



Računarstvo i informatika

Katedra za računarstvo

Elektronski fakultet u Nišu

Sistemi baza podataka

Fluent NHibernate

Letnji semestar 2015



Fluent NHibernate

- Fluent NHibernate je projekat koji predstavlja alternativu kreiranju standardnih NHibernate xml datoteka za mapiranje.
- Fluent NHibernate omogućava kreiranje NHibernate mapiranja korišćenjem C# koda sa jako definisanim tipovima podataka.
- Ovakav pristup obezbeđuje kreiranje lako čitljivih i konciznih mapiranja.
- Fluent NHibernate nije sastavni deo NHibernate projekta već redstavlja nezavistan projekat.
- Homepage: <http://www.fluentnhibernate.org/>
- Tekuća verzija: 2.0.1



Fluent NHibernate

- Svaka nova verzija Fluent NHibernate biblioteke je dostupna kao NuGet paket.
- NuGet predstavlja menadžer paketa za Microsoft Visual Studio razvojno okruženje.
- Homepage: <http://www.nuget.org/>
- NuGet instalacija: <http://docs.nuget.org/consume/installing-nuget>
- Nakon instalacije NuGet Package Manager Console je dostupna u Visual Studio razvojnom okruženju korišćenjem stavke menija:

Tools / NuGet Package Manager / Package Manager Console



Fluent NHibernate

- Fluent NHibernate NuGet paket:
<http://www.nuget.org/packages/FluentNHibernate>
- Instalacija korišćenjem NuGet Package Manager Console:

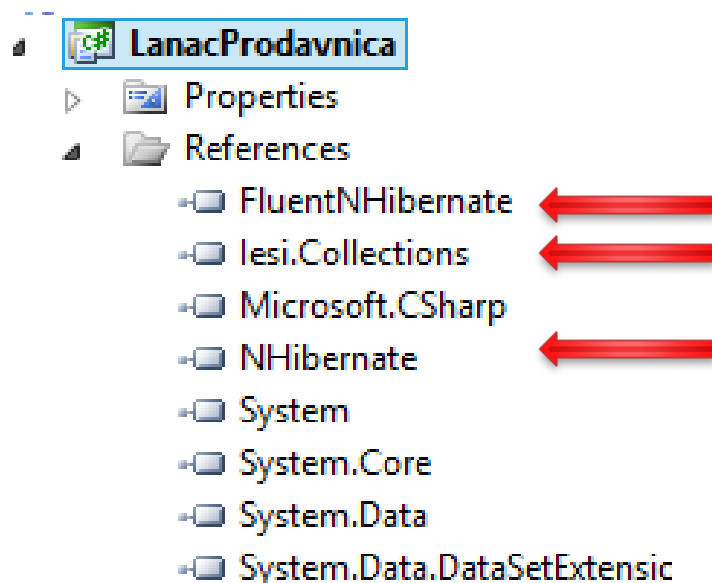
PM> Install-Package FluentNHibernate

```
PM> Install-Package FluentNHibernate
Attempting to resolve dependency 'NHibernate (≥ 4.0)'.
Attempting to resolve dependency 'Iesi.Collections (≥ 4.0 && < 5.0)'.
Installing 'Iesi.Collections 4.0.0.4000'.
Successfully installed 'Iesi.Collections 4.0.0.4000'.
Installing 'NHibernate 4.0.0.4000'.
Successfully installed 'NHibernate 4.0.0.4000'.
Installing 'FluentNHibernate 2.0.1.0'.
Successfully installed 'FluentNHibernate 2.0.1.0'.
Adding 'Iesi.Collections 4.0.0.4000' to LanacProdavnica.
Successfully added 'Iesi.Collections 4.0.0.4000' to LanacProdavnica.
Adding 'NHibernate 4.0.0.4000' to LanacProdavnica.
Successfully added 'NHibernate 4.0.0.4000' to LanacProdavnica.
Adding 'FluentNHibernate 2.0.1.0' to LanacProdavnica.
Successfully added 'FluentNHibernate 2.0.1.0' to LanacProdavnica.
```



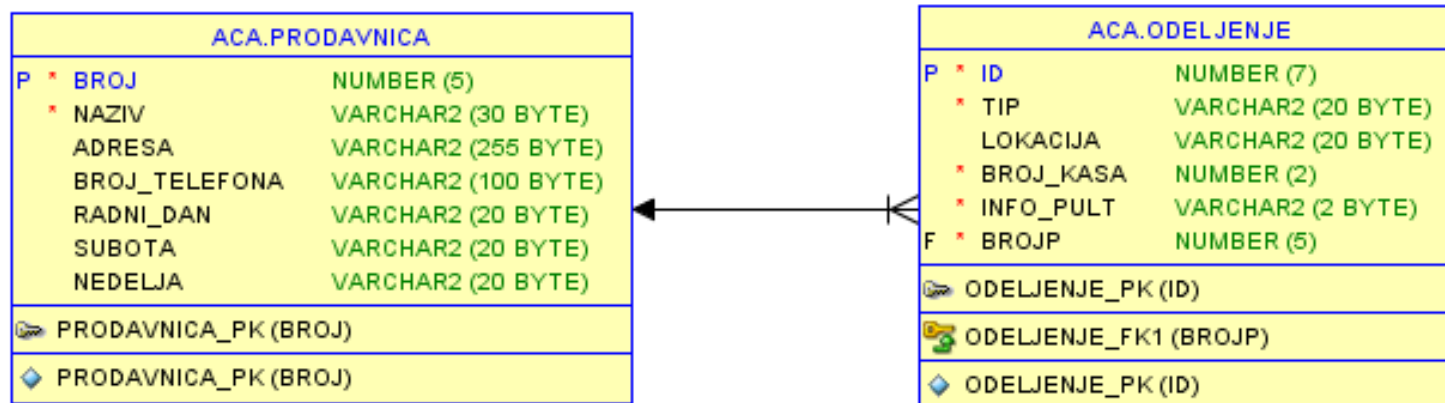
Fluent NHibernate

- Instalacija korišćenjem NuGet menadžera paketa rešava problem međuzavisnosti paketa, pa zajedno sa Fluent Nhibernate bibliotekom instalira i neophodne biblioteke **NHibernate** i **Iesi.Collections**.
- Sve instalirane biblioteke NuGet uključuje kao reference u tekući Visual Studio projekat.



Fluent NHibernate

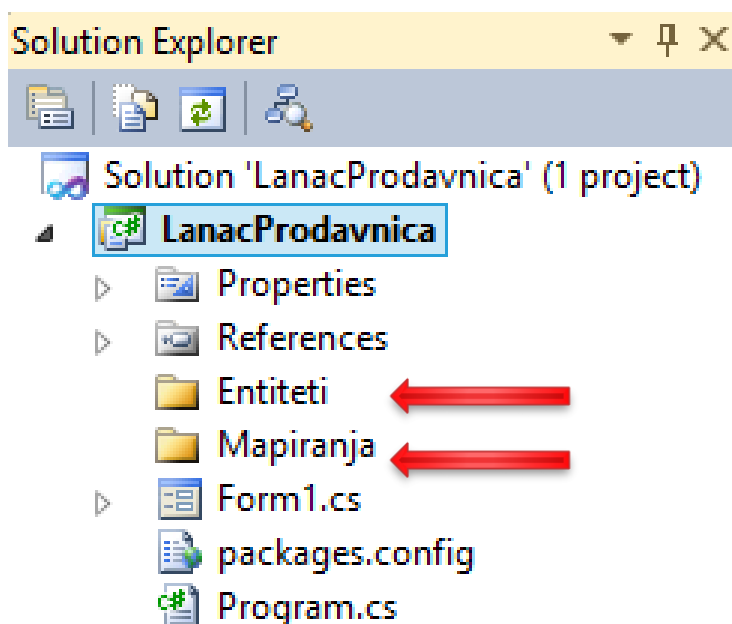
- **Primer: Deo baze podataka Lanac prodavnica igračaka**





Fluent NHibernate

- Struktura projekta:
 - Direktorijum Entiteti sadrže domenske entitete (klase)
 - Direktorijum Mapiranja sadrži klase koje domenske entitete mapiraju na relacionu bazu podataka



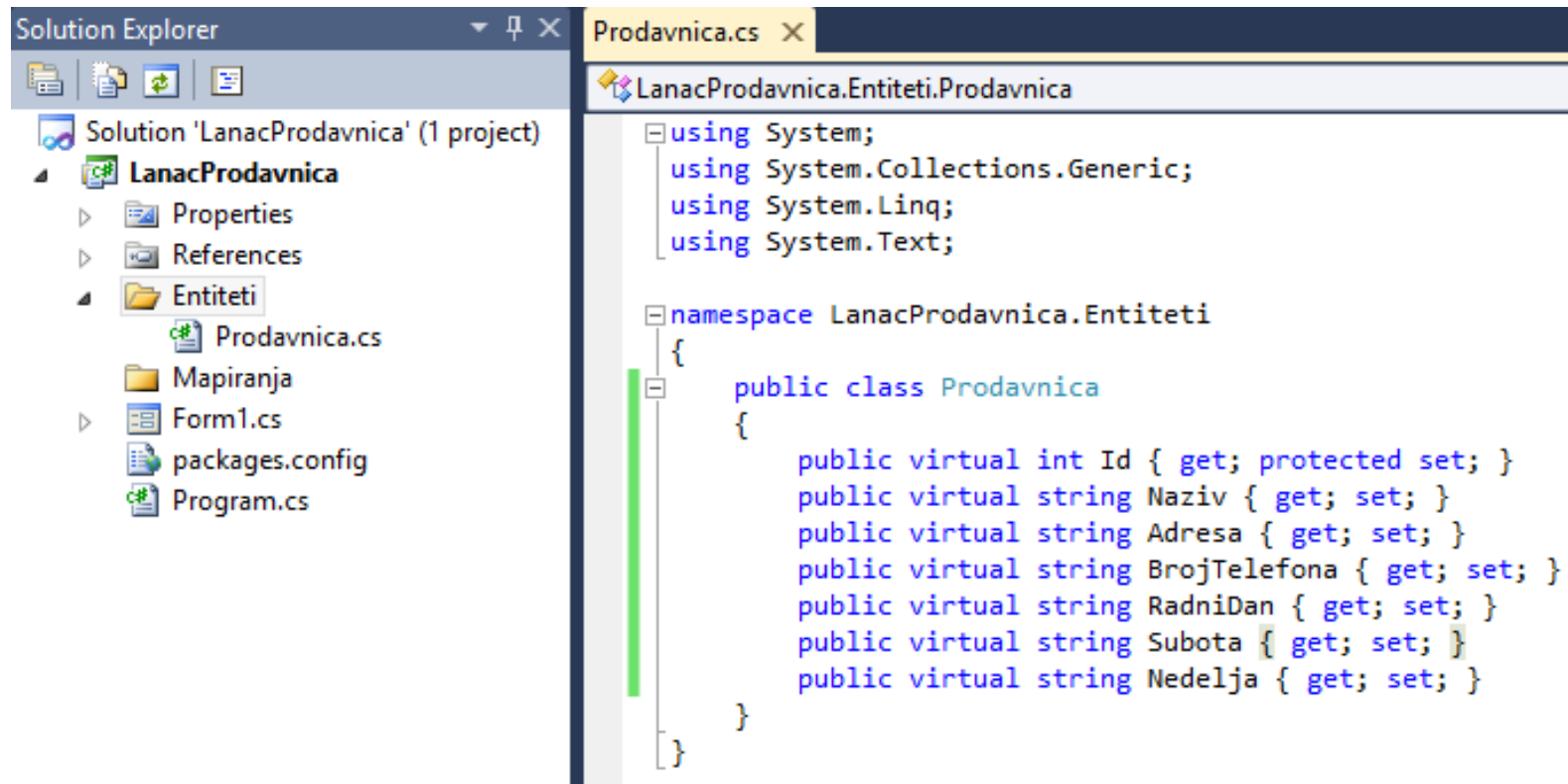


Fluent NHibernate

- Entitet PRODAVNICA
 - Domenska klasa Prodavnica.cs
 - Nalazi se u direktorijumu Entiteti. Ovde će biti kreirane sve domenske klase.
 - Klasa mora da bude deklarirana kao **public** u okviru svog namespace-a
 - Sva svojstva klase koja će biti mapirani na kolone u tabeli relacije baze podataka moraju biti deklarirana kao **public**
 - Sva svojstva klase koja će biti mapirana na kolone u tabeli relacije baze podataka moraju biti deklarirana kao **virtual**. NHibernate predefiniše sva svojstva kako bi obezbedio proxy za **lazy load** učitavanje podataka.



Fluent NHibernate



The screenshot shows the Visual Studio IDE with the Solution Explorer on the left and the code editor on the right. The Solution Explorer displays the project 'LanacProdavnica' with the following structure:

- Solution 'LanacProdavnica' (1 project)
 - LanacProdavnica
 - Properties
 - References
 - Entiteti
 - Prodavnica.cs
 - Mapiranje
 - Form1.cs
 - packages.config
 - Program.cs

The code editor shows the file 'Prodavnica.cs' with the following code:

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

namespace LanacProdavnica.Entiteti
{
    public class Prodavnica
    {
        public virtual int Id { get; protected set; }
        public virtual string Naziv { get; set; }
        public virtual string Adresa { get; set; }
        public virtual string BrojTelefona { get; set; }
        public virtual string RadniDan { get; set; }
        public virtual string Subota { get; set; }
        public virtual string Nedelja { get; set; }
    }
}
```



Fluent NHibernate

- Mapiranje PRODAVNICA
 - Klasa ProdavnicaMapiranje.cs
 - Nalazi se u direktorijumu Mapiranja. Ovde će biti kreirane sve klase koje mapiraju domenske entitete na relacionu bazu podataka.
 - Klasa za mapiranje nasleđuje klasu ClassMap<T> pri čemu T predstavlja domensku klasu koja se mapira. U konkretnom slučaju klasa ProdavnicaMapiranje će naslediti klasu **ClassMap<Prodavnica>**
 - Sva mapiranja se navode u konstruktoru klase za mapiranje



Fluent NHibernate

The screenshot shows a Visual Studio IDE with a solution named 'LanacProdavnica'. The Solution Explorer on the left shows the project structure, including folders for Properties, References, Entiteti, and Mapiranja, and files for Form1.cs, packages.config, and Program.cs. The Mapiranja folder is expanded, showing the file ProdavnicaMapiranja.cs. The code editor on the right shows the implementation of the ProdavnicaMapiranja class, which inherits from ClassMap<Prodavnica>. The class contains several configuration methods for the database mapping, including Table, Id, Map, and GeneratedBy.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using LanacProdavnica.Entiteti;
using FluentNHibernate.Mapping;

namespace LanacProdavnica.Mapiranja
{
    class ProdavnicaMapiranja : ClassMap<Prodavnica>
    {
        public ProdavnicaMapiranja()
        {
            //Mapiranje tabele
            Table("PRODAVNICA");

            //mapiranje primarnog ključa
            Id(x => x.Id, "BROJ").GeneratedBy.TriggerIdentity();

            //mapiranje svojstava
            Map(x => x.Naziv, "NAZIV");
            Map(x => x.Adresa, "ADRESA");
            Map(x => x.BrojTelefona, "BROJ_TELEFONA");
            Map(x => x.RadniDan, "RADNI_DAN");
            Map(x => x.Subota, "SUBOTA");
            Map(x => x.Nedelja, "NEDELJA");
        }
    }
}
```



Fluent NHibernate

- NHibernate aplikacija svoj rad bazira na korišćenju dva ključna interfejsa:
 - **ISession**
 - Primarni interfejs u NHibernate aplikacijama
 - Interfejs za rad sa NHibernate sesijama
 - Obezbeđuje metode za osnovne CRUD operacije
 - Persistence manager pošto obezbeđuje mehanizme za perzistenciju objekata
 - Jednostavno kreiranje i uništavanje sesije (ne zahteva puno resursa)
 - Nije thread safe. NHibernate sesija se koristi samo u okviru jedne niti
 - Odgovara konekciju kod ADO.NET biblioteke



Fluent NHibernate

- **ISessionFactory**

- Obezbeđuje metode za kreiranje NHibernate sesija
- SessionFactory je jako zahtevan sa stanovišta resursa koje zahteva
- Projektovan je tako da se kreira jedna instanca u aplikaciji koaj se deli između veće broja niti.
- Ukoliko aplikacija pristupa različitim bazama podataka, za svaku bazu podataka se kreira po jedna SessionFactory instanca
- Kešira meta podatke i SQL upite koje Nhibernate generiše tokom rada
- Obezbeđuje korišćenje second-level keširanja podataka. Podaci koji su pribavljeni u nekoj Unit of Work sesiji se kasnije mogu koristiti u drugim Unit of Work sesijama. Potrebno je eksplicitno konfigurisati podršku za ovu vrstu keša.



Fluent NHibernate

- Konfiguracija NHibernate mehanizma u aplikaciji je implementirana u klasi `DataLayer` u vidu dve static metode:
 - `public static ISessionFactory CreateSessionFactory()`
 - Konfiguracija baze podataka
 - Kreiranje i konfigurisanje `SessionFactory` objekta
 - `public static ISession GetSession()`
 - Implementacija singleton pattern-a koja obezbeđuje samo jednu instancu `SessionFactory` objekta u aplikaciji
 - Otvaranje NHibernate sesija



Fluent NHibernate

```
//funkcija na zahtev otvara sesiju
public static ISession GetSession()
{
    //ukoliko session factory nije kreiran
    if (_factory == null)
    {
        lock (objLock)
        {
            if (_factory == null)
                _factory = CreateSessionFactory();
        }
    }

    return _factory.OpenSession();
}
```

- Thread safe implementacija singleton pattern-a
- CreateSessionFactory metoda se poziva samo ukoliko prethodno objekat SessionFactory nije kreiran
- Metoda kreira i otvara NHibernate sesiju



Fluent NHibernate

```
//konfiguracija i kreiranje session factory
private static ISessionFactory CreateSessionFactory()
{
    try
    {
        var cfg = OracleClientConfiguration.Oracle10
            .ConnectionString(c =>
                c.Is("Data Source=160.99.9.199:1521/gislab.elfak.ni.ac.rs;User Id=aca;Password=aca"));

        return Fluently.Configure()
            .Database(cfg)
            .Mappings(m => m.FluentMappings.AddFromAssemblyOf<ProdavnicaMapiranja>())
            .BuildSessionFactory();
    }
    catch (Exception ec)
    {
        System.Windows.Forms.MessageBox.Show(ec.Message);
        return null;
    }
}
```



Fluent NHibernate

- Konfiguracija baze podataka:
 - U konkretnom slučaju za Oracle DBMS specificira se Oracle dialect i Oracle connection string
 - Primeri konfiguracija: <https://github.com/jagregory/fluent-nhibernate/wiki/Database-configuration>
- Kreiranje i konfiguracija SessionFactory objekta:
 - Definiše se odgovarajuća konfiguracija baze podataka
 - Specificiraju se mapiranja koja će se koristiti (U konkretnom slučaju specificiraju se sva mapiranja koja se nalaze u istom assembly-iju kao i klasa ProdavnicaMapiranja)
 - Specifikacije mapiranja: <https://github.com/jagregory/fluent-nhibernate/wiki/Fluent-configuration>
- **Napomena: Za slučaj Oracle DBMS-a, na računaru na kome će se aplikacija koristiti neophodno je instalirati odgovarajuću verziju Oracle klijenata.**



Fluent NHibernate

- Primer jednostavne operacije učitavanja podatka:

```
try
{
    ISession s = DataLayer.GetSession();

    //Ucitavaju se podaci o prodavnici za zadatim brojem
    Prodavnica.Entiteti.Prodavnica p = s.Load<Prodavnica.Entiteti.Prodavnica>(61);

    MessageBox.Show(p.Naziv);

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



Mapiranja

- Klasa `ClassMap<T>` predstavlja osnovnu klasu za sva `FluentNHibernate` mapiranja.

```
public class ProdavnicaMapiranja : ClassMap<Prodavnica>
{
    public ProdavnicaMapiranja()
    {
    }
}
```

- Sva mapiranja se definišu u okviru konstruktora klase za mapiranje.
- Sva mapiranja imaju oblik lambda izraza: `x => x.Property`



Mapiranja

- **Id** – metoda koja se koristi za mapiranje atributa klase koji obezbeđuje jednoznačnu identifikaciju domenskih entiteta.
- Ovi atributi klase se mapiraju na primarni ključ u tabel relacione baze podataka.

```
Id(x => x.Id)  
  .Column("BROJ")  
  .GeneratedBy.TriggerIdentity();
```

- U konkretnom primeru atribut klase Id je mapiran na kolonu BROJ u tabeli. Za definisanje vrednosti primarnog ključa se koristi trigger baze podataka.



Mapiranja

- Neki od tipičnih opcija za generator primarnog ključa:
 - **Assigned** – Aplikacija definiše vrednost primarnog ključa pre snimanja objekta u bazu podataka
 - **Increment** – NHibernate generiše vrednosti za primarni ključ inkrementiranjem poslednje upotrebljene vrednosti. Koristi se samo u situacijama kada jedna aplikacija upisuje podatke u tabelu baze podataka.
 - **Identity** – podrška za Identity kolone kod DB2, MySQL, MS SQL Server i Sybase baza podataka.
 - **Sequence** – podrška za generisanje vrednosti primarnih ključeva korišćenjem sekvenci kod DB2, PostgreSQL i Oracle baza podataka. Prilikom upisivanja objekta u bazu podataka izvršavaju se dve SQL naredbe: jedna koja pribavlja primarni ključ i druga INSERT naredba.
 - **TriggerIdentity** – baza podataka sama generiše vrednost primarnog ključa korišćenjem BEFORE INSERT trigera
 - **Guid** – koristi se System.Guid za generisanje vrednosti primarnog ključa
 - **GuidNative** – koristi se GUID generator baze podataka za generisanje vrednosti primarnog ključa. Kao i kod sekvenci neophodne su dve SQL naredbe.



Mapiranja

- **Map** - metoda koja definiše mapiranja između atributa domenske klase i kolona relacione baze podataka.

```
Map(x => x.NazivProdavnice);
```

- U primeru je dati najjednostavniji oblik poziva Map metode. Podrazumeva da atribut domenske klase i kolona u tabeli imaju isto ime.

```
Map(x => x.NazivProdavnice)  
    .Column("NAZIV");
```

- U prethodnom primeru ime atributa klase i ime kolone u tabeli se razlikuju pa je to neophodno eksplicitno specificirati.



Mapiranje

- U većini slučajeva neophodno je obezbediti mapiranje veza između entiteta.
- Fluent Nhibernate podržava mapiranje sledećih tipova relacija:
 - many-to-one
 - one-to-many
 - many-to-many
 - one-to-one



Mapiranja

- **References** - metoda koja omogućava kreiranju many-to-one relacija između entiteta i primenjuje se na many strani.
- Kreira referencu na jedan entitet.
- U terminologiji baza podataka: tabela koja poseduje strani ključ koji referencira primarni ključ u nekoj drugoj tabeli.
- Za slučaj klasa Prodavnica i Odeljenje:

```
class Prodavnica
{
}

class Odeljenje
{
    public virtual Prodavnica PripadaProdavnici{get; set;}
}
```



Mapiranja

- U klasi za mapiranje OdeljenjeMapiranja može da se doda mapiranje oblika:

```
References(x => x.PripadaProdavnicima);
```

- U ovom slučaju se podrazumeva da u tabeli ODELJENJE postoji strani ključ Prodavnica_id.
- Ukoliko je neophodno eksplicitno specificirati strani ključ koristi se sledeća notacija:

```
References(x => x.PripadaProdavnicima)  
    .Column("BROJP")  
    .LazyLoad();
```

- U prethodnom primery eksplicitno je specificirano i LazyLoad svojstvo mada je ono podrazumevano ponašanje.



Mapiranja

- **HasMany** - metoda koja omogućava kreiranje one-to-many relacija između domenskih entiteta.
- Nalazi se na one strani many-to-one relacije.
- Odgovara parent-child relaciji.
- Najčešće korišćeni tip relacija između domenskih entiteta.
- Za slučaj klasa Prodavnica i Odeljenje:

```
class Prodavnica
{
    public virtual IList<Odeljenje> Odeljenja {get; set;}

    public Prodavnica()
    {
        Odeljenja = new List<Odeljenje>();
    }
}
```



Mapiranja

- U klasi `ProdavnicaMapiranja` može da se doda mapiranje oblika:

```
HasMany(x => x.Odeljenja);
```

- U ovom slučaju se podrazumeva da u tabeli `ODELJENJE` postoji strani ključ `Prodavnica_id`.
- Ukoliko je neophodno eksplicitno specificirati strani ključ koristi se sledeća notacija:

```
HasMany(x => x.Odeljenja)  
    .KeyColumn("BROJP");
```

- `LazyLoad` svojstvo je podrazumevano. Ukoliko treba da se isključi koristi se sledeća notacija.

```
HasMany(x => x.Odeljenja)  
    .KeyColumn("BROJP")  
    .Not.LazyLoad();
```



Mapiranje

- Primer korišćenja HasMany relacije:

```
try
{
    ISession s = DataLayer.GetSession();

    //Ucitavaju se podaci o prodavnici sa zadatim brojem
    Prodavnica.Entiteti.Prodavnica p = s.Load<Prodavnica.Entiteti.Prodavnica>(61);

    foreach (Odeljenje o in p.Odeljenja)
    {
        MessageBox.Show(o.Tip + " " + o.Lokacija);
    }

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



Mapiranje

- Primer povezivanje odeljenja sa prodavnicom. U ovom primeru nova prodavnica će biti snimljena u bazu podataka ali ne i odeljena koja su povezana sa njom.

```
ISession s = DataLayer.Session();
```

```
Entiteti.Prodavnica p = new Entiteti.Prodavnica() {Naziv = "Emi Shop",  
                                                    RadniDan = "08-20",  
                                                    Subota = "08-14",  
                                                    Nedelja = "Ne radi"};
```

```
Odeljenje o = new Odeljenje() { Tip = "D05", Lokacija = "Niš",  
                                 BrojKasa = 1, InfoPult = "Da"};
```

```
Odeljenje o1 = new Odeljenje() { Tip = "D05", Lokacija = "Niš",  
                                 BrojKasa = 1, InfoPult = "Da"};
```

```
p.Odeljenja.Add(o);  
p.Odeljenja.Add(o1);
```

```
s.Save(p);
```




Mapiranje

- Da bi se obezbedilo da se sva odeljenja iz kolekcije snime zajedno sa prodavnicom kojoj pripadaju potrebno je uključiti **Cascade** svojstvo HasMany veze.

```
HasMany(x => x.Odeljenja)  
    .KeyColumn("BROJP")  
    .Cascade.All();
```

- Svojstvo Cascade specificira da li će se akcije nad roditeljeme (vlasnikom kolekcije) preneti na decu (elemente kolekcije). Neke od mogućih vrednosti:
 - All – prenose se sve akcije
 - Delete – brisanje roditelja dovodi do brisanja dece
 - SaveUpdate – prenose se samo Save i Update akcije



Mapiranja

- Kod ovakvih slučajeva treba voditi računa o redosledu operacija prilikom snimanja objekata u bazu podataka.
- U ovom slučaju redosled operacija je sledeći:
 1. Snima se roditelj i definiše se vrednost njegovog primarnog ključa.
 2. Snimaju deca (članovi kolekcije) kod kojih se vrednost stranog ključa postavi na NULL.
 3. Odradi se dodatni UPDATE koji za decu definiše vrednost stranog ključa.
- Ovakav pristup može da dovede do narušavanja ograničenja baze podataka. To posebno važi za situaciju gde je za kolonu stranog ključa postavljeno ograničenje NOT NULL. U tom slučaju drugi korak obavezno dovodi do pojave greške.



Mapiranja

- U tim slučajevima se definiše **Inverse** svojstvo HasMany relacije.

```
HasMany(x => x.Odeljenja)
```

```
    .KeyColumn("BROJP")
```

```
    .Inverse()
```

```
    .Cascade.All();
```

- U ovom slučaju se naglašava da je druga strana one-to-many relacije u obavezi da održava relaciju. Na taj način NHibernate zna da najpre mora da snimi objekat roditelja pa tek nakon toga objekte decu.



Mapiranje

- Za slučaj uključenog Inverse svojstva redosled operacija je sledeći:

```
o.PripadaProdavnici = p;
```

```
o1.PripadaProdavnici = p;
```

```
p.Odeljenja.Add(o);
```

```
p.Odeljenja.Add(o1);
```

```
s.Save(p);
```



Mapiranja

- Ukoliko se ne koriste Cascade i Inverse svojstva HasMany relacije korisnik je u obavezi da sam obezbedi pravilan redosled snimanja objekata.

```
s.Save(p);
```

```
o.PripadaProdavnici = p;  
s.Save(o);
```

```
o1.PripadaProdavnici = p;  
s.Save(o1);
```

```
p.Odeljenja.Add(o);  
p.Odeljenja.Add(o1);
```



Mapiranja

- Kod NHibernate relacija za specificiranje kolekcija obavezno je korišćenje interfejsa:
 - `System.Collections.IEnumerable`
 - `System.Collections.ICollection`
 - `System.Collections.IList`
 - `System.Collections.IDictionary`
 - `System.Collections.Generic.IEnumerable<T>`
 - `System.Collections.Generic.ICollection<T>`
 - `System.Collections.Generic.IList<T>`
 - `System.Collections.Generic.IDictionary<K, V>`
 - `System.Collections.Generic.ISet<T>`
- Svaka kolekcija zahteva inicijalizaciju pre korišćenja.

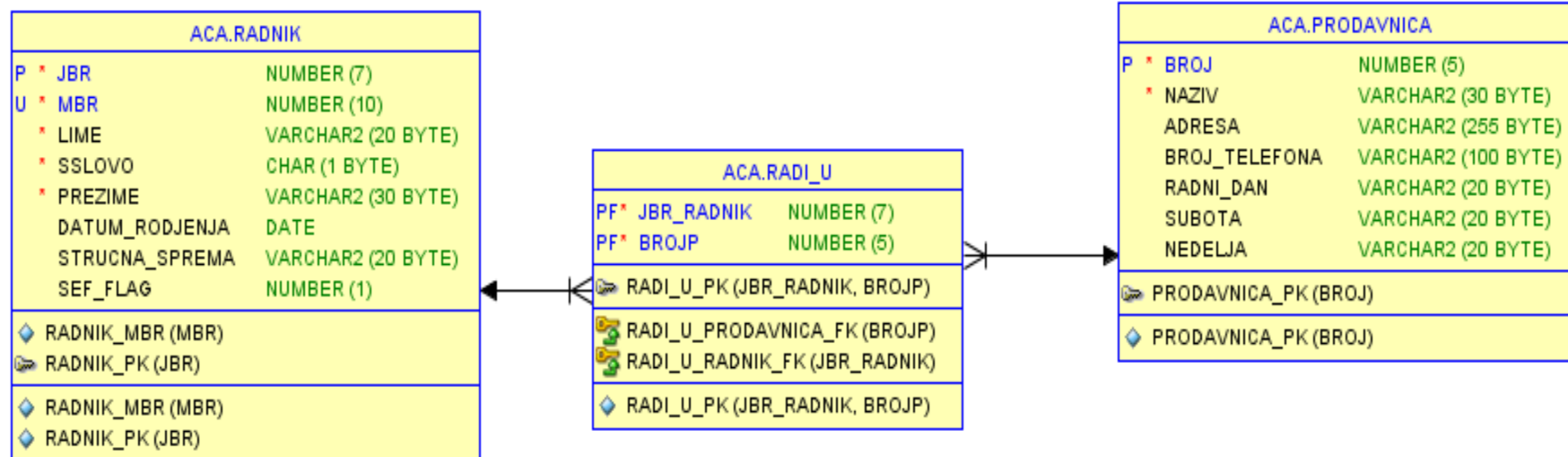


Mapiranja

- **HasManyToMany** – metoda koja omogućava mapiranje many-to-many veza između domenskih klasa.
- Funkcioniše na sličan način kao i HasMany metoda, pri čemu treba voditi računa da je odgovarajuća struktura u bazi podataka potpuno drugačija (strani ključ kod HasMany relacije u odnosu na dodatnu tabelu kod HasManyToMany relacije).

Mapiranja

- Primer deo baze podataka Lanac prodavnica igračka





Mapiranje

```
public class Radnik
{
    public virtual int Jbr { get; set; }
    public virtual int Mbr { get; set; }
    public virtual string Ime { get; set; }
    public virtual char SrednjeSlovo { get; set; }
    public virtual string Prezime { get; set; }
    public virtual DateTime DatumRodjenja { get; set; }
    public virtual string StrucnaSpema { get; set; }
    public virtual bool Sef { get; set; }

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

    public Radnik()
    {
        Prodavnice = new List<Prodavnica>();
    }
}
```



Mapiranje

```
public RadnikMapiranje()  
{  
    Table("RADNIK");  
  
    Id(x => x.Jbr).Column("JBR").GeneratedBy.TriggerIdentity();  
  
    Map(x => x.Mbr).Column("MBR");  
    Map(x => x.Ime).Column("LIME");  
    Map(x => x.SrednjeSlovo).Column("SSLOVO");  
    Map(x => x.Prezime).Column("PREZIME");  
    Map(x => x.DatumRodjenja).Column("DATUM_RODJENJA");  
    Map(x => x.StrucnaSpema).Column("STRUCNA_SPREMA");  
    Map(x => x.Sef).Column("SEF_FLAG");  
  
    HasManyToMany(x => x.Prodavnice)  
        .Table("RADI_U")  
        .ParentKeyColumn("JBR_RADNIK")  
        .ChildKeyColumn("BROJP");  
}
```



Mapiranja

`HasManyToMany(x => x.Prodavnice)`

- Prethodni oblik mapiranja podrazumeva postojanje tabele `ProdavnicaToRadnik` koja poseduje dve kolone `Radnik_id` i `Prodavnica_id`.
- Za eksplicitno specificiranje naziva tabele i odgovarajućih kolona koristi se sledeći oblik:

`HasManyToMany(x => x.Prodavnice)`

```
.Table("RADI_U")  
.ParentKeyColumn("JBR_RADNIK")  
.ChildKeyColumn("BROJP");
```



Mapiranja

- Na sličan način HasManyToMany relacija može da se definiše i na drugoj strani:

```
public virtual IList<Radnik> Radnici { get; set; }
```

```
HasManyToMany(x => x.Radnici)  
    .Table("RADI_U")  
    .ParentKeyColumn("BROJP")  
    .ChildKeyColumn("JBR_RADNIK")  
    .Inverse()  
    .Cascade.All();
```



Mapiranje

```
ISession s = DataLayer.GetSession();

Radnik r1 = s.Load<Radnik>(81);

foreach (Entiteti.Prodavnica p1 in r1.Prodavnice)
{
    MessageBox.Show(p1.Naziv);
}

Entiteti.Prodavnica p2 = s.Load<Entiteti.Prodavnica>(61);

foreach (Radnik r2 in p2.Radnici)
{
    MessageBox.Show(r2.Ime + " " + r2.Prezime);
}

s.Close();
```



Mapiranje

```
try
{
    ISession s = DataLayer.Session();

    Entiteti.Prodavnica p = new Entiteti.Prodavnica()
    {
        Naziv = "Emi Shop XXII",
        RadniDan = "08-20",
        Subota = "08-14",
        Nedelja = "Ne radi"
    };

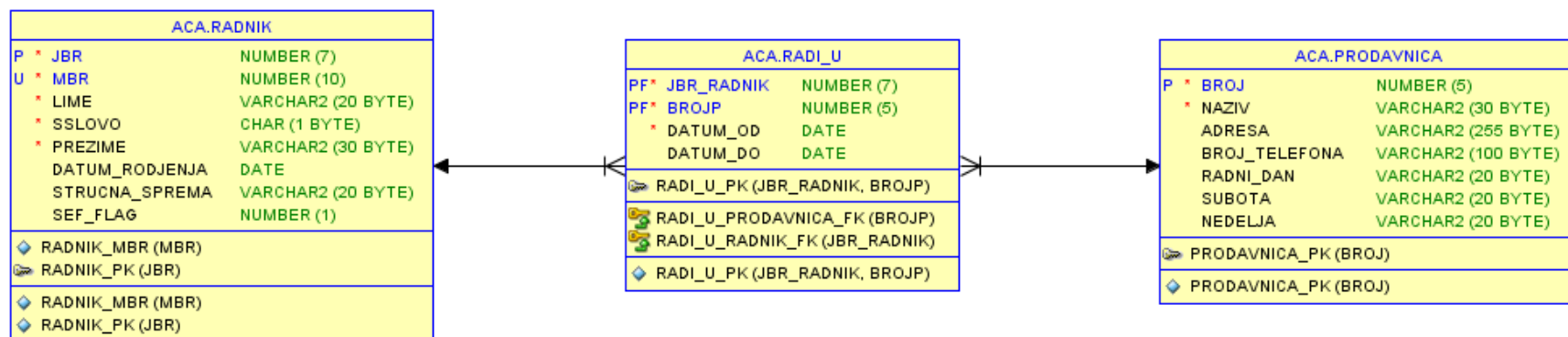
    Radnik r = new Radnik()
    {
        Ime = "Petar",
        SrednjeSlovo = 'P',
        Prezime = "Perić",
        Mbr = 123459,
        DatumRodjenja = new DateTime(1982, 11, 13),
        StrucnaSpema = "VII-1",
        Sef = false
    };

    r.Prodavnice.Add(p);
    p.Radnici.Add(r);
    s.Save(p);

    s.Flush();
    s.Close();
}
```


Mapiranja

- Komplikovaniji slučaj kod koga tabela **RADI_U** poseduje dodatne kolone.
- Neophodno je kreirati poseban domenski entitet **RadiU** koji ima kompozitni Id.
- Klase **Radnik** i **Prodavnica** će imati one-to-many vezu ka domenskom entitetu **RadiU**.





Mapiranja

- **Compositeld** – metoda koja omogućava mapiranje kompozitnih ključeva.
- Za potrebe mapiranja kompozitnog ključa potrebno je napraviti posebnu domensku klasu koja poseduje sve attribute kompozitnog ključa i definiše metode Equals i GetHashCode.
- Compositeld poseduje dva svojstva:
 - KeyProperty – atribut primarnog ključa predstavlja vrednost
 - KeyReference - atribut primarnog ključa predstavlja referencu na neki domenski entitet



Mapiranja

- Primer klase koja predstavlja kompozitni ključ

```
public class RadiUIId
{
    public virtual Radnik RadnikRadiU { get; set; }
    public virtual Prodavnica RadiUProdavnica { get; set; }

    public override bool Equals(object obj)
    {
        if (Object.ReferenceEquals(this, obj))
            return true;

        if (obj.GetType() != typeof(RadiUIId))
            return false;

        RadiUIId recievedObject = (RadiUIId)obj;

        if ((RadnikRadiU.Jbr == recievedObject.RadnikRadiU.Jbr) &&
            (RadiUProdavnica.Id == recievedObject.RadiUProdavnica.Id))
        {
            return true;
        }

        return false;
    }

    public override int GetHashCode()
    {
        return base.GetHashCode();
    }
}
```



Mapiranje

```
public class RadiU
{
    public virtual RadiUId Id {get; set;}
    public virtual DateTime DatumOd { get; set; }
    public virtual DateTime? DatumDo { get; set; }

    public RadiU()
    {
        Id = new RadiUId();
    }
}

class RadiUMapiranje : ClassMap<RadiU>
{
    public RadiUMapiranje()
    {
        Table("RADI_U");

        CompositeId(x => x.Id)
            .KeyReference(x => x.RadnikRadiU, "JBR_RADNIK")
            .KeyReference(x => x.RadiUProdavnica, "BROJP");

        Map(x => x.DatumOd).Column("DATUM_OD");
        Map(x => x.DatumDo).Column("DATUM_DO");
    }
}
```



Mapiranje

```
ISession s = DataLayer.GetSession();  
  
Radnik r = s.Load<Radnik>(109);  
Entiteti.Prodavnica p = s.Load<Entiteti.Prodavnica>(105);  
  
RadiU ru = new RadiU();  
ru.Id.RadnikRadiU = r;  
ru.Id.RadiUProdavnica = p;  
ru.DatumOd = DateTime.Now;  
  
s.Save(ru);  
  
s.Flush();  
s.Close();
```



Mapiranja

- Obično se dodaju i HasMany mapiranja u klasi Radnik i Prodavnica.
- Za slučaj klase Radnik:

```
public virtual IList<RadiU> RadiUProdavnice { get; set; }
```

```
HasMany(x => x.RadiUProdavnice)  
    .KeyColumn("JBR_RADNIK")  
    .Cascade.All()  
    .Inverse();
```



Mapiranja

- **HasOne** – metoda koja omogućava mapiranje specijalnog slučaja relacije između domenskih klasa.
- Umesto HasOne relacije najčešće se koristi References (many-to-one) relacija.



Mapiranja

```
public class CarMap : ClassMap<Car>
{
    public CarMap()
    {
        Table( "Vehicles.dbo.Car" );

        Id( x => x.CarId );
        Map( x => x.Name );
        Map( x => x.Year );

        HasOne( x => x.SteeringWheel ).PropertyRef( x => x.Car);
    }
}

public class SteeringWheelMap : ClassMap<SteeringWheel>
{
    public SteeringWheelMap()
    {
        Table( "Vehicles.dbo.SteeringWheel" );

        Id( x => x.SteeringWheelId );
        Map( x => x.Diameter );
        Map( x => x.Color );

        References( x => x.Car, "CarId" ).Unique();
    }
}
```

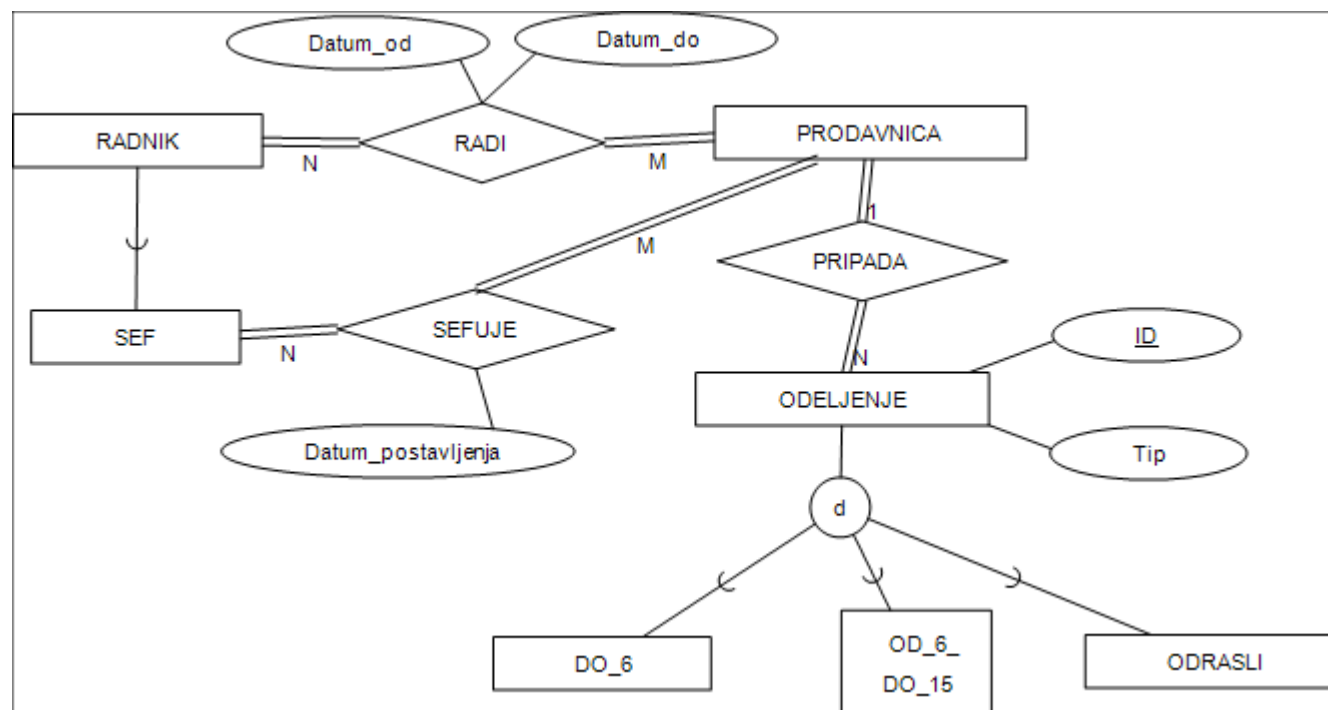


Mapiranje

- Prilikom mapiranja nasleđivanja FluentNHibernate podržava tri strategije:
 - Table-per-Class-Hierarchy (Table-per-Hierarchy)
 - Table-per-Class (Table-per-Type)
 - Table-per-Concrete-Class
- Primer: <http://www.codeproject.com/Articles/232034/Inheritance-mapping-strategies-in-Fluent-Nhibernat>

Mapiranja

- **Primer:** deo EER modela baze podataka Lanac prodavnica igračaka





Mapiranja

- **Primer:** Deo relacionog modela baze podataka Lanac prodavnica igračaka

ACA.ODELJENJE		
P	* ID	NUMBER (7)
	* TIP	VARCHAR2 (20 BYTE)
	LOKACIJA	VARCHAR2 (20 BYTE)
	* BROJ_KASA	NUMBER (2)
	* INFO_PULT	VARCHAR2 (2 BYTE)
	BROJP	NUMBER (7)
ODELJENJE_PK (ID)		
ODELJENJE_PK (ID)		

ACA.RADNIK		
P	* JBR	NUMBER (7)
U	* MBR	NUMBER (10)
	* LIME	VARCHAR2 (20 BYTE)
	* SSLOVO	CHAR (1 BYTE)
	* PREZIME	VARCHAR2 (30 BYTE)
	DATUM_RODZENJA	DATE
	STRUCNA_SPREMA	VARCHAR2 (20 BYTE)
	SEF_FLAG	NUMBER (1)
RADNIK_MBR (MBR)		
RADNIK_PK (JBR)		
RADNIK_MBR (MBR)		
RADNIK_PK (JBR)		



Mapiranja

- Za prevođenje entiteta ODELJENJE i njegovih podklasa iskorišćena je alternativa 8C i kompletna hijerarhija klasa je svedena na jednu tabelu – tabelu ODELJENJE.
- Za mapiranje ovakve strukture Fluent NHibernate koristi strategiju Table-per-Hierarchy.
- Nadklasa Odeljenje se mapira na uobičajeni način.
- Dodatak `DiscriminateSubClassesOnColumn` je metoda koja definiše kolonu (predikat) čija vrednost određuje pripadnost konkretnoj podklasi.
- Za slučaj tabele ODELJENJE, kolona TIP definiše o kom tipu odeljenja se radi odnosno kojoj podklasi odeljenje pripada.



Mapiranja

```
class OdeljenjeMapiranja : ClassMap<Odeljenje>
{
    public OdeljenjeMapiranja()
    {

        //Mapiranje tabele
        Table("ODELJENJE");

        //mapiranje podklasa
        DiscriminateSubClassesOnColumn("TIP"); ←

        //mapiranje primarnog kljuka
        Id(x => x.Id, "ID").GeneratedBy.TriggerIdentity();

        //mapiranje svojstava
        //Map(x => x.Tip, "TIP"); ←
        Map(x => x.Lokacija, "LOKACIJA");
        Map(x => x.BrojKasa, "BROJ_KASA");
        Map(x => x.InfoPult, "INFO_PULT");

        //mapiranje veze 1:N Prodavnica-Odeljenje
        References(x => x.PripadaProdavnici).Column("BROJP").LazyLoad();
    }
}
```




Mapiranja

- Za svaku podklasu se kreira posebna domenska klasa koja je izvedena iz osnovne klase.
- U konkretnom slučaju biće kreirane domenske klase: OdeljenjeDo5, OdeljenjeOd6Do15 i OdeljenjeOdrasli. Ove domenske klase su izvedene iz klase Odeljenje.
- Prilikom mapiranja podklasa, klasa za mapiranje se izvodi iz Fluent NHibernate klase **SubclassMap<T>**.
- Za svaku podklasu se metodom **DiscriminatorValue** definiše vrednost kolone (kolona je definisana u nadklasi) koja određuje da vrsta tabele pripada određenom tipu.
- U konkretnom slučaju vrednost kolone TIP određuje pripadnost podklasi na sledeći način:
 - DO5 - pripadaju tipu OdeljenjeDo5
 - OD6DO15 - pripadaju tipu OdeljenjeOd6Do15
 - ODRASLI - pripadaju tipu OdeljenjeOdrasli

Mapiranja

- Ukoliko podklase poseduju dodatne kolone, one se mapiraju na standardni način.
- Pošto svako odeljenje pripada jednom od tri navedena tipa, klasa Odeljenje može da se definiše kao apstraktna klasa.



```
public abstract class Odeljenje
{
    public virtual int Id { get; set; }
    public virtual string Tip { get; set; }
    public virtual string Lokacija { get; set; }
    public virtual int BrojKasa { get; set; }
    public virtual string InfoPult { get; set; }

    public virtual Prodavnica PripadaProdavnici { get; set; }
}
```



Mapiranja

```
public class OdeljenjeDo5 : Odeljenje
{
}

public class OdeljenjeOd6Do15 : Odeljenje
{
}

public class OdeljenjeOdrasli : Odeljenje
{
}
```

```
class OdeljenjeDo5Mapiranja : SubclassMap<OdeljenjeDo5>
{
    public OdeljenjeDo5Mapiranja()
    {
        DiscriminatorValue("D05");
    }
}

class OdeljenjeOd6Do15Mapiranja : SubclassMap<OdeljenjeOd6Do15>
{
    public OdeljenjeOd6Do15Mapiranja()
    {
        DiscriminatorValue("OD6D015");
    }
}

class OdeljenjeOdrasliMapiranja : SubclassMap<OdeljenjeOdrasli>
{
    public OdeljenjeOdrasliMapiranja()
    {
        DiscriminatorValue("ODRASLI");
    }
}
```



Mapiranja

```
ISession s = DataLayer.GetSession();

Entiteti.Prodavnica p = s.Load<Entiteti.Prodavnica>(61);

//kolona TIP automatski dobija vrednost D05
OdeljenjeDo5 o1 = new OdeljenjeDo5()
{
    Lokacija = "Niš",
    BrojKasa = 1,
    InfoPult = "Da",
    PripadaProdavnici = p
};

s.Save(o1);

s.Close();
```



Mapiranje

```
ISession s = DataLayer.GetSession();

IList<Odeljenje> odeljenja = s.QueryOver<Odeljenje>()
    .List<Odeljenje>();

foreach (Odeljenje o in odeljenja)
{
    if (o.GetType() == typeof(OdeljenjeDo5))
    {
        OdeljenjeDo5 o5 = (OdeljenjeDo5)o;
    }
    else if (o.GetType() == typeof(OdeljenjeOd6Do15))
    {
        OdeljenjeOd6Do15 o615 = (OdeljenjeOd6Do15)o;
    }
    else
    {
        OdeljenjeOdrasli oo = (OdeljenjeOdrasli)o;
    }
}

s.Close();
```



Mapiranje

- Primer: Podklasa Sef

```
public class Radnik
{
    public virtual int Jbr { get; set; }
    public virtual int Mbr { get; set; }
    public virtual string Ime { get; set; }
    public virtual char SrednjeSlovo { get; set; }
    public virtual string Prezime { get; set; }
    public virtual DateTime DatumRodjenja { get; set; }
    public virtual string StrucnaSpema { get; set; }
    public virtual bool Sef { get; protected set; }

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

    public virtual IList<RadiU> RadiUProdavnice { get; set; }

    public Radnik()
    {
        Prodavnice = new List<Prodavnica>();

        RadiUProdavnice = new List<RadiU>();
    }
}
```



Mapiranja

- Primer: Podklasa Sef

```
public RadnikMapiranja()
{
    Table("RADNIK");

    Id(x => x.Jbr).Column("JBR").GeneratedBy.TriggerIdentity();

    //mapiranje podklasa
    //podrazumevana vrednost je 0
    //svi radnici koji nisu sefovi ce imati vrednost 0 u koloni SEF_FLAG
    DiscriminateSubClassesOnColumn("SEF_FLAG", 0);

    Map(x => x.Mbr).Column("MBR");
    Map(x => x.Ime).Column("LIME");
    Map(x => x.SrednjeSlovo).Column("SSLOVO");
    Map(x => x.Prezime).Column("PREZIME");
    Map(x => x.DatumRodjenja).Column("DATUM_RODJENJA");
    Map(x => x.StrucnaSpema).Column("STRUCNA_SPREMA");
    Map(x => x.Sef).Column("SEF_FLAG");

    HasManyToMany(x => x.Prodavnice)
        .Table("RADI_U")
        .ParentKeyColumn("JBR_RADNIK")
        .ChildKeyColumn("BROJP")
        .Cascade.All();

    HasMany(x => x.RadiUProdavnice).KeyColumn("JBR_RADNIK").LazyLoad().Cascade.All().Inverse();
}
```



Mapiranje

- Primer: Podklasa Sef

```
public class Sef : Radnik
{
    public virtual IList<Prodavnica> SefujeProdavnice {get; set;}

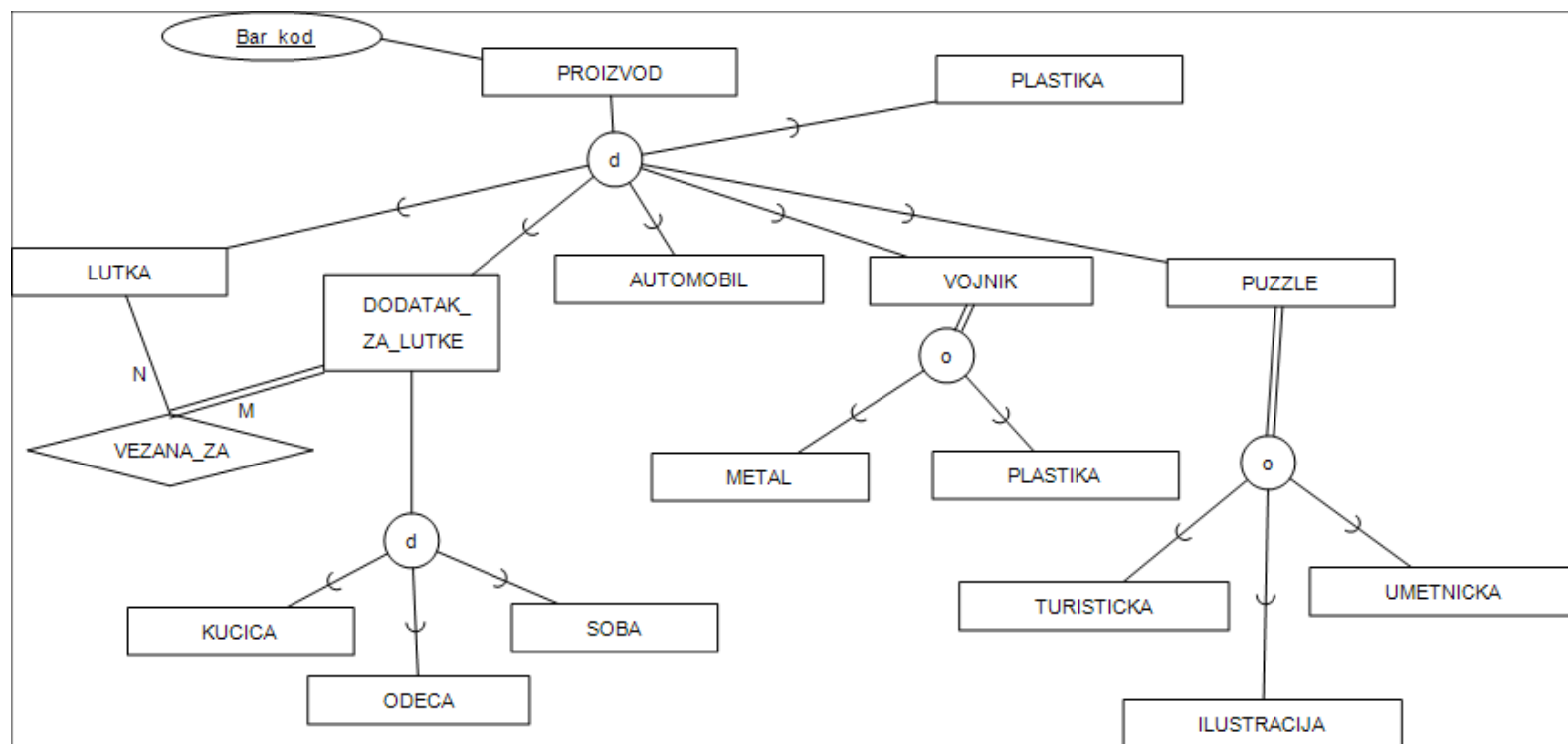
    public Sef()
    {
        SefujeProdavnice = new List<Prodavnica>();
    }
}

public class SefMapiranje : SubclassMap<Sef>
{
    public SefMapiranje()
    {
        DiscriminatorValue(1);

        HasManyToMany(x => x.SefujeProdavnice)
            .Table("SEFUJE")
            .ParentKeyColumn("JBR_RADNIK")
            .ChildKeyColumn("BROJP")
            .Cascade.All();
    }
}
```

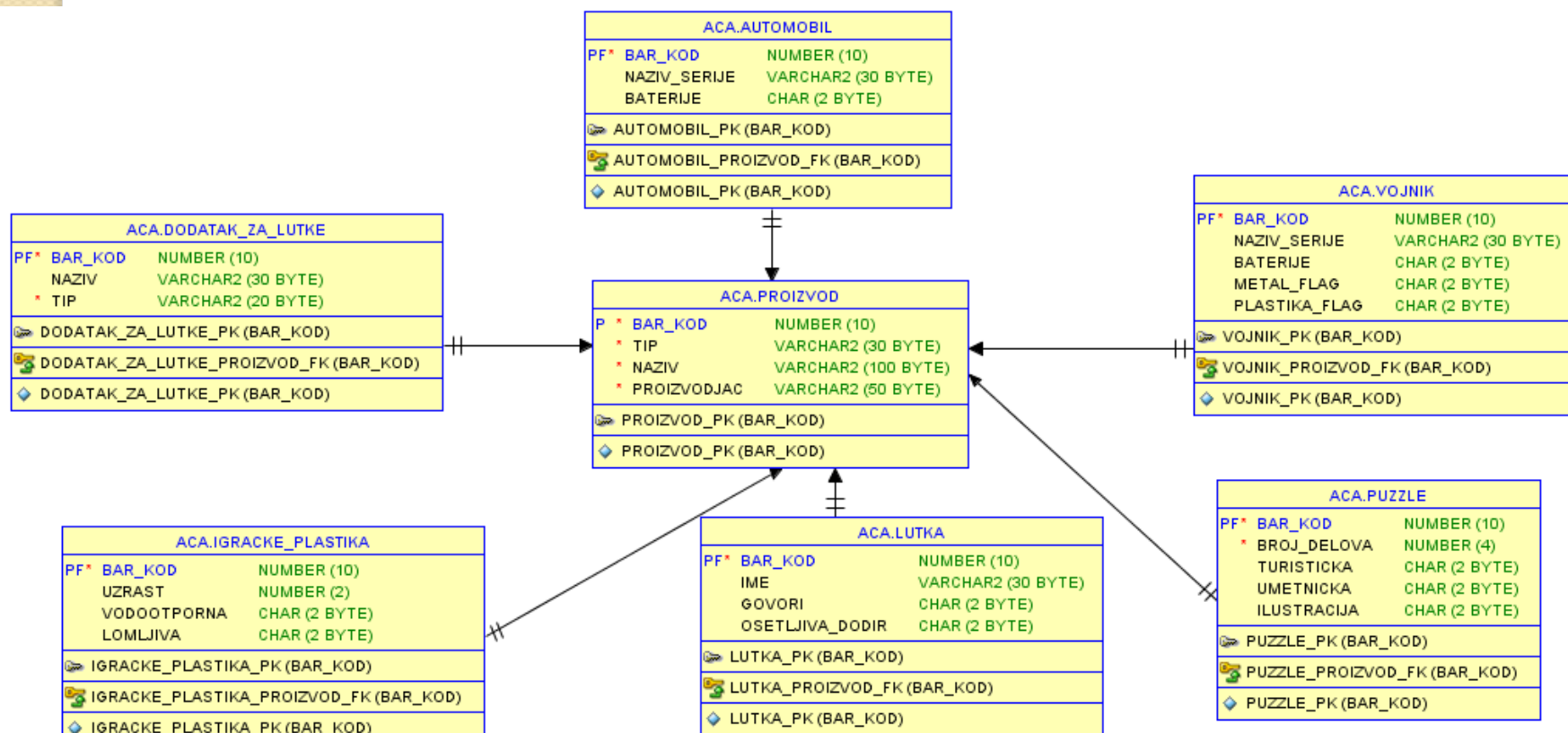

Mapiranje

- **Primer:** deo EER modela za bazu podataka Lanac prodavnica igračaka



Mapiranja

- Primer:** Deo relacionog modela baze podataka Lanac prodavnica igračka





Mapiranje

- Za prevođenje entiteta PROIZVOD i njegovih podklasa iskorišćena je alternativa 8A i hijerarhija klasa je prevedena tako da svakoj podklasi odgovara jedna tabela u relacionoj bazi podataka.
- Za mapiranje ovakve strukture Fluent NHibernate koristi strategiju Table-per-Class.
- Nadklasa Proizvod se mapira na uobičajeni način.



Mapiranje

```
public class Proizvod
{
    public virtual int BarKod { get; set; }
    public virtual string Tip { get; set; }
    public virtual string Naziv { get; set; }
    public virtual string Proizvodjac { get; set; }
}

class ProizvodMapiranje : ClassMap<Proizvod>
{
    public ProizvodMapiranje()
    {
        Table("PROIZVOD");

        //Id(x => x.BarKod).Column("BAR_KOD").GeneratedBy.TriggerIdentity();
        //generisanje kljuca uz pomoc sekvence
        Id(x => x.BarKod).Column("BAR_KOD").GeneratedBy.SequenceIdentity("ACA.PROIZVOD_BARKOD_SEQ");

        Map(x => x.Tip).Column("TIP");
        Map(x => x.Naziv).Column("NAZIV");
        Map(x => x.Proizvodjac).Column("PROIZVODJAC");
    }
}
```



Mapiranja

- Za svaku podklasu se kreira posebna domenska klasa koja je izvedena iz osnovne klase.
- U konkretnom slučaju biće kreirane domenske klase: IgrackaPlastika, Lutka, DodatakLutka, Automobil, Vojnik, Sagalica. Ove domenske klase su izvedene iz klase Proizvod.
- Prilikom mapiranja podklasa, klasa za mapiranje se izvodi iz Fluent NHibernate klase `SubclassMap<T>`.
- Metoda **KeyColumn** specificira primarni ključ podklase koji je istovremeno i strani ključ koji referencira nadklasu.



Mapiranje

```
public class Vojnik : Proizvod
{
    public virtual string NazivSerije { get; set; }
    public virtual string Baterije { get; set; }
    public virtual string Metal { get; set; }
    public virtual string Plastika { get; set; }
}

class VojnikMapiranje : SubclassMap<Vojnik>
{
    public VojnikMapiranje()
    {
        Table("VOJNIK");

        KeyColumn("BAR_KOD");

        Map(x => x.NazivSerije).Column("NAZIV_SERIJE");
        Map(x => x.Baterije).Column("BATERIJE");
        Map(x => x.Metal).Column("METAL_FLAG");
        Map(x => x.Plastika).Column("PLASTIKA_FLAG");
    }
}
```



Mapiranje

```
ISession s = DataLayer.GetSession();
```

```
Vojnik v = new Vojnik();
```

```
v.Tip = "VOJNICI";
```

```
v.Naziv = "Specijalac od olova";
```

```
v.Proizvodjac = "Proizvodjac vojnika";
```

```
v.NazivSerije = "Leto 2015";
```

```
v.Baterije = "Da";
```

```
v.Metal = "Da";
```

```
v.Plastika = "Ne";
```

```
s.Save(v);
```

```
s.Close();
```




Mapiranje

```
ISession s = DataLayer.Session();  
  
IList<Proizvod> proizvodi = s.QueryOver<Proizvod>()  
    .Where(p => p.BarKod == 13)  
    .List<Proizvod>();  
  
Vojnik v = (Vojnik)proizvodi[0];  
  
s.Close();
```



Mapiranje

- Ukoliko se za prevođenje entiteta PROIZVOD i njegovih podklasa iskoristi alternativa 8B, ne postoji posebna tabela za nadklasu dok svaka podklasa dobija posebnu tabelu.
- Za mapiranje ovakve strukture Fluent NHibernate koristi strategiju Table-per-Concrete-Class.
- U našem primeru ne bi više postojala tabela PROIZVOD, dok bi postojale tabele koja odgovaraju podklasama koje bi nasledile sve kolone entiteta PROIZVOD.



Mapiranja

```
class ProizvodMapiranja : ClassMap<Proizvod>
{
    public ProizvodMapiranja()
    {
        //Tabela PROIZVOD ne postoji
        //Table("PROIZVOD");

        //naznaka da se radi o osnovnoj klasi
        //kod Table-per-Concrete-Class strategije
        //svojstva ove klase se pridodaju
        //izvedene klase
        UseUnionSubclassForInheritanceMapping();

        //generisanje ključa uz pomoć sekvence
        Id(x => x.BarKod).Column("BAR_KOD").GeneratedBy.SequenceIdentity("ACA.PROIZVOD_BARKOD_SEQ");

        Map(x => x.Tip).Column("TIP");
        Map(x => x.Naziv).Column("NAZIV");
        Map(x => x.Proizvodjac).Column("PROIZVODJAC");
    }
}
```



Mapiranja

```
class VojnikMapiranja : SubclassMap<Vojnik>
{
    public VojnikMapiranja()
    {
        Table("VOJNIK");

        //osnovna klasa je apstraktna
        Abstract();

        Map(x => x.NazivSerije).Column("NAZIV_SERIJE");
        Map(x => x.Baterije).Column("BATERIJE");
        Map(x => x.Metal).Column("METAL_FLAG");
        Map(x => x.Plastika).Column("PLASTIKA_FLAG");
    }
}
```



Mapiranja

- Mapiranja nadklasa i podklasa su izuzetno problematična.
- Treba izbegavati podklasa koje nemaju dodatna svojstva.
- **Primer:** Nema potrebe uvoditi posebne domenske podklase za klasu Odeljenje.
- **Primer:** Namerno je izostavljena specijalizacija klase Vojnik po tipu materijala. Moguća je, ali nema previše smisla u praktičnoj realizaciji.
- **Primer:** Uvođenje podklase šef zavisi od konkretnog rešenja. Ima dodatno svojstvo koje je specifično samo za šefove.
- Zbog nedostatka podrške za višestruko nasleđivanje nije moguće implementirati deljive podklase.
- Ne postoji posebna podrška za implementaciju kategorija. Kategorije se implementiraju korišćenjem References i/ili HasOne relacija.



Mapiranje

- Za implementaciju deljive specijalizacije koristi se specijalni oblik **DiscriminateSubClassesOnColumn** metode.

```
DiscriminateSubClassesOnColumn("")  
    .Formula("predikat")
```

- Pri tome *predikat* ima oblik

```
CASE WHEN (METAL_FLAG = 'Da' AND PLASTIKA_FLAG = 'Ne' ) THEN 'Metal'  
      WHEN (METAL_FLAG = 'Ne' AND PLASTIKA_FLAG = 'Da' ) THEN 'Plastika'  
      WHEN (METAL_FLAG = 'Da' AND PLASTIKA_FLAG = 'Da' ) THEN  
      'MetalPlastika'  
      ELSE 'Nepoznato'  
END
```



Mapiranja

- **ComponentMap** – metoda koja specificira mapiranje komponente koja se često javlja kod domenskih entiteta.

```
public class Address
{
    public int Number { get; set; }
    public string Street { get; set; }
    public string City { get; set; }
    public string PostCode { get; set; }
}

public class Person
{
    public Address Address { get; set; }
}

public class Company
{
    public Address Address { get; set; }
}
```




Mapiranje

```
public class AddressMap : ComponentMap<Address>
{
    public AddressMap()
    {
        Map(x => x.Number);
        Map(x => x.Street);
        Map(x => x.City);
        Map(x => x.PostCode);
    }
}
```

```
public PersonMap()
{
    Component(x => x.Address);
}
```

```
public CompanyMap()
{
    Component(x => x.Address);
}
```