Internet softverske arhitekture

Slajdovi sa predavanja

Branko Milosavljević

Katedra za informatiku, Fakultet tehničkih nauka, Univerzitet u Novom Sadu

2014.

Sajt predmeta

https://enastava.ftninformatika.com

Ko drži nastavu

- Branko Milosavljević, JUG-213
  `mbranko@uns.ac.rs`
- Milan Stojkov, JUG-108
  `stojkovm@uns.ac.rs`

Ispitne obaveze

- Predispitna obaveza
  - odbranjen projekat na kraju semestra
- Usmeni ispit

Literatura

- R.P. Sriganesh, G. Brose, M. Silverman. *Mastering EJB 3.0*. Wiley, 2006.
- C. Bauer, G. King. *Java Persistence with Hibernate*. Manning, 2007.
- D. Panda, R. Rahman, D. Lane. *EJB 3.0 In Action*. Manning, 2007.

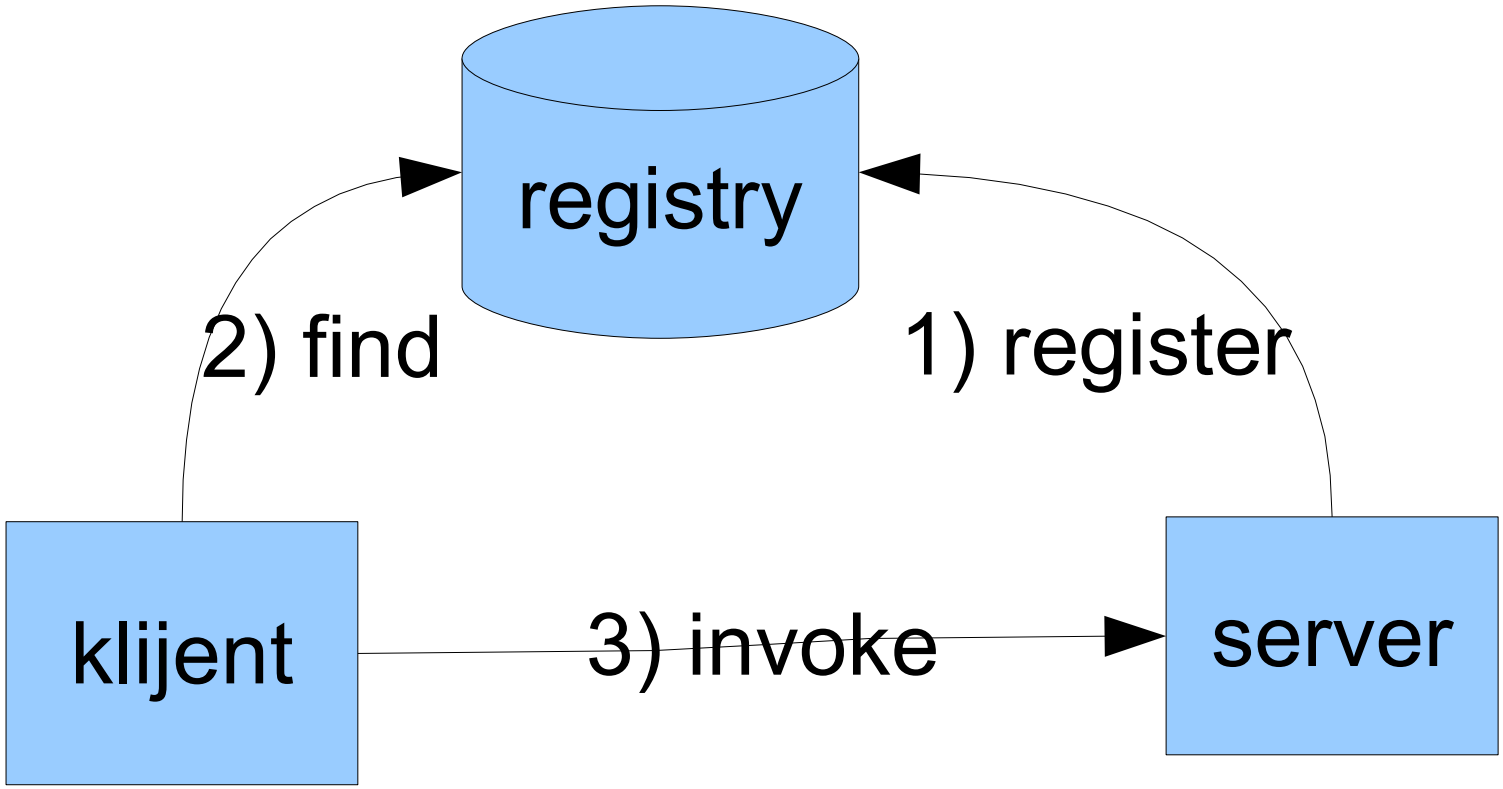
Ovi slajdovi

- Ovi slajdovi su samo pomoćni materijal
- (Verovatno) nisu dovoljni za polaganje ispita

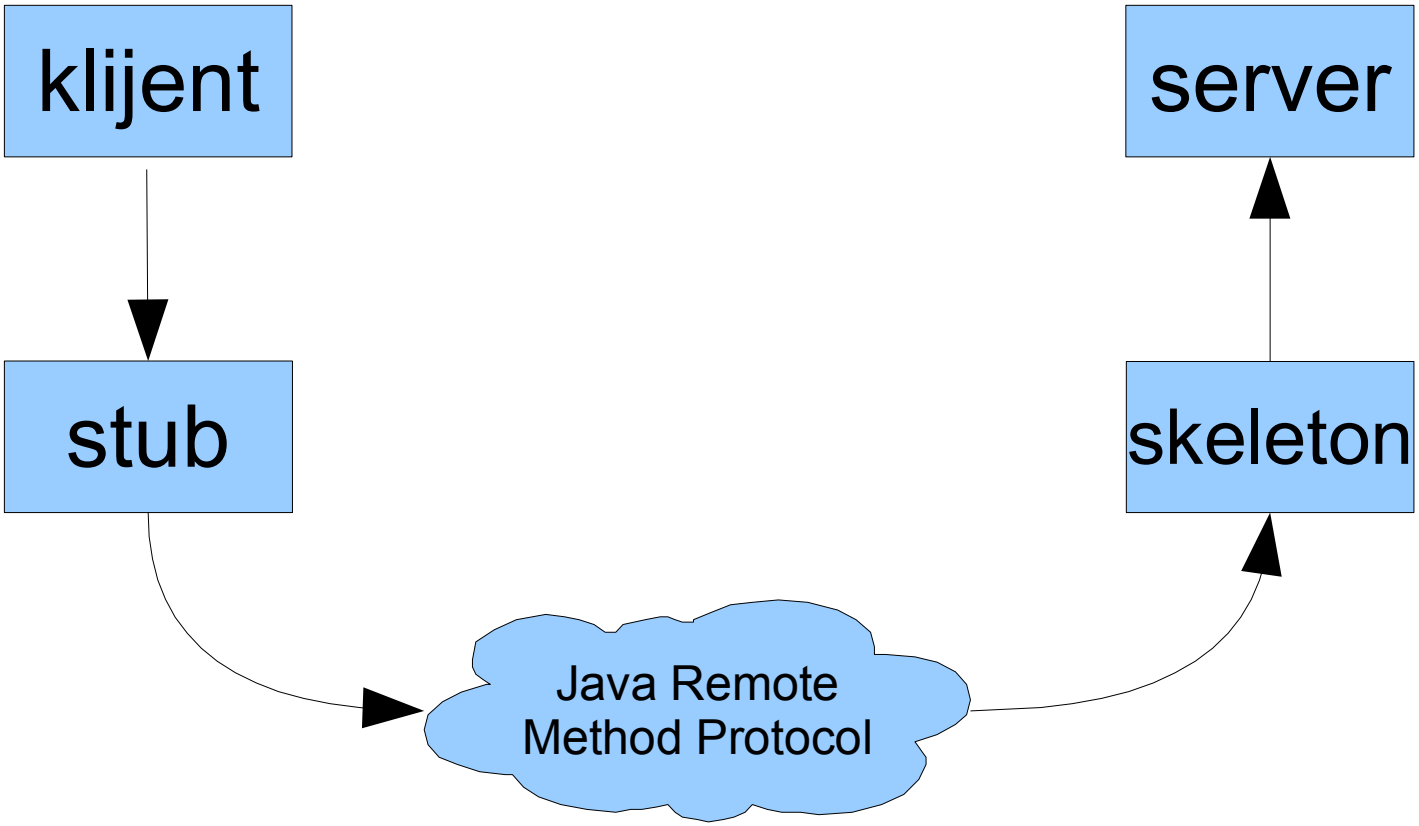
Remote Method Invocation (RMI)

- Osnovna tehnologija za rad sa distribuiranim objektima u Javi
- Objekti na udaljenim JVM su dostupni preko mreže
- Pozivamo njihove metode na isti način kao i za lokalne objekte
- Objekti se registruju u katalozima (registry)
- Klijenti ih tamo pronalaze

RMI registry, server i klijent



RMI klijent i server komuniciraju preko posrednika



Primer elementarnog RMI klijenta i servera

- Primer 1

RMI i prenos programskog koda

- Prilikom poziva metode RMI objekta...
- ...kao parametar možemo proslediti instancu klase koja nasleđuje tip parametra
- Tada će JVM preneti i programski kod potrebne klase!
- Primer 2

JNDI

- Java Naming and Directory Interface
- Standardni API za pristup različitim servisima imena
- I različitim direktorijumskim servisima
- Servis imena: mapira ime (string) $\leftrightarrow$ objekat
  - Fajl-sistem
  - RMI registry
- Direktorijumski servis: objekti se dodatno opisuju pomoću atributa
  - DNS
  - LDAP / Active Directory / NDS...
- Pristup različitim servisima obavlja se kroz isti API ali preko različitog „provajdera“ (tj. drajvera)
  - analogno sa JDBC

JNDI

- Primer 3

RMI + JNDI

- Umesto Naming.lookup() možemo da koristimo JNDI API za pronalaženje RMI objekta
- Primer 4

Java EE

- Enterprise JavaBeans (EJB) 3.0 — JSR-220
- JavaServer Pages (JSP) 2.1 — JSR-245
- JavaServer Faces (JSF) 1.2 — JSR-252
- JSP Standard Template Library (JSTL) 1.1 — JSR-52
- Java API for XML Binding (JAXB) 2.0 — JSR-222
- Java API for XML – Web Services (JAX-WS) 2.0 — JSR-224
- Web Service Annotations (WS Annotations) — JSR-181

EJB 3.0

- Programski model za pisanje distribuiranih komponenti
- Svrha komponenti:
  - Vrše programsku obradu (implementiraju „poslovnu logiku“) — session beans
  - Reprezentuju podatke u (relacionoj) bazi podataka — entities
  - Vrše programsku obradu uz asinhrono pozivanje — message-driven beans
- Distribuirane: dostupne preko mreže

EJB 3.0

- Temeljno prerađena specifikacija bazirana na prethodnim iskustvima
- Loša iskustva sa EJB 2.1
- Dobra iskustva iz različitih (open source) projekata
  - Hibernate: O/R mapiranje „urađeno kako treba“
  - Spring: životni ciklus, dependency injection, AOP
- upotreba anotacija eliminiše XML konfiguracione fajlove

EJB 3.0: Session bean

- Session bean se sastoji iz
  - remote i/ili lokalnog interfejsa
  - bean klase
- Klijent ga pronalazi pomoću JNDI-a
- i poziva njegove metode

Dve vrste session beanova

- Stateless: ne pamti stanje između poziva svojih metoda
  - bean klasa može imati atribute ali se ne garantuje za njihov sadržaj prilikom sledećeg poziva!
- Stateful: pamti stanje između poziva

Stateless session bean

- Primer 5

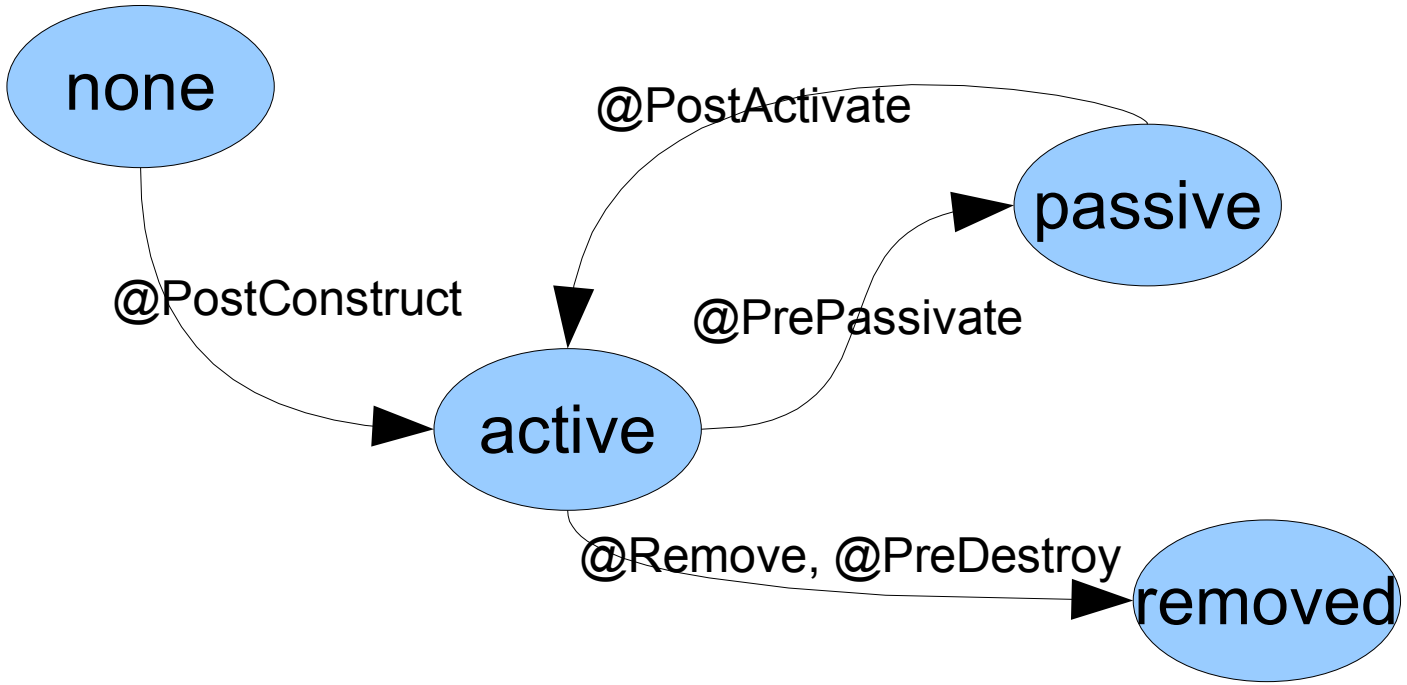
Stateful session bean

- Primer 6

Stateless vs stateful: performanse

- Stateless
  - jednostavni za pooling, zaključavanje na nivou poziva metode
- Stateful
  - komplikovani za pooling, zaključavanje na nivou celog objekta

Životni ciklus session beana



- Primer 7

Session bean poziva drugi session bean

- Prvi SB se ponaša kao klijent za drugi SB
- Ako se nalaze u istom kontejneru, može da koristi lokalni interfejs
- Pronalazi ga preko JNDI konteksta
- Inicijalni kontekst se konstruiše bez parametara
- Primer 8

Session bean i dependency injection

- Drugi način da jedan SB dobije referencu na drugi je pomoću dependency injection mehanizma
- Referencu na drugi SB upisuje **kontejner** u atribut prvog SB
- Atribut je potrebno označiti anotacijom **@EJB**
- Primer 9

Session bean i dependency injection

- Dependency injection je moguć u sledećim slučajevima (bean A $\rightarrow$ (poziva) bean B)
  - stateless $\rightarrow$ stateless
  - stateful $\rightarrow$ stateless
  - stateful $\rightarrow$ stateful
- A zabranjen je u slučaju
  - stateless $\rightarrow$ stateful

Session bean i dependency injection

- Anotacijom **@Resource** mogu da se označe atributi sledećeg tipa
  - javax.ejb.SessionContext
  - javax.sql.DataSource
  - javax.transaction.UserTransaction
  - javax.jms.Queue, javax.jms.Topic
  - ...
- Anotacijom **@PersistenceContext** označava se atribut tipa
  - javax.persistence.EntityManager
- Anotacijom **@PersistenceContextFactory** označava se atribut tipa
  - javax.persistence.EntityManagerFactory

Aspekt-orijentisano programiranje (AOP)

- Sredstvo za izražavanje određenih pravila/procedura koja se mogu primeniti na više mesta u programu
- Npr. logovanje, kontrola pristupa, ...– aspekti programa koji nisu direktno vezani za poslovnu logiku i često izgledaju isto za različite poslovne procedure
- Aspekt je parče koda koji se može vezati za neku metodu tako da se izvrši
  - pre poziva metode
  - posle poziva metode
  - oko poziva metode (obuhvata poziv)

Session beans i AOP

- U EJB 3.0 aspekti mogu da **obuhvate** poziv metode
- Metodu je potrebno označiti anotacijom **@Interceptor**
- Ako ima više aspekata onda **@Interceptors**
- Parametar ove anotacije je klasa koja sadrži aspekt
- Aspekt je metoda u klasi označena anotacijom **@AroundInvoke**
- Primer 10

Interceptori i životni ciklus beana

- Klasa navedena kao @Interceptor može sadržati i metode za obaveštavanje o događajima u životnom ciklusu
- U primeru 7 te metode su bile deo bean klase
- Možemo ih premestiti u interceptor klasu
- Koristimo iste anotacije: @PostConstruct, @PrePassivate, @PostActivate, @PreDestroy
- Primer 11

Pristup relacionim bazama podataka

- standardan API koji omogućava pristup relacionim bazama podataka – **JDBC**
- klase i interfejsi u paketu `java.sql`
- komunikacija putem SQL-a
- API implementira konkretna biblioteka za konkretnu bazu – MySQL, Oracle, ...
- promena baze (npr. MySQL $\rightarrow$ PostgreSQL) ne [bi trebalo da] zahteva promenu našeg koda

Korak 1: inicijalizacija drajvera

```
Class.forName("com.mysql.jdbc.Driver");
```

Korak 2: otvaranje konekcije

```
import java.sql.Connection;
...
Connection conn = DriverManager.getConnection(
  "jdbc:mysql://localhost/isa",   // JDBC URL
  "isa",                          // username
  "isa");                         // password
```

Korak 3a: kreiranje i izvršavanje SQL naredbe

```
import java.sql.Statement;
...
Statement stmt = conn.createStatement();
stmt.executeUpdate(
  "INSERT INTO nastavnici ('Ana', 'Tot', 'docent'");
stmt.close();
```

Korak 3b: kreiranje i izvršavanje SQL upita

```
import java.sql.Statement;
import java.sql.ResultSet;
...
Statement stmt = conn.createStatement();
ResultSet rset = stmt.executeQuery(
  "SELECT ime, prezime, zvanje FROM nastavnici");
while (rset.next()) {
  rset.getString(1); // ime
  rset.getString(2); // prezime
  rset.getString(3); // zvanje
}
rset.close();
stmt.close();
```

Korak 4: zatvaranje konekcije

```
conn.close();
```

- TCP konekcija je otvorena sve vreme dok je JDBC konekcija otvorena!
- konekciju držimo otvorenu sve vreme rada sa bazom podataka
- tipično se više naredbi se izvrši kroz jednu konekciju

Primer postavljanja upita

- primer 12 → isa.pr12.Db1

Ponavljanje istih SQL naredbi

- izvršavanje SQL naredbe na serveru podrazumeva pripremne radnje
- parsiranje komande, kreiranje plana izvršavanja, itd.
- u slučaju ponovljenih istih komandi te pripremne radnje se takođe ponavljaju
- to je čist višak

PreparedStatement

- najavljuje serveru baze podataka izvršavanje jedne naredbe
- server obavlja pripremu za njeno izvršavanje
- šalje podatke za prvu 1. naredbe i izvršava je
- šalje podatke za prvu 2. naredbe i izvršava je
- šalje podatke za prvu 3. naredbe i izvršava je
- ...
- (znatno efikasnije!)

- primer 12 $\rightarrow$ isa.pr12.Db2

Uskladištene procedure

- stored procedure je potprogram napisan jeziku koji predstavlja proceduralno proširenje SQL-a
- može se pozvati sa klijenta
- njihova upotreba znatno smanjuje mrežni saobraćaj između klijenta i servera baze podataka

Primer uskladištene procedure

```
CREATE PROCEDURE povezi(
    IN ime_ VARCHAR(25),
    IN prezime_ VARCHAR(35),
    IN naziv_ VARCHAR(150))
BEGIN
  DECLARE nas_id INT;
  DECLARE pred_id INT;

  SELECT nastavnik_id INTO nas_id FROM nastavnici
    WHERE ime=ime_ AND prezime=prezime_;
  SELECT predmet_id INTO pred_id FROM predmeti
    WHERE naziv=naziv_;
  INSERT INTO predaje (nastavnik_id, predmet_id)
    VALUES (nas_id, pred_id);
END//
```

Primer pozivanja uskladištene procedure

- primer 12 → isa.pr12.Db3

Pristup bazi podataka iz servleta

- jedna instanca servleta opslužuje sve korisnike
- potencijalno više paralelnih niti
- otvaranje konekcije bi se moglo smestiti u **init()**
- zatvaranje konekcije u **destroy()**

- primer 12 $\rightarrow$ isa.pr12.Db4

Pristup bazi podataka iz servleta

- jedna ista konekcija ne sme se koristiti u više paralelnih niti!
- trebaće nam po jedna konekcija **za svaku nit**
- resource pooling slično kao za stateless bean-ove

- primer 13 $\rightarrow$ klasa **ConnectionPool**

JDBC je zamoran

- naš Java program rukuje objektima
- podatke iz objekata treba „prepisati“ u tabele u bazi podataka
- JDBC API je opširan

- primer 14

JDBC je zamoran, i dalje

- možemo da „sakrijemo“ JDBC pozive u klasu koja se snima u bazu
- korišćenje takve klase sada je jednostavnije
- ali i dalje neko mora da napiše taj kod

- primer 15

Java Persistence API (JPA)

- Standardan API koji omogućava snimanje POJO objekata u relacionu bazu
- API implementira neka konkretna biblioteka – Hibernate, TopLink, ...
- JPA-QL: upitni jezik, „objektna varijanta“ SQL-a
- O/R mapiranje se opisuje anotacijama
- Nema posebnih konfiguracionih fajlova (kao za klasičan Hibernate)
- JPA nije vezan za EJB kontejner – može da se koristi i za Java SE aplikacije!

Persistence Unit

- Persistence unit predstavlja jednu grupu perzistentnih klasa i parametara mapiranja
- Jedna aplikacija može raditi sa više persistence unita
- Persistence uniti se opisuju u fajlu META-INF/persistence.xml koji mora biti u CLASSPATH-u

EntityManagerFactory

- Na osnovu persistence unita opisanog XML fajlom u programu se kreira **EntityManagerFactory**
- Predstavlja in-memory reprezentaciju O/R mapiranja
- Thread-safe klasa
- Kreiranje je skupo

EntityManager

- Komunikacija sa bazom odvija se u sesijama
- Svaku sesiju opisuje jedan **EntityManager** objekat
- Kreira ga EntityManagerFactory
- Nije thread-safe
- Kreiranje nije skupo

EntityManager metode

- void persist(Object entity)
- T merge(T entity)
- void remove(Object entity)
- T find(Class<T> entityClass, Object primaryKey)
- Query createQuery(String query)
- EntityTransaction getTransaction()
- close()
- ...

JPA entity

- Entity je POJO klasa sa anotacijom **@Entity**
- Mora imati default konstruktor
- Najčešće se mapira 1 klasa $\leftrightarrow$ 1 tabela
- Atributi klase se mapiraju na kolone tabele
- Parametri mapiranja se opisuju anotacijama
- Anotacije se vezuju za atribute ili getter metode

JPA entity

- Entity ne mora da implementira Serializable
- Ako ga implementira, entitiji se mogu prenositi u druge slojeve aplikacije
- Poseban DTO (Data Transfer Object) nije potreban

JPA entity

- Primer 16
  - AdminTest: primer rukovanja entitijem pomoću EntityManagera
  - Admin: primer entity klase
  - persistence.xml: definicija persistence unita

Identitet u Javi

- Identitet objekta (lokacija u memoriji): `x == y`
- Jednakost objekata: `x.equals(y)`
  - da li su jednaka dva `User` objekta sa istim `username` a različitim `password`?

Identitet u bazi podataka

- Isti red (lokacija na disku)
- Vrednost primarnog ključa

Java identitet vs DB identitet

- Kada je Java identitet $\Leftrightarrow$ DB identitet?
- Kada je Java jednakost $\Leftrightarrow$ DB jednakost?

Primer 1

- U prvoj transakciji:
  `x = em.find(User.class, "mbranko");`
- U drugoj transakciji:
  `y = em.find(User.class, "mbranko");`
- Da li je `x == y` ?

Primer 2

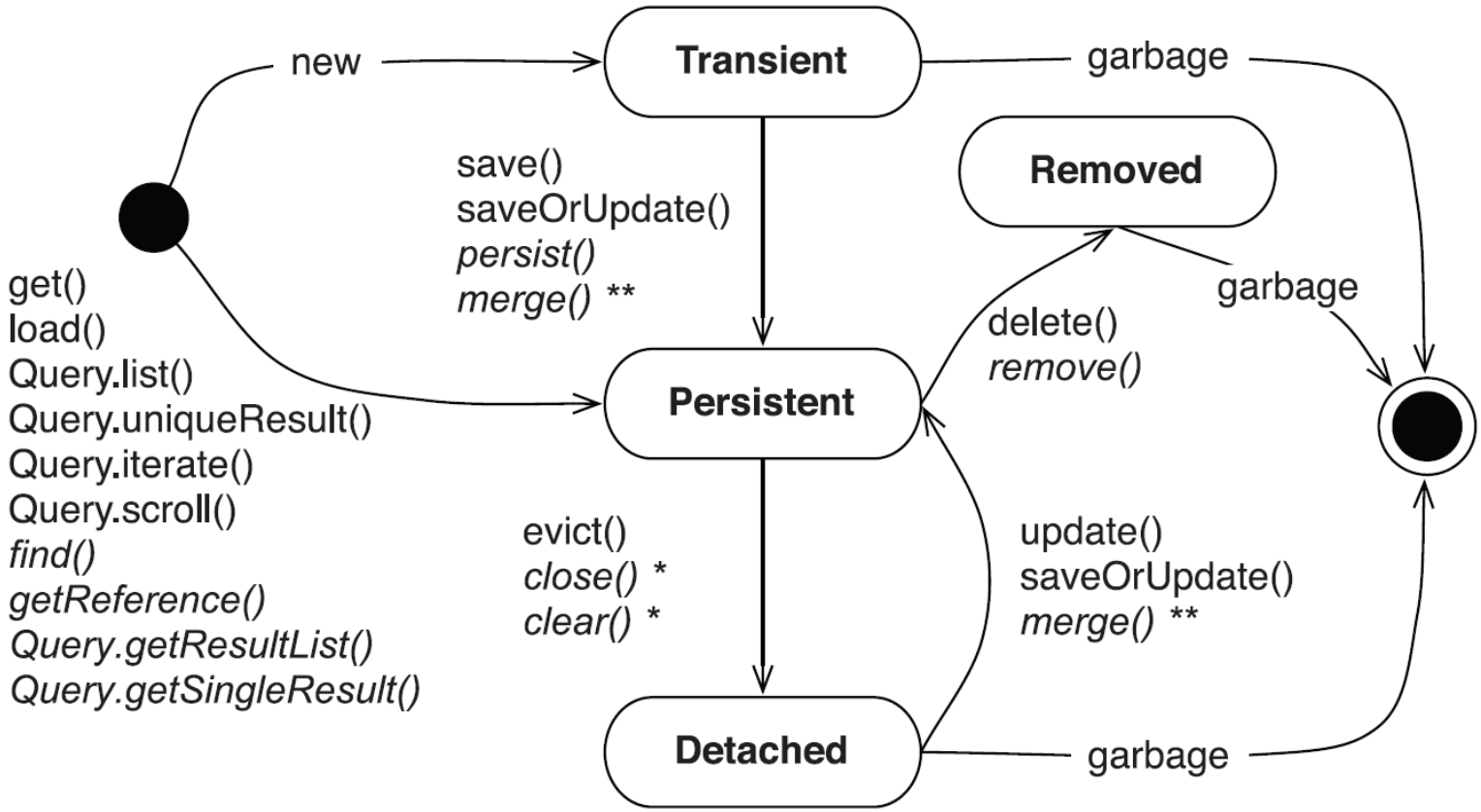
- U prvoj transakciji:
  ```
  x = em.find(User.class, "mbranko");
  ```
- U drugoj transakciji:
  ```
  y = em.find(User.class, "mbranko");
  y.setPassword("trt");
  ```
- Da li je `x.equals(y)` ?

JPA sesija

- Java identitet (i jednakost) važi za perzistentne objekte **unutar jedne sesije**!

```
EntityManager em = emf.createEntityManager();
...
em.close();
```

Životni ciklus entitija



Tipovi veza između entitija

- Posmatramo dve klase, A i B, koje su u vezi
- Veza tipa 1:1
  - klasa A sa atributom tipa B, anotacija @OneToOne
  - klasa B sa atributom tipa A, anotacija @OneToOne
- Veza tipa 1:n
  - 1-strana ima anotaciju @OneToMany, tip atributa je Set<B>
  - n-strana ima anotaciju @ManyToOne, tip atributa je A
  - n-strana obično ima i anotaciju @JoinColumn koja opisuje join uslov
- Veza tipa m:n
  - m-strana ima anotaciju @ManyToMany, tip atributa je Set<B>
  - n-strana ima anotaciju @ManyToMany, tip atributa je Set<A>
  - opciono i @JoinColumn

Jedno- i dvosmerne veze

- Jednosmerna veza: klasa A „vidi“ klasu B, a klasa B „ne vidi“ klasu A
- Dvosmerna veza: klasa A „vidi“ klasu B i obrnuto
- Prethodni slajd podrazumeva dvosmernu vezu
- Jednosmernu vezu pravimo izostavljanjem odgovarajućeg atributa u klasi

Uspostavljanje veze

- Važno pravilo: uspostavljanje veze između objekata mora da se vrši **kao da se ne koristi JPA**
- Ako je dvosmerna, veza mora da se ažurira sa obe strane

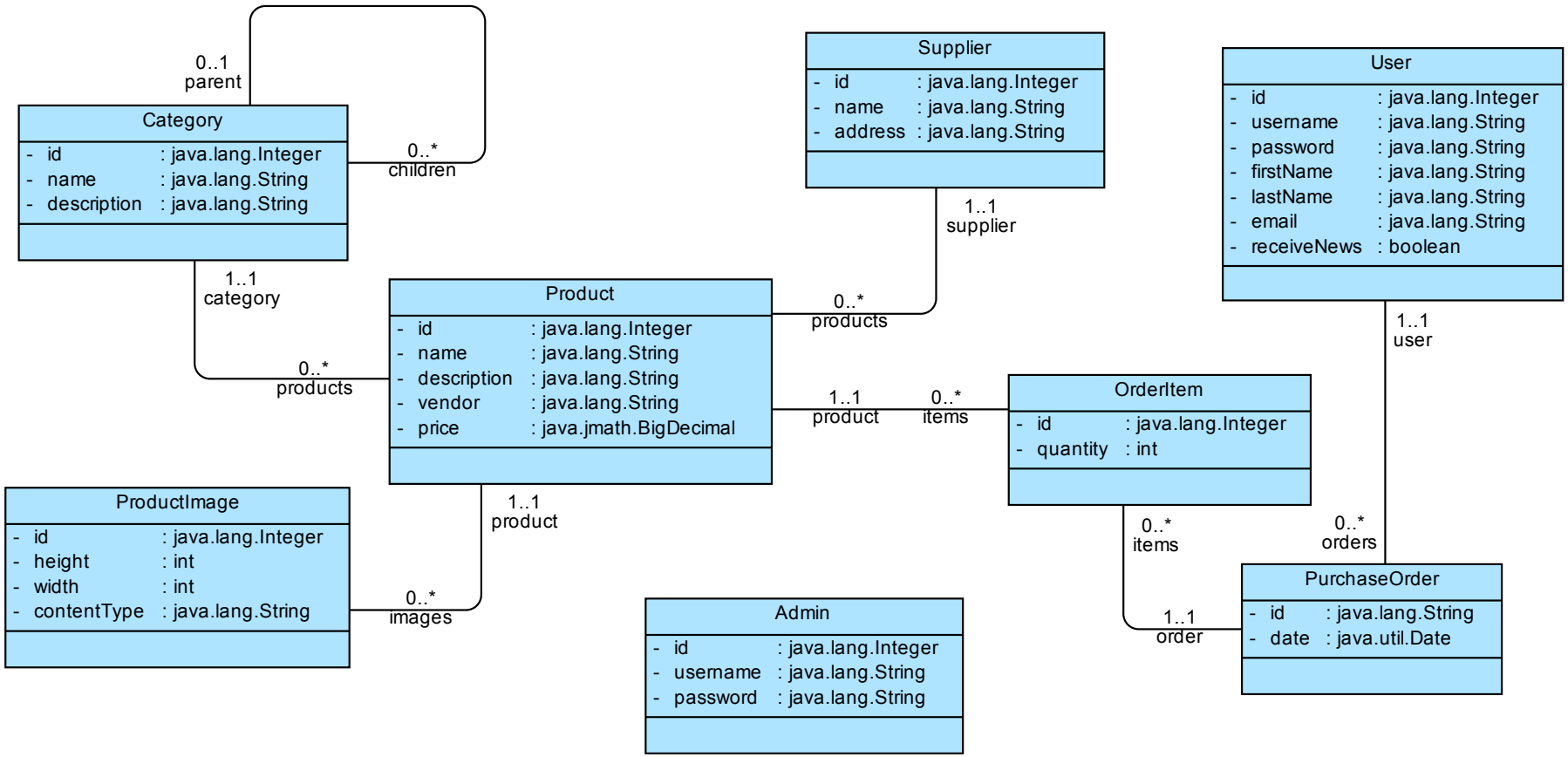
```
// oba reda su obavezna
product.setCategory(category);
category.getProducts().add(product);
```

Inicijalizacija atributa

- Atribut tipa Set<X> se mora inicijalizovati u prilikom konstrukcije objekta
- Obično se za inicijalizaciju koristi HashSet<X>
- JPA engine će taj kasnije taj objekat zameniti svojom Set implementacijom

```
class Category {
  ...
 private Set<Product> products = new HashSet<Product>();
  ...
}
```

Primer 17


Category
- id : java.lang.Integer
- name : java.lang.String
- description : java.lang.String
0..1
parent
0..*
children
1..1
category
0..*
products
Supplier
- id : java.lang.Integer
- name : java.lang.String
- address : java.lang.String
1..1
supplier
0..*
products
Product
- id : java.lang.Integer
- name : java.lang.String
- description : java.lang.String
- vendor : java.lang.String
- price : java.jmath.BigDecimal
1..1
product
0..*
items
OrderItem
- id : java.lang.Integer
- quantity : int
0..*
items
1..1
order
User
- id : java.lang.Integer
- username : java.lang.String
- password : java.lang.String
- firstName : java.lang.String
- lastName : java.lang.String
- email : java.lang.String
- receiveNews : boolean
1..1
user
0..*
orders
PurchaseOrder
- id : java.lang.String
- date : java.util.Date
ProductImage
- id : java.lang.Integer
- height : int
- width : int
- contentType : java.lang.String
1..1
product
0..*
images
Admin
- id : java.lang.Integer
- username : java.lang.String
- password : java.lang.String

Dodavanje novog elementa u Set

- Set ne prihvata duplikate
- Prilikom dodavanja novog elementa, proverava se da li je element već tamo
- Provera se oslanja na `equals()` i `hashCode()`
- ...a oni su nasleđeni iz klase `Object` i ne rade kako treba!

equals() za entitije

- `Object.equals()` $\Leftrightarrow$ `==`
- Dva različita objekta u memoriji mogu predstavljati isti red u bazi!
- Treba redefinisati `equals()` tako da koristi primarni ključ u poređenju

```
public boolean equals(Object o) {
  return this.id.equals(o.id);
}
```

equals() za entitije

- `Object.equals()` $\Leftrightarrow$ `==`
- Dva različita objekta u memoriji mogu predstavljati isti red u bazi!
- Treba redefinisati `equals()` tako da koristi primarni ključ u poređenju

```
public boolean equals(Object o) {
  return this.id.equals(o.id);
}
```

- Međutim, `id` nije definisan pre nego što se objekat snimi u bazu!

equals() za entitije

- Dodatak: ako je `id == null` za bilo koji od dva objekta, smatramo da su različiti

```
public boolean equals(Object that) {
  if (this == that)
    return true;
  if (this.id == null || that.id == null)
    return false;
  return this.id.equals(other.id);
}
```

equals() za entitije

- Dodatak: ako je `id == null` za bilo koji od dva objekta, smatramo da su različiti

```
public boolean equals(Object that) {
  if (this == that)
    return true;
  if (this.id == null || that.id == null)
    return false;
  return this.id.equals(other.id);
}
```

- Sledeći problem: ako su dva objekta jednaka, moraju imati isti `hashCode()`
- Pri tome vrednost `hashCode()` ne sme da se menja – izgubićemo objekte u Setu

hashCode() za entitije

```
private Integer hashcodeValue = null;
public int hashCode(){
  if (hashcodeValue == null) {
    if (id == null)
      hashcodeValue = new Integer(super.hashCode());
    else
      hashcodeValue = id;
  }
  return hashcodeValue.intValue();
}
```

- Kada se jednom upotrebi `hashCode()`, više se neće menjati

hashCode() za entitije

```
private Integer hashcodeValue = null;
public int hashCode(){
  if (hashcodeValue == null) {
    if (id == null)
      hashcodeValue = new Integer(super.hashCode());
    else
      hashcodeValue = id;
  }
  return hashcodeValue.intValue();
}
```

- Kada se jednom upotrebi `hashCode()`, više se neće menjati
- Problem: napravimo novi objekat, snimimo ga, zatvorimo sesiju, kasnije učitamo objekat u novoj sesiji i dobijemo dva objekta za koje važi `a.equals(b)` ali je `a.hashCode() != b.hashCode()`

Drugo rešenje za equals() i hashCode()

- Za poređenje koristimo one atribute koji su po svojoj prirodi jedinstveni
- Npr. za klasu `User` atribut `username` je jedinstven i **ne menja se nakon što se inicijalizuje** (tj. korisnik ne može da promeni svoj username kada se jednom registruje)

```
public int hashCode(){
  return username.hashCode();
}
public boolean equals(Object that) {
  if (this == that)
    return true;
  if (that == null)
    return false;
  return this.username.equals(that.username);
}
```

Drugo rešenje za equals() i hashCode()

- Za poređenje koristimo one atribute koji su po svojoj prirodi jedinstveni
- Npr. za klasu `User` atribut `username` je jedinstven i **ne menja se nakon što se inicijalizuje** (tj. korisnik ne može da promeni svoj username kada se jednom registruje)

```
public int hashCode(){
  return username.hashCode();
}
public boolean equals(Object that) {
  if (this == that)
    return true;
  if (that == null)
    return false;
  return this.username.equals(that.username);
}
```

Idealno rešenje za equals() i hashCode()

- Ne postoji
- Sve zavisi od načina upotrebe entitija
- Ako imamo atribut(e) sa jedinstvenim vrednostima, druga varijanta je najbolja
- Ako ih nemamo, prva varijanta može biti dovoljno dobra
- Treća varijanta: ne redefiniši `equals()` i `hashCode()` i pazi šta radiš
- Četvrta varijanta: GUID koji se inicijalizuje kod kreiranja objekta i koristi za `equals()` i `hashCode()` – može kao dodatni atribut ili čak primarni ključ
- Dobra diskusija: `http://www.hibernate.org/109.html`

Pozivanje session beanova iz servleta

- Servlet može da uradi JNDI lookup i pronađe session bean
- A može da koristi i dependency injection
- Servlet je po prirodi stateless, pa nema smisla injektovati stateful beanove

- Primer 18

Arhitektura web aplikacije koja koristi EJB

- Klijent: servleti+JSP stranice
- Servleti pristupaju session beanovima
  - SLSB se injektuje u atribut servleta
  - SFSB se pronađe preko JNDI i smesti u HttpSession
- Session beanovi pristupaju entitijima preko EntityManagera
  - EntityManager se injektuje u atribut SB-a

Data Access Object (DAO) sloj

- U praksi su za svaki entity potrebne uobičajene CRUD (create, retrieve, update, delete) operacije
- Njih obično implementiraju posebne DAO klase
- Jedan entity – jedan DAO
- Ima dosta „pešačkog“ posla

Generički DAO: implementacija zajedničkih operacija

```
public interface GenericDao<T, ID extends Serializable> {
  public Class<T> getEntityType();
  public T findById(ID id);
  public List<T> findAll();
  public List<T> findBy(String query);
  public T persist(T entity);
  public T merge(T entity);
  public void remove(T entity);
  public void flush();
  public void clear();
}
```

Generički DAO: implementacija zajedničkih operacija

```
public abstract class GenericDaoBean<T, ID extends Serializable>
    implements GenericDao<T, ID> {
  ...

  @PersistenceContext
  protected EntityManager em;
  ...
}
```

Konkretni DAO za entity User

```
public interface UserDao extends GenericDao<User, Integer> {
  public User login(String username, String password);
}
```

```
@Stateless
@Local(UserDao.class)
public class UserDaoBean extends GenericDaoBean<User, Integer>
    implements UserDao {

  public User login(String username, String password) { ... }
}
```

Primer 19

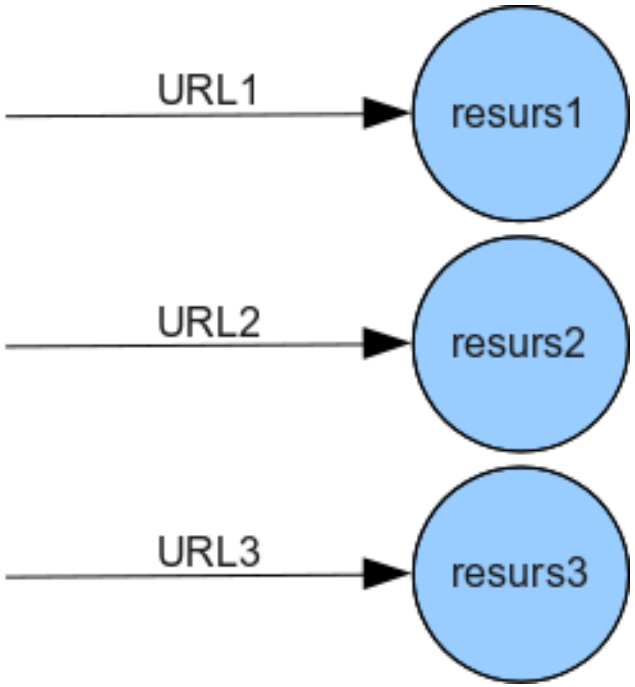
- Entity klase — isa.pr19.entity.*
- DAO klase — isa.pr19.dao.*
- SB klase — isa.pr19.session.*
- servleti — isa.pr19.servlet.*
- JSP stranice — isa.pr19.*

REST: REpresentational State Transfer

- Roy Fielding: „Architectural Styles and the Design of Network-based Software Architectures“
  - PhD rad sa University of California, Irvine, 2000.
- definiše principe softverske arhitekture za web
- alternativa za razvoj web servisa u odnosu na standardni SOAP+WSDL+...
- autor je učestvovao u razvoju:
  - HTTP (RFC 1945, RFC 2616, RFC 2145, RFC 2068)
  - URI (RFC 2396, RFC 1808)
  - Apache HTTP Server

REST principi

- pojam **resursa**: svaki entitet na webu je resurs. Npr. web sajt, HTML strana, XML fajl, web servis, fizički uređaj, ...
- **adresa** resursa: svaki resurs je identifikovan svojim URI-jem
- nad resursima se obavljaju jednostavne **operacije**

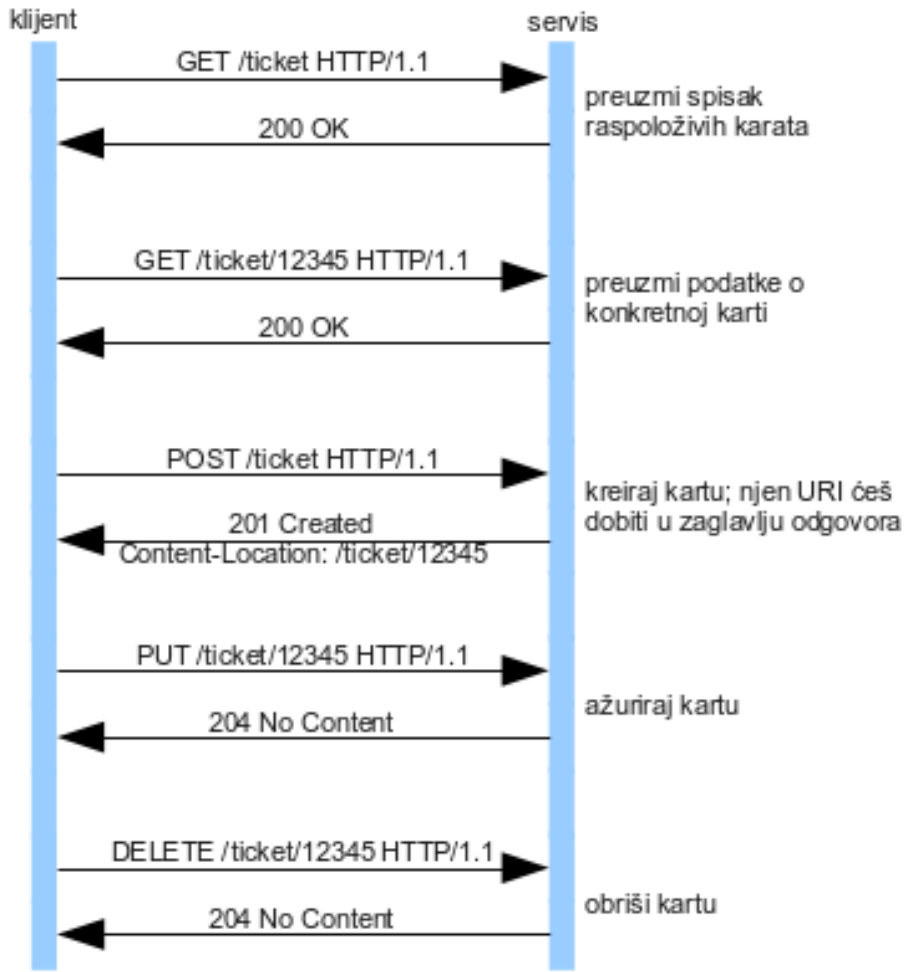


REST principi

- jednostavne operacije nad resursima kao HTTP metode:
  - GET – čitanje
  - POST – kreiranje
  - PUT – ažuriranje
  - DELETE – brisanje

Primer REST komunikacije: kupovina avionskih karata

- resurs = avionska karta
- HTTP metoda jasno označava operaciju:



Šta znači „representational state transfer“?

- klijent se obraća resursu putem URI-ja
- dobija **reprezentaciju** resursa
- ta reprezentacija pomera klijenta u novo **stanje**
- klijent se zatim obraća drugom resursu, itd.
- seoba klijenta iz stanja u stanje = **transfer**

Motiv za razvoj RESTa

- definisanje dizajn šablona koji opisuje kako bi web trebalo da radi
- tako da predstavlja okvir za razne web standarde
- i dizajn web servisa

REST nije standard

- W3C ga neće propisati kao standard
- IBM/Oracle/Microsoft/itd neće prodavati REST razvojne alate
- REST stimuliše **upotrebu** standarda:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/itd. (formati za reprezentaciju resursa)

REST + XML/JSON

- nije obavezno koristiti XML za podatke
- možemo koristiti i HTML, ...
- ali ako nam trebaju **machine-readable** podaci, XML ili JSON su najzgodniji
- pri tome, niko ne nameće neku posebnu šemu za podatke, niti format poruka!

Povezivanje podataka

- podaci koje vraća web servis treba da sadrže linkove ka drugim podacima
- $\rightarrow$ dizajn podataka kao mreže informacija
- nasuprot tome, OO dizajn promoviše enkapsulaciju

Primer RESTful servisa: studentska služba

- neka je naš web servis dostupan na adresi `http://www.ftn.uns.ac.rs`
- neka je opis jednog studenta u JSON-u ovakav:

```
{
  "name": "Žika",
  "age": 20
}
```

Lista studenata kao XML resurs

```
[
  {
    "name": "Žika",
    "age": 20
  },
  {
      "name": "Laza",
      "age": 21
  }
]
```

Pristup podacima

- na adresi `http://www.ftn.uns.ac.rs/students` je lista studenata
- na adresi `http://www.ftn.uns.ac.rs/students/<ime>` je konkretan student

Čitanje podataka o studentu: zahtev

```
GET /students/Žika HTTP/1.1
Host: www.ftn.uns.ac.rs
Date: Fri, 20 May 2011 12:00:00 GMT
Accept: application/json
```

Čitanje podataka o studentu: odgovor

```
HTTP/1.1 200 OK
Date: Fri, 20 May 2011
Server: Apache 2.2.0
Content-Length: 123
Connection: close
Content-Type: application/json

{
  "name": "Žika",
  "age": 20
}
```

Dodavanje novog studenta: zahtev

```
POST /students HTTP/1.1
Host: www.ftn.uns.ac.rs
Date: Fri, 20 May 2011 12:00:00 GMT
Accept: text/xml
Content-Length: 123
Content-Type: application/json

{
  "name": "Pera",
  "age": 20
}
```

Dodavanje novog studenta: odgovor

```
HTTP/1.1 201 Created
Date: Fri, 20 May 2011 12:00:00 GMT
Location: http://www.ftn.uns.ac.rs/students/Pera
Content-Length: nnn
Content-Type: application/json

{
  "name": "Pera",
  "age": 20
}
```

Pojedinačni resursi i kolekcije resursa

- URI za kolekciju:
  `http://www.ftn.uns.ac.rs/students`
- URI za pojedinačni resurs:
  `http://www.ftn.uns.ac.rs/students/<ime>`
- operacije nad različitim vrstama URI-ja imaju različito značenje!

Pojedinačni resursi i kolekcije resursa

	Collection URI	Element URI
GET	Izlista URI-je i eventualno druge podatke o elementima kolekcije	Dobija reprezentaciju elementa kolekcije u obliku odgovarajućeg MIME tipa
POST	Kreira novi element kolekcije; URL novog elementa se vraća u odgovoru	Tretira dati element kao kolekciju i kreira novi element u njoj
PUT	Zameni celu kolekciju novom	Zameni element novim; ako ne postoji, kreira ga
DELETE	Uklanja celu kolekciju	Uklanja dati element kolekcije

Java klijent za RESTful web servis

- potrebni sastojci:
  - rukovanje HTTP konekcijama
  - parsiranje XML-a

Twitter klijent

```
URL twitter = new URL(
  "http://twitter.com/statuses/public_timeline.xml");
URLConnection tc = twitter.openConnection();
BufferedReader in = new BufferedReader(
  new InputStreamReader(tc.getInputStream(), "UTF8"));
```

Odgovor Twittera

```
<?xml version="1.0" encoding="UTF-8"?>
<statuses type="array">
<status>
  <created_at>Fri May 20 18:49:46 +0000 2011</created_at>
  <id>71648829237248000</id>
  <text>So high school is done. LET THE PARTY BEGIN</text>
  <truncated>false</truncated>
  <favorited>false</favorited>
  <in_reply_to_status_id></in_reply_to_status_id>
  <in_reply_to_user_id></in_reply_to_user_id>
  <in_reply_to_screen_name></in_reply_to_screen_name>
  <retweet_count>0</retweet_count>
  <retweeted>false</retweeted>

  <user>
    <id>250394499</id>
    <name>Taylor Stricklin</name>
    <screen_name>TaylorStricklin</screen_name>
...
```

Odgovor Twittera u raznim formatima

- XML:
  `http://twitter.com/statuses/public_timeline.xml`
- JSON:
  `http://twitter.com/statuses/public_timeline.json`
- RSS:
  `http://twitter.com/statuses/public_timeline.rss`
- ATOM:
  `http://twitter.com/statuses/public_timeline.atom`

Korisne biblioteke

- Apache HttpComponents: precizna i detaljna implementacija HTTP protokola sa klijentske strane
- Apache Commons Codec: konverzija različitih formata

Klijent sa Apache bibliotekama

```
HttpClient client = new HttpClient();
GetMethod get = new GetMethod(
  "http://twitter.com/statuses/public_timeline.json");
int statusCode = client.executeMethod(get);
if (statusCode == HttpStatus.SC_OK) {
   ... method.getResponseBody() ...
```

RESTful servisi i Java

- Java API for RESTful Web Services: JAX-RS
- implementacije:
  - Jersey
  - Apache CXF
  - RESTEasy
  - Restlet
  - Apache Wink
- pisanje servisa pomoću anotiranih Java klasa

Resurs

- resurs = anotirana POJO klasa

```
@Path("/students")
public class Students {
...

@Path("/students/{username}")
public class Student {
...

@Path("/teachers/{username: [a-zA-Z]}")
public class Teacher {
...
```

Operacije

- operacije = anotirane metode u resurs klasi

```
@Path("/students")
public class Students {

  @GET
  public String handleGet() { ... }

  @POST
  public String handlePost(String payload) { ... }

  @PUT
  public String handlePut(String payload) { ... }

  @DELETE
  public String handleDelete() { ... }
...
```

URI promenljive

- prijem parametara iz URI-ja

```
@Path("/student/{username}")
public class Student {

  @GET
  public String handleGet(
      @PathParam("username") String username) { ... }

...
```

URI promenljive

- može i više parametara odjednom

```
@Path("/student/{gender}/{age}")
public class Student {

  @GET
  public String handleGet(
      @PathParam("gender") String gender,
      @PathParam("age") age) { ... }

...
```

Različiti formati podataka

- ista operacija može primiti podatke u različitim formatima

```
@Path("/student/{username}")
public class Student {

  @PUT
  @Consumes("application/xml")
  public String updateXML(String payload) { ... }

  @PUT
  @Consumes("application/json")
  public String updateJSON(String payload) { ... }

...
```

Različiti formati podataka

- ista operacija može **vratiti** podatke u različitim formatima

```
@Path("/student/{username}")
public class Student {

  @GET
  @Produces("application/xml")
  public String getXML() { ... }

  @GET
  @Produces("application/json")
  public String getJSON() { ... }

...
```

Prijem podataka iz HTML formi

- primer HTML forme:

```
<form action="users" method="POST">
  Name: <input type="text" name="name"/>
  Age: <input type="text" name="age"/>
  Address: <input type="text" name="address"/>
</form>
```

Prijem podataka iz HTML formi

```
@Path("/students")
public class Students {

  @POST
  @Consumes("application/x-www-form-urlencoded")
  public void addUser(
      @FormParam("name") String name,
      @FormParam("age") String age,
      @FormParam("address") String address) { ... }

...
```

Karakteristike operacija

- PUT i DELETE – idempotentne
  više identičnih zahteva daje isti rezultat
- GET – bezbedna (safe method)
  samo za čitanje; ne sme da menja stanje na serveru

Karakteristike operacija

- RESTful servisi su stateless – stanje je isključivo na klijentu
- transakcije su u nadležnosti klijenta

Primer REST API-ja

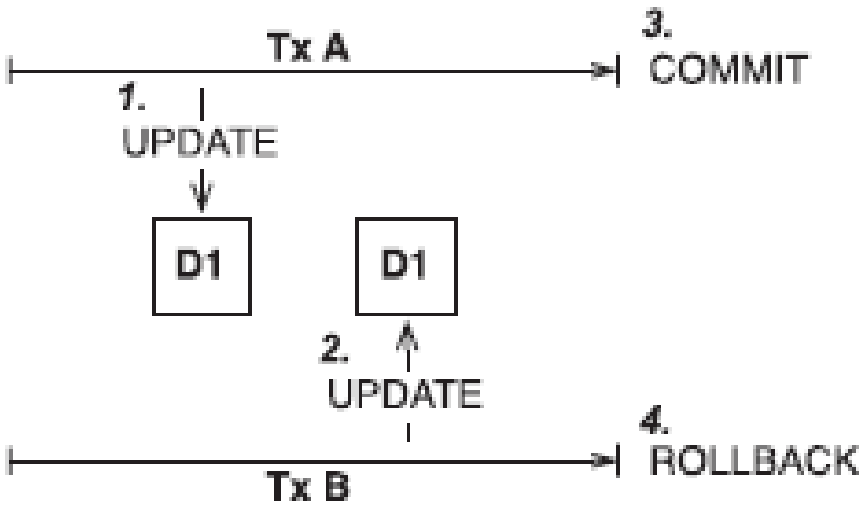
- isa.pr19.rest.*
- info: isa/pr19/readme.txt

Transakcije i konkurentni pristup podacima

- Prilikom istovremenog pristupa podacima može da dođe do štetnog preplitanja rada više transakcija
- Tom prilikom može da se javi više problema

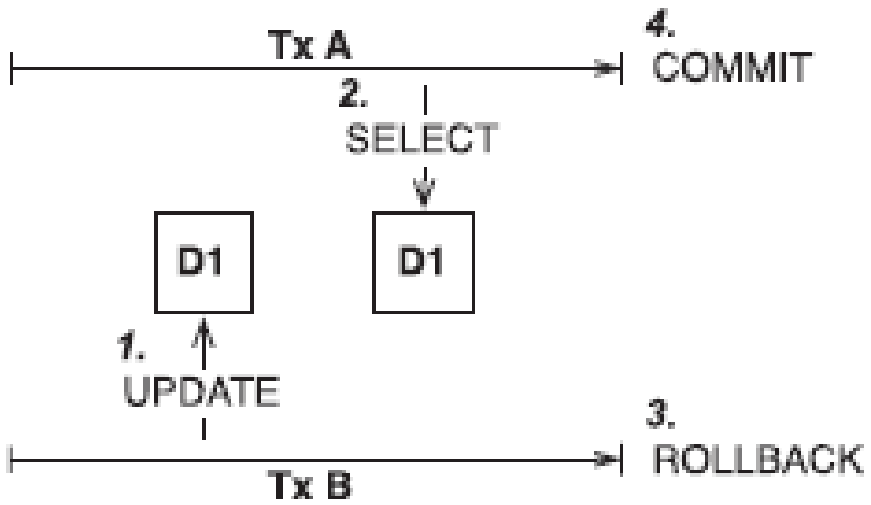
Lost update

- Lost update: dve transakcije menjaju isti podatak bez zaključavanja



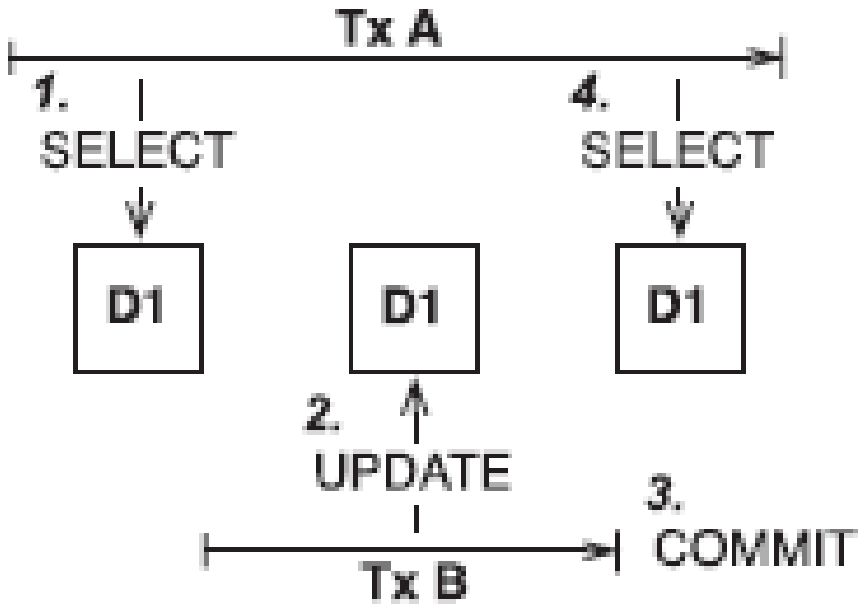
Dirty read

- Dirty read: transakcija A čita podatke pre nego što su commit-ovani



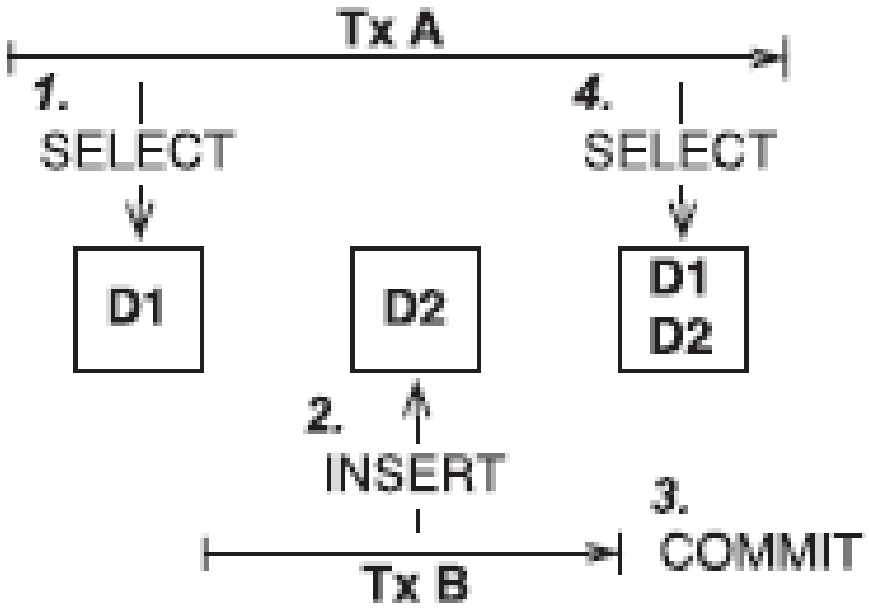
Unrepeatable read

- Unrepeatable read: transakcija A dva puta čita iste podatke i dobija različiti sadržaj



Phantom read

- Phantom read: transakcija A u drugom čitanju dobija i podatke kojih nije bilo prilikom prvog čitanja



Transakcije na nivou JDBC konekcije

- Transakcijama upravlja baza podataka
- Možemo da biramo nivo izolacije transakcija za svaku konekciju
- connection.setTransactionIsolation(...)

Nivo izolacije	**Eliminiše problem**
READ_UNCOMMITTED	lost update
READ_COMMITTED	dirty read
REPEATABLE_READ	unrepeatable read
SERIALIZABLE	phantom read

Ko upravlja transakcijama kod EJB komponenti?

- container-managed tx: transakcijama upravlja kontejner na osnovu anotacija dodeljenih metodama
- bean-managed tx: transakcijama programski upravlja bean (JTA API)
- client-managed tx: transakcijama programski upravlja klijent (JTA API)

Container-managed transakcije

- Anotacija **@TransactionAttribute**

Vrednost	**Značenje**
REQUIRED	metoda se priključuje tekućoj tx, otvara novu ako tx ne postoji
REQUIRES_NEW	metoda uvek pokreće novu tx, ako postoji tekuća tx ona se suspenduje
MANDATORY	metoda mora da se izvršava u tx, koja mora biti ranije pokrenuta; ako je nema javlja se greška
SUPPORTS	metoda će se priključiti tekućoj tx, ako ona postoji; ako ne postoji, izvršava se bez tx
NOT_SUPPORTED	metoda se izvršava bez tx, čak i ako postoji tekuća tx
NEVER	metoda se izvršava bez tx; ako postoji tekuća tx, javlja se greška

Container-managed transakcije

- Primer 20: isa.pr20.container.*

Bean-managed transakcije

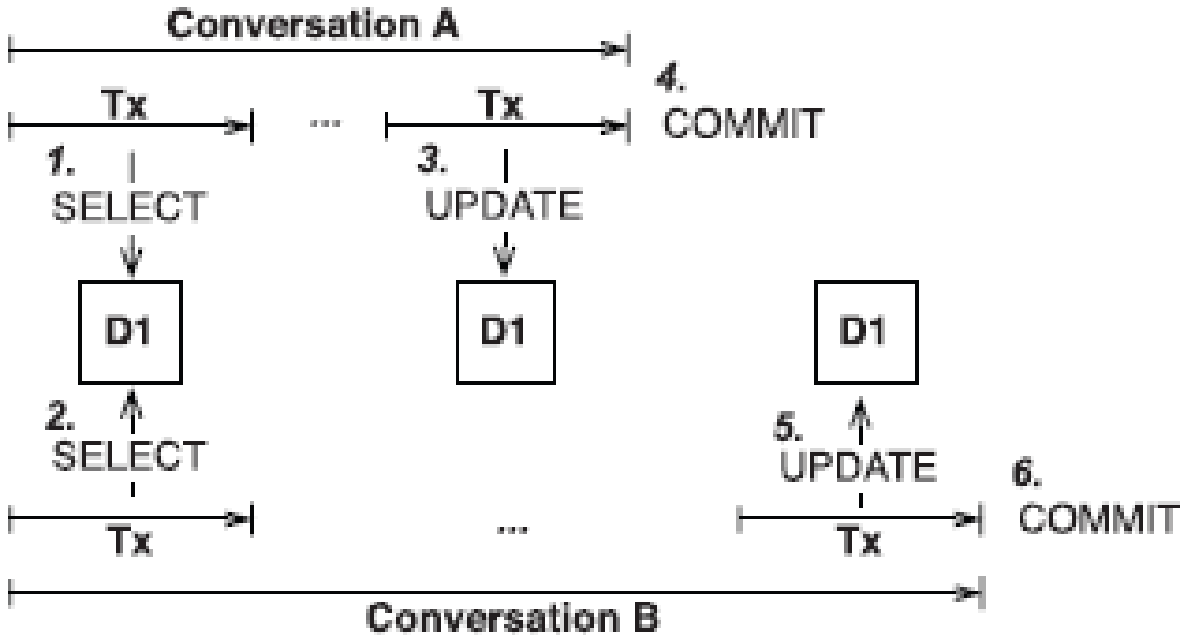
- Class-level anotacija **@TransactionManagement(BEAN)**
- Injekcija UserTransaction objekta pomoću @Resource anotacije
- Ručno pozivanje metoda
  - UserTransaction.begin()
  - UserTransaction.commit()
  - UserTransaction.rollback()
- Primer 20: isa.pr20.bean.*

Client-managed transakcije

- Klijent dobija UserTransaction preko JNDI lookup-a
- UserTransaction tx = (UserTransaction)ctx.lookup("java:comp/UserTransaction");
- Ručno pozivanje metoda
  - tx.begin()
  - tx.commit()
  - tx.rollback()
- Primer 20: isa.pr20.client.*
- (Beanovi koji se pozivaju su označeni kao bean-managed tx)

Optimističko i pesimističko zaključavanje

- Problem: operacija B će pregaziti izmene koje napravi operacija A, ne znajući za njih



Optimističko i pesimističko zaključavanje

- Rešenje 1 – pesimističko zaključavanje: svaka operacija treba da zaključa podatke **i za čitanje i za pisanje** sve dok se ne završi
  - u prethodnom primeru operacija B bi bila blokirana sve dok A ne otključa podatke
- Rešenje 2 – optimističko zaključavanje: svaka operacija pre izmene podataka treba da proveri da li je podatke neko drugi u međuvremenu menjao
  - poredi verziju podataka koje je pročitala sa onim što se trenutno nalazi u bazi
  - ovo poređenje mora da se izvodi u režimu pesimističkog zaključavanja
  - ako su podaci menjani, prijavi se greška korisniku

Optimističko i pesimističko zaključavanje

- Pesimističko zaključavanje garantuje ispravan rad
- Ali ima loše performanse
  - čak i ako dve transakcije pristupaju različitim redovima u tabeli može doći do blokiranja
- Optimističko zaključavanje polazi od pretpostavke da u praksi do kolizije dolazi jako retko
  - a situacije kada dođe do kolizije se otkrivaju i kontrola se vraća korisniku

Implementacija optimističkog zaključavanja

- Varijanta 1: poredimo sve vrednosti objekta sa vrednostima u bazi
  - nezgodno ako tabela ima puno kolona
- Varijanta 2: dodamo novu kolonu koja služi kao brojač izmena
  - na svaku izmenu u datom redu tabele inkrementiramo njenu vrednost

Optimističko zaključavanje i JPA

- Implementacija pomoću „varijante 2“
- Entity dobija još jedan atribut tipa int koji se označava anotacijom **@Version**
- Atribut se mapira na novu kolonu u tabeli
- Ako dođe do kolizije generiše se OptimisticLockException
- Primer 21 – isa.pr21.optimistic.*

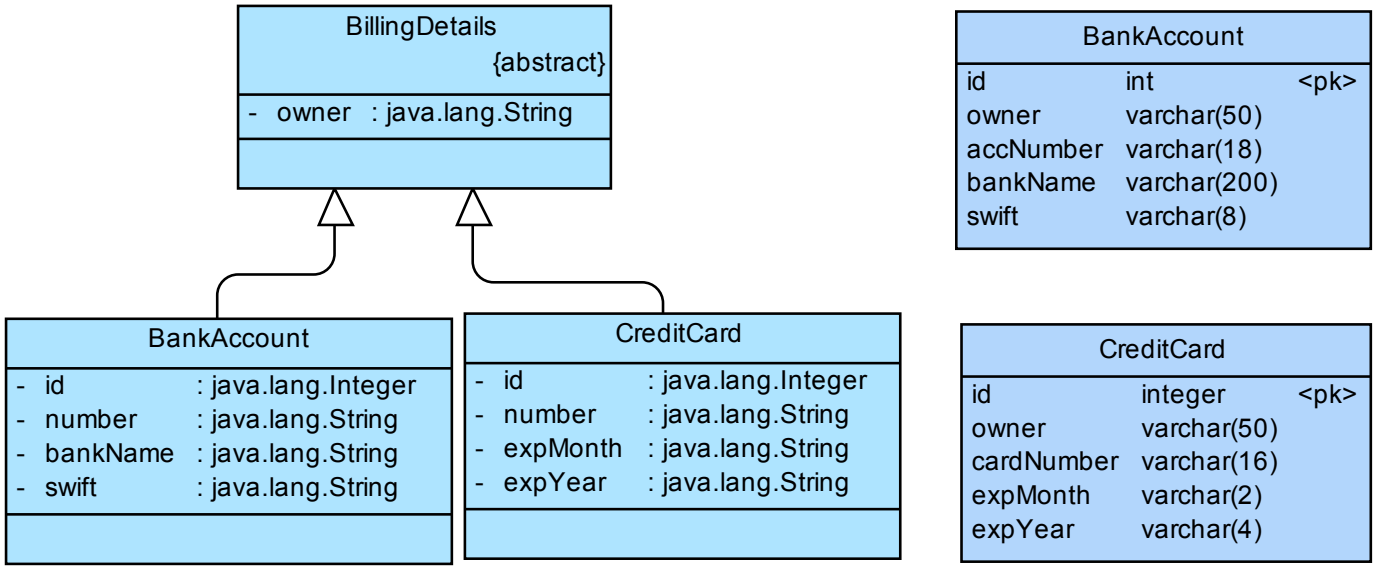
Pesimističko zaključavanje i JPA

- Učitani entity može da se zaključa za čitanje pomoću EntityManagera:
- em.lock(entity, READ);
- Entity je zaključan do kraja transakcije
- Druga transakcija koja proba da zaključa objekat dobiće izuzetak
- Primer 21 – isa.pr21.pessimistic.*

Četiri varijante mapiranja nasleđivanja

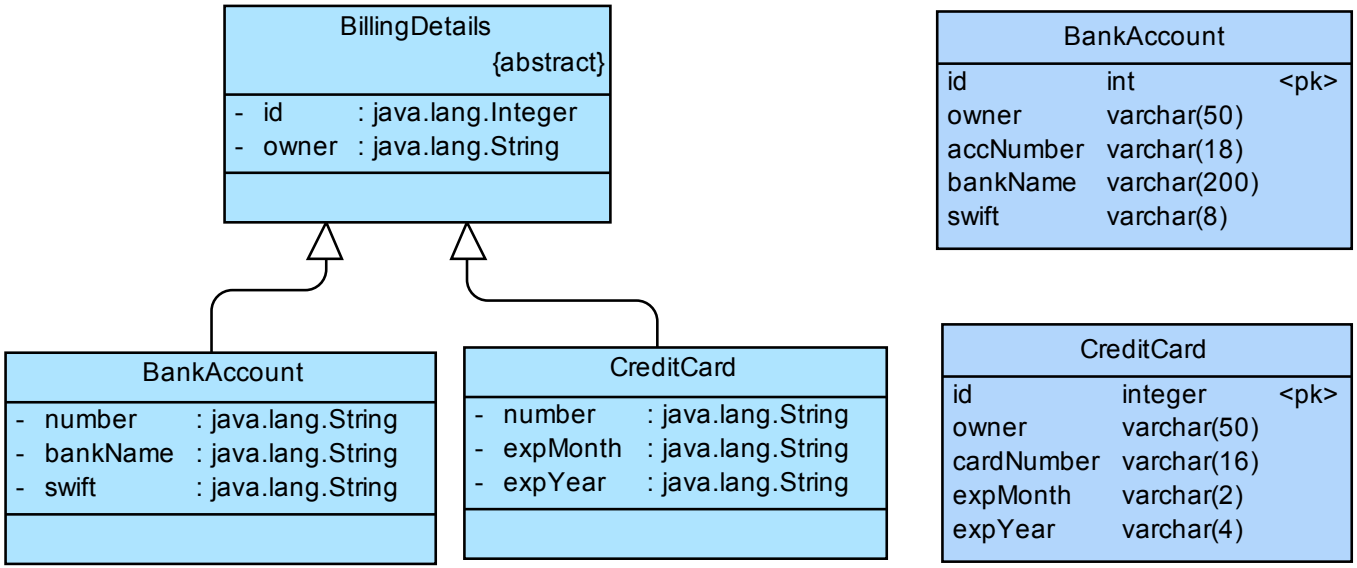
- Jedna tabela po konkretnoj klasi sa implicitnim polimorfizmom
- Jedna tabela po konkretnoj klasi
- Jedna tabela po hijerarhiji nasleđivanja
- Jedna tabela za svaku klasu

1 tabela po konkretnoj klasi sa implicitnim polimorfizmom



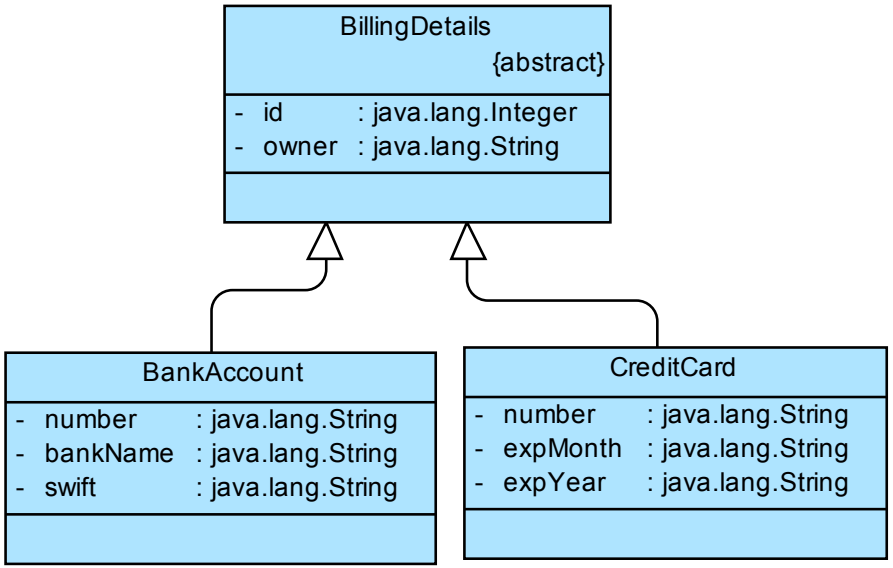
- Primer 22 – isa.pr22.v1.*

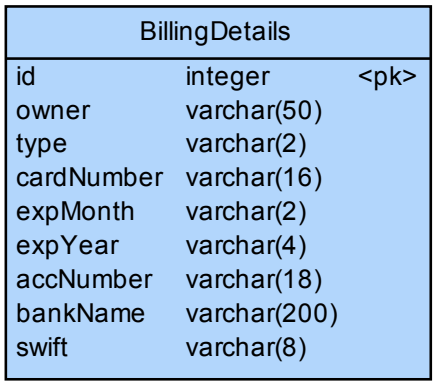
Jedna tabela po konkretnoj klasi



- Primer 22 – isa.pr22.v2.*

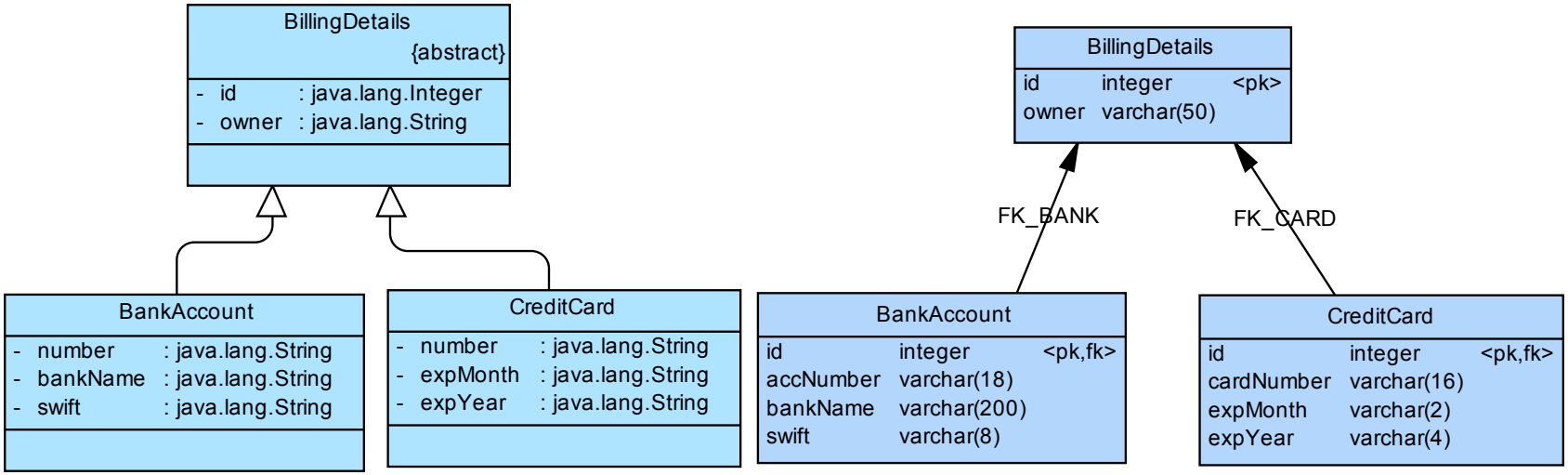
Jedna tabela po hijerarhiji nasleđivanja





- Primer 22 – isa.pr22.v3.*

Jedna tabela za svaku klasu



- Primer 22 – isa.pr22.v4.*

Šta biramo za primarni ključ?

- Neko prirodno obeležje koje je jedinstveno i nepromenljivo – prirodni ključ
  - JMBG, PIO broj, ...
  - može i grupa obeležja, npr. kontni okvir: šifra klase + šifra grupe + ...
- Veštačko obeležje koje je jedinstveno – surogatni ključ
  - integer brojač, UUID, ...
  - ključ čini uvek jedno obeležje

Prirodni vs surogatni ključ

- Prirodni ključevi

za
ne mora se izmišljati novo obeležje
protiv
obeležje nije baš nepromenljivo
ne mora biti integer tipa $\rightarrow$ manje efikasno indeksiranje

- Surogatni ključevi

za
efikasno indeksiranje
nema više od jednog obeležja u ključu
protiv
vrednost nema drugi smisao osim da bude jedinstvena

Prirodni i surogatni ključevi i JPA

- Za JPA se preporučuje upotreba surogatnih ključeva
- Znatno jednostavnije mapiranje
- Jednostavna provera da li treba raditi update (ključ $\neq$ null) ili insert (ključ $=$ null)

- Podržani su i prirodni ključevi
- Manje efikasan rad
- Manje elegantan objektni model u slučaju kompozitnih ključeva

Generisanje vrednosti surogatnih ključeva

- Identity / auto_increment / ...kolona u bazi
- Sekvenca
- Tabela sa brojačima

counter_name	counter_value
users	731
products	8432
...	...

- Primer 23 – isa.pr23.surrogate.*

JPA i prirodni ključevi

- Ako prirodni ključ čini jedno obeležje, on se označava sa @Id, kao i ranije
- Ako ima više obeležja u ključu, mora se napraviti posebna PK klasa
- Atribut tipa PK klase se dodaje u osnovnu klasu i označava sa @EmbeddedId

- Spoljni ključ koji se sastoji iz više obeležja se opisuje @JoinColumns anotacijom
- Osim ako je spoljni ključ deo primarnog ključa – tada se izražava u PK klasi
- Objektni model više nije elegantan!

- Primer 23 – isa.pr23.natural.*

Message-driven beans

- Komponente koje se pozivaju asinhrono – ne čeka se na rezultat izvršavanja
- Komunikacija klijenta i MDBa se odvija putem poruka
- Klijent šalje poruke MDBu, ovaj ih obrađuje kada stigne

- MDB je po prirodi stateless
- Treba implementirati jednu metodu – public void onMessage(Message msg)

Mehanizmi za distribuciju poruka

- Poruke od klijenta do MDBa mogu stići putem dva načina
- Queue: FIFO red poruka
  - svaka poruka konzumira se tačno jednom
  - MDBi za obradu poruka se zahvataju iz poola
  - iako se poruke dele u FIFO redosledu, prva poruka ne mora biti prva obrađena – ako je CPU vreme dobio drugi MDB
- Topic: svi pretplaćeni na jedan topic dobijaju sve poruke koje stižu u njega
  - jednu poruku može primiti više primalaca (različitih MDBa)
  - redosled obrade iste poruke nije definisan
- API za rad sa porukama definiše Java Message Service (JMS)

MDB i queue/topic

- Komunikacija preko **queue** mehanizma
- Primer 24 – isa.pr24.queue.*

- Komunikacija preko **topic** mehanizma
- Primer 24 – isa.pr24.topic.*

MDB i queue/topic

- Komunikacija preko **queue** mehanizma
- Primer 24 – isa.pr24.queue.*

- Komunikacija preko **topic** mehanizma
- Primer 24 – isa.pr24.topic.*

WebSocket

- RFC 6455
- dvosmerna komunikacija preko više kanala kroz jednu TCP vezu

Klasičan HTTP

- dizajniran za prenos dokumenata
  - interakcija zahtev/odgovor
- dvosmerna ali half-duplex komunikacija
  - samo u jednom smeru u jednom trenutku
- stateless
  - višak u headeru
  - podaci se šalju u svakom zahtevu i odgovoru

Full duplex vs half duplex

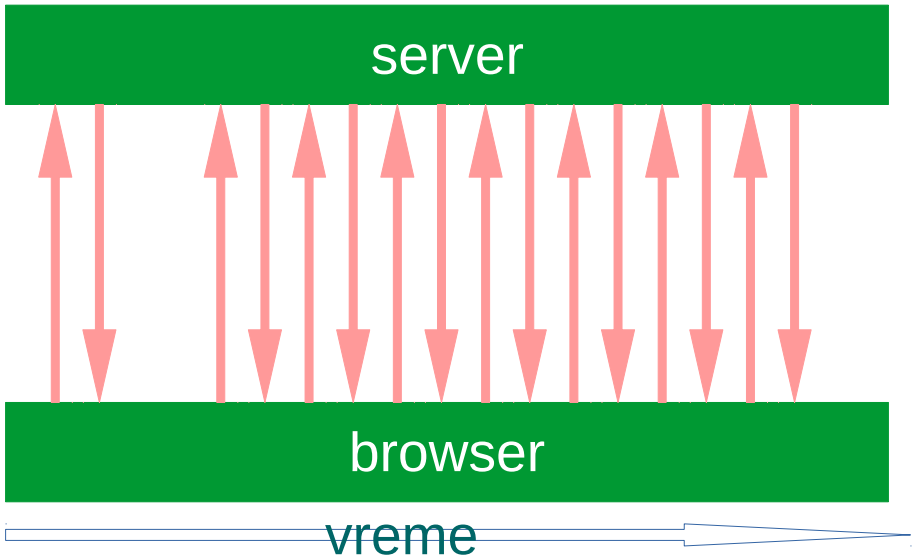
- full duplex: komunikacija u oba smera **istovremeno**
- half duplex: komunikacija u oba smera, ali samo u jednom smeru u jednom trenutku

AJAX folira full duplex vezu

- AJAX: Asynchronous JavaScript and XML
- sadržaj može da se menja bez osvežavanja cele stranice
- stvara utisak brzog odziva

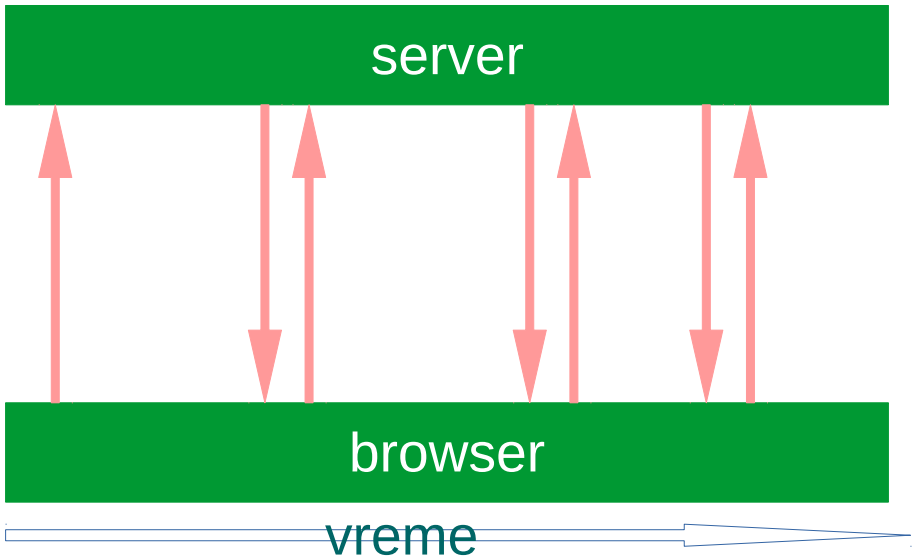
AJAX trik #1: polling

- nakon inicijalnog preuzimanja stranice
- browser šalje zahteve u regularnim intervalima
- i odmah prima odgovor
- skoro real-time



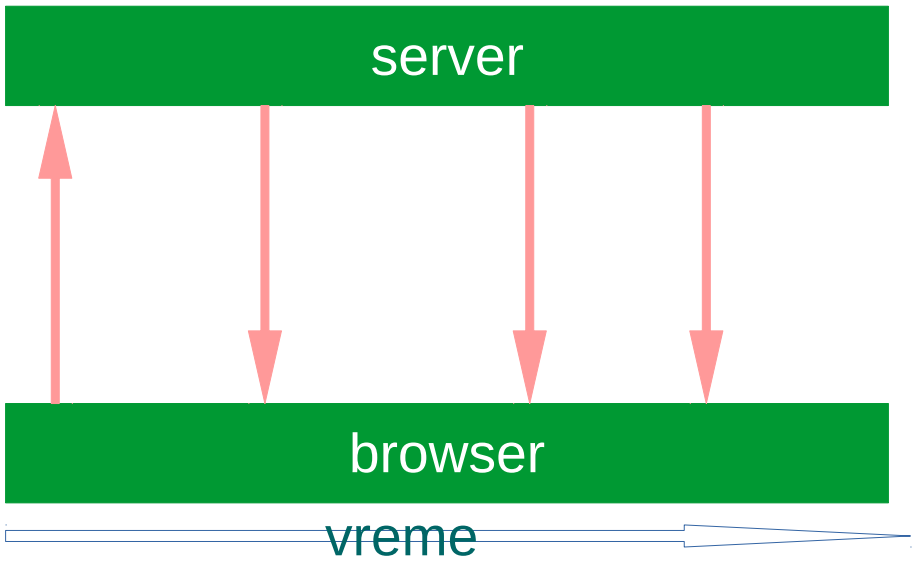
AJAX trik #2: long polling

- server kasnije šalje odgovor
- browser odmah šalje novi zahtev
- ciklus odgovor-zahtev-odgovor



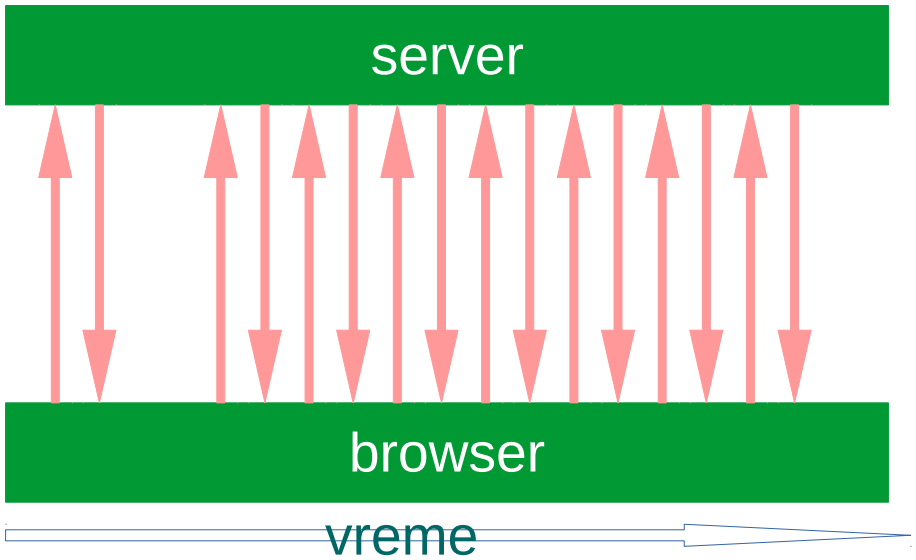
AJAX trik #3: streaming

- server šalje odgovor u delovima
- problemi sa proxy i firewall sistemima
- potreban periodičan flush
- ograničen broj otvorenih veza u browseru



AJAX folira full duplex vezu

- sadržaj može da se menja bez osvežavanja cele stranice
- stvara utisak brzog odziva
- polling je skoro real-time
  - browser šalje zahteve u regularnim intervalima
  - i odmah prima odgovor

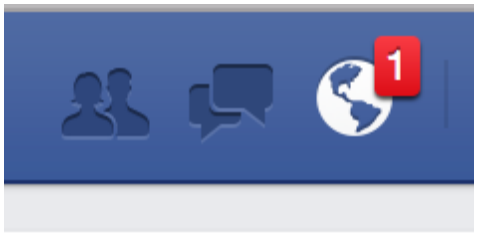


Višak u HTTP headeru

br.klijenata	bajtova viška	Mbps viška
1.000	871.000	6,6
10.000	8.710.000	66
100.000	87.100.000	665

WebSockets vs AJAX

- AJAX: otvara TCP vezu, šalje HTTP zahtev, uključuje rezultat u DOM stablo
- WebSocket: slanje podataka server→klijent bez prethodnog slanja zahteva klijent→server
- push notifications



WebSocket istorija

- originalno dodat u HTML5 kao `TCPConnection`
- kasnije izdvojen u posebnu specifikaciju
- W3C API i IETF protokol
- dve šeme: `ws://` i `wss://`

WebSocket API

```
if (window.WebSocket) { ... }

var ws = new WebSocket("ws://www.xyz.com")
ws.onopen = function(event) { };
ws.onclose = function(event) {
  alert("closed with status: " + event.code);
};
ws.onmessage = function(event) {
  alert("received msg: " + event.data);
};
ws.onerror = function(event) {
  alert("error");
};
// ...
ws.send("Hello world");
// ...
ws.close();
```

WebSocket API

dostupno?	`window.WebSocket` ili `Modernizr.websocket`
događaji	`onopen`, `onmessage`, `onclose`, `onerror`
funkcije	`send`, `close`
atributi	`url`, `readyState`, `bufferedAmount`, ...

Podrška u browserima

- Chrome 4+
- Safari 5+
- Firefox 4+
- Opera 10.7+
- Internet Explorer 10+

WebSocket server-side API

```
@ServerEndpoint("/echo")
public class EchoServer {

  @OnOpen
  public void onOpen(Session session){
    session.getBasicRemote().sendText("Hello!");
  }

  @OnMessage
  public void onMessage(String message, Session session){
    session.getBasicRemote().sendText("Echo: " + message);
  }

  @OnClose
  public void onClose(Session session){
  }
}
```

WebSocket handshake: zahtev klijenta

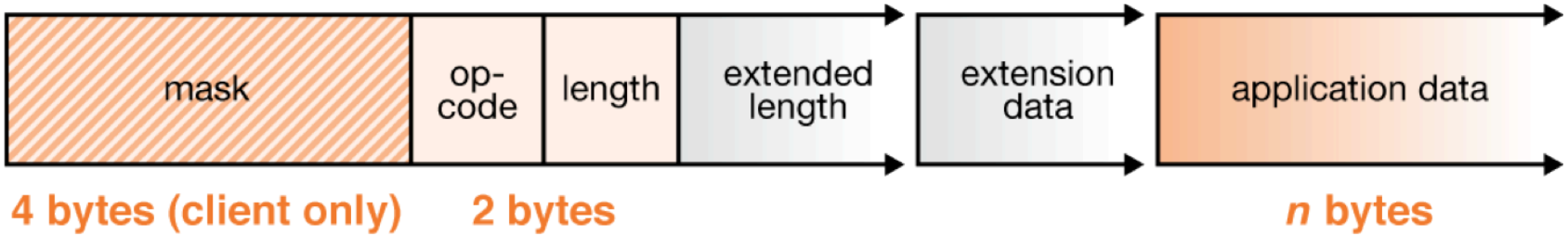
```
GET /chat HTTP/1.1
Host: xyz.com
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: x3JJHMbDL1EzLkh9GBhXDw==
Sec-WebSocket-Version: 13
--- (opciono) ---
Origin: http://xyz.com
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Extensions: ...
Cookie: ...
```

WebSocket handshake: odgovor servera

```
HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: HSmrc0sM1YUkAGmm5OPpG2HaGWk=
--- (opciono) ---
Sec-WebSocket-Protocol: chat
Sec-WebSocket-Extensions: ...
```

WebSocket frejm

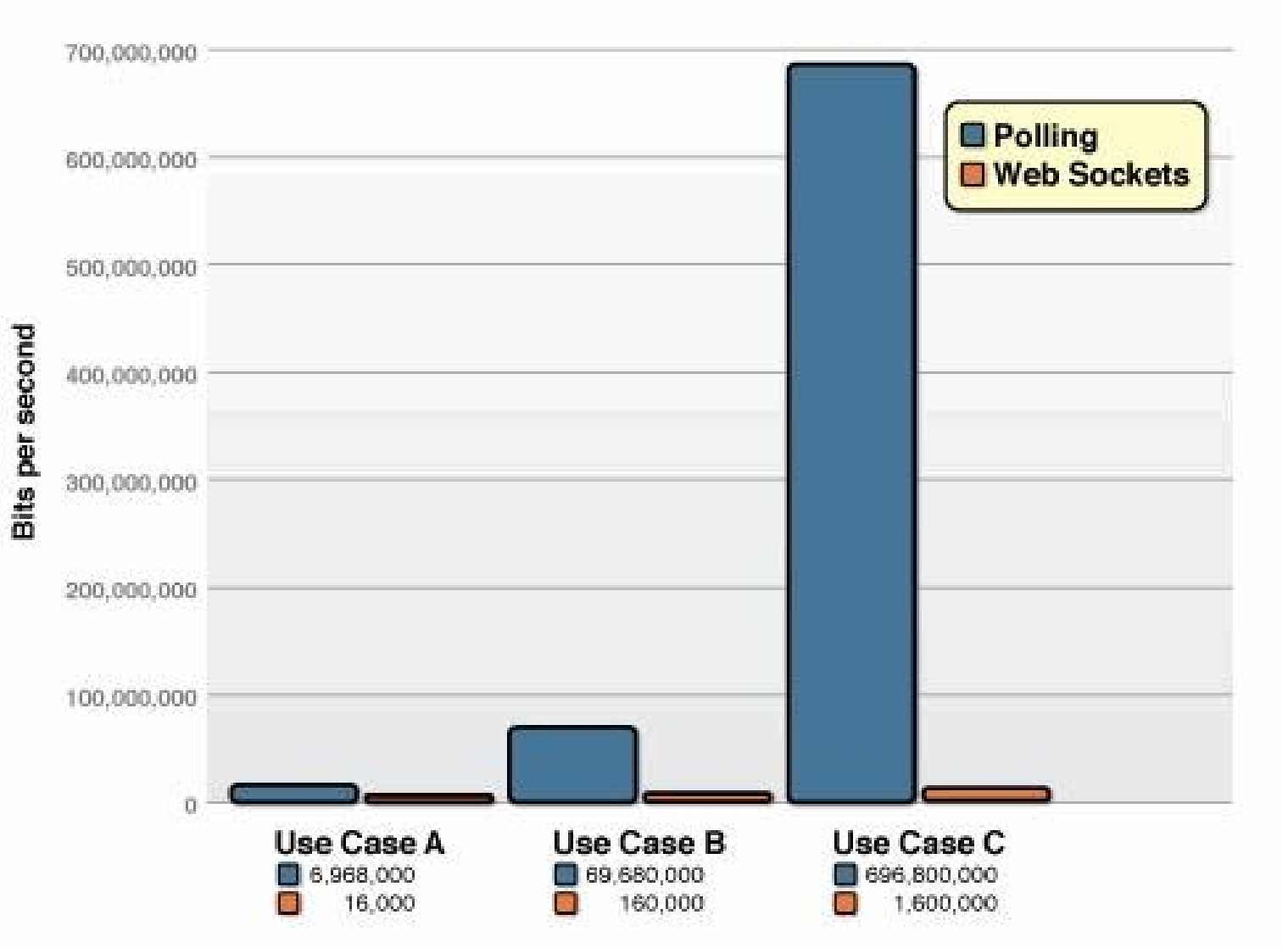
- nekoliko bajtova headera
- tekstualni ili binarni podaci



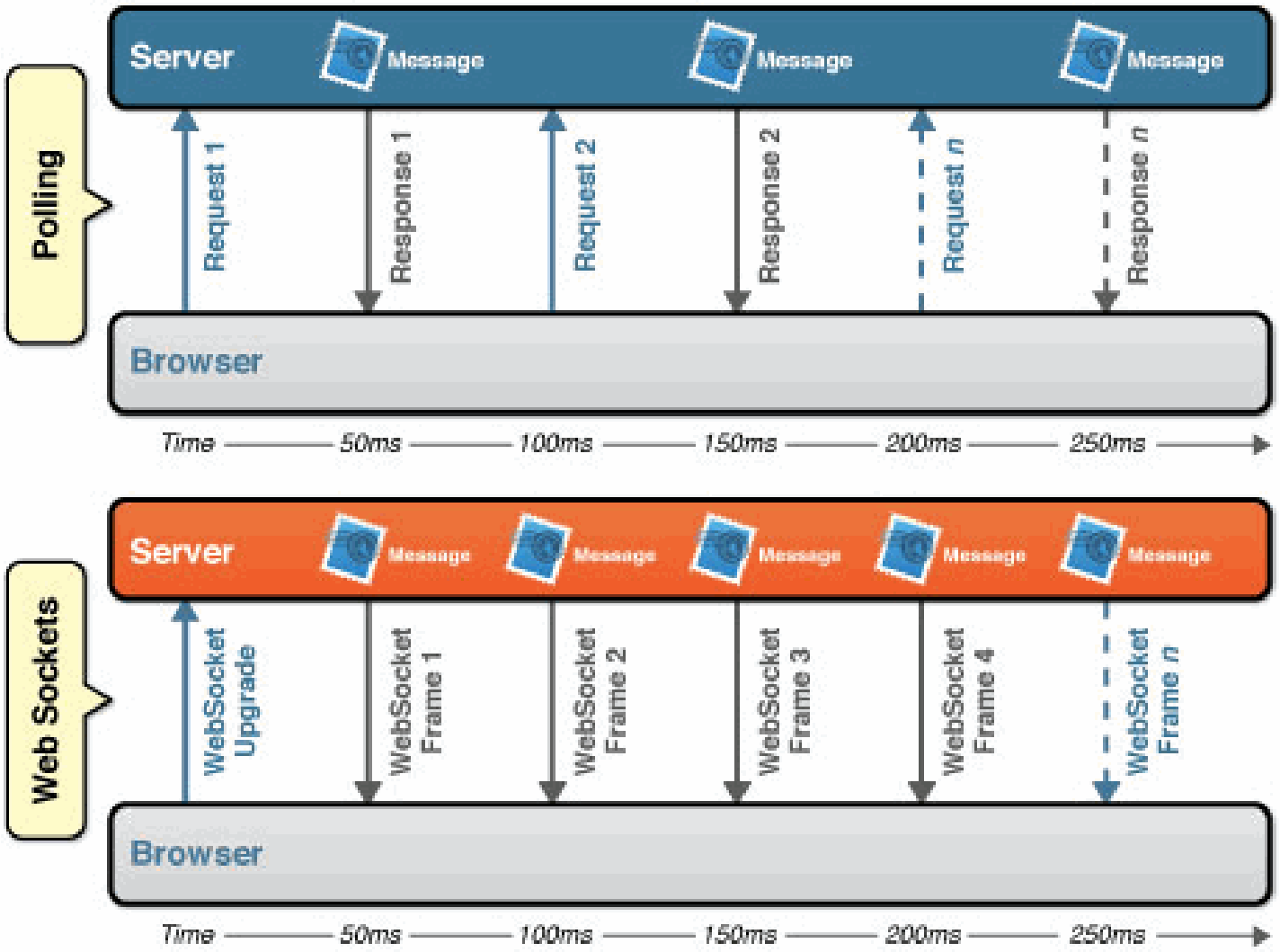
Efikasnost

	HTTP	WebSocket
overhead	100-tine B	2-6 B
odziv	nova konekcija	ista konekcija
polling	interval	bez čekanja
long polling	odgovor-zahtev	bez čekanja

Overhead: Polling vs WebSockets



Latency: Polling vs WebSockets



WebSocket prednosti

- **performanse**: efikasna real-time komunikacija
- **jednostavnost**: prosta klijent/server komunikacija preko weba
- **standardizacija**: WebSocket je standardni IETF protokol
- **HTML5**: deo HTML5 specifikacije