLIČITI OBLICI METODE WRITE() I DODATNA RAZMATRANJA

*Postoji podrška za nekoliko različitih načina potpisivanja (overloading) metode write().*

Kao što je istaknuto u prethodnom izlaganju, biće neophodno više pažnje posvetiti metodi **write()** klase **JsonGenerator** čiji je zadatak dodavanje **JSON** osobina u **JSON** podatke. Metoda uvek ima dva parametra. Prvi parametar odgovara nazivu osobine, dok se drugi parametar odnosi na odgovarajuću vrednost. Postoji podrška za nekoliko različitih načina potpisivanja (overloading) metode **write()**. U konkretnom primeru korišćeni su parametri tipa **String** i **Integer** u metodama **write()**. Međutim, zbog budućeg rada i sticanja šire slike u vezi sa primenom ove metode, u sledećem izlaganju sledi navođenje različitih verzija ove metode sa odgovarajućim opisom:

- **write(String name, BigDecimal value)** - Dodaje numeričku osobinu tipa **BigDecimal** u **JSON** podatke;
- **write(String name, BigInteger value)** - Dodaje numeričku osobinu tipa **BigInteger** u **JSON** podatke;
- **write(String name, JsonValue value)** - Dodaje osobinu tipa **JsonValue** ili nekog od njegovih podinterfejsa (**JsonArray**, **JsonNumber**, **JsonObject**, **JsonString** ili **JsonStructure**) u **JSON** podatke;
- **write(String name, String value)** - Dodaje osobinu tipa **String** u **JSON** podatke;
- **write(String name, boolean value)** - Dodaje osobinu tipa **Boolean** u **JSON** podatke;
- **write(String name, double value)** - Dodaje numeričku osobinu tipa **double** u **JSON** podatke;
- **write(String name, int value)** - Dodaje numeričku osobinu tipa **int** u **JSON** podatke;
- **write(String name, long value)** - Dodaje numeričku osobinu tipa **long** u **JSON** podatke.

Nakon što je obavljeno dodavanje svih osobina u **JSON** podatke, neophodno je izvršiti pozivanje metode **writeEnd()** klase **JsonGenerator** koja dodaje završni karakter objekta **JSON** - a (zatvorena velika zagrada **"}**") u **JSON String**.

U ovom trenutku, **Writer** ili **OutputStream** koji je prosleđen ka **Json.createGenerator()** sadrži kompletan **JSON** objekat. Sudbina ovog objekta dalje zavisi od zahteva aplikacije. U konkretnom slučaju jednostavno se poziva metoda **toString()** instance **StringWriter** koja je upotrebljena i njena povratna vrednost je, potom, pridružena varijabli **jsonStr**.

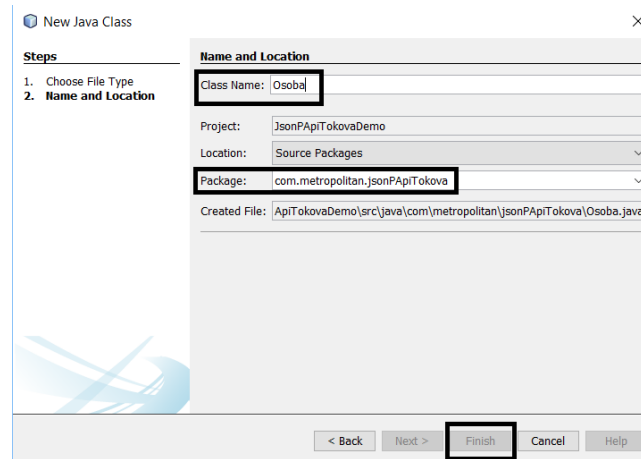
## KREIRANJE CDI ZRNA ZA PRIMER JSON-P API TOKOVA

*Neophodno je kreirati klasu iz koje se generiše JSON String.*

U nastavku je neophodno pozabaviti se klasom **Osoba.java** na osnovu čijeg sadržaja se i generiše traženi **JSON String**. Ova klasa predstavlja CDI zrno, a to znači da će biti obeležena anotacijama **@Named** i **@SessionScoped**. Takođe, ova klasa će morati da implementira interfejs **Serializable**.

Veoma lako je naslutiti da će ova klasa sadržati privatna polja: *ime*, *prezime*, *pol* i *starost*, zajedno sa pripadajućim *setter* i *getter* metodama.

Sledi listing klase *Osoba.java*, u paketu *com.metropolitan.jsonPApiTokova*, koja je kreirana na dobro poznat način primenom *NetBeans IDE* čarobnjaka (sledeća slika).



Slika 3.1.2 Kreiranje CDI zrna *Osoba.java* za JSON-P API tokova [izvor: autor]

```
package com.metropolitan.jsonPApiTokova;
import java.io.Serializable;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@SessionScoped
public class Osoba implements Serializable{
    private String ime;
    private String prezime;
    private String pol;
    private Integer starost;

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getPrezime() {
        return prezime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }
}
```



```
}

public String getPol() {
    return pol;
}

public void setPol(String pol) {
    this.pol = pol;
}

public Integer getStarost() {
    return starost;
}

public void setStarost(Integer starost) {
    this.starost = starost;
}
}
```

## JSF STRANICA SA FORMOM ZA GENERISANJE JSON STRING - A

*I u slučaju, kada se koristi JSON-P API tokova, neophodno je kreirati odgovarajuće JSF stranice.*

Kao što je bio slučaj u prethodnom izlaganju, koje se odnosilo na primenu *JSON-P API objektnog modela*, i u ovom slučaju, kada se koristi *JSON-P API tokova*, neophodno je kreirati odgovarajuće *JSF* stranice preko kojih korisnici aplikacije mogu da vrše unos podataka za generisanje *JSON String* - a, preko odgovarajuće forme, i prikazivanje generisanog *JSON String* - a.

Kao i u slučaju kada je korišćen *JSON-P API objektnog modela*, stranica *index.xhtml* će, takođe, da sadrži formu koja sadrži nekoliko polja za unos teksta koja su u direktnoj vezi sa osobinama CDI zrna *Osoba.java*.

Važna kontrola na ovoj formi jeste komandno dugme *Kreiraj JSON* koje prenosi kontrolu metodi *generateJson()* nove klase *JsonPApiTokovaZrno.java* koja predstavlja zrno *JSON-P API tokova* o kojem je bilo detaljno govora u prethodnom izlaganju.

Sledi listing koda koji se čuva u datoteci *index.xhtml*, a koji odgovara navedenoj *JSF* stranici. Listing je veoma sličan listingu priloženom za JSF stranicu *index.xhtml* primera koji koristi *JSON-P API objektnog modela*. Kada se uporede navedeni listinzi uočava se razlika u prosleđivanju kontrole različitim kontrolerima, u zavisnosti od izabranog pristupa JSON-P API objektnog modela ili SON-P API tokova.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
```

```
<h:head>
  <title>Primena JSON - P API tokova</title>
</h:head>
<h:body>
  <h:form>
    <h:panelGrid columns="2">
      <h:outputLabel for="ime" value="Ime"/>
      <h:inputText id="ime" value="#{osoba.ime}"/>
      <h:outputLabel for="prezime" value="Prezime"/>
      <h:inputText id="prezimee" value="#{osoba.prezime}"/>
      <h:outputLabel for="pol" value="Pol"/>
      <h:inputText id="pol" value="#{osoba.pol}"/>
      <h:outputLabel for="starost" value="Starost"/>
      <h:inputText id="starost" value="#{osoba.starost}"/>
    </h:panelGrid>
    <h:commandButton value="Kreiraj JSON"
                      action="#{jsonPApiTokovaZrno.generateJson()}" />
  </h:form>
</h:body>
</html>
```

Iz atributa **action** taga **<h:commandButton>** se jasno vidi kako se nakon klika na ovo dugme kontroler prenosi ka klasi kontrolera koji poziva metodu za generisanje **JSON String** - a. Vrednost atributa **action**, kao što je moguće videti iz listinga, odgovara izrazu **`#{jsonPApiTokovaZrno.generateJson()}`**.

## JSF STRANICA ZA PRIKAZIVANJE KREIRANOG JSON STRING - A

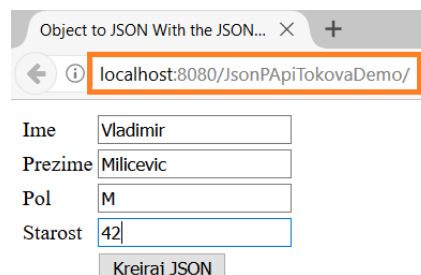
*Neophodno je kreirati stranicu koja će prikazati generisanu JSON reprezentaciju.*

Da bi posao, koji je do sada urađen na konkretnom primeru, bilo moguće testirati, neophodno je kreirati još jednu **JSF** stranicu čiji će zadatak biti prikazivanje kreiranog **JSON stringa** primenom **JSON-P API tokova**. Ako se pogleda ponovo kod kontrolera **JsonPApiTokovaZrno.java**, moguće je primetiti da je za prikazivanje **JSON Stringa** moguće kreirati **JSF** stranicu **prikazi\_popunjen\_obj.xhtml**. Na ovoj stranici, pored prikazanog generisanog JSON stringa (videti Sliku 4) nalazi se i komandno dugme **Potvrdi** koji se kontrola prenosi na kontroler **JsonPApiTokovaZrno.java**. Tačnije rečeno, biće angažovana metoda **parseJson()** o kojoj će više biti reči u narednom izlaganju. Sledi listing navedene JSF stranice.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Generisan JSON pomoću JSON-P API objektnog modela</title>
  </h:head>
  <h:body>
```

```
<h:form>
  <h:panelGrid columns="2">
    <h:outputLabel for="parsiranJson" value="Parsiran JSON"/>
    <h:inputTextarea
      value="#{jsonPApiTokovaZrno.jsonStr}" rows="4"/>
    <h:panelGroup/>
    <h:commandButton value="Potvrdi"
      action="#{jsonPApiTokovaZrno.parseJson()}" />
  </h:panelGrid>
</h:form>
</h:body>
</html>
```

Sada je moguće testirati urađeni posao. Popunjavanjem forme na stranici [index.xhtml](#) i klikom na dugme *Kreiraj JSON* (sledeća slika) kreira se JSON String.



Object to JSON With the JSON... X +

localhost:8080/JsonPApiTokovaDemo/

Ime: Vladimir

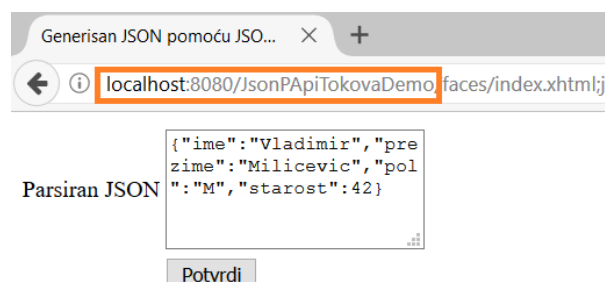
Prezime: Milicevic

Pol: M

Starost: 42

Kreiraj JSON

Slika 3.1.3 Forma za kreiranje JSON Stringa [izvor: autor]



Generisan JSON pomoću JSO... X +

localhost:8080/JsonPApiTokovaDemo/faces/index.xhtml;j

Parsiran JSON

```
{"ime": "Vladimir", "prezime": "Milicevic", "pol": "M", "starost": 42}
```

Potvrdi

Slika 3.1.4 Kreirani JSON String [izvor: autor]

Moguće je primetiti iz priloženih slika da je ovaj deo zadatka, koji se osnosi na kreiranje JSON Stringa primenom JSON-P API tokova urađen na uspešan način kao što je prethodno bio slučaj sa primenom JSON-P API objektnog modela.

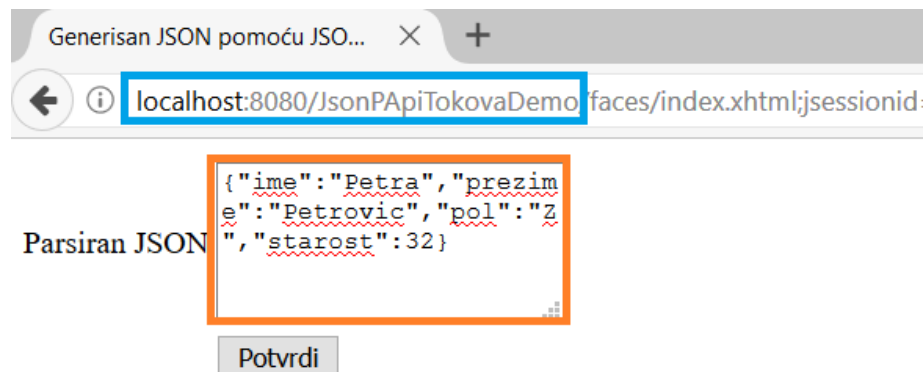
## ▼ 3.2 Parsiranje JSON podataka za JSON-P API tokova

### AŽURIRANJE CDI ZRNA IZ JSON STRINGA ZA JSON-P API TOKOVA

*Neophodno je pokazati obrnutu funkcionalnos iz prethodnog izlaganja.*

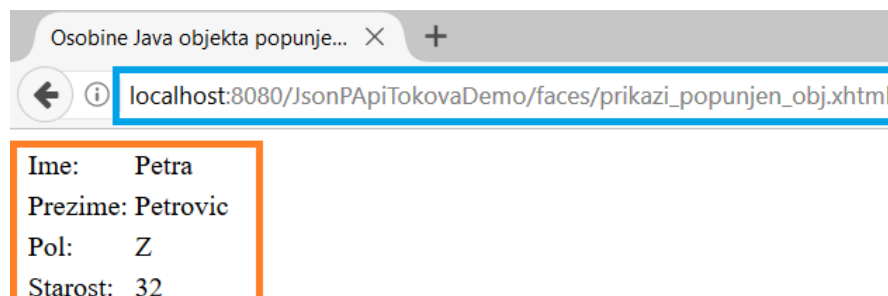
Nakon fokusiranja na kreiranje *JSON* reprezentacije Java objekata, sada je moguće posmatrati problem sa druge strane. Moguće je iz *JSON String* - a izvršiti popunjavanje Java objekata odgovarajućim sadržajem. Ako se pogleda Slika 4, prethodnog objekta učenja, moguće je primetiti da se na njoj nalazi polje za unos teksta sa generisanim *JSON String* - om i jedno kontrolno dugme. Ideja je da se klikom na to dugme vrši ažuriranje sadržaja Java objekta iz kojeg je, u prethodnoj analizi i diskusiji i kreiran *JSON String*.

Neka je tako i urađeno, pojedini podaci u *JSON String* - u su izmenjeni, kao na sledećoj slici, nakon čega je kliknuto na dugme *Potvrdi*. Nakon toga se otvara stranica *generisan\_json.xhtml* koja pokazuje ažurirane podatke objekta klase *Osoba.java* (Videti Sliku 2).



Slika 3.2.1 Izmenjen originalno generisani JSON String [izvor: autor]

Sledećim listingom je priložena sadržaj JSF datoteke *generisan\_json.xhtml*.



Slika 3.2.2 Ažuriran sadržaj objekta klase Osoba.java [izvor: autor]

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
```

```
xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
  <title>Osobine Java objekta popunjenog iz JSON</title>
</h:head>
<h:body>
  <table>
    <tr>
      <td>
        Ime:
      </td>
      <td>
        #{osoba.ime}
      </td>
    </tr>
    <tr>
      <td>
        Prezime:
      </td>
      <td>
        #{osoba.prezime}
      </td>
    </tr>
    <tr>
      <td>
        Pol:
      </td>
      <td>
        #{osoba.pol}
      </td>
    </tr>
    <tr>
      <td>
        Starost:
      </td>
      <td>
        #{osoba.starost}
      </td>
    </tr>
  </table>
</h:body>
</html>
```

## METODA ZA PARSIRANJE ZRNA JSON-P API TOKOVA

*Neophodno je analizirati kod za parsiranje priložene klase zrna JSON-P API tokova.*

Dakle, u prethodnoj sekciji je pokazana obrnuta funkcionalnost u odnosu na demonstraciju iz objekta učenja "*Generisanje JSON podataka*". Dakle, iz *JSON String* - a su popunjene osobine objekta tipa *Osoba*. U ovom delu lekcije će biti izolovan kod klase *JsonPApiTokovaZrno.java* zadužen za omogućavanje navedene funkcionalnosti.

```
package com.metropolitan.jsonPApiTokova;
//import instrukcije su izostavljene jer su već pokazane

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@RequestScoped
public class JsonPApiTokovaZrno {
    @Inject
    private Osoba osoba;
    private String jsonStr;

    //Metoda generateJson() je izostavljena jer je već pokazana

    public String parseJson() {
        StringReader stringReader = new StringReader(jsonStr);

        JsonParser jsonParser = Json.createParser(stringReader);

        Map<String, Object> jsonMap = new HashMap<>();
        String jsonKeyNm = null;
        Object jsonVal = null;

        while (jsonParser.hasNext()) {
            JsonParser.Event event = jsonParser.next();

            if (event.equals(Event.KEY_NAME)) {
                jsonKeyNm = jsonParser.getString();
            } else if (event.equals(Event.VALUE_STRING)) {
                jsonVal = jsonParser.getString();
            } else if (event.equals(Event.VALUE_NUMBER)) {
                jsonVal = jsonParser.getInt();
            }

            jsonMap.put(jsonKeyNm, jsonVal);
        }

        osoba.setIme((String) jsonMap.get("ime"));
        osoba.setPrezime((String) jsonMap.get("prezime"));
        osoba.setPol((String) jsonMap.get("pol"));
        osoba.setStarost((Integer) jsonMap.get("starost"));

        return "generisan_json";
    }

    public String getJsonStr() {
        return jsonStr;
    }

    public void setJsonStr(String jsonStr) {
        this.jsonStr = jsonStr;
    }
}
```

```
}  
  
}
```

Kao što je moguće naslutiti, glavni akcenat u narednom izlaganju će biti na metodi **`parseJson()`** klase **`JsonPapiTokovaZrno.java`**.

Sa ciljem čitanja i parsiranja **JSON** objekata primenom **JSON-P API tokova**, neophodno je obezbediti implementaciju interfejsa **`JsonParser`**. Klasa **`Json`** poseduje dve različite verzije metode **`createParser()`** koje je moguće upotrebiti za dobijanje implementacije interfejsa **`JsonParser`**. U prvoj verziji **`Json.createParser()`** uzima instancu klase **`java.io.InputStream`** (ili neke od njenih potklasa) kao jedini vlastiti parametar. Druga verzija **`Json.createParser()`** uzima instancu klase **`java.io.Reader`** (ili neke od njenih potklasa) takođe kao jedini vlastiti parametar. U konkretnom slučaju primenjen je drugi slučaj koji podrazumeva prosleđivanje instance **`java.io.StringReader`** (predstavlja potklasu klase **`java.io.Reader`**) i koja sadrži **JSON String** kao parametar.

Kada je obezbeđena referenca na **`JsonParser`**, vrši se pozivanje metode **`hasNext()`** u odgovarajućoj **`while`** petlji. Poziv **`JsonParser.hasNext()`** će vratiti vrednost tačno (**`true`**) ukoliko postoji još naziva osobina, sa odgovarajućim vrednostima, koje je neophodno pročitati iz **JSON String** - a. U suptornom, navedeni izraz će vratiti vrednost netačno (**`false`**).

## METODA PARSEJSON() - DODATNA RAZMATRANJA

*Metoda **`parseJson()`** poseduje dodatne funkcionalnosti koje je neophodno diskutovati.*

Metoda **`parseJson()`**, klase **`JsonPapiTokovaZrno.java`**, poseduje dodatne funkcionalnosti koje je neophodno diskutovati. Fokus će biti na petlji koja je pomenuta u prethodnom izlaganju. Unutar petlje **`while`** vrši se poziv **`JsonParser.next()`**. Metoda vraća instancu klase **`JsonParser.Event`**. Specifična vrednost za **`JsonParser.Event`**, koja se dobija iz poziva **`JsonParser.next()`**, omogućava da se sagleda koji tip podataka se čita (naziv ključa - **`key name`**, vrednost stringa - **`string value`**, numerička vrednost - **`numeric value`** i tako dalje). U konkretnom primeru **JSON String** sadrži samo **String** i **numeričke vrednosti** pa je neophodno izvršiti proveru samo za ove dve vrednosti poredeći instancu **`JsonParser.Event`**, dobijenu iz **`JsonParser.next()`**, sa **`Event.VALUE_STRING`** i **`Event.VALUE_NUMBER`**, respektivno. Takođe je obavljena i provera naziva **JSON** ključa poređenjem dobijene vrednosti sa **`Event.KEY_NAME`**. Kada se pročita par **ključ / vrednost** kombinacija parova iz **JSON String** - a događa se aktivnost koja je u direktnoj zavisnosti sa zahtevima aplikacije. U konkretnom slučaju popunjava se heš mapa (**hash map**) odgovarajućim vrednostima dobijenim iz **JSON String** - a.

U prethodnom izlaganju su prikazane tri, od mogućih, vrednosti koje je moguće dobiti sekvencijalnim čitanjem **JSON** podataka angažovanjem poziva **`JsonParser.next()`**.

U sledećem izlaganju su priložene i ostale vrednosti, zajedno sa odgovarajućim opisom, koje je moguće dobiti sekvencijalnim čitanjem **JSON** podataka angažovanjem poziva **`JsonParser.next()`**:

- *Event.START\_OBJECT* - Ukazuje na početak *JSON* objekta;
- *Event.END\_OBJECT* - Ukazuje na kraj *JSON* objekta;
- *Event.KEY\_NAME* - Ukazuje na naziv *JSON* osobine;
- *Event.VALUE\_STRING* - Ukazuje da je **String** vrednost pročitana;
- *Event.VALUE\_NUMBER* - Ukazuje da je numerička vrednost pročitana;
- *Event.VALUE\_TRUE* - Ukazuje da je logička vrednost tačno (**true**) pročitana;
- *Event.VALUE\_FALSE* - Ukazuje da je logička vrednost netačno (**false**) pročitana;
- *Event.VALUE\_NULL* - Ukazuje da je **null** vrednost pročitana;
- *Event.VALUE\_START\_ARRAY* - Ukazuje da početna vrednost niza pročitana;
- *Event.VALUE\_END\_ARRAY* - Ukazuje da je završna vrednost niza pročitana.

Sa poslednjim izlaganjem stavljena je tačka na teorijsko izlaganje vezano za problematiku JSON procesiranja primenom *JSON-P API tokova*.

## REZIME ZA JAVA JSON-P API KROZ VIDEO MATERIJAL

### *JSR 353: Java API for JSON Processing*

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



## ▼ Poglavlje 4

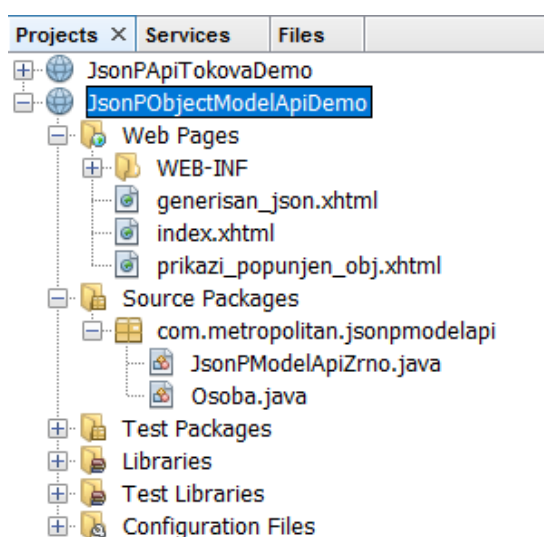
# Java API za JSON procesiranje - pokazni primer 1

## JSON-P API OBJEKTNOG MODELA (25 MIN)

*Vežbanje na računaru prektičnih zadataka iz problematike JSON-P API objektnog modela.*

Primenom Java EE 7 platforme i odgovarajućeg razvojnog okruženja (na primer NetBEans IDE) kreirati Java veb aplikaciju sa sledećim zahtevima:

1. Aplikacija ima mogućnost kreiranja i parsiranja JSON objekata primenom JSON-P API objektnog modela;
2. Osnova za kreiranje JSON dokumenta je odgovarajuća Java klasa (CDI zrno);
3. Kreirati odgovarajući kontroler za navedeno CDI zrno;
4. Aplikacija ima sposobnost da svaku promenu u generisanom JSON Stringu prenese u izvorno Java CDI zrno;
5. Kreirati odgovarajuće JSF stranice za prikupljanje podataka kojima će biti popunjeno CDI zrno, a potom i JSON String, za prikazivanje generisanog JSON Stringa i za prikazivanje parsiranih podataka iz JSON Stringa u Java CDI zrno.
6. Aplikacija može da ima strukturu priloženu sledećom slikom:



Slika 4.1 Predložena struktura veb aplikacije rešenja [izvor: autor]

## JSF STRANICE

*Prvo će biti priložene JSF stranice projekta.*

Sledi listing stranice index.xml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Object to JSON With the JSON-P Object Model API</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="ime" value="Ime"/>
                <h:inputText id="ime" value="#{osoba.ime}"/>
                <h:outputLabel for="prezime" value="Prezime"/>
                <h:inputText id="prezime" value="#{osoba.prezime}"/>
                <h:outputLabel for="pol" value="Pol"/>
                <h:inputText id="pol" value="#{osoba.pol}"/>
                <h:outputLabel for="starost" value="Starost"/>
                <h:inputText id="starost" value="#{osoba.starost}"/>
                <h:panelGroup/>
                <h:commandButton value="Kreiraj JSON"
                    action="#{jsonPModelApiZrno.generateJson()}" />
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

Sledi listing stranice prikazi\_popunjen\_obj.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Generisan JSON pomoću JSON-P API objektnog modela</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="parsiranJson" value="Parsiran JSON"/>
                <h:inputTextarea
                    value="#{jsonPModelApiZrno.jsonStr}" rows="4"/>
                <h:panelGroup/>
                <h:commandButton value="Potvrđi" />
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

```

        action="#{jsonPModelApiZrno.parseJson()}" />
    </h:panelGrid>
</h:form>
</h:body>
</html>

```

Sledi listing stranice generisan\_json.xhtml.

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Osobine Java objekta popunjenog iz JSON</title>
    </h:head>
    <h:body>
        <table>
            <tr>
                <td>
                    Ime:
                </td>
                <td>
                    #{osoba.ime}
                </td>
            </tr>
            <tr>
                <td>
                    Prezime:
                </td>
                <td>
                    #{osoba.prezime}
                </td>
            </tr>
            <tr>
                <td>
                    Pol:
                </td>
                <td>
                    #{osoba.pol}
                </td>
            </tr>
            <tr>
                <td>
                    Starost:
                </td>
                <td>
                    #{osoba.starost}
                </td>
            </tr>
        </table>
    </h:body>
</html>

```

## CDI ZRNO I KONTOLER

*Slede Java klase primera.*

Sledećim listingom je priložena klasa CDI zrna iz čijih osobina se generiše JSON String:

```
package com.metropolitan.jsonmodelapi;
import java.io.Serializable;
import javax.enterprise.context.SessionScoped;
import javax.inject.Named;

/**
 *
 * @author Vladimir Milicevic
 */
@Named
@SessionScoped
public class Osoba implements Serializable{
    private String ime;
    private String prezime;
    private String pol;
    private Integer starost;

    public String getIme() {
        return ime;
    }

    public void setIme(String ime) {
        this.ime = ime;
    }

    public String getPrezime() {
        return prezime;
    }

    public void setPrezime(String prezime) {
        this.prezime = prezime;
    }

    public String getPol() {
        return pol;
    }

    public void setPol(String pol) {
        this.pol = pol;
    }

    public Integer getStarost() {
        return starost;
    }
}
```

```
    public void setStarost(Integer starost) {  
        this.starost = starost;  
    }  
  
}
```

Konačno sledi klasa koja implementira metode `generateJson()` i `parseJson()` primenom JSON - P API Objektnog modela:

```
package com.metropolitan.jsonpmodelapi;  
  
import java.io.StringReader;  
import java.io.StringWriter;  
import javax.enterprise.context.RequestScoped;  
import javax.inject.Inject;  
import javax.inject.Named;  
import javax.json.Json;  
import javax.json.JsonObject;  
import javax.json.JsonObjectBuilder;  
import javax.json.JsonReader;  
import javax.json.JsonWriter;  
/**  
 *  
 * @author Vladimir Milicevic  
 */  
@Named  
@RequestScoped  
public class JsonPModelApiZrno {  
    @Inject  
    private Osoba osoba;  
    private String jsonStr;  
  
    public String generateJson() {  
        JsonObjectBuilder jsonObjectBuilder = Json.createObjectBuilder();  
  
        JsonObject jsonObject = jsonObjectBuilder.  
            add("ime", osoba.getIme()).  
            add("prezime", osoba.getPrezime()).  
            add("pol", osoba.getPol()).  
            add("starost", osoba.getStarost()).  
            build();  
  
        StringWriter stringWriter = new StringWriter();  
  
        try (JsonWriter jsonWriter = Json.createWriter(stringWriter)) {  
            jsonWriter.writeObject(jsonObject);  
        }  
        setJsonStr(stringWriter.toString());  
  
        return "prikazi_popunjen_obj";  
    }  
  
    public String parseJson() {
```

```
JsonObject jsonObject;

try (JsonReader jsonReader = Json.createReader(new StringReader(jsonStr))) {
    jsonObject = jsonReader.readObject();
}

osoba.setIme(jsonObject.getString("ime"));
osoba.setPrezime(jsonObject.getString("prezime"));
osoba.setPol(jsonObject.getString("pol"));
osoba.setStarost(jsonObject.getInt("starost"));

return "generisan_json";
}

public String getJsonStr() {
    return jsonStr;
}

public void setJsonStr(String jsonStr) {
    this.jsonStr = jsonStr;
}
}
```

## ▼ Poglavlje 5

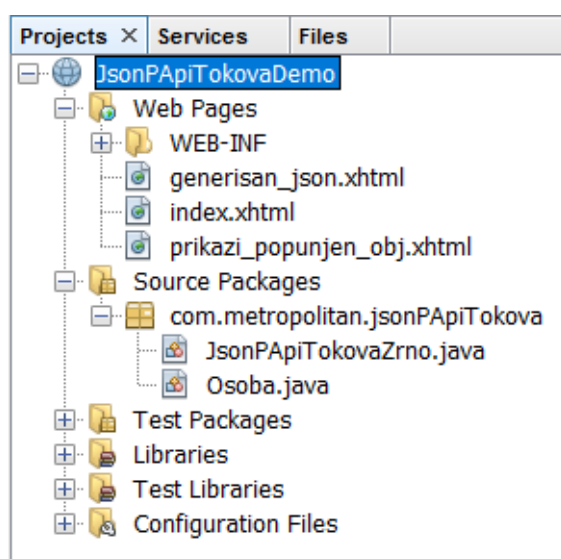
# Java API za JSON procesiranje - pokazni primer 2

## JSON-P API TOKOVA (25 MIN)

*Vežbanje na računar u prethodnim zadacima iz problematike JSON-P API tokova.*

Primenom Java EE 7 platforme i odgovarajućeg razvojnog okruženja (na primer NetBEans IDE) kreirati Java veb aplikaciju sa sledećim zahtevima:

1. Aplikacija ima mogućnost kreiranja i parsiranja JSON objekata primenom JSON-P API tokova;
2. Osnova za kreiranje JSON dokumenta je odgovarajuća Java klasa (CDI zrno);
3. Kreirati odgovarajući kontroler za navedeno CDI zrno;
4. Aplikacija ima sposobnost da svaku promenu u generisanom JSON Stringu prenese u izvorno Java CDI zrno;
5. Kreirati odgovarajuće JSF stranice za prikupljanje podataka kojima će biti popunjeno CDI zrno, a potom i JSON String, za prikazivanje generisanog JSON Stringa i za prikazivanje parsiranih podataka iz JSON Stringa u Java CDI zrno.
6. Aplikacija može da ima strukturu priloženu sledećom slikom:



Slika 5.1 Predložena struktura veb aplikacije rešenja [izvor: autor]

## JSF STRANICE

*Prvo će biti priložene JSF stranice projekta koji koristi JSON-P API tokova..*

Sledi listing stranice index.xml.

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Object to JSON With the JSON-P Object Model API</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="ime" value="Ime"/>
                <h:inputText id="ime" value="#{osoba.ime}"/>
                <h:outputLabel for="prezime" value="Prezime"/>
                <h:inputText id="prezime" value="#{osoba.prezime}"/>
                <h:outputLabel for="pol" value="Pol"/>
                <h:inputText id="pol" value="#{osoba.pol}"/>
                <h:outputLabel for="starost" value="Starost"/>
                <h:inputText id="starost" value="#{osoba.starost}"/>
                <h:panelGroup/>
                <h:commandButton value="Kreiraj JSON"
                    action="#{jsonPApiTokovaZrno.generateJson()}" />
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

Sledi listing stranice prikazi\_popunjen\_obj.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Generisan JSON pomoću JSON-P API objektnog modela</title>
    </h:head>
    <h:body>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel for="parsiranJson" value="Parsiran JSON"/>
                <h:inputTextarea
                    value="#{jsonPApiTokovaZrno.jsonStr}" rows="4"/>
            </h:panelGroup>
        </h:form>
    </h:body>
</html>
```



```

        <h:commandButton value="Potvrdi"
                        action="#{jsonPApiTokovaZrno.parseJson()}" />
    </h:panelGrid>
</h:form>
</h:body>
</html>

```

Sledi listing stranice generisan\_json.xhtml.

```

<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/
TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
  <h:head>
    <title>Osobine Java objekta popunjenog iz JSON</title>
  </h:head>
  <h:body>
    <table>
      <tr>
        <td>
          Ime:
        </td>
        <td>
          #{osoba.ime}
        </td>
      </tr>
      <tr>
        <td>
          Prezime:
        </td>
        <td>
          #{osoba.prezime}
        </td>
      </tr>
      <tr>
        <td>
          Pol:
        </td>
        <td>
          #{osoba.pol}
        </td>
      </tr>
      <tr>
        <td>
          Starost:
        </td>
        <td>
          #{osoba.starost}
        </td>
      </tr>
    </table>

```

```
</h:body>  
</html>
```

## CDI ZRNO I KONTOLER

*Slede Java klase primera koji koristi JSON - P API tokova.*

Sledećim listingom je priložena klasa CDI zrna iz čijih osobina se generiše JSON String:

```
package com.metropolitan.jsonPApiTokova;  
import java.io.Serializable;  
import javax.enterprise.context.SessionScoped;  
import javax.inject.Named;  
  
/**  
 *  
 * @author Vladimir Milicevic  
 */  
@Named  
@SessionScoped  
public class Osoba implements Serializable{  
    private String ime;  
    private String prezime;  
    private String pol;  
    private Integer starost;  
  
    public String getIme() {  
        return ime;  
    }  
  
    public void setIme(String ime) {  
        this.ime = ime;  
    }  
  
    public String getPrezime() {  
        return prezime;  
    }  
  
    public void setPrezime(String prezime) {  
        this.prezime = prezime;  
    }  
  
    public String getPol() {  
        return pol;  
    }  
  
    public void setPol(String pol) {  
        this.pol = pol;  
    }  
}
```

```
public Integer getStarost() {  
    return starost;  
}  
  
public void setStarost(Integer starost) {  
    this.starost = starost;  
}  
  
}
```

Konačno sledi klasa koja implementira metode `generateJson()` i `parseJson()` primenom JSON - P API tokova:

```
package com.metropolitan.jsonPApiTokova;  
import java.io.StringReader;  
import java.io.StringWriter;  
import java.util.HashMap;  
import java.util.Map;  
import javax.enterprise.context.RequestScoped;  
import javax.inject.Inject;  
import javax.inject.Named;  
import javax.json.Json;  
import javax.json.stream.JsonGenerator;  
import javax.json.stream.JsonParser;  
import javax.json.stream.JsonParser.Event;  
  
/**  
 *  
 * @author Vladimir Milicevic  
 */  
@Named  
@RequestScoped  
public class JsonPApiTokovaZrno {  
    @Inject  
    private Osoba osoba;  
    private String jsonStr;  
  
    public String generateJson() {  
        StringWriter stringWriter = new StringWriter();  
        try (JsonGenerator jsonGenerator  
            = Json.createGenerator(stringWriter)) {  
            jsonGenerator.writeStartObject().  
                write("ime", osoba.getIme()).  
                write("prezime", osoba.getPrezime()).  
                write("pol", osoba.getPol()).  
                write("starost", osoba.getStarost()).  
                writeEnd();  
        }  
  
        setJsonStr(stringWriter.toString());  
        return "prikazi_popunjen_obj";  
    }  
}
```

```
public String parseJson() {
    StringReader stringReader = new StringReader(jsonStr);

    JsonParser jsonParser = Json.createParser(stringReader);

    Map<String, Object> jsonMap = new HashMap<>();
    String jsonKeyNm = null;
    Object jsonVal = null;

    while (jsonParser.hasNext()) {
        JsonParser.Event event = jsonParser.next();

        if (event.equals(Event.KEY_NAME)) {
            jsonKeyNm = jsonParser.getString();
        } else if (event.equals(Event.VALUE_STRING)) {
            jsonVal = jsonParser.getString();
        } else if (event.equals(Event.VALUE_NUMBER)) {
            jsonVal = jsonParser.getInt();
        }

        jsonMap.put(jsonKeyNm, jsonVal);
    }

    osoba.setIme((String) jsonMap.get("ime"));
    osoba.setPrezime((String) jsonMap.get("prezime"));
    osoba.setPol((String) jsonMap.get("pol"));
    osoba.setStarost((Integer) jsonMap.get("starost"));

    return "generisan_json";
}

public String getJsonStr() {
    return jsonStr;
}

public void setJsonStr(String jsonStr) {
    this.jsonStr = jsonStr;
}
}
```

## ▼ Poglavlje 6

# Individualna vežba 11

## INDIVIDUALNA VEŽBA (135 MIN)

*Pokušajte sami.*

1. Kreirajte Java veb aplikaciju koja će integrisati primere urađene u pokaznoj vežbi u jedinstvenu aplikaciju;
2. **Primenite u aplikaciji oba obrađena pristupa;**
3. Na index.xhtml stranici dajte mogućnost izbora između obrađenih pristupa JSON procesiranja;
4. Srediti JSF stranice primenom neke poznate JSF biblioteke komponenata;

NAPOMENA: Korisnik na početnoj stranici bira jedan od dva obrađena načina manipulacije JSON stringovima. U zavisnosti od izbora navigacija veb aplikacije ide ka stranicama i klasama za primenu objektnog modela ili modela tokova.

## ▼ Poglavlje 7

# Domaći zadatak 11

## ZADATAK 1

### *Uradite domaći zadatak.*

Primenom Java EE 7 platforme i odgovarajućeg razvojnog okruženja (na primer NetBEans IDE) kreirati Java veb aplikaciju sa sledećim zahtevima:

1. Aplikacija ima mogućnost kreiranja i parsiranja JSON objekata primenom JSON-P API objektnog modela i JSON-P API tokova;
2. Na početnoj stranici se bira kojim ćete pristupom kreirati i parsirati JSON String;
3. Osnova za kreiranje JSON dokumenta je odgovarajuća Java klasa (CDI zrno);
4. CDI zrno kreirati tako da mu osobine odgovaraju strukturi JSON Stringa sa slike;
5. Kreirati odgovarajući kontroler za navedeno CDI zrno;
6. Kontroler implementira metode generateJson() i parseJson() po analogiji sa primerima urađenim na vežbama i predavanjima;
7. Aplikacija ima sposobnost da svaku promenu u generisanom JSON Stringu prenese u izvorno Java CDI zrno;
8. Kreirati odgovarajuće JSF stranice za prikupljanje podataka kojima će biti popunjeno CDI zrno, a potom i JSON String, za prikazivanje generisanog JSON Stringa i za prikazivanje parsiranih podataka iz JSON Stringa u Java CDI zrno.
9. Na stranici za unos podataka kreirati formu koja će preuzeti podatke koji odgovaraju osobinama CDI zrna i strukturi JSON Stringa sa slike;
10. Struktura JSON Stringa je priložena sledećom slikom:

```
{
    "broj indeksa":123,
    "ime":"Pera Peric",
    "status":"student",
    "tradicionalni":false
    "adresa":{
        "ulica":"Neznanog junaka 1",
        "grad":"Beograd",
        "postanskiBroj":11000
    },
    "brojeviTelefona":[9988664422, 1234567890],
    "uloga":"programer"
}
```

Slika 7.1 Struktura JSON Stringa

Nakon urađenog obaveznog zadatka studenti na email dobijaju različite zadatke od strane predmetnog asistenta.

## ▼ Poglavlje 8

# Zaključak

## ZAKLJUČAK

*Lekcija se bavila mehanizmima JSON procesiranja u distribuiranim Java EE aplikacijama.*

U ovoj lekciji, obavljena je detaljna analiza i diskusija u vezi sa mehanizmima *JSON* procesiranja u *distribuiranim Java EE veb aplikacijama*. Iako je većini studenata je verovatno bio poznat *JSON* pojam, za one koji nisu bili upućeni u njega, napravljen je adekvatan uvod u problematiku. Posebno je istaknuto da *JSON* (skraćenica za *JavaScript Object Notation*) predstavlja lak (*lightweight*) format razmene poruka. Takođe je istaknuto da vremenom ovaj format je stekao veliku popularnost i široku primenu. Posebno je bilo potrebno istaći njegove prednosti u odnosu na *XML* baziranu razmenu podataka koja se ogleda u sledećem:

- *JSON je lakši za upotrebu programerima jer se lakše čita i piše;*
- *JSON je lakši za upotrebu računarima jer se lakše generiše i parsira.*

Posebno je bilo važno napomenuti da je *JSON* vremenom stekao široku primenu u brojnom savremenim veb aplikacijama.

U nastavku, lekcija se fokusirala *Java EE 7* budući da ona uvodi *Java API* za *JSON procesiranje (JSON-P)* koji je postao standardan *Java EE API* za generisanje i parsiranje *JSON* podataka. Dalje je istaknuto da *JSON-P* obezbeđuje dva načina za parsiranje i generisanje *JSON* podataka. To su: API objektnog modela (*the object model API*) i API tokova (*the streaming API*).

Na osnovu navedenog lekcija je poseban akcenat stavila na sledeće teme:

- *JSON-P API objektnog modela* - Poseban akcenat je bio na analizi, diskusiji i demonstraciji generisanja i parsiranja *JSON* podataka pomoću *JSON-P API objektnog modela*;
- *JSON-P API tokova* - Poseban akcenat je bio na analizi, diskusiji i demonstraciji generisanja i parsiranja *JSON podataka* pomoću *JSON-P API modela tokova*;

Savladavanjem ove lekcije, student će u potpunosti moći da razume i koristi *Java EE* mehanizme za *JSON procesiranje*. Takođe, student je ovladao dodatnim naprednim alatom za razvoj distribuiranih veb sistema.

Lekcija 11 je pokrila problematiku koja se odnosi na *JSON - P* procesiranje koja predstavlja potpuno nov dodatak za *Java EE* specifikaciju. Posebno je pokazano kako se generišu i parsiraju podaci primenom jednostavnog *JSON - P* API objektnog modela, a zatim je



pažnja prebačena na pristup sa boljim performansama i boljim iskošrišćenjem memorije, ali bez mogućnosti direktnog pristupa JSON podacima, JSON - P API tokova. Takođe, je pokazano kako se generišu i parsiraju podaci primenom JSON - P API tokova.

## LITERATURA

*Za pripremu lekcije korišćena je najnovija literatura.*

1. Eric Jendrock, Ricardo Cervera-Navarro, Ian Evans, Kim Haase, William Markito. 2014. Java Platform, Enterprise Edition The Java EE Tutorial, Release 7, ORACLE
2. David R. Heffelfinger. 2015. Java EE7 Development With NetBeans 8, PACK Publishing
3. Yakov Fain. 2015. Java 8 programiranje, Kombib (Wiley)
4. Josh JUnenau. 2015. Java EE7 Recipes, Apress