Objektno - Relaciono Mapiranje

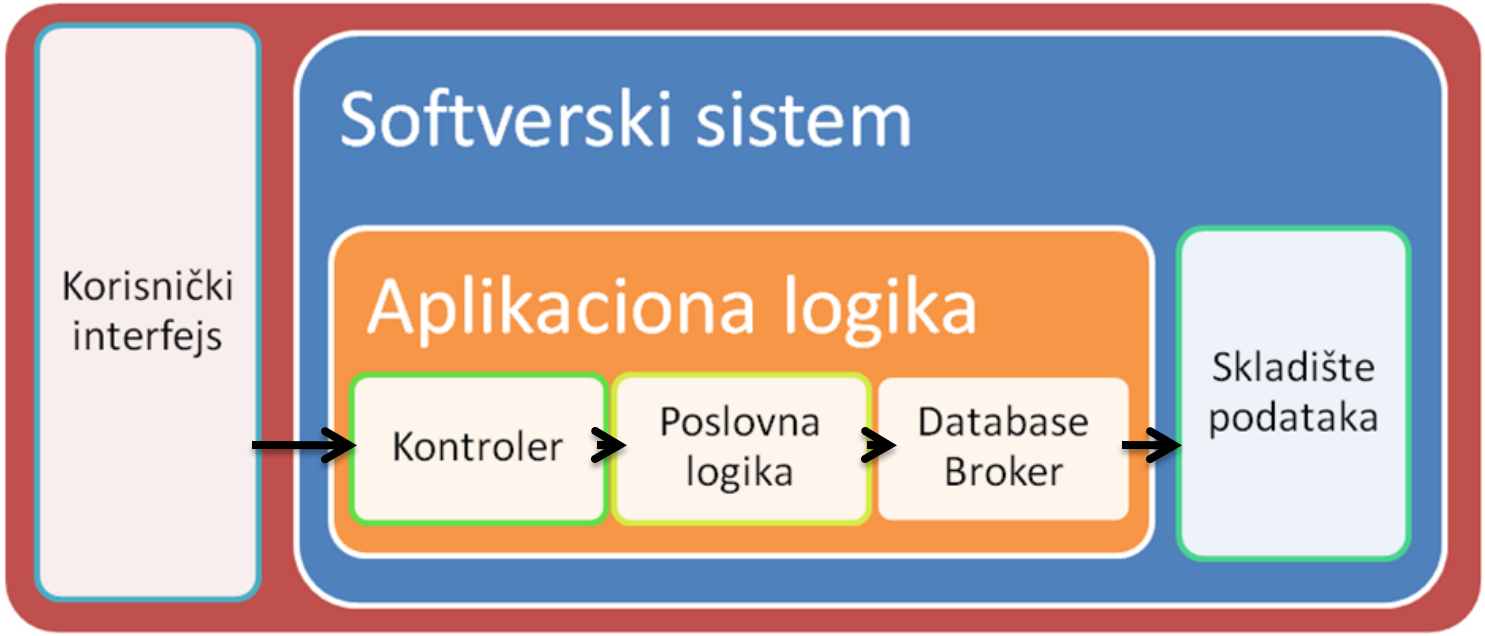
doc. dr Ilija Antović

ilijaa@fon.bg.ac.rs

Sadržaj kursa

- ORM – problem transformacije modela podataka
  - Perzistencija podataka
  - Objektni model
  - Relacioni model
  - Problem transformacije
- Alati za ORM
  - JDBC
  - iBatis
  - Hibernate
  - JPA
- Hibernate
  - Karakteristike, koraci u pisanju programa, različita mapiranja, operacije...
  - Hibernate search
- Primjeri

Tronivojska arhitektura



Perzistencija podataka

Objekat je perzistentan ukoliko nastavi da postoji i nakon prestanka rada programa koji ga je stvorio (G. Booch).

Objekat je perzistentan ukoliko se može materijalizovati i dematerijalizovati.

Materijalizacija predstavlja proces transformacije slogova iz baze podataka u objekte programa.

Dematerijalizacija predstavlja proces transformacije objekta iz programa u slogove baze podataka.

Perzistentni okvir je skup interfejsa i klasa koji omogućava perzistentnost objektima različitih klasa.

Relacione baze podataka

- Standard u domenu perzistencije podataka
- Jednostavnost kreiranja i pristupa podacima korišćenjem SQL-a
- Jednostavnost strukture modela podataka – Relacioni model
- Integritet podataka

Objektno orjentisane aplikacije

- Čuvanje objekata u relacionim bazama podataka
- Objektni model
- Perzistentni i transijentni objekti

Perzistentnost – proces transformacije objektnog modela u relacioni i obratno

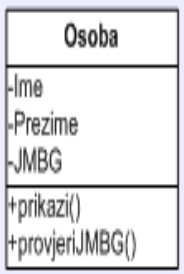
Objektni model

Objekat – entitet sposoban da čuva svoja stanja, i koji okolini stavlja na raspolaganje skup operacija preko kojih se tim stanjima upravlja.

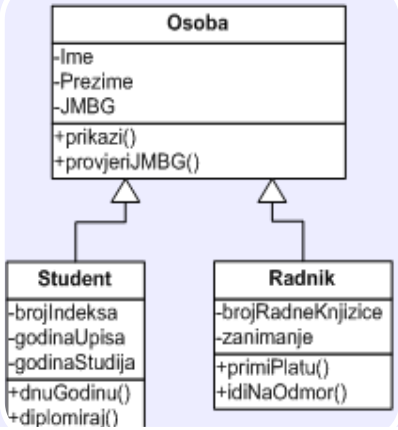
Stanje objekta – vrijednosti njegovih osobina-atributa i njegovih veza sa drugim objektima u sistemu.

Klasa – Apstraktna predstava skupa objekata koji imaju iste osobine

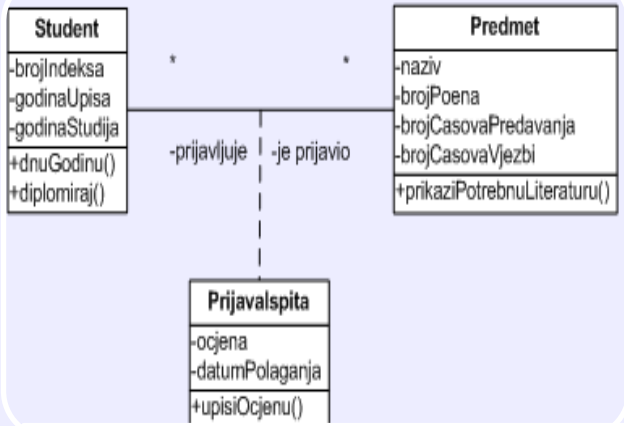
Objektni model :: apstrakcije podataka



Tipizacija - klasifikacija



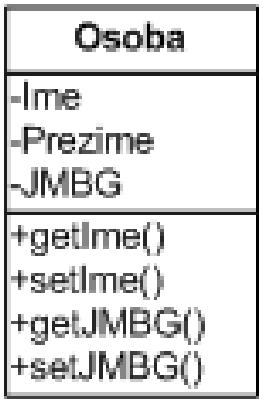
Generalizacija - specijalizacija



Agregacija

Objektni model - enkapsulacija

- Enkapsulacija – učaurenje
- Sakrivanje detalja implementacije stanja objekta
- Skup operacija kojima se omogućava pristup i promjena stanja
- Izmjena implementacije stanja jednog objekta ne zahtijeva izmjenu objekata koji su sa njim povezani



Objektni model

- Preklapanje metoda - *overloading*
- Prekrivanje (redefinisanje) metoda - *overriding*
- Late binding
- Polimorfizam
- Apstraktne klase
- Interfejsi

Relacioni model

- Sistem – skup relacija
- Relacija – tip entiteta i/ili veza između entiteta
- Relacija: tabela, kolone: atributi, vrste: n-torke
- N-torka: red u tabeli sa konkretnim vrijednostima atributa, konkretno pojavljivanje određene relacije
- Vrijednosno orjentisan – veze se ostvaruju preko vrijednosti atributa

Relacioni model :: apstrakcije podataka

Tipizacija – klasifikacija

- Artikal (ArtikalID, Naziv, DatumProizvodnje, Opis, JedinicaMjere, Cijena)

Generalizacija – specijalizacija

- Artikal (ArtikalID, Naziv, Opis, JedinicaMjere, Cijena)
- PrehrambeniArtikal (ArtikalID, RokTrajanja, Sastav)
- AparatZaDomaćinstvo (ArtikalID, TehničkeOsobine, UputstvoZaUpotrebu, Garancija)

Agregacija

- Artikal (ArtikalID, Naziv, Opis, JedinicaMjere, Cijena)
- Dobavljač (DobavljačID, Naziv, Adresa, KontaktTelefon)
- Dobavlja(DobavljačID, ArtikalID, Marža)

Relacioni model :: SQL

- Jezik sistema za upravljanje (relacionim) bazama podataka (DBMS)
- Na osnovu relacionog modela kreira tebele baze podataka, postavlja ograničenja na vrijednosti atributa, određuje *dinamička pravila integriteta*
- Izvršavanje *C R U D* operacija

Object/relational impedance mismatch

Problem transformacije između objektnog i relacionog modela

- Relacioni model – relacione baze podataka : : trajno čuvanje podataka
- Objektni model – objektno orjentisane aplikacije :: modelovanje poslovnih problema

Problem transformacije – *koncept identiteta*

- Objekat kao referenca i objekat kao skup vrijednosti
- Objekat postoji nezavisno od svoje vrijednosti

o1==o2 ; vs. o1.equals(o2);

- Veze između objekata identifikuju tip vezanih objekata
- U relacionom modelu identifikator N-torke je vrijednost koja se nalazi u ćeliji koja je primarni ključ relacije
- Za svaki objekat pri transformaciji odrediti atribut koji će ga jedinstveno identifikovati, i koji će imati ulogu reference u relaciji sa kojom se vrši povezivanje (spoljni ključ)

Problem transformacije – *koncept naslijeđivanja*

- Transformacija hijerarhije objekata u relacije
- Strategije transformacije:
  - Hijerarhija klasa – jedna relacija
  - Relacija po konkretnoj klasi
  - Relacije za natklasu i potklase
- Postizanje efekta specijalizacije - generalizacije

Problem transformacije – *asocijacije*

one-to-one

- Najčešće se transformacija vrši na taj način što se oba objekta predstavljaju jednom relacijom. Na taj način podaci jednog objekta proširuju se podacima drugog objekta i postaju jedinstvena relacija. Atributi relacije postaju atributi i jednog i drugog objekta.
- kupac-maticniBroj -> Kupac (KupacID, Ime, Prezime, JMBG)

one-to-many

- Kada jedan objekat pravi vezu sa više objekata nekog tipa, tada se transformacija vrši tako što se naprave posebne relacije za svaki od tipova objekata, a zatim se primarni ključ objekta koji pravi vezu pamti kod svih objekata sa kojima je on u vezi, tj. primarni ključ relacije na strani ***one*** predstavlja se kao spoljni ključ relacije na strani ***many***.
- racun-stavkaRacuna

many-to-many

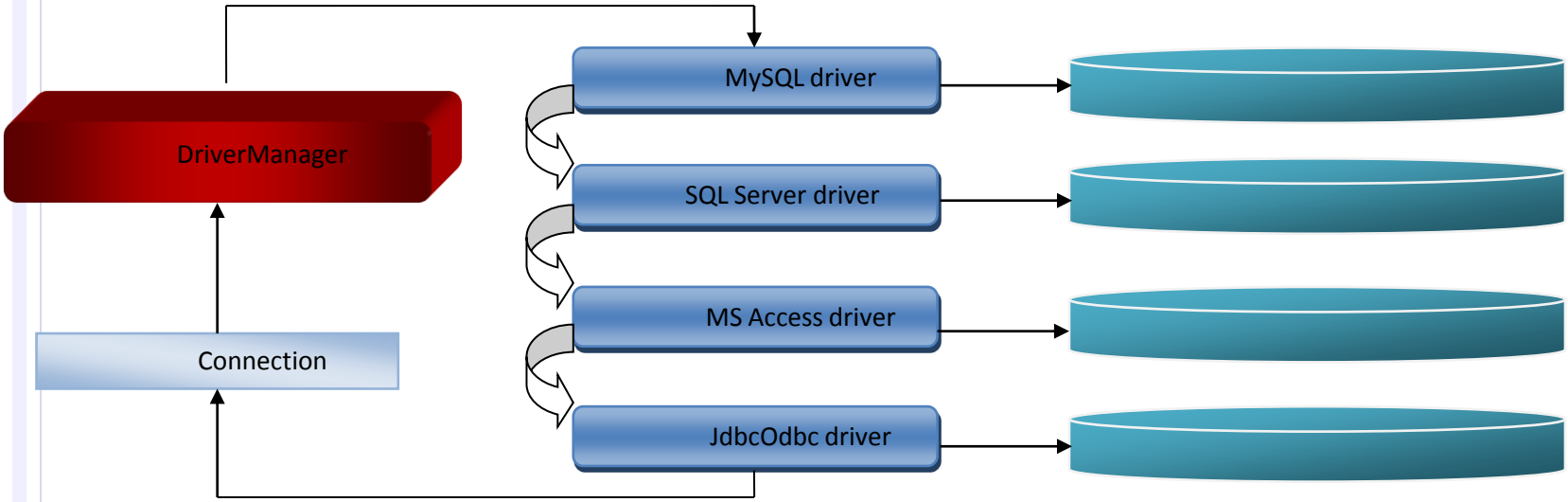
- Transformacija se vrši na taj način što se pored relacija koje se prave za svaki od tipova objekata pravi i dodatna agregirajuća relacija koju čine primarni ključevi relacija koje stupaju u vezu.
- Student-Prijavljeni ispiti

Problem transformacije – *struktura vs. ponašanje*

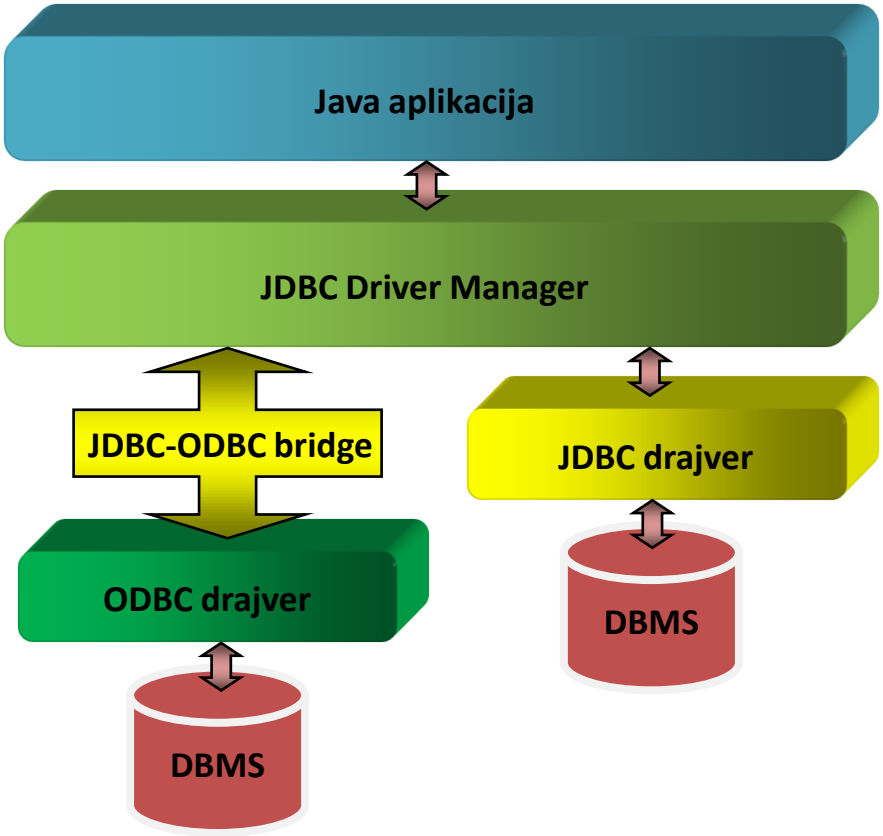
- Relacija – podaci kao interfejs
- Objekat – ponašanje kao interfejs
- Preslikavanje atributa objekta u atribute relacije
  - Poslijedica enkapsulacije: vezivanje atributa relacije za metodu koja pristupa željenom atributu objekta

JDBC

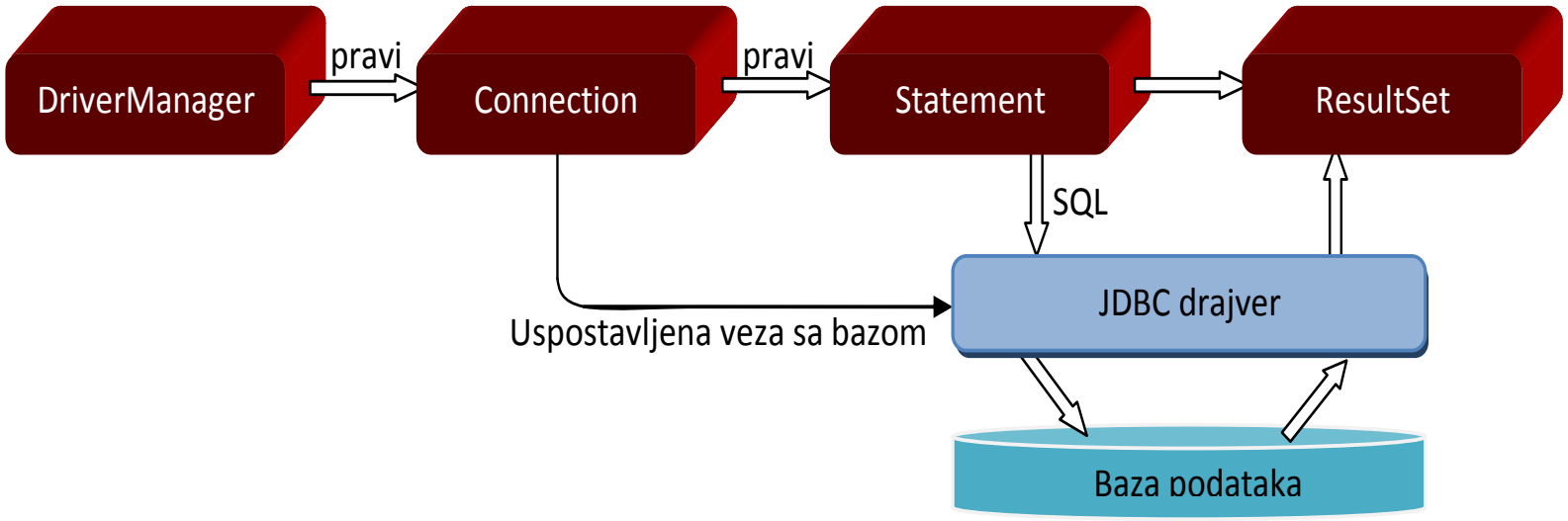
- Java Database Connectivity
- Povezivanje na DBMS, manipulacija podacima
- Nezavisnost od konkretnog DBMS-a
- SQL – specifične naredbe za različite DBMS



JDBC – komunikacija sa DBMS



JDBC – interakcija Java aplikacije i DBMS



JDBC – primjer (`Proizvod.java`)

```
public class Proizvod  {
    private int proizvodID;
    private String naziv;
    private double cena;
    public Proizvod() {
    }
    public Proizvod(int proizvodID, String naziv, double cena) {
        this.proizvodID=proizvodID;
        this.naziv=naziv;
        this.cena=cena;
    }
    public int getProizvodID() {
        return proizvodID;
    }
    public void setProizvodID(int proizvodID) {
        this.proizvodID = proizvodID;
    }
    public String getNaziv() {
        return naziv;
    }
    public void setNaziv(String naziv) {
        this.naziv = naziv;
    }
    public double getCena() {
        return cena;
    }
    public void setCena(double cena) {
        this.cena = cena;
    }
}
```

JDBC – primjer (`Main.java`)

```
...
/* učitavanje MYSQL drajvera */
Class.forName("com.mysql.jdbc.Driver").newInstance();

String url = "jdbc:mysql://127.0.0.1:3306/IME_BAZE?user=USER&password=PASS";
/* Otvaranje konekcije na bazu */
Connection con = DriverManager.getConnection(url);

/* pravljenje objekta klase Statement */
Statement stmt = con.createStatement();

/* izvršavanje upita */
ResultSet rs = stmt.executeQuery("SELECT * FROM Proizvod");
List<Proizvod> listaProizvoda = new ArrayList<Proizvod>();
while ( rs.next() )
        {
  Proizvod p = new Proizvod();
  p.setProizvodID(rs.getInt("proizvodID"));
  p.setNaziv(rs.getString("naziv"));
  p.setCena(rs.getDouble("cena"));
  listaProizvoda.add(p);
}
...
```

Alati za objektno-relaciono mapiranje

Automatizacija procesa perzistencije

- Automatsko generisanje SQL-a
- Sakrivanje JDBC koda

Kategorizacija ORM alata

Pure relational

- aplikacija,bazirana na relacionom modelu i direktno se oslanja na SQL operacije. U ovakvim aplikacijama se najčešće dio aplikacione logike prebacuje u bazu podataka korišćenjem uskladištenih procedura.

Light object mapping

- Entiteti su predstavljeni klasama koje se ručno preslikavaju u tabele baze podataka. SQL i JDBC kod se takođe piše ručno, ali se od sloja poslovne logike sakriva korišćenjem softverskih paterna

Medium object mapping

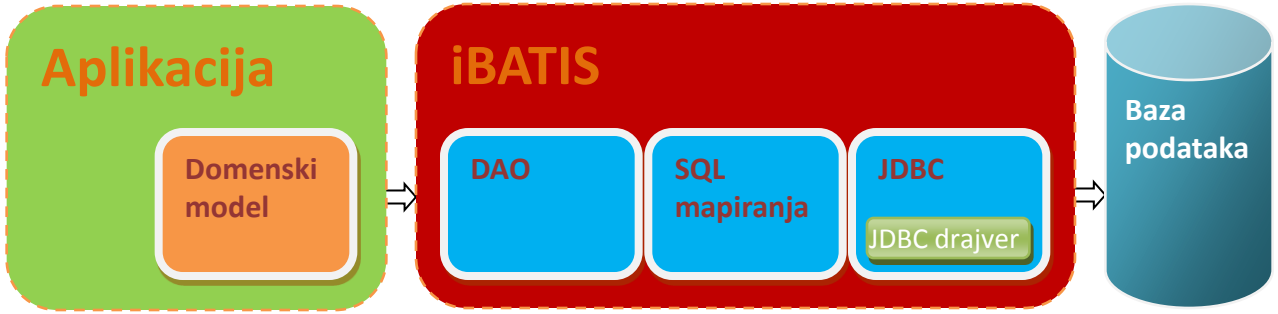
- Aplikacija se projektuje na osnovu objektnog modela. SQL se automatski generiše. Podržava asocijacije između objekata, a upiti najčešće kreirani korišćenjem nekog objektno-orjentisanog jezika. Izbjegava se korišćenje uskladištenih procedura.

Full object mapping

- Podržavaju: kompoziciju, naslijeđivanje, polimorfizam. Realizuju tzv. transparentnu perzistentnost. Mehanizmi keširanja i punjenja podataka iz baze u memoriju su ”nevidljivi” za aplikaciju.

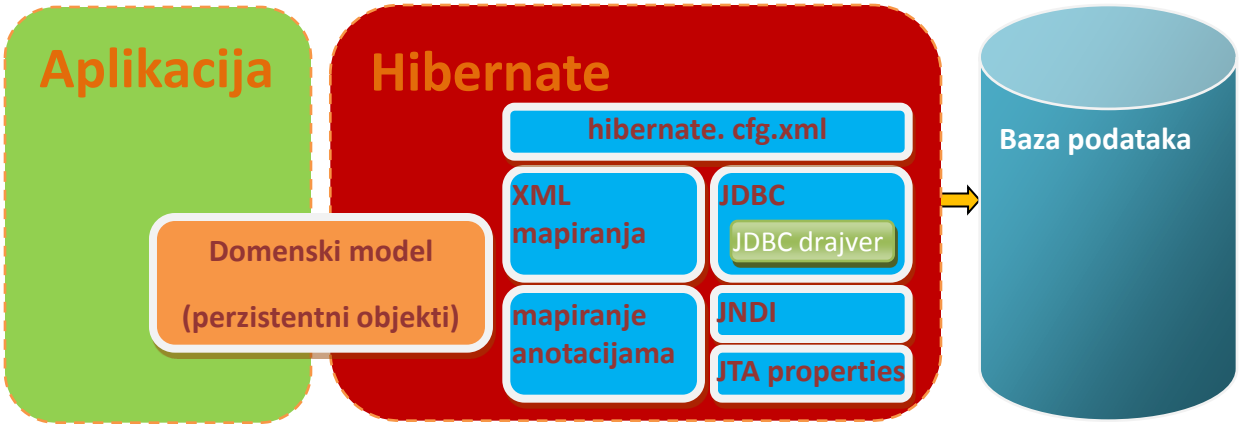
ORM alati :: *iBatis*

- Direktna upotreba SQL-a
- Ne zahtijeva precizno poklapanje modela podataka
- Domenski model se povezuje sa SQL upitom kroz XML datoteke



ORM alati :: *Hibernate*

- Podrška za objektno orjentisano modelovanje
- Nezavisan od DBMS korišćenjem SQL dijalekta
- HQL – objektni upitni jezik
- Transpatentna perzistencija POJO + XML mapiranje ili mapiranje anotacijama
- Upravljanje transakcijama, keširanje, lazy loading, connection pooling...



ORM alati :: *JPA*

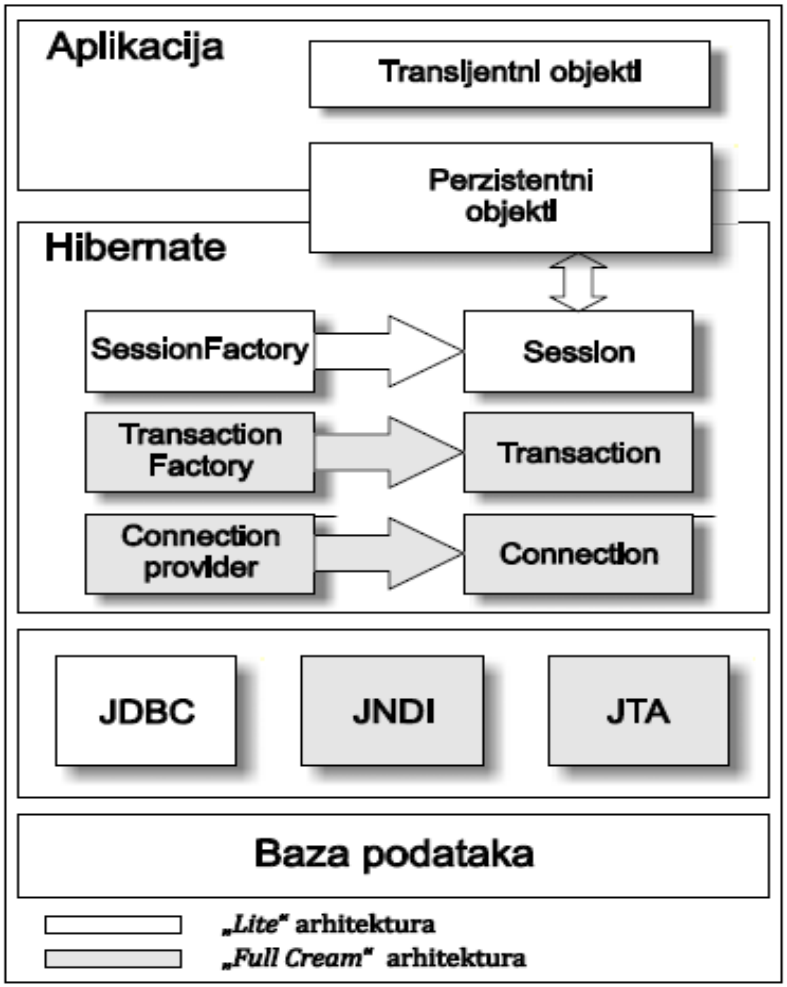
- Pojednostavljivanje izrade EE i SE aplikacija uvođenjem jedostavnijeg perzistencionog mehanizma
- Okupljanje cjelokupne Java zajednice oko jedinstvenog – standardnog perzistencionog API-a
- Zasnovan na dobroj praksi kod postojećih projekata
- Korišćenje sa i bez EE kontejnera
- Implementacije JPA
  - Hibernate (JBoss)
  - TopLink (Oracle)
  - JDO (Oracle, BEA)

Hibertnate

Okvir za Objektno Relaciono Mapiranje

Hibernate :: osobine

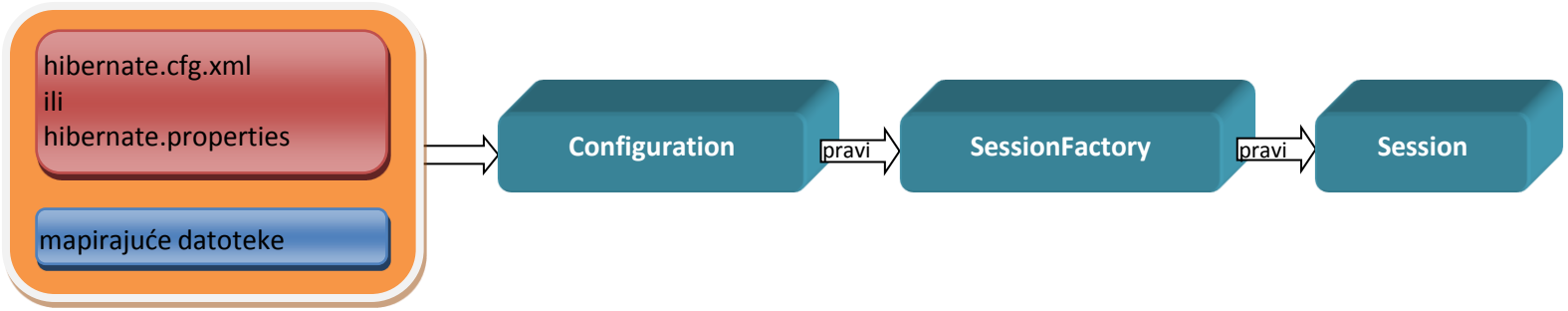
Hibernate :: arhitektura



Hibernate :: konfiguracija

Polje	Opis
hibernate.connection.url	JDBC URL do instance baze.
hibernate.connection.driver_class	Klasa JDBC drajvera.
hibernate.connection.username	Korisničko ime za pristup bazi.
hibernate.connection.password	Lozinka za pristup bazi.
hibernate.dialect	SQL dijalekt koji zavisi od baze. Koristi se kako bi se izbjegli problemi koji mogu da nastanu korišćenjem SQL naredbi koje su specifične za određeni sistem za upravljanje bazom podataka.
hibernate.connection.pool_size	Definiše broj konekcija koje se otvaraju i čekaju da budu iskorišćene. Na ovaj način poboljšavaju se performanse aplikacije.
hibernate.connection.autocommit	Podešavanje automatskog potvrđivanja transakcije poslije svakog izvršenog upita. Ne preporučuje se.
hibernate.jdbc.fetch_size	Podešavanje broja redova koji će iz baze podataka biti učitan u memoriju. Na ovaj način racionalizuje se korišćenje memorije.
hibernate.transaction.auto_close_session	Automatski zatvara sesiju poslije transakcije.
hibernate.connection.isolation	Nivo izolacije transakcije u JDBC konekciji.
hibernate.show_sql	Pravi Log generisanih SQL naredbi.
hibernate.cache.use_query_cache	Određuje da li će se koristiti keširanje upita.

Hibernate :: konfiguracija >> API



Hibernate :: koraci u pravljenju aplikacije

1. Pripremanje baze podataka (pravljenje tabela, podešavanje referencionalnih integriteta, pokretanje instance baze podataka)
2. Pravljenje domenskog modela aplikacije
3. Definisanje mapiranja objekata i njihovih veza na tabele baze podataka korišćenjem XML datoteka ili anotacija
4. Pravljenje hibernate.cfg.xml ili hibernate.properties datoteke sa definiciom konfiguracije
5. Pravljenje HibernateUtility klase
6. Pravljenje klasa koje pozivaju funkcionalnosti Hibernate okvira

Hibernate :: potrebne biblioteke

- Mapiranje XML datotekama
  - {Hibernate_Distribution}\HibernateX.jar <www.hibernate.org>
  - {Hibernate_Distribution}\lib\required*.jar (u novijim verzijama ova biblioteka je dovoljna)
  - {Hibernate_Distribution}\lib\optional\c3p0*.jar
  - {apache-log4j_Distribution}\log4j-*.jar <logging.apache.org/log4j/>
  - {slf4j-*_Distribution}\slf4j-log4j12-*.jar < www.slf4j.org >
- Mapiranje anotacijama
  - Sve što treba za mapiranje XML datotekama <www.hibernate.org>
  - {HibernateAnnotations_Distribution}\hibernate-annotations.jar
  - {HibernateAnnotations_Distribution}\lib\hibernate-commons-annotations.jar
  - {HibernateAnnotations_Distribution}\lib\ejb3-persistence.jar

+ Drajver baze podataka

Hibernate (hibernate.cfg.xml)

```
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
    <session-factory>
      <property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
      <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/hibernatetest</property>
      <property name="dialect">org.hibernate.dialect.MySQLDialect</property>
      <property name="hibernate.connection.username">root</property>
      <property name="hibernate.connection.password">root</property>
          <property name="hibernate.c3p0.min_size">5</property>
          <property name="hibernate.c3p0.max_size">20</property>
          <property name="hibernate.c3p0.timeout">300</property>
          <property name="hibernate.c3p0.max_statements">50</property>
          <property name="hibernate.c3p0.idle_test_period">3000</property>
        <property name="show_sql">true</property>
        <mapping resource="model/Proizvod.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Hibernate (`log4j.properties`)

```
log4j.appender.stdout=org.apache.log4j.ConsoleAppender
log4j.appender.stdout.Target=System.out
log4j.appender.stdout.layout=org.apache.log4j.PatternLayout
log4j.appender.stdout.layout.ConversionPattern=%d{ABSOLUTE} %5p %c{1}:%L - %m%n

log4j.rootLogger=warn, stdout

log4j.logger.org.hibernate=debug

log4j.logger.org.hibernate.type=info

log4j.logger.org.hibernate.tool.hbm2ddl=debug
```

Hibernate – primjer (`model.Proizvod.java`)

```
package model;
public class Proizvod  {
    private int proizvodID;
    private String naziv;
    private double cena;
    public Proizvod() {    }
    public Proizvod(int proizvodID, String naziv, double cena) {
        this.proizvodID=proizvodID;  this.naziv=naziv;   this.cena=cena;
    }
    public int getProizvodID() {
        return proizvodID;
    }
    public void setProizvodID(int proizvodID) {
        this.proizvodID = proizvodID;
    }
    public String getNaziv() {
        return naziv;
    }
    public void setNaziv(String naziv) {
        this.naziv = naziv;
    }
    public double getCena() {
        return cena;
    }
    public void setCena(double cena) {    this.cena = cena;  }

    public String toString(){
      return "ProizvodID: "+getProizvodID()+"\tNaziv: "+naziv+"\tCena: "+cena;
    }
}
```

Hibernate (`Proizvod.hbm.xml`)

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd">
<hibernate-mapping package="model">

    <class name="model.Proizvod" table="Proizvod" >
        <id name="proizvodID" column="proizvodID">
             <generator class="assigned"/>
        </id>
        <property name="naziv" column="naziv"/>
        <property name="cena" column="cena" type="double"/>
    </class>

</hibernate-mapping>
```

hbm.xml datoteke ne moraju biti u istom direktorijumu kao i klase koju mapiraju

Hibernate (HibernateUtility.java)

```
package util;

import …;

public class HibernateUtility {
    private static final SessionFactory sessionFactory;

    static {
     try{
        // Kreira SessionFactory na osnovu hibernate.cfg.xml
      StandardServiceRegistry standardRegistry = new StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
      Metadata metaData = new MetadataSources(standardRegistry).getMetadataBuilder().build();
      sessionFactory = metaData.getSessionFactoryBuilder().build();
     }catch(Throwable ex){
      System.err.println("Inicijalno kreiranje SessionFactory nije uspjelo! " + ex);
            throw new ExceptionInInitializerError(ex);
     }
    }

    public static SessionFactory getSessionFactory(){
        return sessionFactory;
    }
}
```

Hibernate (main.Main.java)

```
package main;
import java.util.*;
import model.Proizvod;
import org.hibernate.Query;
import org.hibernate.Session;
import util.HibernateUtility;

public class Main {
public void kreirajIUbaciProizvod(int id, String naziv, double cena){
        Session session=HibernateUtility.getSessionFactory().openSession();
        session.beginTransaction();
        Proizvod  p=new Proizvod();
        p.setProizvodID(id);
        p.setNaziv(naziv);
        p.setCena(cena);
        session.saveOrUpdate(p);
        session.getTransaction().commit();
    }
    public List listProizvod(){
        Session session=HibernateUtility.getSessionFactory().openSession();
        session.beginTransaction();
        Query query=session.createQuery("from Proizvod");
        List lista=query.list();
        session.getTransaction().commit();
        return lista;
    }
}
```

Hibernate (main.Main.java)

```
package main;
import java.util.*;
import model.Proizvod;
import org.hibernate.Query;
import org.hibernate.Session;
import util.HibernateUtility;

public class Main {
…
 public static void main(String[] args) {
        Main m=new Main();
        m.kreirajIUbaciProizvod(4,"Brasno",40);
        List l=m.listProizvod();
        Iterator i=l.iterator();
        while(i.hasNext()){
            Object p=i.next();
            System.out.println(p);
        }
        HibernateUtility.getSessionFactory().close();
    }
…
}
```

Hibernate :: stanja objekata

Transijentno (Transient)

- Objekat je transijentan ukoliko je kreiran korišćenjem new operatora, i još uvijek nije povezan sa Hibernate Session objektom. Objekat nema svoju perzistentnu reprezentaciju u bazi podataka, i ako se radi o objektu sa automatskim generisanjem primarnog gljuča, još uvijek mu nije dodijeljen identifikator. Ukoliko aplikacija nema referencu na transijentni objekat, taj objekat biva uništen od strane *garbage collector* mehanizma.

Perzistentno (Persistent)

- Perzistentni objeti imaju odgovarajuću reprezentaciju u bazi podataka i vrijednost identifikatora. Objekat prethodno može da bude snimljen u bazu ili učitan iz baze, ali u oba slučaja mora da bude povezan sa Hibernate Session objektom. Kada je objekat u perzistentnom stanju Hibernate automatski prepoznaje sve promjene na objektu, i sinhronizuje stanje objekta sa stanjem u bazi podataka po završetku transakcije.

Odvojeno (Detached)

- Objekat se nalazi u odvojenom stanju ukoliko je prethodno bio u perzistentnom, ali je u međuvremenu Session objekat, za koji je bio vezan, zatvoren. Referenca na odvojen objekat ostaje validna, što znači da je moguće pristupiti i promijeniti stanje ove instance. Odvojen objekat može ponovo da se prevede u perzistentno stanje ukoliko se poveže sa novim Session objektom.

CRUD operacije

Realizacija osnovnih operacija za manipulaciju podacima u bazi podataka (Create, Retrieve, Update, Delete

Hibernate :: Create

```
public void kreirajIUbaciProizvod(int id, String naziv, double cena){

    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    Proizvod  p=new Proizvod();
    p.setProizvodID(id);
    p.setNaziv(naziv);
    p.setCena(cena);
    Integer dodijeljeniID=(Integer) session.save(p);//saveOrUpdate
    session.getTransaction().commit();

}
```

Hibernate :: Retrieve

```
public Proizvod procitajProizvod(int i){

    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    Proizvod p=(Proizvod)session.load(Proizvod.class, i);//p ili exception
    session.getTransaction().commit();
    return p;

}
```

```
public Proizvod procitajProizvod(int i){

    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    Proizvod p=(Proizvod)session.get(Proizvod.class, i);//p ili null
    session.getTransaction().commit();
    return p;

}
```

Hibernate :: Update

- Ako je objekat u perzistentnom stanju, svaka promjena nad njim automatski će promijeniti stanje u tabeli.

- Ako je objekat transijentan ili odvojen (detached) tada se update realizuje na sledeći način:

```
public void azurirajProizvod(Proizvod p){
    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    session.update(p);
    session.getTransaction().commit();
}
```

Hibernate :: Delete

```
public void obrisiProizvod(Proizvod p){
    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    session.delete(p);
    session.getTransaction().commit();
}
```

Hibernate :: HQL :: Querry

```
public List listProizvod(){
    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    Query query=session.createQuery("from Proizvod");
    List lista=query.list();
    session.getTransaction().commit();
    return lista;
}
```

Hibernate :: HQL :: Criteria

```
public List criteriaProizvod(){
    Session sesija=HibernateUtility.getSessionFactory().openSession();
    sesija.beginTransaction();
    Criteria crit = sesija.createCriteria(Proizvod.class);
    //postavlja maksimalni broj procitanih slogova
    crit.setMaxResults(3);
    //uvodi kriterijum da naziv proizvoda treba da pocinje slovom 'p'
    crit.add(Restrictions.like("naziv", "p%"));
    List lista = crit.list();
    sesija.getTransaction().commit();
    System.out.println(lista);
    return lista;
}
```

Hibernate :: HQL :: Example

```
public Proizvod loadByExample() {
      Session sesija=HibernateUtility.getSessionFactory().openSession();
      sesija.beginTransaction();
      Proizvod p = new Proizvod();
      p.setNaziv("Cokoladna Bananica");
      p.setCijena(70);
      Criteria crit = sesija.createCriteria(Proizvod.class);
      crit.add(Example.create(p));
      p = (Proizvod) crit.uniqueResult();
      sesija.getTransaction().commit();
      return p;
}
```

Hibernate :: SQL

```
public List listProizvodSQL(){

    Session session=HibernateUtility.getSessionFactory().openSession();
    session.beginTransaction();
    String sql = "select {proizvod.*} from Proizvod proizvod";
    SQLQuery query = session.createSQLQuery(sql);
    query.addEntity("proizvod", Proizvod.class);
    List lista = query.list();
    session.getTransaction().commit();
    return lista;

}
```

Hibernate :: Mehanizmi za poboljšanje performansi

- Lazy loading
- Keširanje

Hibernate Search

Full-text search – "Google(tm) for your entities"

Hibernate Search:: integracija

- Apache Lucene
- Elasticsearch

Eksterni repozitorijum indeksiranih polja, automatska sinhronizacija

- hibernate.cfg.xml mora da sadrži informaciju o lokaciji repozitorijuma

```
...
    <property name="hibernate.search.default.indexBase">
            /hibernatesearchindexes  <!— putanja do repozitorijuma --!>
    </property>
...
```

Hibernate Search:: Anotacije nad entitetima modela

```
@Entity
@Indexed
public class Dobavljac {

@Id
private int dobavljacID;
@Field(index=Index.YES, analyze=Analyze.YES, store=Store.NO)
private String naziv;
@Field(index=Index.YES, analyze=Analyze.YES, store=Store.NO)
private String adresa;
...
}
```

Hibernate Search:: Primer izvršenja upita

```
public List<Dobavljac> pronadjiDobavljaceFullText(String keyword) {
        List dobavljaci = null;
        try {
            Session session = HibernateUtility.getSessionFactory().openSession();
            //kreiranje sesije za FullText upite
            FullTextSession fullTextSession = Search.getFullTextSession(session);
            //pravljenje/osvjezavanje indeksera
            fullTextSession.createIndexer().startAndWait();
            //pocetak transakcije
            fullTextSession.beginTransaction();
            QueryBuilder qb =
fullTextSession.getSearchFactory().buildQueryBuilder().forEntity(Dobavljac.class).get();
            //pravljenje Lucene upita
            org.apache.lucene.search.Query luceneQuery = qb
                    .keyword()
                    .onFields("naziv", "adresa")
                    .matching(keyword)
                    .createQuery();
            // konveryija Lucene upita u Query
            Query query = fullTextSession.createFullTextQuery(luceneQuery,Dobavljac.class);
            dobavljaci = query.list(); // izvrsenje upita
            return dobavljaci;
        } catch (InterruptedException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
        return dobavljaci;
    }
```

Objektno - Relaciono Mapiranje

doc. dr Ilija Antović

ilijaa@fon.bg.ac.rs