Fakultet tehničkih nauka, DRA, Novi Sad

Predmet:
Organizacija podataka

Dr Slavica Kordić,
Milan Čeliković,
Vladimir Dimitrieski,
Nemanja Igić

JavaScript Object Notation
JSON

JSON

- JavaScript Object Notation (JSON)
  - format za razmenu i čuvanje podataka
    - nezavisan od programskih jezika
  - minimalna količina dodatnih informacija
    - *minimum overhead*
  - laka računarska obrada podataka u JSON formatu
  - ljudi mogu lako čitati podatke sačuvane u ovom formatu

JSON

- Zasnovan na podskupu JavaScript programskog jezika
  - Standard ECMA-262 3rd Edition – decembar 1999.
- JSON specifikacija
  - http://www.json.org/
  - spisak biblioteka za veliki broj programskih jezika
- JSON validator
  - http://jsonlint.com/

JSON

- Upotreba
  - komunikacija na Internetu
    - web servisi (REST …)
    - JSON RPC
    - podaci sa socijalnih mreža
      - *Facebook, Twitter, LinkedIn* ...
  - NoSQL baze podataka
  - konfiguracioni fajlovi

JSON

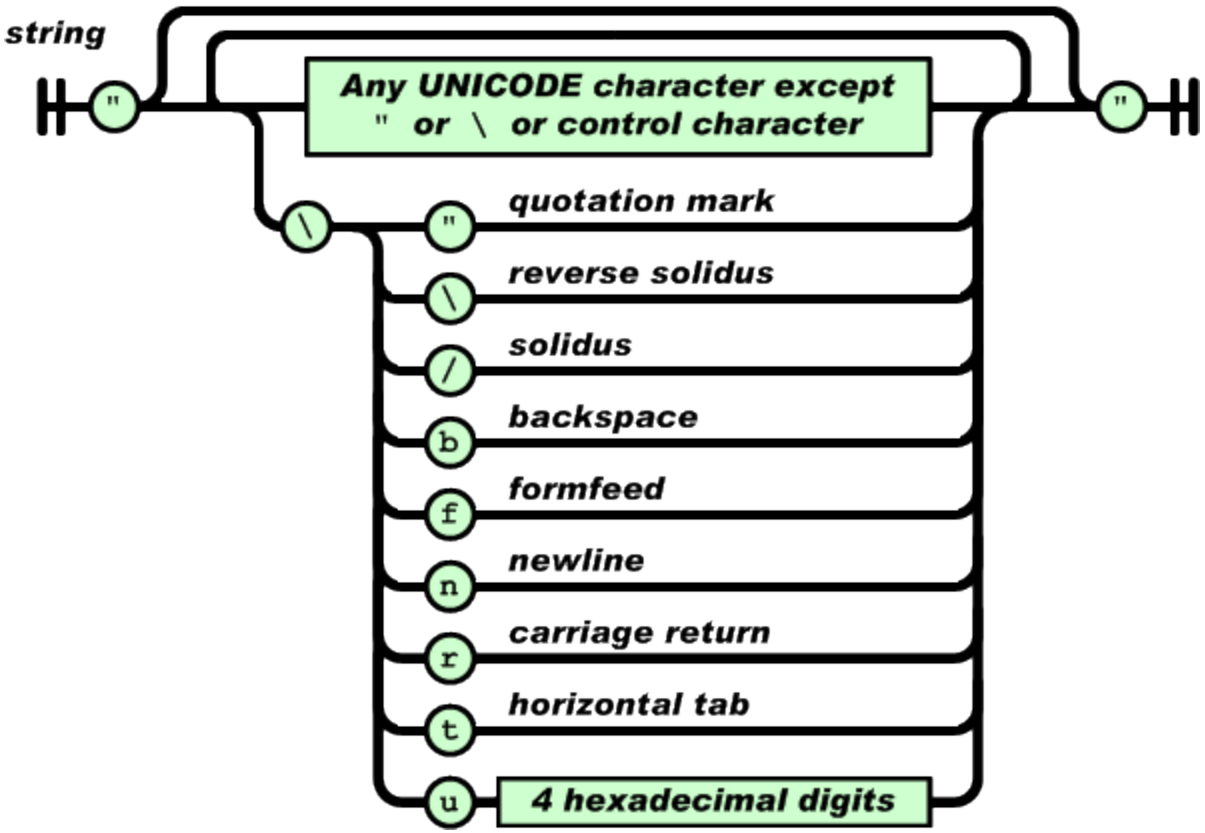
- Sintaksa
  - osnovni element
    - par naziv/vrednost
  - JSON je sagrađen na dve strukture
    - objekat
    - niz

JSON

- Par naziv/vrednost
  - *naziv : vrednost*
  - *naziv*
    - naziv atributa
      - „opis“ vrednosti koja sledi
    - **uvek je tipa String**
  - *vrednost*
    - vrednost atributa
    - **može biti String, broj, *true, false, null*, objekat ili kolekcija**

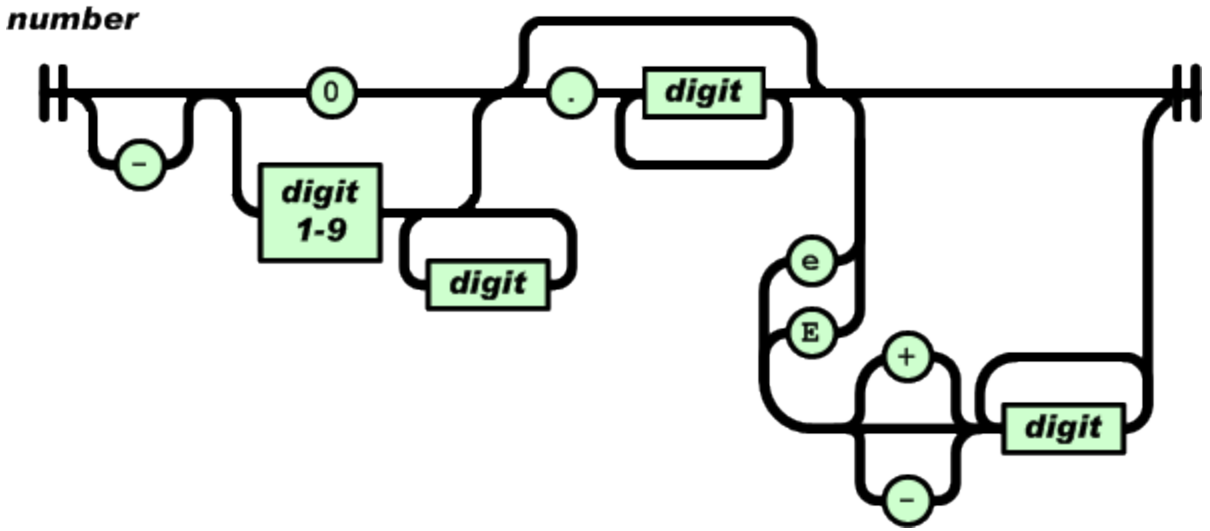
JSON

- String


string
"
Any UNICODE character except
" or \ or control character
"
\
"
quotation mark
\
reverse solidus
/
solidus
b
backspace
f
formfeed
n
newline
r
carriage return
t
horizontal tab
u
4 hexadecimal digits

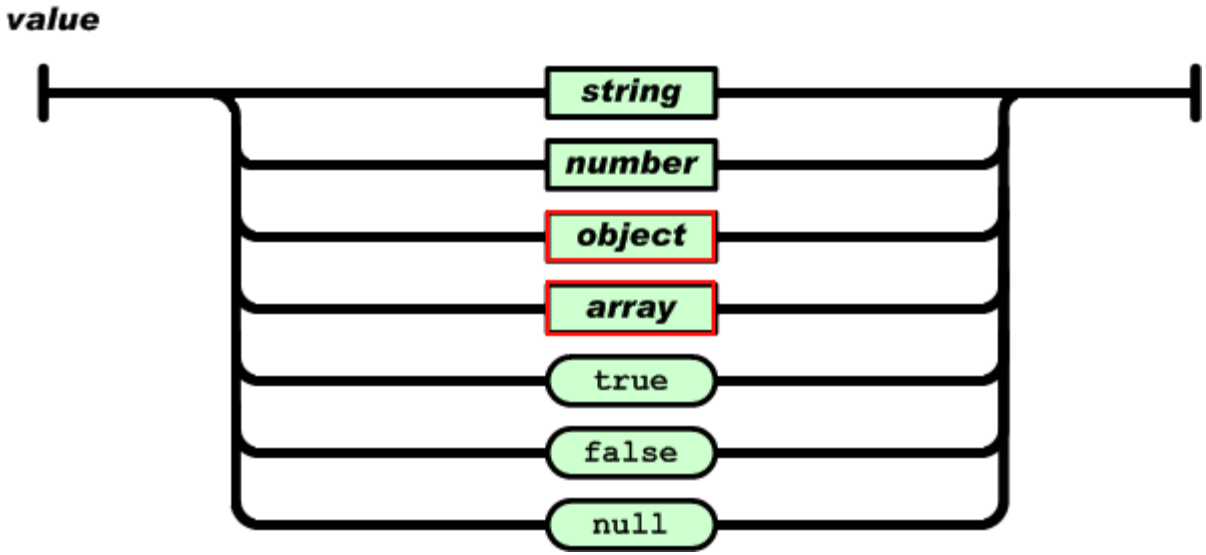
JSON

- Broj



JSON

- Vrednost



JSON

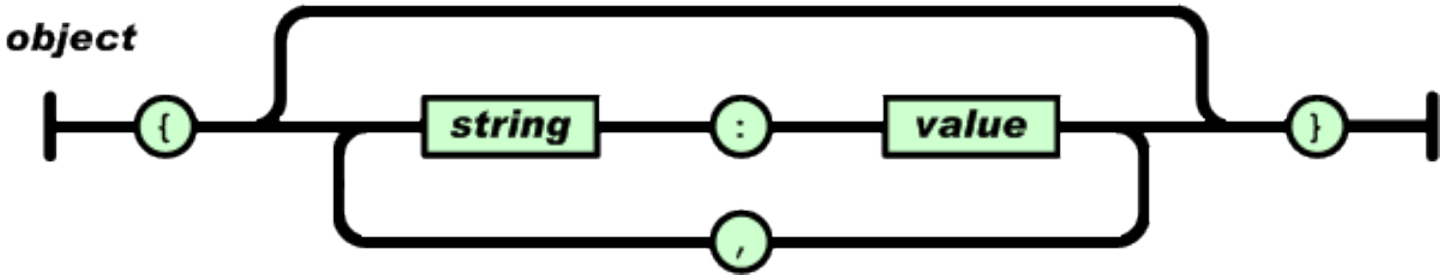
- Par naziv/vrednost
  - osnovni primeri:
    - “NazivKnjizare” : “Moja knjizara”
    - “BrojKnjiga” : 24
    - “BrojKnjiga” : “24”
    - “ClanLancaKnjizara” : true
    - “Direktor” : null

JSON

- Objekat
  - **neuređeni skup parova naziv/vrednost**
    - međusobno razdvojeni znakom “,”
  - počinje sa znakom “{”
  - završava se znakom “}”
  - može biti vrednost u paru naziv/vrednost
    - moguće ugneždavanje objekata
    - imenovani objekat
- JSON objekat (datoteka)
  - neimenovani objekat

JSON

- Objekat



JSON

- Objekat primer

objekat JSON
objekat Book

objekat Author

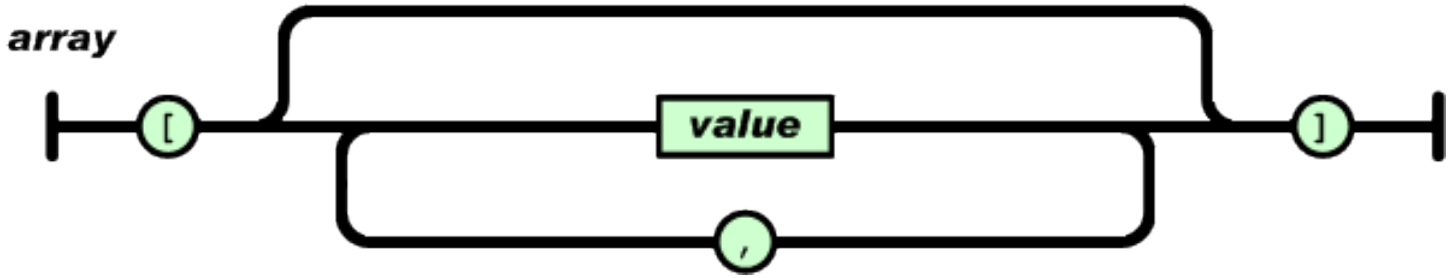
```
{
    "Book": {
        "ISBN": "ISBN-0-13-713526-2",
        "Price": 85,
        "Edition": 3,
        "Title": "A First Course in Database Systems",
        "Author": {
            "First_Name": "Jeffrey",
            "Last_Name": "Ullman"
        }
    }
}
```

JSON

- Niz
  - **uređeni skup vrednost**
    - međusobno razdvojeni znakom “,”
  - počinje sa znakom “[”
  - završava se znakom “]”
  - može biti vrednost u paru naziv/vrednost
    - imenovani niz
    - moguće ugneždavanje nizova

JSON

- Niz



JSON

- Niz primer

niz Books

niz Authors

niz Authors

```
{
"Books":  [
    { "ISBN":"ISBN-0-13-713526-2",
      "Price":85,
      "Edition":3,
      "Title":"A First Course in Database Systems",
      "Authors":[ {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
    ,
    { "ISBN":"ISBN-0-13-815504-6",
      "Price":100,
      "Remark":"Buy this book bundled with 'A First Course'",
      "Title":"Database Systems:The Complete Book",
      "Authors":[ {"First_Name":"Hector", "Last_Name":"Garcia-Molina"},
                  {"First_Name":"Jeffrey", "Last_Name":"Ullman"},
                  {"First_Name":"Jennifer", "Last_Name":"Widom"} ] }
  ]
}
```

JSON, Eclipse IDE i Java

- Eclipse JSON tools
  - *syntax highlighting* za JSON datoteke
  - preuzeti sa Eclipse *marketplace*-a
- Jackson (v 2.4.3)
  - biblioteka za rad sa formatom podataka JSON u programskom jeziku Java
  - http://wiki.fasterxml.com/JacksonHome

Jackson

- Tri metode za obradu JSON-a
  - **direktno mapiranje JSON-a na Java objekte**
    - *data binding*
    - *Plain Old Java Object* (POJO)
    - pristup koji je najjednostavniji za korišćenje
  - **inkrementalno parsiranje/generisanje**
    - *streaming API*
    - čita i piše JSON uz pomoć diskretnih događaja
      - za svaki element JSON-a se generiše događaj koji treba obraditi
    - pristup sa najboljim performansama

Jackson

- Tri metode za obradu JSON-a
  - **mapiranje JSON-a na strukturu tipa stabla**
    - *tree model*
    - struktura u radnoj memoriji u koju se smeštaju isprasirani podaci
    - najfleksibilniji pristup

Primer 1

- Napisati *Java* program koji:
  - mapira sadržaj datoteke *book.json* na odgovarajuće *Java* objekte
  - ažurira objekte sa novim podacima
  - sačuvava izmene u datoteku *Book_changed.json*
- Zadatak uraditi koristeći:
  - Jackson biblioteku
  - **direktno mapiranje JSON-a na Java objekte**

Primer 1 – *Book.json*

```
{
    "ISBN": "ISBN-0-13-713526-2",
    "Price": 85,
    "Edition": 3,
    "Title": "A First Course in Database Systems",
    "Author": {
        "First_Name": "Jeffrey",
        "Last_Name": "Ullman"
    }
}
```

Primer 1 – *POJOs*

```java
public class Book {

    private String ISBN;
    private int Price;
    private int Edition;
    private String Title;
    private Author Author;

    @JsonProperty("ISBN")
    public void setISBN(String iSBN) {
        ISBN = iSBN;
    }
    public String getISBN() {
        return ISBN;
    }
    @JsonProperty("Price")
    public void setPrice(int price) {
        Price = price;
    }
    public int getPrice() {...}
    public void setEdition(int edition) {...}
    public void setTitle(String title) {...}
    public void setAuthor(Author author) {...}
    public int getEdition() {...}
    public String getTitle() {...}
    public Author getAuthor() {...}
    public String toString() {...}
}
```

```java
@JsonAutoDetect(fieldVisibility = Visibility.ANY)
public class Author {
    private String First_Name;
    private String Last_Name;

    public String toString() {...}
}
```

Primer 1 - *JSONMapper*

```
10 public class JSONMapper {
11     public static void main(String[] args) throws JsonParseException, JsonMappingException, IOException {
12         /*****************************/
13         /**Convert JSON to Java object*/
14         /*****************************/
15         ObjectMapper mapper = new ObjectMapper();
16 
17         //configure the Jackson parser to see private fields
18         //this global configuration and it is alternative to the @JsonAutoDetect annotation
19         //mapper.setVisibilityChecker(mapper.getSerializationConfig().getDefaultVisibilityChecker()
20         //      .withFieldVisibility(Visibility.ANY));
21 
22         //deserialize a book from Book.json
23         Book book = mapper.readValue(new File("Book.json"), Book.class);
24 
25         //print loaded object to the console
26         System.out.println(book.toString());
27 
28         /*****************************/
29         /**Convert Java object to JSON*/
30         /*****************************/
31         //change a property of the book
32         book.setTitle("Changed book title");
33 
34         //serialize the book to Book_changed.json
35         mapper.writeValue(new File("Book_changed.json"), book);
36     }
37 }
```

Primer 1 - *JSONSimpleMapper*

```
11 public class JSONSimpleMapper {
12     public static void main(String[] args) throws JsonParseException, JsonMappingException, IOException {
13         /******************************/
14         /**Convert JSON to Java object*/
15         /******************************/
16         ObjectMapper mapper = new ObjectMapper();
17 
18         //deserialize a hash map from Book.json
19         //no need for POJOs
20         Map<String, Object> book = mapper.readValue(new File("Book.json"), Map.class);
21 
22         //print loaded hash map to the console
23         //Authors are also deserialized as a hash map inside book hash map
24         System.out.println(book);
25 
26         /******************************/
27         /**Convert Java object to JSON*/
28         /******************************/
29         //add a property to the hash map
30         book.put("newAttribute", "newValue");
31 
32         //serialize the book to Book_changed.json
33         mapper.writeValue(new File("Book_changed.json"), book);
34     }
35 }
```

Primer 2

- Napisati *Java* program koji:
  - generiše novi JSON sa podacima o autoru i njegovoj knjizi
    - generiše datoteku *Book_generated.json*
  - učitava autora i naslov njegove knjige
- Zadatak uraditi koristeći:
  - Jackson biblioteku
  - **inkrementalno parsiranje/generisanje**

Primer 2 – *POJO*

```
 6  @JsonAutoDetect(fieldVisibility = Visibility.ANY)
 7  public class AuthorsBook {
 8      private String firstName;
 9      private String lastName;
10      private String bookTitle;
11
12      public String getFirstName() {..
15      public void setFirstName(String firstName) {..
18      public String getLastName() {..
21      public void setLastName(String lastName) {..
24      public String getBookTitle() {..
27      public void setBookTitle(String bookTitle) {..
31      public String toString() {..
35  }
```

Primer 2 – *generateJSON()*

```
56    private static void generateJSON() throws JsonParseException, IOException {
57        JsonFactory f = new JsonFactory();
58        JsonGenerator g = f.createGenerator(new File("Book_generated.json"), JsonEncoding.UTF8);
59
60        g.writeStartObject();
61        g.writeObjectFieldStart("Author");
62        g.writeStringField("First_Name", "Stephen");
63        g.writeStringField("Last_Name", "King");
64        g.writeEndObject(); // for field 'Author'
65        g.writeStringField("Title", "The Green Mile");
66        g.writeEndObject();
67        //it is important to close the generator
68        //it will force flushing of output, close underlying output stream
69        g.close();
70    }
```

Primer 2 – *parseJSON()*

```
27    private static void parseJSON(AuthorsBook authorBooks, String bookName) throws JsonParseException, IOException {
28        JsonFactory f = new JsonFactory();
29        JsonParser jp = f.createParser(new File(bookName));
30        //the following line will return JsonToken.START_OBJECT (first curly bracket)
31        jp.nextToken();
32        //parse JSON file for needed tokens
33        while (jp.nextToken() != JsonToken.END_OBJECT) {
34            String fieldname = jp.getCurrentName();
35            jp.nextToken(); // move to value, or START_OBJECT/START_ARRAY
36            if ("Author".equals(fieldname)) { //field value contains an object
37                //parse this object in the same way the whole JSON is parsed
38                while (jp.nextToken() != JsonToken.END_OBJECT) {
39                    String namefield = jp.getCurrentName();
40                    jp.nextToken(); // move to value
41                    if ("First_Name".equals(namefield)) {
42                        authorBooks.setFirstName(jp.getText());
43                    } else if ("Last_Name".equals(namefield)) {
44                        authorBooks.setLastName(jp.getText());
45                    }
46                }
47            } else if ("Title".equals(fieldname)) {
48                authorBooks.setBookTitle(jp.getText());
49            } else {
50                //throw new IllegalStateException("Unrecognized field '" + fieldname + "'!");
51            }
52        }
53        jp.close(); // ensure resources get cleaned up timely and properly
54    }
```

Primer 2 – *JSONStreamer*

```
15 public class JSONStreamer {
16     public static void main(String[] args) throws JsonParseException, IOException {
17         //POJO for holding data
18         AuthorsBook ab = new AuthorsBook();
19         generateJSON();
20         parseJSON(ab, "Book_generated.json");
21         System.out.println(ab);
22         parseJSON(ab, "Book.json");
23         System.out.println(ab);
24 
25     }
26 
27     private static void parseJSON(AuthorsBook authorBooks, String bookName) throws JsonParseException, IOException {
55     private static void generateJSON() throws JsonParseException, IOException {
70 }
```

Primer 3

- Napisati *Java* program koji:
  - učitava autora i naslov njegove knjige
  - ažurira naslov knjige
  - sačuvava izmene u datoteci *Book_changed.json*
- Zadatak uraditi koristeći:
  - Jackson biblioteku
  - **mapiranje JSON-a na strukturu tipa stabla**

Primer 3 - *JSONTreeStream*

```
13 public class JSONTreeStream {
14     public static void main(String[] args) throws JsonProcessingException, IOException {
15         ObjectMapper m = new ObjectMapper();
16         AuthorsBook authorsBook = new AuthorsBook();
17 
18         //create root JSON node from a file
19         JsonNode rootNode = m.readTree(new File("Book.json"));
20 
21         //find required attributes and read their values
22         authorsBook.setBookTitle(rootNode.path("Title").textValue());
23 
24         JsonNode nameNode = rootNode.path("Author");
25         authorsBook.setFirstName(nameNode.path("First_Name").textValue());
26         authorsBook.setLastName(nameNode.path("Last_Name").textValue());
27 
28         //print author info to the console
29         System.out.println(authorsBook);
30 
31         //change title of the book in the tree model
32         ((ObjectNode)rootNode).put("Title", "ChangedBookTitle");
33         m.writeValue(new File("Book_changed.json"), rootNode);
34     }
35 }
```

Zadatak 1

- Napisati *Java* program koji iz JSON datoteke *bookstore.json* učitava sve knjige i časopise i ispisuje ih na standardni izlaz.

Zadatak 2

- Napisati Java program koji:
  - učitava sve podatke iz CSV datoteke *counties_cities.csv*
  - sortira države po kontinentu kojem pripadaju
  - za svaki kontinent snima po jednu JSON datoteku koja sadrži sve podatke o državama koje se nalaze na tom kontinentu

Zadatak 3

- Napisati Java program koji vrši pretragu po JSON datoteci sa podacima preuzetim sa *Twitter*-a. Program treba da omogući:
  - čuvanje id-a korisnika i teksta njegovog *tweet*-a
  - čuvanje ukupnog osećanja nekog *tweet*-a
  - čuvanje ukupnog osećanja datog korisnika na osnovu reči u svim njegovim *tweet*-ovima
  - prikaz odnosa pozitivnih i negativnig *tweet*-ova

Zadatak 3

- Date su dve datoteke sa *tweet*-ovima
  - output_1.txt
    - *tweet*-ovi su sakupljani 1 minut
    - za inicijalnu analizu podataka
  - output_10.txt
    - *tweet*-ovi su sakupljani 10 minuta
    - za detaljniju analizu podataka
- Značenje JSON atributa iz datih datoteka
  - https://dev.twitter.com/overview/api/tweets

Zadatak 3

- Data je datoteka sa engleskim rečima i osećanjem koju svaka reč nosi
  - *AFIN-111.txt*
    - svaka reč i njeno osećanje su u posebnom redu
    - reč i jačina osećanja su razdvojeni tabom
    - jačina osećanja je predstavljena celim brojem u intervalu od -5 do 5
      - -5 veoma negativno osećanje
      - 5 veoma pozitivno osećanje
    - preuzeto sa http://www2.imm.dtu.dk/pubdb/views/publication_details.php?id=6010

Zadatak 4

- Napraviti Java program koji za bilo koju ulaznu JSON datoteku proverava da li je sintaksa ispravna (da li je datoteka dobro formirana).

JSON Schema

- Opisuje strukturu JSON dokumenata
- Takođe je JSON dokument
  - mogu se koristiti isti alati za učitavanje ovog dokumenta
- Upotreba
  - dokumentacija
  - automatizacija rada sa JSON datotekama
  - generisanje koda

JSON Schema

- JSON Schema specifikacija
  - http://json-schema.org/latest/json-schema-core.html
- JSON Schema validator
  - http://jsonschemalint.com/
- Primer
  - schema u datotece *Book2Schema.json*
  - podaci u datoteci *Book2.json*

JSON Schema

- *Book2.json*

```
{
    "ISBN": "ISBN-0-13-713526-2",
    "Price": 85,
    "Edition": 3,
    "Title": "A First Course in Database Systems",
    "Author": {
        "First_Name": "Jeffrey",
        "Last_Name": "Ullman"
    },
    "tags": ["Databases", "Data", "Organization"]
}
```

JSON Schema

- Korenski element je objekat i sadrži informacije o samoj schemi
- *metadata* ključne reči
  - *“title“*
    - naziv elementa
  - *“description“*
    - opis elementa

JSON Schema

- *“type“*
  - tip podataka JSON elementa
  - dozvoljene vrednosti
    - array, boolean, integer, number, null, object, string
  - **za korenski element mora biti *object***

JSON Schema

- *“properties“*
  - specifikacija atributa nekog objekta
  - sam po sebi JSON objekat
  - svaka specifikacija objekta u schemi mora imati *properties* objekat
  - specifikacija parova naziv/vrednost

JSON Schema

```json
{
    "title": "Book schema",
    "description": "A schema for the book JSON object from the Data organization course",
    "type": "object",
    "properties": {
        "ISBN": {
            "description": "The unique identifier for a book",
            "type": "string"
        },
        "Price": { "type": "integer" },
        "Edition": { "type": "integer" },
        "Title": { "type": "string" },
        "Author": {
            "type": "object",
            "properties": {
                "First_name": { "type": "string" },
                "Last_name": { "type": "string" }
            }
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            }
        }
    }
}
```

JSON Schema

- ograničenja
  - za svaki element mogu se napisati ograničenja koja važe za vrednosti tog elementa
  - postoji skup opštih elemenata za kontrolu vrednosti
  - za svaki tip postoji skup predefinisanih funkcija

JSON Schema

- globalna ograničenja
  - važe za element bilo kog tipa
  - *“optional“ : true/false*
    - za svaki element može se reći da li je obavezan ili ne
  - *“enum“ : lista vrednosti*
    - lista dozvoljenih vrednosti
  - *“allOf“, “anyOf“, “oneOf“ : lista vrednosti*
    - lista schema koje vrednosti moraju zadovoljiti
  - *“not“ : lista vrednosti*
    - lista schema koje vrednosti ne smeju zadovoljiti

JSON Schema

- ograničenja za *number* i *integer* tipove
  - *“multipleOf“ : broj*
    - proverava da li je vrednost elementa deljiva sa brojem specificiranim u ograničenju
  - *“maximum“ : broj*
  - *“exclusiveMaximum“ : true/false*
  - *“minimum“ : broj*
  - *“exclusiveMinimum“ : true/false*

JSON Schema

- ograničenja za *string* tip
  - *“maxLength“ : broj*
  - *“minLength“ : broj*
  - *“pattern“ : string*
    - vrednost ovog elementa mora biti ispravan regularni izraz

JSON Schema

- ograničenja za *array* tip
  - *“items“ : string*
    - vrednost označava podschemu za opis vrednosti u listi
  - *“additionalItems“ : true/false* ili *objekat*
  - *“maxItems“ : broj*
  - *“minItems“ : broj*
  - *“uniqueItems“ : true/false*

JSON Schema

- ograničenja za *object* tip
  - *“maxProperties“ : broj*
  - *“minProperties“ : broj*
  - *“required“ : lista vrednosti*
  - *“additionalProperties“ : true/false* ili *objekat*
  - *“patternProperties“ : objekat*

JSON Schema

```
{
    "title": "Book schema",
    "description": "A schema for the book JSON object from the Data organization course",
    "type": "object",
    "properties": {
        "ISBN": {
            "description": "The unique identifier for a book",
            "type": "string",
            "pattern" : "ISBN*",
            "optional": false
        },
        "Price": {
            "type": "integer",
            "minimum": 0,
            "maximum": 100000
        },
        "Edition": { "type": "integer" },
        "Title": { "type": "string" },
        "Author": {
            "type": "object",
            "properties": {
                "First_name": { "type": "string" },
                "Last_name": { "type": "string" }
            }
        },
        "tags": {
            "type": "array",
            "items": {
                "type": "string"
            },
            "minItems": 1,
            "uniqueItems": true
        }
    }
}
```

Zadatak 5

- Napisati JSON Schema dokument za podatke koji se nalaze u datoteci *Bookstore.json*

Zadatak 6

- Napisati Java program koji validira zadatu JSON datoteku i proverava njenu sintaksnu i semantičku tačnost u odnosu na zadatu schemu.