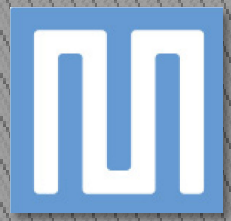
# Java API for RESTful Web Services (JAX-RS)

# **Re**presentational **S**tate **T**ransfer (REST)

- A REST (Representational State Transfer) egy szoftverarchitektúra típus elosztott hipermédia rendszerek számára, amilyen például a világháló.
- A REST típusú architektúra kliensekből és szerverekből áll.
  - A kliensek kéréseket indítanak a szerverek felé; a szerverek kéréseket dolgoznak fel és a megfelelő választ küldik vissza.

# **Re**presentational **S**tate **T**ransfer (REST)

- A kérések és a válaszok erőforrás–reprezentációk szállítása köré épülnek.
  - Az erőforrás lényegében bármilyen koherens és értelmesen címezhető koncepció lehet.
  - Egy erőforrás–reprezentáció tipikusan egy dokumentum, mely rögzíti az erőforrás jelenlegi vagy kívánt állapotát.
- Az alapelv nem köti meg a reprezentáció formátumát, gyakran XML, HTML, JSON, de lehet kép, egyszerű szöveg is


# RESTful webszolgáltatás

- HTTP fölött, URI alapon egyszerűen címezhető szolgáltatás, melyhez meg kell adni
  - a szolgáltatás URI-jét
  - a szolgáltatás által támogatott MIME type-ot
  - a szolgáltatás által támogatott HTTP metódusokat (általában GET, POST, PUT, DELETE)
- Nincs rá szabvány, mint pl. XML webszolgáltatások esetén SOAP, WSDL

- Egy REST alkalmazás a HTTP protokoll meglévő tulajdonságait használja és így lehetővé teszi a proxyknak és az átjáróknak, hogy együttműködjenek az alkalmazással (például gyorsítótárazás vagy biztonsági funkciók formájában).

# CRUD – HTTP leképezése

| Alkalmazás művelet | HTTP protokoll |
|---|---|
| Create | POST |
| Retrieve | GET |
| Update | PUT |
| Delete | DELETE |

# RESTful URI példák

- Új ügyfél létrehozása:
  POST http://www.example.com/customers
- Adott ügyfél lekérdezése:
  GET http://www.example.com/customers/33245
- Adott ügyfél törlése
  DELETE http://www.example.com/customers/33245
- Adott ügyfél módosítása
  PUT http://www.example.com/customers/33245
- Adott ügyfél megrendeléseinek lekérdezése:
  GET http://www.example.com/customers/33245/orders
- Adott ügyfél adott megrendelése adott tételének lekérdezése GET http://www.example.com/customers/33245/orders/8769/lineitems/1

# Kapcsolódó fogalmak: Média típusok

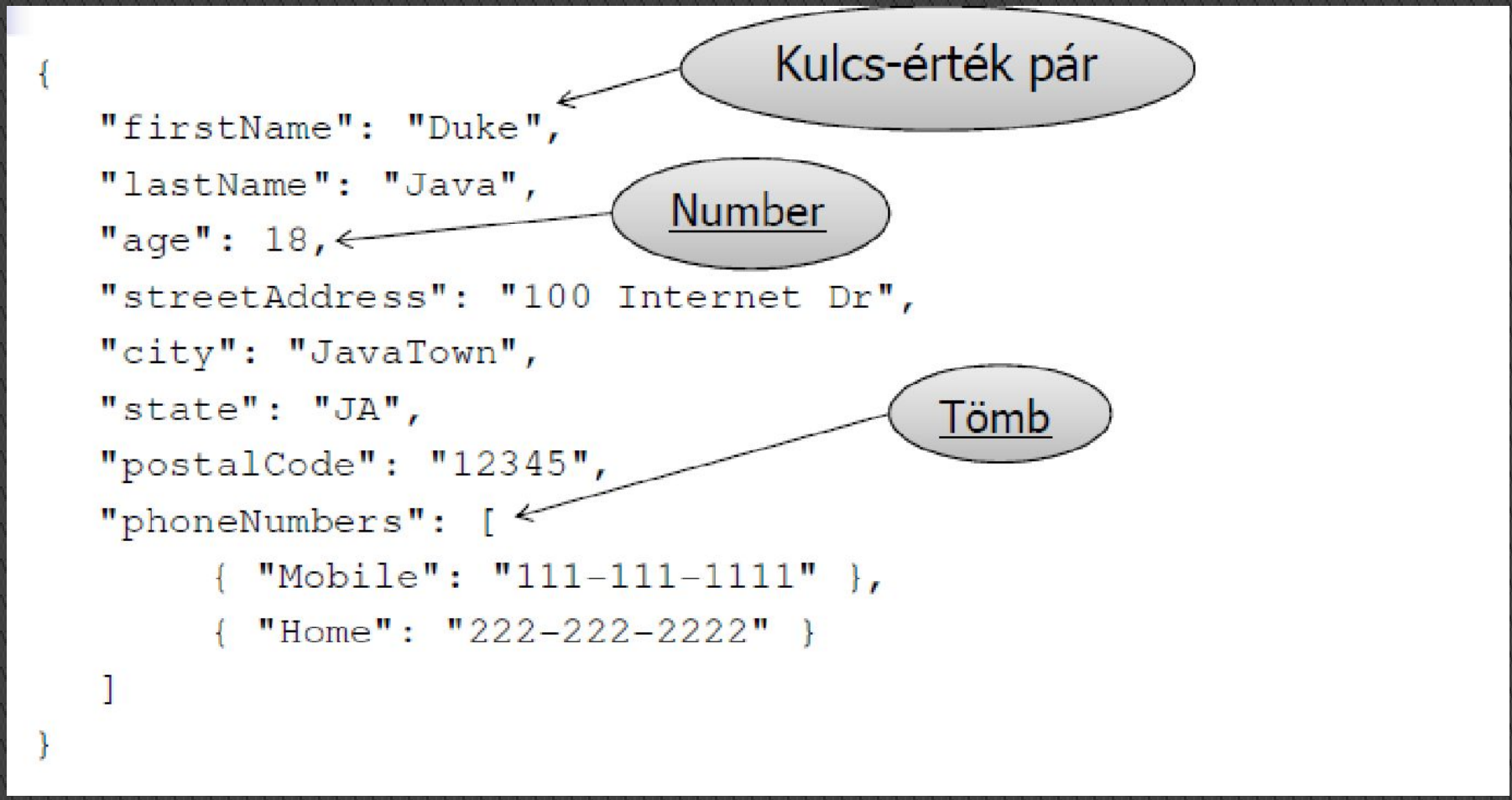
- MIME type (Multipurpose Internet Mail Extensions) típus – tartalomtípust definiálja egy típus/altípus alakú kóddal

| Típus/altípus | Magyarázat |
|---|---|
| Text/plain | ASCII szöveg |
| Text/html | Html fájl |
| Application/json | json |
| Image/jpeg | kép |
| Audio/mp3 | Mp3 fájl |
| Application/xml | xml |
| | |

# Kapcsolódó fogalmak: JSON (JavaScript Object Notation)

- Egy kis méretű, szöveg alapú szabvány ember által olvasható formátum adatcserére, mely kulcs–érték párokat használ
- A JavaScript natív objektumleíró eszköze
- RESTful webszolgáltatások gyakran ezt használják
- Adat struktúrák:
  - Objektumok
  - Tömbök
- Adattípusok: String, Number, Object, Array, True, False, Null
- Ha JSON a HTTP csomag törzse, a fejlécben:
  - Content-Type: application/json

# JSON példa




```
{
    "firstName": "Duke",
    "lastName": "Java",
    "age": 18,
    "streetAddress": "100 Internet Dr",
    "city": "JavaTown",
    "state": "JA",
    "postalCode": "12345",
    "phoneNumbers": [
        { "Mobile": "111-111-1111" },
        { "Home": "222-222-2222" }
    ]
}
```

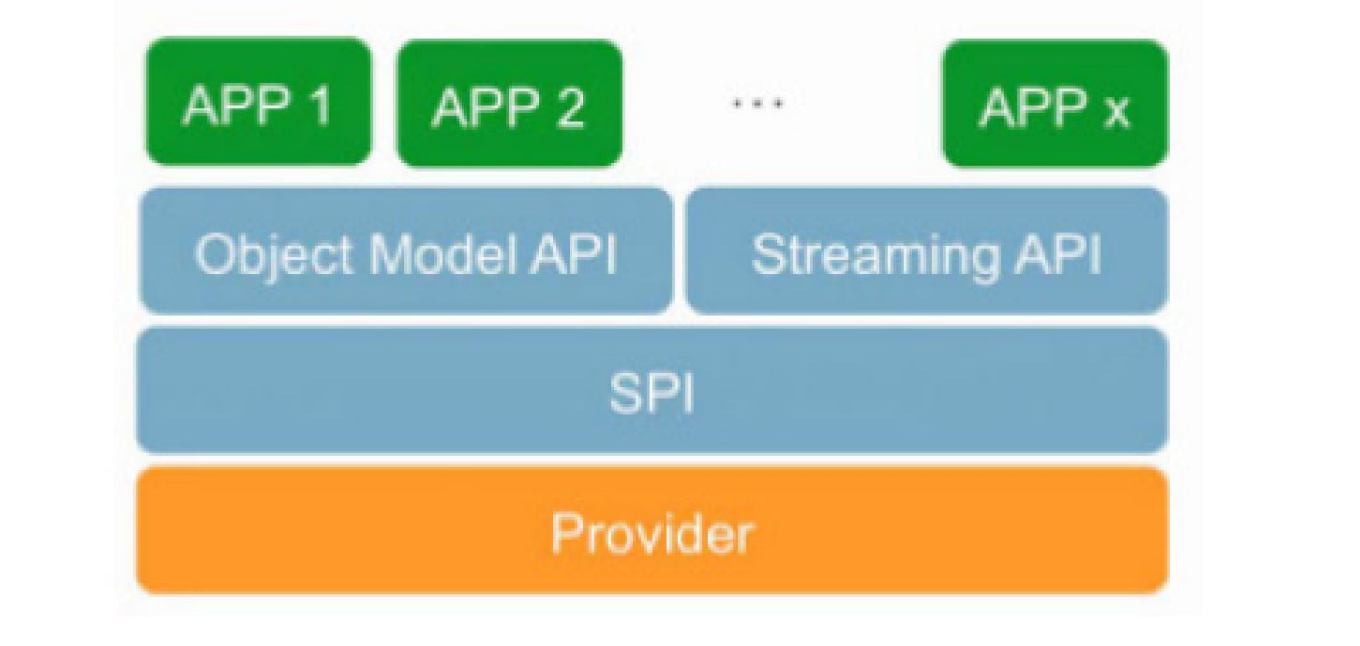

# JSON adatok kezelése

- Két modell:

1. Object modell
   - Fa reprezentálja az adatokat
   - Navigálható, módosítható
   - Akkor érdemes használni, ha az egész fára szükség van
   - Gyakran lassabb, mint a streaming
2. Streaming modell
   - Esemény alapú, egyszerre egy elemet olvas ki
   - Ha csak egy elemre van egyszerre szükségünk

# Java API for JSON Processing

- Eddig 3rd party megoldások (jackson, gson)
- JAVA EE 7-ben mindkét modellre található implementáció
  - Object modell –> javax.json
  - Streaming modell –> javax.json.stream



# JAX-RS

- RESTful webszolgáltatást Java-ban lehetne egyszerű Servlettel is fejleszteni, de a JAX-RS magasabb szintű támogatást nyújt hozzá
- Jellemzők:
  - POJO alapú
  - HTTP-centrikus
  - sok formátum támogatása, és bővíthető újakkal
- javax.ws.rs és alcsomagjai

# JAX-RS példa

```
@Path ("items")
@Produces (MediaType.APPLICATION_XML)
public class ItemsResource {
   @Context
   UriInfo uriInfo;

   @GET List<Item> listItems() { return getAllItems(); }

   @POST
   @Consumes (MediaType.APPLICATION_XML)
   public Response create(Item item)
        throws ItemCreationException {
        Item newItem = createItem(item);
        URI newItemURI = uriInfo.getRequestUriBuilder()
                .path(newItem.getId()).build();
        return Response.created(newItemURI).build();
   }
...
}
```

# JAX–RS erőforrások

- POJO, @Path–szal annotálva, vagy legalább egy metódusa @GET, @POST stb.–vel annotálva
- Az Application absztrakt ősosztályból leszármazva, annak getClasses metódusában adjuk vissza az alkalmazás által kezelt erőforrástípusokat
- Egy JAX–RS alkalmazás Java SE, JAX–WS, Servlet és teljes Java EE környezetben is futhat, egyes implementációk máskörnyezetet is támogathatnak
  - állapotmentes EJB is lehet JAX–RS erforrás

# URI sablonok

- A @Path bemen paraméterében adható meg, az URI egyes részeinek parszolására használható, a sablonban lévő névre hivatkozunk a @xxxParam annotációkkal, pl.

```
@Path("messages/{msgNum}")
@GET
public Message getMessage(
   @PathParam("msgNum") int msgNum
   )
```

- Regex is használható, pl. @Path("widgets/{path:.+}")

# URI sablonok

- @Path egy metóduson
  - mindig az osztályhoz tartozó @Path értékhez relatívan értelmezendő
  - az ilyen metódus vagy maga szolgál ki REST kérést (sub-resource metódus), ekkor @GET/@POST/stb. is kell rá (lásd fent)
  - vagy csak egy olyan objektumot ad vissza, ami majd kiszolgálja (sub-resource locator), ilyenkor a @GET/@POST/stb. elmarad

# Paraméterek

- @Path: relatív útvonal, amelyen elérünk egy erőforrást@GET, @PUT, @POST, @DELETE: milyen http kérést használjon az erőforrás eléréshez
- @PathParam(„msgNum”) – sablonban levő név alapján url-ből paraméter
- @QueryParam: `/users/query?url=pelda.com` -> @QueryParam("url")

# Média típusok

- @Consumes(MediaType.APPLICATION_JSON): milyen MIME típusú kéréseket fogad
- @Produces(„text/plain”): válasz MIME típusa

- Egy erőforrás osztály több formátumot is támogathat, különböző metódusokkal: @Consumes({test/plain, text/html})
- A JAX-RS implementáció nem hív meg olyan metódust, ahol a kérés Content-Type fejléce nem egyezik meg a @Consumes értékével
- vagy a kérés Accept fejléce nem egyezik meg @Producesértékével

# Kontextusok

- @Context–tel injektálhatók resource osztályba, providerbe vagy Application gyermekosztályba
- Context providerrel szabadon bővíthető az injektáltatható kontextusok köre
- Az alapból támogatott típusok:
  - Application (kivéve magába az Applicationbe) pl. konfigurációt érdemes ide gyűjteni
  - UriInfo: hozzáférés a kérés URI–hez
  - HttpHeaders: hozzáférés a kérés fejléceihez
  - Request: hozzáférés a kéréshez, pl. elfeltételek kiértékelése (lastmodified, stb.)
  - SecurityContext: biztonsági kontextus (felhasználó, szerep, stb.)
  - + Java EE környezetben managed beanek, CDI

# Kontextusok

```
@Path("http-headers")
@GET
public String httpHeaders(@Context HttpHeaders headers) {
  List<MediaType> acceptableMediaTypes =
    headers.getAcceptableMediaTypes();
  String xFoo = headers.getHeaderString("X-Foo");

  return String.format("acceptableMediaTypes: %s, X-Foo: %s",
      acceptableMediaTypes, xFoo);
}
```

```
@Path("context-uri-info/{thing}")
@GET
public String uriInfo(@Context UriInfo info) {
  URI baseUri = info.getBaseUri();
  MultivaluedMap<String, String> queryParams = info.getQueryParameters();
  MultivaluedMap<String, String> pathParams = info.getPathParameters();

  return String.format("base URI: %s, query params: %s, path params: %s",
      baseUri, queryParams.entrySet(), pathParams.entrySet());
}
```