A programozás alapjai 3.

Strukturált adatreprezentációk

Goldschmidt Balázs
balage@iit.bme.hu

1

Bevezetés

- Objektumok a memóriában vannak
  - hogyan küldjük át őket a hálózaton?
  - hogyan tároljuk őket fájlokban?
- Strukturált adatreprezentációk
  - Sorosítás
  - XML
  - JSON

2

Szerializálás

3

Sorosítás alapok

- Feladat
  - mentsük el az objektumokat, majd töltsük vissza őket
  - mit kell tárolni?
    - objektumok attribútumai
    - asszociációk
    - statikus mezők?
- Egyszerű megoldás: *sorosítás (serialization)*
  - Java beépítetten tartalmazza
  - objektumok adatait és asszociációit bájtfolyamba tudja menteni és onnan visszatölteni

4

Sorosítás fogalmak

- Sorosítás (serialization)
  - objektumok konvertálása bináris formába (bájtfolyam)
    - további angol elnevezések: marshalling, deflating
  - bináris adat tárolható, hálózaton átküldhető, stb.
- Visszasorosítás (deserialization)
  - bináris adat (bájtfolyam) konvertálása objektumokká
    - további angol elnevezések: unmarshalling, inflating
  - bináris adat olvasható fájlból, érkezhet hálózaton, stb.

Sorosítás példa: mentés

```
import java.io.Serializable;
public class SerializableClass implements Serializable
{ ... }
```

```
//import java.io.*;
...
SerializableClass ser = ...;
try {
      FileOutputStream f =
                  new FileOutputStream("filename");
      ObjectOutputStream out =
                  new ObjectOutputStream(f);
      out.writeObject(ser);
      out.close();
}
catch(IOException ex) { ... }
```

Sorosítás példa: visszaolvasás

```
//import java.io.*;
...
SerializableClass ser2;
try {
       FileInputStream f =
                     new FileInputStream("filename");
       ObjectInputStream in =
                     new ObjectInputStream(f);
       ser2 = (SerializableClass)in.readObject();

       in.close();

} catch(IOException ex) {
       ...
} catch(ClassNotFoundException ex) {
       ...
}
```


Sorosítás szabályai

- Csak olyan osztályok sorosíthatók, amelyek implementálják a *Serializable* interfészt
  - az is elég, ha az ősosztály implementál
  - tömbök, String, Integer, Double, stb. sorosíthatók
- *Serializable* interfész
  - nincsenek metódusai
  - csak egy jelzés a fordítónak
- *Nem sorosíthatók:*
  - *Object, Socket, Input/OutputStream, System, stb.*

Sorosítás szabályai

- Mi lesz sorosítva?
  - primitív típusú attribútumok
  - sorosítható típusú attribútumok
    - rekurzívan
- Mi nem lesz sorosítva?
  - statikus mezők
  - *transient* mezők

```
public class Serial implements Serializable {
       transient private String secret;
       private String other;
       ...
```


9

Sorosítás folyamata

- A sorosítás rekurzív
  - hogyan kerüljük el a köröket az objektumgráfban?
  - az objektum tartalma csak egyszer íródik ki
  - minden további alkalommal csak egy referencia

```
out.writeObject(a);
out.writeObject(b);
out.writeObject(a); // csak referencia!
```

- Örökölt tagok sorosítása
  - a legfelső sorosítható őstől indul

10

Sorosítás folyamata 2

- Vegyük a következő osztályokat!

```
class Student implements
    Serializable {
  String name;
  University uni;
  Address a;
  double average;
  // ctr, set, get
}
```

```
class Address implements
    Serializable {
  String country,
    city, street;
 // ctr, set, get
}
```

```
class University implements
    Serializable {
  String name;
  Address a;
 // ctr, set, get
}
```


Sorosítás folyamata 3

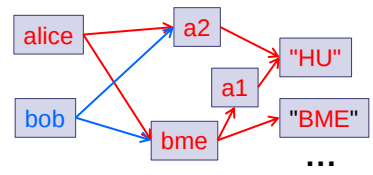
- Vegyük a következő objektumokat!

```
Address a1 = new Address("HU", "Bp", "Műegyetem rkp 3");
Address a2 = new Address("HU", "Bp", "Alkotás u. 10");

University bme = new University("BME", a1);

Student alice = new Student("Alice", bme,  a2);
Student bob   = new Student("Bob", bme, a2);
```

```
ObjectOutputStream oos = ...;
oos.writeObject(alice);
oos.writeObject(bob);
oos.close();
```

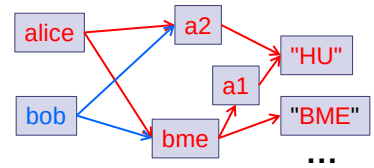


Sorosítás folyamata 4

- Olvassunk vissza!

```
ObjectInputStream ois = ...;
Student alice2 = (Student)ois.readObject();
Student bob2 = (Student)ois.readObject();
ois.close();

System.out.println(bob2.getUni().getName());
```



Saját sorosítás

- Implementáljuk a következő függvényeket:

```
private void writeObject(ObjectOutputStream out)
        throws IOException
private void readObject(ObjectInputStream in)
        throws IOException, ClassNotFoundException
```

- Ős/leszármazott adatok kezelése automatikus
- Alapértelmezett implementáció
  - `out.defaultWriteObject()`, `in.defaultReadObject()`
- *out* és *in* segédfüggvényeket adnak
  - primitív és sorosítható típusok írása/olvasása

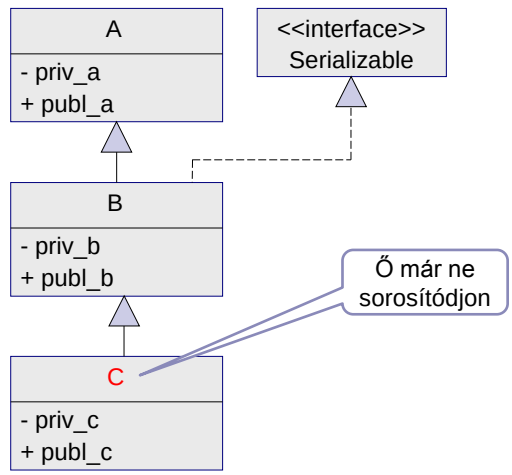
Saját sorosítás 2

- Teljes irányításhoz implementáljuk:

```
interface Externalizable extends Serializable {
  public void writeExternal(ObjectOutput out)
    throws IOException;
  public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException;
}
```

- Ős/leszármazott adatok kezelését explicit kell megoldani

Sorosítás megállítása



Sorosítás megállítása

```
private void writeObject(ObjectOutputStream out)
throws IOException {

      throw new NotSerializableException("C");
}


private void readObject(ObjectInputStream in)
throws IOException, ClassNotFoundException {

      throw new NotSerializableException("C");
}
```


Verziózás

- Minden osztálynak egyedi verzióazonosítója van
  - lekérdezés: `serialver` *`ClassName`*
- Kompatibilitás biztosítása
  - ugyanannak a verzióazonosítónak (ID) a használata:

```
static final long serialVersionUID
 = 10275539472837495L;
```

- Kompatibilis változások
  - metódus/mező hozzáadása vagy láthatóság módosítása
  - static/transient → perzisztens

XML

Egyéb sorosítási formátumok

- A Java sorosítás alacsony szintű
  - sorosított adatok kezelése nehéz
- XML
  - saját protokoll
  - JAXB
    - *javax.xml.bind* csomag
- JSON
  - **J**ava**S**cript **o**bject **n**otation
  - http://www.json.org/
- …

XML bevezetés

- e**X**tensible **M**arkup **L**anguage
- Cél: szabványos, emberileg olvasható formátum
  - tárolás
  - átvitel
- a HTML sikerén alapul
- szöveges
  - vö. `toString()`
- Manapság: „Bármi is a kérdés, XML a válasz”
  - vagy JSON…

XML tulajdonságai

- Fastruktúra: hierarchikus
  - elemek (tag, element), attribútumok és szöveg
- W3C szabvány
  - szintaxis
  - feldolgozási szabályok
  - jólformált ill. valid
- Meta-struktúra, kiterjeszthető

XML jólformáltság

- Szintaktikai szabályok
  - opcionális fejléc

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

  - minden elem le van zárva

```
<p>Hello <img src="kitty.png"/> </p>
```

  - egyetlen gyökér
  - elemek egymás után vagy egymásba ágyazva

XML szintaxis

- megjegyzés

```
<!– ez egy megjegyzés -->
```

- elem attribútuma

```
<img src="cat.png"/>
```

  - mindig idézőjel
- speciális jelek

XML	text
&amp;	&
&lt;	<
&gt;	>
&apos;	'
&quot;	"

Valid XML

- Szemantikai szabályok
- Kényszerek megadhatók…
  - schema
  - DTD (document type definition)
- Specifikálja a dokumentum struktúráját
  - megengedett elemek
  - megengedett hierarchia
  - megengedett attribútumok
  - ...

XML szemantika leírása

- DTD
  - régi
  - nem teljes
  - elavult
- XML schema (XSD)
  - új
  - strukturált: elemek és kapcsolatok
  - névterek
  - önmaga is XML

XML példa

```
<?xml version="1.0" encoding="UTF-8"?>
<people_list>
  <person>
    <name>Neil Armstrong</name>
    <birthdate>1930-08-05</birthdate>
    <gender>Male</gender>
  </person>
  <person>
    <name>Buzz Aldrin</name>
    <birthdate>1930-01-20</birthdate>
    <gender>Male</gender>
    <neptuncode>M00N02</neptuncode>
  </person>
</people_list>
```


DTD leírás

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate, gender?,
      neptuncode?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT neptuncode(#PCDATA)>
```

XSD leírás

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="people_list">
    <xs:complexType><xs:sequence>
        <xs:element name="person" maxOccurs="unbounded">
          <xs:complexType><xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="birthdate" type="xs:string"/>
            <xs:element name="gender" type="xs:string"
                minOccurs="0"/>
            <xs:element name=„neptuncode" type="xs:string"
                minOccurs="0"/>
          </xs:sequence></xs:complexType>
        </xs:element>
      </xs:sequence></xs:complexType>
  </xs:element>
</xs:schema>
```


XML kezelés

- kézzel
  - inkább ne
- DOM *(Document Object Model)*
  - buta objektumstruktúra
- JDOM *(Java DOM)*
  - okos objektumstruktúra
- SAX *(simple API for XML)*
  - esemény alapú
  - soros feldolgozás

Document Object Model

DOM

- *Document Object Model*
  - objektummodellt épít az XML tartalom alapján
- Objektummodell módosítható
  - kiírható XML-ként
- Validálni is tud

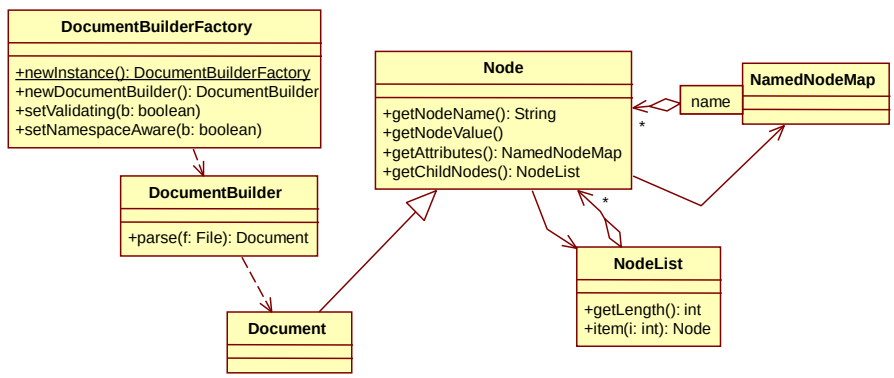
DOM bevezető példa

```
try {
  DocumentBuilderFactory factory =
    DocumentBuilderFactory.newInstance();
  factory.setValidating(true);
  factory.setNamespaceAware(true);

  DocumentBuilder builder =
    factory.newDocumentBuilder();

  Document document =
    builder.parse(new java.io.File(args[0]));
  ...
} catch (Exception e) {
  e.printStackTrace();
}
```

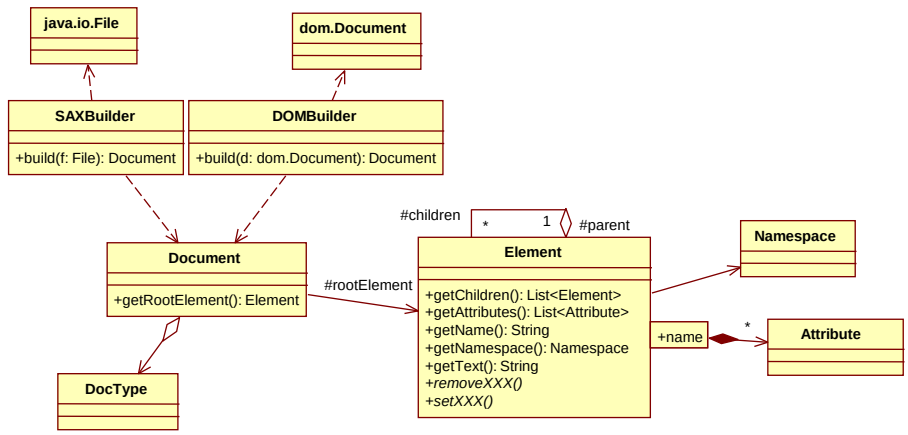

DOM osztályok



JDOM

- Objektummodell közelebb áll az XML-hez
  - **`Attribute, CDATA, Comment, Content, DefaultJDOMFactory, DocType, Document, Element, EntityRef, Namespace, ProcessingInstruction, Text`**
- Attribútumokat és gyerekeket könnyebb elérni és módosítani
  - nincs **`NamedNodeMap`**, **`NodeList`**
  - **`java.util.List`**-et használ

JDOM osztályok (részlet)



JDOM tulajdonságok

- Beolvasás kiszervezve
  - **`DOMBuilder`**
  - **`SAXBuilder`**
- Dokumentum elmenthető
  - XML dokumentumként
  - DOM modellként
  - SAX eseménygenerátorként

JDOM tulajdonságok

- Szűrők definiálhatók kereséshez
  - **`org.jdom.filter.Filter`**
    - **`boolean matches(java.lang.Object obj)`**
- XSL transzformációkat is támogat
  - **`org.jdom.transform.XSLTranformer`**
- XPATH kereséseket is támogat
  - **`org.jdom.xpath.XPath`**

JDOM példa: egyszerű kiírás

```
public class JDOMParse {
  static void print(Element n, String tab) ...
  public static void main(String[] args) {
      SAXBuilder b = new SAXBuilder();
      File f = new File("test.xml");
          try {
              Document doc = (Document)b.build(f);
              Element r = doc.getRootElement();
              print(r, "");
          } catch (IOException io) {
               System.out.println(io.getMessage());
          } catch (JDOMException je) {
              System.out.println(je.getMessage());
          }
    }
}
```


JDOM példa: egyszerű kiírás

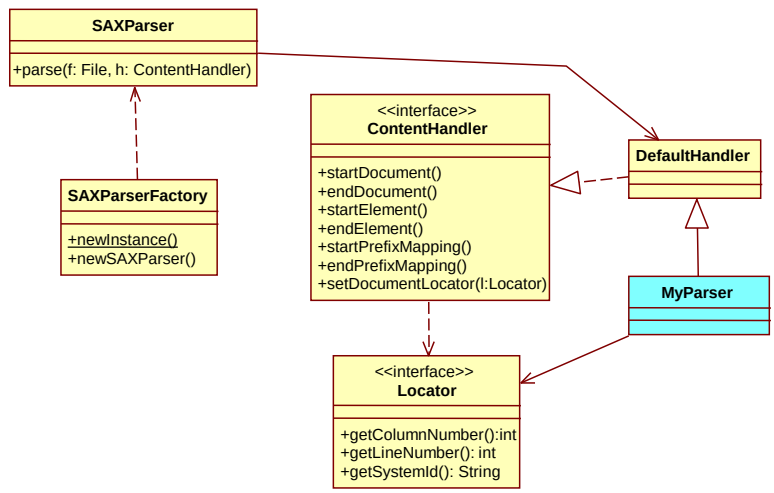
```
static void print(Element n, String tab) {
    System.out.println(tab+"("+n.getName()
       +") \""+n.getValue()+"\"");
    if (n.hasAttributes()) {
        List<Attribute> list = n.getAttributes();
        for (Attribute a : list) {
            System.out.println(tab+"attr: "+a);
        }
    }
    List<Element> nl = n.getChildren();
    for (Element e : nl) {
        print(e, tab+"  ");
  }
}
```

Rekurzió

SAX olvasó használata

- Eseménykezelés
  - dokumentum eleje/vége
  - elem eleje/vége
  - prefix leképzés eleje/vége
  - karakterek (szöveg)
  - whitespace (szóköz, tab, stb.)
  - átugrott entitások
  - feldolgozási utasítások

SAX olvasó osztályai



ContentHandler interfész

- **`org.xml.sax.ContentHandler`**
- Ezt kell implementálni
- Callback jellegű eseménykezelés
  - minden egyes eseménykezelő egy függvény
- Alapértelmezett implementáció
  - **`org.xml.sax.helpers.DefaultHandler`**
  - minden metódus üres

ContentHandler

```
void startDocument()
void endDocument()

void startElement(String uri,
    String localName, String qName,
    Attributes atts)
void endElement(String uri,
    String localName, String qName)

void startPrefixMapping(String prefix,
    String uri)
void endPrefixMapping(String prefix)
```

ContentHandler

- `void characters(char[] ch, int start, int length)`
- `void ignorableWhitespace(char[] ch, int start, int length)`
- `void processingInstruction(String target, String data)`
- `void skippedEntity(String name)`
- `void setDocumentLocator(Locator locator)`
  - a locator segítségével további adatfeldolgozás válik elérhetővé

ContentHandler példa

- Feladat:
  - Készítsünk egy egyszerű Java alkalmazást
  - amely kiírja az XML fát
    - attribútumok nélkül
    - szövegek nélkül

ContentHandler példa

- Megoldás:
  - *ContentHandler* interfész implementálása
    - *DefaultHandler* osztály segítségével
  - handler regisztrálása
  - XML fájl feldolgozása

ContentHandler példa

```
public class MyParser extends DefaultHandler {

    public static void main(String[] args) {
        DefaultHandler h = new MyParser();
        SAXParserFactory factory =
            SAXParserFactory.newInstance();
        try {
            SAXParser p = factory.newSAXParser();
            p.parse(new java.io.File(args[0]), h);
        } catch (Exception e) {e.printStackTrace();}
    }

    ...
```

ContentHandler példa

```
    ...
    int tab=0;
    public void println(String s) {
        for (int i = 0; i < tab; i++) {
            System.out.print("  ");
        }
        System.out.println(s);
    }
    public void startDocument() throws SAXException {
        println("Start document");
    }
    public void endDocument() throws SAXException {
        println("End document");
    }
    ...
```


ContentHandler példa

```
    ...
    public void startElement(String namespaceURI,
       String sName, String qName, Attributes attrs)
       throws SAXException {
         tab++;
         println("start element: "+qName);
    }

    public void endElement(String namespaceURI,
       String sName, String qName)
       throws SAXException {
         println("end element: "+qName);
         tab--;
    }
}
```

ContentHandler példa bemenet

```
<!-- test.xml -->
<level1>
        <level2>
                <level3 attr1="test1">
                </level3>
                <level3 attr1="test2" attr2="second">
                </level3>
                <level3 attr1="test3">
                </level3>
        </level2>
</level1>
```

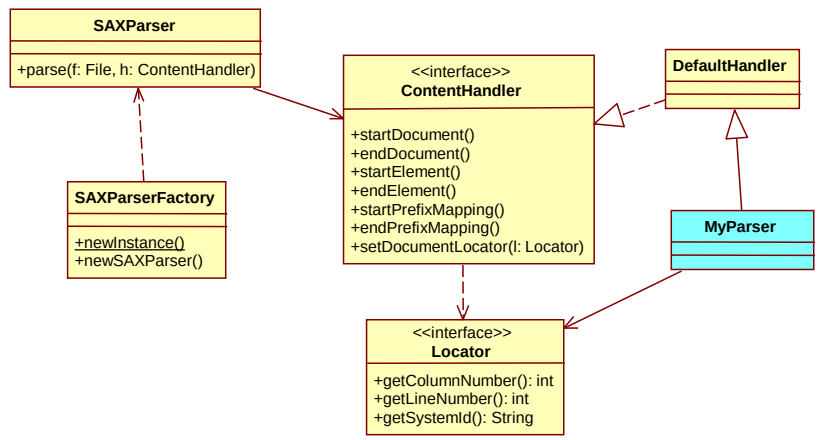

ContentHandler példa kimenet

```
$ java MyParser test.xml
Start document
  start element: level1
    start element: level2
      start element: level3
      end element: level3
      start element: level3
      end element: level3
      start element: level3
      end element: level3
    end element: level2
  end element: level1
End document
```

Locator

- A feldolgozott fájlról ad információt
- **`void setDocumentLocator(Locator l)`**
  - **`int getColumnNumber()`**
    - a karakter pozíciója a handler által éppen feldolgozott sorban
  - **`int getLineNumber()`**
    - a feldolgozott sor száma
  - **`String getSystemId()`**
    - a dokumentum neve (pl. fájlnév) URL-ként

Locator helye



Locator példa

```
Locator loc = null;
public void setDocumentLocator(Locator l) {
    println("LOCATOR");
    loc = l;
}
```

```
public void startElement(String namespaceURI,
   String sName, String qName, Attributes attrs)
   throws SAXException {
     tab++;
     println("start element: "+qName);
     println("  Locator: ("+loc.getLineNumber()
       +","+loc.getColumnNumber()+") "
       +loc.getPublicId()+", "+loc.getSystemId());
     ...
}
```


Locator példa bemenet

```
<!-- test.xml -->
<level1>
        <level2>
                <level3 attr1="test1">
                </level3>
                <level3 attr1="test2" attr2="second">
                </level3>
                <level3 attr1="test3">
                </level3>
        </level2>
</level1>
```

Locator példa kimenet

```
$ java MyParser test.xml
LOCATOR
Start document
  start element: level1
    Locator: (1,9) null,
file:/home/balage/sax-example/example4/test.xml
    start element: level2
      Locator: (3,10) null,
file:/home/balage/sax-example/example4/test.xml
      start element: level3
        Locator: (4,25) null,
file:/home/balage/sax-example/example4/test.xml
        attr0: attr1=test1
      end element: level3
...
```


Dokumentum validálás

- Adjunk hozzá validációt!

```
SAXParserFactory factory = SAXParserFactory.newInstance();
factory.setValidating(true);
factory.setNamespaceAware(true);
```

```
SAXParser p = factory.newSAXParser();
String JAXP_SCHEMA_LANGUAGE =
 "http://java.sun.com/xml/jaxp/properties/schemaLanguage";
String W3C_XML_SCHEMA =
 "http://www.w3.org/2001/XMLSchema";
p.setProperty(JAXP_SCHEMA_LANGUAGE, W3C_XML_SCHEMA);
```

XML hibakezelés

- Hiba típusok
  - fatal error
    - a dokumentum nem jólformált
  - error
    - a dokumentum nem valid
  - warning
    - kis hiba, pl. ugyanaz a típus kétszer van deklarálva

JSON

JSON

- Könnyűsúlyú adatcsere formátum
  - emberek számára is könnyen olvasható és írható
    - többnyire…
  - gépek számára könnyen feldolgozható és generálható
  - vigyázat: nincs metaadat, extra tesztelés szükséges!
- Két struktúra
  - *Objektum*: név-érték párok rendezetlen halmaza
    - mint egy objektum vagy struktúra
  - *Tömb*: értékek rendezett listája
    - mint egy tömb, vektor, lista vagy sorozat

JSON szintaxis

- Objektum (Object)
  - eleje: **{**
  - vége: **}**
  - minden név után kettőspont
  - név-érték párok vesszővel elválasztva
- Tömb (Array)
  - eleje: **[**
  - vége: **]**
  - értékek vesszővel elválasztva

```
{
    "name" : "alice",
    "university" : {
        "name" : "BME"
    },
    "average" : 4.2
}
```

```
[ "mon", "tue", "wed",
  "thu", "fri", "sat",
  "sun"
]
```

JSON szintaxis 2

- Érték (Value)
  - karakterlánc *(string)* idézőjelek között
  - szám *(number)*
  - *true* vagy *false* vagy *null* érték
  - objektum *(object)*
  - tömb *(array)*
- Struktúrák egymásba ágyazhatók

```
{
    "name" : "alice",
    "university" : {
        "name" : "BME"
    },
    "average" : 4.2,
    "siblings" : [
        "bob",
        "charlie"
    ],
    "degree" : null
}
```


JSON kezelés Javában

- Könyvtárak
  - org.json (pl. Android)
    - típusok osztályokra leképezve
    - elemenként felépítve
  - javax.json (JEE 7)
    - objektumok felépítése builder minta segítségével
  - Google gson
  - …

javax.json: JSON objektum létrehozása

```
// sometimes trivial
// nesting is used

{ "student" : {
    "name" : "alice",
    "university" : {
        "name" : "BME"
    },
    "average" : 4.2
  }
}
```

```
JsonObject value =
  Json.createObjectBuilder()
  .add("name", "alice")
  .add("university",
     json.createObjectBuilder()
     .add("name", "BME")
  )
  .add("average", 4.2)
  .build();

JsonObject student =
  Json.createObjectBuilder()
  .add("student", value).build();
```


javax.json: olvasás és írás

```
// src lehet InputStream vagy Reader
JsonReader reader = Json.createReader(src);
JsonObject obj = reader.readObject();
// JsonArray arr = reader.readArray();
reader.close();
```

```
// target lehet OutputStream vagy Writer
JsonWriter w = Json.createWriter(target);
JsonObject o = ...;
w.writeObject(o);
w.close();
```

javax.json: adatok elérése

- *JsonObject* és *JsonArray* gettereket tartalmaz:
  - boolean, int, String (alapértelmezett értékkel is)
  - JsonArray, JsonNumber, JsonObject, JsonString
  - *isNull*
- Indexelés
  - JsonObject: *String*
  - JsonArray: *int*

```
JsonObject obj = ...;
String name = obj.getString("name");
String uni_name =
  obj.getJsonObject("university")
  .getString("name");
```


Köszönöm a figyelmet!