*Računarstvo i informatika*

*Katedra za računarstvo*
*Elektronski fakultet u Nišu*

Sistemi baza podataka

# Objektno-relacioni maperi

Letnji semestar 2015

# Sadržaj

- Pristup podacima
- Layered architecture
- Modeliranje domena
- O/R maperi
- NHibernate

# Pristup podacima

- Većina aplikacija ime potrebu da vrši određenu obradu podataka.
- Podaci mogu poticati iz velikog broj različitih izvora:
  - Relacione beze podataka
  - Objektno – relacione baze podataka i Objektne baze podataka
  - NoSQL baze podataka
  - XML dokumenti
  - Binarne datoteke
- Svaki tip izvora podataka ima određene specifičnosti i skup dozvoljenih operacija.
- Različiti delovi aplikacije obrađuju podatke u različitom obliku pa je neophodno obezbediti stalnu transformaciju podataka iz jednog u drugi oblik (vrste u relacionoj tabelu se transformišu u memorijske objekte, memorijski objekti se transformišu u XML strukture i sl.).

# Pristup podacima

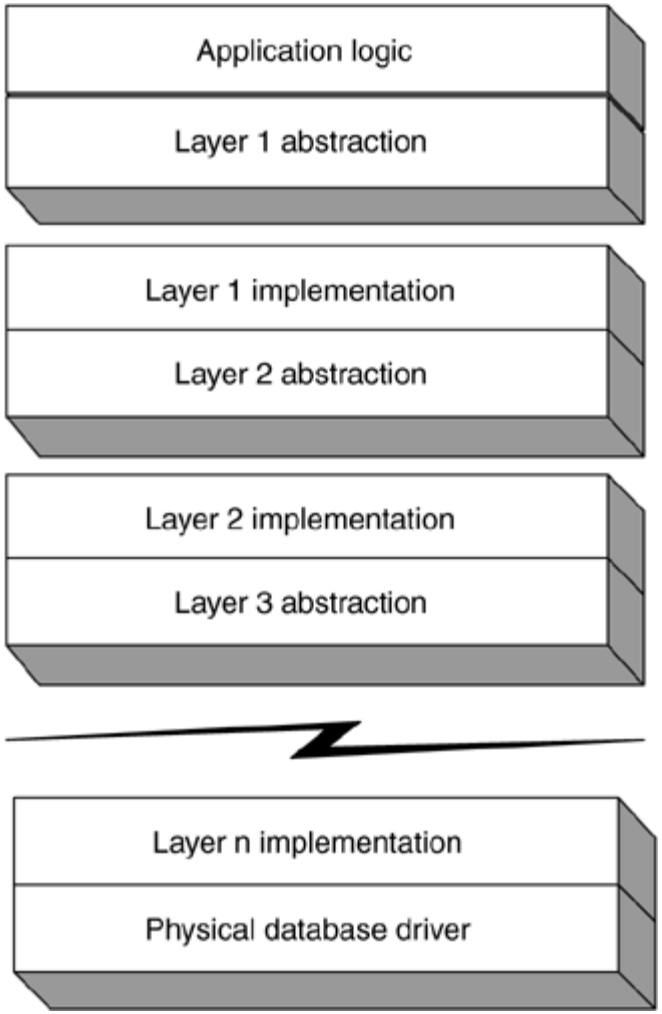
- Jedan od načina na koji može biti rešen problem pristupanja podacima je izdvajanje koda koji pristupa podacima u posebnu komponentu/sloj/biblioteku.
- Prilikom izdvajanja posebne komponente za pristup podacima treba voditi računa:

1. Kompleksnost rešenja
   - Smanjuje se kompleksnost problema koji se rešava
   - Jednostavnije održavanje jer se komponenta može razvijati, testirati i menjati potpuno nezavisno od ostatka sistema
   - Jednostavnija optimizacija pristupa podacima
   - Povećana složenost implementacije
   - Pad performansi usled kompleksnije strukture
2. Jednostavniji interfejs
   - Interfejs za pristupanje podacima postaje jednostavniji
   - Ograničena funkcionalnost u pristupanju relacionoj bazi podataka

# Pristup podacima

3. Performanse
   - Optimizacija pristupa podacima može biti od kritičnog značaja za performanse sistema
   - Optimizacija pristupa podacima je iterativni proces
   - Optimizacija se vrši: promenom parametara funkcionisanja DBMS-a, promenom u šemi relacione baze podataka, zamenom programskog interfejsa (API) za pristup podacima.
4. Fleksibilnost sistema
   - Sistem postaje nezavisna od izvora podataka
   - Povećana je kompleksnost sistema
5. Legacy izvori podataka
   - Retke su situacije u kojima se sistem projektuje od početka
   - Česte su situacije u kojima se podaci kojima se pristupa nalaze u već postojećim sistemima, koji se ne mogu menjati a ne odgovaraju potrebama

# Layered architecture



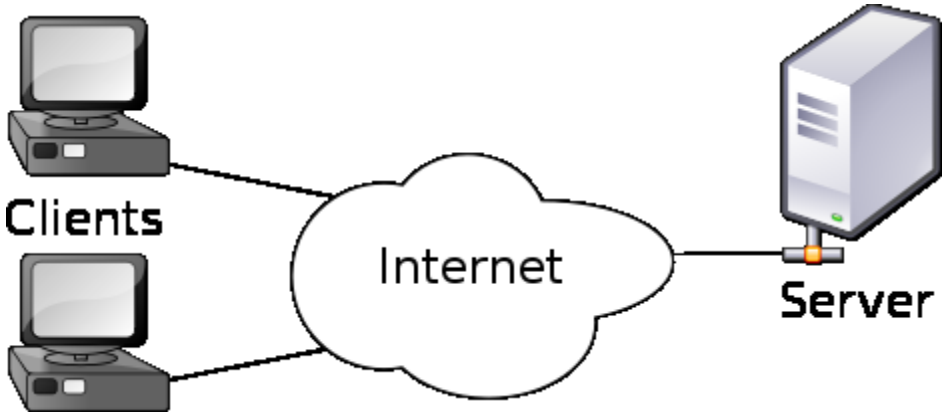

- Slojevita arhitektura je uobičajena tehnika koja se koristi za razdvajanje kompleksnog softverskog sistema na delove (slojeve).
- Ova arhitektura omogućava da se različiti delovi sistema izoluju u različite nivoe abstrakcije.
- Ova arhitektura podrazumeva postojanje većeg broja podsistema koji su raspoređeni u slojeve.
- Svaki sloj višeg nivoa koristi servise koje obezbeđuje sloj nižeg nivoa. Pri tome je sloj nižeg nivoa potpuno nesvestan postojanja sloja višeg nivoa.
- Svaki sloj skriva detalje o slojevima koji se nalaze ispod njgea od sloje koji se nalaze iznad njega.

# Layered architecture

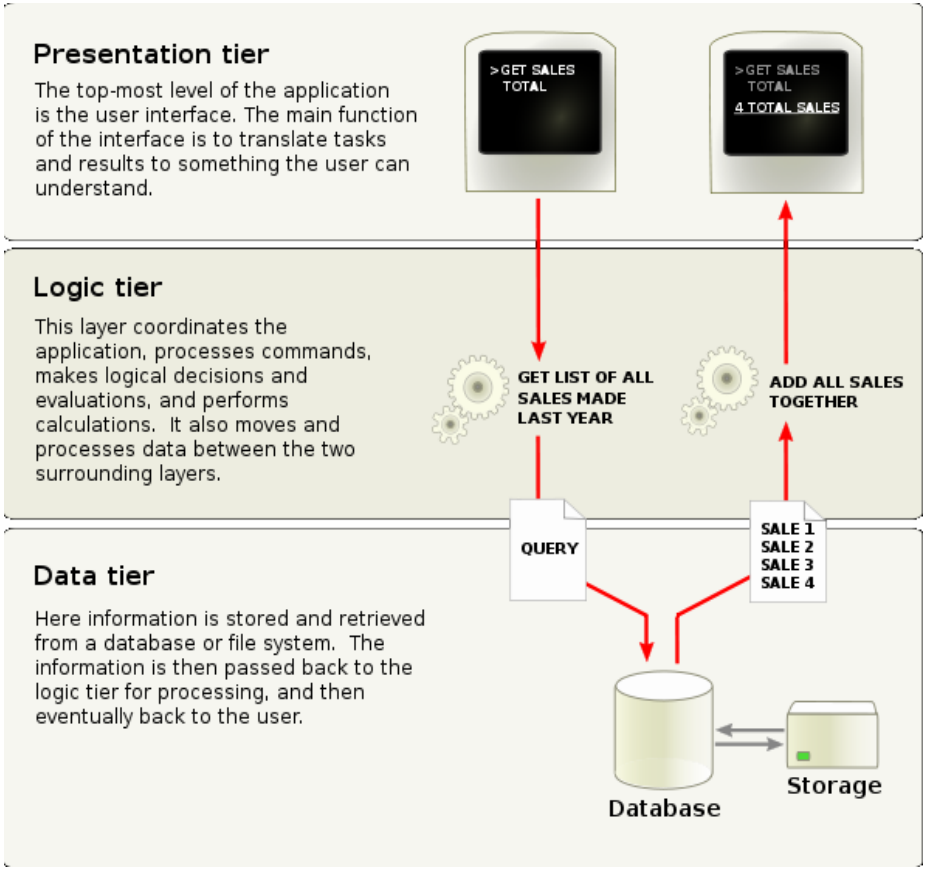
- Prednosti arhitekture:
  - Svaki sloj se može tretirati kao celina bez potrebe poznavanja detalja o ostalim slojevima
  - Svaki sloj se može zameniti alternativnom implementacijom servisa koje sloj treba da nudi (čak i dinamički u toku rada sistema)
  - Minimizuju se međuzavisnosti između različitih slojeva
  - Slojevi predstavljaju dobru osnovu za standardizaciju rešenja nekog problema
  - Implementirani slojevi se mogu koristiti u različitim sistemima (samim tim je ubrzan razvoj i olakšano održavanje)
- Nedostaci arhitekture:
  - Problem kaskadnih promena – izmena u najnižem sloju često mora da se propagira kroz čitavu arhitekturu
  - Pad performansi – veliki broj nivoa može da dovede do pada performansi sistema (neophodan veliki broj transformacija između nivoa)
  - Problem izbora nivoa i servisa koje će implementirati
  - Kompleksnija inicijalizacija sistema

# Layered architecture

- Dvoslojna arhitektura:
  - Klijent (korisnički interfejs i poslovna logika)
  - Server (baza podataka)



# Layered architecture




- Troslojna arhitektura:
  - Prezentacioni sloj
  - Sloj poslovne logike ili domenski sloj
  - Sloj podataka
- Dominantna arhitektura kod Web aplikacija
- Različiti slojevi se mogu izvršavati na različitim lokacijama

# Modeliranje domena

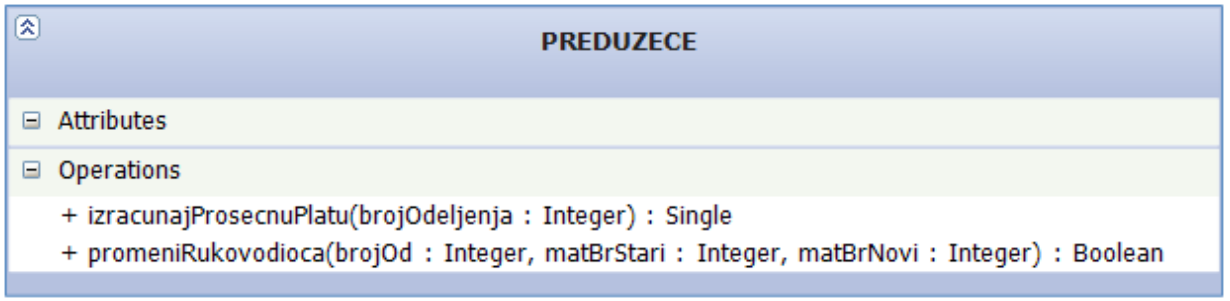
- Domenska logika ili poslovna logika
- Posao koji sistem zaista obavlja odnosno implementacija različitih poslovnih procesa.
- Zbog toga veliku pažnju treba posvetiti ovom sloju i njegovim vezama ka drugim slojevima (posebno ka izvoru podataka)
- Za modeliranje domena mogu da se iskoriste tri projekta obrasca:
  1. Transaction Script
  2. Table Module
  3. Domain Model

# Modeliranje domena

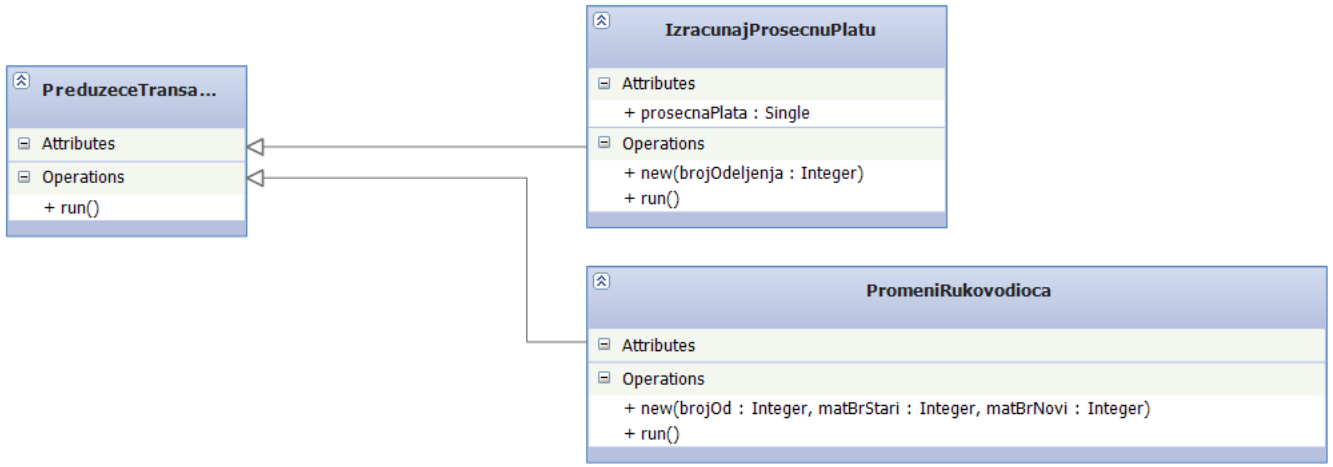
- **Transaction Script** organizuje poslovnu logiku u vidu procedura, pri čemu je svaka procedura zadužena za obradu jednog zahteva od strane prezentacionog sloja.
- Većina poslovnih aplikacija se mogu tretirati kao niz transakcija pri čemu transakcije variraju od jednostavnog pribavljanja podataka pa do kompleksnih izračunavanja.
- Transaction Script logiku organizuje u jednu proceduru pri čemu se vrši direktna komunikacija sa izvorom podataka.
- Transaction script se može implementirati na nekoliko načina:
  - Korišćenjem globalnih procedura
  - Grupisanje transakcija u jednu klasu ili više klasa koje obuhvataju povezane transakcije
  - Korišćenjem Command projektnog obrasca

# Modeliranje domena

Klasa koja sadrži Transaction Script implementacije




Transaction Script implementiran kao Command pattern



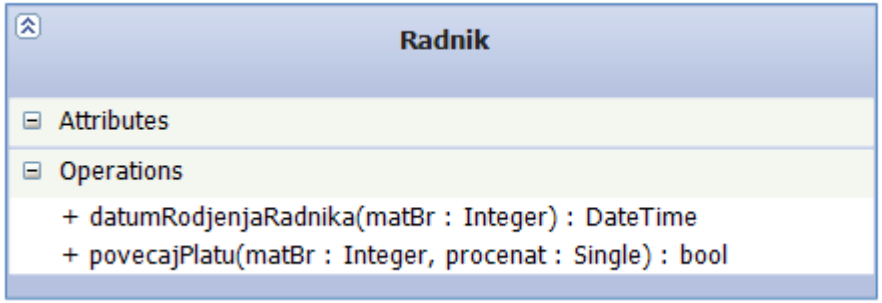
# Modeliranje domena

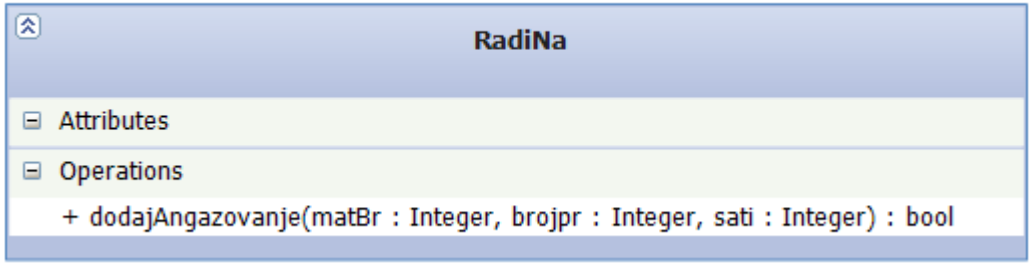
- Transaction Script karakteristike:
  - Osnovna prednost korišćenja je jednostavnost
  - Koristi se u situacijama kada je domenska logika jako jednostavna
  - Sa povećanjem kompleksnosti poslovne logike rastu problemi vezani za projektovanje i održavanje koda
    - Ograničen interfejs za pristup podacima
    - Kod postaje teško razumljiv
    - Dupliranje koda usled transakcija koje obavljaju slične poslove

# Modeliranje domena

- **Table Module** : Za svaku tabelu/pogled/upit postoji posebna instanca koja sadrži kompletnu poslovnu logiku za rad sa vrstama te tabele/pogleda/upita.
- Podaci i poslovna logika su "upakovani" zajedno i maksimalno se koriste prednosti relacionog modela.
- Poslovna logika je implementirana kroz niz metoda koje pristupaju odgovarajućoj tabeli.
- Table Modul se može implementirati kao instanca klase ili kao niz statičkih funkcija.
- Za pristup pojedinačnim slogovima neophodno je proslediti parametar koji će obezbediti njihovu identifikaciju.
- Nije pogodan za predstavljanje kompleksne logike:
  - Nema direktne veze između instanci
  - Problem sa implementacijom polimorfizma

# Modeliranje domena

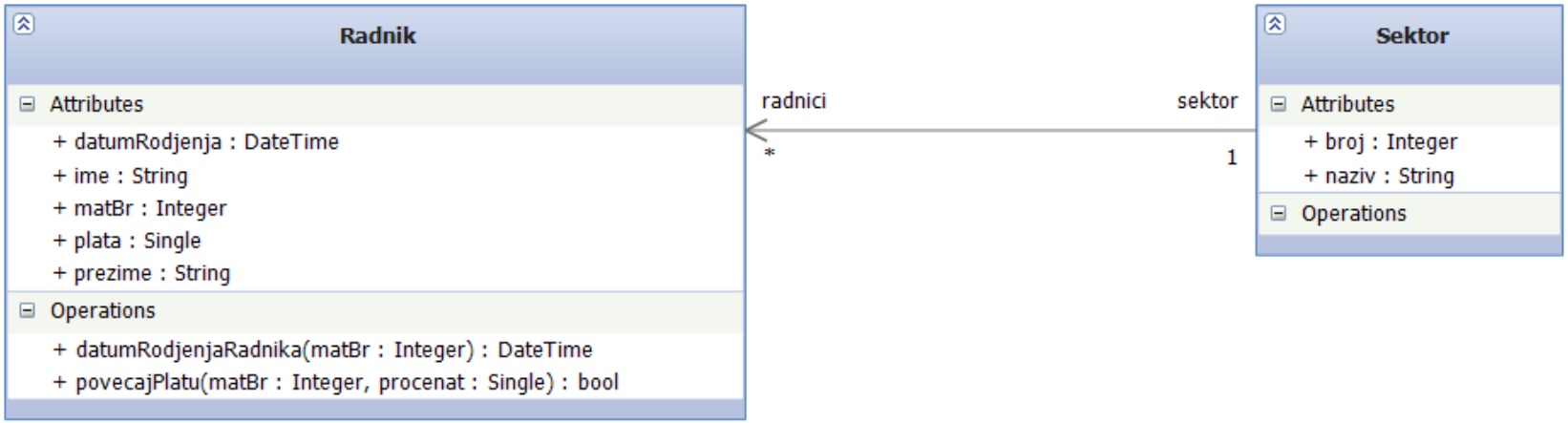


# Modeliranje domena

- **Domain Model**: OO model domena koji uključuje i podatke i poslovnu logiku.
- Mogu da postoje objekti koji samo predstavljaju podatke, objekti koji predstavljaju poslovnu logiku ili objekti koji sadrže i jedno i drugo.
- Svaki objekat predstavlja neki od domenskih entiteta.
- Koristi se u situacijama kada je potrebno modelirati kompleksne sisteme.
- **Problem su razlike koje postoje između OO modela i relacionog modela.** Neophodna stalna transformacija između jednog i drugog modela podataka.

# Modeliranje domena

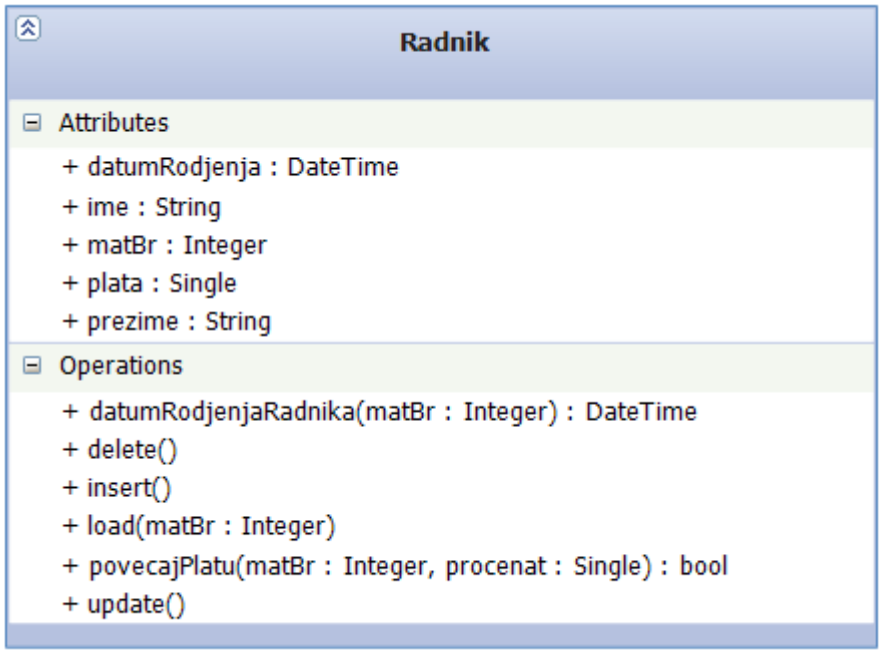


# Modeliranje domena

- Za implementaciju Domain Model mogu se iskoristiti dva pristupa:

1. **Active Record**
   - Pored podataka i poslovne logike objekat uključuje i metode za rad sa relacionom bazom podataka
   - Može zameniti Table Modul kada se kreira po jedna klasa za svaku tabelu u relacionoj bazi podataka
   - Active Record predstavlja slog u tabeli odnosno za svaki slog u tabeli može se kreirati instanca objekta
2. **Data Mapper**
   - Pored objekata koji sadrže podatke i poslovnu logiku postoje posebni objekti koji su zaduženi za komunikciju sa relacionom bazom podataka (transformaciju iz jednog modela u drugi)
   - OO model je potpuno odvojen od baze podataka i ne mora da bude svestan njenog postojanja

# Modeliranje domena

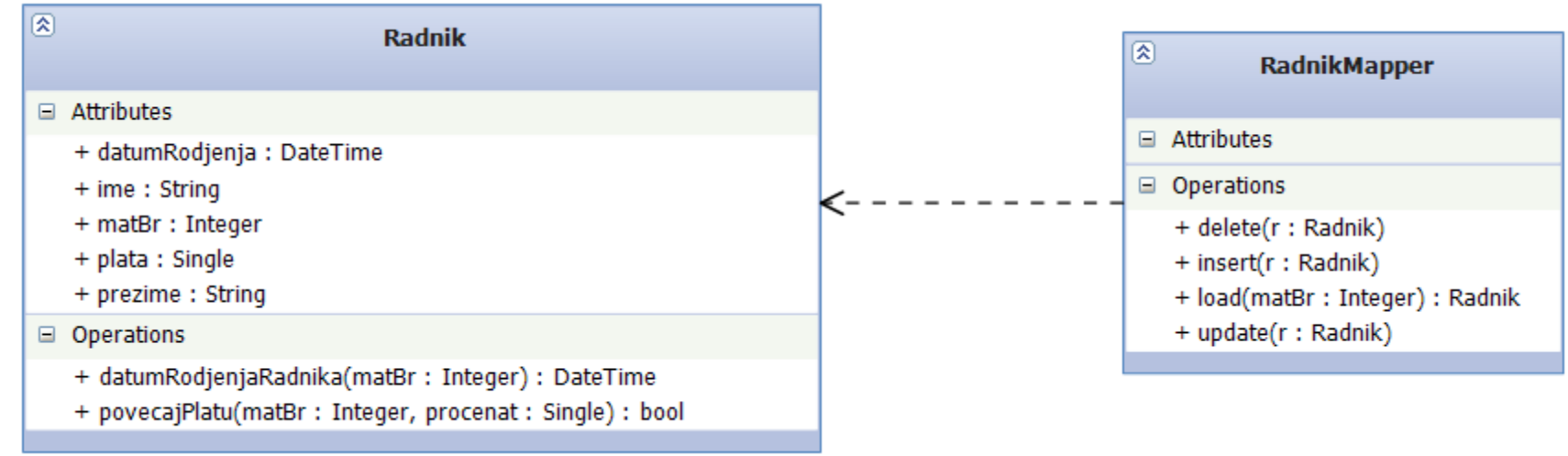



Active Record:
- podaci
- poslovna logika
- CRUD operacije
  - insert
  - delete
  - update
  - load

- Svestan postojanja baze podataka

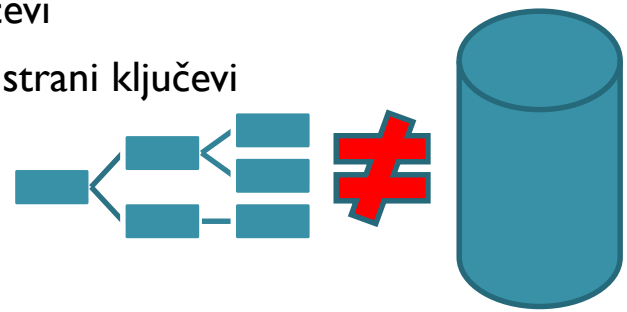
# Modeliranje domena

**Data mapper**



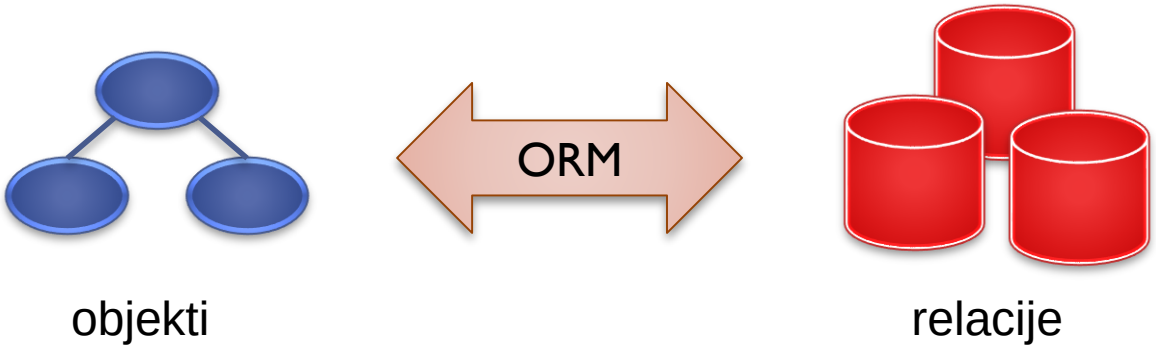
# O/R maperi

- **"Object-Relational Impedance Mismatch" (Paradigm Mismatch)**
- Skup konceptualnih i tehničkih problema koji se javljaju u situacijama kada OO sistem treba da koristi podatke u relacionoj bazi podataka, naročito u situacijama kada je objekte i klase potrebno mapirati u šemu relacione baze podataka.
- Tipični problemi:
  - **Granularnost** – broj klasa ne odgovara broju tabela u bazi
  - **Nasleđivanje** – nije podržano u relacionom modleu
  - **Identitet** – jednakost primarnih ključeva vs. identičnost i jednakost objekata
  - **Asocijacije** – veze između klasa vs. strani ključevi
  - **Navigacija podataka** – veze između klasa vs. strani ključevi



# O/R maperi

- O/R maper predstavlja implementaciju Data Mapper projektnog obrasca.
- Object-Relational mapper (ORM, O/RM, O/R mapper) predstavlja tehniku koja omogućava konverziju podataka između nekompatibilnih sistema OO aplikacija i relacionih baza podataka.
- O/R maperi obezbeđuju **perzistentnost** objekata odnosno omogućavaju da objekti postoje nezavisno od same aplikacije.



# O/R maperi

- Prednosti korišćenja O/R mapera:
  - Skalabilnost
  - Skraćeno vreme razvoja
  - Domain driven design
  - Nezavisnost od RDBMS-a
- Nedostaci O/R maper tehnologije
  - Performanse
  - Problemi prilikom mapiranja između OO modela i relacionog modela

# O/R maperi

- **Unit of Work**
- Projektni obrazac koji omogućava O/R maperima:
  - Vođenje evidencije o svim objektima koji su izmenjeni tokom transakcije
  - Kordinaciju upisa izmena u bazu podataka
  - Rešavanje problema konkurencije
- Nije pogodan pristup kod koga se svaka izmena nad objektima u memoriji prenosi u bazu podataka. Izmene se prosleđuju bazi podataka tek po završetku transakcije.

# O/R maperi

- **Identity map**
- Projektni obrazac koji omogućava O/R maperima:
  - svaki objekat može u memoriju da se učita samo jednom
  - sve objekte učitane u memoriju čuva u mapi (dictionary)
  - prilikom referenciranja objekat akoristi se njihova instanca koja je učitana u mapu
- Identity map je validna tokom trajanja Unity of Work.

# O/R maperi

- **Lazy load**
- Projektni obrazac koji omogućava O/R maperima:
  - Prilikom učitavanja objekta ne učitavaju se svi podaci koje objekat sadrži
  - Inicijalno su učitani samo neophodni podaci
  - Ostali podaci se učitavaju na zahtev odnosno u trenutku kada su potrebni
  - Objekat zna kako da učita nedostajuće podatke
- Kada se objekat učitava u memoriju korisno je učitati i povezane objekte. Time se znatno olakšava navigacija podataka. Takva strategija može da dovede do nepotrebnog učitavanja velikih količina podataka. Zbog toga je korišćenje lazy load tehnika od ključne važnosti.

# O/R maperi

- **NHibernate**
- ADO.NET Entity Framework (.NET 3.5 SP1)
- LINQ to SQL (.NET 3.5)
- Castle ActiveRecord
- MyBatis
- LLBLGenPro
- ...

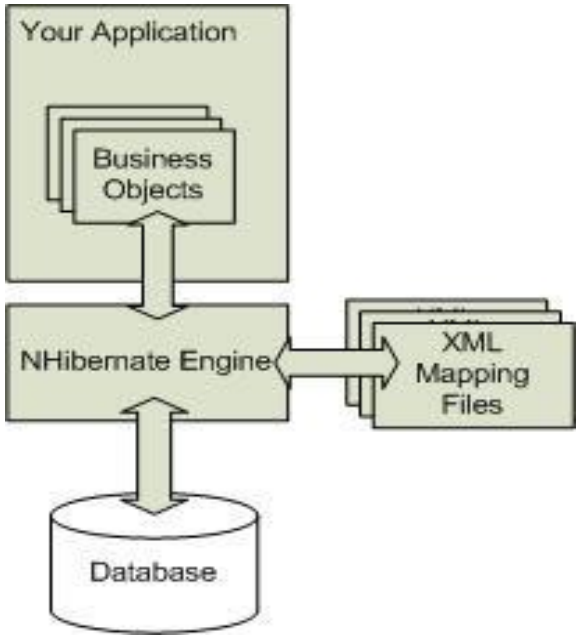
# NHibernate

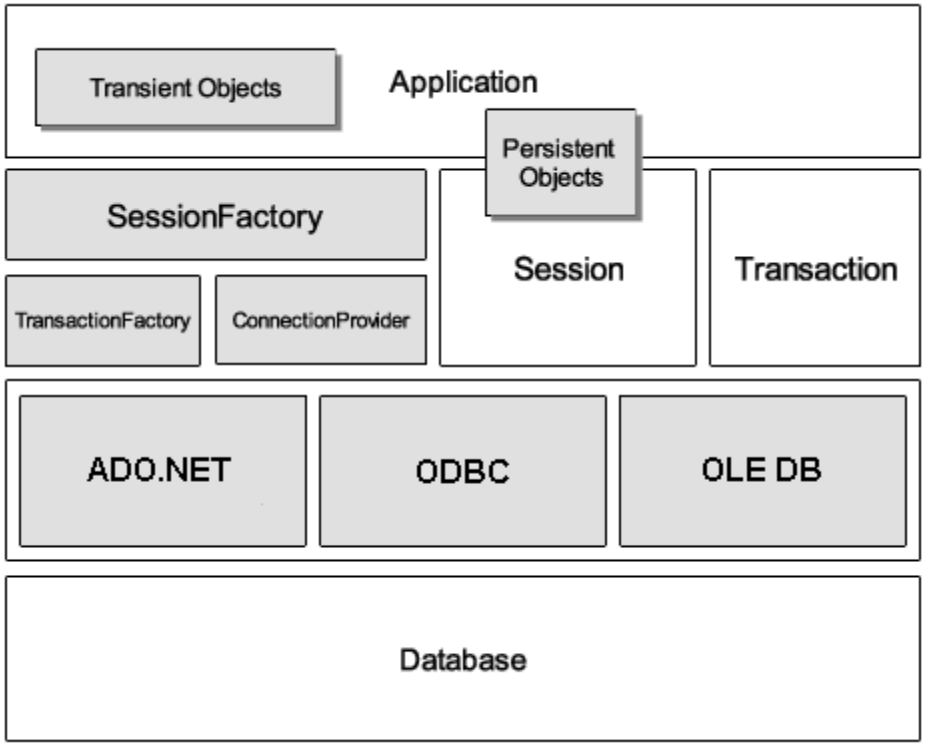
- Hibernate je inicijalno razvijen za JAVA programski jezik
  - kreiran 2001, Gavin King
  - Trenutno pod kontrolom JBossGroup / Red Hat
- Nhibernate - port za .NET 1.1, 2.0, 3.5, 4.0, 4.5
- Podrška za većinu popularnih RDBMS sistema:
  - Oracle, SQL Server, DB2, SQLite, PostgreSQL, MySQL, Sybase, Firebird, …
- XML konfiguracione datoteke
- Podrška od strane zajednice (puno nezvisnih projekata)
- Free/open source -NHibernate podleže LGPL (Lesser GNU Public License)
- Tekuća verzija je 4.0.3

# NHibernate

- Glavna funkcionalnost koju NHibernate obezbeđuje je mapiranje između .NET klasa i odgovarajućih tabela u relacionoj bazi podataka.
- Osim toga NHibernate obezbeđuje mehanizme za pretraživanje i učitavanje podataka.
- NHibernate omogućava programerima da razvijaju poslovne objekte u obliku .NET klasa. Svaki poslovni objekat je predstavljen jednom POCO (plain old C# object) .NET klasom.
- POCO .NET klasa je standardna klasa koja (osim generičke Object klase) ne nasleđuje bilo koju specijalnu klasu.

# NHibernate


Your Application
Business Objects
NHibernate Engine
XML Mapping Files
Database

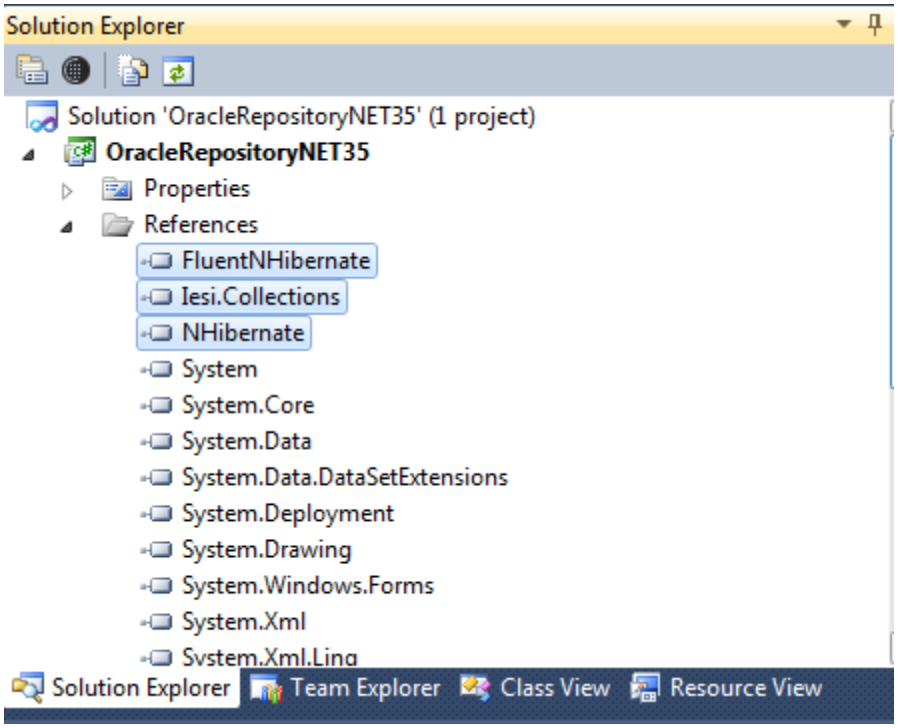

Transient Objects
Application
Persistent Objects
SessionFactory
Session
Transaction
TransactionFactory
ConnectionProvider
ADO.NET
ODBC
OLE DB
Database

# NHibernate

- NHibernate koristi XML mapping datoteke za odgovarajuće .NET klase kako bi obezbedio podršku za sve CRUD operacije.
- NHibernate učitava XML mapping datoteke tokom izvršavanja kako bi odredio kako su .NET klase povezane sa tabelama u relacionoj bazi podataka ali i kako su povezane međusobno.
- NHibernate trenutno podržava veći broj DBMS-ova koji su dostupni na tržištu i generalno podržava bilo koji DBMS za koji je implementiran odgovarajući OLE DB drajver.
- NHibernate automatski generiše sve neophodne SQL komande koje se izvršavaju u pozadini.

# NHibernate

- Dodavanje referenci na na neophodne biblioteke



# NHibernate

## NHibernate XML mapiranja

**SEKTOR**

| IMEO | BROJOD | MATBRR | DATPOST |
|---|---|---|---|
| RAZVOJ | 5 | 333445555 | 22-JUN-78 |
| ADMINISTRACIJA | 4 | 987654321 | 01-JAN-85 |
| RUKOVODSTVO | 1 | 888665555 | 19-JUN-71 |

```xml
<hibernate-mapping xmlns="urn:nhibernate-mapping-2.2">
  <class name="DataLayer.Sektor, DataLayer" table="SEKTOR">
    <id name="BrojOd" column ="BROJOD"  type="Int32">
      <generator class="assigned" />
    </id>
    <property name="ImeO" column="IMEO" type="String" length="15"/>
    <property name="MatBrR" column="MATBRR" type="Int32" />
    <property name="DatPost" column="DATPOST" type="DateTime" />
  </class>
</hibernate-mapping>
```

```csharp
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace DataLayer
{
    public class Sektor
    {
        public Sektor()
        {
        }

        public virtual int BrojOd {get; set;}
        public virtual string ImeO { get; set; }
        public virtual int MatBrR { get; set; }
        public virtual DateTime DatPost { get; set; }

    }
}
```

# NHibernate

- Fluent NHibernate

```
public class RADNIK
{
    public virtual int Id { get; protected set; }
    public virtual int MBR { get; set; }
    public virtual string LIME { get; set; }
    public virtual char SSLOVO { get; set; }
    public virtual string PREZIME { get; set; }
    public virtual DateTime DATUM_RODJENJA { get; set; }
    public virtual string STRUCNA_SPREMA { get; set; }
    public virtual int SEF_FLAG { get; set; }

    public virtual IList<PRODAVNICA> Prodavnice { get; set; }

    public RADNIK()
    {
        Prodavnice = new List<PRODAVNICA>();
    }
}
```

# NHibernate

- Fluent NHibernate

```
public RADNIKMAPIRANJE()
{
    //mapiranje primarnog kljuca
    Id(x => x.Id, "JBR").GeneratedBy.TriggerIdentity();

    //mapiranje svojstava
    Map(x => x.MBR);
    Map(x => x.LIME);
    Map(x => x.SSLOVO);
    Map(x => x.PREZIME);
    Map(x => x.STRUCNA_SPREMA);
    Map(x => x.SEF_FLAG);


    //mapiranje many-to-many veze
    HasManyToMany(x => x.Prodavnice)
        .Table("RADI_U").ParentKeyColumn("JBR_RADNIK").ChildKeyColumn("BROJP").Cascade.All();
}
```

# NHibernate

- Kreiranje sesije

```
private void button1_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();

        //Obrada podataka

        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# NHibernate

- Učitavanje objekta

```
private void button3_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();

        Sektor sek;
        sek = s.Load<Sektor>(5);

        //sek = (Sektor)s.Load(typeof(Sektor), 5);

        Sektor sek1 = new Sektor();
        s.Load(sek1, 4);


        MessageBox.Show("Učitani sektor: " + sek.BrojOd + " " + sek.ImeO);

        MessageBox.Show("Učitani sektor: " + sek1.BrojOd + " " + sek1.ImeO);

        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# NHibernate

- Kreiranje objekta

```
private void button2_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();

        Sektor sek = new Sektor();

        sek.BrojOd = 10;
        sek.ImeO = "Test";

        s.Save(sek);

        s.Flush();

        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# NHibernate

- Pretraživanje

```
private void button5_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();

        IQuery q = s.CreateQuery("FROM Sektor AS s ORDER BY s.BrojOd");


        IList<Sektor> sektori = q.List<Sektor>();

        foreach (Sektor sek in sektori)
        {
            MessageBox.Show("Sektor: " + sek.BrojOd + " " + sek.ImeO);
        }

        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# NHibernate

- Pretraživanje

```
private void button6_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();

        IQuery q = s.CreateQuery("FROM Sektor AS s WHERE s.ImeO LIKE ? ORDER BY s.BrojOd");
        q.SetString(0, "%A%");

        IList<Sektor> sektori = q.List<Sektor>();

        foreach (Sektor sek in sektori)
        {
            MessageBox.Show("Sektor: " + sek.BrojOd + " " + sek.ImeO);
        }

        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# NHibernate

- Ažu

```
private void button4_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();
        IQuery q = s.CreateQuery("FROM Sektor AS s WHERE s.ImeO LIKE ?");

        q.SetString(0, "T%");

        IList<Sektor> sektori = q.List<Sektor>();

        foreach (Sektor sek in sektori)
        {
            if (sek.ImeO == "Test")
            {
                sek.MatBrR = 987654321;
                s.Save(sek);
            }
            else
                MessageBox.Show("Sektor: " + sek.BrojOd + " " + sek.ImeO);

        }

        s.Flush();
        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

# NHibernate

- Brisanje podataka

```
private void button7_Click(object sender, EventArgs e)
{
    try
    {
        ISession s = _factory.OpenSession();

        Sektor sek;
        sek = s.Load<Sektor>(10);

        s.Delete(sek);

        s.Flush();
        s.Close();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }

}
```