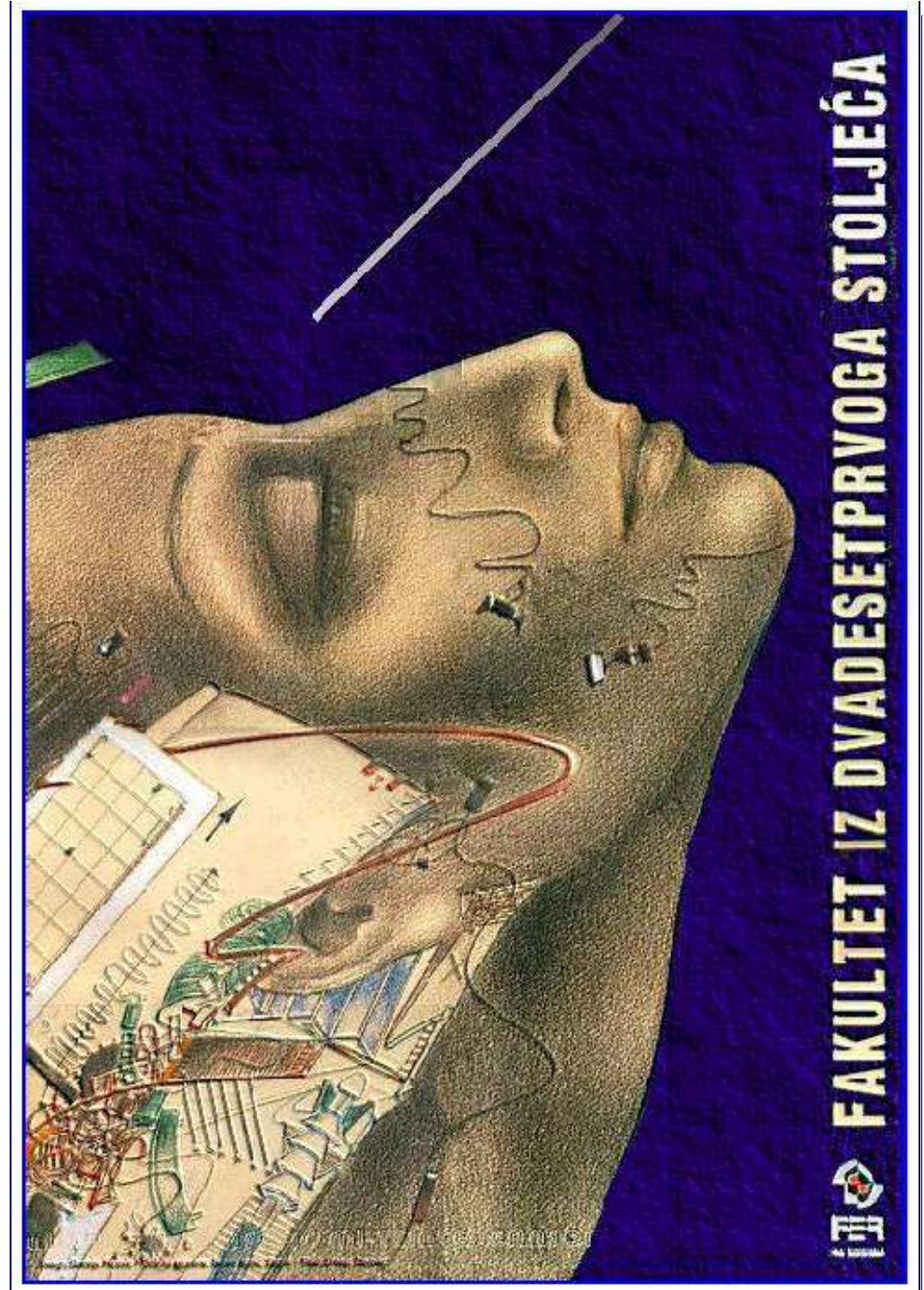


Napredni modeli i baze podataka

Predavanja
rujan 2008.

2. Objektno orijentirane baze podataka



Pregled

- Zašto objektno orijentirane baze podataka?
- Osnovni koncepti
- Objektno orijentirani sustavi za upravljanje bazama podataka
- ODMG standard
- Db4o sustav za upravljanje bazama podataka
- Objektno-relacijsko preslikavanje

Zašto objektno orijentirane baze podataka?

- Model relacijskih baza podataka bitno se razlikuje od objektnog modela aplikacija realiziranih objektno orijentiranim jezicima (Java, C#)
- **Objektno relacijska neusklađenost** (*object relational impedance mismatch*):
 - **Veze između entita:**

Objektni model: `kupac.getArtikli()`

 - Referencama na druge objekte

Relacijski model: tablice `kupac`, `kupac_artikl`, `artikl`

 - Primarnim i stranim ključevima
 - **Dohvat podataka:**

Objektni model: `kupac.getArtikli().get(0).getNaziv()`

 - Kretanjem po objektnom grafu

Relacijski model: `SELECT FIRST artikl.naziv FROM artikl, artikl_osoba, osoba WHERE`

 - Projekcijom i spajanjem tablica
 - **Nasljeđivanje nije podržano u relacijskom modelu**

Zašto objektno orijentirane baze podataka?

- Preslikavanje između dva modela je zahtjevan posao – potreba za transparentnim rukovanjem podacima iz baze, koristeći paradigme objektno orijentiranih jezika
- Ilustrativni primjer korištenja objektno baze podataka i odgovarajućeg okvira:

```
Kupac prototip = new Kupac("2011987335065");  
Kupac kupac = (Kupac)db.get(prototip).next();           //dohvati iz baze  
Artikl artikl = new Artikl("a1", "leteća vjeverica plišana");  
kupac.addArtikl(artikl);  
db.set(kupac);                                           //spremi u bazu
```

- Nema kompleksnih upita
- Automatizirano preslikavanje u objektni model
- Automatizirano pohranjivanje u objektnu bazu – moguće je pohraniti čitav objektni graf

Zašto objektno orijentirane baze podataka?

- Fokus se prebacuje na poslovni proces – objekt prolazi kroz potrebne faze procesa i takav izmijenjen se pohranjuje u bazu
- Dohvaćanje podataka iz relacijske baze može postati "usko grlo" aplikacije za kompleksne modele – potreba za efikasnijim načinom dohvaćanja podataka


Objektno orijentirani model

- **objekt** - entitet koji postoji u stvarnom svijetu
- **tip objekta - razred (class)** - skup entiteta iz stvarnog svijeta
 - svaki objekt pripada jednom razredu
 - razredi su hijerarhijski organizirani
- **struktura objekta - skup varijabli** - definiran je razredom
- **protokol** - definiran skupom poruka
- **poruka** se šalje nekom objektu da bi on obavio neku akciju. Ako parametri zadovoljavaju dana pravila, objekt će interpretirati poruku i obaviti akciju.
- **operacija - metoda** - skup procedura koje su definirane razredom. Metoda predstavlja implementaciju poruke. To je dio kôda koji obavlja željenu akciju

<u>Objektni pristup</u>	<u>Relacijski pristup</u>
OBJEKT	Entitet
RAZRED	Relacijska shema
VARIJABLE	Atributi
PORUKA	Poziv procedure
METODA	Procedura

SHEMA RAZREDA = SKUP VARIJABLI + METODE

-
- **Apstraktni tipovi podataka** (*abstract data types, user-defined data types*) - mogućnost definiranja vlastitih tipova podataka
 - **Učahurivanje** (*encapsulation*) - tehnika strukturiranja u kojoj se sustav izgrađuje od skupa modula kojima se pristupa preko dobro definiranog sučelja. Sučelje se definira s pomoću skupa strogo tipiziranih operacija (poruka).
 - **Nasljeđivanje** (*inheritance*) razred nasljeđuje svojstva hijerarhijski nadređenog razreda.
 - **Polimorfizam** - višeobličnost
 - **Ponovno korištenje** (*reusability*)



GOM (generic object model)
model neovisan o implementaciji
razvili su ga Kemper i Moerkotte

GOM - Tipovi objekata

- Definicija novog tipa objekata obavlja se s pomoću **okvira za definiciju tipa** (*type definition frame*)

```
[persistent] type TypeName [supertype Supertype Name] is
  [public OperationList]
  body TypeStructure
  [operations
    OperationSignature;
    ...
    OperationSignature;
  implementation
    OperationImplementation;
    ...
    OperationImplementation; ]
```

end type TypeName;

... GOM - Tipovi objekata

- ime tipa mora biti jedinstveno unutar modela
- jedan tip ima jedan i samo jedan nadtip - ukoliko nije eksplicitno naveden, smatra se da je nadtip **ANY**
- podtip nasljeđuje sva svojstva (strukturnu reprezentaciju i operacije) od svojeg nadtipa
- u odjeljku **public** navode se operacije (pristup i izmjena internog stanja objekata danog tipa) predviđene za klijente (korisnike), te sučelje objekata prema vanjskom svijetu, čime je osigurano učajurivanje
- unutar tijela (**body**) definira se interna struktura objekata, koja služi za
 - održavanje internog stanja objekata između sukcesivnih poziva operacija
 - u slučaju perzistentnih objekata, za održavanje internog stanja između obavljanja različitih programa

... GOM - Tipovi objekata

- u odjeljku **operation** navode se imena operacija koje su pridružene danom tipu (imena operacija moraju biti jedinstvena za dani tip), tipovi argumenata (objekata), te tipovi rezultata koji se vraćaju nakon poziva operacije
- implementacija operacije sadrži kôd za svaku navedenu operaciju.
- interna struktura objekata može biti definirana kao
 - n-torka
 - skup
 - lista

... GOM - Tipovi objekata

- Definicija tipova objekata sa strukturom n-torke:
attr₁: Type₁;
...
attr_n: Type_n;
 - Tipovi atributa (Type₁, ..., n) mogu biti
 - osnovne vrste podataka - cijeli broj, niz znakova,...
 - posebne vrste - korisnički definirane
 - neki od definiranih tipova objekata
- Definicija tipova objekata sa strukturom skupa:
{ElementType}
- Definicija tipova objekata sa strukturom liste: <ElementType>

GOM - Identifikacija objekta

- na osnovi sadržaja
- na osnovi smještaja
- ➡ pogreške
 - ➡ izmjenom sadržaja objekta
 - ➡ brisanjem objekta na čije mjesto se pohranjuje novi objekt ili premještanjem objekta, npr. prilikom kompresije
- svaki objekt prilikom kreiranja dobiva svoj jedinstveni identifikator, neovisan o sadržaju i smještaju objekta
- objekt se može prikazati trojkom:

$$o = (id_{\#}, Type, Rep)$$

... GOM - Tipovi objekata - primjeri

```
type Vrh is  
  public ...  
  body [ x,y,z: float;]  
  operations  
  ...  
  implementation  
  ...  
end type Vrh;
```

```
type Materijal is  
  public ...  
  body [ime: string;  
         specTez: float;]  
end type Materijal;
```

```
type VrhList is  
  public ...  
  body  
    <Vrh>  
end type VrhList;
```

```
type Kvadar is  
  public ...  
  body [ v1, v2, v3, v4, v5, v6, v7, v8: Vrh;  
         mat: Materijal;]  
end type Kvadar;
```

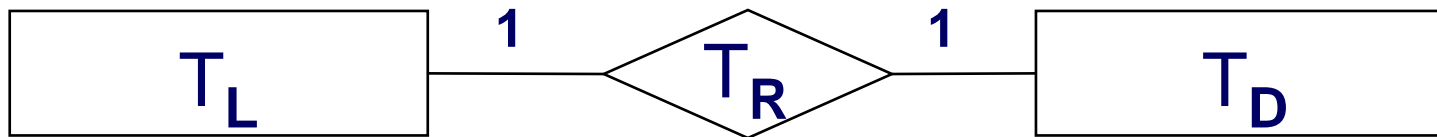
```
type Kvadar1 is  
  body [ vrhovi: VrhList;  
         mat: Materijal;]  
end type Kvadar1;
```

```
type KvadarSet is  
  body  
    { Kvadar}  
end type KvadarSet;
```


GOM - Tipovi veza među objektima

- Tipovi veza među objektima, prema preslikavanju i strukturnim karakteristikama dijele se u tri osnovne kategorije:
 - binarne veze bez atributa
 - binarne veze s atributima
 - n-arne veze s atributima

GOM - Binarna veza 1:1



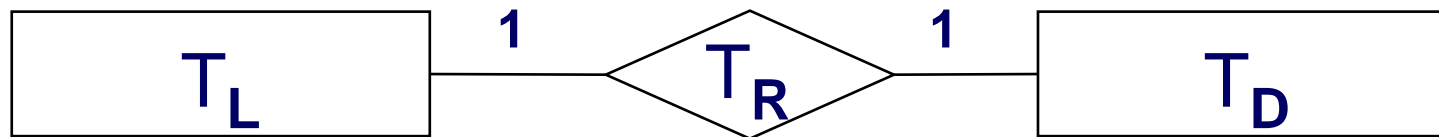
type T_L is
body
[...
R: T_D ;
...]
end type T_L ;

type T_D is
body
[...
R⁻¹: T_L ;
...]
end type T_D ;

type T_R is
body
[lijevi: T_L ;
desni: T_D ;
end type T_R ;

... GOM - Binarna veza 1:1

- zbog fleksibilnosti preporuča se uvođenje redundancije:

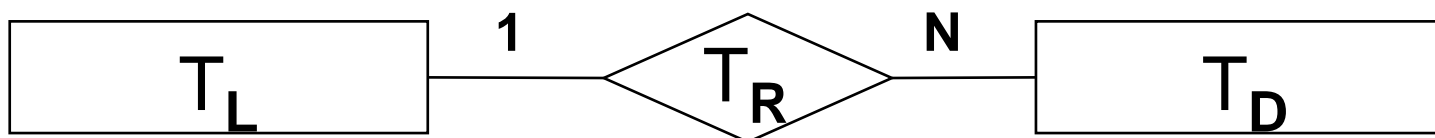


```
type  $T_L$  is
  body
    [ ...
      R:  $T_R$  ;
    ... ]
end type  $T_L$  ;
```

```
type  $T_R$  is
  body
    [lijevi:  $T_L$  ;
     desni:  $T_D$  ;]
end type  $T_R$  ;
```

```
type  $T_D$  is
  body
    [ ...
       $R^{-1}$ :  $T_R$  ;
    ... ]
end type  $T_D$  ;
```

GOM - Binarna veza 1:N



type T_L is

body

[...

R: $\{T_R\}$;

...]

end type T_L ;

type T_R is

body

[lijevi: T_L ;

desni: T_D ;]

end type T_R ;

type T_D is

body

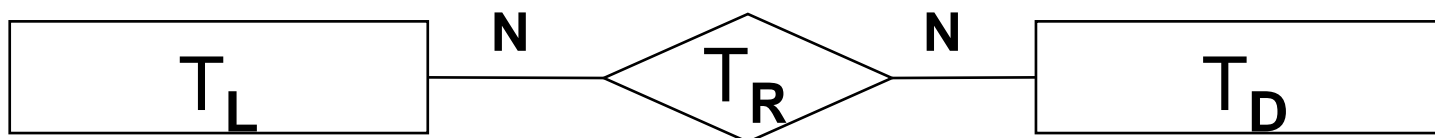
[...]

R^{-1} : T_R ;

...]

end type T_D ;

GOM - Binarna veza N:N



type T_L is

body

[...

R: $\{T_R\}$;

...]

end type T_L ;

type T_R is

body

[lijevi: T_L ;

desni: T_D ;

end type T_R ;

type T_D is

body

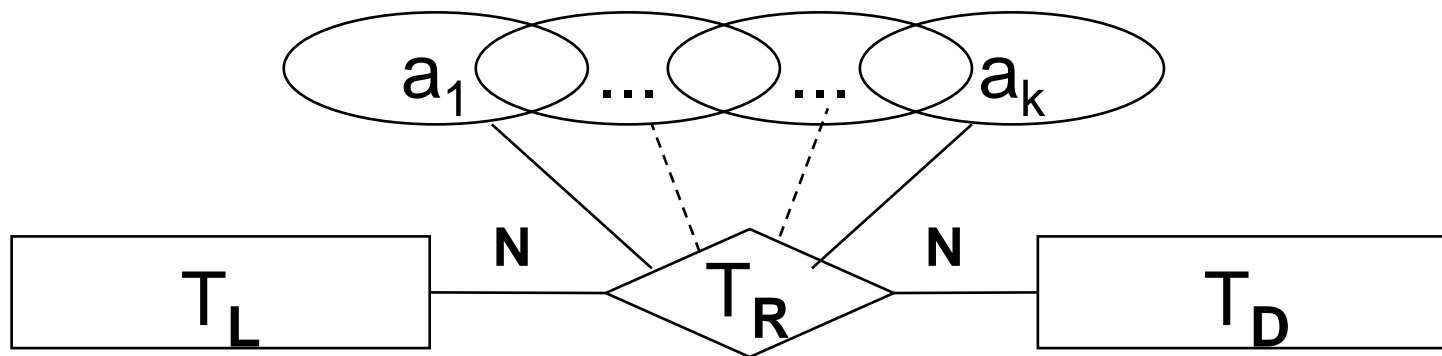
[...]

R^{-1} : $\{T_R\}$;

...]

end type T_D ;

GOM - Binarna veza s atributima



type T_R is

body

[lijevi: T_L ;

desni: T_D ;

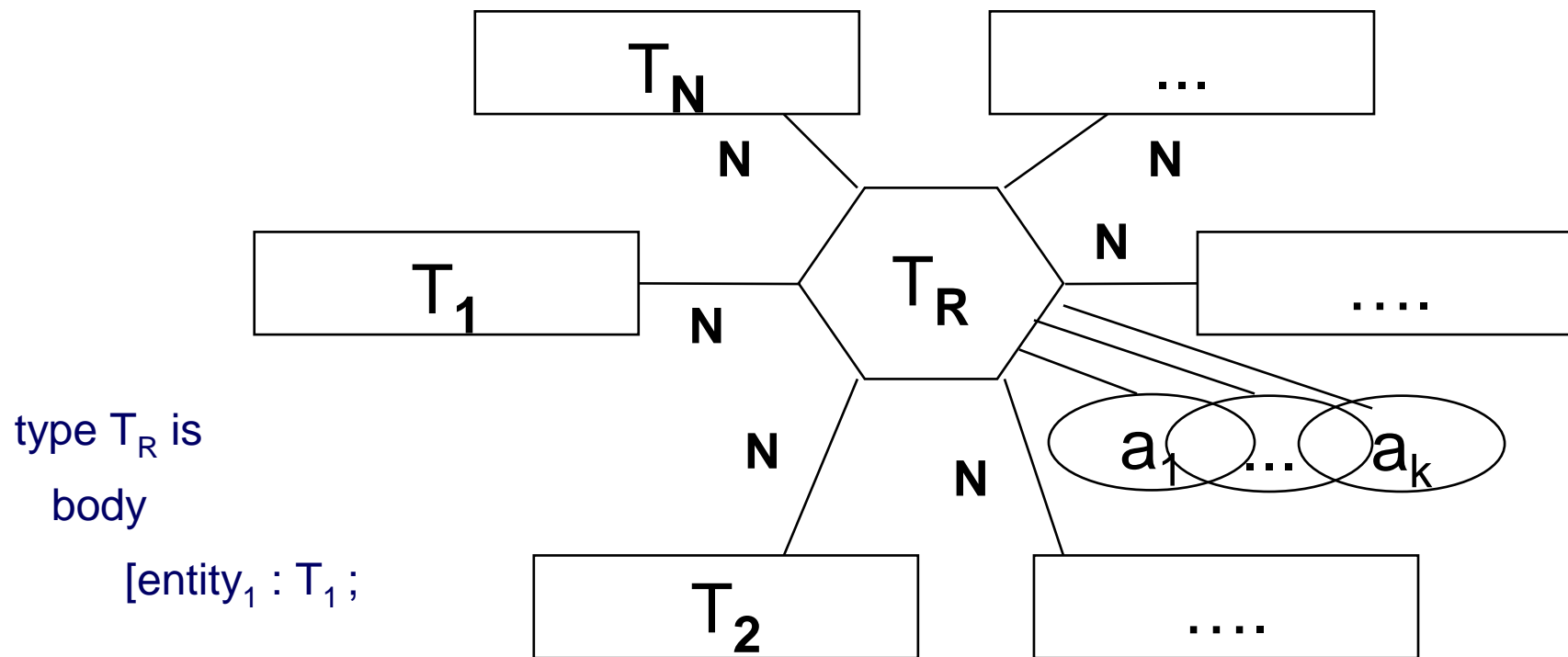
$a_1 : T_1^R$;

...

$a_k : T_k^R$;

end type T_R ;

GOM - n-arna veza s atributima



type T_R is
body

[entity₁ : T_1 ;

.....

entity_n : T_N ;

a_1 : T_1^R ;

...

a_k : T_k^R ;]

end type T_R ;

GOM - Inverzni atributi

- Kod veze opisane u redundantnom obliku, kontrolu konzistentnosti podataka sadržanih u vezi i inverznoj vezi (referencijski integritet) moguće je osigurati s pomoću **inverznih atributa**

```
type Muškarac is
```

```
....
```

```
  suprug: Žena inverse Žena$suprug ;
```

```
...
```

```
end type Muškarac ;
```

```
type Žena is
```

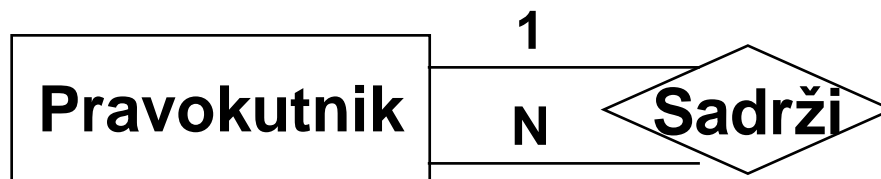
```
....
```

```
  suprug: Muškarac inverse Muškarac$supruga ;
```

```
....
```

```
end type Žena ;
```

GOM - Rekurzivna veza 1:N



type Pravokutnik **is**

....

širina, visina : float;

sadrži: { Pravokutnik } **inverse** Pravokutnik\$sadržanU;

sadržanU: Pravokutnik **inverse** Pravokutnik\$ sadrži;

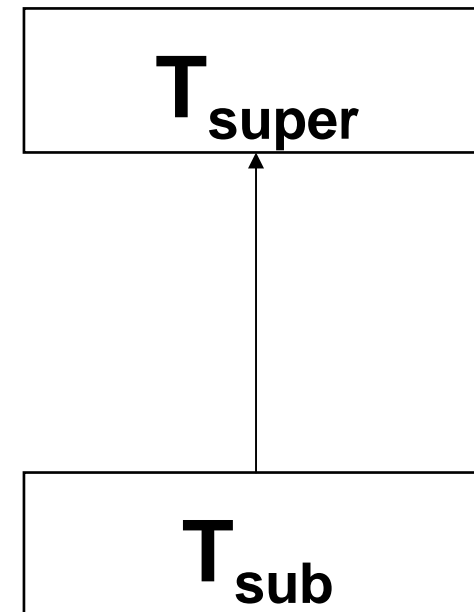
....

end type Pravokutnik;

$$\text{sadržanU}(r') = \{ r \mid r' \in \text{sadrži}(r) \}$$

GOM - Nasljeđivanje - veza tipa IS A

- podtipovi nasljeđuju sva svojstva (strukturalnu reprezentaciju i operacije) od svojeg nadtipa
 - definira se hijerarhija tipova - klasifikacija
-
- svaki objekt tipa T_{sub} automatski pripada i tipu T_{super}
 - OT_{sub} **is-a** OT_{super}
 - **načelo zamjenjivosti** osigurava da svaki podtip, s obzirom da nasljeđuje sva svojstva nadtipa, može supstituirati svoj nadtip



GOM – Primjer nasljeđivanja

```
persistent type Osoba is
  public ime, dob, bračniDrug, uBraku
  body [ime: string; dob: int; bračniDrug: Osoba;]
  operations
    declare uBraku: Osoba → void;
    ....
  implementation
    define uBraku (žrtva) is
      self.bračniDrug := žrtva;
    ....
end type Osoba;

persistent type Djelatnik supertype Osoba is
  public mbr, plaća, šef, uMirovini
  body [mbr: int; plaća: float; šef: Djelatnik;]
  operations
    declare uMirovini: → bool;
    ....
  implementation
    define uMirovini is
      return (self.dob > 70);
    ....
end type Djelatnik;
```

... GOM - Nasljeđivanje

- izvjestan skup operacija u bazi podataka mora se moći obaviti nad bilo kojim tipom objekta
 - kontrola paralelnog pristupa
 - proces obnove
 - transformacija strukture pohrane
 - ...
- poželjno je da svi tipovi objekata imaju zajednički nadtip
- u hijerarhijskoj strukturi tipova objekata postoji jedan tip koji predstavlja korijen stabla (GOM - ANY, Smalltalk-80 - Object)
- svojstva nasljeđuju svi tipovi objekata iz dotičnog stabla (tj. u dotičnoj bazi podataka).

GOM - Virtualni tipovi objekata i virtualne operacije

- Neki tipovi objekata ne posjeduju dovoljno funkcionalnosti za kreiranje objekata (ANY)
- Takvi tipovi - **virtualni tipovi** - predstavljaju samo **apstraktan okvir za definiciju funkcionalnosti** zajedničke za sve njegove podtipove
- Ne mogu se kreirati instance virtualnog tipa
- Definiraju se virtualne operacije koje se zatim u podtipovima redefiniraju
- Umjesto virtualne operacije obavlja se implementacija operacije tipa objekta koji je najniži u hijerarhiji
- Tipovi objekata koji nisu virtualni mogu također imati definirane virtualne operacije

GOM - Polimorfne operacije i generički tipovi

- polimorfizam = višeobličnost
- potreba da se definiraju operacije koje treba primjenjivati na tipove objekata koji nisu neophodno unutar iste grane u hijerarhiji tipova
- da bi se izbjegla redundancija operacija i olakšalo održavanje sustava :
 - dvije ili više praktički iste operacije koje su definirane za različite tipove objekata
- omogućena je parametrizacija tipova - specificiranje polimorfne operacije ili generičkog tipa objekta na apstraktnom nivou
- polimorfne operacije definiraju se za objekte s različitim strukturama.

GOM - Polimorfizam

- Objektni model podržava tri različite vrste polimorfizma:
 - **Ad hoc polimorfizam** - ostvaruje se redefinicijom operacija
 - **Inkluzivni polimorfizam** - ostvaruje se nasljeđivanjem svojstava i operacija od nadtipova
 - **Ograničeni polimorfizam** - polimorfne operacije koje su sposobne djelovati na objekte različitih tipova, a koji se ne nalaze u istom podstablu u hijerarhiji tipova objekata
- Struktura i/ili ponašanje objekata na koje se primjenjuje polimorfna operacija mora biti takva da se operacija može korektno provesti

Osnovni koncepti

- Objektно-orijentirane baze nazivaju se još i *bazama objekata* (*object databases*)
 - U bazu se pohranjuju objekti – model u bazi ne razlikuje se od onoga u aplikaciji
 - Implementacije OOSUBP uglavnom su programirane za specifičan programski jezik i međusobno se bitno se razlikuju
- **Manifest o objektно-orijentiranim sustavima baza podataka** (*The Object-oriented Database System Manifesto*), 1989. – teoretski manifest o svojstvima koje mora zadovoljavati OOSUBP
 - Koncepti objektно-orijentiranog sustava
 - Koncepti sustava za upravljanje bazama podataka

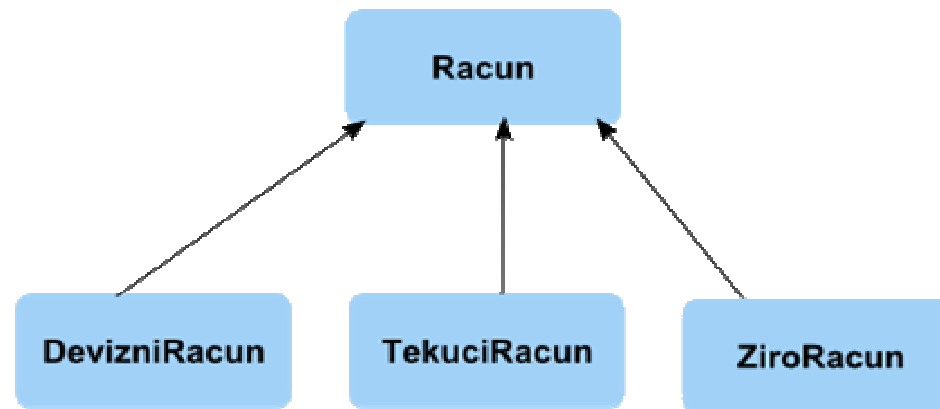
Osnovni koncepti

- Koncepti objektno-orijentiranog sustava
 - Tipovi podataka i klase
 - Kompleksni objekti
 - Proširljivost (*extensibility*)
 - Identitet objekta
 - Hijerarhije klasa i tipova podataka
 - Učahurivanje (*encapsulation*)
 - Nadjačavanje (*overriding*), preopterećivanje(*overloading*) i kasno vezivanje (*late binding*)
 - Računalna potpunost (*computational completeness*)
- Koncepti sustava za upravljanje bazama podataka
 - Perzistencija podataka
 - Fizička organizacija (*secondary storage management*)
 - Paralelni rad (*concurrency*)
 - Oporavak baze podataka (*recovery*)
 - Ad hoc upitni jezik (*Ad Hoc Query Facility*)

Tipovi podataka i klase, kompleksni objekti, proširljivost, računalna potpunost

- **Tipovi podataka i klase**
 - = shema baze podataka u OOSUBP
 - Sustav mora podržavati osnovne tipove: integer, float, boolean, character, string i klase
- **Kompleksni objekti**
 - Sustav mora podržavati kompleksne objekte: n-torke (*tuple*) , setove, liste
- **Proširljivost**
 - Omogućiti korištenje korisnički definiranih tipova podataka
- **Računalna potpunost**
 - OOSUBP zahtijevaju integraciju sa nekim računalno potpunim programskim jezikom za izvođenje operacija nad objektima
 - Postojeći programski jezik (Java, C++) ili vlastiti

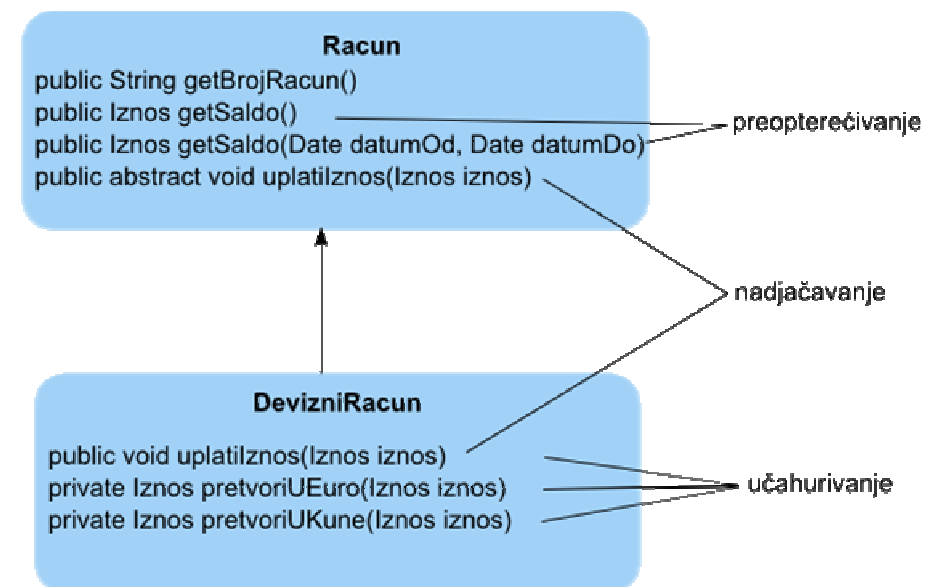
Hijerarhije klasa i tipova podataka



- Ostvaruju se nasljeđivanjem
- Dostupni atributi i metode nasljeđuju se od nadređene klase
- Podklasa može definirati nove attribute i metode
- Generalizacija (*Racun*) i specijalizacija (*DevizniRacun*)

Učahurivanje, nadjačavanje, preopterećivanje

- **Učahurivanje** - Komunikacija s korisnicima objekta preko sučelja uz skrivanje detalja o implementaciji ponašanja
- **Nadjačavanje metoda** – mogućnost redefinicije metode u podklasama
 - Kasno vezivanje - ispravna implementacija nadjačane metode objekta određuje se za vrijeme izvođenja
- **Preopterećivanje metoda** – mogućnost definiranja više istoimenih metoda s različitim potpisom unutar iste klase ili u podklasama



Identitet objekta

- Objekt mora biti jedinstveno identificiran
- Koristi se za referenciranje objekata (slično kao u OO jezicima)
- Identitet objekta je u idealnom slučaju neovisan o svim svojim vrijednostima – generiran od sustava (npr. fizička lokacija) ili korisnika (umjetna varijabla)
- Dva objekta su **identična** ako im je svojstvo koje ih jedinstveno identificira isto – identitet objekta.
- Npr. svaki račun ima jedinstveni broj računa, ali to nije nužno identitet objekta!

```
Racun  
public String getBrojRacun()  
public Iznos getSaldo()  
public Iznos getSaldo(Date datumOd, Date datumDo)  
public abstract void uplatiIznos(Iznos iznos)
```


Perzistencija (*Persistence*)

- Definira se kao "svojstvo objekta da nadživi izvođenje procesa"
- Sa stanovišta baze podataka perzistencija podataka se podrazumijeva
- Sa stanovišta programskog jezika:
 - Mogućnost transparentne manipulacije objektima pohranjenima u bazi koristeći OO programski jezik
- Perzistencija bi trebala biti :
 - Ortogonalna
 - Mogućnost pohranjivanja svakog objekta bez potrebe za pretvorbu u neki specijalan tip podataka
 - Implicitna
 - `db.set(kupac)`

Perzistencija (2)

- Sa stanovišta aplikacije objekti mogu biti:
 - Perzistentni
 - Odgovaraju podacima u bazi
 - Tranzitivni
 - Tek se trebaju spremiti u bazu
 - Tranzijentni
 - Nemaju potrebe za spremanjem u bazu

```
Kupac kupacPrototip = new Kupac ("Mirko");           //tranzijentan
List<Kupac> kupciKojiSeZovuMirko = (List<Kupac>) db.get(kupacPrototip);
Kupac prviMirko = kupciKojiSeZovuMirko.get(0);        //perzistentan
prviMirko.setNadimak("Mirky");                       //tranzitivan
db.set(prviMirko);                                   //perzistentan
```

Perzistencija (3)

- Problemi s konzistentnošću – u objektnom stablu istovremeno postoje i perzistentni i tranzitivni objekti

```
Kupac prototip = new Kupac("2511982335065");  
Kupac kupac = (Kupac)db.get(prototip).get(0);    //dohvati kupca iz baze  
kupac.addArtikl(new Artikl("Novi artikl"));      //dodan novi artikl  
//spremanje u bazu - što je s artiklima?  
db.set(kupac);
```

- Rješenje : ako jedan objekt postane perzistentan, onda i svi tranzitivni objekti koji se preko njega mogu dohvatiti trebaju postati perzistentni – **perzistencija po dohvatljivosti** (*Persistence By Reachability*)

Veze između objekata

- Ostvaruju se **referenciranjem**
- 1:1 veza
 - `kupac.getKosarica()`
 - `kosarica.getKupac()`
- N:1 (1:N) veza
 - `kupac.getMjestoStanovanja()`
 - `mjesto.getKupci()`
- N:N veza
 - `kupac.getArtikli()`
 - `artikl.getKupci()`
- Sve veze mogu biti **bidirekcionalne**
- Bidirekcionalnost nije nužno izraziti u objektnom modelu, ako nije bitna za poslovni proces aplikacije

Koncepti SUBP

- Mora osigurati mehanizme **oporavka podataka**
- Mora podržavati **paralelan rad**
- **Fizička organizacija**
 - OOSUBP moraju podržavati vrlo velike baze podataka
 - Indeksi
 - Klasteriranje podataka
 - Optimizacija upita
- **Ad hoc upitni jezici** - trebaju biti:
 - Deklarativni – što treba napraviti, a ne kako
 - `db.get(Artikl.class) //dohvati sve artikle iz baze`
 - Izražajni – sposobnost sažetog izražavanja kompliciranih upita
 - `db.get(Artikl proptotip) //dohvati artikle prema prototipu`
 - Neovisni o modelu aplikacije
 - Efikasni

Objektno orijentirani sustavi za upravljanje bazama podataka (OOSUBP)

- Temeljeni na razvoju:
 - Objektno orijentirane analize i dizajna (OOA, OOD)
 - Objektno orijentiranog programiranja (OOP)
 - Objektno orijentiranih distribuiranih sustava (npr. CORBA)
- Ciljevi:
 - Napredni semantički koncepti za opisivanje činjenica iz stvarnog svijeta
 - Poboljšana efikasnost nestandardnih aplikacija
 - Premostiti razlike između relacijskih baza i objektno orijentiranog programiranja
- Definicija:
 - OOSUBP je sustav za upravljanje bazama podataka koji implementira objektno orijentirani model podataka
 - Podaci kao objekti
 - Potpora za klase i nasljeđivanje

OOSUBP – povijest

- Prvi prototipi za istraživanja se javljaju početkom 80tih godina (EXODUS, ORION, IRIS, ODE)
- A prvi komercijalni sustavi početkom 90tih (GemStone, Objectivity, ObjectStore, POET, O2)
- Prva verzija ODMG standarda izlazi 1994.
- 90tih se u relacijske sustave uvode objektno orijentirani koncepti → nastanak objektno relacijskih baza podataka
- Objektne baze se polako prestaju koristiti, tako da je malo komercijalnih proizvoda preživjelo do danas
- Od 2004. se bilježi drugi rast zahvaljujući *open source* proizvodima (db4o, Perst)
 - Besplatni i laki za korištenje budući da su potpuno napisani Smalltalk, C# ili Java programskim jezicima

OOSUBP – različiti pristupi

- Temeljeni na OOP modelima podataka
 - Koriste standardizirane podatkovne modele popularnih objektno orijentiranih programskih jezika kao što je C++ (Objectivity, ObjectStore), Smalltalk (GemStone) ili Java (Jasmine, JD4O), te njima dodali funkcionalnost sustava za upravljanje bazama podataka
- Proširenja na relacijske modele
 - Uvode objektno orijentirane koncepte u relacijske modele (PostGres, Illustra) ili grade nad postojećim sustavima za upravljanje bazama podataka (Oracle, DB2, ...) → Objektno relacijski sustavi
- Prirodni OO modeli baza podataka
 - Razvijeni neovisno o postojećim modelima i sustavima (O2, ORION, ITASCA)

OOSUBP – prednosti

- Bolje i brže upravljaju s kompleksnim objektima i vezama u odnosu na relacijske sustave
 - Umjesto deklarativnih sučelja koriste navigacijska, koja se vrlo efikasno implementiraju s pokazivačima
- Podržavaju hijerarhiju, klase i nasljeđivanje
- Nema objektno relacijske neusklađenosti
- Nema potrebe za primarnim ključem (?)
 - Identifikacija objekata je skrivena od korisnika
- Jedan podatkovni model
 - Objekti u bazi i objekti u aplikaciji su jednaki, pa nema potrebe za dva modela
- Koristi se samo jedan programski jezik (za aplikaciju i za pristup bazi)
- Nema potrebe za posebnim upitnim jezikom

OOSUBP – mane

- Nema logičke neovisnosti podataka
 - Izmjene na bazi podataka zahtijevaju izmjene u aplikaciji i obrnuto
- Nedostatak dogovorenih standarda, tj. postojeći standard (ODMG) nije u potpunosti implementiran
- Ovisnost o jednom programskom jeziku. Tipični OOSUBP je svojim programskim sučeljem vezan za samo jedan programski jezik
- Nedostatak interoperabilnosti s velikim brojem alata i mogućnosti koje se koriste u SQL-u
- Nedostatak Ad-Hoc upita (upiti na novim tablicama koje se dobiju spajanjem postojećih s *join*)

OOSUBP – kada koristiti?

- Ugrađene (*embedded*) DBMS aplikacije gdje je potrebno osigurati perzistenciju podataka na najjednostaviji način
- Kompleksni odnosi između podataka. U aplikacijama gdje klase definiraju mnogostruke križne reference između sebe ili u aplikacijama koje uključuju mrežne podatkovne strukture.
- Objekti duboke strukture. Npr. u aplikacijama gdje su podaci organizirani kao duboko stablo.
- Kad se struktura podataka (objekata) mijenja tijekom vremena
- Ako programerski tim koristi tehnike brzog (*agile*) programiranja
- Kada objekti uključuju kolekcije
- Kada se koristi objektno orijentirani jezik

OOSUBP u stvarnom svijetu

- Chicago Stock Exchange - upravljanje s trgovinom dionica (Versant)
- Radio Computing Services – automatiziranje radio stanica (POET)
- Ajou University Medical Center u Južnoj Koreji – sve funkcije bolnice, uključujući one kritične poput patologije, laboratorija, banke krvi, ljekarne i rendgena (Cachè)
- CERN – veliki znanstveni setovi podataka (Objectivity/DB)
- Federal Aviation Authority – simulacija prometa putnika i prtljage
- Electricite de France – upravljanje elektroenergetskim mrežama
- Naglasak je na:
 - Efikasnosti
 - Lakoći implementacije

OOSUBP proizvodi

- Versant
- Progress ObjectStore
- Objectivity/DB
- Intersystems Cachè
- POET fastObjects
- **db4o**
- Computer Associates Jasmine
- GemStone

ODMG standard

- Object Managment Group (OMG)
 - Konzorcij inicijalno namijenjen postavljanju standarda za distribuirane objektno orijentirane sustave (CORBA)
 - Danas se bavi modeliranjem (programa, sustava i poslovnih procesa) i standardiziranjem tih modela
 - Najpoznatiji po UML
- Object Data Managment Group (ODMG)
 - Nadopunjuje OMG, u smislu da uključuje specifikacije za objektnu bazu podataka
 - Prenosivost i interoperabilnost između proizvoda koji uključuju objektnu bazu podataka

ODMG standard

- Nije potpuno podržan od sustava za upravljanje bazama podataka, ali obuhvaća sva glavna obilježja OOSUBP
- Sastoji se od slijedećih dijelova:
 - ODMG objektnog modela
 - Objektnog definicijskog jezika (ODL, eng. Object Definition Language)
 - Objektnog upitnog jezika (OQL, eng. Object Query Language)
 - Veze na programske jezike
 - Java
 - C++
 - Smalltalk



ODMG standard - objektni model

- Temeljen na OMG objektnom modelu
- Glavni koncepti su *objekti, tipovi, operacije, svojstva, identitet i podtipovi*
- Osnovni primitivi za modeliranje:
 - Objekti (imaju jedinstveni identifikator)
 - Literali (nemaju jedinstveni identifikator, već se identificiraju po svojoj vrijednosti)
- Stanje objekta je definirano svojstvima
- Ponašanje objekta je definirano operacijama
- Objekti i literali su određeni tipom koji definira zajednička svojstva i zajedničko ponašanje
- Tip se sastoji od sučelja (svojstva, operacije, iznimke) i jedne ili više implementacija (veza na programski jezik)
- Postoje dvije vrste veza kod nasljeđivanja:
 - IS-A (nasljeđivanje ponašanja): `interface Bicikl : Artikl {...};`
 - EXTENDS (nasljeđivanje ponašanja i svojstava): `class ArtiklBickl extends Artikl : Bicikl {...};`

ODMG standard - ODL

- ODL - Object Definition Language
- Temeljen na OMG IDL (CORBA Interface Definition Language)
- Specifikacijski jezik namijenjen definiranju sučelja objektnog tipa
- Trebao bi podržavati sve koncepte objektnog modela
- Trebao bi biti neovisan o programskom jeziku
- Trebao bi biti praktičan i proširiv

ODMG standard - ODL

- Deklaracija klase:

```
interface <name> {elements = attributes, relationships,  
  methods}
```

- Deklaracija elemenata:

```
attribute <type> <name> ;  
relationship <rangetype> <name> ;
```

- Primjer:

```
Type Date Tuple {year, day, month}
```

```
Type year, day, month integer
```

```
Class Manager
```

```
  attributes(  
    id : string unique  
    name : string  
    phone : string  
    set employees : Tuple { [Employee], Start_Date : Date  
  })
```

```
Class Employee
```

```
  attributes (
```

- Deklaracija metoda:

```
<returnType> <name> ((in|out|inout: <type> <name> )*) raises(  
    <exception> );
```

- Primjer:

```
class Article extends Publication(extent Articles){  
    exception IllegalPageNumber{ unsigned short pageNumber };  
    attribute unsigned short beginPage;  
    attribute unsigned short endPage;  
    unsigned short getBeginPage();  
    void setBeginPage(in unsigned short beginPage)  
        raises (IllegalPageNumber);  
    unsigned short getEndPage();  
    void setEndPage(in unsigned short endPage)  
        raises (IllegalPageNumber);  
};
```

ODMG standard - ODL

- Primjer – veze:

```
class Author {  
    ...  
    relationship set<Publication> authors  
    inverse Publication::authoredBy;  
    ...  
}  
  
class Publication {  
    ...  
    relationship list<Author> authoredBy  
    inverse Author::authors;  
    ...  
}
```

ODMG standard - OQL

- Upitni jezik za objektne baze podataka napravljen po uzoru na SQL
- Veoma svestran i fleksibilan, ali zbog svoje kompleksnosti niti jedan ga proizvođač nije kompletno implementirao
- Razlike između OQL i SQL:
 - OQL podržava referenciranje na objekte unutar tablica. Objekti mogu ugnježdjavati druge objekte.
 - OQL ne podržava sve ključne riječi iz SQL
 - OQL podržava matematičke izračune unutar OQL izraza
- Sintaksa:
 - Koristi upite u *Select-From-Where* stilu
 - Navigacija kod kompleksnih objekata:
`knjiga.izdavac.kontakt.email`

ODMG standard - OQL

- Primjer 1 – Dohvati jelovnik u restoranu "Snack":

```
SELECT s.dish.name, s.price  
FROM Sells s  
WHERE s.restaurant.name = "Snack"
```

- Primjer 2 – Dohvati jelovnik, ali preko objekta *restaurant*:

```
SELECT s.dish.name, s.price  
FROM Restaurants r, r.dishesSold s  
WHERE r.name = "Snack"
```

ODMG standard – Veze na programske jezike

- Postoji za C++, Java i Smalltalk
- Uključuje ODL koji je ovisan o odabranom programskom jeziku
- Pruža aplikacijsko programsko sučelje za preslikavanje tipova podataka
- Preslikavanje tipova za Java programski jezik:

ODL	Java
Long	int
Float	float
Boolean	boolean
Date	java.sql.data
Time	java.sql.time
...	...

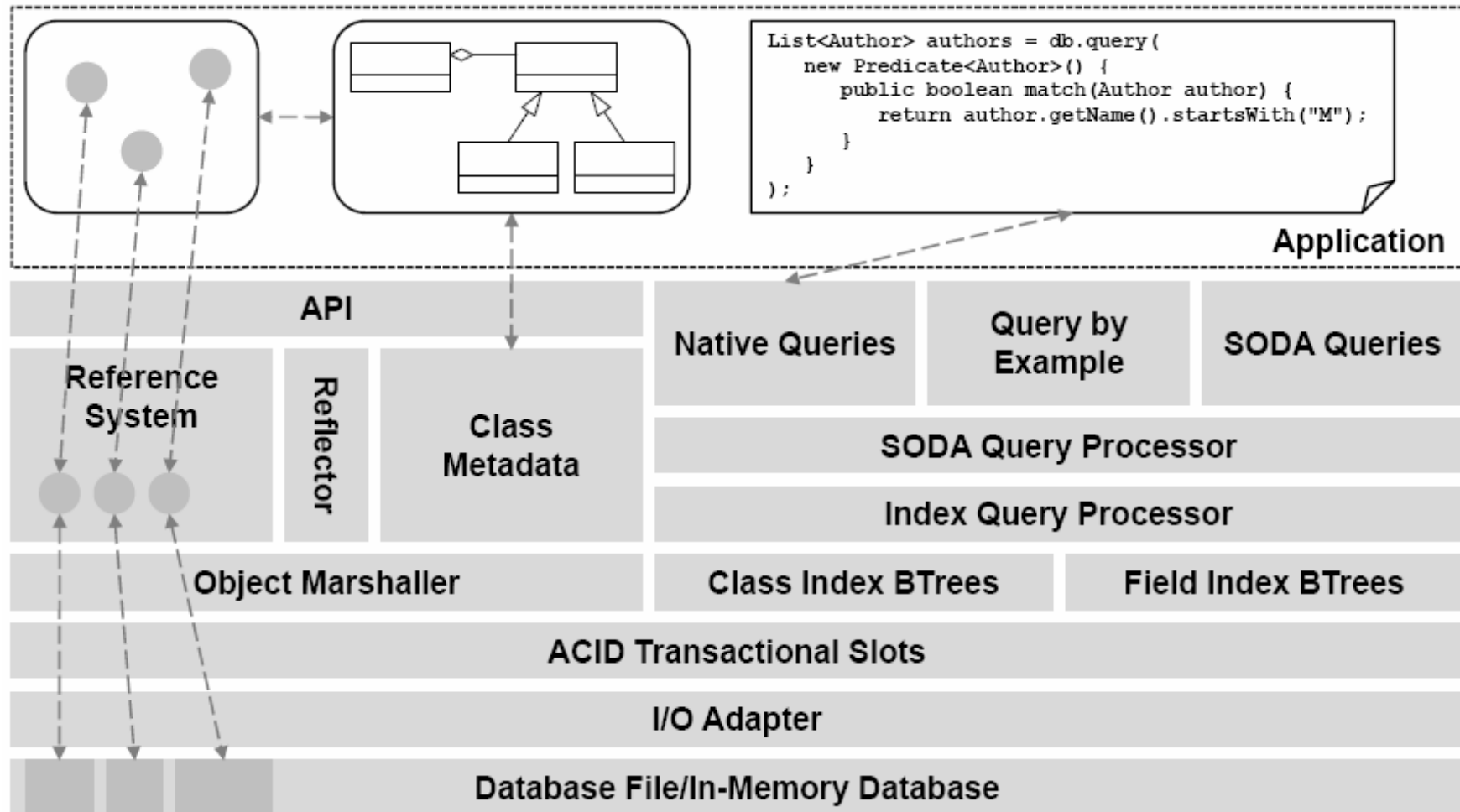
Db4O

- *Database For Objects*, otvoreni kôd (*open source*)
- Podržani jezici:
 - Java
 - .NET jezici (C#, VB)
- Podržava ACID svojstva transakcija, indekse, replikaciju
- *Native language* baza
 - Npr. db4o za Javu
 - Napisan u Java jeziku – DDL i DML je Java
 - Pohranjuje Java objekte
 - Pričuvna memorija objekata
 - Automatska sinkronizacija sheme i podatka u slučaju promjene objektnog modela aplikacije
- Načini rada:
 - Lokalno – moguća samo jedna istodobna konekcija
 - Korisnik-poslužitelj

Db4o vs. Relacijski SUBP

	Sql server	Db4o
Model u bazi	<ul style="list-style-type: none">▪ Za svaki objekt aplikacije tipično barem jedna tablica▪ Povezivanje tablica ključevima	<ul style="list-style-type: none">▪ Model aplikacije = Model u bazi▪ Isti objekti koji se koriste u aplikaciji, mogu se spremiti u bazu▪ Veze izražene samim modelom
Pohranjene procedure	<ul style="list-style-type: none">▪ SPL – izdvojeni jezik▪ Više programskog koda za održavanje – dio u aplikaciji, a dio u procedurama	<ul style="list-style-type: none">▪ ne postoje▪ sve operacije izvode se u istom jeziku
Pogledi (views)	<ul style="list-style-type: none">▪ standardan način za kontrolu pristupa, agregiranje podataka itd.	<ul style="list-style-type: none">▪ ne postoje▪ svi podaci su već u objektnom grafu

Arhitektura db4o



Način pohranjivanja objekata u bazi

- Identitet objekta (OID) – jedinstveno identificira objekt:
 - Generiran od strane db4o i skriven od korisnika
 - Odgovara lokaciji u bazi gdje je pohranjen objekt
- Objekti se međusobno povezuju putem OID-a
- Uobičajeno postojanje "duplikata" u bazi – objekata koji imaju iste vrijednosti, a različit OID
- Provjera integriteta – kontrolira se u aplikaciji, a ne u bazi
- Kolekcije i polja – spremaju se kao zasebni objekti, sa vlastitim OID-om

Klasa *ObjectContainer* (OC)

- Sučelje prema db4o bazi podataka
- Kreiranje, otvaranje/zatvaranje baze
- Upiti, izmjena, brisanje objekata
- Transakcije
 - Sve operacije izvode se unutar transakcije
 - Transakcija počinje kada se otvori baza podataka
 - Nakon potvrde/poništenja automatski započinje nova transakcija
- Upravljanje objektima (već pohranjenim i novim)
 - Upravljanje identitetima objekata
 - Reference na objekte u radnoj memoriji se odbacuju pri zatvaranju OC

```
// otvaranje baze podataka - početak transakcije
ObjectContainer db = Db4o.openFile("C:\\db4oBaza.odt");

// upit nad bazom - dohvati sve knjige
db.get(Knjiga.class);

// zatvaranje baze - automatski se potvrđuje transakcija
db.close();
```

Osnove rada s db4o - primjer objektnog modela

```
public abstract class Osoba {
    private String jmbg;
    private String ime;
    private String prezime;
    ...
    // pristupne metode i konstruktori
    ...
}

public class Autor extends Osoba{
    private List<Knjiga> knjige;
    ...
    // pristupne metode i konstruktori
    ...
    // ostale metode
    public void addKnjiga(Knjiga
knjiga){
        knjige.add(knjiga);
    }
    ...
}
```

```
public class Lektor extends Osoba{
    private List<Knjiga> knjige;
    ...
    // pristupne metode i konstruktori
    ...
}

public class Knjiga {
    private String oznaka;
    private String naziv;
    private int godina;
    private List<Autor> autori;
    ...
    // pristupne metode i konstruktori
    ...
    // ostale metode
    ...
}
```

Dohvat objekata

- Dohvaćaju se kolekcije unaprijed određenih objekata (a ne n-torke)
- Tri načina dohvata objekata u db4o - moguće ih je kombinirati
- **Upiti prema primjeru** (*Query by example*)
 - Jednostavni upiti, temeljeni na primjeru (prototipu)
 - Dohvat po atributima objekta – primjera
- **Native upiti** (*Native queries*)
 - Izražavaju se u jeziku aplikacije koristeći db4o API
 - Preporučeno korištenje u većini slučaja
 - Podržavaju generičke tipove (*type-safe*)
- **SODA upiti** (*Simple Object Data Access*)
 - Dinamičko generiranje upita
 - Sortiranje
 - *Native* upiti i upiti prema primjeru transformiraju se u SODA upite

Upiti prema primjeru (*Query by example*)

```
// prototip
Knjiga knjigaPrototip = new Knjiga();
knjigaPrototip.setGodina(1995);

// dohvati knjige iz 1995. godine
ObjectSet knjige = db.get(knjigaPrototip);

// dohvati sve knjige - dovoljno specificirati tip objekta
List sveKnjige = db.get(Knjiga.class);
```

- Metoda **get** klase *ObjectContainer*
- Vraća kolekciju objekata proširenu dodatnom funkcionalnošću
- **Nedostaci:**
 - Moguće izraziti samo ekvivalenciju (nema >, < ...)
 - Nema složenih izraza (AND, OR, NOT ...)
 - Vrijednosti postavljene na *null*, 0, i "" ne mogu se postaviti kao uvjet dohvata – ignoriraju se

Native upiti

```
// dohvati sve knjige koje su napisane između 1350. i 1650. godine ili im naziv
// počinje s B
List<Knjiga> knjige = db.query(new Predicate<Knjiga>() {
    public boolean match(Knjiga knjiga) {
        return (knjiga.getGodina() >= 1350 && knjiga.getGodina() <= 1650)
            || knjiga.getNaziv().startsWith("B");
    }
});
```

- Metoda **query** klase *ObjectContainer*
- Moguće postaviti **proizvoljan uvjet** koristeći programski jezik aplikacije
- Transformira se u ekvivalentan SODA upit
 - Ako db4o API ne zna transformirati neku operaciju, ona se izvodi nad objektima nakon što se dohvate iz baze ili nad međurezultatima – neefikasno

SODA upiti (*Simple Object Data Access*)

```
// dohvati sve knjige koje su napisane između 1350. i 1650. godine ili im naziv
// počinje s B i sortiraj po godini uzlazno
Query query = db.query();
query.constrain(Knjiga.class);
Query godinaQuery = query.descend("godina");

//sortiraj po godini
query.descend("godina").orderAscending()

query.descend("naziv").constrain("B").startsWith()
    .or(godinaQuery.constrain(new Integer(1349)).greater()
    .and(pointQuery.constrain(new Integer(1651)).smaller()));
ObjectSet result = query.execute();
```

- Unutarnji API db4o (Sučelja *Query* i *Constraint*)
- Osigurava efikasne upite nad bazom podataka
- Kompliciranije za korištenje nego *native* upiti

SODA upiti (2)

- Objekti tipa *Query* – metode:
 - **descend** – dodaje čvor u stablo upita
 - **constrain** – dodaje uvjet na čvor u stablu upita
 - **sortBy** – sortiranje rezultata
 - **orderAscending** i **orderDescending** - sortiranje uzlazno/izlazno
 - **execute** – izvođenje upita
- Sučelje *Constraint* – metode:
 - **greater** i **smaller** usporedba
 - **identity**, **equal** i **like** usporedba
 - **and**, **or** i **not** operatori
 - **startsWith** i **endsWith** – usporedba za nizove znakova
 - **contains** – testiranja članstva za kolekcije

Unos objekata

```
Knjiga knjiga = new Knjiga("knj1", "Uzgoj konja u Hrvatskoj");

// dodavanje autora
knjiga.addAutor(new Autor("1811981336065", "Petar", "Balen"));

// unos nove knjige u bazu
db.set(knjiga);
```

- Metoda **set** klase *ObjectContainer*
- Pohranjivanje objekata **proizvoljne složenosti** – perzistencija po dohvatljivosti
 - db4o održava pričuvnu memoriju objekata u radnoj memoriji
 - Zna koje objekte u objektnom grafu treba unijeti u bazu, a koje treba izmijeniti

Izmjena objekata

```
Knjiga knjigaPrototip = new Knjiga("knj1", "Uzgoj konja u Hrvatskoj");

// dohvat knjige iz baze
Knjiga knjiga = (Knjiga)db.get(knjigaPrototip).get(0);

// promjena naslova knjige
knjiga.setNaziv("Uzgoj lipicanaca");

// izmjena knjige u bazi
db.set(knjiga)
```

- Metoda **set** klase *ObjectContainer*
- Razlika u odnosu na unos
 - OID objekta mora biti dohvaćen:
 - Objekt dohvaćen upitom, ili
 - Objekt netom unesen kao nov metodom *set*
 - Predodređeno ponašanje: izmjene se vrše samo nad jednostavnim atributima objekta (String, Boolean ...), ne pohranjuje se čitav objektni graf

Izmjena objekata – referencirani objekti

```
// dohvat autora iz baze
Autor autor = (Autor)db.get(new Autor("Mate", "Balen")).get(0);

// promjena godine za sve knjige
for(Knjiga knjiga : autor.getKnjige()){
    knjiga.setGodina(1977);
}

// izmjena autora - ne pohranjuju se referencirane knjige
db.set(author);

// pohranjivanje knjiga
for(Knjiga knjiga : autor.getKnjige()){
    db.set(knjiga);
}
```

- Izmjena referenciranih objekata zasebno
- Ili definiranje kaskadnih operacija

Izmjena objekata - kaskadiranje

```
// definiranje kaskadne operacije za objekte tipa Autor
Db4o.configure().objectClass(Author.class).cascadeOnUpdate(true);
...
// promjena naslova knjige
knjiga.setNaziv("Uzgoj lipicanaca");

// dodavanje autora
knjiga.addAutor(new Autor("1811981336065", "Petar", "Balen"));

// izmjena knjige u bazi - pohranjuju se i autori
db.set(knjiga)
```

- Metoda *cascadeOnUpdate* klase *ObjectClass* – definiranje kaskadne izmjene objekata na razini klase
- Moguće je definirati dubinu izmjene
 - Metoda *updateDeapth(deapth)* klase *ObjectClass* – na razini klase
 - Metoda *updateDeapth(deapth)* sučelja *Configuration* – globalno
- Globalna razina nije dovoljno fleksibilna

Brisanje objekata

```
// definiranje kaskadne operacije za objekte tipa Autor
Db4o.configure().objectClass(Author.class).cascadeOnDelete(true);

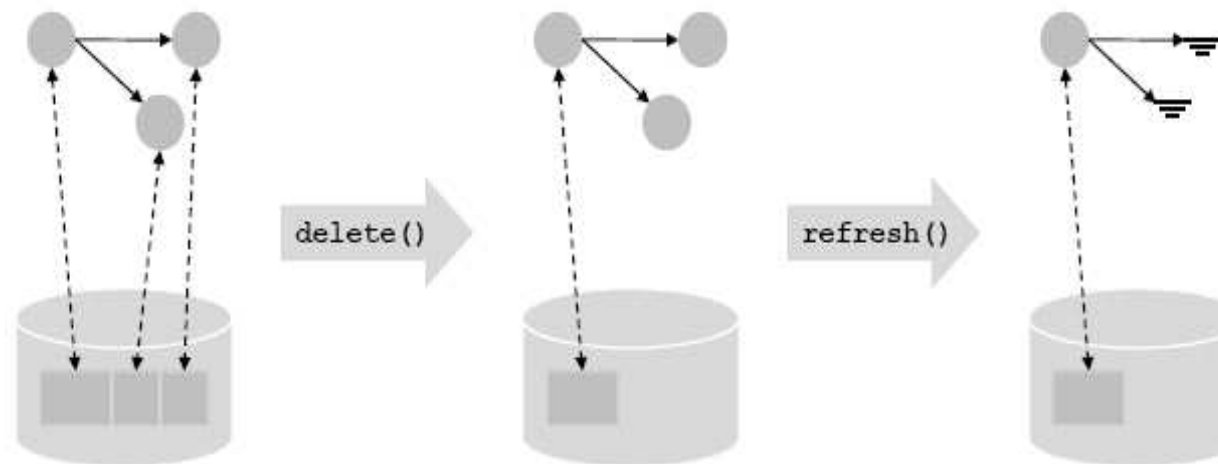
Knjiga knjigaPrototip = new Knjiga("knj1", "Uzgoj konja u Hrvatskoj");

// dohvat knjige iz baze
Knjiga knjiga = (Knjiga)db.get(knjigaPrototip).get(0);

// brisanje knjige iz baze - autori se brišu samo ako je tako definirano
db.delete(knjiga)
```

- Metoda ***delete*** klase *ObjectContainer*
- Kao i kod izmjene objekta – OID objekta mora biti dohvaćen
- Predodređeno ponašanje: ne brišu se referencirani objekti
- Moguće definiranje kaskadnog brisanja na razini klase metodom *cascadeOnDelete*
- Što se događa kada je obrisani objekt referenciran negdje drugdje?

Brisanje objekata (2)



- Prilikom brisanja objekata može doći do nekonzistentnosti podataka u bazi i memoriji - objekt je obrisao u bazi, a još uvijek postoji referenca u memoriji
- Metoda *refresh()* klase *ExtObjectContainer* sinkronizira stanje u memoriji

Hijerarhije objekata u db4o

```
// dohvat svih osoba koje se zovu "Mark" - pretražuju se svi objekti koji su
// tipa Osoba (Autor, Lektor, ..)
List<Osoba> osobeMarko = (List<Osoba>) db.get(new Osoba("Mark"));

// brisanje knjige iz baze - autori se brišu samo ako je tako definirano
db.delete(knjiga)

// promjena svih imena u "Marko"
for(Osoba osoba : osobeMarko){
    osoba.setIme("Marko");
    db.set(osoba);
}
```

- Dohvat objekata – moguć dohvat po nadređenoj klasi
- Unos, izmjena i brisanje objekata – pohranjuje se (odnosno briše, mijenja) konkretna implementacija objekta, podaci o sučeljima i nadređenim klasama nalaze se u meta-podacima baze

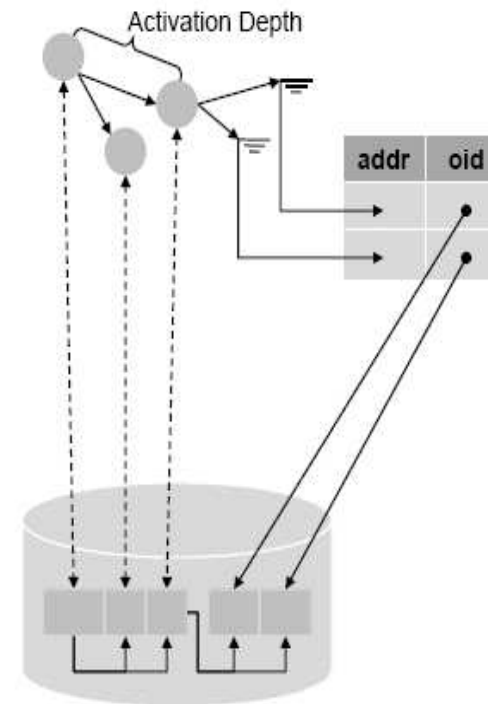
Aktivacija objekata

- Za objekt se kaže da je aktiviran ako je dohvaćen iz baze
- Kako i do koje razine aktivirati attribute objekta?
 - Ako se dubina aktivacije postavi na maksimalnu vrijednost čitavi objektni grafovi dohvaćaju se iz baze - nepoželjno
 - Ako se dubina aktivacije postavi na minimalnu vrijednost potrebno je kontrolirati aktivaciju u aplikaciji – nepraktično
 - Atributi se dohvaćaju samo do zadane dubine, nakon koje se postavljaju na *null*
- Db4o – kontrola aktivacije
 - Globalna kontrola : predodređena dubina aktivacije je 5, minimalna dubina je 1 – dohvaćaju se samo jednostavni atributi objekata
 - Metode *activate* i *deactivate* klase *ObjectClass*
 - Konfiguracija dubine aktivacije i kaskadiranja na razini klase

```
// globalno postavljanje dubine aktivacije na 1
Db4o.configure().activationDepth(1);
// aktivacija objekta lektor, knjige se ne dohvaćaju!
Lektor lektor = (Lektor) db.get(new Lektor("Marko", "Sopek"));
// aktivacija kolekcije knjiga sa dubinom 1
db.activate(lektor.getKnjige(), 1);
```

Aktivacija (2)

- Atributi objekta se dohvaćaju samo do zadane dubine aktivacije
- Objekti se kasnije mogu aktivirati metodom *activate*
- Objekti se aktiviraju preko tablice preslikavanja između memorijske adrese i adrese u bazi (oid)



Transakcije

- ACID model
- Sve operacije unutar db4o *ObjectContainer*-a (OC) izvode se unutar transakcije
 - **Implicitno**
 - Otvaranje OC – početak transakcije
 - Zatvaranje OC – potvrda transakcije/ponišćavanje kod neuspjele potvrde
 - **EksPLICITNO**
 - Metode OC: **commit** i **abort**
 - Nakon završetka transakcije implicitno se stvara nova
- Db4o koristi **read committed** izolacijski nivo – moguće kolizije
- Nakon poništavanja transakcije moguća je nekonzistentnost između objekata u bazi i aplikaciji

Indeksi

- Osiguravaju brži dohvat podataka, s usporavaju unos, izmjenu i brisanje
- Db4o podržava indekse tipa B-stablo nad atributima pojedinog tipa objekta
 - Konfiguracija putem *Configuration* sučelja
 - Indeks se automatski kreira (ukoliko već ne postoji) ili uklanja kada se otvori baza podataka

```
// kreiraj indeks za atribut "godina" klase "Knjiga"  
Db4o.configure().objectClass("Knjiga").objectField("godina").indexed(true);  
  
// ukloni indeks  
Db4o.configure().objectClass("Knjiga").objectField("godina").indexed(false);
```

Promjena objektnog modela

- Često je potrebno mijenjati postojeći model aplikacije (redizajn, novi zahtjevi ...) – olakšano u db4o:
 - Uklanjanje atributa
 - Novi objekti pohranjuju se u novom formatu
 - U pohranjenim objektima uklonjeni atribut se ignorira
 - Dodavanje atributa
 - Novi objekti pohranjuju se u novom formatu
 - U pohranjenim objektima dodani atribut postavlja se na *null*
 - Promjena tipa podataka atributa
 - Automatska pretvorba podataka ako su stari i novi tip kompatibilni
 - Preimenovanje atributa ili klase
 - Moguće kroz sučelje za konfiguraciju db4o (*Configuration*)

```
// preimenovanje klase "Knjiga" u "Knjiga2"  
Db4o.configure().objectClass("Knjiga").rename("Knjiga2");
```

Objektno-relacijsko preslikavanje

- Objektno orijentirane baze nisu dovoljno pouzdane niti standardizirane za većinu aplikacija
- Ipak je najbolja kombinacija:
 - Relacijska baza podataka
 - OO programski jezik za razvoj aplikacije
- **Objektno-relacijsko preslikavanje** rješava problem objektno-relacijske neusklađenosti između relacijskog modela baze i objektnog modela aplikacije
- Temelji se na automatiziranju pretvorbe iz jednog modela u drugi koristeći meta-podatke za preslikavanje
- Uobičajen pristup u današnjim aplikacijama
- Najpopularniji okviri za objektno-relacijsko preslikavanje:
 - Hibernate, Ibatis , JPA (Java)
 - NHibernate, Linq (.NET)
 - Ruby on Rails (Ruby)

Hibernate – meta-podaci za preslikavanje

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

<hibernate-mapping>
<class name="Autor" table="autor">
    <id name="jmbg" column="jmbg">
        <generator class="native" />
    </id>
    <property name="ime" column="ime"/>
    <property name="prezime" column="prezime"/>
    <set name="knjige" table="autor_knjiga" cascade="all">
        <key column="jmbg" />
        <many-to-many column="oznakaKnjiga" class="Knjiga" />
    </set>
</class>
</hibernate-mapping>
```

- Preslikavanja atributa objekata u kolone tablice
- Preslikavanje veza između entiteta
- Transparentan dohvat podataka iz baze, odnosno pohranjivanje – nema SQL-a i "ručnog" preslikavanja podataka u objekte

Hibernate – HQL (*Hibernate Query Language*)

```
// dohvati sve knjige napisane između 1350. i 1750. godine
List<Knjiga> knjige = (List<Knjiga>) session.createQuery("from Knjiga knjiga
    where knjiga.godina >= 1350 and knjiga.godina <= 1750").list();

for(Knjiga knjiga : knjige){
    // tek sada se dohvaćaju autori iz baze - lijeni dohvat
    List<Autor> autori = knjiga.getAutori();
}
```

- Moćan upitni jezik – dovoljno fleksibilan i jednostavan za korištenje
- Na način dohvata utječu meta-podaci za preslikavanje, podržava spajanje, agregatne upite, grupiranje, sortiranje ...
- Sličan SQL-u, ali koristi objektnu notaciju – Hibernate generira SQL iz HQL-a koristeći meta-podatke za preslikavanje
- Hibernate podržava još dva tipa upita koji se transformiraju u HQL:
 - Upiti prema primjeru (Query By Example – QBE)
 - Upiti prema kriterijima (Query By Criteria – QBC)
- Pojam "lijenog dohvata" (aktivacija u db4o):
 - Ako atribut objekta nije dohvaćen iz baze – dohvaća se pri prvom korištenju

Hibernate – unos, izmjena, brisanje

```
Autor autor = new Autor("3112981335065", "Marica", "Bjelinski");

// unos novog autora
session.save(author);

// dodavanje nove knjige autoru
autor.addKnjiga(new Knjiga("200ab", "Cvijeće jadrana"));

// izmjena autora
session.save(author);

// brisanje autora
session.delete(author);
```

- Perzistencija po dohvatljivosti kod unosa i izmjene
- Moguće definirati kaskadne operacije u meta-podacima za preslikavanje
- Hibernate generira SQL naredbe i izvodi ih pravilnim redoslijedom vodeći računa o referencijskom integritetu

Korisni linkovi i literatura

- <http://OOSUBP.org>
 - Sve o OO bazama
 - Predavanja, članci, knjige, projekti ...
- <http://www.db4o.com>
 - Sve o db4o
 - Priručnici i tutorijali
 - *Open source* projekti za učenje db4o
 - Razni pomoćni programi – ObjectManager
- The Definitive Guide to Db4o , Stefan Edlich, Jim Paterson, Henrik Horning, Reidar Horning
 - Koncepti OO baza
 - Detaljan opis db4o
 - Može se čitati online