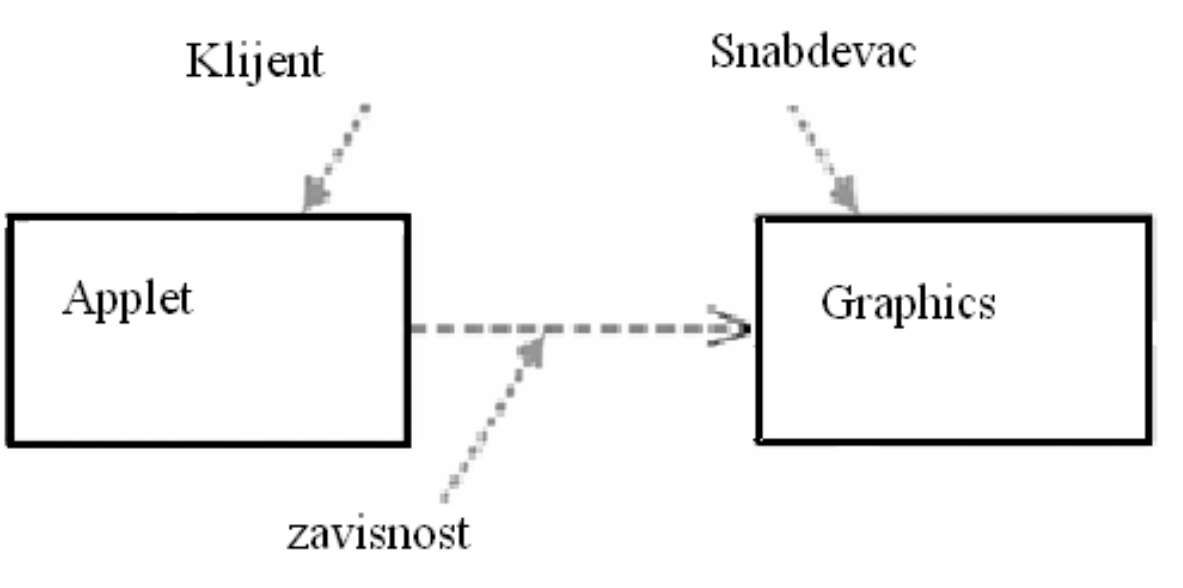
Napredni elementi u klasnim dijagramima

Zavisnost između elemenata

Zavisnost postoji između 2 elementa ako promene u jednom elementu tzv. snabdevaču (*supplier*), mogu izazvati promene u drugom element tzv. klijentu (*client*). .
Česte su zavisnosti između klasa (jedna klasa šalje poruku drugoj, jedna klasa koristi drugu kao deo njenih podataka, ...)

U UML-u zavisnost se može izraziti između ma kakvih elemenata. Za to se koristi isprekidana linije koja ima jedno usmerenje.

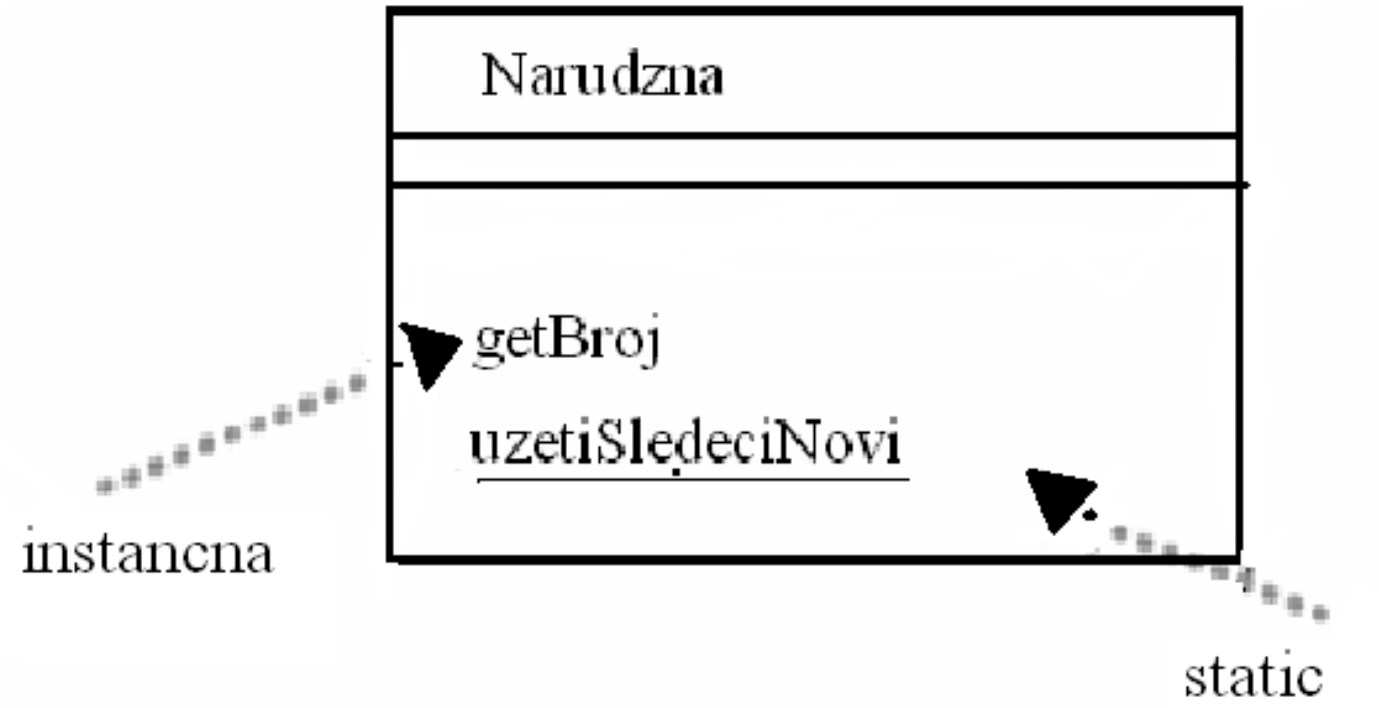
Evo jednog konkretnog primera opisa zavisnosti:



Mogu postojati razne vrste zavisnosti kao što su: poziva, koristi, kreira, ...

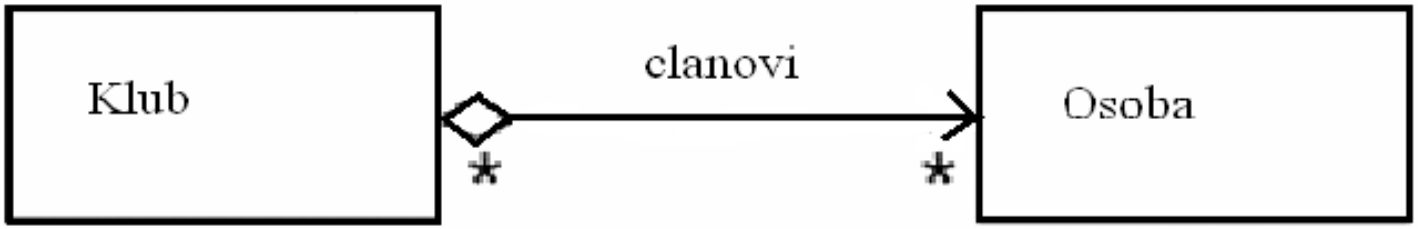
Statičke komponente

Statičke osobine su okarakterisane ključnom rečju static. One se obeležavaju podvlačenjem kao na sledećoj slici:



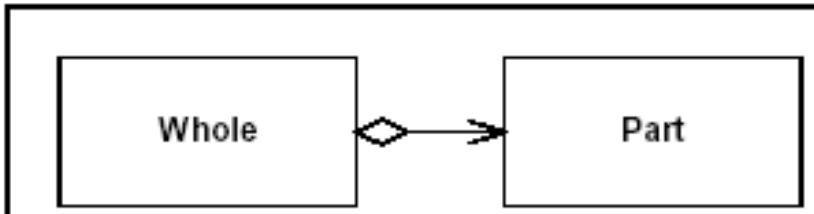
Agregacija i kompozicija

Agregacijom se opisuje relacija pripadnosti. (Motor pripada kolima, točkovi pripadaju kolima, ...) Problem nastaje, jer je teško praviti razliku između agregacije i asocijacije.

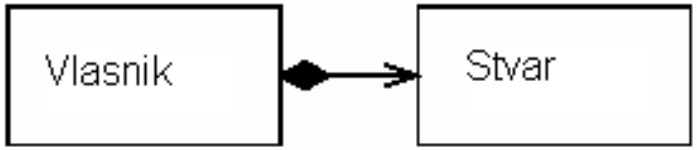


Kompozicija kazuje da instanca može pripadati samo jednom vlasniku. (Klasa može biti komponenta raznih drugih klasa, ali instanca, ne!)

Primeri sa Java-kôdom



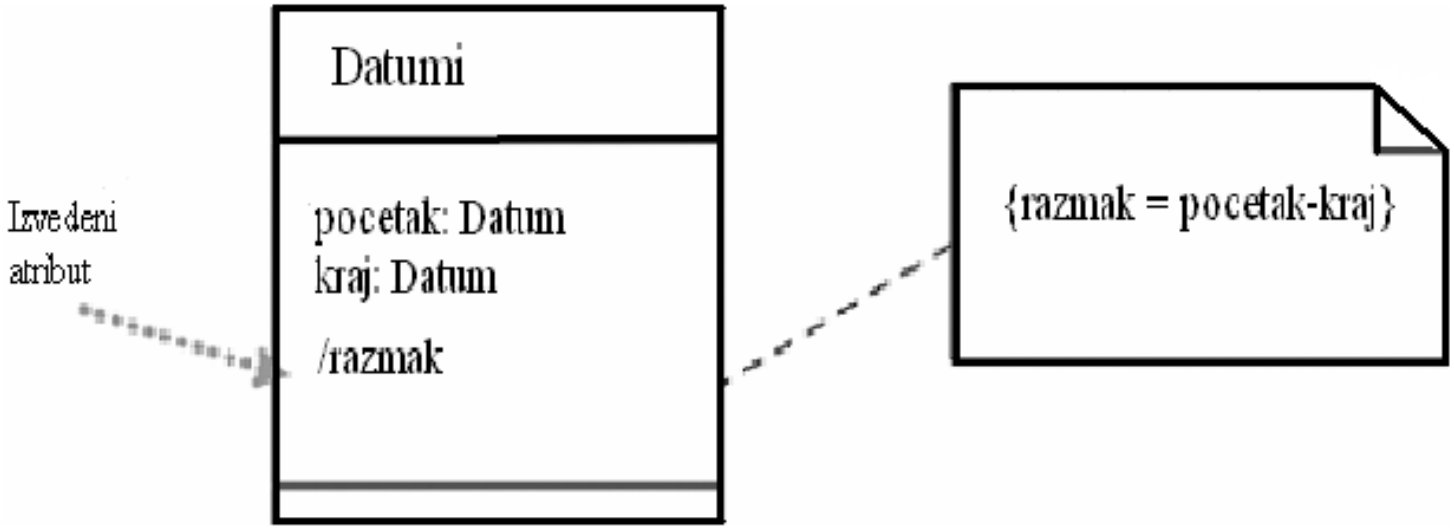

```
public class Whole
{
  private Part itsPart;
}
```




```
public class Vlasnik
{
  public  Stvar  njegovaStvar;
}
```

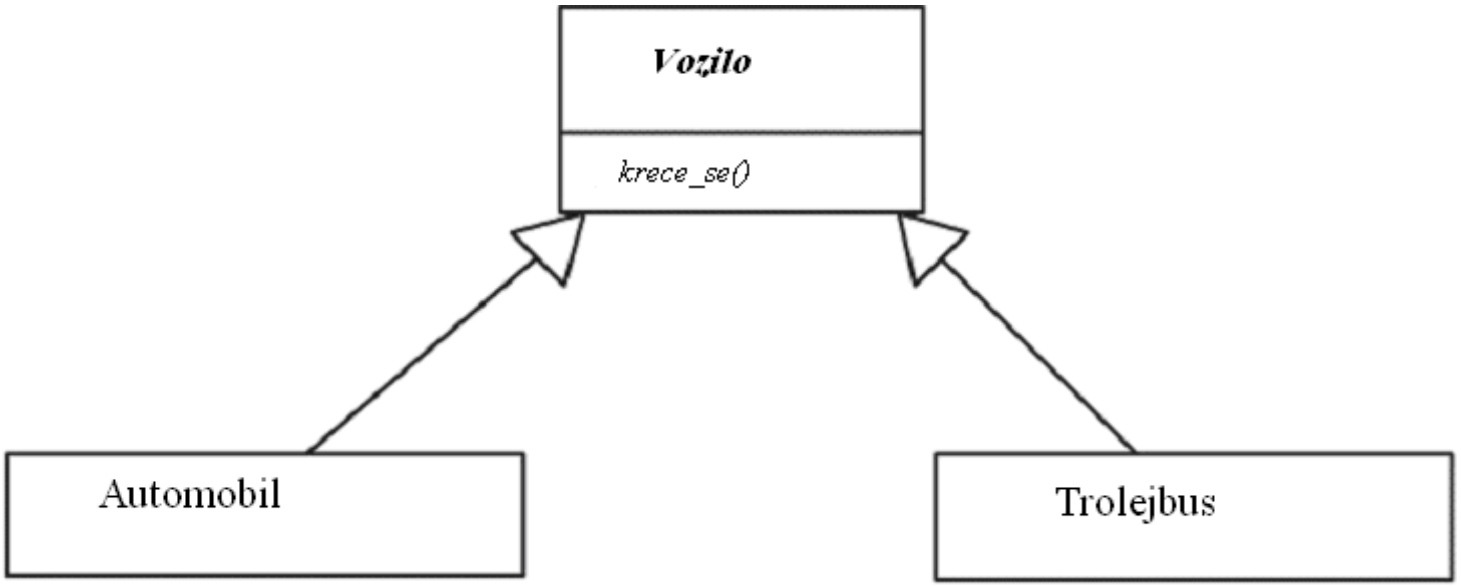
Izvedena svojstva

Izvedena svojstvo možemo tretirati kao podatak dobijen na osnovu već raspoloživih podataka. Izvedeno svojstvo (podatak) označava se kosom crtom kao na slici.



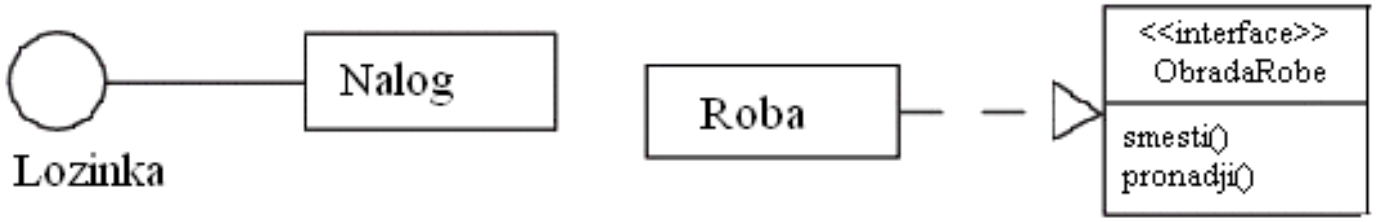
Apstraktne klase i interfejsi

Uobičajen način za predstavljanje apstraktne klase je da se koriste iskošena slova (italic) u imenu ili oznaka {abstract}.

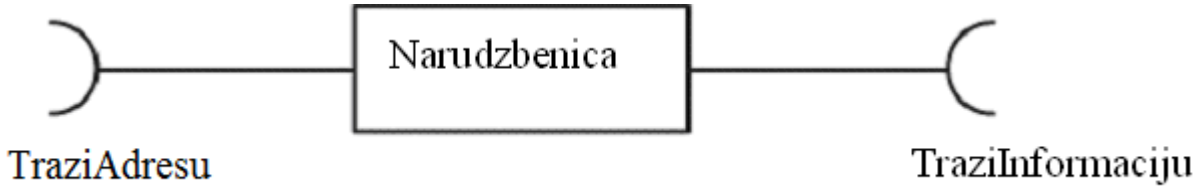


Slično, za interfejs se može koristiti reč <<interface>>.
Klasa može da zahteva ili da obezbeđuje interfejs.
Klasa obezbeđuje interfejs ako ga impementira. To se može opisati na 2 načina u UML-u.

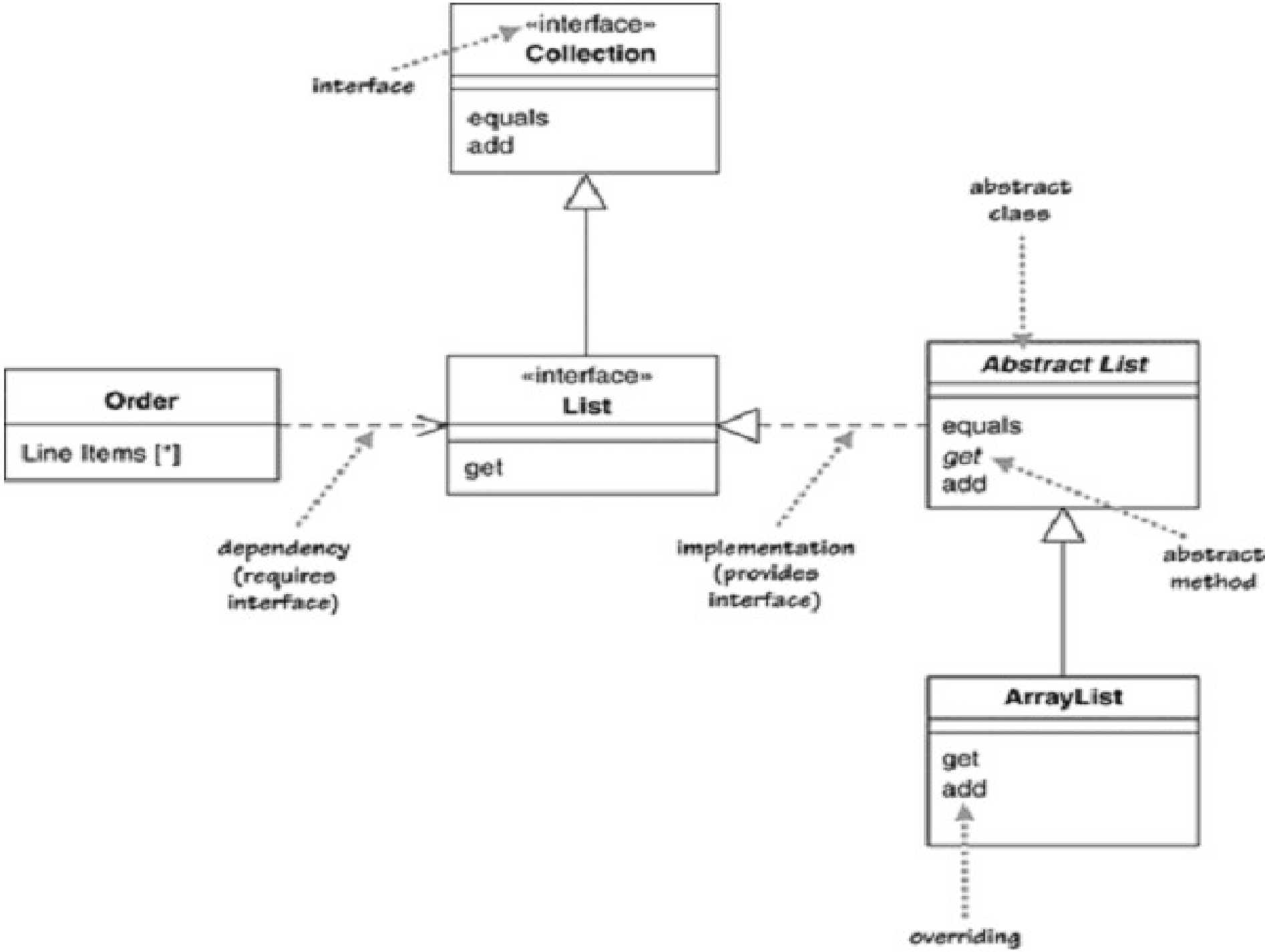
U sledećim primerima klasa nalog obezbeđuje interfejs Lozinka, a klasa Roba, interfejs ObradaRobe.



Klasa zahteva interfejs ako joj je potebna instanca tipa interfejsa da bi radila. To se izražava tzv. sklopka-simbolom, kao na sledećoj slici:

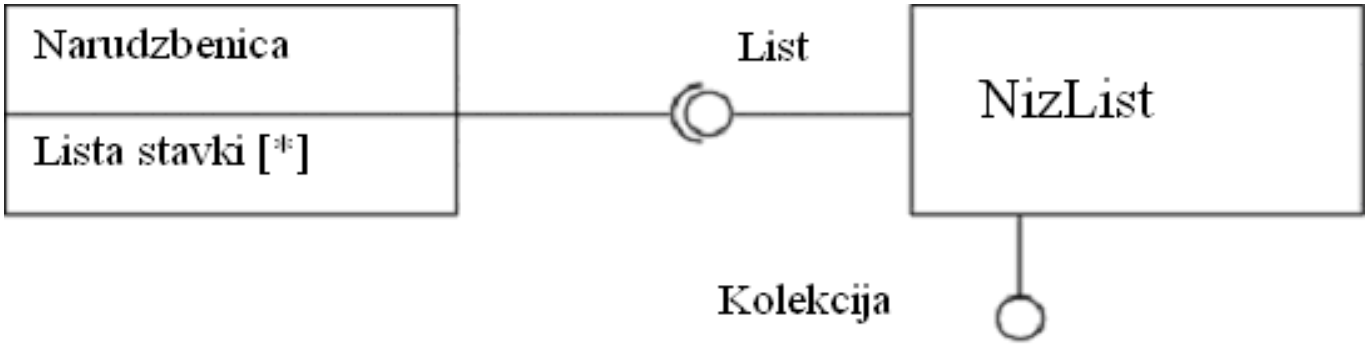


U sledećem primeru klasa Order zavisi od List-interfejsa, a abstraktna klasa AbstractList, obezbeđuje ovaj interfejs.

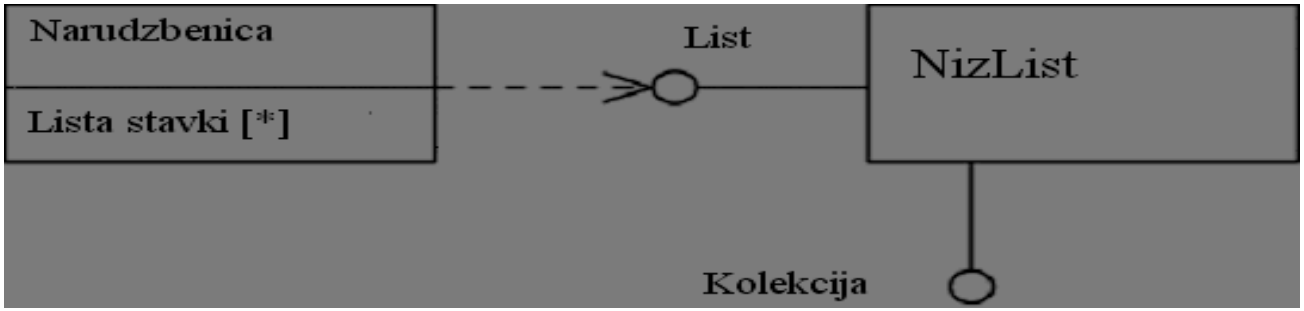

«interface»
Collection
equals
add
interface
abstract
class
Order
Line Items [*]
«interface»
List
get
Abstract List
equals
get
add
dependency
(requires
interface)
implementation
(provides
interface)
abstract
method
ArrayList
get
add
overriding

Na sledećoj slici je prikazana druga notacija interfejsa.

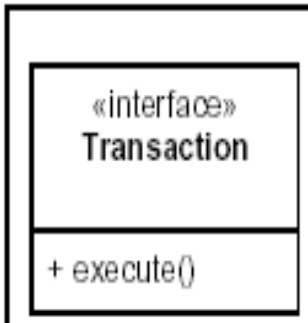
Činjenica da klasa NizList implemetira interfejse List i Kolekcija je prikazana kružićem. Činjenica da klasa Narudzbenica zahteva interfejs List je označena sklopka-ikonicom (novi grafički simbol u UML 2).



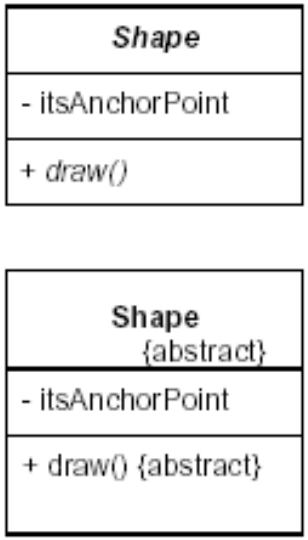
U starijim verzijama UML-a to je opisivano na sledeći način:



Primeri sa Java-kôdom




```
interface Transaction
{
  public void execute();
}
```




```
public abstract class Shape
{
  private Point itsAnchorPoint;
  public abstract void draw();
}
```

Klasifikacije

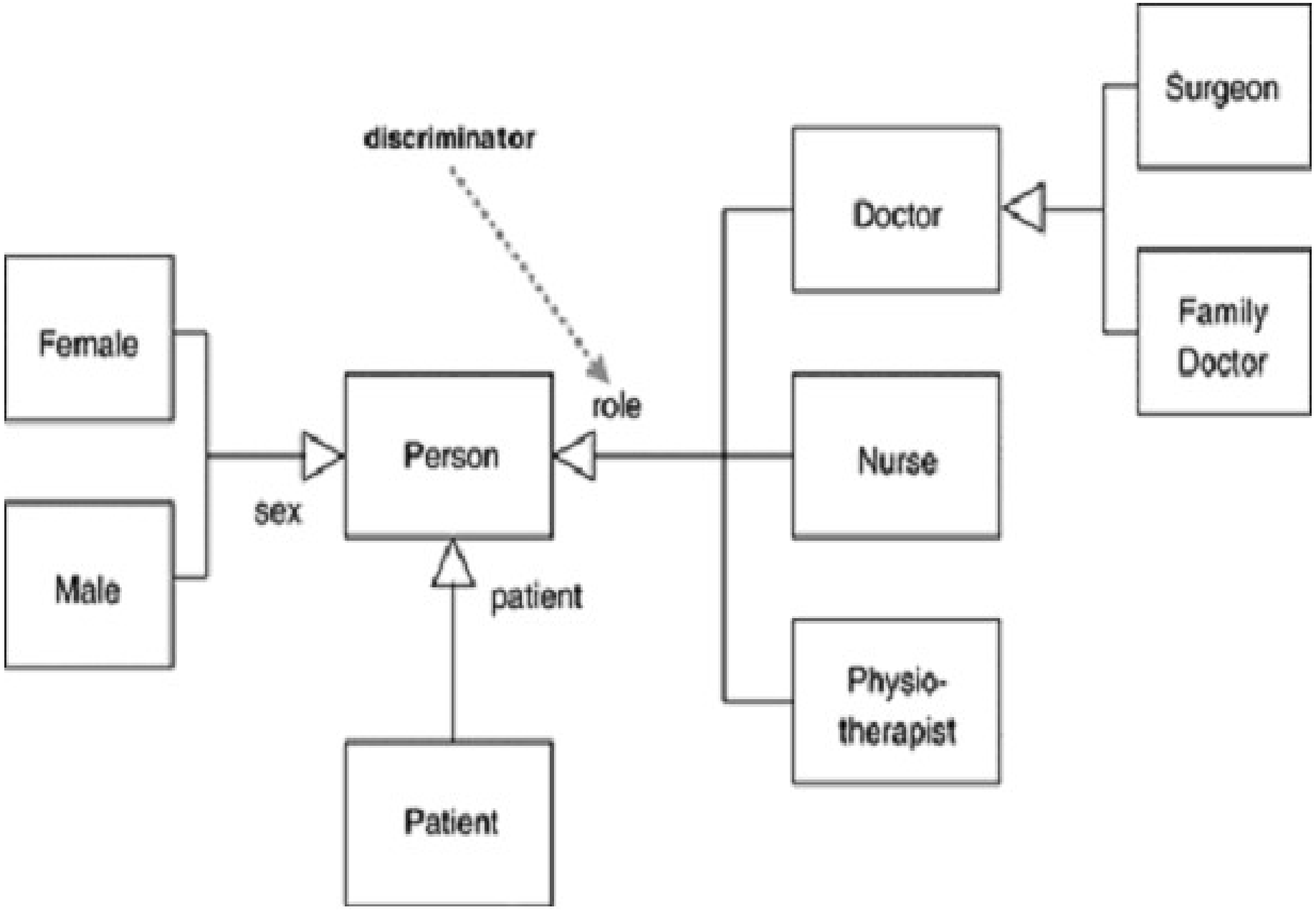
Klasifikacija se odnosi na relaciju između objekta i njegovog tipa. Uglavnom jedna instanca pripada jednoj klasi.

U jednostrukoj klasifikaciji jedan objekt pripada jednom tipu.

U višestrukoj klasifikaciji jedan objekat može pripadati većem broju klasa koje nisu povezane nasleđivanjem.

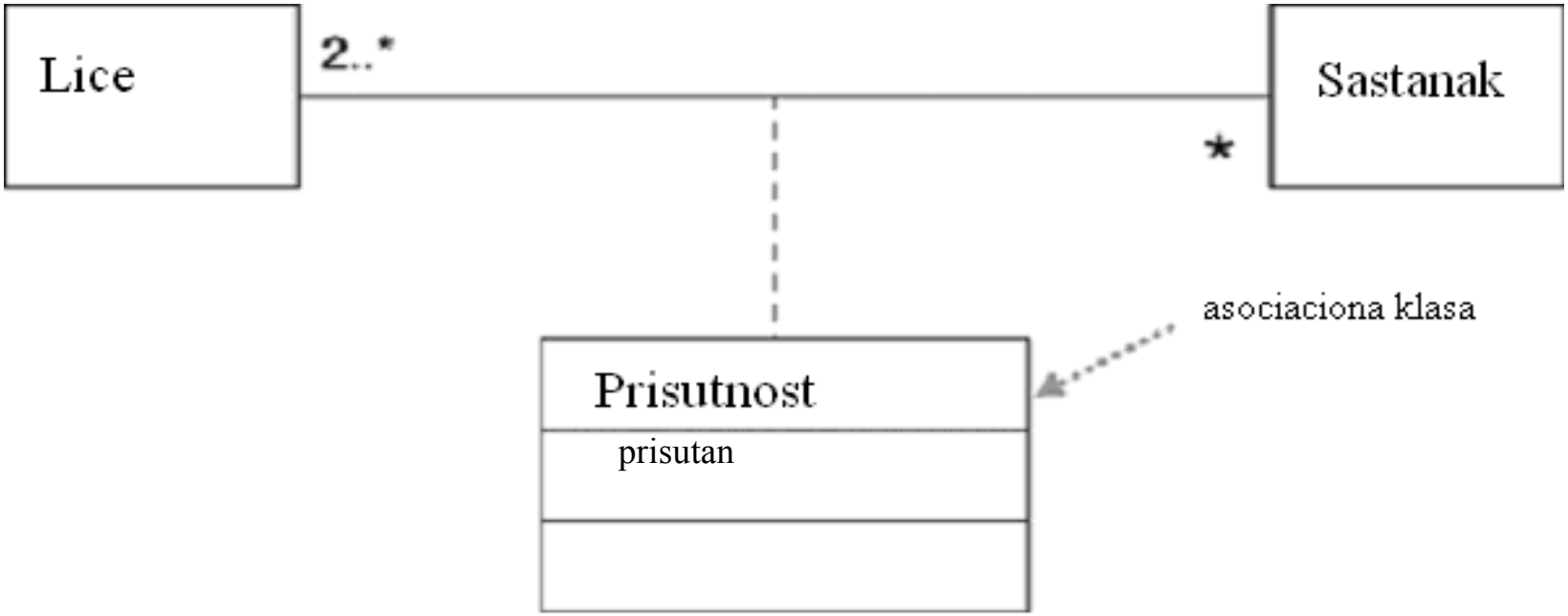
Razlikuje se od višestrukog nasleđivanja. Kod višestrukog nasleđivanja jedna klasa može imati više natklasa. Ali jedinstvena klasa mora biti za svaki objekat. Višestruka klasifikacija dozvoljava postojanje više klasa za jedan objekat, bez definisanja specifčne klase za to.

U sledećem primeru Person može biti objekat različitih klasa.

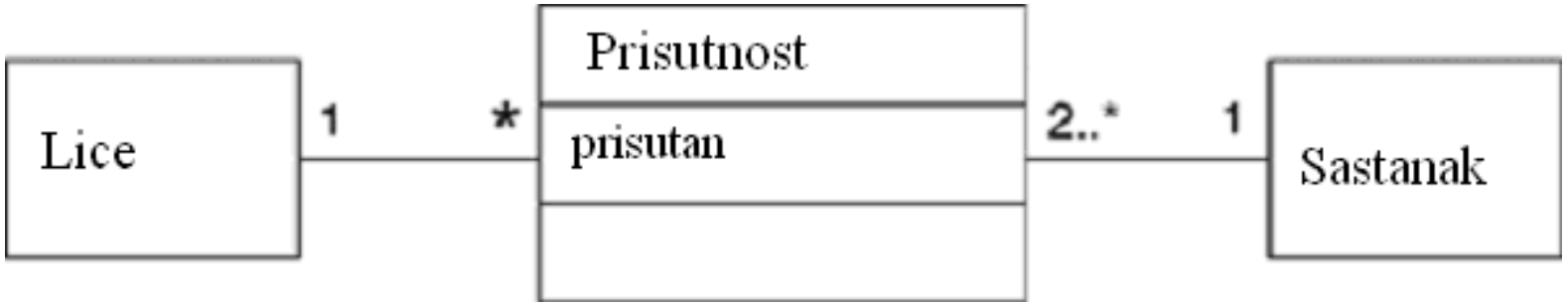

Surgeon
discriminator
Doctor
Female
Family
Doctor
role
Person
Nurse
sex
Male
patient
Physio-
therapist
Patient

Asocijaciona klasa

Asocijaciona klasa dozvoljava da se dodaju atributi, operacije i druge osobine za asocijacije. Sa slike se vidi da Lice može prisustvovati mnogim sastancima. Pridružujemo klasu Prisutnost gde se dodaje atribut za beleženje informacije o prisutnoti na sastancima.



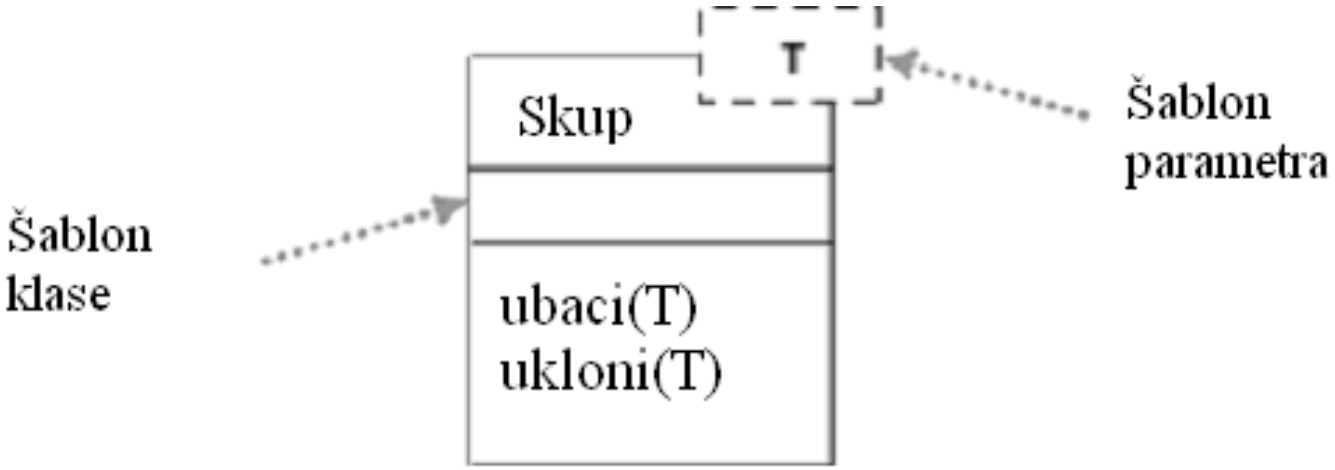
Još jedan način da se to izrazi je sledeći:



Asocijaciona klasa se može koristiti za opis privremenih relacija:

Šabloni (parametrizovane) klase

Jezik C++ ima mogućnosti opisa klasa-šablona. (Od verzije 1.5 to je moguće koristiti i u Javi preko generičkih klasa.) Na sledećoj slici je predstavljena klasa-šablon za skupove. Poseban simbol (pravougaonik sa slovom T) je korišćen za tip parametra u šablonu. Iz šablon-kase prave se izvedene klase.



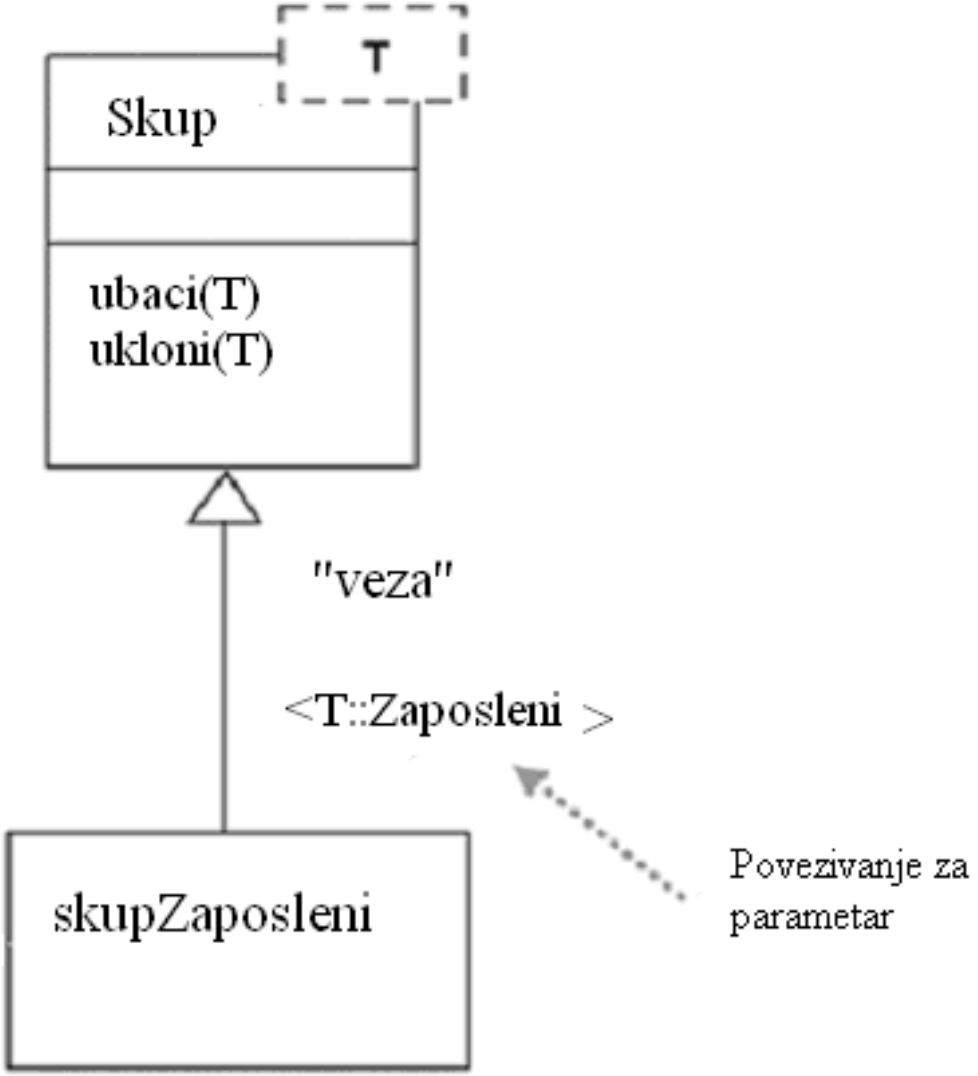
Na primer, u C++ se može definisati šablon-klasa:

```
class Skup <T> {
    void ubaci (T noviElement);
    void ukloni (T nekiElement);
    ..............
```

Tada se izvedena klasa (iz šablon-klase) može definisati na sledeći način:

```
Skup <Zaposleni> skupZaposleni;
```

Postoje posebne grafičke oznake za izvedene klase. Kako se to može uraditi za ovaj (konkretni) primer prikazano je na sledećoj slici.

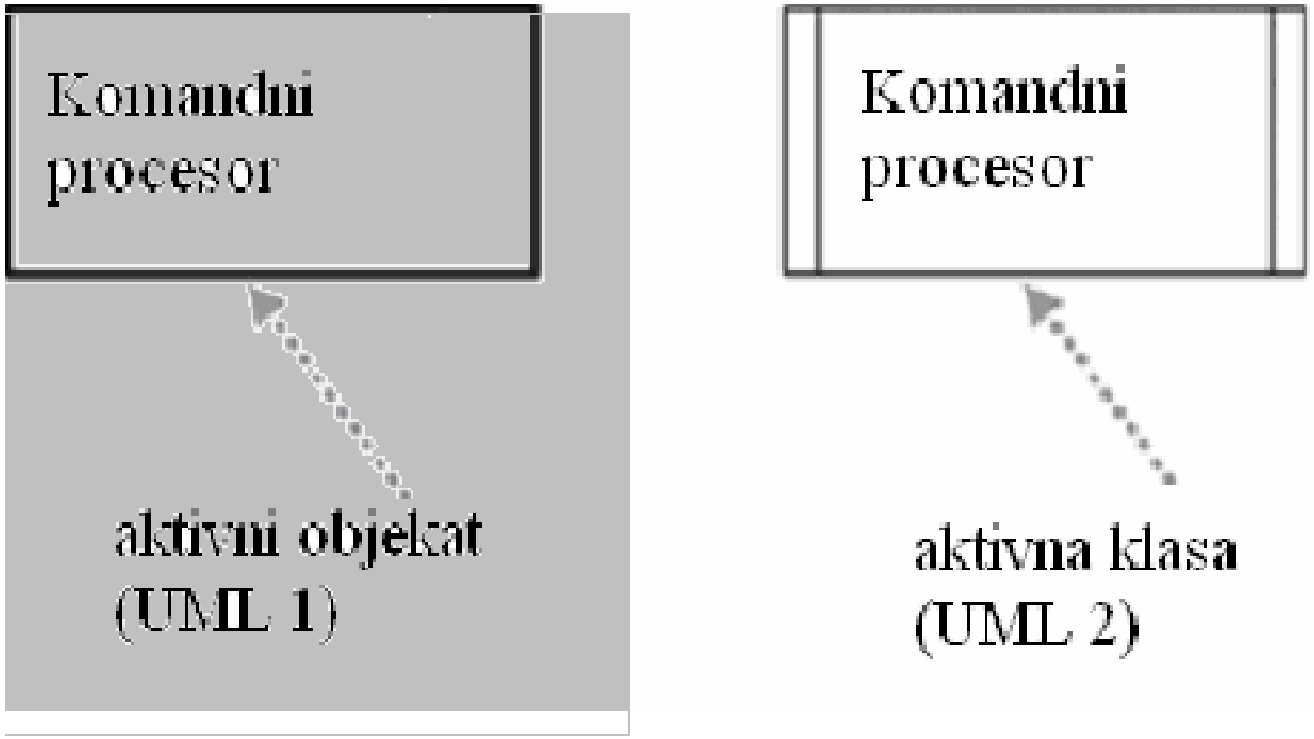

T
Skup
ubaci(T)
ukloni(T)
"veza"
<T::Zaposleni >
skupZaposleni
Povezivanje za
parametar

Enumeracije (nabrajanja)

Nabrojiv tip podatka se može koristiti u jezicima C++ i C#. Od verzije 1.5, može s ekoristititi i u programskom jeziku Java. Pomoću enumeracije prikazuje se fiksiran skup vrednosti. Te vrednosti nemaju nikakvo drugo svojstvo osim simboličko značenje. Prikazuju se kao klasa sa ključnom rečiju «enumeration» .

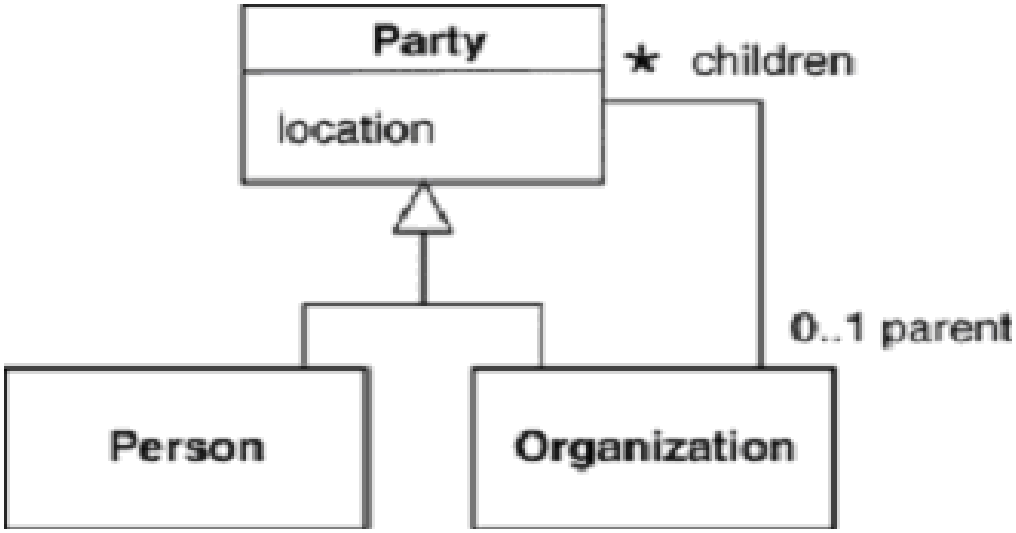
Aktivne klase

Aktivna klasa ima instance od kojih svaka izvršava i kontroliše sopstveni tred.

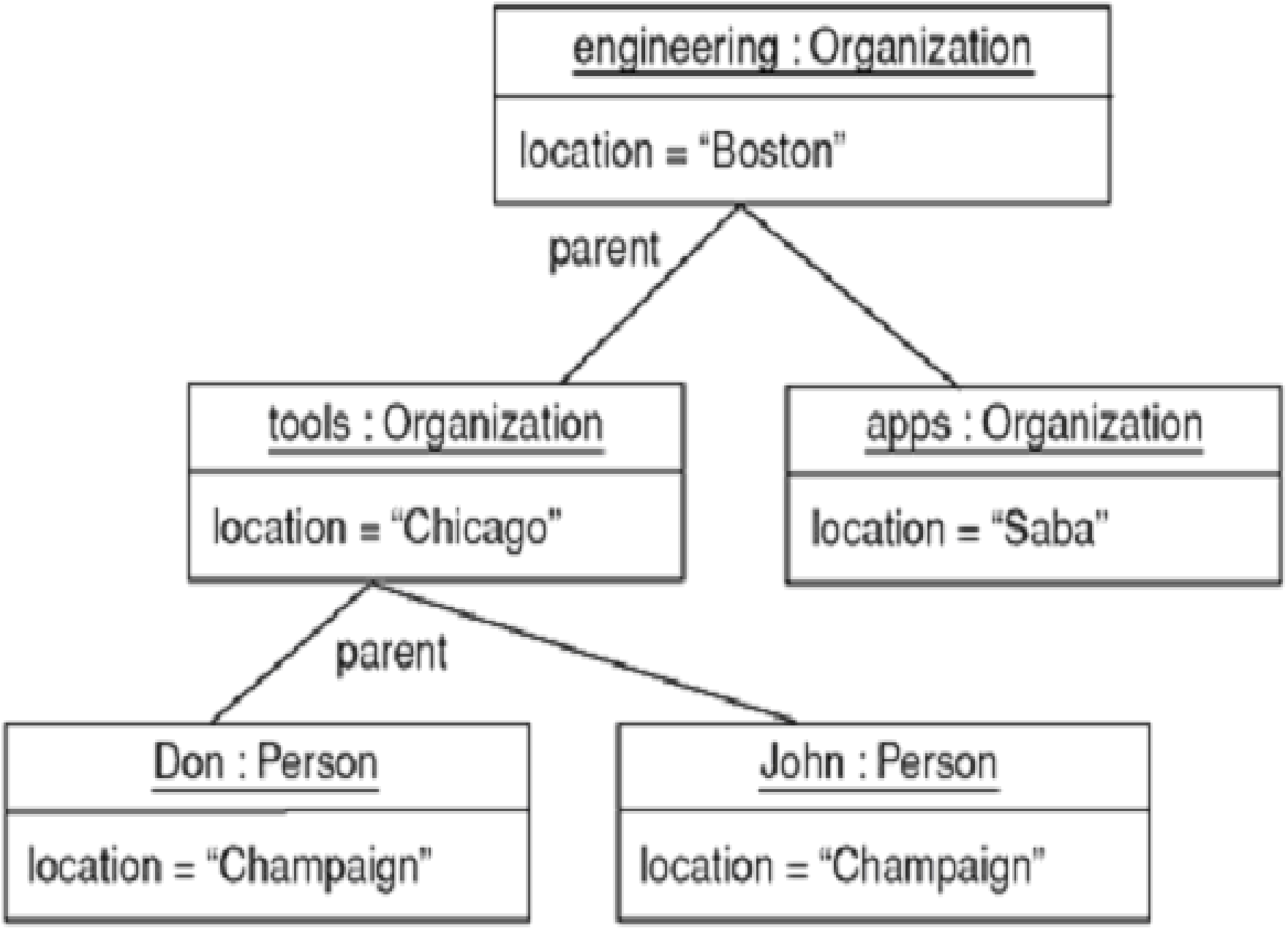


Objektni dijagrami

Objektni dijagram je snimak objekata sistema u jednom vremenskom trenutku. Često se nazivaju dijagrami instanci. Objektni dijagram se koristi za pokazivanje konfiguracija objekata. Za sledeću strukturu klasa:



objektni dijagram može ovako izgledati:


engineering : Organization
location = "Boston"
parent
tools : Organization
location = "Chicago"
apps : Organization
location = "Saba"
parent
Don : Person
location = "Champaign"
John : Person
location = "Champaign"

Imena u instancama su podvučena. Svako ime je oblika:

instance name : class name.

Treba da se navede vrednost za atribute i međusobne veze.

Elementi objektnog dijagrama su pre specifikacije instanci, negoli same instance. (Može se izostaviti vrednost obaveznih atributa!) Za specifikaciju instance može se reći da predstavlja delimično definisanu instancu.

S druge strane, za objektni dijagram može se reći da je to komunikacioni dijagram bez prisustva poruka.

Objektni dijagrami se koriste za prikaz međusobno povezanih objekata. Ako je strukturu teško razumeti kroz klasne dijagrame, pomažu objektni dijagrami.

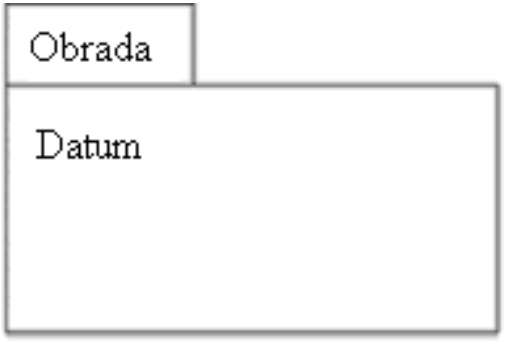
Paketni dijagrami

Paket je konstrukcija za grupisanje u UML-u. Paket dozvoljava da se uzmu bilo kakve konstrukcije UML-a i grupišu u jedinice višeg nivoa. Najčešće se koriste za grupisanje klasa, ali mogu se koristiti za grupisanja ma kakvih elemenata u UML-u.

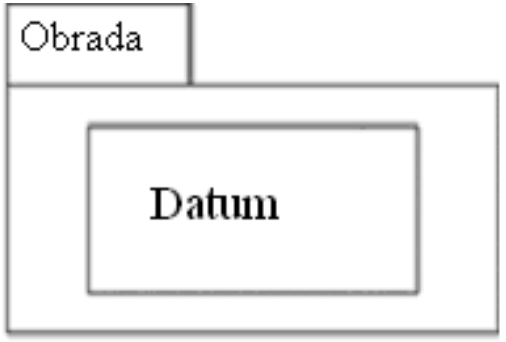
U UML-modelu svaka klasa je član jedinstvenog paketa. Paketi mogu biti članovi drugih paketa itd. Paket može sadržati podpakete i klase. Paketi odgovaraju pojmu **paketa** u Javi i pojmu **namespace** u C++ i .NET-u.

Paket se predstavlja ikonicom za folder. U samom paketu, klase mogu biti predstavljene na razne načine kao što pokazuje sledeća slika.

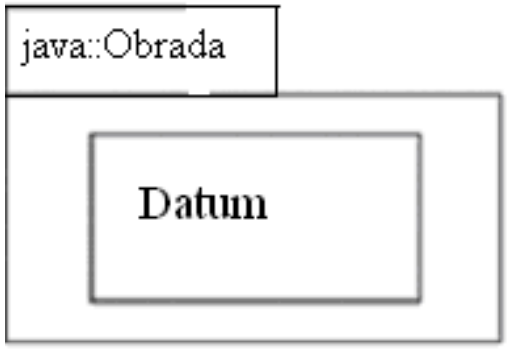
Kako se predstavlja vidljivost klasa u paketu? Kako se grupišu klase po paketima?



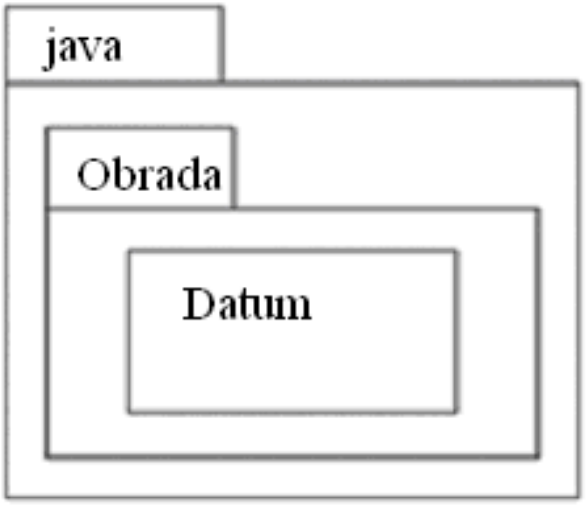
Sadrzaj prikazan u bloku



Sadryaj prikazan pomo'u dijagrama u bloku



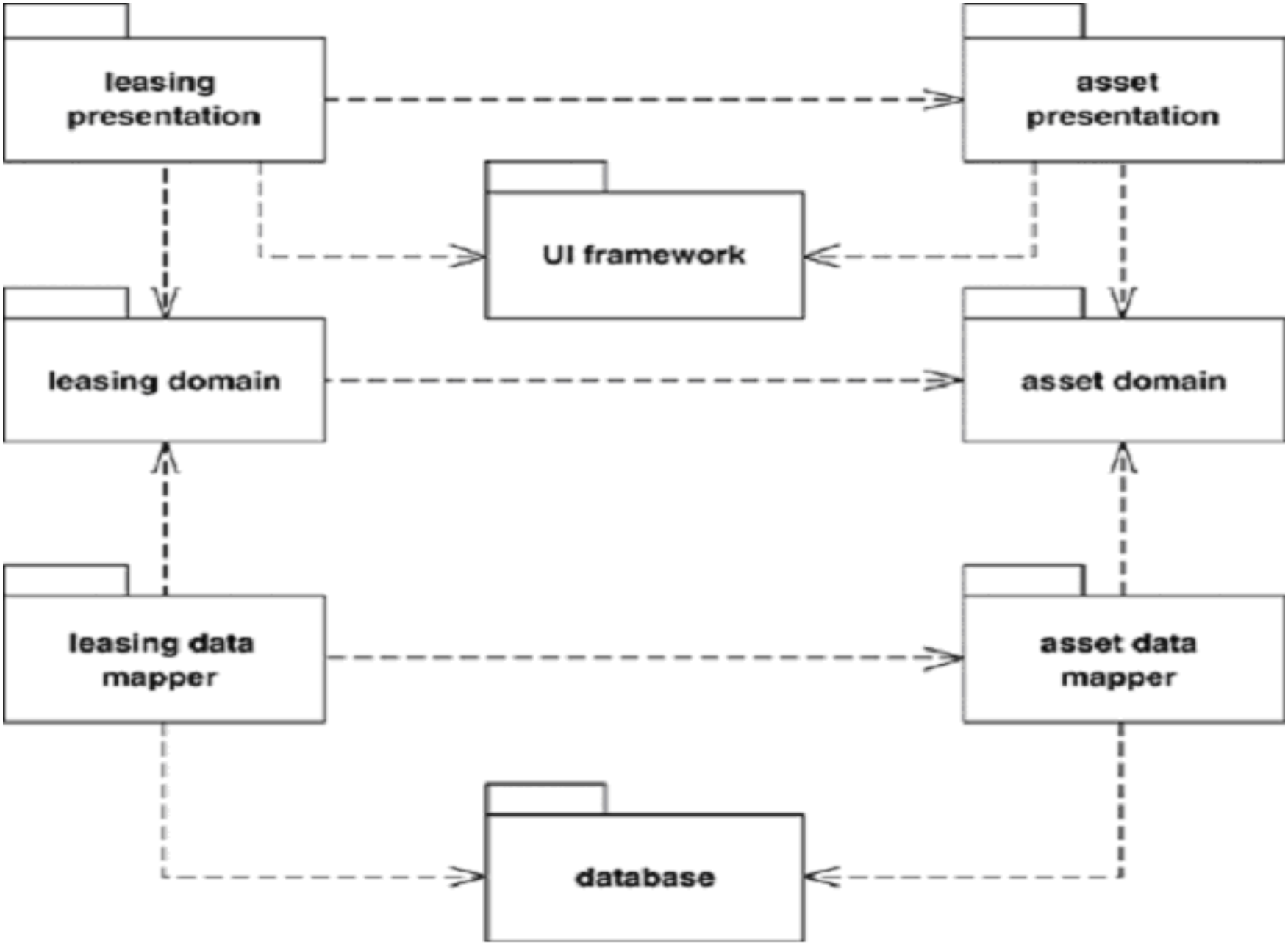
Pun prikaz imena paketa



Ugnjezdenji paketi

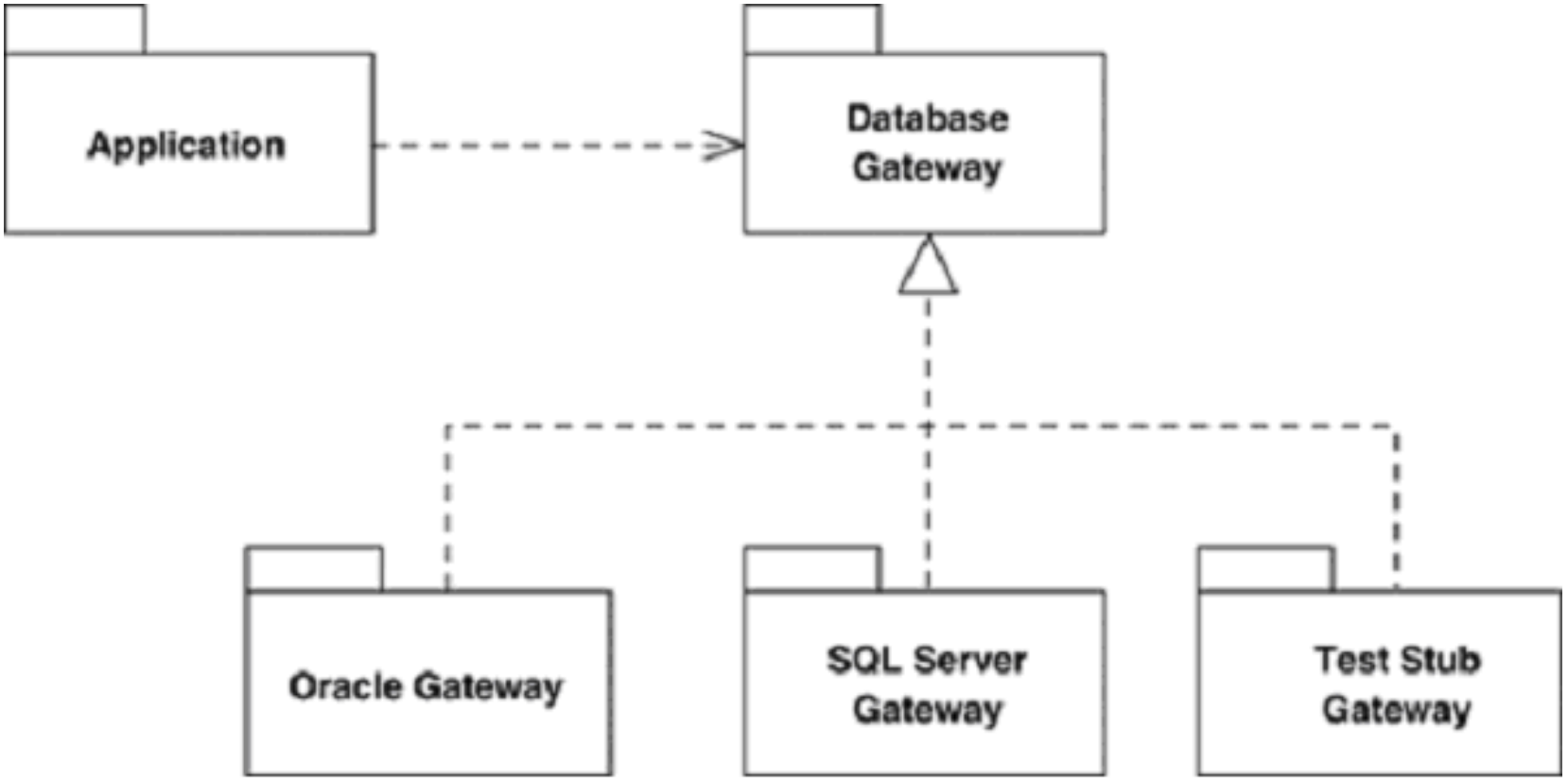
Zavisnost između paketa

Dijagramima se može opisivati zavisnost između paketa. Dva paketa su zavisna ako sadrže zavisne klase (odnosno ako postoji zavisnost između njihovih sadržaja).

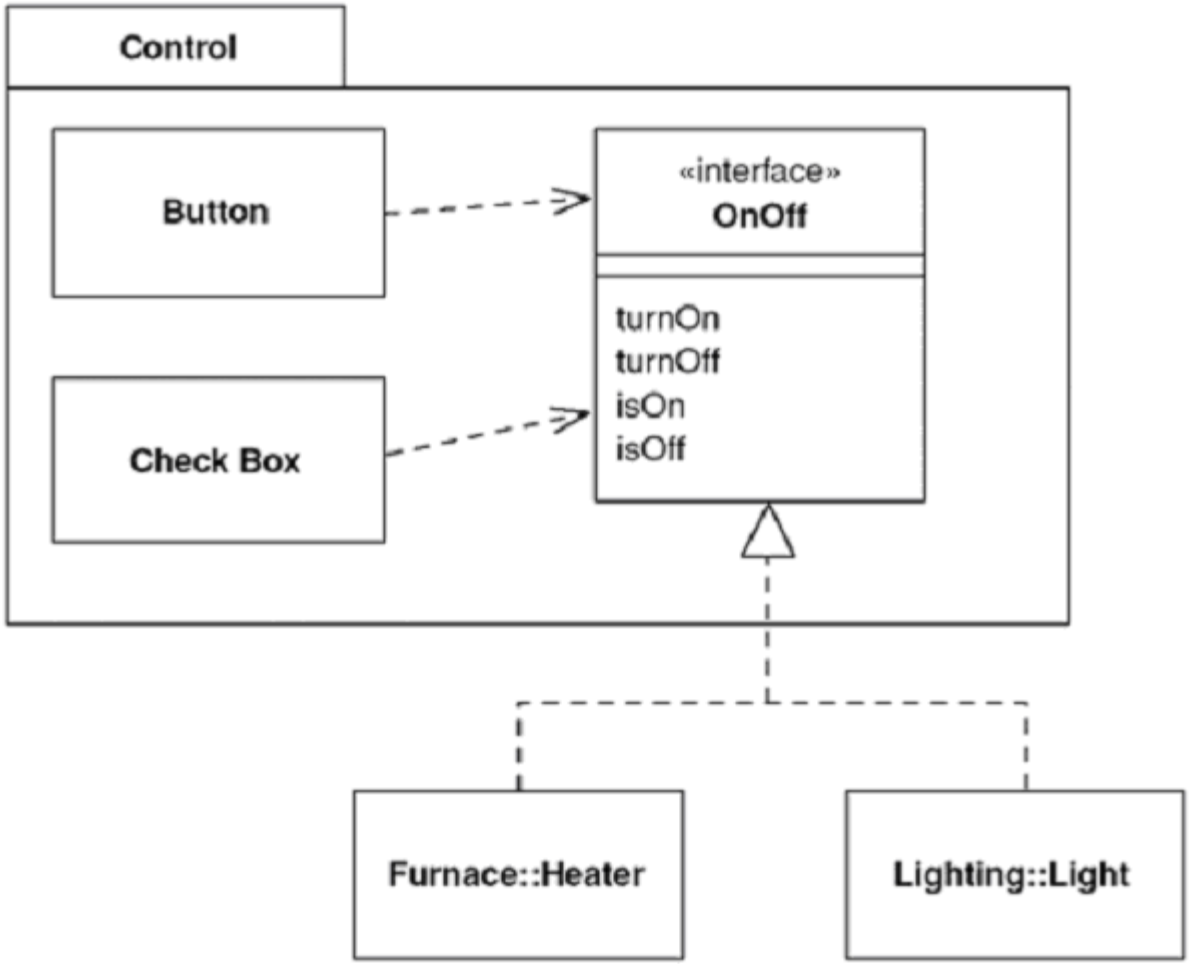

leasing presentation
asset presentation
UI framework
leasing domain
asset domain
leasing data mapper
asset data mapper
database

Implementacija paketa

Jedan paket može definisati interfejs koji se implementira u drugim paketima. To se može videti na sledećoj slici.



Još jedan primer implemetacije interfejsa preko paketa dat je na sledećoj slici.

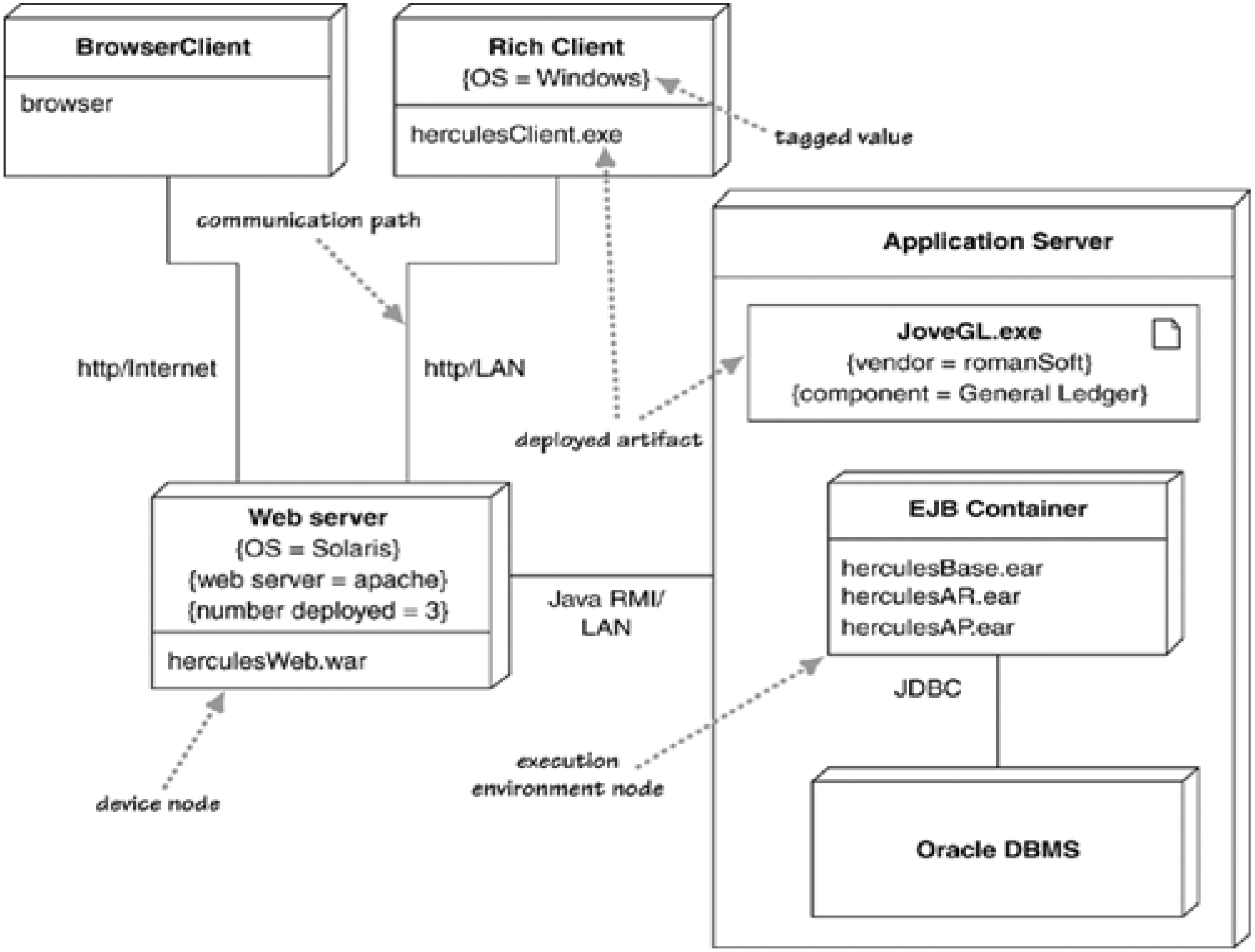


Razvojni dijagrami

Razvojni dijagrami pokazuju sistemsku fizičku postavku. Oni pokazuju koji delovi softvera se izvršavaju na kojim delovima hardvera.

Glavni elementi ovih dijagrama su čvorovi povezani komunikacionim putevima. Čvor je nešto što može biti domaćin softveru. Čvor može biti uređaj hardvera ili neka izvršna okolina (softver) koja opslužuje drugi softver.

Čvorovi obično sadrže fajlove.


BrowserClient
browser
Rich Client
{OS = Windows}
herculesClient.exe
tagged value
communication path
Application Server
http/Internet
http/LAN
JoveGL.exe
{vendor = romanSoft}
{component = General Ledger}
deployed artifact
Web server
{OS = Solaris}
{web server = apache}
{number deployed = 3}
herculesWeb.war
Java RMI/
LAN
EJB Container
herculesBase.ear
herculesAR.ear
herculesAP.ear
JDBC
Oracle DBMS
execution
environment node
device node