

# **Projekt sustava, Dizajn sustava (Systems Design)**

---

**2014/15.08**

# Opći dizajn

- ❑ **Opći dizajn (konceptualni, visoke razine, arhitekturni) → funkcionalne specifikacije**
- ❑ **Odabir tehničke arhitekture sustava**
  - Centralizirana ili distribuirana obrada i pohrana? Kako? Tehnologije?
  - Softver: nabavljeni, napravljen po mjeri, mješavina? Razvojni alati?
- ❑ **Analiza i distribucija podataka**
  - pretvorba konceptualnog modela podataka u logički model (relacijski, postrelacijski, objektno-relacijski), ako nije učinjena ranije
- ❑ **Analiza i distribucija procesa**
  - pretvorba logičkog modela procesa u fizički model za odabranu arhitekturu  
→ shema aplikacije
- ❑ **Opći dizajn sučelja**
  - izgled, ergonomija, tzv. 'look and feel'

# Detaljni dizajn

## ❑ Detaljni dizajn, dizajn na nižoj razini → tehničke specifikacije

## ❑ Izrada fizičkog modela podataka

- pretvorba logičkog modela podataka u fizički model podataka za odabrani SUBP → shema baze podataka
  - prilagodba modela mogućnostima i ograničenjima SUBP
  - fizički parametri (volumetrija) i ugađanje baze podataka (indeksi)
- oblikovanje fizičkih datoteka

## ❑ Dizajn programa

- utvrđivanje strukture programa na temelju modela procesa
  - (logički) proces ili skup procesa ↔ jedan ili više modula/razreda
- preciziranje programske logike

## ❑ Dizajn sučelja

- dizajn sučelja sustava – protokoli pristupa i razmjene podataka
- oblikovanje zaslonskih maski i izvješća

# Dokumentiranje dizajna

## ❑ Dizajn programa (program design)

- proces pretvorbe zahtjeva na programsku podršku u oblik koji omogućuje programiranje
- opis jezikom za projektiranje programa (PDL - Program Design Language) pri čemu program napisan pomoću PDL nema oblik izvedbenog programa

## ❑ Primjer: \Prilozi\SpecifikacijaDizajna.dot

- predložak specifikacije

## ❑ Primjer: \Prilozi\Firma-Dizajn.doc

- primjer specifikacije

# Arhitektura sustava

---

# Dizajn arhitekture sustava

## ❑ Dizajn arhitekture

- sastoji se od planova koji definiraju pojedine komponente sustava
  - računalnu opremu
  - programsku podršku
  - komunikacije
  - sustav zaštite
  - globalnu podršku aplikacije

## ❑ Uobičajeni modeli arhitekture

- poslužiteljska (server-based) – obrada se obavlja na poslužitelju
- klijentska (client-based) – obrada se obavlja na osobnom računalu
- klijent-poslužitelj (client-server based) – kombinacija prethodne dvije

## ❑ Model mreže

- prikaz glavnih komponenti sustava, njihove fizičke lokacije i način njihovog međusobnog povezivanja

## ❑ Specifikacije računalne opreme i programske podrške

- podloga (stavke) za nabavu ili izradu

# Elementi arhitekture sustava

## ❑ Osnovne funkcije sustava

- Pohrana podataka (data storage) – baza podataka
- Pristup podacima (data access logic) – npr. ADO.NET, Entity Framework ...
- Elementi obrade (application logic) – aplikacijski program, pohranjene proc.
- Sučelje (presentation logic) – zaslonske maske

## ❑ Osnovne hardverske komponente: klijenti, poslužitelji i mreža

### ❑ Poslužitelji

- Velika računala (Mainframe)
- Mala računala (Minicomputer)
- Mikroračunala (Microcomputer) – PC

### ❑ Klijenti

- Klasični terminali - ansi, vt220, IBM 3270, ...
- Mikroračunala – PC, pametni telefoni, ...
  - pristup emulatorima terminala ili udaljenim radnim ploham (RDC)
- Terminali posebne namjene
  - bankovni terminali (bankomati), Internet kiosk, ...

# Centralizirana obrada

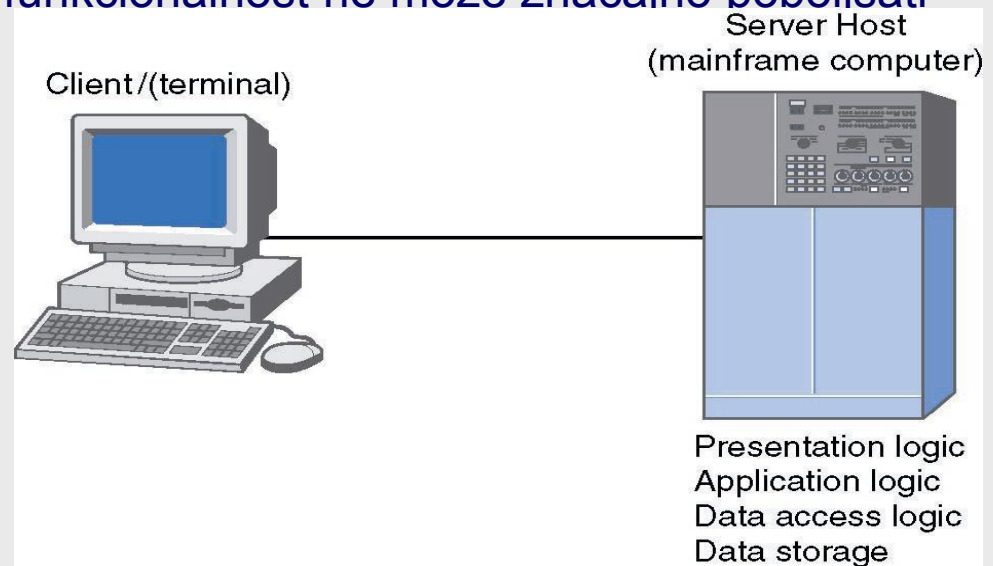
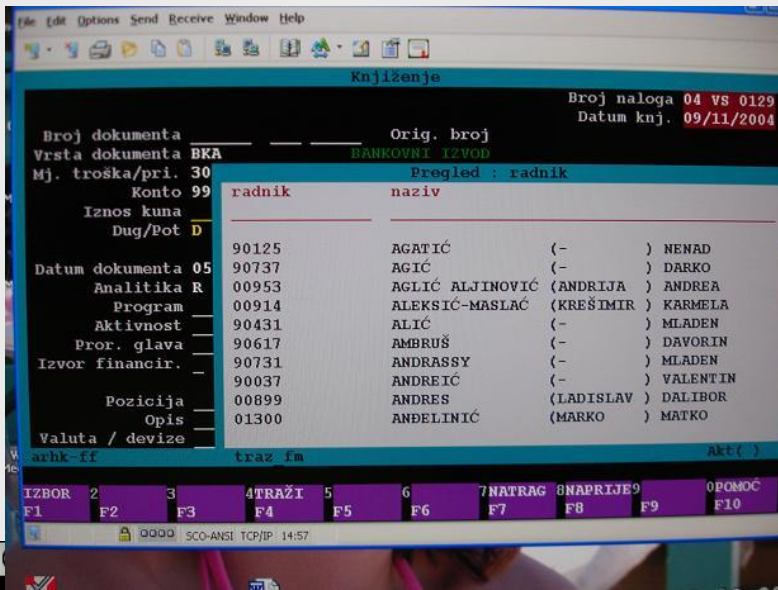
❑ Sve funkcije sustava obavljaju se na poslužitelju

❑ Višekorisnički poslužitelj + terminali

- pohrana podataka - datoteke i baze podataka
- poslovna logika - programska podrška
- korisničko sučelje - uobičajeno znakovno sučelje
- sučelje sustava - mrežne i druge komponente

❑ Distribuirana prezentacija

- nadgradnja zamjenom znakovnog sučelja grafičkim, koje se izvodi na PC
- produljuje vijek aplikacija, ali se funkcionalnost ne može značajno poboljšati





# Dvoslojna arhitektura klijent-poslužitelj (client-server)

## ❑ Klijent - jednokorisničko računalo

- sučelje, obrada (i pristup podacima)
- povezljivost na poslužitelje i druge klijente

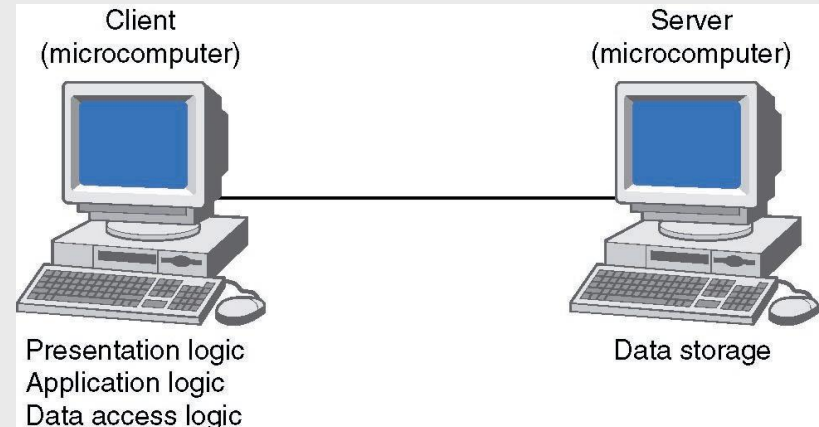
## ❑ Poslužitelj - višekorisničko računalo

- baza podataka (i pristup podacima)
- povezljivost s klijentima i poslužiteljima

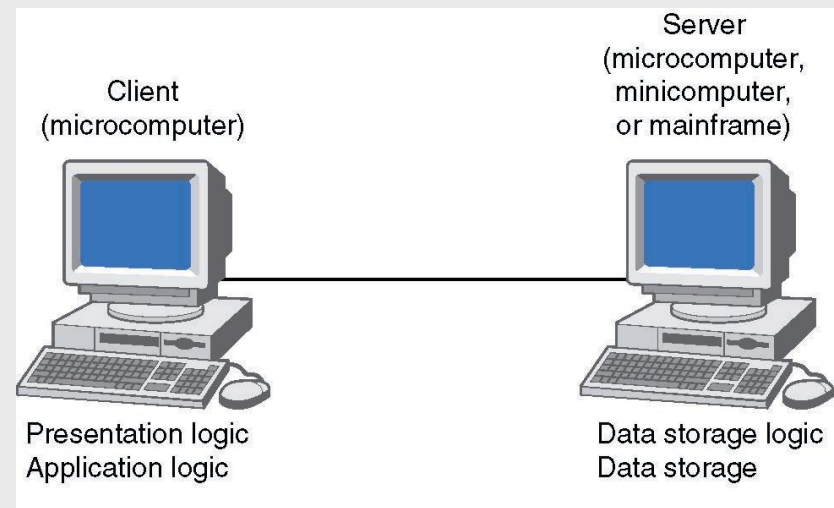
## ❑ Korisnicima izgleda kao da njihovo računalo (PC) obavlja cijeli posao

- Istina ako je i pohrana podataka na klijentskom računalu

## ❑ Primjer: debeli klijent



## ❑ Primjer: tanki klijent



# Debeli klijent

## ❑ Debeli klijent, u novije vrijeme “bogat” klijent (rich client)

- Podatkovna logika integrirana u klijenta
- Nema obrade podataka na poslužitelju ili je obrada minimalna
- Klijent može imati lokalno spremište (bazu) podataka
- Minimalna ili nikakva elastičnost na promjene poslovne politike

## ❑ Prednosti

- brzi početni razvoj aplikacije
- veća samostalnost klijenta
- rasterećenje glavnog računala (poslužitelja)

## ❑ Nedostaci

- poslovna logika integrirana u klijenta
- promjena logike zahtijeva instaliranje nove verzije na svim klijentima
- razvoj velike aplikacije s vremenom postaje vrlo složen
- potreban veći broj klijentskih računala dovoljne procesne moći

# Tanki klijent

## ❑ Tanki klijent (thin client)

- Podatkovna logika (a nekad i većina elemenata obrade) se nalazi na poslužitelju
- Osnovna namjena klijenta je prikaz podataka
- Tipični primjeri tankog klijenta: web preglednik, Remote Desktop, Team Viewer, ...

## ❑ Prednosti

- manja složenost razvoja velikih aplikacija (serverski dio i klijentski dio)
- olakšana distribucija, kao tanki klijent može se koristiti npr. općenito dostupan web preglednik
- lakše održavanje – centralizirana promjena poslovne logike
- klijentska računala ne moraju imati veliku moć obrade

## ❑ Nedostaci

- veliko opterećenje glavnog računala, a to znači skupo glavno računalo
- veće mrežno opterećenje (gotovo za svaku promjenu ide se na server)
- lošija funkcionalnost kada se kao klijent koristi web preglednik
- naglo povećanje složenosti zahtjevnog grafičkog sučelja

# Primjer arhitektura klijent-poslužitelj

## ❑ Napraviti aplikaciju koja će

- prikazivati listu najboljih 10 partnera poredanih po iznosu prometa
- za odabranog partnera prikazati njegove dokumente
- problem oblikovanja dinamičkog skupa podataka i pisanja upita
  - potpuna informacija o partneru zahtijeva uniju zbog specijalizacije
  - kako dinamički povezati uniju i dokumente ?

## ❑ Primjer: Arhitecture\DebeliKlijent

- Entity Framework + Linq upit za dohvat podataka o najboljim partnerima
- dinamička selekcija dokumenata za odabranog partnera, EF ili Linq upitom ugrađenim u programski kod programa

## ❑ Primjer: Arhitecture\TankiKlijent

- poziv pohranjene procedure (kroz Entity Framework) koja vraća skup najboljih partnera
- poziv pohranjene procedure (kroz Entity Framework) za dohvat dokumenata određenog partnera

# Primjer debelog klijenta

## ❑ Primjer: 📁 Arhitekture\DebeliKlijent - FormDebeli\_Load

- Model podataka definiran kroz EF-model
- Povezivanje kontrole (dataGridViewPromet) s izvorom - u dizajnu
- Dohvat podataka za izvor podataka (prometPartneraBindingSource) - u kodu

```
var osobe = context.Partner.AsNoTracking().OfType<Osoba>().  
    Select(o => new{  
        o.IdPartnera, Broj = o.OIB,  
        Naziv = o.PrezimeOsobe + " " + o.ImeOsobe,  
        Promet = o.Dokumenti.Sum(d => d.IznosDokumenta)  
    });  
  
var tvrtke = context.Partner.AsNoTracking().OfType<Tvrtka>().  
    Select(t => new{ t.IdPartnera, Naziv = t.NazivTvrtke,  
        Broj = t.MatBrTvrtke,  
        Promet = t.Dokumenti.Sum(d => d.IznosDokumenta) }  
    );  
  
var partneri = osobe.Union(tvrtke);  
var najboljiPartneri = partneri.  
    OrderByDescending(p => p.Promet).  
    Take(10);  
  
prometPartneraBindingSource.DataSource = najboljiPartneri.ToList();
```

# Primjer debelog klijenta – selekcija detalja

## ❑ Primjer: 📁 Arhitekture\DebeliKlijent

- Dinamičko kreiranje upita i povezivanje kontrole za prikaz

```
private void dataGridViewPromet_CurrentCellChanged(...) {  
    ...  
    int idPartnera =  
        (int) dataGridViewPromet.CurrentRow.Cells[0].Value;  
  
    dataGridViewDokument.DataSource = context.Dokument.  
        AsNoTracking().  
        Where(d => d.IdPartnera == IdPartnera).  
        Select(d => new {  
            d.IdDokumenta, d.VrDokumenta,  
            d.DatDokumenta, d.IznosDokumenta  
        }).  
    ToList();  
}
```

# Primjer tankog klijenta

## ❑ Primjer: Arhitekture\TankiKlijent

- Model podataka definiran kroz Entity Framework
- Procedure dodane u model, stvoren novi složeni tip podataka
- povezivanje se svodi na poziv pohranjenih procedura za skup partnera i dokumente partnera

```
FormTanki_Load () {  
    dataGridPromet.DataSource =  
        context.ap_PrometPartnera();  
}
```

```
private void dataGridPromet_CurrentCellChanged(...) {  
    ...  
    int idPartnera =  
        (int)dataGridPromet.CurrentRow.Cells[0].Value;  
    dataGridDokument.DataSource =  
        context.ap_DokumentPartnera(idPartnera);  
}
```

# Primjer promjene zahtjeva

## ☐ Promjena zahtjeva

- Korisnik se, naravno, predomislio i želi pregled 20 najboljih partnera, ali dobrima smatra samo one koji kupuju i ljeti i zimi i ostvarili su barem  $X$  kn prometa u zadnje 2 godine.

## ☐ Na debelom klijentu treba

- promijeniti upite u izvornom kodu
- prevesti ga u novu izvršnu inačicu
- instalirati novu verziju aplikacije pojedinom korisniku
- zahtjevno, sporo i skupo

## ☐ Za tanki klijent treba

- na poslužitelju BP promijeniti pohranjenu proceduru za dohvat podataka
- centralizirano, jednostavno, brzo i jeftinije

## ☐ Napomena: promjena modela mijenja i tanki i debeli klijent



# Troslojna ili višeslojna arhitektura klijent-poslužitelj

## ❑ Distribucija baza podataka i poslovne logike na zasebne poslužitelje

- poslužitelj aplikacija + poslužitelj baza podataka + klijent
- poslužitelj baza podataka
  - upravljanje podacima
- poslužitelj aplikacija
  - upravljanje transakcijama, "preuzeto" s podatkovnog poslužitelja
  - dio ili čitava poslovna logika, "preuzeta" s klijenta
- klijent
  - korisničko sučelje
  - dio poslovne logike - onaj koji se ne mijenja ili je osobnog karaktera

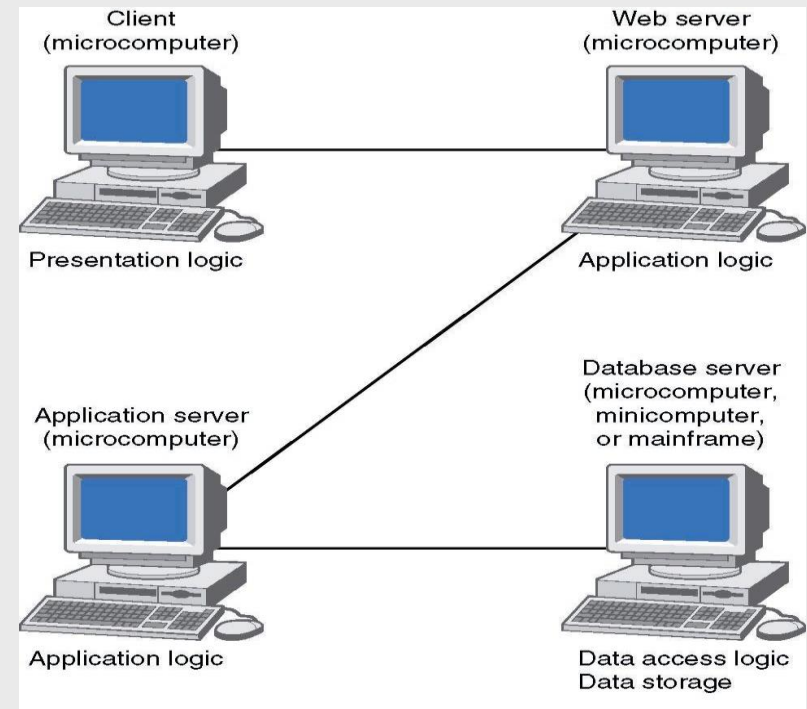
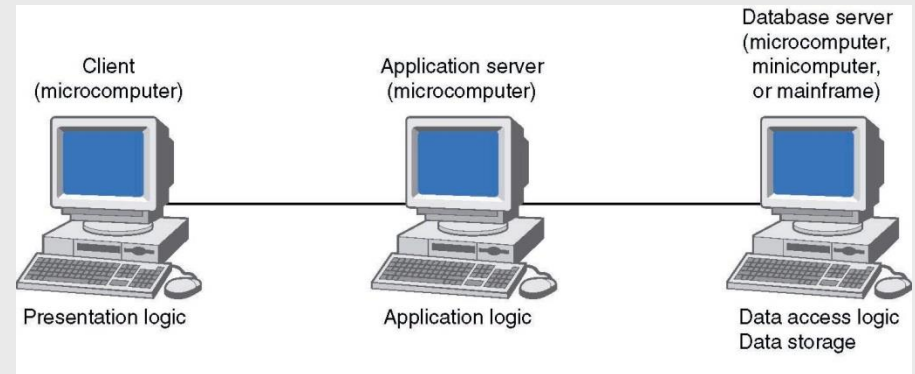
## ❑ Prednosti

- bolja raspodjela opterećenja
- veća skalabilnost - mogućnost ekspanzije, npr. povećanja broja korisnika, bez preopterećenja ili promjene procedura

## ❑ Nedostaci

- vrlo složen dizajn i razvoj
- problem raspodjele podataka, procesa, sučelja
- veće opterećenje mreže

## ❑ Primjeri



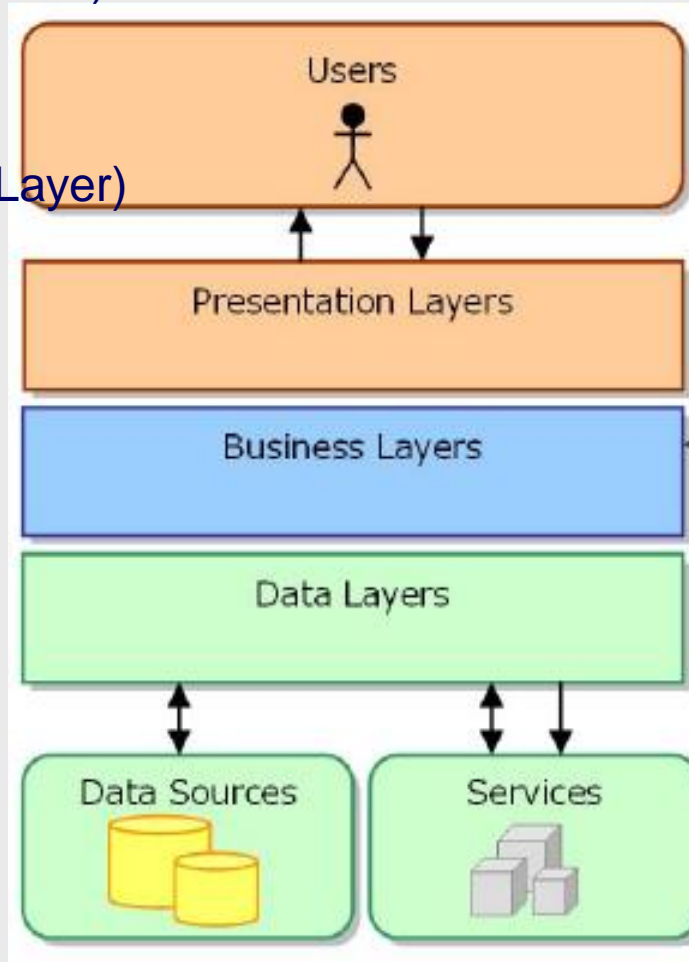
# Višeslojna aplikacija

---

# Višeslojna aplikacija

## □ Aplikacija se može podijeliti u više razina – logičkih slojeva, npr.

- prezentacijski sloj (PL – Presentation Layer)
  - grafičko sučelje (GUI - Graphical User Interface) – Win / Web / Mobile
  - skup javnih servisa
- poslovni sloj (BL – Business Layer)
  - sloj poslovne logike (BLL - Business Logic Layer)
  - poslovne klase - ne samo podaci već i ponašanje i validacija
- podatkovni sloj (DL – Data Layer)
  - Sloj pristupa podacima (DAL – Data Access Layer)
    - ADO.NET, Entity Framework, ...
- pohrana podataka (Data Storage) –
  - + kod na BP (stored procedure)
  - + pogledi na BP (view)
  - ...



# Odnosi među slojevima

## ❑ Kako postići neovisnost slojeva?

- promjena nekog sloja treba imati minimalni utjecaj na ostale slojeve

## ❑ Interakcija prema dolje (Top-down interaction)

- viši slojevi mogu komunicirati s nižim slojevima, ali ne i obrnuto
- **stroga (strict):** komunikacija samo sa slojem koji je direktno ispod
  - npr. promjena načina pristupa podacima ne mijenja ništa u prezentacijskom sloju i izaziva minimalne promjene u poslovnom sloju
- labava (loose): moguća interakcija s bilo kojim slojem ispod
  - Povećava performanse, ali i ovisnost slojeva

## ❑ Poprečne komponente (Crosscutting Concerns), zajedničke za sve slojeve

- Npr. za iznimke, praćenje traga, zajedničke biblioteke funkcija, ...

# Gdje provjeriti podatke ?

- ❑ **Je li potrebno raditi validaciju ako BP čuva integritet podataka?**
  - Ne propuštati neispravne podatke, ako znamo da su neispravni!
  - Kako oblikovati složenija pravila?
  - Što ako promijenimo spremište?
  
- ❑ **Validaciju uvijek treba napraviti na poslovnom sloju**
  - Ne smijemo vjerovati podacima pristiglim iz prezentacijskog sloja
  - Validacija na prezentacijskom sloju poželjna zbog brzine aplikacije

# Gdje stvarati poslovni objekt?

## ❑ Poslovni objekt

- posjeduje svojstva poslovnog entiteta (podaci, ponašanje, validacija)

## ❑ Poslovna logika treba biti neovisna o načinu pristupa podacima

- Kako prenijeti podatke iz podatkovnog sloja u poslovni sloj ?
- Može li se ostvariti potpuna neovisnost BL o DL ?
  - Teško, ali se ovisnost može minimizirati

Različiti pristupi rješavanju ovog problema

## ❑ Kad bi DL stvarao poslovni objekt

- Problem unakrsne veze DL-BL, jer bi DL trebao interakciju „prema gore”
- Staviti DL i BL unutar istog dll-a? → narušava podjelu po slojevima
- Izdvojiti poslovni objekt u zasebni, dijeljeni sloj?
  - poslovni objekt bio bi sveden samo na podatke, a logika bi ostala u BL
  - gubi se smisao poslovnog objekta
  - otvorila bi se mogućnost nekontroliranog pisanja po objektu (npr. iz PL)

## ❑ Poslovni objekt stvara se u BL-u

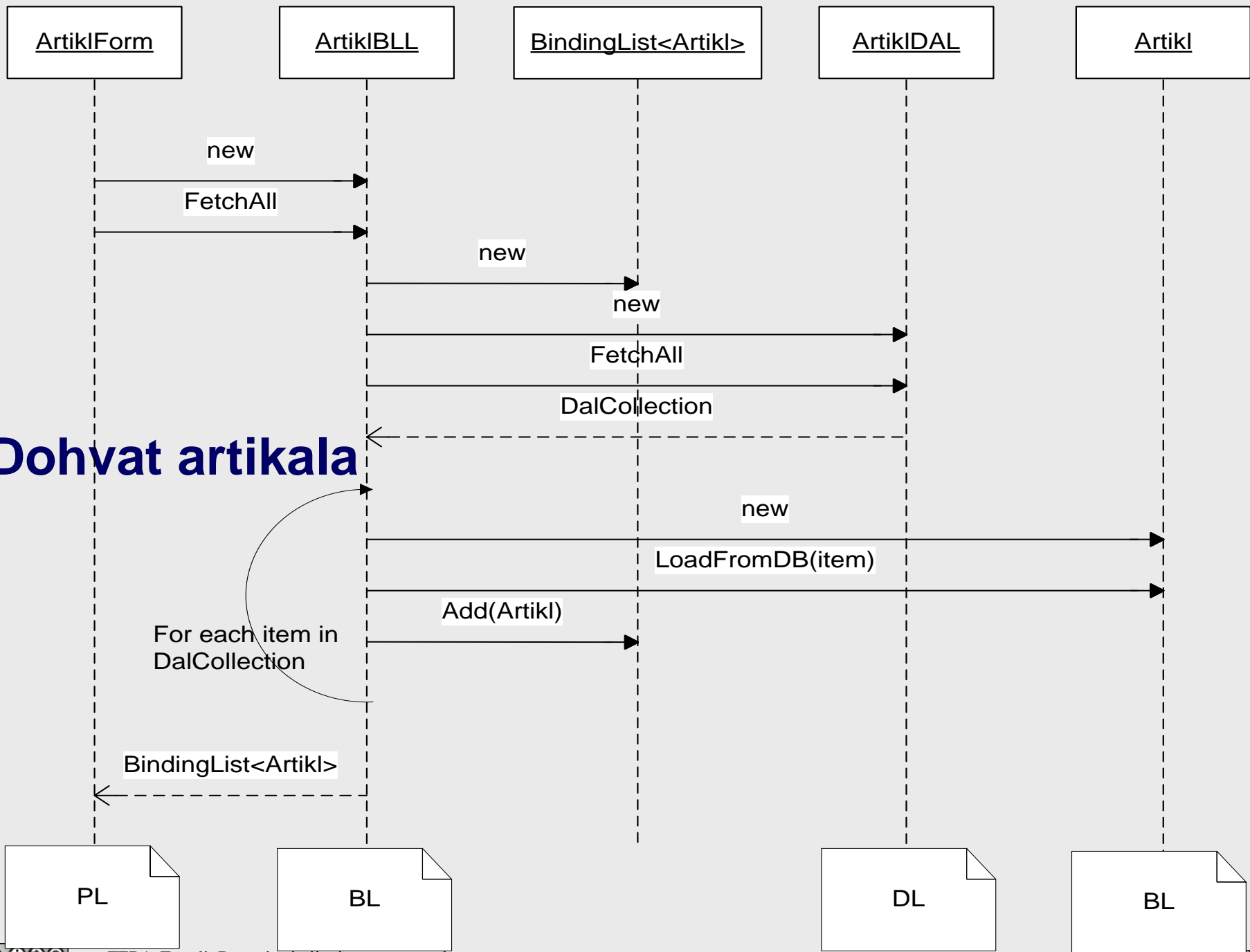
# Poslovni objekt

- Definiran u poslovnom sloju
- Posjeduje svojstva poslovnog entiteta
  - Podaci, ponašanje, validacija
- Poslovna logika treba biti neovisna o načinu pristupa podacima
  - Dohvat odvojen u drugi dio parcijalnog razreda (naknadno objašnjeno)

## ❑ Primjer: Viseslojna\BLL\Artikl.cs

```
public partial class Artikl : IDataErrorInfo    {  
    // prvi dio parcijalne klase  
    public int SifArtikla { get; set; }  
  
    ...  
  
    public string Error  
    {  
        get  
        {  
  
            StringBuilder sb = new StringBuilder();  
            string s = this["SifArtikla"];  
            if (!string.IsNullOrEmpty(s))
```

# Dohvat artikala





# Kako sučelje dođe do podatka ?

## ❑ Pretpostavimo da je aplikacija dobro uslojena

- Prezentsijski sloj zove postupke posrednika koji dalje komunicira s DAL-om (striktna interakcija s vrha prema dolje)
- u našem primjeru to bude *BIIProvider*

## ❑ Sučelju se vraća poslovni objekt (podaci + poslovna logika) ili kolekcija poslovnih objekata





- Alternativa XML ili neki drugi oblik koji sadrži isključivo podatke
  - prednost: općenitost, neovisnost
  - nedostatak: otežana validacija, moguće nekonzistentni podaci, poslovna logika i validacija na raznim mjestima

# Primjer dohvata i prikaza podataka

## ❑ Primjer: Viseslojna \ PL \ ArtiklForm

- Tip podataka se postavlja na razred Artikl definiran u BL
- prilikom učitavanja forme instancira se **BLL provider** i vrši dohvat

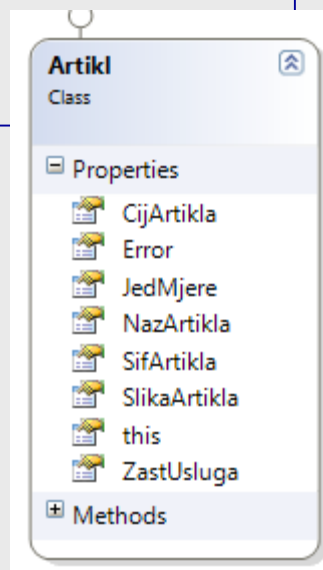
```
// referenca na BLL sloj
ArtiklBLL bll = new ArtiklBLL();
...
private void ArtiklForm_Load(object sender, EventArgs e) {
    artiklBindingSource.DataSource = bll.FetchAll();
    ...
}
```

artiklBindingSource System.Windows.Forms	
   	
+	(ApplicationSettings)
(Name)	artiklBindingSource
AllowNew	True
DataMember	
DataSource	BLL.Artikl


## ❑ Primjer: Viseslojna \ BLL \ Artikl.cs

- Artikl je primjer poslovnog objekta

❑ **PL (forma) ne zna ništa o načinu pohrane**  
**to jest ne ovisi o podatkovnom sloju**



# Načini punjenja poslovnog objekta

- ❑ Različiti pristupi punjenja podacima iz izvora podataka
  - a) Metode poslovnog sloja pišu u poslovni objekt
    - Metodama postaviti prava pristupa na `internal`
  - b) Poslovni objekt ima odgovornost za svoju perzistenciju
    - `internal` postupci koji primaju konkretni objekt iz DAL sloja
  - c) posebna sučelja u zajedničkom sloju koja DAL treba implementirati
    - Ovisnost samo o sučelju – bolje, ali zahtjevnije za izvesti
  - d) ... XML, DataSet, DTO (Data Transfer Object), Dependency Injection, Object Factory
  
- ❑ Za slučajeve a ili b praktično je definirati parcijalne razrede i odvojiti postupke u različite datoteke
  - Očuvana poslovna logika i učajurenje
  - Ovisnost o DAL-u postoji, ali je lokalizirana
  
- ❑ Primjer:  Viseslojna \ BLL \ DALSpecific \ Artikl.cs  
(objašnjenje uskoro slijedi)

# Izvedba podatkovnog sloja pomoću ADO.NET-a

## ❑ DAL isporučuje **IDataReader**

- Brzo i bez kopiranja podataka
- Ograničeno na ADO.NET
- Neprikladno ako BLL i DAL nisu na istom fizičkom sloju
- Za ažuriranje podataka treba slati pojedinačne podatke ili strukturirane podatke (XML, Dictionary<string, object>, DTO...)

## ❑ Primjer: **Viseslojna\DAL.ADO\ArtiklDAL.cs**

```
public class ArtiklDAL {  
    public IDataReader FetchAll() {  
        SqlConnection db = ...  
        SqlCommand cmd = db.CreateCommand();  
        cmd.CommandText = "[dbo].[ap_ArtiklList_R]";  
        cmd.CommandType = CommandType.StoredProcedure;  
        db.Open();  
        return cmd.ExecuteReader(CommandBehavior.CloseConnection);  
    }  
}
```

# Komunikacija između BL i DL sloja (IDataReader)

## ❑ BLL sadrži referencu na DAL sloj

- Uzima objekt iz DAL-a, iterira po njemu, a učitavanje pojedinačnog podatka prepušta samom poslovnom objektu
- Vraća kolekciju poslovnih objekata

## ❑ 📁 Viseslojna\ BLL - BLLProviders \ ArtiklBLL.cs

```
public class ArtiklBLL{  
    private ArtiklDAL dal = new ArtiklDAL();  
    public BindingList<Artikl> FetchAll() {  
        BindingList<Artikl> list = new BindingList<Artikl>();  
        using (IDataReader dr = dal.FetchAll()) {  
            while (dr.Read()) {  
                Artikl artikl = new Artikl();  
                artikl.LoadFromDB(dr);  
                list.Add(artikl);  
            }  
        }  
        ...  
    }  
}
```

# Punjenje poslovnog objekta (IDataReader)

## ❑ **Artikl definiran kao parcijalni razred**

- Nije potpuna neovisnost, ali zadovoljavajuća
- (vidi slajd “Načini punjenja poslovnog objekta”)

## ❑ **Primjer: Viseslojna \ BLL \ DALSpecific \ Artikl.cs**

```
public partial class Artikl{    // drugi dio parcijalne klase
    internal void LoadFromDB(IDataReader dr){
        this.SifArtikla = (int) dr["SifArtikla"];
        ...
        this.CijArtikla = (decimal)dr["CijArtikla"];
        ...
        this.SlikaArtikla = dr["SlikaArtikla"] == DBNull.Value ?
                               null : (byte[])dr["SlikaArtikla"];
    }
}
```

# Alternativa IDataReader-u

- ❑ **Kada IDataReader ne bude zadovoljavajuće rješenje? Što onda?**
  - DAL isporučuje DTO objekt
  - DTO se lako se prenosi preko mreže (dvosmjerno)
- ❑ **Treba stvoriti nove razrede – potencijalno dupliranje posla**
  - kao prikladno rješenje za DTO može poslužiti EF
  - DAL isporučuje objekt iz Entity Frameworka

## ❑ **Primjer:** **Viseslojna \ DAL.EF \ ArtiklDAL.cs**

```
public class ArtiklDAL {  
    public List<DAL.EF.Artikl> FetchAll() {  
        using (FirmaEntities ctx = new FirmaEntities()) {  
            return ctx.Artikl.AsNoTracking().ToList();  
        }  
    }  
    ...  
}
```

# Komunikacija između BL i DL sloja (EF)

## ❑ Sličan princip kao i kod korištenja IDataReader-a

- Potrebne manje preinake u poslovnom sloju
- Prezentacijski sloj ostaje potpuno isti!

## ❑ BLL sadrži referencu na DAL sloj

- Uzima objekt iz DAL-a, iterira po njemu, a učitavanje pojedinačnog podatka prepušta samom poslovnom objektu
- Vraća kolekciju poslovnih objekata

## ❑ Viseslojna \ BLL \ BLLProviders \ ArtiklBLL.cs

```
public class ArtiklBLL{  
    private ArtiklDAL dal = new ArtiklDAL();  
    public BindingList<Artikl> FetchAll() {  
        BindingList<Artikl> list = new BindingList<Artikl>();  
        foreach(DAL.EF.Artikl dalObject in dal.FetchAll()) {  
            Artikl artikl = new Artikl();  
            artikl.LoadFromDB(dalObject);  
            list.Add(artikl);  
        }  
    }  
}
```



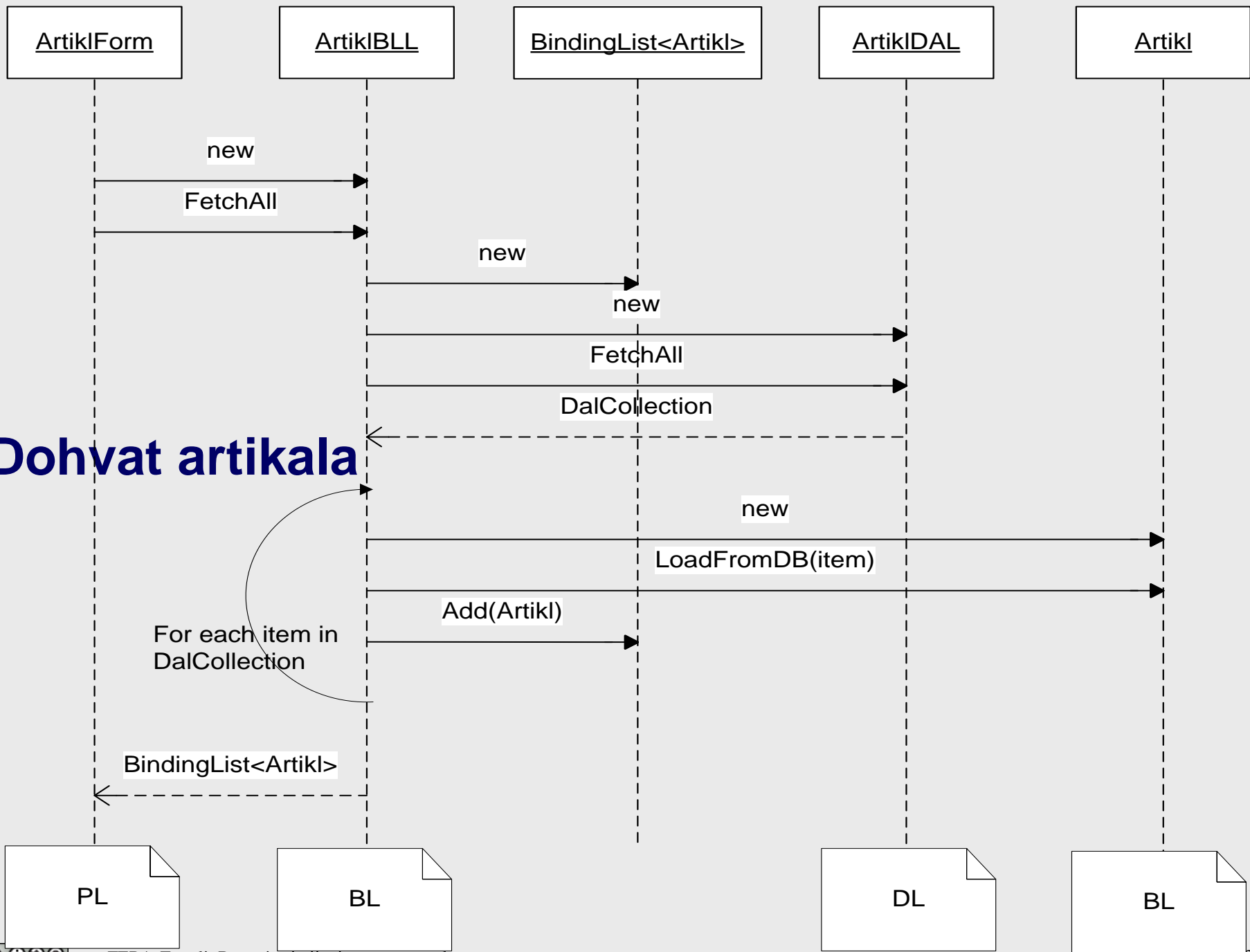
# Punjenje poslovnog objekta (EF)

- ❑ Slično kao i kod rješenja s `IDataReaderom`
- ❑ Primjer:  `Viseslojna \ BLL \ DALSpecific \ Artikl.cs`

```
public partial class Artikl
{
    internal void LoadFromDB(DAL.EF.Artikl a)
    {
        this.SifArtikla = a.SifArtikla;
        this.NazArtikla = a.NazArtikla;
        this.JedMjere = a.JedMjere;
        this.CijArtikla = a.CijArtikla;
        this.ZastUsluga = a.ZastUsluga;
        this.SlikaArtikla = a.SlikaArtikla;
    }
}
```

- ❑ Kad su nazivi svojstava isti, kopiranje se može automatizirati korištenjem *refleksije* (više na sljedećim predavanjima)

# Dohvat artikala



# Dodavanje, ažuriranje i brisanje artikla

- Obavlja se na pojedinačnom podatku
- Poziva se odgovarajući postupak na poslovnom sloju
- BL obavlja validaciju pa poziva konkretni postupak iz DAL sloja

❑ **Primjer:**  **Viseslojna \ BLL \ BLLProviders \ ArtiklBLL.cs**

```
public class ArtiklBLL{  
    public void Insert(Artikl a){  
        if (string.IsNullOrEmpty(a.Error)){  
            dal.Insert(a.SifArtikla, a.NazArtikla, a.JedMjere,  
                a.CijArtikla, a.ZastUsluga, a.SlikaArtikla);  
        }  
        else  
            throw new Exception("Validacija neuspješna: " +  
                a.Error);  
    }  
    ...  
}
```

# Primjer šifarničke forme izvedene višeslojno

## ❑ Primjer: uređivanje popisa Država u tablici/matrici

📁 Viseslojna \ BLL \ BLLProviders \ DrzavaBLL.cs

📁 Viseslojna \ BLL \ Drzava.cs

📁 Viseslojna \ BLL \ DALSpecific \ Drzava.cs

📁 Viseslojna \ DAL.ADO \ DrzavaDAL.cs

## ❑ Promjene se grupno spremaju u bazu

- Treba voditi evidenciju o promijenjenim, novim i obrisanim podacima

- Poslovni sloj

- prati promjene na *BindingList*
- obradom događaja *ListChanged*

- Dio logike nalazi se i u prezentacijskom sloju

- evidentiranje podataka koji će se obrisati
- u do sada realiziranom poslovnom sloju to se ne može
  - Obrada događaja *UserDeletingRow* u prezentacijskom sloju
- Slijedi bolje rješenje, primjer Firma.Win

# Nedostaci postojećeg rješenja

- ❑ **Nepraktična izvedba šifarničkih formi (prethodno opisano)**
  - Evidencija promjena se dijelom vodi kroz prezentacijski sloj
- ❑ **Svaka forma se brine sama za sebe i posebno dizajnira**
  - Navigacijska traka
  - Statusna traka
  - Snimanje i otkazivanje promjena na razini pojedine forme
- ❑ **Multipliciranje istog koda u poslovnim objektima**
  - Kod za evidentiranje stanja objekta
  - Obavijesti o promjenama stanja objekta
  - ...
- ❑ **Ideja: napraviti generičko rješenje**
  - Primjer: Firma.Win
  - Uspostaviti osnovni skup sučelja i baznih razreda (Firma.Framework)
  - Olakšati i automatizirati izradu GUI-a (Firma.Core)

# Primjer višeslojne aplikacije – FirmaWin

---

Primjer:  FirmaWin

# Struktura rješenja

## ❑ Prezencijski sloj

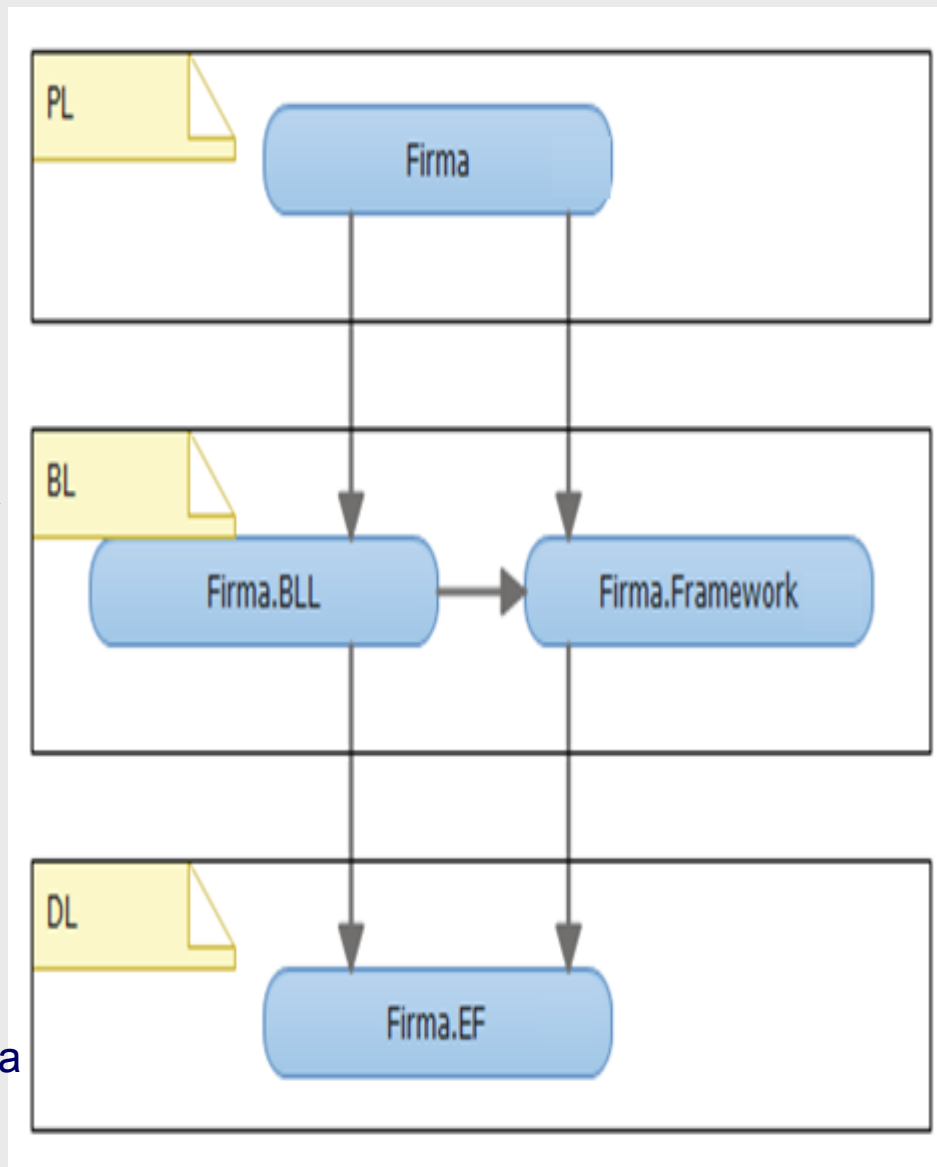
- FirmaWin \ Firma
  - Windows aplikacija

## ❑ Poslovni sloj

- FirmaWin \ Firma.Framework
  - Skup temeljenih razreda i sučelja za izradu poslovnih objekata
- FirmaWin \ Firma.BLL
  - BusinessEntities\DALSpecific za postupke koji ovise o implementaciji podatkovnog sloja

## ❑ Podatkovni sloj

- FirmaWin \ Firma.EF
- Realiziran korištenjem Entity Frameworka



# Slojevi aplikacije na primjeru Artikla

## PL - Firma.exe

**ArtiklForm**  
Class  
→ BaseForm

**BusinessBaseList<T>**  
Generic Class  
→ BindingList<T>

**BusinessBase**  
Abstract Class

## Firma.Framework.dll

## BL - Firma.BLL.dll

IBllProvider  
**ArtiklBllProvider**  
Class  
Methods  
Fetch  
FetchAll  
FetchLazy  
FetchLookup  
Save  
SaveChanges  
Validate

**Artikl**  
Class  
→ BusinessBase

## DL - Firma.EF.dll

**ArtiklDalProvider**  
Class  
→ AbstractDALProvider<Ar...  
Methods  
ArtiklDalProvider  
Exists  
Fetch  
FetchAll  
FetchAtPosition  
FetchLookup  
ItemCount

artikelBll

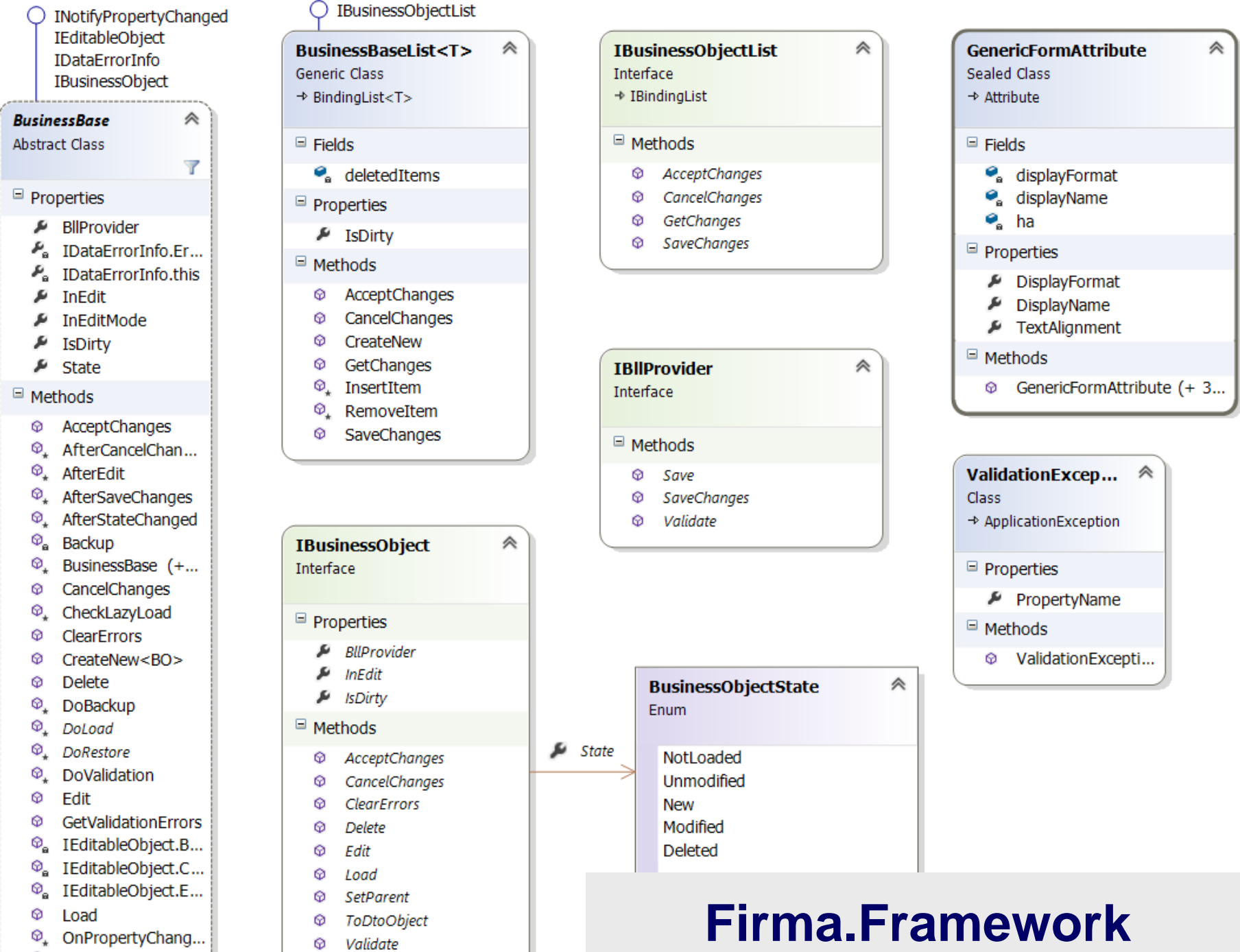
dal



# Firma.Framework

## ❑ Skup temeljenih razreda i sučelja za izradu poslovnih objekata

- *IBllProvider* – sučelje objekta iz poslovnog sloja zaduženog za rad s podacima (sučelje sadrži postupke za snimanje i validaciju)
- *IBusinessObject* – sučelje poslovnog objekta
- *BusinessBase* – apstraktni razred s osnovnom implementacijom poslovnog objekta
- *IBusinessObjectList* – sučelje liste poslovnih objekata (postupci za dohvat promjena i otkazivanje novo dodanih u listu)
- *BusinessBaseList* – razred s osnovnom implementacijom liste poslovnih objekata
- *BusinessObjectState* – enumeracija s mogućim stanjima objekta
- *GenericFormAtributte* – vlastiti atributi koji će se dodavati poslovnim objektima u svrhu automatskog generiranja formi
- *ValidationException* – vlastiti razred za iznimke



# Firma.Framework

# Poslovni objekt

## ❑ Implementacija poslovnog entiteta

- Sadrži neko od stanja poslovnog entiteta
- Implementira postupke za provjeru i promjenu stanja, validaciju (u kombinaciji s BLL *providerom*), pohranu, osvježavanje, itd.

## ❑ Sučelje poslovnog objekta *IBusinessObject*

- Svojstva stanja poslovnog objekta (*IsDirty*, *State*, *InEdit*)
- Veza prema BLL objektu za dohvat i spremanje (tipa *IBllProvider*)
- Postupci za početak ažuriranja ili brisanja (*Edit*, *Delete*)
- Postupci za otkazivanje promjena (*CancelChanges*) i označavanje objekta spremljenim (*AcceptChanges*)
- Postupci za validaciju (*ClearErrors*, *Validate*)
- Postupak *SetParent* za postavljanje roditelja (tipa *IBusinessObjectList*)
  - Kada je objekt dio neke liste
- Postupci za učitavanje podataka iz DTO objekta (*Load*) i obrnuto (*ToDtoObject*)

# Bazni razred za poslovni objekt

## ❑ ***BusinessBase*** – apstraktni razred koji implementira sučelja potrebna za povezivanje podataka na formi

- *IBusinessObject* (vidi prethodnu foliju)
- *INotifyPropertyChanged*
  - Omogućava slanje obavijesti o promjeni povezanog objekta
- *IEditableObject*
  - Omogućava stvaranje kopije podataka prije početka ažuriranja i vraćanja u slučaju odustajanja
- *IDataErrorInfo*
  - Standardno sučelje za validaciju
- Apstraktni postupci
  - *Validate(string propertyName)*
  - *DoLoad(IDTOObject dtoObject)*
  - *ToDtoObject()*
- Svojstvo *InEditMode* za provjeru nalazi li se objekt u stanju koje dozvoljava ažuriranje
- Ostali *protected* postupci koji se koriste kod promjene stanja objekta

# Sučelje liste poslovnih objekata

## ❑ ***IBusinessObjectList*** - sučelje za dodatnu funkcionalnost u odnosu na ***BindingList***

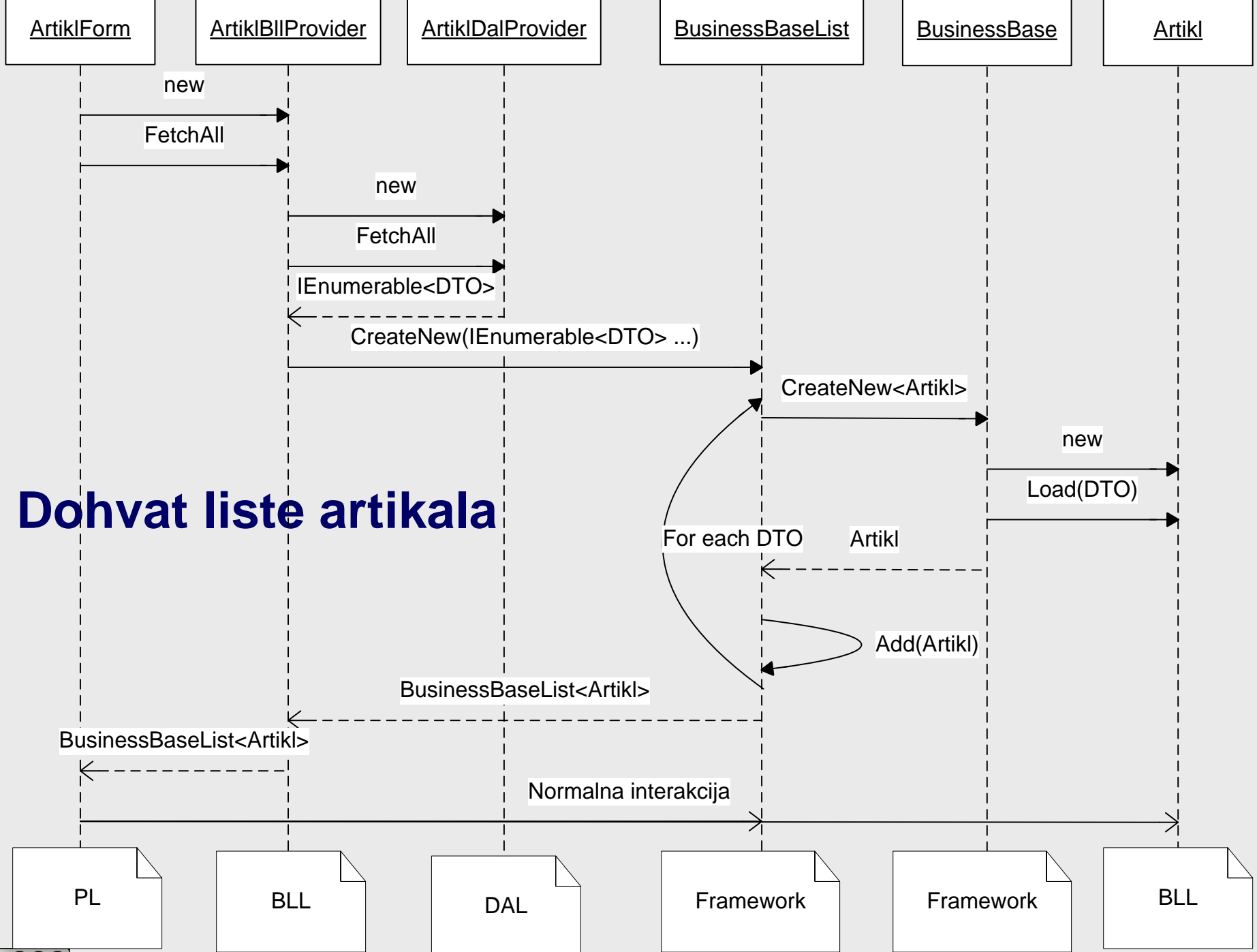
- ***GetChanges ()***
  - lista izmijenjenih poslovnih objekata
- ***SaveChanges(IBllProvider bll)***
  - promijenjene elemente šalje na snimanje konkretnom *BllProvideru*
- ***CancelChanges()***
  - otkazivanje svih promjena u listi
- ***AcceptChanges()***
  - prihvrat promjena u listi
  - uobičajeno se poziva nakon što je snimanje podatka uspješno

# Liste poslovnih objekata

## ❑ ***BusinessBaseList<T>***

- Generički razred za listu poslovnih objekata
- Nasljeđuje *BindingList<T>* i prekriva postupke
  - *InsertItem*
    - Prije stavljanja u listu stavlja sebe kao roditelja poslovnog objekta
  - *RemoveItem*
    - Čuva obrisani element u posebnoj listi
- Implementira *GetChanges* iz *IBusinessObjectList*
- Vlastiti postupci
  - *IsDirty*
    - provjera da li se lista mijenjala ili neki njen element
  - *CreateNew<T>*
    - statički postupak koji stvara novu listu na osnovu kolekcije DTO objekata (primjer uskoro)
- Kad generička implementacija nije dovoljna naslijediti *BusinessBaseList<T>*
  - *Npr. DokumentList : BusinessBaseList<Dokument>*

# Dohvat liste artikala



# Zahtjev poslovnom sloju

## ❑ Primjer: FirmaWin \ Firma \ Forms \ ArtikelForm

```
// referenca na BLL sloj
private ArtikelBllProvider artikelBll = new ArtikelBllProvider();
public LoadData() {
    var data = artikelBll.FetchAll(); //naknadno FetchLazy
    artikelBindingSource.DataSource = data;
}
```

## ❑ Primjer: FirmaWin \ Firma.BLL \ ArtikelBllProvider.cs

- Od DAL-a se uzima kolekcija DTO objekata (FetchAll)
- Statičkim postupkom CreateNew stvara se lista konkretnih poslovnih objekata (u ovom primjeru BusinessBaseList<Artikl>)

```
public class ArtikelBllProvider : IBllProvider {
    private ArtikelDalProvider dal = new ArtikelDalProvider();

    public BusinessBaseList<Artikl> FetchAll() {
        var dalRecord = dal.FetchAll();
        return BusinessBaseList.CreateNew<Artikl>(dalRecord);
    }
}
```




# Zahtjev podatkovnom sloju

## ❑ Primjer: FirmaWin \ Firma.EF \ ArtiklDalProvider.cs

- DAL isporuči kolekciju objekata tipa Firma.EF.Artikl koji služi kao DTO objekt

```
public class ArtiklDalProvider : AbstractDALProvider<Artikl> {  
    public override List<Artikl> FetchAll() {  
        using (var ctx = new FirmaEntities()) {  
            return ctx.Artikl.AsNoTracking().ToList();  
        }  
        ...  
    }  
}
```

- Svi entiteti iz EF modela prošireni tako da implementiraju sučelje IDTOObject
  - Prazno sučelje kojim se želi objediniti različite tipove DTO-a
  - Primjer:  FirmaWin \ Firma.EF \ DTO \ Artikl.cs

```
namespace Firma.EF {  
    public partial class Artikl : IDTOObject { }  
}
```

# Stvaranje liste poslovnih objekata

## ❑ Primjer: FirmaWin \ Firma.Framework \ BusinessBaseList.cs

- Lista se stvara na osnovu kolekcije DTO objekata
- Kolekciju isporuči BLL provider prethodnim pozivom DAL providera
- Za pojedinačni objekt koristi se statički postupak *CreateNew* u razredu *BusinessBase*

```
public class BusinessBaseList<T> : BindingList<T>, IBusinessObjectList
    where T : IBusinessObject, new() {
    ...
    public static BusinessBaseList<T> CreateNew
        (IEnumerable<IDTOObject> items) {
        BusinessBaseList<T> list = new BusinessBaseList<T>();
        foreach (var dto in items)
        {
            T businessObject = BusinessBase.CreateNew<T>(dto);
            list.Add(businessObject);
        }
        return list;
    }
}
```

# Stvaranje poslovnog objekta iz DTO objekta

## ❑ Primjer: FirmaWin \ Firma.Framework \ BusinessBase.cs

- Tip podatka mora implementirati sučelje `IBusinessObject` i mora imati prazni konstruktor

```
public abstract class BusinessBase : INotifyPropertyChanged,
    IEditableObject, IDataErrorInfo, IBusinessObject {
    public static BO CreateNew<BO>(IDTOObject dto)
        where BO : IBusinessObject, new() {
        BO item = new BO();
        item.Load(dto);
        return item;
    }

    public void Load(IDTOObject dtoObject) {
        DoLoad(dtoObject);
        SetState(BusinessObjectState.Unmodified);
    }
}
```

- Svojstva se postavljaju na osnovu vrijednosti DTO objekta u virtualnom postupku *DoLoad*

# Učitavanje podataka iz DTO objekta

❑ **Primjer:**  **FirmaWin \ Firma.BLL\ BusinessEntities \ DALSpecific \ Artikl.cs**

- Parcijalni razred
- konkretno učitavanje (postupak *DoLoad*)
- Izoliran dio koji ovisi o konkretnom tipu DTO objekta

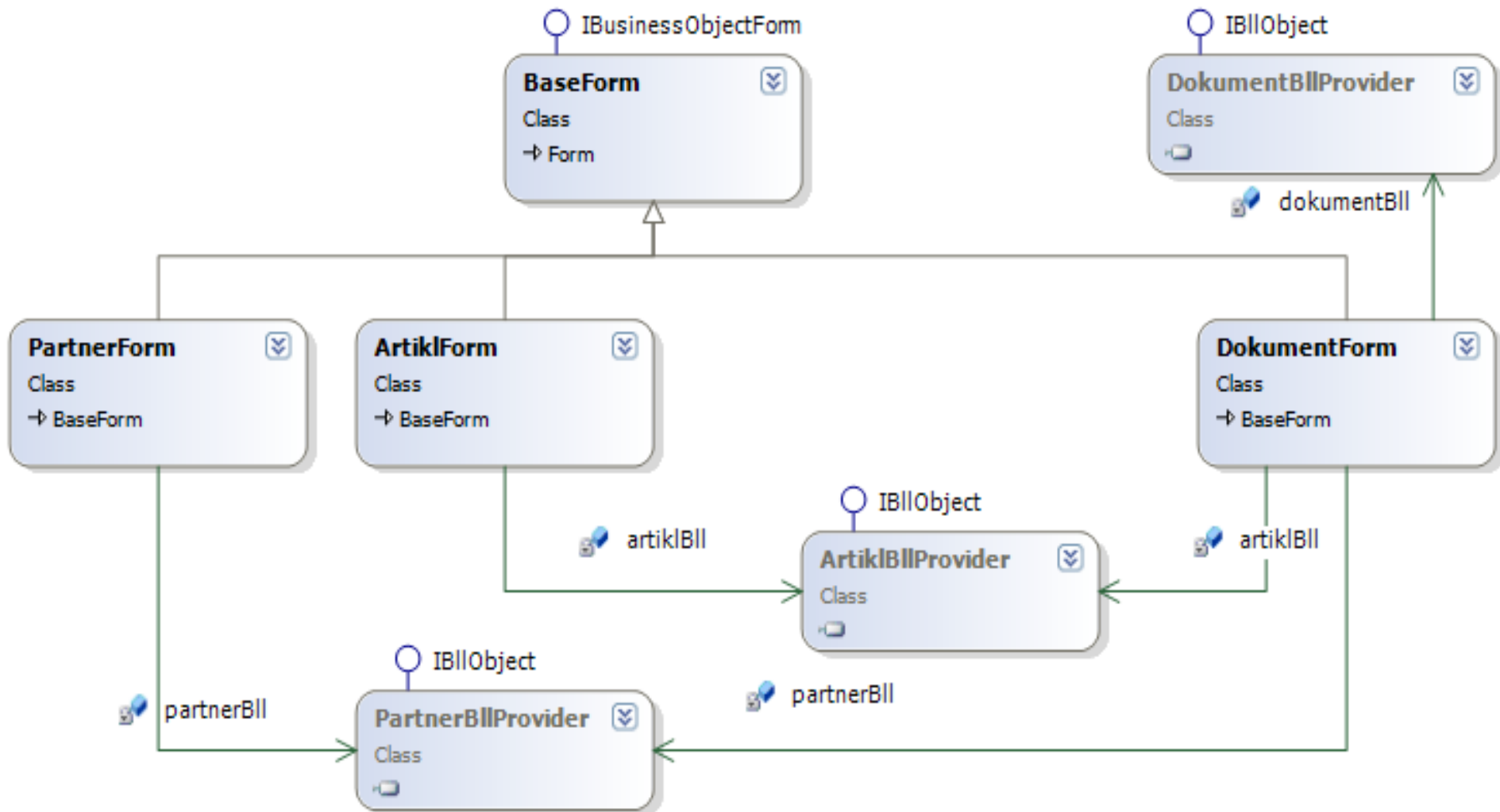
```
using DAL = Firma.EF;  
public partial class Artikl {  
    protected override void DoLoad(IDTOObject dtoObject)  
    {  
        DAL.Artikl artikl = (DAL.Artikl)dtoObject;  
        this.sifArtikla = artikl.SifArtikla;  
        this.nazArtikla = artikl.NazArtikla ?? "";  
        ...  
        this.slikaArtikla = artikl.SlikaArtikla;  
    }  
}
```

# Prezentacijski sloj

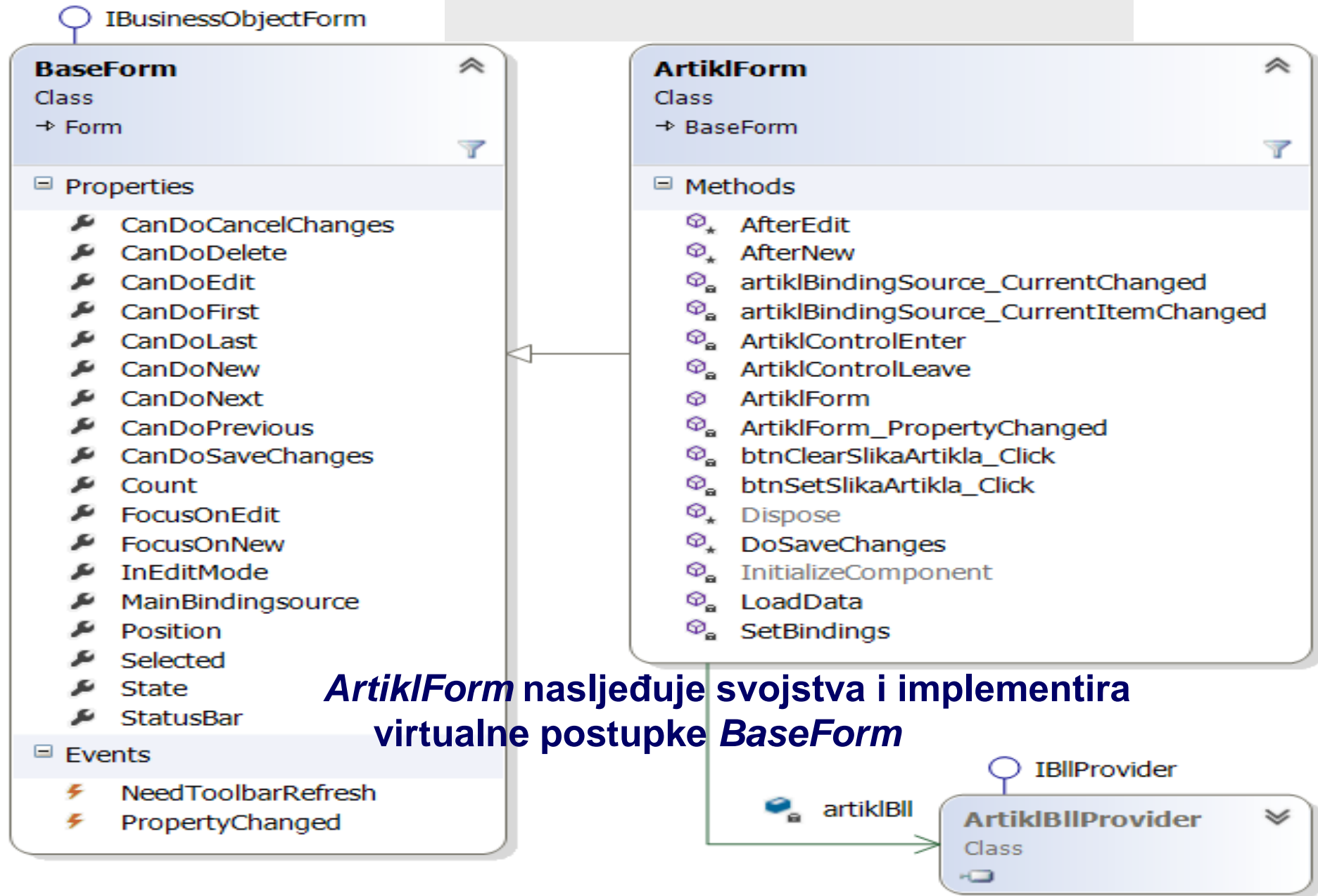
---

# Prezentacijski sloj

## ❑ Razredi sloja i veza na poslovni sloj



# Zaslonska maska



**ArtiklForm** nasljeđuje svojstva i implementira virtualne postupke **BaseForm**

# Povezivanje na objekte

## ❑ Primjer: FirmaWin \ Firma \ Forms \ ArtikelForm

- *Data – Add New Data Source – Object ... Firma.BLL.Artikl*

## ❑ Povezivanje u dizajnu

- *artiklBindingSource.DataSource = Firma.BLL.Artikl*
- *nazArtiklaTextBox.Text* povezan na "*artiklBindingSource – NazArtikla*"

## ❑ Povezivanje programski - radi formata prikaza

```
cijArtiklaTextBox.DataBindings.Add(  
    new Binding(  
        "Text", artiklBindingSource, "CijArtikla",  
        true,    // formatiranje omogućeno  
        DataSourceUpdateMode.OnPropertyChanged, // ažuriranje izvora  
        string.Empty, "N2")); // vrijednost za null, dvije decimale
```



# Rukovanje povezanim podacima

## ❑ Primjer: FirmaWin \ Firma \ Core \ BaseForm

- osnovna forma ima referencu na *BindingSource* kako bi rukovala podacima, što izvedena forma izloži u popisu svojstava u dizajnu
- (o atributima više naknadno)

```
private BindingSource mainBindingSource;  
  
[Browsable(true), Category("Data")]  
public BindingSource MainBindingsource  
{  
    get { return mainBindingSource; }  
    set { mainBindingSource = value; }  
}
```

```
public object Selected {  
    get {  
        if (mainBindingSource != null)  
            return mainBindingSource.Current;  
  
        return null;  
    }  
}
```

# Navigacija

## ❑ Primjer: FirmaWin \ Firma \ Core \ BaseForm

```
public bool CanDoNext
{
    get { return mainBindingSource != null && !InEditMode
    && mainBindingSource.Position < mainBindingSource.Count - 1; }
}
```

## ❑ BaseForm.BaseForm\_KeyDown

## ❑ ... case Keys.PageDown: Next();

```
public virtual void Next()
{
    // Odlazak na drugi zapis moguć je ako ne traje unos/izmjena
    if (mainBindingSource != null && !InEditMode)
    {
        mainBindingSource.MoveNext();
        OnNeedToolBarRefresh();
    }
}
```

# Prijenos kontrole nad podacima

## ❑ Primjer: FirmaWin \ Firma \ Core \ FormToolbar

```
// Forma s kojom komunicira toolbar.  
// Property se postavlja u dizajnu forme.  
private IBusinessObjectForm f;
```

```
public void RefreshToolbar()  
{  
    this.Enabled = f != null;  
    ..  
    btnNext.Enabled = f != null ? f.CanDoNext : false;
```

```
private void btnNext_Click(object sender, EventArgs e)  
{  
    if (f != null)  
        f.Next();  
}
```

# Prikaz informacije o aktualnom zapisu

## ❑ Postavljanje naslova forme kad se promijeni aktualni artikl ...

```
private void artiklBindingSource_CurrentItemChanged(  
    object sender, EventArgs e)  
{  
    this.Text = "Artikl";  
    if (artiklBindingSource.Current != null)  
    {  
        this.Text = "Artikl: "  
            + ((Artikl)artiklBindingSource.Current).ToString();  
    }  
}
```

## ❑ ... izazove promjenu statusne trake

```
private void BaseForm_TextChanged(object sender, EventArgs e)  
{  
    ...  
    StatusBar.NazivModula = this.Text;  
}
```

# Postavljanje reference na poslovni sloj

## ❑ Pridruživanje BLL objekta poslovnemu objektu

```
private void artiklBindingSource_CurrentChanged(...)  
{  
    Artikl a = artiklBindingSource.Current as Artikl;  
  
    // objekt koji još ne zna gdje mu je BLL objekt... ažuriraj.  
    if (a != null && a.BllProvider == null)  
    {  
        a.BllProvider = artiklBll;  
    }  
}
```

# Nalog poslovnom sloju za spremanje i brisanje

## ❑ Primjer: FirmaWin \ Firma \ Forms \ ArtiklForm

```
// Spremanje svih izmjena
protected override void DoSaveChanges() {
    IBusinessObjectList list =
        (IBusinessObjectList)artiklBindingSource.DataSource;
    list.SaveChanges(artiklB11);
}
```

## ❑ Brisanje naslijeđeno iz BaseForm (po potrebi se može promijeniti)

### ■ Primjer: FirmaWin \ Firma \ Forms \ ArtiklForm

```
protected virtual void DoDelete() {
    mainBindingSource.RemoveCurrent();
    DoSaveChanges(); //metoda iz naslijeđene forme
}
```

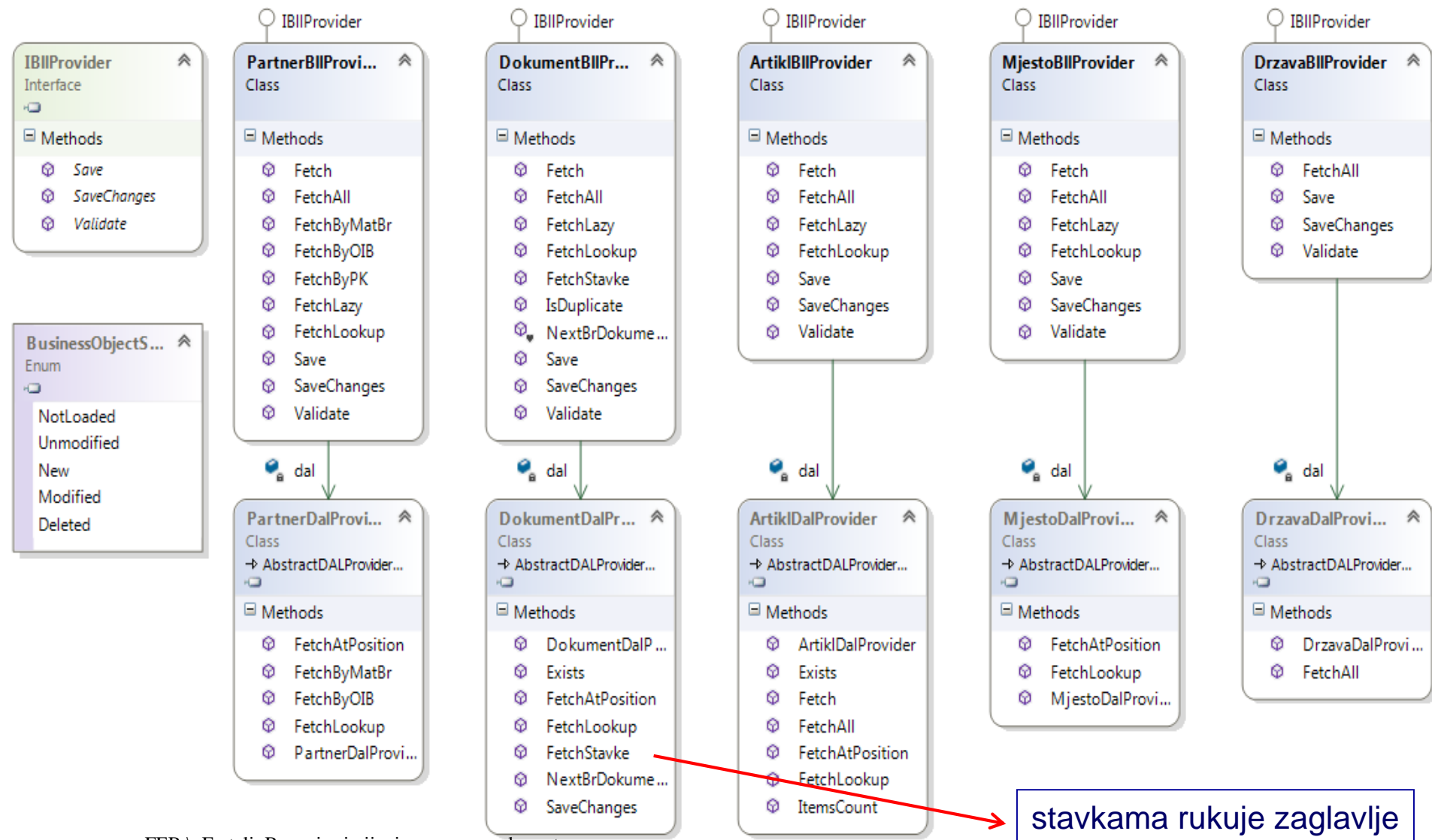
```
// ostale metode
```

# Poslovni sloj

---

# Poslovni sloj

- Prosljeđuje objekte (poslovne entitete) između PL i DL
- Implementira provjeru (validaciju) poslovnih pravila (business rules)

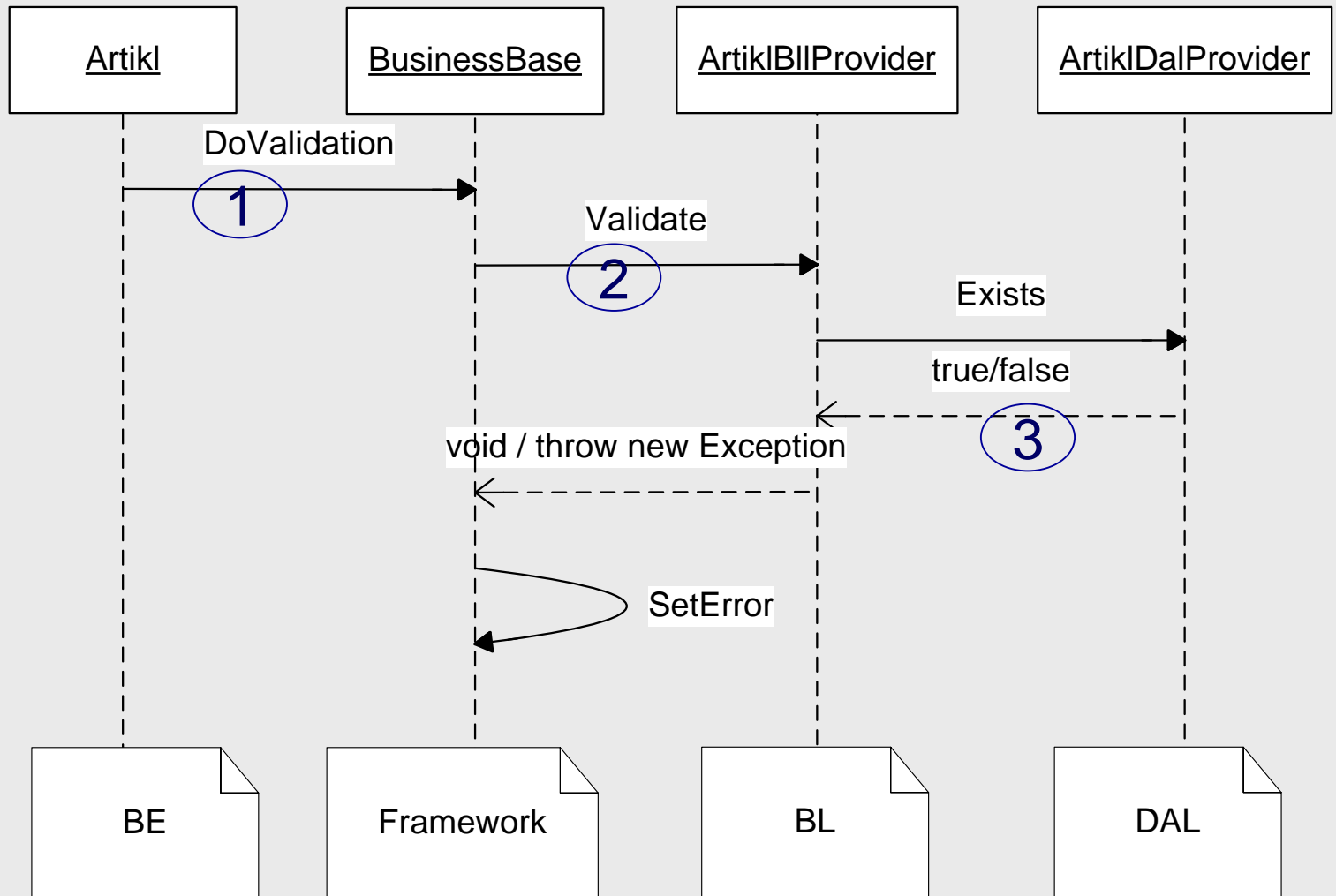




# Provjera ispravnosti

## □ Primjer: FirmaWin – Artikl (edit : *Artikl.property.set*)

- validacija svojstva poslovnog objekta (šifra, naziv)



# 1: Pokretanje validacije

## ❑ objekt **Artikl** - promjenom svojstva putem forme (ili programski)

```
public int? SifArtikla
{
    get {... return sifArtikla; }
    set
    {
        ...
        if (InEditMode)
        {
            sifArtikla = value;
            PropertyChanged(
                "SifArtikla");
        }
    }
    ...
}
```

```
protected void PropertyChanged(
    string propertyName)
{
    isDirty = true;
    // Osvježavanje data-binding-om
    OnPropertyChanged(propertyName);

    // Validacija
    DoValidation(propertyName);
}
```

## ❑ **BusinessBase**

```
protected void DoValidation(string propertyName)
{
    ...
    BlProvider.Validate(this, propertyName);
}
```

## 2: Validacija pojedinog svojstva

❏ **Primjer:**  **FirmaWin \ Firma.BLL \ ArtiklBllProvider.cs**

```
public void Validate(object businessObject, string propertyName)
{
    Artikl target = (Artikl)businessObject;
    switch (propertyName)
    {
        case "SifArtikla":
            {
                if (!target.SifArtikla.HasValue)
                    throw new Exception("Šifra artikla je obavezno polje!");

                if (target.State == BusinessObjectState.New)
                {
                    bool sifraPostoji = dal.Exists(target.SifArtikla.Value);
                    if (sifraPostoji)
                        throw new Exception(string.Format(
                            "Artikl {0} već postoji.",
                            target.SifArtikla.Value));
                }
            }
        }
    }
```

### 3: Provjera jedinstvenosti

❏ **Primjer:**  **FirmaWin \ Firma.DAL \ ArtiklDalProvider .cs**

- Postupak Exists vraća postoji li već artikl s tom šifrom

```
public bool Exists(int sifArtikla)
{
    using (FirmaEntities ctx = new FirmaEntities())
    {
        int count = ctx.Artikl.
            Where(a => a.SifArtikla == sifArtikla).
            Count();
        return count > 0;
    }
    ...
}
```

# Stanja objekata

```
public enum BusinessObjectState
{
    NotLoaded, Unmodified,
    New, Modified,
    Deleted
}
```

```
private BusinessObjectState state = BusinessObjectState.New;
```

```
[Browsable(false)]
public BusinessObjectState State
{
    get { return state; }
    protected set
    {
        state = value;
        AfterStateChanged();
        OnNeedToolBarRefresh();
        OnPropertyChanged("InEditMode");
        OnPropertyChanged("State");
    }
}
```

# Poništavanje promjena

## ❏ Primjer: FirmaWin \ Firma.Framework \ BusinessBase.cs

```
public void CancelChanges() {  
    if (State != BusinessObjectState.New) {  
        Restore();  
        SetState(BusinessObjectState.Unmodified);  
    }  
    else {  
        if (parent != null) {  
            parent.Remove(this);  
            parent = null;  
        }  
    }  
    AfterCancelChanges();  
}
```

```
void Restore() {  
    if (backupObject != null) {  
        ...  
        DoRestore(); // Pozovi konkretni restore (virtualna metoda)  
        ...  
        // Refresh GUI-a dojavom da su se property-i promijenili  
        OnPropertyChanged(string.Empty);  
    }  
}
```

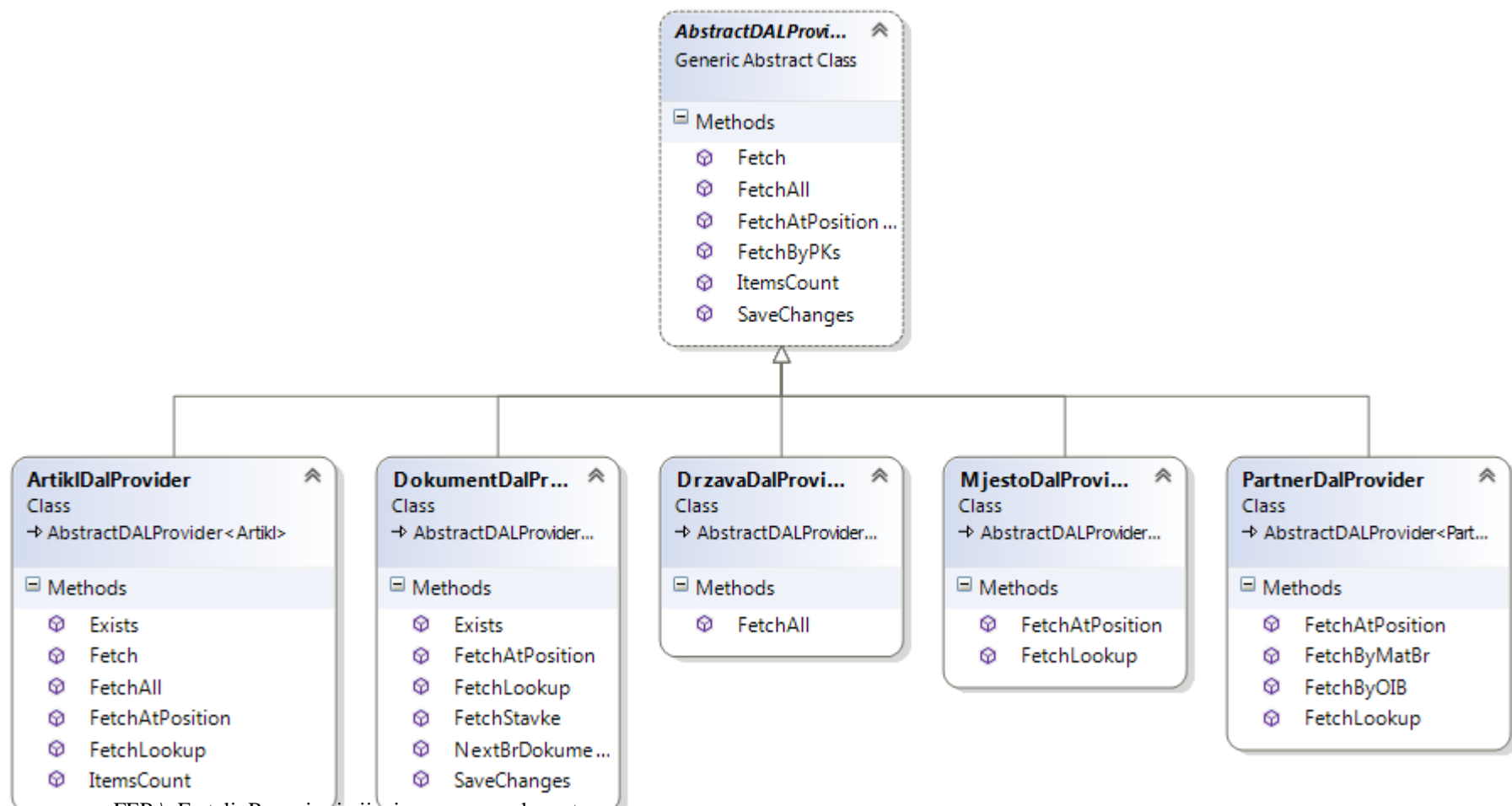
# Podatkovni sloj

---

# Podatkovni sloj

## ❑ Primjer: FirmaWin \ Firma.EF

- Razredi za rukovanje podacima
- Uobičajeni postupci izdvojeni u apstraktni razred, da se ne kopira kod
- Izgrađen nad EF modelom slično kao u RPPP06





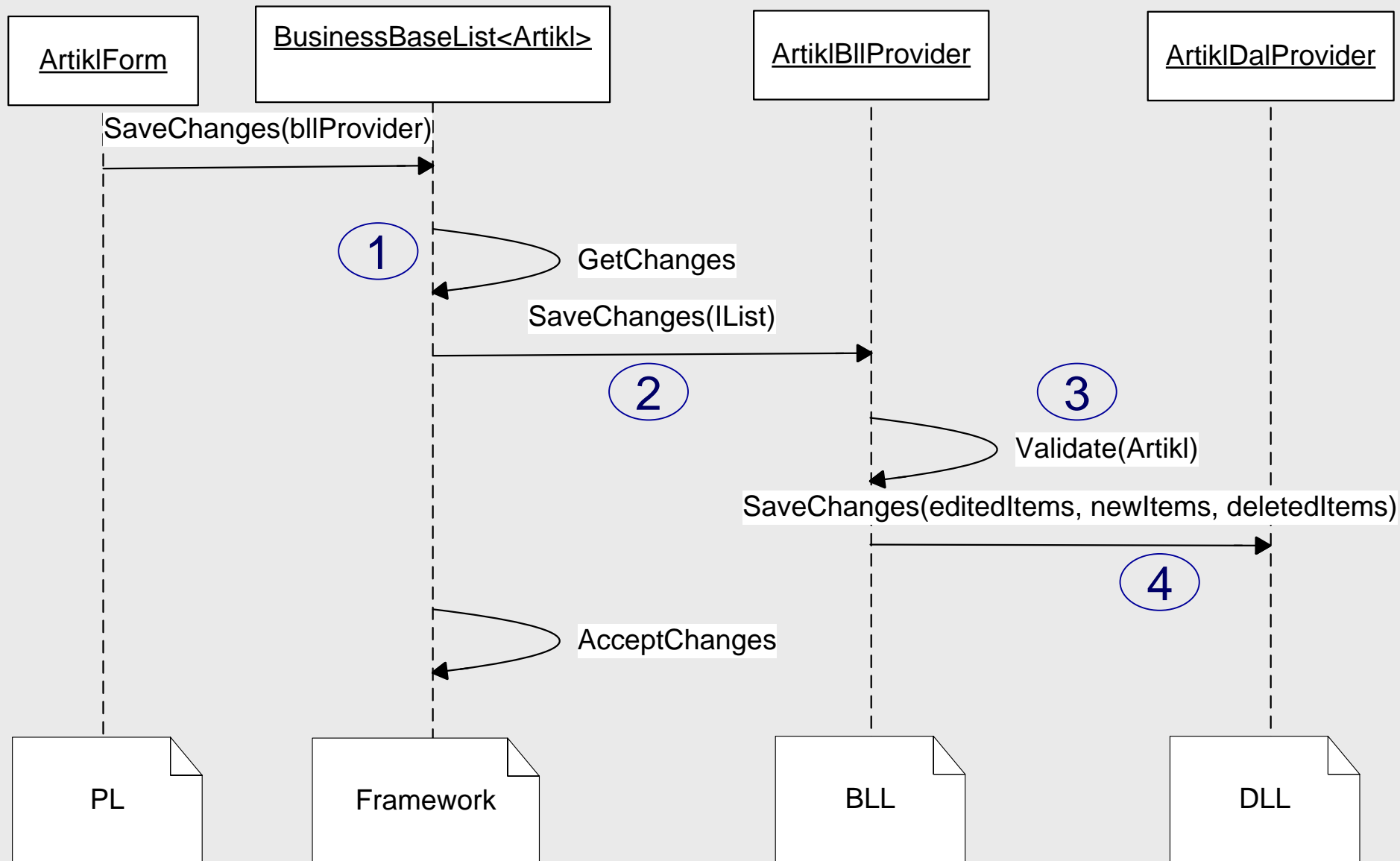
# Uobičajeni postupci dohvata

## ❑ Primjer: FirmaWin \ Firma.EF \ AbstractDALProvider.cs

- Postupak `Set<T>()` vraća odgovarajući *DbSet* za tip entiteta *T*
- Postupci virtualni, pa izvedeni DAL provider može imati vlastite

```
public virtual int ItemsCount() {  
    using (FirmaEntities ctx = new FirmaEntities()) {  
        return ctx.Set<T>().Count();  
    }  
}  
  
public virtual List<T> FetchAll() {  
    using (FirmaEntities ctx = new FirmaEntities()) {  
        var list = ctx.Set<T>().AsNoTracking().ToList();  
        return list;  
    }  
}  
  
public virtual T FetchByPKs(params object[] keyValues) {  
    using (FirmaEntities ctx = new FirmaEntities()) {  
        T item = ctx.Set<T>().Find(keyValues);  
        return item;  
    }  
}  
...
```

# Spremanje promjena u bazu podataka



# Nalog poslovnem sloju za spremanje i brisanje

❑ Primjer:  FirmaWin – ArtikelForm # pokazano uz prezentacijski sloj

```
// Spremanje svih izmjena
protected override void DoSaveChanges()
{
    var list = (IBusinessObjectList)artiklBindingSource.DataSource;
    list.SaveChanges(artiklBll);
}
```

❑ Primjer:  FirmaWin – BaseForm

```
protected virtual void DoDelete()
{
    // Uklanjanje poslovnog objekta iz liste dohvaćenih objekata
    artiklBindingSource.RemoveCurrent();
    DoSaveChanges();
}
```

# 1: Stvaranje liste izmijenjenih objekata

## ❑ Primjer: FirmaWin \ Firma.Framework \ BusinessBaseList .cs

- Označeni za brisanje + novi + „prljavi” objekti
- One koji još nisu učitani, ignoriramo

```
public IList GetChanges()  
{  
    List<T> changes = new List<T>();  
    foreach (T item in deletedItems)  
    {  
        changes.Add(item);  
    }  
    foreach (T item in this.  
        Where(i => i.State != BusinessObjectState.NotLoaded)) {  
  
        if (item.IsDirty || item.State == BusinessObjectState.New)  
            changes.Add(item);  
    }  
  
    return changes;  
}
```

## 2: Spremanje izmjena

### ❏ Primjer: FirmaWin \ Firma.BLL \ ArtiklBllProvider .cs

```
public void SaveChanges(IList changedItems){
    var newItem = new List<DAL.Artikl>();
    ...
    foreach (Artikl item in changedItems){
        if (item.IsDirty){
            switch(item.State){
                case BusinessObjectState.New:
                    item.Validate();
                    if (!string.IsNullOrEmpty(((IDataErrorInfo)item).Error))
                        throw new Exception(((IDataErrorInfo)item).Error);
                    newItem.Add((DAL.Artikl) item.ToDtoObject());
                    break;
                case BusinessObjectState.Modified:
                    item.Validate();
                    ...
                    editedItems.Add((DAL.Artikl) item.ToDtoObject());
                    break;
                case BusinessObjectState.Deleted:
                    deletedItems.Add((DAL.Artikl) item.ToDtoObject());
                    ...
            }
        }
    }
    // Proslijedi DAL sloju na spremanje u bazu
    dal.SaveChanges(newItem, editedItems, deletedItems);
}
```

### 3: Validacija poslovnog objekta

- ❑ Validacija svih svojstava poslovnog objekta zove *DoValidation* iz *BusinessBase...*

```
public override void Validate()
{
    DoValidation("SifArtikla");
    DoValidation("NazArtikla");
    DoValidation("JedMjere");
    DoValidation("CijArtikla");
    DoValidation("ZastUsluga");
    DoValidation("SlikaArtikla");
}
```

- ❑ ...koji proslijeđuje validaciju *BLL provideru*

```
public void Validate(object businessObject, string propertyName)
{
    Artikal target = (Artikal)businessObject;
    switch (propertyName)
    {
        case "SifArtikla":
```

## 4: Pohrana u bazu podataka

### ❑ Primjer: FirmaWin \ Firma.EF \ AbstractDALProvider.cs

- Primaju se kolekcije izmijenjenih, novih i objekata označenih za brisanje
- Primljeni EF objekti nastali su van konteksta, pa ih treba zakačiti / dodati u kontekst i postaviti odgovarajuće stanje

```
public virtual void SaveChanges(IEnumerable<T> changedItems,
    IEnumerable<T> newItem, IEnumerable<T> deletedItems) {
    using (FirmaEntities ctx = new FirmaEntities()) {
        foreach (var item in changedItems) {
            ctx.Set<T>().Attach(item);
            ctx.Entry(item).State = System.Data.EntityState.Modified;
        }
        foreach (var item in newItem) {
            ctx.Set<T>().Add(item);
        }
        foreach (var item in deletedItems) {
            ctx.Set<T>().Attach(item);
            ctx.Entry(item).State = System.Data.EntityState.Deleted;
        }
        ctx.SaveChanges();
    }
}
```

# Pojedinačno učitavanje podataka

## ❑ Primjer: FirmaWin \ Firma.Framework i Firma.BLL

- Forme iz primjera prikazuju pojedinačni zapis
- Svejedno dohvaćamo sve podatke zbog navigacije putem *BindingSource*
  - nepraktično/sporo zbog količine podataka kada BP nije lokalno ili blizu
- Za pojedinačnu obradu treba stvoriti „prazne” objekte
  - Konstruktor je *internal* da se ne može upotrijebiti van BLL-a
  - Prezentacijski sloj nije svjestan razlike
  - Objekt se nalazi stanju *NotLoaded*
  - Učitavanje se naknadno obavlja pomoću delegata tipa *Func<int, IDTOObject>* - postupak koji prima *int* i vraća *IDTOObject*

```
internal Artikal(Func<int, IDTOObject> lazyLoadFunction) :  
    base(lazyLoadFunction) { }  
  
...  
protected BusinessBase(Func<int, IDTOObject> loadFunction)  
{  
    state = BusinessObjectState.NotLoaded;  
    this.loadFunction = loadFunction;  
}
```



# Naknadno učitavanje

## ❑ Primjer: FirmaWin \ Firma.BLL \ BusinessEntities \ Artikl.cs

- Prilikom dohvata provjerava se je li objekt učitao ili ne

```
public int? SifArtikla {  
    get {  
        CheckLazyLoad() ;  
        return sifArtikla;  
    }  
}
```

## ❑ Primjer: FirmaWin \ Firma.Framework \ BusinessBase.cs

- Ako podatak nije učitao dolazi do poziva delegata i postupka *Load*
- Pozicija određena pozicijom u listi
  - Potencijalni problem kod dodavanja novih – pozicija u listi ne mora odgovarati poziciji u BP
  - Alternativa: Inicijalno učitati samo PK, a tek naknadno ostale podatke

```
protected void CheckLazyLoad() {  
    if (State == BusinessObjectState.NotLoaded) {  
        int i = parent.IndexOf(this); //pozicija u listi  
        var dtoObject = loadFunction(i);  
        Load(dtoObject);  
    }  
}
```

# Primjer za objekt s naknadnim učitavanjem

## ❑ Primjer: FirmaWin \ Firma.BLL \ ArtikelBllProvider.cs

- Stvorena lista s N (broj zapisa u konkretnoj tablici) elementa koji imaju delegat za naknadno učitavanje
  - Dohvat na osnovu pozicije u bazi

```
public BusinessBaseList<Artikl> FetchLazy() {  
    int count = dal.ItemsCount();  
    var list = new BusinessBaseList<Artikl>();  
    for (int i = 0; i < count; i++) {  
        Func<int, IDTOObject> lazyLoadFunction =  
            (position) => dal.FetchAtPosition(position);  
        Artikl artikl = new Artikl(lazyLoadFunction);  
        list.Add(artikl);  
    }  
    return list;  
}
```

- U prezentacijskom sloj potrebno samo promijeniti iz FetchAll u FetchLazy

# Dohvat n-tog artikla

## ❑ Primjer: FirmaWin \ Firma.EF \ ArtiklDalProvider.cs

- Kombiniranjem Linq postupaka Skip i Take
- Mora postojati OrderBy
  - Jednostavnosti radi - dohvat uvijek po šifri

```
public Artikal FetchAtPosition(int position)
{
    using (FirmaEntities ctx = new FirmaEntities())
    {
        return ctx.Artikal.AsNoTracking().
            OrderBy(a => a.SifArtikla).
            Skip(position).
            Take(1).
            FirstOrDefault();
    }
}
```

# Reference

## ❑ Rockford Lhotka, CSLA.NET Framework

- <http://www.lhotka.net/cslanet/>

## ❑ Microsoft Application Architecture Guide - Layered Application Guidelines

- <http://msdn.microsoft.com/en-us/library/ee658109.aspx>