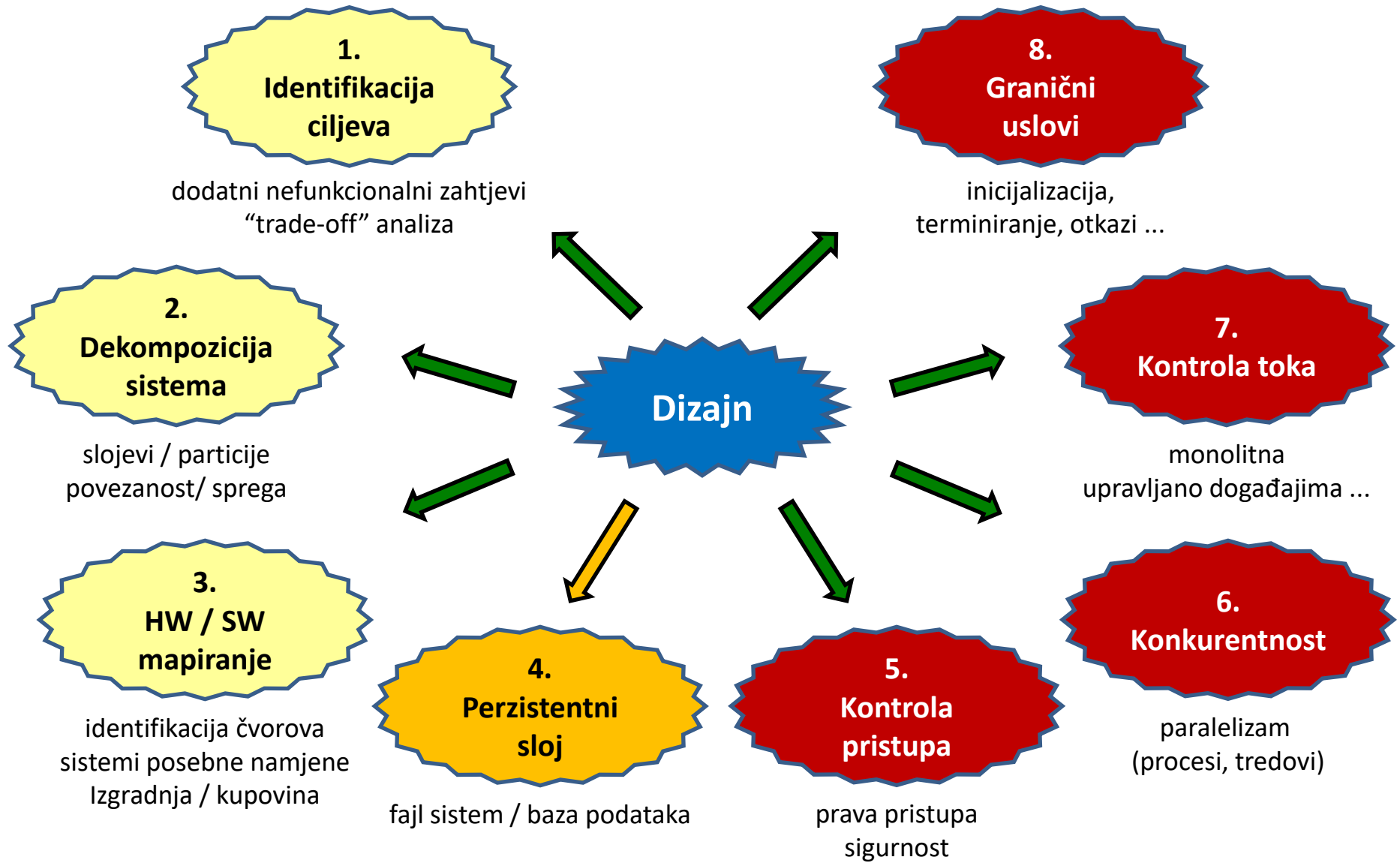
UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET

Prof. dr Dražen Brđanin

PROJEKTOVANJE SOFTVERA
/perzistentni sloj/

Banja Luka
2024.

8 bitnih aktivnosti u projektovanju



4. Perzistentni sloj

Perzistentni (trajni) objekti

- Objekti čiji je životni vijek duži od jednog izvršavanja aplikacije i čije stanje mora da se sačuva između dva izvršavanja aplikacije (domenski objekti, korisnička podešavanja, ...).
- **Dobar dizajn:** perzistentni objekti u zasebnom podsistemu sa dobro definisanim interfejsima

Manipulacija perzistentnim objektima

Fajl sistem

- **Tipičan slučaj: jedan proces upisuje, a veći broj procesa čita podatke**
- **Osnovne karakteristike:**
  - niska cijena, jednostavnost
  - low-level I/O (read, write)
  - aplikacije moraju da sadrže kôd koji obezbjeđuje odgovarajući nivo apstrakcije

Baza podataka

- **Tipičan slučaj: više konkurentnih procesa koji čitaju i/ili upisuju podatke**
- **Osnovne karakteristike:**
  - portabilnost, integritet podataka, sigurnost, ...
  - high-level I/O

4. Perzistentni sloj

Neka pitanja vezana za upravljanje podacima

Učestanost pristupa podacima

- Koliko često se pristupa podacima?
- Kolika je očekivana učestanost postavljanja upita? Najgori slučaj?

Arhiviranje podataka

- Da li je potrebno arhiviranje podataka?
- Da li je dovoljna reprezentacija trenutnog (posljednjeg) stanja objekata ili mora da se pamti istorija (ranija) stanja objekata?

Distribuiranost podataka

- Mogu li podaci da se drže centralizovano ili moraju distribuirano?
- Da li kod distribuiranog rasporeda treba obezbijediti lokacijsku transparentnost? (korisnik ima percepciju centralizovane organizacije podataka)

Interfejs za pristup podacima

- Da li je potreban jedinstven interfejs za pristup podacima?
- Da li korisnici imaju različite poglede na podatke?
- Kakav je format upita za pristup podacima?

Format podataka

- Da li format podataka treba da bude fleksibilan?
- Da li ciljna baza može biti relaciona / ne-relaciona?

Objektno-Relaciono mapiranje (ORM)

Mapiranje objektnog modela u relacioni model

Ekvivalencija E-R i UML objektnog modela (dijagram klasa)

Dijagram klasa (bez metoda) = E-R dijagram (MOV)

Mapiranje UML dijagrama klasa u šemu relacione baze podataka

Sličan skup pravila za mapiranje kao i za mapiranje E-R dijagrama

- svaka perzistentna klasa mapira se u korespondentnu tabelu
- svaki atribut klase mapira se u kolonu korespondentne tabele
- asocijacija sa kardinalnostima 1:* mapira se u dodatnu kolonu u tabeli koja odgovara klasi na strani * i tamo predstavlja strani ključ
- asocijacija sa kardinalnostima *:* mapira se u tabelu koja sadrži kolone koje reprezentuju strane ključeve prema tabelama koje korespondiraju krajevima date asocijacije
- atributi klase pridružene asocijaciji (*association class*) mapiraju se u korespondentne kolone u odgovarajućoj tabeli
- svaka veza nasljeđivanja mapira se u dodatnu kolonu u tabeli koja korespondira potklasi koja reprezentuje strani ključ (vertikalno mapiranje)

Objektno-Relaciono mapiranje (ORM)

Mapiranje klasa

Mjesto

idM (PK)	naziv

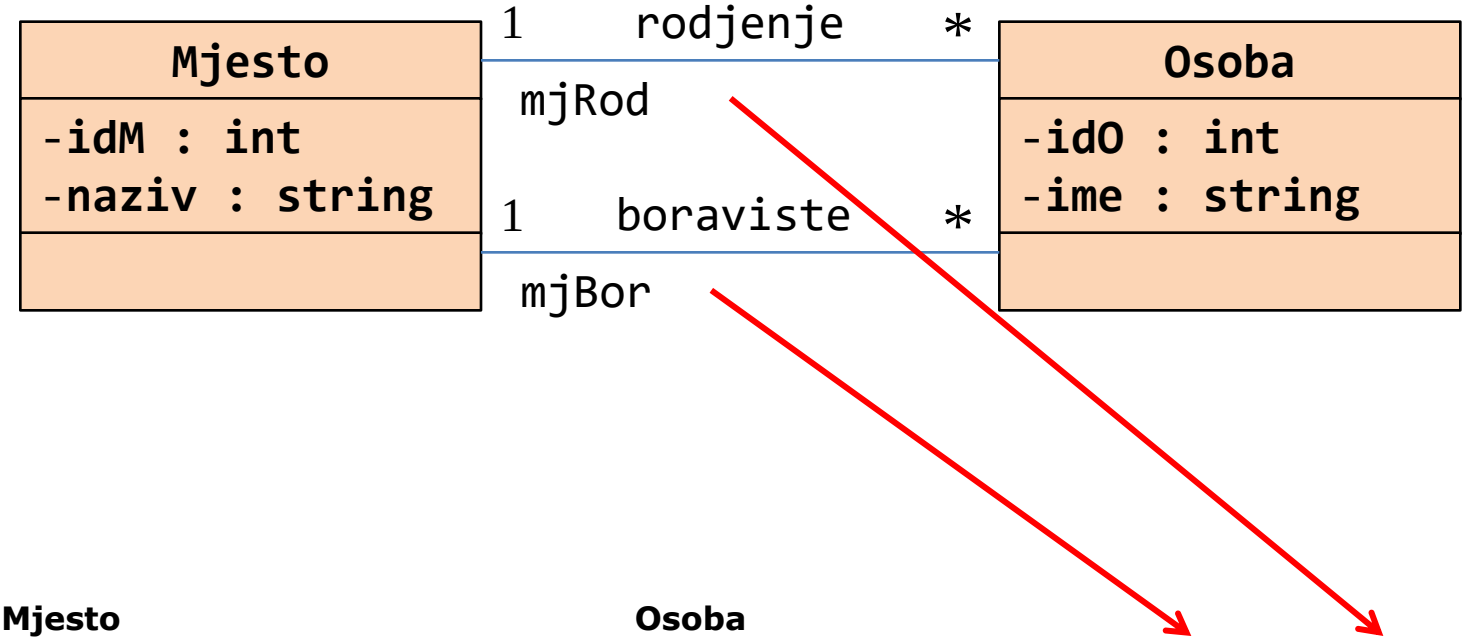
Osoba

idO (PK)	ime

Objektno-Relaciono mapiranje (ORM)

Mapiranje asocijacija (1:*)

Asocijacija sa kardinalnostima 1:* mapira se u dodatnu kolonu u tabeli koja odgovara klasi na strani * i tamo predstavlja strani ključ



Mjesto

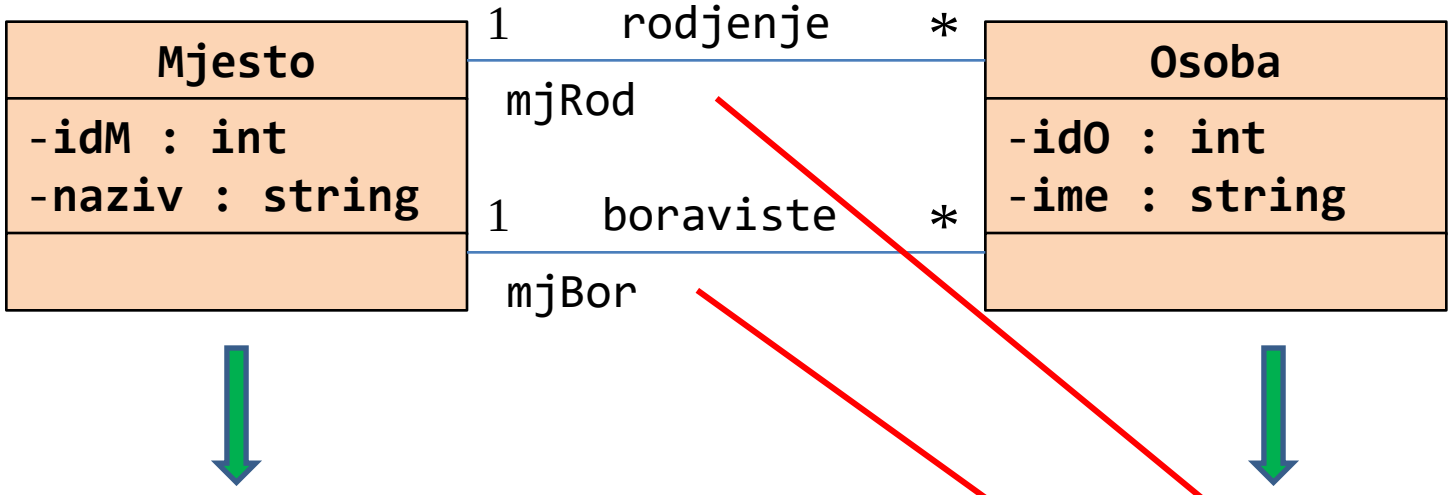
idM (PK)	naziv

Osoba

idO (PK)	ime	mjBor (FK)	mjRod (FK)

Objektno-Relaciono mapiranje (ORM)

Mapiranje asocijacija (1:*)



Mjesto

idM (PK)	naziv
100	BL
200	NS

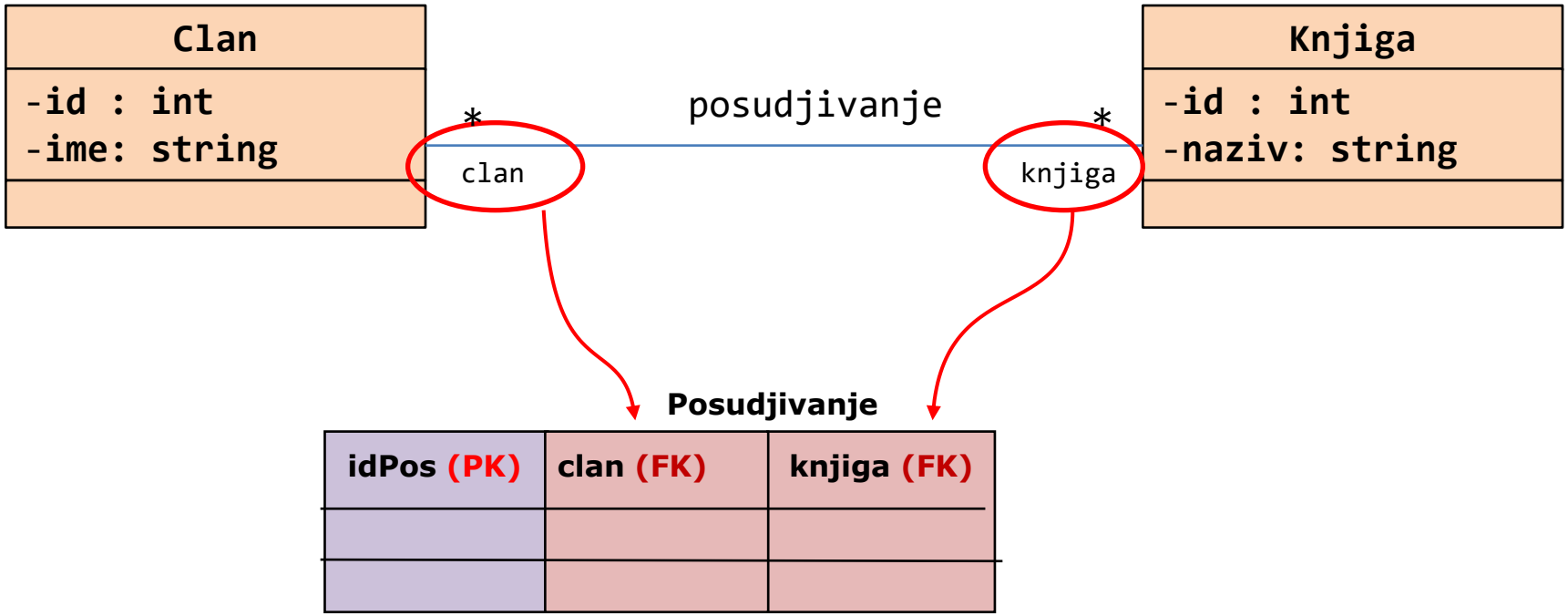
Osoba

idO (PK)	ime	mjBor (FK)	mjRod (FK)
1001	Marko	200	100

Objektno-Relaciono mapiranje (ORM)

Mapiranje asocijacija (*:*)

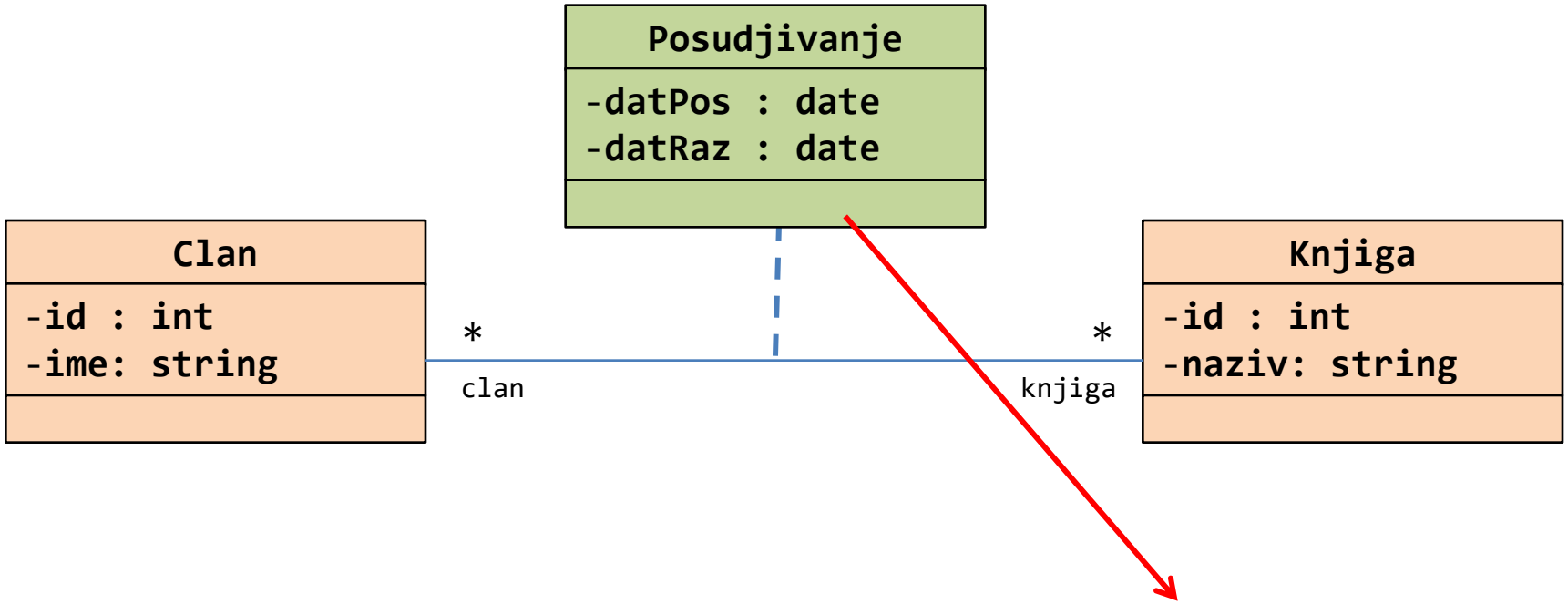
Asocijacija sa kardinalnostima *:* mapira se u tabelu koja sadrži kolone koje reprezentuju strane ključeve prema tabelama koje korespondiraju krajevima date asocijacije



Objektno-Relaciono mapiranje (ORM)

Mapiranje klase pridružene asocijaciji (*association class*)

Atributi klase pridružene asocijaciji (*association class*) mapiraju se u korespondentne kolone u odgovarajućoj tabeli

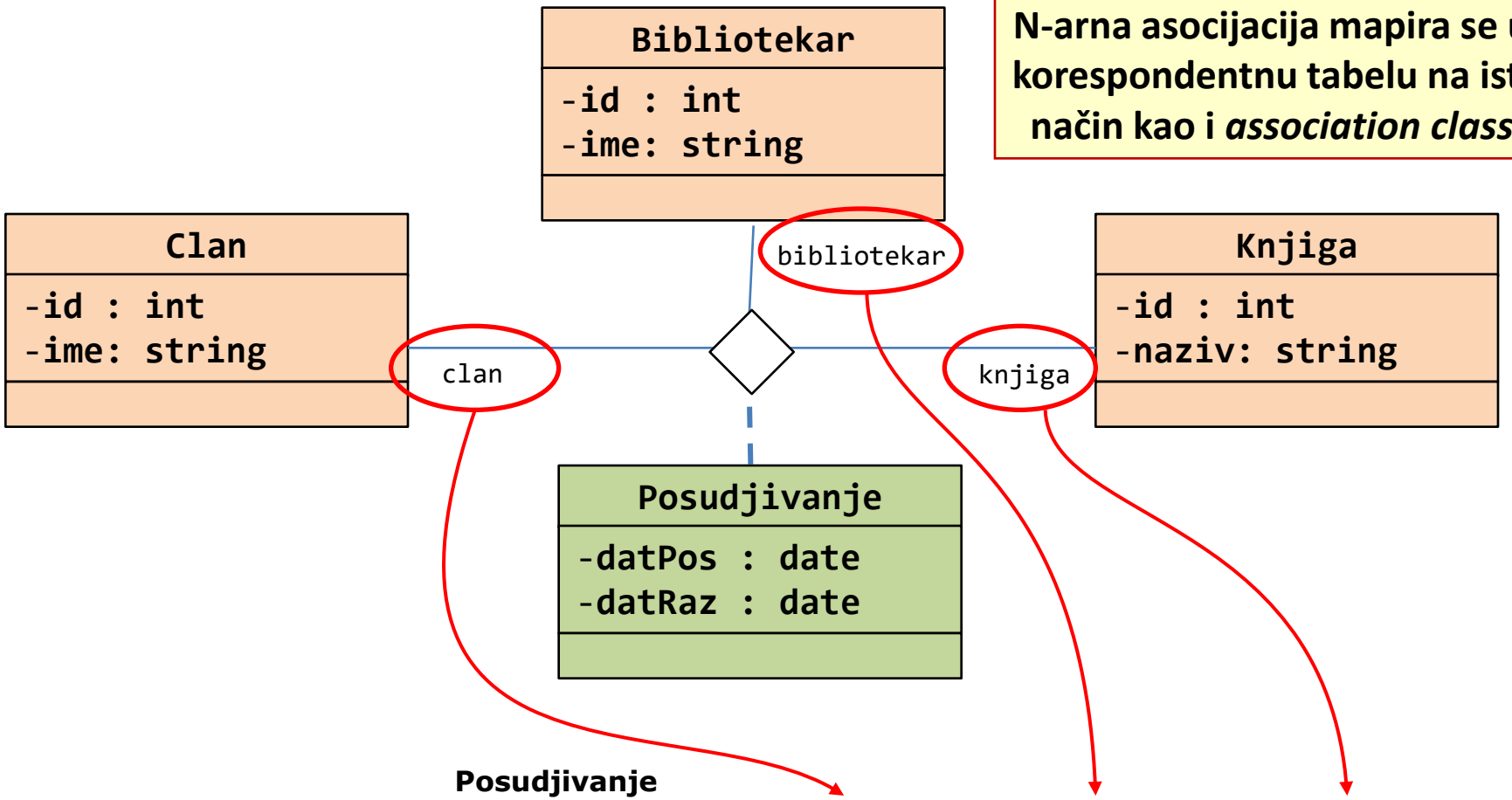


Posudjivanje

idPos (PK)	clan (FK)	knjiga (FK)	datPos	datRaz

Objektno-Relaciono mapiranje (ORM)

Mapiranje *n*-arne asocijacije

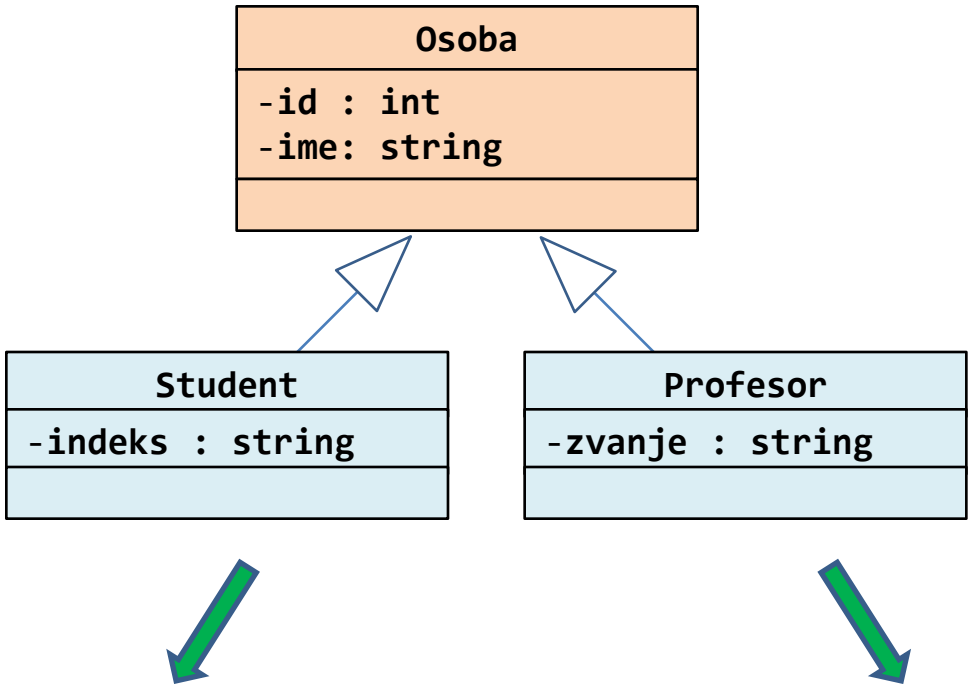


Posudjivanje

idPos (PK)	datPos	datRaz	clan (FK)	bibliotekar (FK)	knjiga (FK)

Objektno-Relaciono mapiranje (ORM)

Mapiranje nasljeđivanja



Horizontalno mapiranje nasljeđivanja

Svaka potklasa mapira se u korespondentnu tabelu kojoj se dodaju sve kolone koje odgovaraju atributima natklase

Student

id (PK)	ime	indeks

Profesor

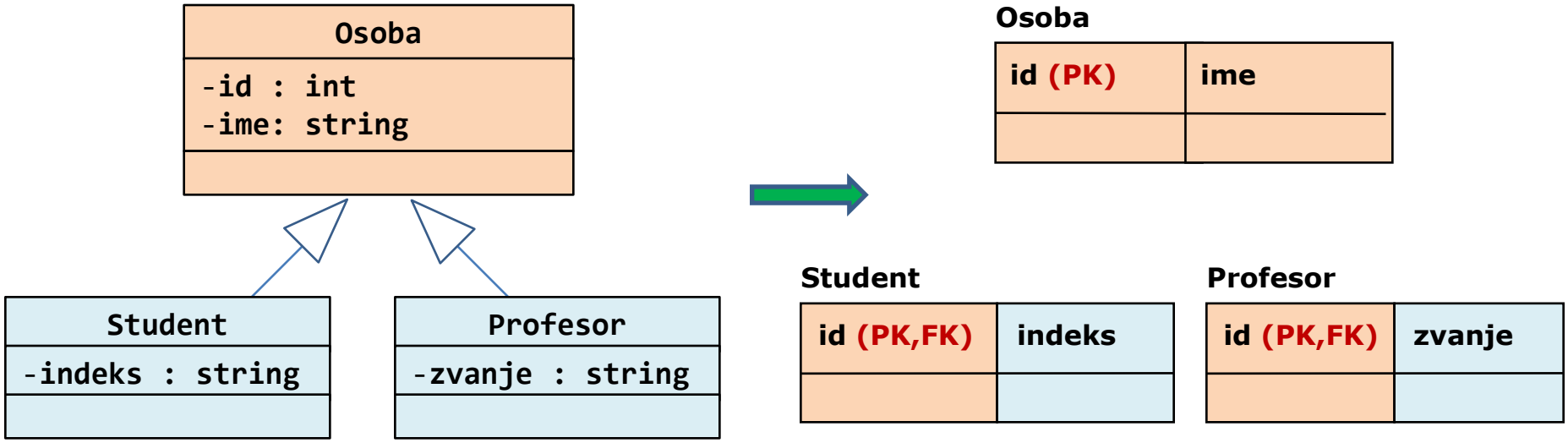
id (PK)	ime	zvanje

Objektno-Relaciono mapiranje (ORM)

Mapiranje nasljeđivanja

Vertikalno mapiranje nasljeđivanja

Svaka veza nasljeđivanja mapira se u dodatnu kolonu u tabeli koja korespondira potklasi koja reprezentuje strani klljuč



Objektno-Relaciono mapiranje (ORM)

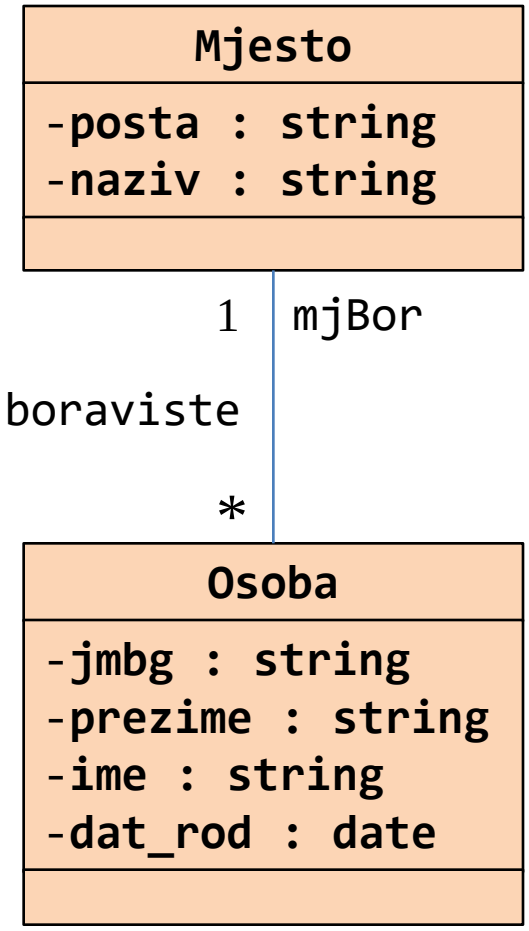
SQL – Strukturni Upitni Jezik / Structured Query Language

- standardni jezik (ANSI: ‘87, ‘92, ‘99, ‘03, ‘06, ‘08) za manipulaciju (relacionim) BP
- **DDL (Data Definition Language)**
  - **dio SQL jezika za definisanje baze podataka**
  - **kreiranje, modifikacija i brisanje tabela i indeksa**
  - **najznačajnije komande:**
    - **CREATE TABLE** – kreira novu tabelu u bazi podataka
    - **ALTER TABLE** – mijenja strukturu postojeće tabele u bazi podataka
    - **DROP TABLE** – briše tabelu iz baze podataka
    - **CREATE INDEX** – kreira indeks za neku tabelu
    - **DROP INDEX** – briše indeks za neku tabelu

Objektno-Relaciono mapiranje (ORM)

SQL – DDL

CREATE TABLE – kreiranje tabele




```
CREATE TABLE MJESTO
(
    POSTA  VARCHAR(5) NOT NULL,
    NAZIV  VARCHAR(20),
    PRIMARY KEY (POSTA)
);
```

```
CREATE TABLE CLAN
(
    JMBG        VARCHAR(13) NOT NULL,
    MJBOR       VARCHAR(5),
    PREZIME     VARCHAR(20),
    IME         VARCHAR(20),
    DAT_ROD     DATE,
    PRIMARY KEY (JMBG),
    FOREIGN KEY (MJBOR) REFERENCES MJESTO
);
```

Objektno-Relaciono mapiranje (ORM)

SQL – DDL

ALTER TABLE – promjena strukture tabele

Mjesto
-posta : string -naziv : string

```
CREATE TABLE MJESTO
(
    POSTA  VARCHAR(5) PRIMARY KEY,
    NAZIV  VARCHAR(20),
);
```

MJESTO

POSTA	NAZIV

```
ALTER TABLE MJESTO ADD DRZAVA VARCHAR(20);
```

MJESTO

POSTA	NAZIV	DRZAVA

```
ALTER TABLE MJESTO DROP DRZAVA;
```

MJESTO

POSTA	NAZIV

```
DROP TABLE MJESTO;
```

Objektno-Relaciono mapiranje (ORM)

SQL – DDL

CREATE INDEX – kreiranje indeksa za neku tabelu

CLAN

JMBG	PREZIME	IME

```
CREATE INDEX CLAN_JMBG ON CLAN
(
        JMBG
);
```

```
CREATE INDEX CLAN_IME ON CLAN
(
        PREZIME ASC,
        IME DESC
);
```

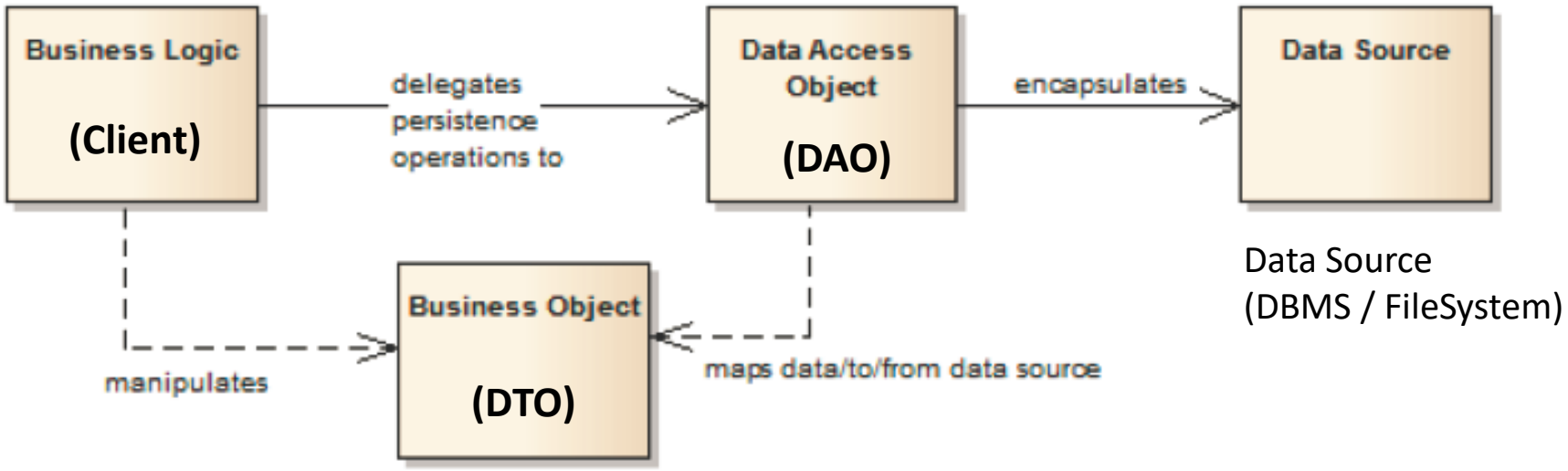
DROP INDEX – brisanje indeksa za neku tabelu

```
DROP INDEX CLAN_JMBG ;
```

```
DROP INDEX CLAN_IME ;
```

Pristup perzistentnom sloju

DAO obrazac (Data Access Object)



Business Logic (Client)

- Kontroler koji pristupa (perzistentnom) domenskom objektu
- Kontroler zna kad i zašto mu trebaju podaci, ali ne zna (i ne mora da zna) kako je riješena perzistencija

Data Access Object (DAO)

- DAO razdvaja logičku i fizičku reprezentaciju domenskih objekata
- DAO zna gdje su i kako se smješteni podaci, ali ne zna kad i zašto treba da im se pristupi

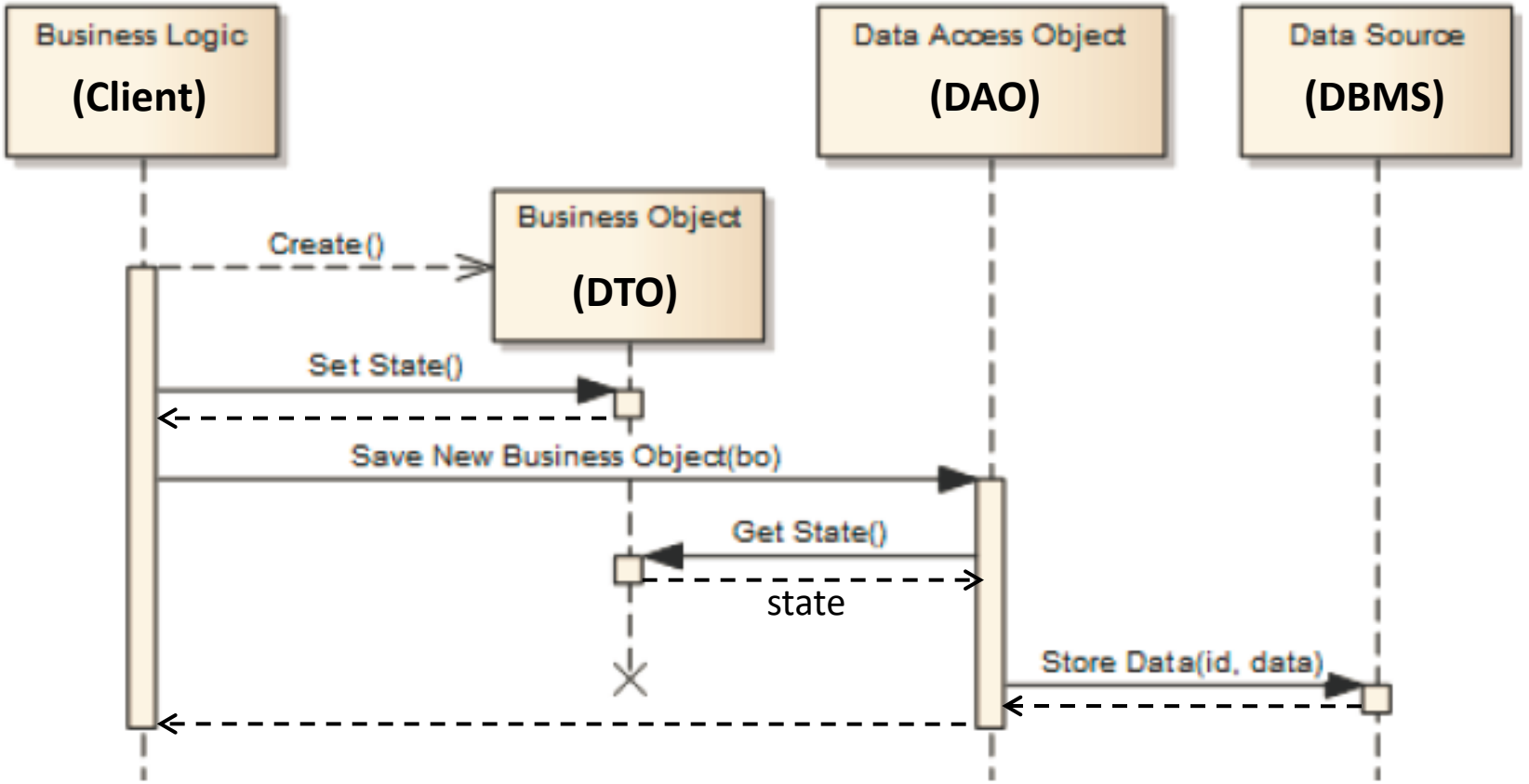
Business Object (DTO – Data Transfer Object)

- DTO reprezentuje domenski objekat
- DTO je nosilac informacije u komunikaciji Client ↔ DAO

Pristup perzistentnom sloju

DAO obrazac – komunikacija prilikom pristupa perzistentom sloju

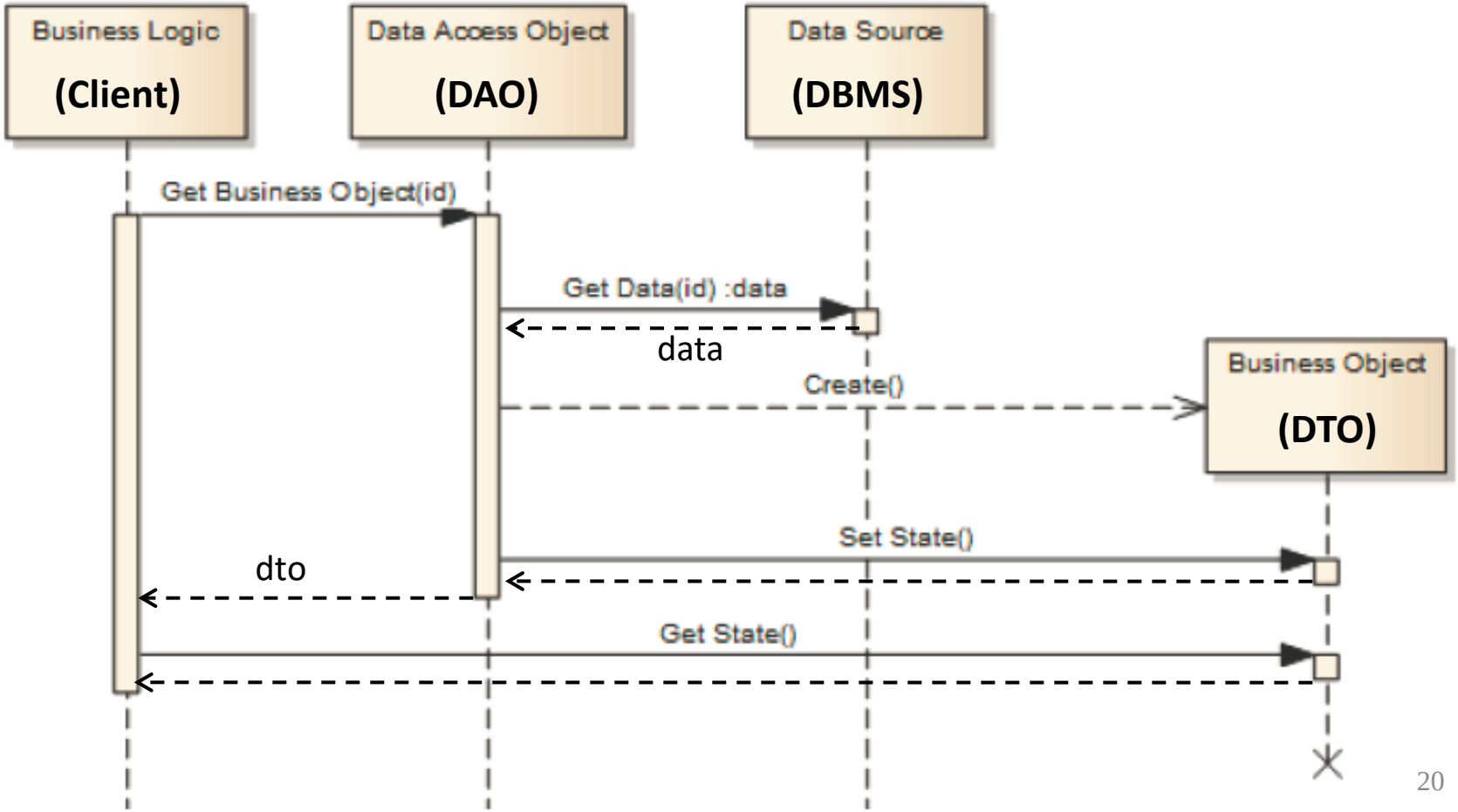
Kreiranje i perzistencija DTO



Pristup perzistentnom sloju

DAO obrazac – komunikacija prilikom pristupa perzistentom sloju

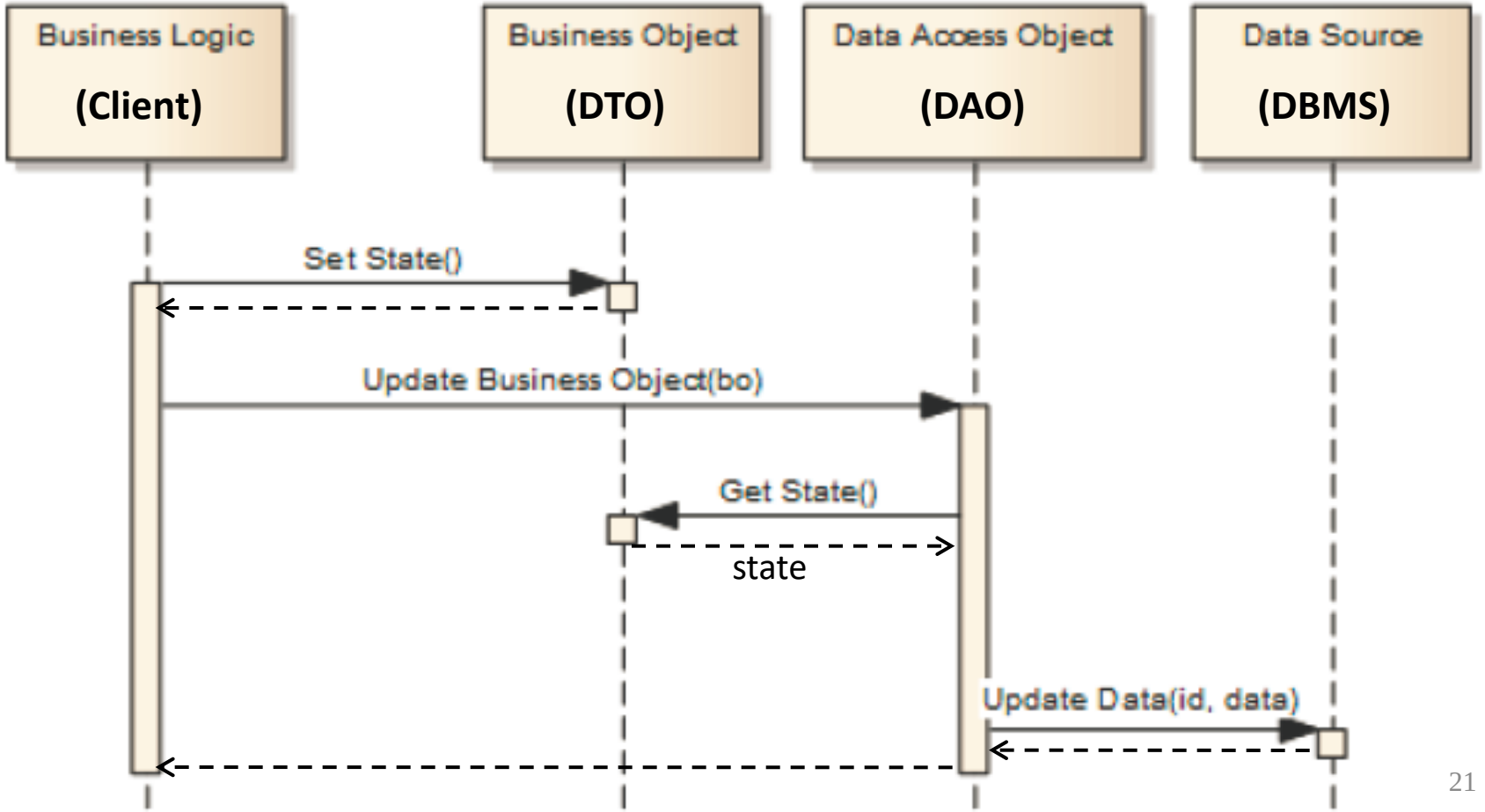
Pribavljanje DTO



Pristup perzistentnom sloju

DAO obrazac – komunikacija prilikom pristupa perzistentom sloju

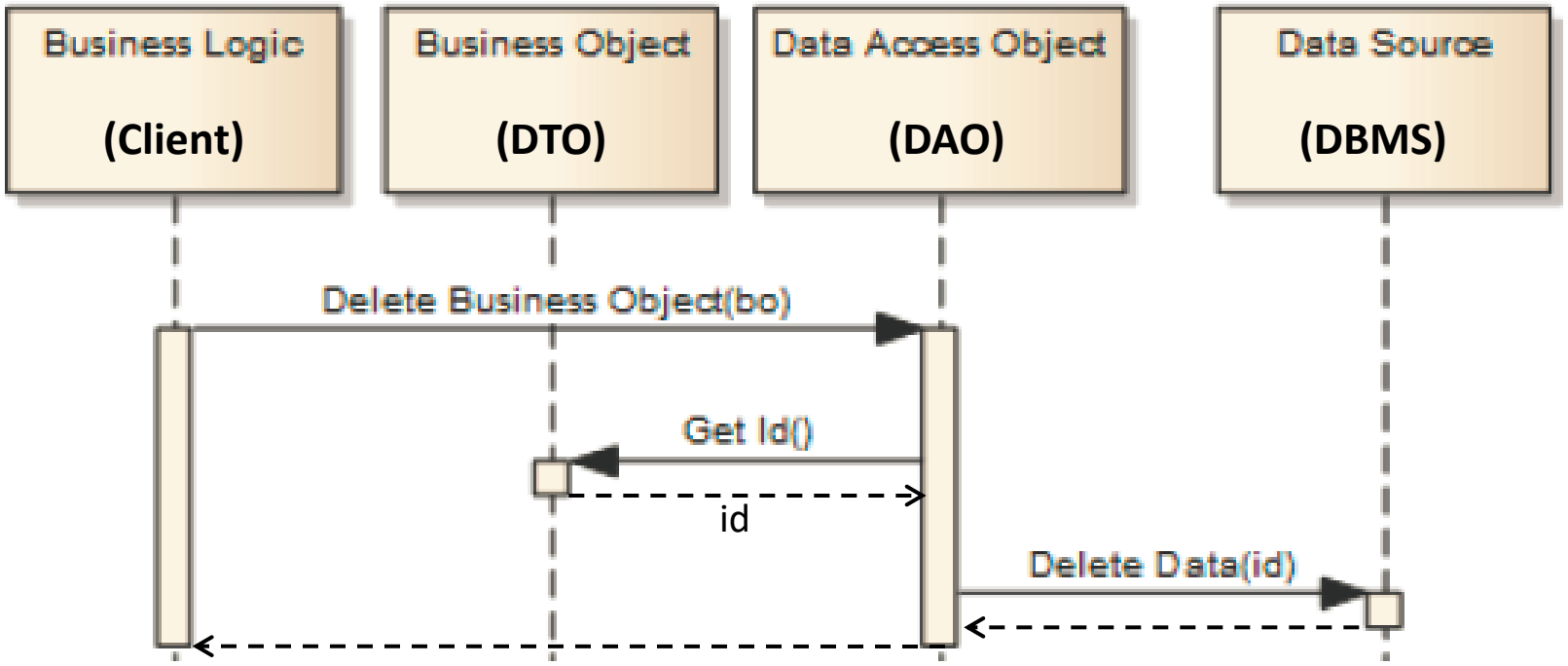
Modifikacija perzistentnog objekta



Pristup perzistentnom sloju

DAO obrazac – komunikacija prilikom pristupa perzistentom sloju

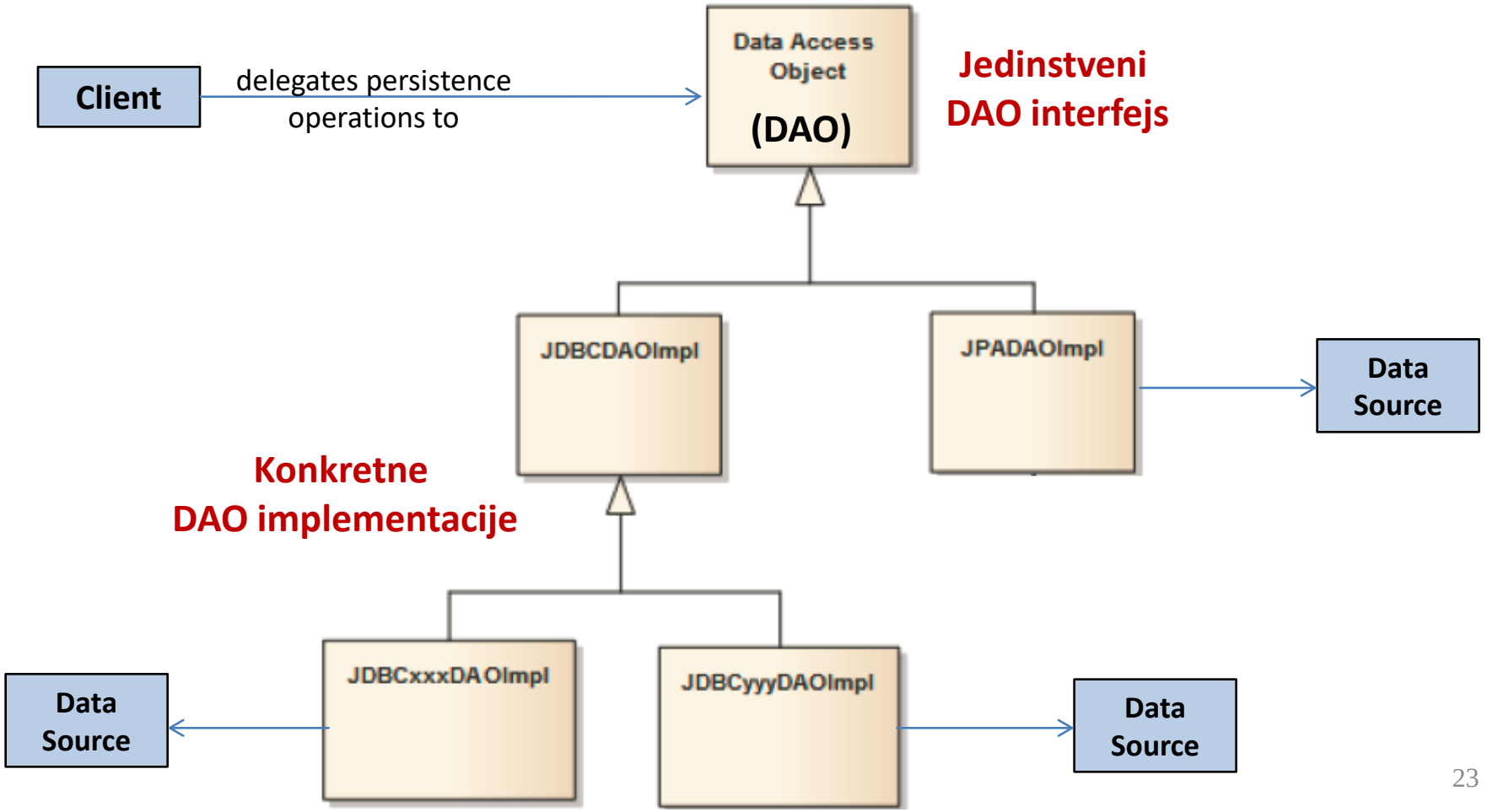
Brisanje perzistentnog objekta



Pristup perzistentnom sloju

DAO obrazac – implementacioni detalji

Različite "strategije" za pristup različitim vidovima perzistencije



Pristup perzistentnom sloju

DAO obrazac – prednosti korištenja

- Centralizacija svih operacija vezanih za pristup perzistentnim podacima u zaseban podsistem (komponentu)
- Jednostavnije održavanje
- Transparentnost
- Implementacioni detalji za pristup perzistentnom sloju skriveni u DAO klasama
- Lakša migracija i podrška za različite načine pristupa
- **“Separation of concerns”:**
  - Klijent realizuje poslovnu logiku – DAO realizuje pristup podacima
  - Manja kompleksnost koda u poslovnoj logici (nema SQL koda u poslovnoj logici)

Pristup perzistentnom sloju

DAO obrazac – implementacioni detalji

DAO interfejs

- Tehnološki nezavisan i fokusiran na operacije sa DTO
- Deklaracije CRUD operacija (Create-Retrieve-Update-Delete)
- Deklaracije dodatnih agregatnih funkcija
- Deklaracije dodatnih funkcija za specifične slučajeve upotrebe

```
...
import javax.persistence.PersistenceException;

public interface MjestoDAO
{
  Mjesto create(Mjesto m)   throws PersistenceException;
  Mjesto update(Mjesto m)   throws PersistenceException;
  Mjesto retrieve(String p) throws PersistenceException;
  void delete(Mjesto m)     throws PersistenceException;
  List<Mjesto> readAll()    throws PersistenceException;
}
```

DAO Exceptions

Runtime Exceptions
- neočekivane greške (npr. nema konekcije)

Checked Exceptions
- očekivane greške (npr. pogrešan format, nedozvoljena vrijednost)

Pristup perzistentnom sloju

DAO obrazac – implementacioni detalji

DAO implementacija

- Svaka konkretna implementacija prilagođena konkretnom DataStore
- Implementacija CRUD operacija
- Implementacija dodatnih funkcija

```
// primjer DAO implementacije
public class MjestoDAOImp implements MjestoDAO
{

   @Override
   public Mjesto create(Mjesto m) { ... }
   ...
}
```

```
// primjer klijenta
public class MjestoDAOTest
{
  protected Mjesto createM()
  {
     MjestoDAO dao =
         new MjestoDAOImp();
     Mjesto m = new Mjesto();
     m = dao.create(m);
     return m;
  }
  ...
}
```

Pristup bazi podataka – JDBC

JDBC (*Java Database Connectivity*)

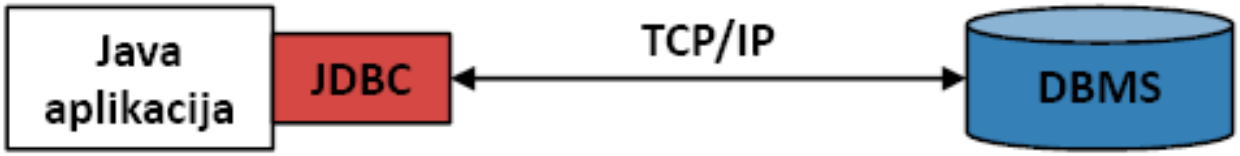
- Java API koji obezbjeđuje klase i metode za interakciju sa različitim RDBMS
- Omogućava jednostavan i transparentan rad sa RDBMS
- Cjelokupan podsistem je definisan u **java.sql** paketu

JDBC drajveri

- Svaki proizvođač obezbjeđuje drajver za svoj RDBMS (MS SQL Server, MySQL ...)
- Svi drajveri koriste se na isti način

Pristup RDBMS iz Java aplikacije

- Klijent-server arhitekturni stil (klijent: java aplikacija | server: RDMBS)
- Klijentska java aplikacija komunicira direktno sa serverom
- JDBC komponente nalaze se u klijentskom sloju



Dvoslojna arhitektura sa JDBC komponentama

Pristup bazi podataka – JDBC

Osnovni koraci u radu sa bazom

1. Učitavanje drajvera – automatski
2. Uspostavljanje konekcije
3. Kreiranje iskaza (*Statement*)
4. Izvršavanje iskaza
5. Obrada rezultata
6. Zatvaranje konekcije

Uspostavljanje konekcije

- Konekcija na DBMS predstavljena je objektom tipa **Connection**
- Metoda **getConnection** klase **DriverManager** vraća objekat tipa **Connection** ako je konekcija sa DBMS-om uspješno uspostavljena:

```
Connection conn = DriverManager.getConnection(
    "jdbc:mysql://host:port/baza_podataka", "user_ime", "pass");
```

Pristup bazi podataka – JDBC

SQL - DML (*Data Manipulation Language*)
Detaljno na BAZAMA PODATAKA!

Kreiranje i izvršavanje iskaza

- **Statement** – koristi se za implementaciju jednostavnih SQL iskaza
- **ResultSet** – rezultat izvršavanja upita

```
Connection c = DriverManager.getConnection(
          "jdbc:mysql://localhost:3306/baza", "user", "pass");

Statement s = c.createStatement();

ResultSet rs = s.executeQuery("select * from mjesto");

while (rs.next())
   System.out.println(rs.getString("posta") + " " +
                      rs.getString("naziv"));

rs.close();
s.close();
c.close();
```

Pristup bazi podataka – JDBC

DAO implementacija

```
// primjer DAO implementacije
public class MjestoDAOImp implements MjestoDAO
{
  @Override
  public List<Mjesto> readAll() throws PersistenceException
  {
    List<Mjesto> lista = new ArrayList<Mjesto>();
    try
    {
      Connection c = DriverManager.getConnection(
             "jdbc:mysql://localhost:3306/baza", "user", "pass");
      Statement s = c.createStatement();
      ResultSet rs = s.executeQuery("select * from mjesto");
      while (rs.next())
         lista.add(new Mjesto(rs.getString("posta"), rs.getString("naziv")));
      rs.close();
      s.close();
      c.close();
    }
    catch (SQLException e) { e.printStackTrace(); }
    return lista;
  }
}
```

Tehnologije i alati za automatsko ORM

Prethodno je prikazano:

- objektno-relaciono mapiranje,
- manuelni proces projektovanja perzistentnog sloja na osnovu objektnog modela
- projektovanje sloja za pristup perzistentnom sloju
- aplikativna manipulacija relacionim bazama podataka zasnovana na JDBC

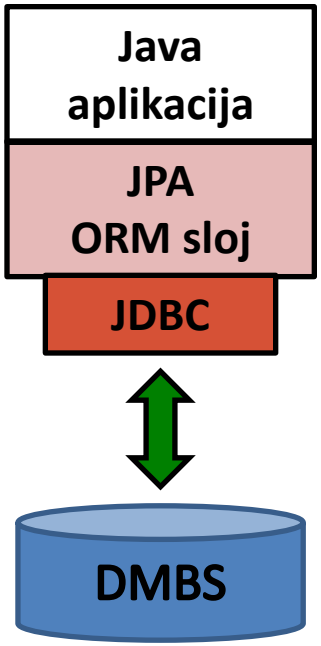
Automatizacija O-R mapiranja i pristupa perzistentnom sloju

- **API za ORM**
  - **Java**: **JPA**, EJB, Hibernate, ORMLite (Android), JDO, ...
  - **.NET**: EntityFramework, ADO.NET, Nhibernate, ...
  - **Python**: Django, ...

Pristup perzistentnom sloju – JPA

JPA (*Java Persistence API*)

- Skup koncepata za manipulaciju perzistentnim slojem iz Java aplikacija
- JPA nije poseban alat niti poseban AF (aplikativni okvir)
- JPA je originalno zasnovan na Hibernate (AF za ORM i pristup RDBMS putem JDBC)
- JPA je originalno namijenjen za rad sa RDBMS, ali postoje implementacije (EclipseLink, Hibernate OGM) koje omogućavaju rad sa NoSQL (nerelacionim) bazama podataka
- JPA i JPA-zasnovani alati formiraju sloj za ORM
- ORM sloj je adapter – međusloj između aplikacije i DBMS, koji programerima stvara osjećaj potpune O-O paradigme
- **JPA podiže nivo apstrakcije i omogućava programerima da samo definišu mapiranje aplikativnih objekata na odgovarajući DBMS i da se ne bave stvarnim pristupom DBMS-u – JPA završava snimanje i čitanje preko JDBC**



Pristup perzistentnom sloju – JPA

JPA (*Java Persistence API*)

- **JPA podiže nivo apstrakcije i omogućava programerima da samo definišu mapiranje aplikativnih objekata na odgovarajući DBMS i da se ne bave stvarnim pristupom DBMS-u – JPA završava snimanje i čitanje preko JDBC**

```
// primjer JDBC
public class MjestoDAOImp implements MjestoDAO
{
  public void insert(Mjesto m) throws Persi
  {
    // ...
    Connection c = DriverManager.getConnect
      "jdbc:mysql://localhost:3306/baza“,“u
    String upit =
      "insert into mjesto(posta, naziv) val
    PreparedStatement ps = c.prepareStateme
    ps.setString (1, m.getPosta());
    ps.setString (2, m.getNaziv());
    ps.execute();
    c.close();
    // ...
  }
}
```

```
// primjer JPA
public class MjestoTest
{
  protected Mjesto makeM()
  {
    // ...
    Mjesto m =
       new Mjesto("78000", "BL");
    entityManager.persist(m);
    // ...
    return m;
  }
  // ...
}
```

Pristup perzistentnom sloju – JPA

JPA anotacije

- **JPA koristi anotacije za mapiranje aplikativnih objekata na perzistentni sloj** (pored anotacija mapiranje može da se definiše i u eksternim XML fajlovima)
- Svaka JPA implementacija ima **procesor (*engine*) za obradu JPA anotacija**

```
@Entity
public class Mjesto { // ... }
```

Objekti klase Mjesto su perzistentni objekti

```
@Entity
@Table(name="mjesto")
public class Mjesto { // ... }
```

Objekti klase Mjesto su perzistentni objekti, koji se nalaze u tabeli "mjesto"

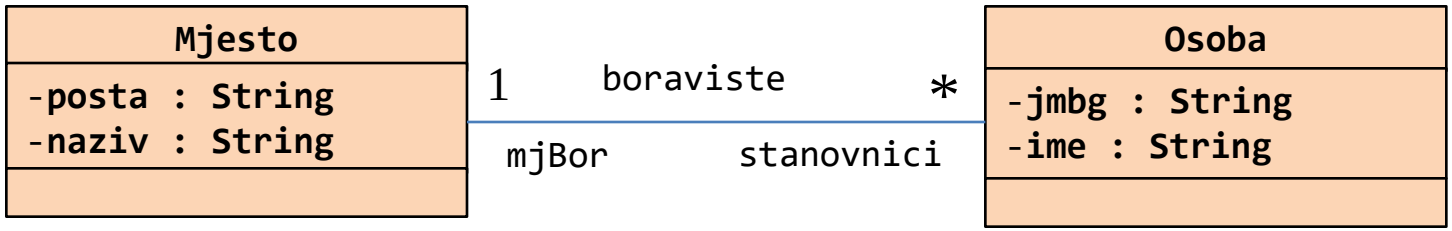
```
@Entity
@Table(name="mjesto")
public class Mjesto
{
   @Id
   private String posta;
}
```

Atribut posta predstavlja identifikator (primarni ključ)

Pristup perzistentnom sloju – JPA

JPA anotacije za asocijacije

@ManyToOne	@OneToOne
@OneToMany	@ManyToMany




```
@Entity
@Table(name="mjesto")
public class Mjesto
{
  @Id
  private String posta;

  @OneToMany(targetEntity=Osoba.class)
  private List stanovnici;

  private String naziv;
  // ...
}
```

```
@Entity
@Table(name="osoba")
public class Osoba
{
  @Id
  private String jmbg;

  @ManyToOne
  private Mjesto mjBor;

  private String ime;
  // ...
}
```

Pristup perzistentnom sloju – JPA

JPA (*Java Persistence API*)

- **EntityManager** – pristupna tačka JPA sloja – komunikacija **aplikacija ↔ JPA sloj**

```
public class MjestoTestPersistence
{
  public static void main( String[] args )
  {
    EntityManagerFactory emfactory =
        Persistence.createEntityManagerFactory( "Eclipselink_JPA" );
    EntityManager em = emfactory.createEntityManager();
    em.getTransaction().begin();
    Mjesto m = new Mjesto("78000", "BL");
    em.persist(m);
    Osoba o1 = new Osoba("1111", "Marko", m);
    Osoba o2 = new Osoba("2222", "Janko", m);
    em.persist(o1); em.persist(o2);
    em.getTransaction().commit();
    em.close();
    emfactory.close();
  }
}
```

Pristup perzistentnom sloju – JPA

JPA – CRUD operacije

```
public class MjestoTestCRUD
{
  public static void main( String[] args )
  {
    EntityManagerFactory emfactory =
        Persistence.createEntityManagerFactory( "Eclipselink_JPA" );
    EntityManager em = emfactory.createEntityManager();
    Mjesto m = em.find(Mjesto.class, "78000");
    em.getTransaction().begin();
    m.setName("Banja Luka");
    em.getTransaction().commit();
    em.close();
    emfactory.close();
  }
}
```