FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

Seminar iz kolegija

**Objektno oblikovanje**

za **Laboratorij Profila**

**Sustav za iznajmljivanje**

**Case study**

Vinko Kodžoman

0036470625

Uvod

Cilj seminara je prikazati i objasniti način provedbe OR mapiranja pomoću Nhibernatea, koji služi za spremanje podataka u bazu podataka. U nastavku sljedi opis i objašnjenje realizacije mapiranja.

Design storry

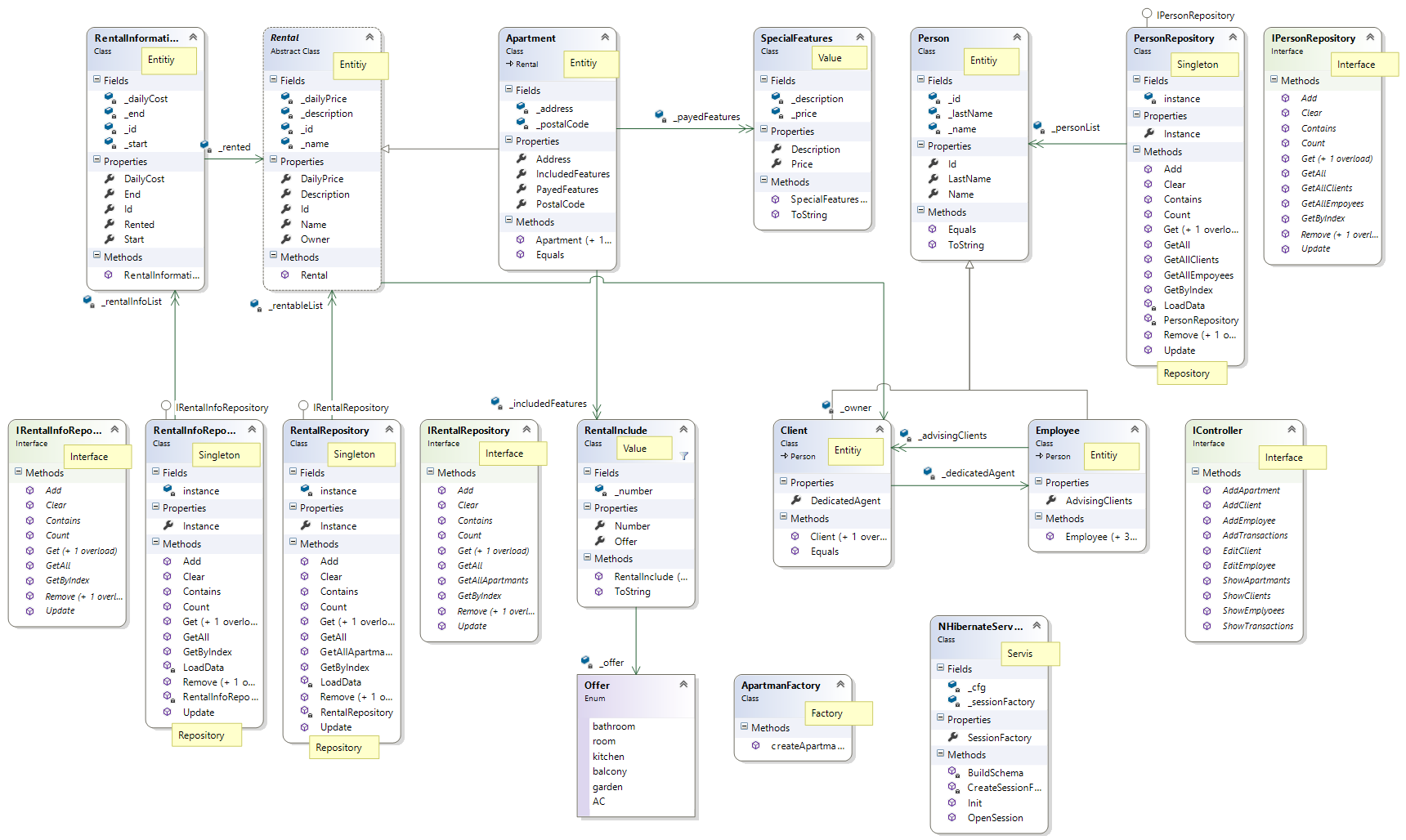
Agencija za vođenje apartmana omogućuje reklamiranje apartmana za svoje klijente i rezerviranje istih za kupce. Agencija uzima proviziju za svaku uspješnu transakciju – najam apartmana. Jedine vrste nekretnina sa kojima agencija radi su apartmani, ali sustav mora dopuštati jednostavno proširenje sa novim stvarima (auti, bicikli, skuteri, hosteli, ...) u budućnosti. Svi apartmani u sustavu agencije su aktivni – mogu se iznamljivati, i u trenutku kada klijent agencije odlući prestati iznamljivati svoj apartman, oni se brišu iz sustava zajedno sa klijentom. Svaki klijent se uparuje sa jednim agentom agencije, koji je zadužen za klijenta kao i za njegove apartmane.

Za svaki apartman klijent mora dati sve potrebne informacije (broj soba, broj kupaonica, iznos po noći, lokacija, klima uređaj, ...), ukoliko su informacije nepotpune apartman se ne može pohraniti unutar sustava. Uz same karakteristke apartmana bitno je i da su navedene sve ponuđene usluge koje klijent nudi korisniku (kabelska televizija, internet, kućni telefon, ...). Svaki apartman mora imati informacije o prošlim korisnicima (ljudi koju su iznamljivali apartmani u prošlosti), kao što su podaci o osobi, iznos plaćanja i vrijeme boravka u apartmanu.

Za svaki apartman je naveden vlasnik kao i njegovi vlastiti podaci (ime, prezime). U slućaju da vlasnik više ne posluje sa agencijom treba izbrisati sve njegove apartmane unutar sustava.

Da bi dijelatnik agencije mogao pristupiti sustavu mora se prijaviti u njega sa svojim korisničkim imenom i lozinkom. Nakon ulaska u sustav prikazuju mu se sve ponude, vlasnici i povijest svih transakcija – rezervacija (prošlih, ali i trenutnih). Djelatnik agencije može saznati više o vlasniku kao i o apartmanu, odabirom njih unutar sustava. Djelatniku sustava (agentu) je omogućeno pretraživanje apartmana po njihovim imenima.

Sustav mora voditi evidenciju koji agenti agencije su zadužen za koje vlasnike. Sustav u niti jednom trenu ne smije imati informacije od ljudi koji iznamljuju apartmane i trenutno se nalaze u njima. Nakon što korisnik plati rezervaciju; bilježi se cijena rezervacije po danu, od kada do kada je rezerviran i koji apartman je rezerviran.

Class diagram

Agregate

Slika 1 Dijagram razreda sustava

Baza podataka

Kako bi aplikacija imali perzistenciju podataka korištena je baza podataka SQLite. Za pregled same baze sam koristio SQLite Expert Profesional kako bih omogućio pisanje SQL upita i gledanje na bazu izvan aplikacije. Stvaranje baze i relacije je napravljeno preko Fluent NHiberneta unutar same aplikacije (build schema). Baza podatak se zove „MyDatabase.sqlite“ i stvara se nakon prevođenja izvornog koda u izvršni (Microsoft Visual Studio).

OR Mapiranje

OR mapiranje je realizirano pomoću Fluent Nhibernate. Glavna prednost NHibernate je produktivnost. Umjesto pisanja SQL class library coda za rad sa bazom podataka NHibernate sam stvara relacije, bazu i omogućava CRUD (create, read, update and delete) operacije nad bazom sa razredima iz modela domene pomoću sessiona. Fluent NHibernate omogućava pisanje funkcija za mapiranje u samom kodu (strong typed) što uvelike olakšava pisanje mapiranje kao što i smanjuje greške programera, tj. greške su vidljive prije run-tima. NHibernate nudi tri naćina mapiranja objektnog modela u relacijski, u mojoj aplikaciji realizrano je preko per subclass mapiranje, tj. svaka klasa podklasa je relacija a veze su realizrane preko stranih ključa.

**Podešavanje i mapiranje**

Za podesiti okruženje treba dohvatiti preko NuGet packet managera (unutar MS Visual Studia) SQLite.Core i FluentNHibernat. Nakon dohvata potrebnih paketa treba dodati i konfiguracijski file za NHibernate „hibernate.cfg.xml“ (Slika 2). Datoteka se nalazi unutar Controller projekta zajedno sa repositorijma.

<?xml version="1.0" encoding="utf-8" ?>

<hibernate-configuration xmlns="urn:nhibernate-configuration-2.2">

<session-factory name="NHibernate.Test">

<property name="connection.driver\_class">NHibernate.Driver.SQLite20Driver</property>

<property name="connection.connection\_string">Data Source=nhibernate.db;Version=3</property>

<property name="dialect">NHibernate.Dialect.SQLiteDialect</property>

<property name="query.substitutions">true=1;false=0</property>

<property name="show\_sql">true</property>

<mapping assembly="Rental.DomainModel"/>

</session-factory>

</hibernate-configuration>

**Slika 2** Konfiguracijska datoteka za korištenje Fluent NHibernate sa SQLiteom

Za lakši rad sa Nhibernatom zadužen je servis NhibernateService koji nudi inacijalizaciju NHibernate i baze podataka, te dohvaćanje seesiona za razgovor s pazom (preko IsessionFactorya). NHibernate framework zajedno sa bazom podataka je konfiguriran na Slici 3.

private static ISessionFactory CreateSessionFactory()

{

return Fluently.Configure()

.Database(

SQLiteConfiguration.Standard

.UsingFile("MyDatabase.db")

)

.Mappings(m =>

{

m.FluentMappings.AddFromAssemblyOf<Rental>();

}

)

.ExposeConfiguration(BuildSchema)

.BuildSessionFactory();

}

**Slika 3** Inicijalizacija NHibernatea i SQLite baze podataka preko servisa

Kako bi rad sa podacima bio decouplan sa domenom realiziran je preko repository patterna. Svaki repostiroy ima sučelje (prošrirenje CRUD operacija na desetak bitnih funkcija) koje realizira. *View* prima repozitorij preko *Controller* projekta i zna da pirma repositorij koji implementira sučelje. Sustav ima tri repositorija RentalRepository (Rental je bazna klasa za sve što se može iznamjiti u sustavu, Slika 1), RentalInfoRepository (repository za rad sa transakcijama, tj. informacijama o svakom iznajmljivanju) i PersonRepository (za sve osobe u sustavu). Svi repositroy koriste NHibernateService servis za dohvat sjednice i razgovor sa bazom podataka. Repozitoriji su realzirani kao singleton razredi.

private void LoadData()

{

using (var session = NHibernateService.SessionFactory.OpenSession())

{

using (var transaction = session.BeginTransaction())

{

\_personList = session.CreateCriteria<Person>().List<Person>();

}

}

}

**Slika 4** Primjer dohvaćanja podatka preko NHibernateServis servisa (sve osobe)

Slika 5 prikazuje sučelje koje realizira PersonRepository, sučelje je jednako i za sve ostale repositorije. Sučelje omogućava da View zna operacije koje repositorij nudi bez direktne reference na Controller u kojem se nalaze repositoriji (unutar mape Repositories).

public interface IPersonRepository

{

int Count();

Person Get(int id);

Person Get(Person person);

Person GetByIndex(int index);

IList<Person> GetAll();

IList<Employee> GetAllEmpoyees();

IList<Client> GetAllClients();

void Add(Person person);

void Remove(int id);

void Remove(Person person);

void Clear();

bool Contains(Person person);

void Update(Person person);

}

**Slika 5** Primjer sučelja koji implementira PersonRepository

Cijelo mapiranje je odrađeno unutar jedne klase FluentNHibernaateMapper.cs koja se nalazi unutar DomainModel projekta u Mappings mapi. Kao što je i prethodno spomenuto, korištena je relation per subclass metoda preslikavanja objekata u bazu. Fluent NHibernate omogućava brzo mapiranje sa minimalnim pisanjem koda. Na slici 6 se nalazi primjer mapiranje osobe iz modela domene u bazu podataka. Person je baza klasa i ima Id (entity object), ime i prezime te se iz nje izvode dva specifična razreda za sustav; Client i Employee.

public class PersonMap : ClassMap<Person>

{

public PersonMap()

{

Id(x => x.Id);

Map(x => x.Name);

Map(x => x.LastName);

}

}

public class ClientMap : SubclassMap<Client>

{

public ClientMap()

{

References(x =>

x.DedicatedAgent).Cascade.

SaveUpdate().Nullable().Not.LazyLoad();

}

}

public class EmployeeMap : SubclassMap<Employee>

{

public EmployeeMap()

{

HasMany(x => x.AdvisingClients).Cascade.SaveUpdate().

Not.LazyLoad().ForeignKeyConstraintName("none");

}

}

**Slika 6** Mapiranje Person, Client i Empoyee razreda u bazu sa Nhibernatom

Za mapirati razred treba implementirati ClassMap<ime>, gdje je <ime> naziv razreda kojeg mapiramo, za izvedene klase umjesto ClassMap stavljamo SubclassMap<ime> (Slika 6). Klijent u sustavu se dodjeljuje osoba (zaposlenika) koja je zadužena za njega, to se realizira preko References (x => ...). Ne želimo da brisanjem klijenta se obriše i zaposlenik ne stavljamo Cascade.All() opciju već Cascade.SaveUpdate().Client može napočetu ne imati zaduženog zaposlenika dodaj se svojstvo .Nullable(). Kod dohvata klijenta u većinu slučajeva trebao i dohvatiti preko reference zposlenika pa forsiramo eager umjesto lazy loading. Jednog zaposlenika mogu imati više klijenata kao privatnog agent (HasMany svojstvo unutar employee razreda).

public class RentalMap : ClassMap<Rental>

{

public RentalMap()

{

Id(x => x.Id);

Map(x => x.Name);

Map(x => x.Description);

References(x => x.Owner).Cascade.All().Not.Nullable().Not.LazyLoad();

Map(x => x.DailyPrice);

}

}

public class ApartmanMap : SubclassMap<Apartment>

{

public ApartmanMap()

{

Map(x => x.Address);

Map(x => x.PostalCode);

HasMany(x => x.IncludedFeatures).Cascade.All().Not.LazyLoad();

HasMany(x => x.PayedFeatures).Cascade.All().Not.LazyLoad();

}

}

public class RentalIncludeMap : ClassMap<RentalInclude>

{

public RentalIncludeMap()

{

Id(x => x.Id);

Map(x => x.Offer);

Map(x => x.Number);

}

}

public class SpecialFeaturesMap : ClassMap<SpecialFeatures>

{

public SpecialFeaturesMap()

{

Id(x => x.Id);

Map(x => x.Price);

Map(x => x.Description);

}

}

**Slika 7** Mapiranje Rental i izvedenog agregata Apartmant

Za rental razred vrijedi sve isto kao i za osobu. Bazna klasa je Rental a izvedena Apartmant, vidljivo preko SubclassMap<ime>. Bilo koja stvar koja se iznamljuje u sustavu mora imati vlasnika (Client), veza izražena preko References. Apartman ima brojne uključene i posebne ponude koje su realizirane preko HasMany odnosa sa RentalInclude i SpecialInclude razreda. Sve promjene vezane uz Rental bazni razred, a i Apartmant te njegovih componenata se propagiraju preko Cascade.All().

public class RentalInformationMap : ClassMap<RentalInformation>

{

public RentalInformationMap()

{

Id(x => x.Id);

Map(x => x.Start);

Map(x => x.End);

Map(x => x.DailyCost);

References(x => x.Rented).Cascade.SaveUpdate().Not.LazyLoad();

}

}

**Slika 8** Mapiranje transkacija

Transakcija je realizirana razredom RentalInformation i mapiranje je prikazano na Slici 8. Referencira se na Rental razred za koji je transakcija vezana. Sva ostala svojstva su mapirana kao i u prošlim mapiranjima.