

Specifikacija zahtjeva

3/13

Zahtjevi

- ❑ **IEEE standard definira zahtjeve kao [IEEE Std 610.12-1990]**
 - (1) Uvjet ili sposobnost koje korisnik treba da bi riješio problem ili ostvario cilj. (2) Uvjet ili sposobnost koji mora posjedovati sustav ili komponenta sustava da bi zadovoljila ugovor, standard, specifikacije ili neki drugi ugovoreni dokument. (3) Dokumentiranu reprezentaciju uvjeta ili mogućnosti definiranih pod (1) ili (2). [IEEE Std 830-1998], [IEEE Std 610.12-1990]

Vrste zahtjeva

❑ Poslovni zahtjevi (zašto)

- ciljevi organizacije ili korisnički zahtjevi na višoj razini, opis problema koje treba riješiti
 - primjer: poslovna potreba, "Poboljšanje usluge postojećim klijentima i pridobivanje novih"
- sadržani u dokumentima u kojima se opisuje vizija i opseg projekta
 - primjer: poslovni zahtjev, "Omogućiti Internet prodaju"

❑ Korisnički zahtjevi (zahtjevi krajnjih korisnika)

- opisuju zadatke koje korisnik mora moći obaviti služeći se aplikacijama
- sadržani u opisima slučajeva korištenja, tj. opisima scenarija rada

❑ Funkcionalni zahtjevi (što)

- definiraju softversku funkcionalnost (očekivano ponašanje i operacije koje sustav može izvoditi) koju treba ugraditi u proizvod da bi omogućio korisnicima obavljanje njihovih zadataka
- posebno zanimljiva mogućnost programa (feature) – skup logički povezanih funkcionalnih zahtjeva koje korisniku omogućuju ispunjavanje poslovnih zahtjeva

❑ Nefunkcionalni zahtjevi (kako ili kako dobro)

- standardi, pravila i ugovori kojih se proizvod mora pridržavati, opisi vanjskih sučelja, zahtjevi na performance, ograničenja na dizajn i implementaciju, te svojstva kvalitete (preciziraju opis proizvoda navodeći karakteristike proizvoda u različitim dimenzijama koja su važne ili korisniku, ili graditelju)

❑ Prioriteti pojedinih elemenata

Primjeri zahtjeva vlasnika sustava

☐ Očekivana novčana ušteda

- Sustav mora biti tako koncipiran da prava na subvencioniranu prehranu može koristiti samo student koji ih je stekao i da ih može koristiti samo u svrhu prehrane.

☐ Sustav mora onemogućiti:

- korištenje subvencije od strane osoba koje nemaju na to pravo
- zaradu ilegalnih posrednika
- korištenje subvencije za druge svrhe osim prehrane
- naplatu usluga koje nisu pružene

☐ U idealnom slučaju zahtjevi vlasnika podudaraju se s poslovnim ciljevima!

Primjeri zahtjeva krajnjih korisnika

☐ **Prehrana kod bilo kojeg pružatelja usluga**

- Novi sustav mora omogućiti da student ostvaruje svoje pravo kod bilo kojeg pružatelja usluge subvencionirane prehrane. Dosadašnja praksa je bila da svaki pružatelj usluga izdaje svoje bonove koji se mogu koristiti samo u određenim restoranima.

☐ **Ukinuti plaćanje unaprijed**

- Treba izbjeći bilo kakvo plaćanje od strane studenata za potrebe ostvarivanja prava, a posebice unaprijed.

☐ **Ukloniti nepotrebne postupke za ostvarivanje prava**

- Sustav mora biti tako koncipiran da kad se studentu jednom zavedu prava na matičnoj ustanovi nisu potrebna nikakva daljnja dokazivanja za ostvarivanje prava kod pružatelja usluga.

☐ **Smanjiti rizik gubitka ostvarenih prava**

- Sustav mora onemogućiti zlorabu stečenih prava.

☐ **Lakše ostvarivanje ostalih prava iz studentskog standarda**

- Npr. javni prijevoz po povlaštenoj cijeni, kazališta, kina, smještaj u studentskim domovima, student-servis, itd.

Primjer nepotpunog zahtjeva

☐ **"Proizvod će dostaviti statusnu poruku u redovitim intervalima ne manjim od 60 sekundi."**

- Što je statusna poruka i pod kojim uvjetima će biti dostavljena?
- Koliko dugo ostaje vidljiva?
- Koji dio proizvoda će dostaviti poruku?
- Koliko dosljedni intervali moraju biti?

☐ **Zahtjev, preciznije i detaljnije**

- Modul za nadzor će ispisivati statusnu poruku u za to određeni dio sučelja.
- Poruka će se ažurirati svakih 60 sekundi (plus minus 10 sekundi) nakon što započne izvođenje pozadinskog zadatka i bit će vidljiva cijelo vrijeme.
- Ukoliko se pozadinski zadatak izvodi normalno, modul za nadzor će ispisivati postotak obavljenog posla.
- Modul za nadzor će ispisati "Zadatak obavljen." nakon što se zadatak obavi.
- Modul će ispisati poruku o pogrešci ukoliko dođe do zastoja u izvođenju.
- Problem je rastavljen u više zahtjeva jer će svaki zahtijevati posebno testiranje.
- Ukoliko je više zahtjeva grupirano u jedan lakše je previdjeti neki od njih tijekom izrade ili testiranja.

☐ **Primijetiti da u zahtjevu nije detaljno opisano kako će se poruka i gdje ispisivati. To će biti odlučeno tijekom dizajna.**

Primjer neostvarivog zahtjeva

☐ "Proizvod će se trenutno prebaciti između ispisivanja i skrivanja znakova koji se ne mogu tiskati."

- Računala ništa ne mogu napraviti trenutno te je ovaj zahtjev neostvariv.
- Da li programska podrška sama odlučuje kad će se prebaciti iz jednog stanja u drugo ili je to inicirano akcijom korisnika?
- Na koji dio teksta će se primijeniti promjena prikaza: da li samo označeni tekst, cijeli dokument ili nešto treće?
- Nejednoznačnost: da li su "znakovi koji se ne mogu tiskati" skriveni znakovi, posebne oznake ili kontrolni znakovi?

☐ Bolji zahtjev:

- "Korisnik će posebno dogovorenom akcijom, odabrati da li će se HTML oznake u trenutno otvorenom dokumentu prikazivati ili ne."
- Sad je jasno da je riječ o HTML oznakama te da korisnik mora obaviti nekakvu akciju, ali nije točno navedeno kakvu (npr. kombinacija tipki), što se prepušta dizajnerima.

Primjer neodređenog zahtjeva

- ❑ **"Parser će brzo generirati izvješće o pogreškama HTML oznaka, koje omogućava brzi ispravak pogrešaka kada program koriste početnici u HTML-u."**
 - Riječ "brzo" je neodređena.
 - Nije definirano što tvori izvješće i to čini zahtjev nekompletnim.
 - Kako se ovjerava zahtjev? Pronaći nekoga tko se smatra početnikom u HTML-u i zatim vidjeti kako brzo će, uz pomoć izvješća, ispraviti pogreške?
 - Kada se generira izvješće?
- ❑ **Bolje:**
 - Nakon što je HTML analizator obradio datoteku generirat će izvješće koje sadrži broj linije i tekst pronađenih HTML pogrešaka, te opis svake pogreške.
 - Ukoliko nema pogrešaka prilikom analize, neće se generirati izvješće.

Postavljanje prioriteta

☐ **Nužno svojstvo - Da li vlasnik sustava nešto stvarno mora imati?**

- Postoji tendencija da se previše zahtjeva proglasi nužnim!
- Po definiciji, ako sustav ne uključuje nužne zahtjeve, taj sustav ne može ispuniti svoju svrhu.
- Treba testirati svaki zahtjev koji se smatra nužnim i probati ga rangirati.
 - Ako se zahtjev može rangirati onda nije obavezan!
 - Potpuno obvezni zahtjevi se ne mogu rangirati jer su nužni za prvu verziju sustava!

☐ **Poželjno svojstvo - Funkcije koje korisnik želi na kraju imati**

- Ranije verzije sustava mogu pružiti (ne potpunu) funkcionalnost bez tih zahtjeva.
- Poželjni zahtjevi mogu i trebaju biti rangirani.

☐ **Neobvezna svojstva - Proizvoljni zahtjevi**

- Svojstva i mogućnosti bez kojih se može.
- Iako bi ih lijepo bilo imati, to nisu pravi zahtjevi.
- Ovi zahtjevi također mogu biti rangirani.

Dokumentiranje analize (zahtjeva)

❑ Definicija zahtjeva (Requirements Definition)

- izjava o stanju i ograničenjima sustava te potrebama
- narativni dokument namijenjen korisniku ili ga piše korisnik
 - poslovni i korisnički zahtjevi te njihovi prioriteti
 - uočeni problemi, ključne pretpostavke i preporuke rješenja

❑ Specifikacija zahtjeva (Requirements Specification)

- često se naziva i funkcionalnom specifikacijom
- strukturirani dokument s detaljnim opisom očekivanog ponašanja sustava
- namijenjen ugovarateljima i izvoditeljima razvoja
- ugradbeno nezavisan pogled na sustav
 - funkcionalni i nefunkcionalni zahtjevi te njihovi prioriteti
 - model organizacijske strukture (strukturni dijagrami)
 - opis protoka dokumenata (dijagrami toka)
 - model procesa (dijagram toka podataka)
 - konceptualni model podataka (dijagram entiteti-veze)

❑ Primjeri:

-  Firma-Zahtjevi.doc (SpecifikacijaZahtjeva.dot)

Predložak dokumenta specifikacije zahtjeva

1. Uvod

- 1.1 Namjena
- 1.2 Konvencije dokumenta
- 1.3 Tko treba čitati dokument i savjeti za čitanje dokumenta
- 1.4 Opseg proizvoda
- 1.5 Reference

2. Sveobuhvatni pregled

- 2.1 Kontekst proizvoda
- 2.2 Funkcije proizvoda
- 2.3 Kategorije korisnika i svojstva
- 2.4 Okruženje u kojem se izvodi proizvod
- 2.5 Ograničenja dizajna i ugradnje
- 2.6 Pretpostavke i ovisnosti

3. Zahtjevi za sučeljem

- 3.1 Korisničko sučelje
- 3.2 Hardversko sučelje
- 3.3 Softversko sučelje
- 3.4 Komunikacijsko sučelje

4. Svojstva sustava

- 4.x Svojstvo X
 - 4.x.1 Opis i prioriteti
 - 4.x.2 Nizovi pobuda/odziv
 - 4.x.3 Funkcijski zahtjevi

5. Ostali nefunkcionalni zahtjevi

- 5.1 Zahtjevi za performansama sustava
- 5.2 Zahtjevi za sigurnošću korisnika
- 5.3 Zahtjevi za sigurnošću podataka
- 5.4 Kvaliteta programske podrške
- 5.5 Poslovna pravila
- 5.6 Korisnička dokumentacija

6. Ostali zahtjevi

Dodatak A: Rječnik

Dodatak B: Modeli i dijagrami

Dodatak C: Lista nedovršenih/neodređenih zahtjeva

Objektno orijentirana analiza i dizajn

Primjer stupnjevitog projektiranja aplikacije

Postavljanje zahtjeva

❑ Korisnički zahtjevi

- Krajnji korisnik ili njegov predstavnik pišu specifikaciju sustava.

❑ Primjer aplikacije za vođenje knjižnice

- Trebamo sustav za vođenje knjižnice.
- Knjižnica posuđuje knjige i časopise članovima koji su evidentirani u sustavu, kao i knjige i časopisi.
- Knjižnica naručuje nove naslove. Popularni naslovi kupuju se u više primjeraka. Stare knjige i časopisi uklanjaju se kada su zastarjeli ili su u lošem stanju.
- Knjižničar je zaposlenik knjižnice koji je na usluzi posuđivačima, a sustav mora podržavati njegov rad.
- Posuđivač može rezervirati knjigu ili časopis koji trenutno nije raspoloživ tako da ga sustav obavijesti kad primjerak bude vraćen. Rezervacija se otkazuje kada se knjiga ili časopis podignu ili kroz proceduru za otkazivanje.
- Knjižničar može jednostavno u sustavu dodavati, mijenjati ili brisati informacije o naslovima, posuđivačima, posudbama i rezervacijama.
- Aplikacija treba biti raspoloživa na svim popularnim operacijskim sustavima, kao što su Linux, Unix, Windows, itd., te imati moderno grafičko sučelje (GUI).
- Sustav se jednostavno može nadograđivati novim funkcionalnostima.
- Prva inačica sustava ne mora slati obavijesti kada rezervirani naslovi postanu dostupni, niti mora provjeravati kada je naslov predugo posuđen.

Analiza zahtjeva

❑ Analiza zahtjeva

- Analiza je namijenjena uočavanju i opisivanju svih zahtjeva vezanih uz sustav i stvaranju modela koji definira ključne domenske razrede u sustavu (što je to čime sustav upravlja).
- U analizi najprije treba otkriti za što će se koristiti sustav i tko će ga koristiti. To se opisuje slučajevima korištenja i sudionicima (Actors).
- Slučajevi korištenja opisuju funkcionalne zahtjeve na sustav.
- Analiza slučajeva korištenja uključuje čitanje i analizu specifikacija, kao i raspravu o sustavu s potencijalnim korisnicima ili naručiteljem sustava.
- Odluke o načinu ugradnje biti će načinjene tijekom oblikovanja sustava.

❑ Slučaj korištenja (*Use Case*)

- Slučaj korištenja je skup povezanih scenarija za ostvarivanje neke korisničke akcije.
- Osim opisa primarnog scenarija u obliku numeriranih koraka, može sadržavati i opise alternativnih scenarija.
- Predstavlja vanjski pogled na sustav pa ne korelira s razredima u sustavu.
- U manjem (jednogodišnjem projektu) s do 10ak članova može biti 10ak osnovnih dijagrama slučajeva korištenja. Svaki osnovni slučaj može imati više scenarija i varijanti pa ukupno može biti i preko 100 dijagrama.

Analiza zahtjeva

☐ **Sudionici u knjižnici identificirani su kao knjižničar i posuđivač.**

- Knjižničari su izravni korisnici sustava.
- Posuđivači posredno posuđuju, rezerviraju i vraćaju knjige i časopise,
- Knjižničari ili druga knjižnica također mogu biti posuđivači.
- Sudionici su ljudi ili drugi sustavi izvan sustava kojeg razvijamo.
 - Npr. pisac ili baza podataka mogu biti sudionici.

☐ **Slučajevi korištenja u knjižnici**

- posudba, vraćanje, rezervacija, otkazivanje rezervacije, dodavanje naslova
- izmjena ili brisanje naslova, dodavanje knjige ili časopisa (primjerka),
brisanje primjerka, dodavanje posuđivača, izmjena ili brisanje posuđivača.

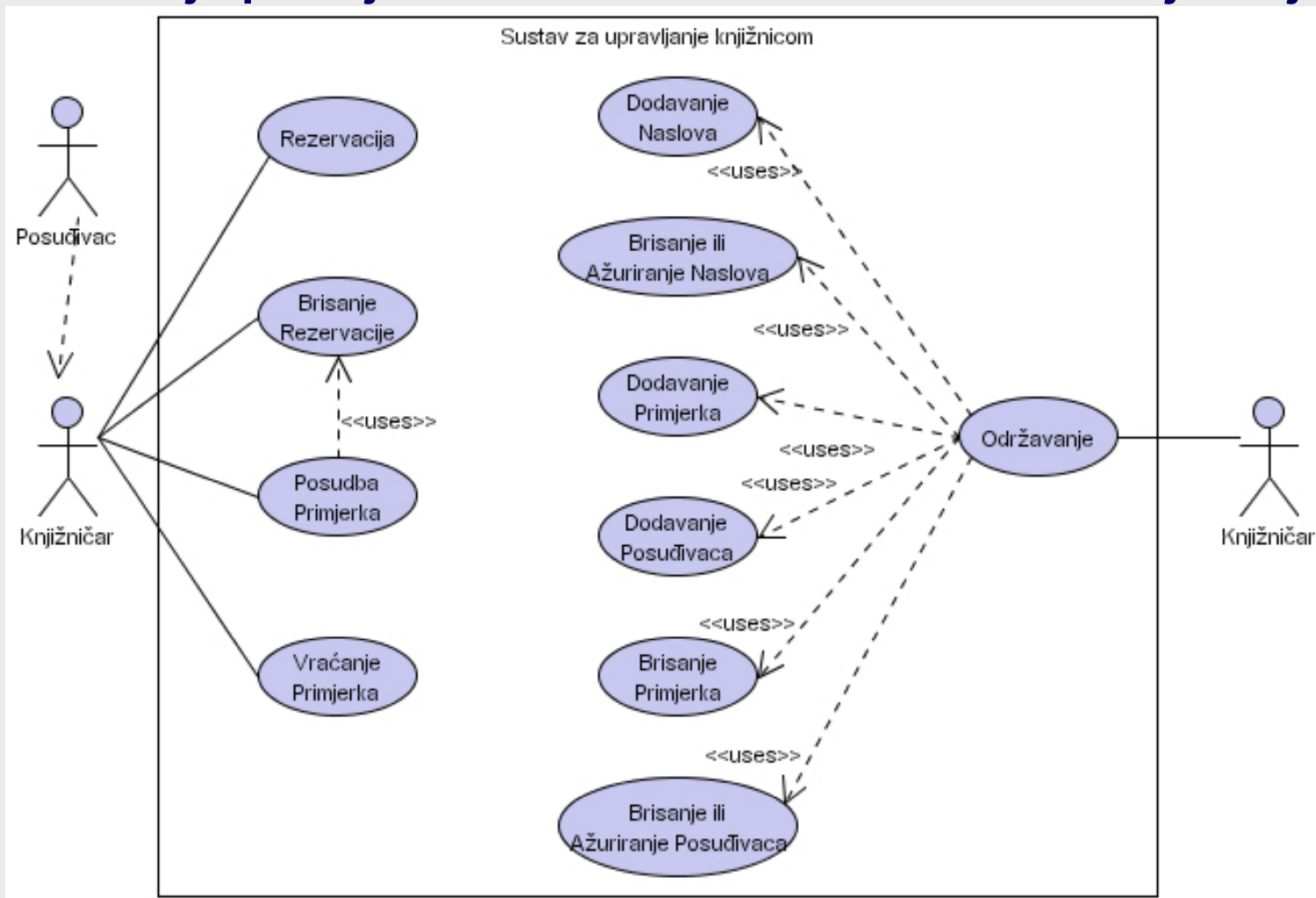
☐ **Budući da knjižnica često ima više kopija popularnih naslova, sustav mora odvojiti koncept naslova od koncepta primjerka.**

☐ **Analiza sustava za upravljanje knjižnicom je dokumentirana u dijagramu slučaja korištenja**

☐ **Svaki slučaj korištenja je dokumentiran i opisno.**

Dijagram slučajeja korištenja

- ❑ Prvi korak - utvrditi tko i kako koristi sustav
- ❑ Svi UC moraju počinjati sa sudionicima a neki i završavaju s njima.



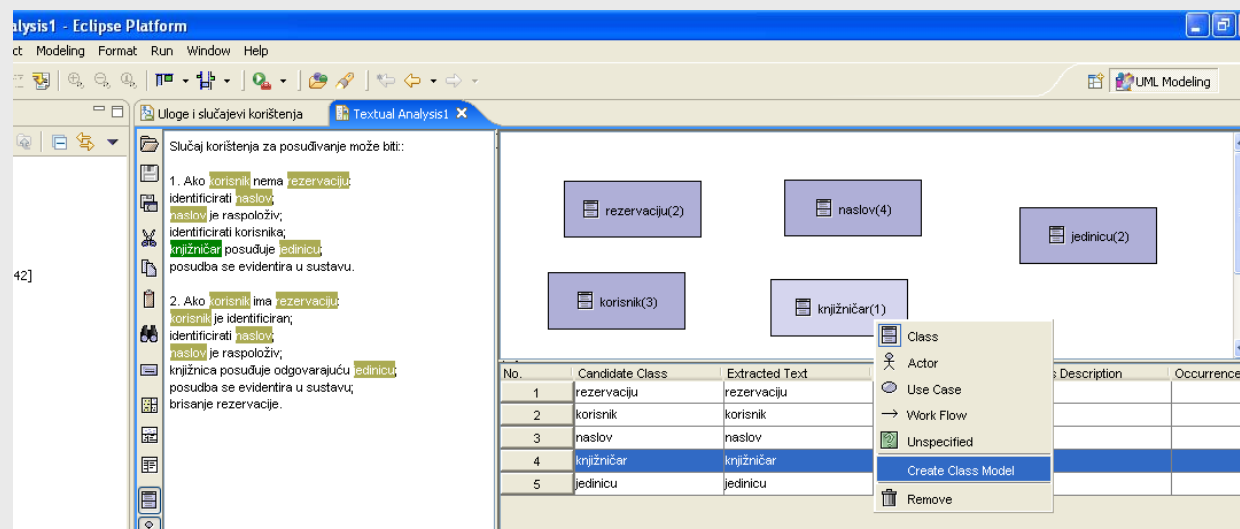
Opisivanje scenarija

❑ Slučaj korištenja za posuđivanje:

- 1. Ako korisnik nema rezervaciju:
 - identificirati naslov;
 - naslov je raspoloživ;
 - identificirati posuđivača;
 - knjižničar posuđuje jedinicu;
 - posudba se evidentira u sustavu.
- 2. Ako korisnik ima rezervaciju:
 - identificirati posuđivača;
 - identificirati naslov;
 - naslov je raspoloživ;
 - knjižnica posuđuje odgovarajuću jedinicu;
 - posudba se evidentira u sustavu;
 - brisanje rezervacije.

Analiza teksta CASE pomagalom

- ❑ Dijagram slučaja korištenja može se dopuniti tekstovnom analizom (*Textual Analysis*).
- ❑ Primjer: upotreba tekstovne analize na slučaju posudbe primjerka.
 - Dodavanje započinje desnim klikom na element *Posudba Primjerka*, zatim redom *Sub Diagrams – Textual Analysis – Create Textual Analysis*.
 - Slučaj se opisuje u nekoliko rečenica iz kojih se odabiru kandidati za razrede u sustavu, dovlačenjem kandidata za razrede na desni panel.
 - Desnim klikom na kandidata otvara se izbornik u kojem se može odabrati opcija *Create Class Model* koja će kandidata smjestiti u UML model iz kojega će se razred dalje moći koristiti u dijagramima razreda.



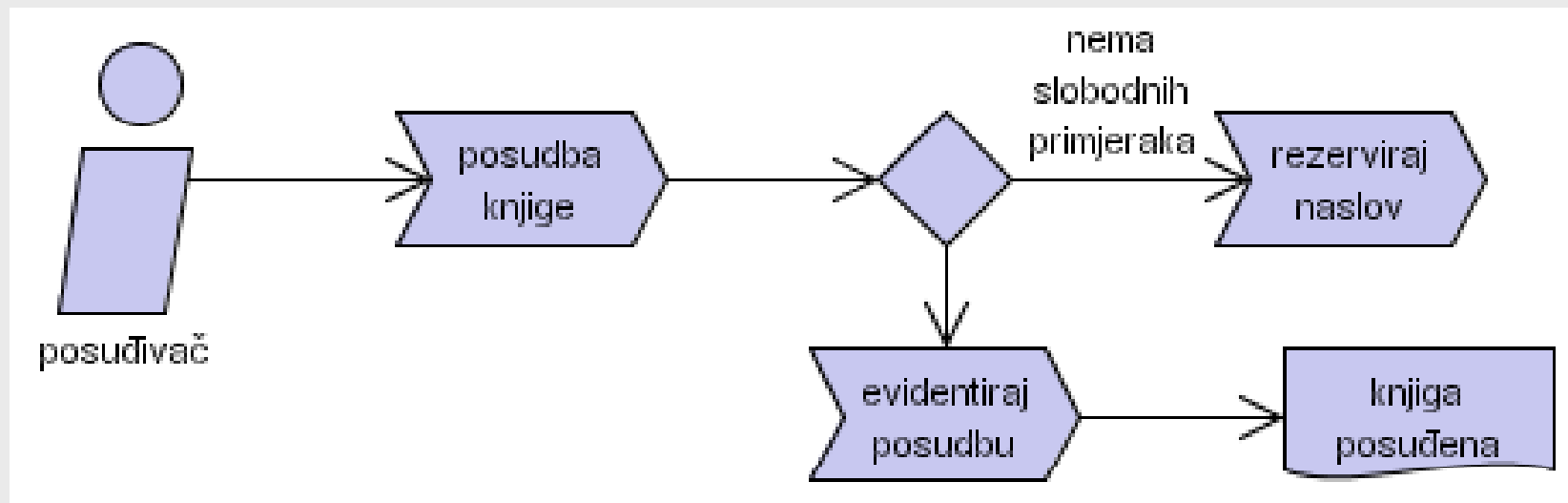
CRC kartice

- ❑ CRC (*Class-Responsibility-Collaboration*) kartice mogu se koristiti umjesto dijagrama
- ❑ Umjesto atributa i metoda, na kartici se mogu prikazati odgovornosti razreda.
- ❑ Suradnja se odnosi na druge razrede s kojima razred opisan CRC karticom surađuje čime se modelira interakcija među razredima.
- ❑ Preporučuje se da kartica sadrži najviše tri odgovornosti visoke razine. U protivnom treba razmisliti o razdvajanju razreda u nekoliko manjih razreda koji se mogu efikasno opisati.

Naslov	
Super Classes :	
Sub Classes : Knjiga, Casopis	
Description : Sadrži podatke o naslovima knjiga i časopisa	
Attributes :	
Name	Description
ime	ime naslova
Responsibilities :	
Name	Collaborator
može biti više primjeraka k. ili č	Primjerak
rezervacija pojedinih naslova	Rezervacija

Dijagrami radnih tokova

- ❑ Dijagrami poslovnih procesa - Poslovni dijagrami toka (*Business Workflow*) prikazuju poslovne procese na visokoj razini.



Generiranje dijagrama slijeda

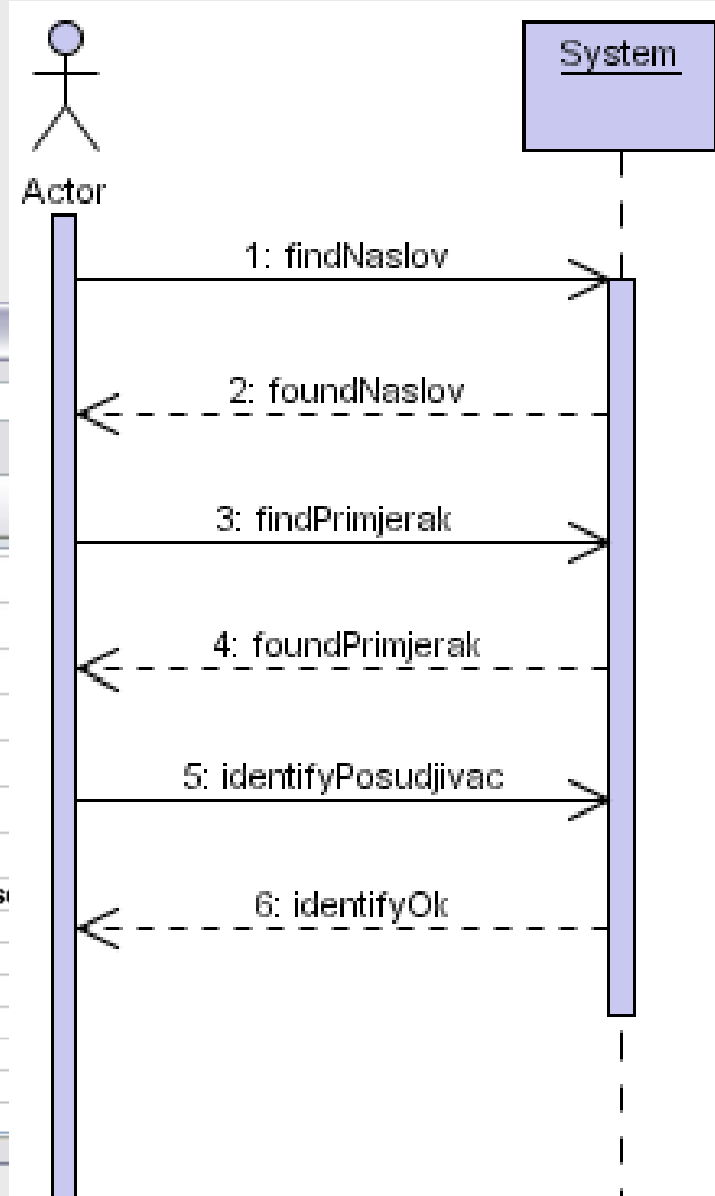
- *npr. Posudba – UC Details ... - Description.*
- *Flow Of Events*
- *Generate Sequence Diagram*
- *dijagram nadopuniti objektima i događajima*

PosudbaPrimjerka Details

Name:

Info Description Diagrams

Name																						
Super Use Case																						
Author																						
Date																						
Brief Description																						
Pre-conditions																						
Post-conditions																						
Flow of Events	<table border="1"><thead><tr><th></th><th>Actor Input</th><th>System Response</th></tr></thead><tbody><tr><td>1</td><td>findNaslov</td><td></td></tr><tr><td>2</td><td></td><td>foundNaslov</td></tr><tr><td>3</td><td>findPrimjerak</td><td></td></tr><tr><td>4</td><td></td><td>foundPrimjerak</td></tr><tr><td>5</td><td>identifyPosudjivac</td><td></td></tr><tr><td>6</td><td></td><td>identifyOk</td></tr></tbody></table>		Actor Input	System Response	1	findNaslov		2		foundNaslov	3	findPrimjerak		4		foundPrimjerak	5	identifyPosudjivac		6		identifyOk
	Actor Input	System Response																				
1	findNaslov																					
2		foundNaslov																				
3	findPrimjerak																					
4		foundPrimjerak																				
5	identifyPosudjivac																					
6		identifyOk																				



Domenska analiza

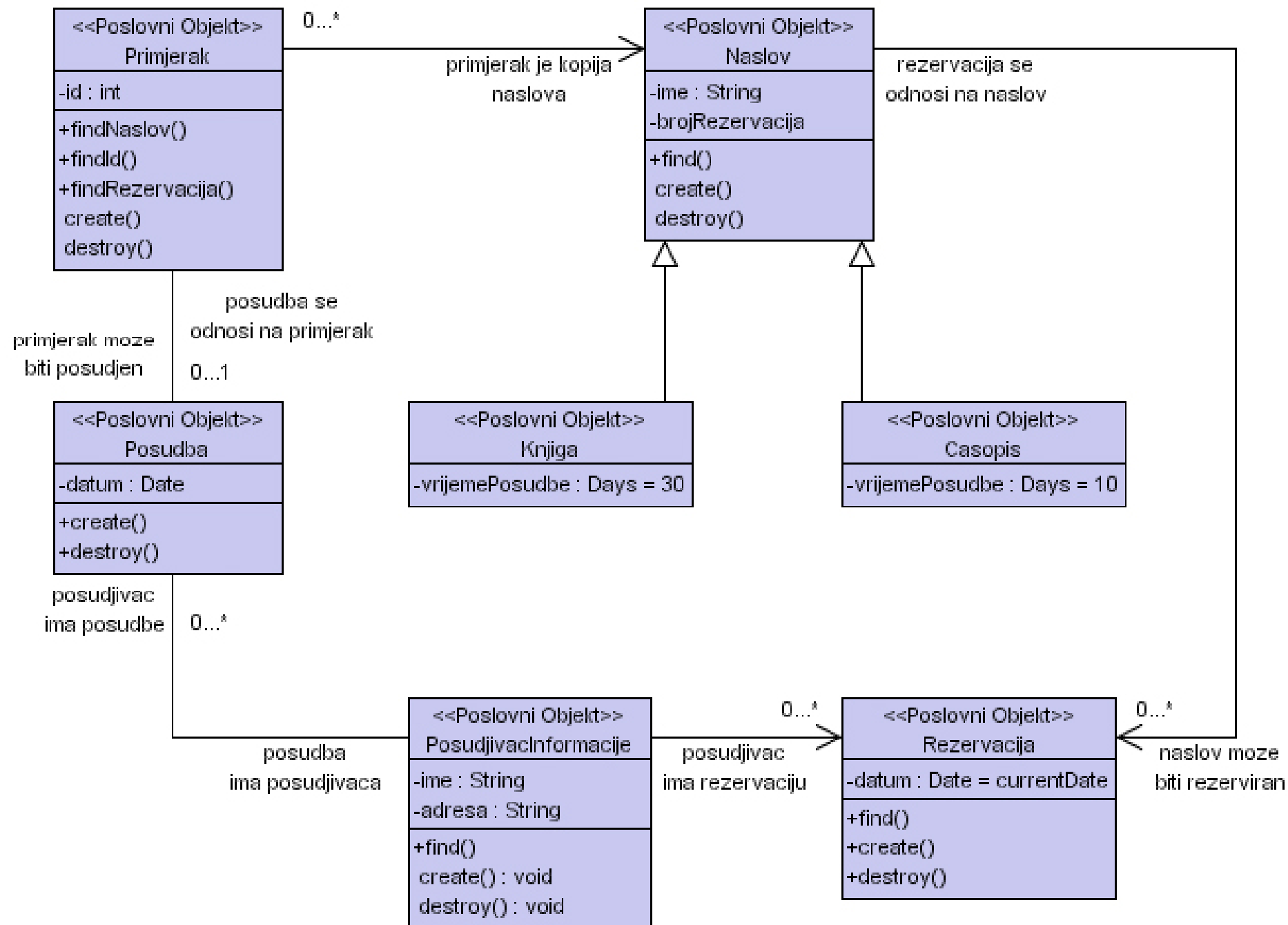
❑ Analiza definira problemsko područje to jest ključne razrede

❑ Primjer

- Domenski razredi (*Domain Class*) u sustavu za upravljanje knjižnicom mogu biti npr.: *PosudjivacInformacije* (treba razlikovati od *Posudjivac* kao sudionika u slučajevima korištenja), *Naslov*, *Knjiga*, *Casopis*, *Primjerak*, *Rezervacija* i *Posudba*.
- Domenski razredi su definirani upotrebom korisnički definiranog stereotipa "Poslovni Objekt", koji određuje da su objekti razreda dijelovi ključne domene te da stoga moraju biti trajno pohranjeni u sustavu.

❑ Temeljem analize može se proširiti osnovni dijagram slijeda

- npr. *Posudba Primjerka* - otvoriti izbornik te odabrati *Sub Diagrams – Sequence Diagram – Create Sequence Diagram*
- istom slučaju korištenja može biti pridruženo više dijagrama



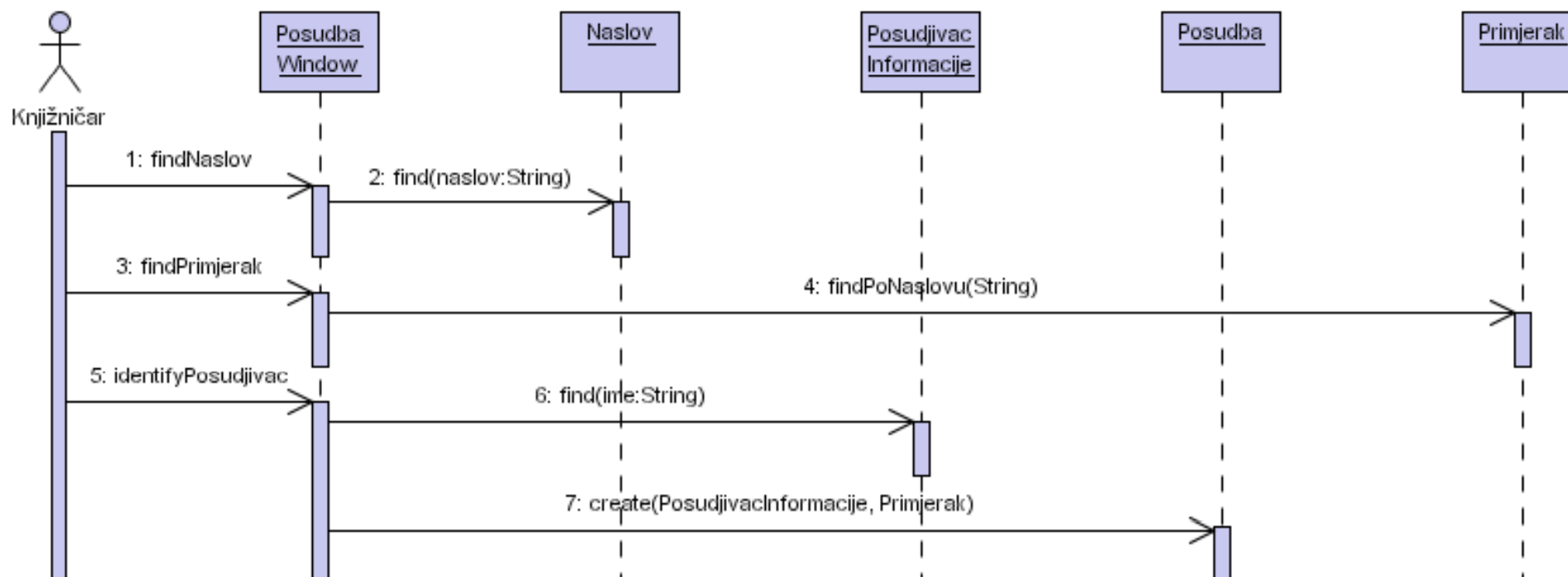
Proširenje dijagrama slijeda

❑ Primjer,

- Dijagram slijeda za slučaj korištenja *Posudba* kada korisnik nema rezervaciju.

❑ Interakciju uvijek potiče sudionik putem korisničkog sučelja

- Detalji korisničkog sučelja mogu biti razrađeni naknadno.



Proširenje dijagrama slijeda

- ❑ **Pri modeliranju dijagrama slijeda postaje očito da je potrebno sučelje prema sudionicima.**
 - U ovoj analizi dovoljno je biti svjestan da su dijelovi sučelja (prozori ili dijalozi) potrebni za rezervaciju, posuđivanje i vraćanje jedinica.
 - Detalji korisničkog sučelja mogu biti razrađeni naknadno.

- ❑ **Za razdvajanje GUI razreda u analizi od domenskih razreda koristi se razdvajanje i grupiranje GUI razreda u paket *GUI Package*, a domenskih razreda u paket *Business Package*.**

Dizajn sustava

❑ Faza dizajna proširuje i detaljizira analitički model uzimajući u obzir sve tehničke implikacije i restrikcije.

- Svrha dizajna je specificiranje operativnog rješenja koje se može ugraditi u programski kôd.
- Tijekom projektiranja može doći do razdvajanja i grupiranja pojedinih razreda radi uslojavanja (npr. razredi sučelja, razredi poslovne logike, razredi pristupa podacima).

❑ Razina dizajna

- Dizajn arhitekture je dizajn na visokoj razini gdje se definiraju paketi (podsustavi) uključujući ovisnosti i osnovne komunikacijske mehanizme između paketa.
- Detaljni dizajn opisuje sve razrede koji predstavljaju jasnu specifikaciju za ugradnju. Dinamički UML modeli koriste se za demonstraciju kako se objekti razreda ponašaju u specifičnim situacijama.

Dizajn arhitekture

❑ Paket korisničko sučelje (*User-Interface Package, UI*)

- Paket surađuje s poslovnim paketom koji sadrži razrede u kojima se zapravo pohranjuju podaci. UI paket poziva operacije poslovnih objekata kako bi dohvatio ili upisao podatke u njih.

❑ Poslovni paket (*Business-Objects Package, BO*)

- Uključuje domenske razrede iz analitičkog modela kao što su *PosudjivacInformacije*, *Naslov*, *Primjerak*, *Posudba*, itd.
- Dizajn potpuno definira njihove operacije i dodaje podršku za trajnu pohranu (*Persistence*).
- U našem primjeru, poslovni objekti surađuju s paketom za pohranu tako što svi poslovni razredi nasljeđuju razred *Persistent* iz tog paketa.

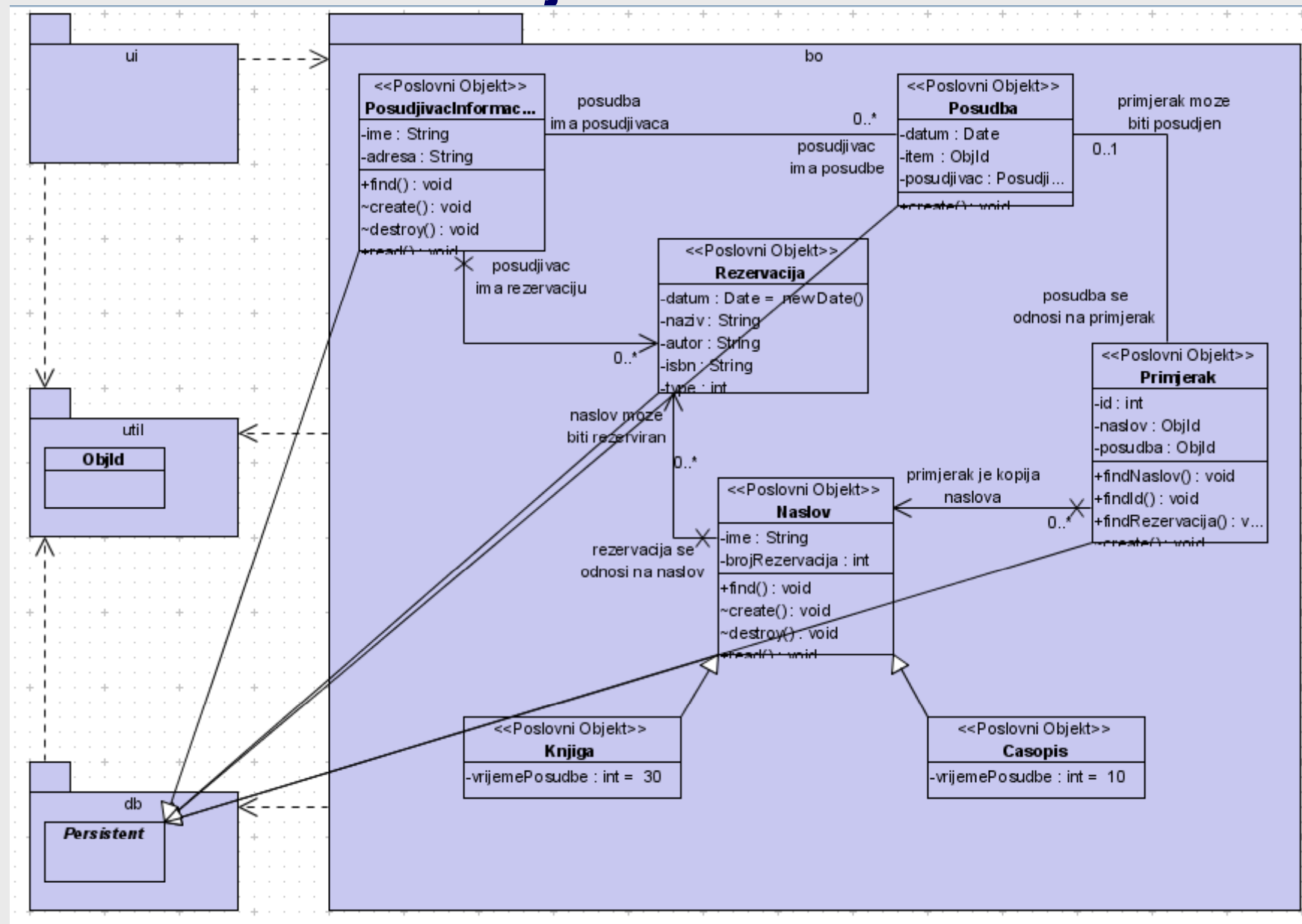
❑ Paket za rad s bazom podataka (*Database Package, DB*)

- Služi kao servis ostalim razredima iz poslovnog paketa.

❑ Uslužni paket (*Utility Package, UTIL*).

- Sadrži usluge koje se koriste u ostalim paketima u sustavu.
- Npr. *ObjId* razred koristi se za dohvrat trajno pohranjenih objekata u sustavu uključujući UI i DB pakete te poslovni paket.

Dizajn arhitekture



Detaljni dizajn

❑ Paket za rad s bazom podataka.

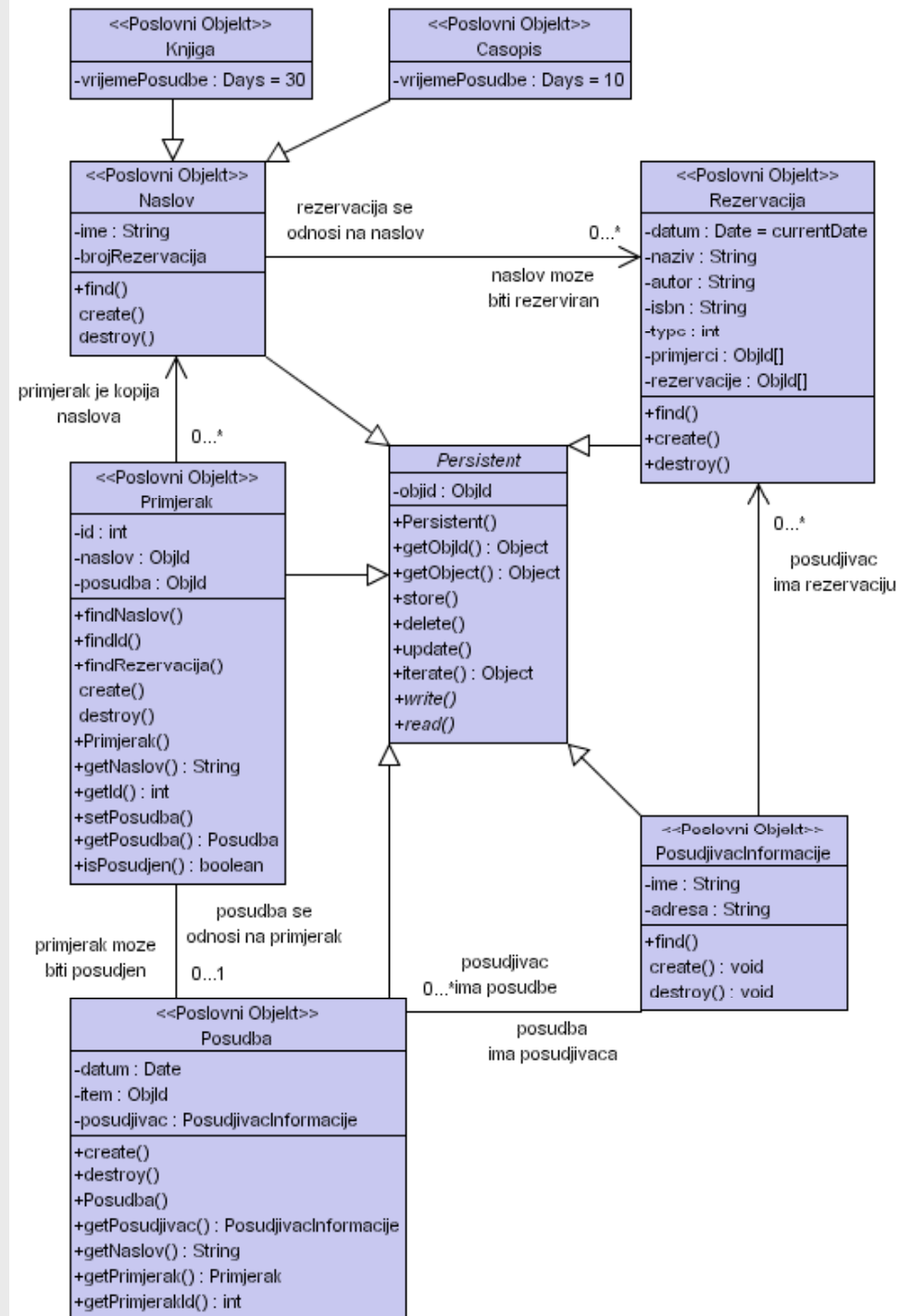
- Aplikacija mora imati mogućnost trajne pohrane objekata.
- Detalji o smještaju su skriveni od aplikacije koja samo poziva zajedničke operacije kao što su *store()*, *update()*, *delete()* ili *find()*.
- Ovo su članovi osnovnog razreda *Persistent*, kojeg nasljeđuju drugi razredi koji zahtijevaju pohranu.
- Kao identifikator podataka koristi se *ObjId* (*Object Identity*), razred čiji se objekti koriste za referenciranje bilo kojeg perzistentnog objekta u sustavu (bez obzira je li objekt na disku ili je učitao u aplikaciju)
- *ObjId* je općeniti razred koji koriste svi paketi u sustavu pa je smješten u uslužni paket, a ne u DB paket.

❑ Poslovni paket

- Zasnovan je na odgovarajućem paketu iz analize, tj. domenskim razredima. Razredi, njihovi odnosi i ponašanja su očuvani, ali su razredi detaljnije opisani, što uključuje i opis implementacije odnosa i ponašanja.

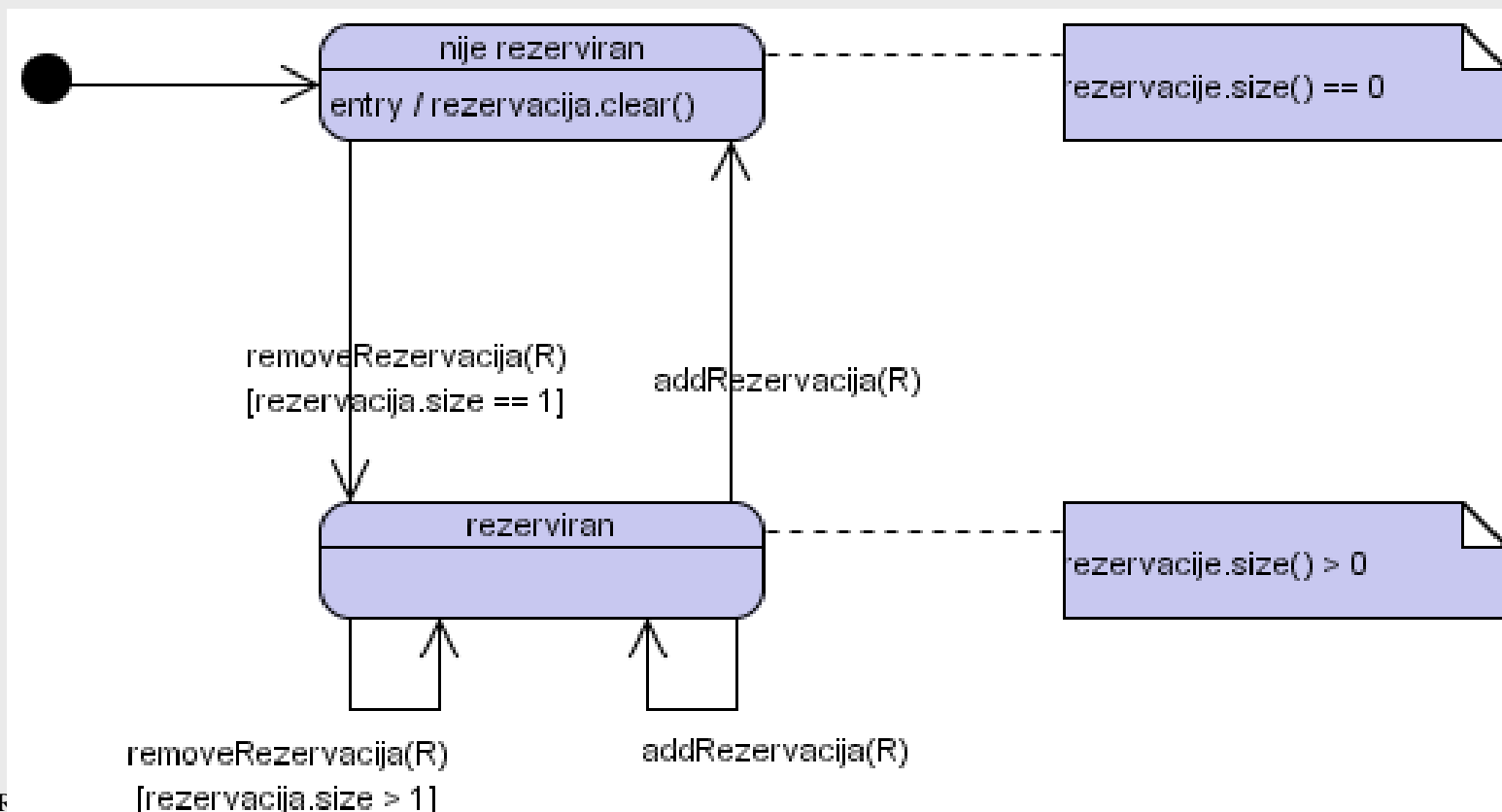
Detaljni dizajn

- ❑ Primjer: Dizajn poslovnih objekata
- ❑ Dizajn počinje određivanjem detalja modela. Sučelja su potpunije specificirana, odabrani su tipovi atributa, itd.
- ❑ Na slici nisu prikazani paketi kojima razredi pripadaju: svi razredi poslovnih objekata pripadaju *bo* paketu, osim razreda *Persistent* koji pripada *db* paketu.
- ❑ Neke operacije su razrađene u nekoliko operacija, a neke su preimenovane u odnosu na prethodne modele



Razrada stanja

- ❑ Dijagrami stanja iz analize su također detaljnije razrađeni u dizajnu, prikazujući kako su stanja prezentirana i vođena u pravom sustavu.
- ❑ Primjer: Stanja za naslov (rezervirano i slobodno) implementirana su u vektorom *rezervacije*. Promjene se obavljaju pozivom metoda *addRezervacija()* i *removeRezervacija()*.



Paket korisničkog sučelja

☐ UI paket

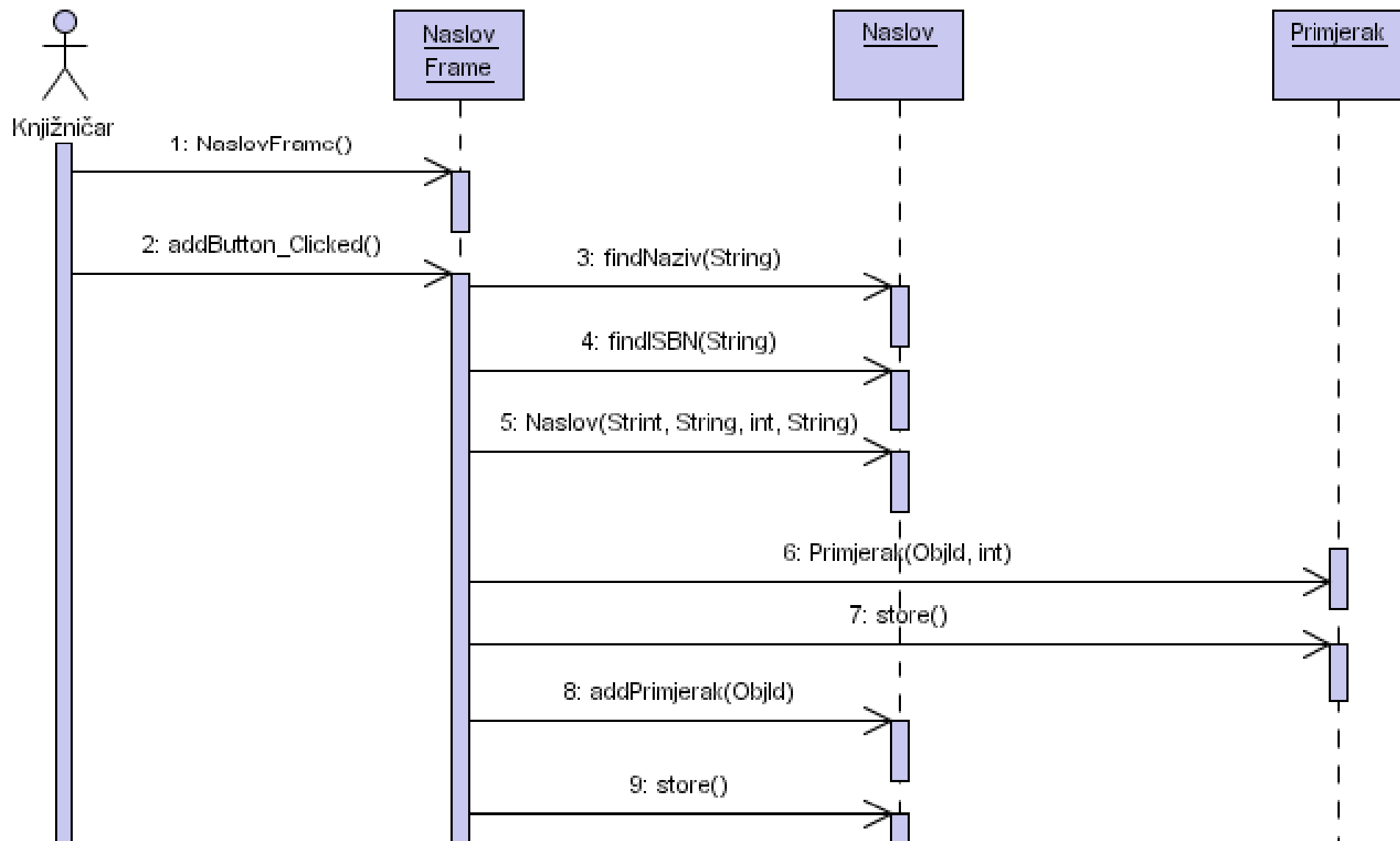
- Prezentira usluge i informacije korisniku sustava.
- Sadrži dinamičke modele, jer se sva interakcija s korisnikom ostvaruje putem korisničkog sučelja.

☐ Realizacija slučajeva korištenja u modelu dizajna prikazana je detaljno, uključujući stvarne operacije razreda.

☐ Dijagrami slijeda razrađuju se iterativno, čak i tijekom kodiranja.

- Metode na slici detaljnog dizajna sučelja mogu biti stvarno kodirane.

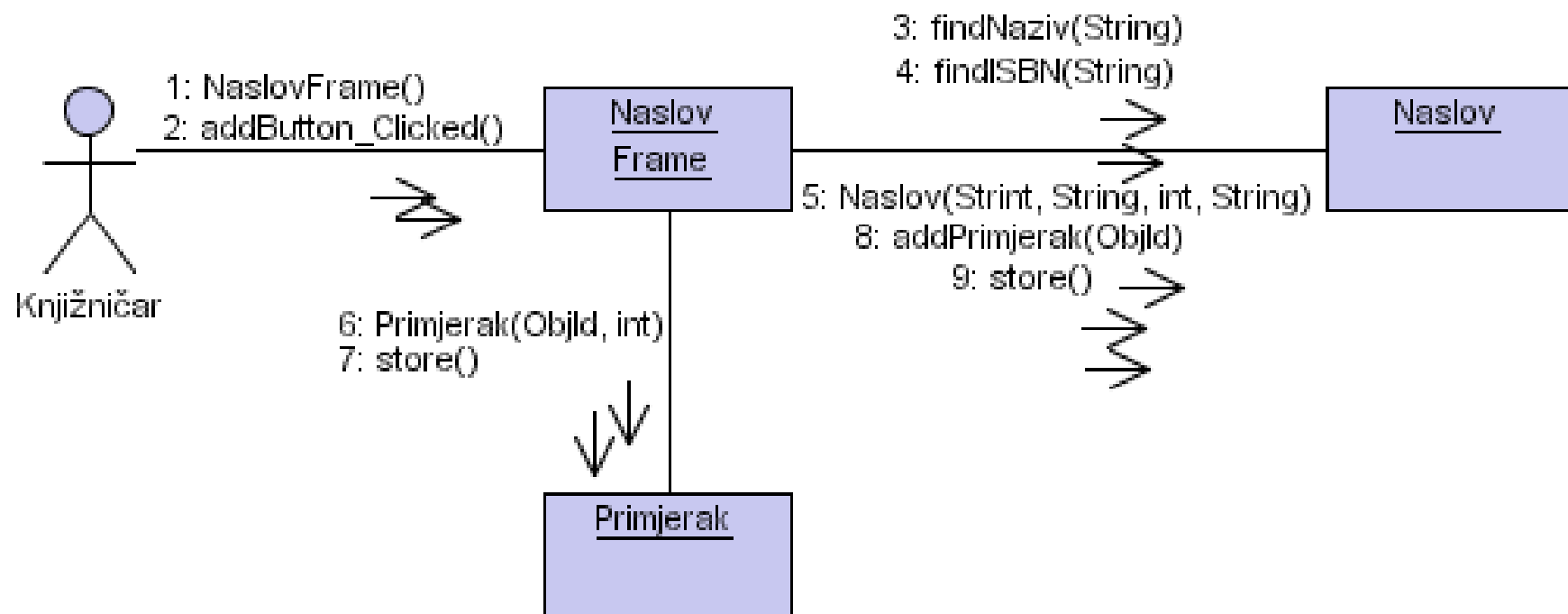
Dizajn UI paketa



Konverzija u dijagram suradnje

□ Dijagrami suradnje (*Collaboration Diagram*)

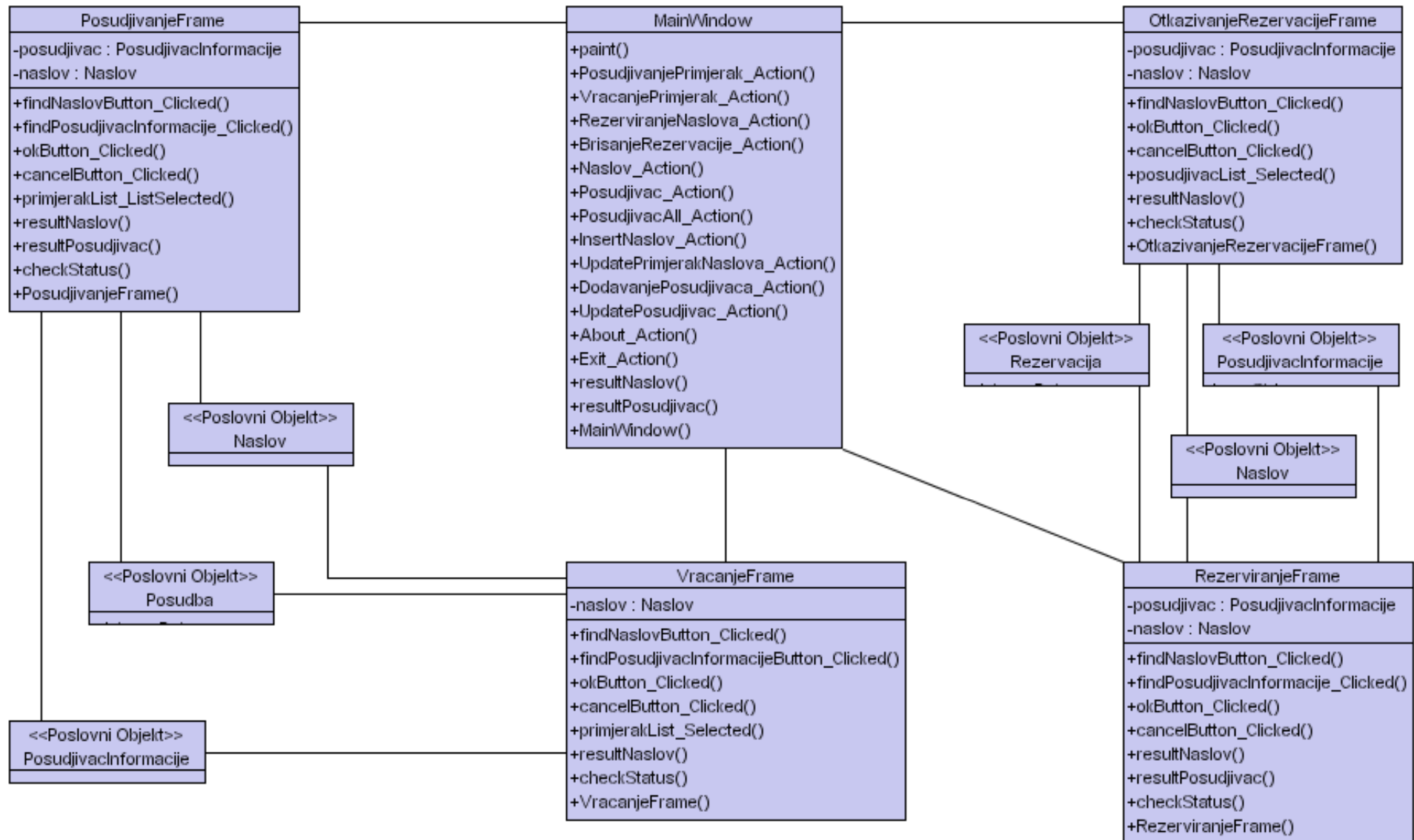
- Suradnja obuhvaća više od jednog razreda.
- Dijagram suradnje koristan je kada se želi opisati interakcija.
- Mogu se koristiti kao alternativa dijagramima slijeda
- mogu se generirati iz dijagrama slijeda i obrnuto (*VP-UML Synchronize To Colaboration Diagram, To Sequence*).



Dizajn korisničkog sučelja

- ❑ **Stvaranje korisničkog sučelja posebna je aktivnost tijekom faze dizajna.**
 - Korisničko sučelje u aplikaciji za upravljanje knjižnicom zasnovano je na slučajevima korištenja te je podijeljeno na sljedeće cjeline, od kojih svaka ima svoj posebni izbornik u glavnom izborniku:
- ❑ **Funkcije.**
 - Prozori za primarne funkcije u sustavu; posuđivanje i vraćanje jedinica te rezervaciju naslova.
- ❑ **Informacije.**
 - Prozori za pregled informacija u sustavu o naslovima i korisnicima.
- ❑ **Održavanje.**
 - Prozori za održavanje sustava; dodavanje, izmjena i brisanje naslova, korisnika ili jedinica
- ❑ **Sljedeća slika prikazuje primjer dijagrama razreda u UI paketu koji sadrži tipične rukovatelje događajima (*Event Handlers*).**
 - Atributi za gumbe, natpise i polja za unos nisu prikazani.

Razredi korisničkog sučelja



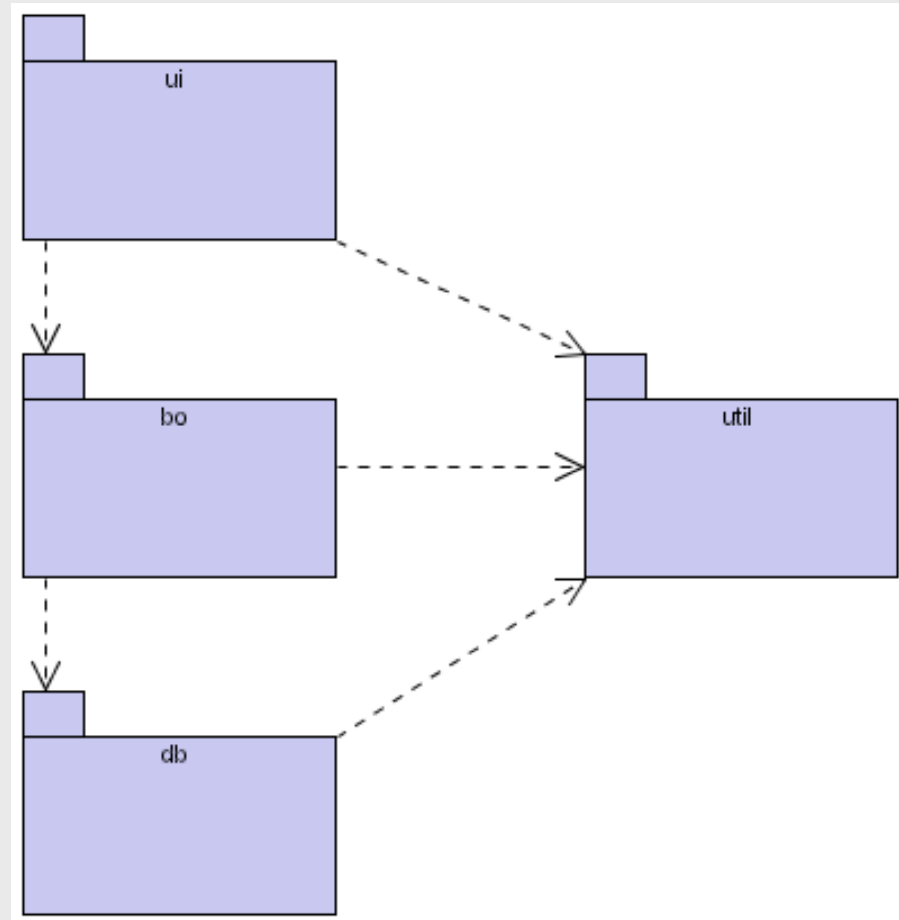
Ugradnja (implementacija)

❑ Programiranje tijekom faze konstrukcije ili implementacije

- Organizacija koda može biti prikazana dijagramom komponenti koji u modelu dizajna sadrži (u ovom slučaju) jednostavno preslikavanje (mapiranje) logičkih paketa u komponente.
- Razredi su također mapirani prema paketima

❑ Dijagram komponenti (*Component Diagram*)

- Dijagram komponenti prikazuje različite komponente sustava i njihove međuzavisnosti.
- Komponenta je fizički programski modul.
- Ovisnosti među komponentama prikazuju kako promjena u jednoj komponenti može utjecati na ostale, a mogu biti komunikacijske ovisnosti, ovisnosti tijekom prevođenja, itd.



Ugradnja

- ❑ **Ručno kodiranje ili generiranje izvornog koda**
 - specifikacije se dohvaćaju iz sljedećih dijagrama
- ❑ **specifikacija razreda:**
 - specifikacija razreda detaljno prikazuje potrebne attribute i operacije
- ❑ **dijagram razreda:**
 - statička struktura i ovisnosti o drugim razredima
- ❑ **dijagram stanja:**
 - dijagram stanja za razred koji prikazuje moguća stanja i prijelaze koje treba obrađivati (s operacijama koje okidaju prijelaze)
- ❑ **dinamički dijagrami (slijeda, suradnje i aktivnosti) u kojima su uključeni objekti razreda:**
 - prikaz ugradnje metoda i uporabe objekata razreda
- ❑ **dijagrami slučajeva korištenja i specifikacije:**
 - prikazuju rezultat sustava
 - koriste se kada programer treba više informacija o tome kako će se sustav koristiti (kada se ne vidi kontekst).

Aktivnosti

❑ Aktivnost (*Activity*)

- bilo stvarni proces ili izvršavanje neke procedure, npr. metode u razredu.

❑ Dijagrami aktivnosti (*Activity Diagrams*)

- opis ponašanja, naročito kod paralelne obrade
- Dijagrami aktivnosti opisuju redoslijed aktivnosti s mogućnostima opisa uvjetnih i paralelnih ponašanja. Oni su varijanta dijagrama stanja.

❑ Uvjetna ponašanja prikazana grananjem i spajanjem (*branch, merge*)

- Grananje ima jedan ulaz i više izlaza od kojih je moguće odabrati samo jedan (međusobno isključivanje).
- "e/se" smjer grananja odabire se ako su svi drugi uvjeti nezadovoljeni.
- Spajanje ima više ulaza i jedan izlaz. Predstavlja kraj uvjetnog grananja.

❑ Paralelna ponašanja prikazana dijeljenjem i ujedinjavanjem (*fork, join*).

- Dijeljenje ima jedan ulaz i više izlaza.
- Svi izlazni procesi odvijaju se paralelno pri čemu nije važan redoslijed ili međusobno preplitanje koraka pojedinih procesa.
- Ujedinjavanje služi i za sinkronizaciju, što znači da se radnja nastavlja tek kada svi paralelni procesi završe.

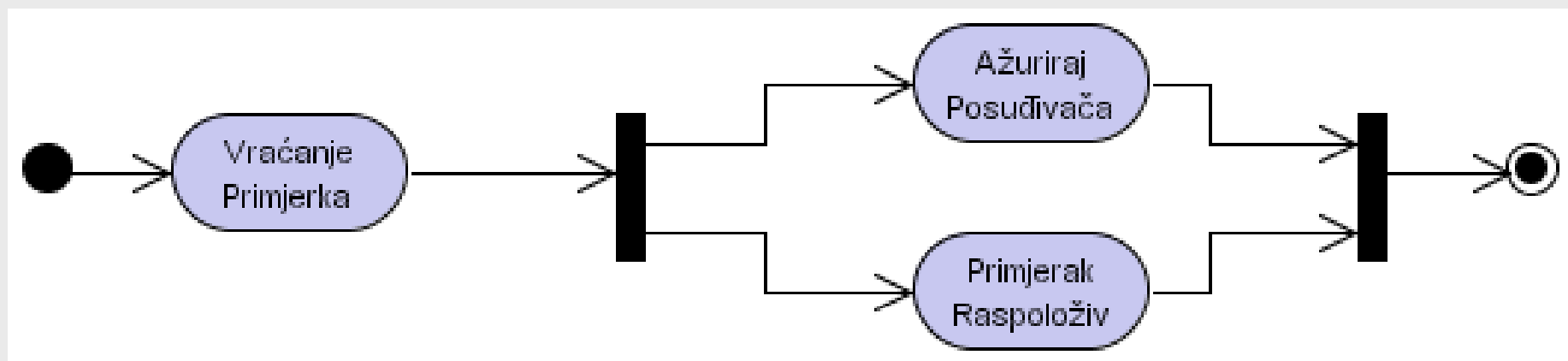
Dijagram aktivnosti

❑ Plivaće staze (*Swimlanes*)

- koriste se kada se želi navesti tko obavlja određenu operaciju.
- Dijagram se prikazuje tako da se vertikalno podijeli na zone zaduženja.
- U svaku zonu ucrtavaju se aktivnosti za koje je zona zadužena (osoba, odjel i sl.).

❑ Dijagrami aktivnosti korisni su pri opisivanju paralelnih aktivnosti (npr. dijagram toka ili višenitnost).

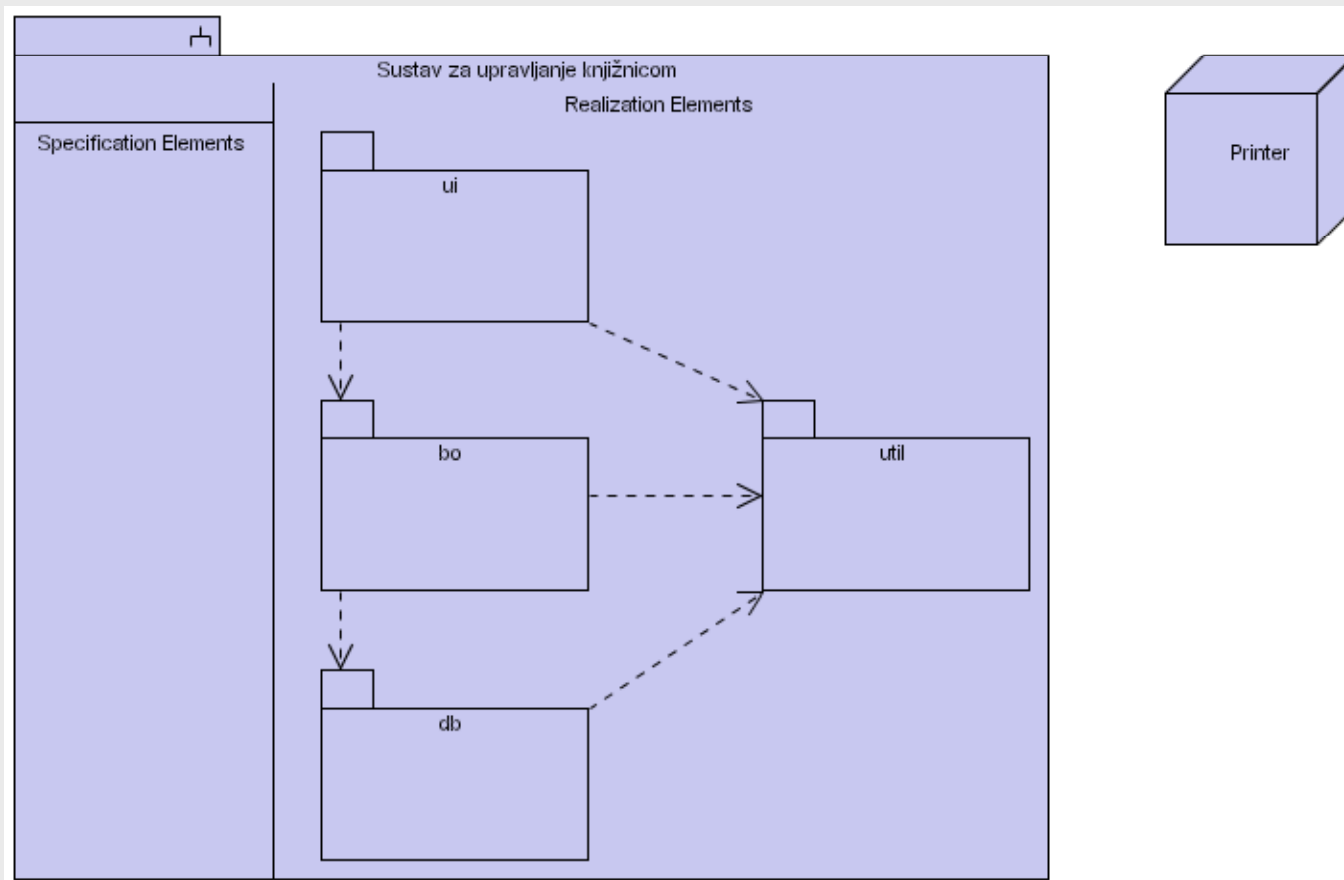
- Nedostatak je što ne prikazuju izravnu vezu između aktivnosti i objekata.
- Plivaće staze donekle ublažavaju ovaj nedostatak tako što prikazuju i nositelje aktivnosti.



Dijagrami ugradnje

❑ Dijagrami isporuke (*Deployment Diagrams*)

- prikazuju fizički odnos softverskih i hardverskih komponenti u sustavu
- Svaki čvor u dijagramu predstavlja fizičku cjelinu – vrlo često komad hardvera, npr. osobno računalo, pisač, poslužitelj.





Zadaci

❑ Domaća zadaća – evidencija zahtjeva

- Članovi ekipe evidentiraju zahtjeve
 - verificiranim zapisnicima intervjua obavljenim na lab. vježbama
- Svaki član isporučuje *Zapisnik-Autor.doc*
 - svaki član zapisuje sve što je bilo rečeno
 - svaki član opisuje scenarij slučaja korištenja kojim će doprinijeti projektu

❑ Zadatak projekta (isporuka na međuispitu)

- Dokumentirati zahtjeve
 - integracijom pojedinačnih zapisnika u jedinstveni dokument
 - izradom dijagrama slučajeva korištenja
 - *SpecifikacijaZahtjeva.doc*
- Ažurirati plan projekta temeljem evidentiranih zahtjeva
 - *PlanProjekta.doc* i *PlanProjekta.mpp*

Reference

- ❑ **Design Java Apps with UML; Hans-Erik Eriksson, Magnus Penker**
 - http://www.ecestudents.ul.ie/Course_Pages/MEng_CS/Modules/EE6421/Examples/UML/Design Java Apps with UML.htm
- ❑ **UML Distilled, Second Edition, A Brief Guide to the Standard Object Modeling Language; Martin Fowler, Kendall Scott, Addison-Wesley, 2000.**
- ❑ <http://www.patterndepot.com/put/8/JavaPatterns.htm>
- ❑ <http://www.eclipse.org>
- ❑ <http://www.visual-paradigm.com>

Ujedinjeni jezik za modeliranje

UML

UML

❑ UML = Unified Modeling Language

- Ujedinjeni (objedinjeni, univerzalni) jezik za modeliranje
- Standardni jezik za prikaz, specifikaciju, izradu i dokumentiranje elemenata sustava zasnovanih na programskoj podršci
- Proširenje osnovne funkcionalnosti stereotipovima (<<kalupima>>)

❑ Dijagrami strukture, Strukturni dijagrami

- Class Diagram – dijagram razreda
- Object Diagram – dijagram objekata
- Component Diagram – dijagram komponenti
- Deployment Diagram – dijagram ugradnje

❑ Dijagrami ponašanja

- Use Case Diagram – dijagram slučajeva korištenja
- Sequence Diagram – slijedni dijagrami
- Collaboration Diagram – dijagrami kolaboracije
- Statechart Diagram – dijagrami stanja
- Activity Diagram – dijagrami aktivnosti

Slučajevi korištenja

- ❑ **Use case = Slučaj korištenja (primjene)**
 - Skup scenarija sa zajedničkim ciljem korištenja sustava
- ❑ **Scenarij**
 - niz koraka koji opisuju interakciju korisnik-sustav
- ❑ **Analiza slučajeva korištenja (Use Case Analysis)**
 - tehnika analize poslovnih procesa iz perspektive korisnika
- ❑ **Tipični sadržaj**
 - Kako kreće i završava slučaj
 - Normalni tok događaja
 - Varijabilni (alternativni) tok događaja
 - Iznimni tok događaja
- ❑ **Primjer scenarija**
 - Kupac pregledava Web katalog i stavlja proizvode u košaricu. Kada odluči platiti, unosi podatke o otpremi i kreditnoj kartici te potvrđuje kupnju. Sustav provjerava autorizaciju kartice i potvrđuje prodaju interaktivno te putem elektroničkog pisma.
 - Autorizacija kreditne kartice je međutim mogla ne uspjeti!
 - Ovo bi bio posebni scenarij.

Primjer scenarija

❑ **Kupi proizvod**

1. Kupac pregledava Web katalog i stavlja proizvode u košaricu
2. Kupac odluči platiti
3. Unosi podatke o otpremi
4. Sustav prikazuje punu informaciju cijeni, uključujući otpremu
5. Kupac unosi podatke o kreditnoj kartici
6. Sustav autorizira kupnju
7. Sustav potvrđuje prodaju interaktivno
8. Sustav šalje elektroničko pismo potvrde.

❑ **Alternativa: Neuspješna autorizacija**

- U koraku 6, sustav ne uspijeva autorizirati karticu
- Dozvoliti kupcu ponovni unos podataka o kartici

❑ **Alternativa: Redoviti kupac**

- 3a. Sustav prikazuje informaciju o otpremi i cijeni te zadnje četiri znamenke broja kreditne kartice
- 3b. Kupac potvrđuje prikazane pretpostavljene podatke ili unosi nove
- Nastavak u koraku 6 primarnog scenarija

Dijagrami slučajeve korištenja

❑ Sudionik, glumac (actor)

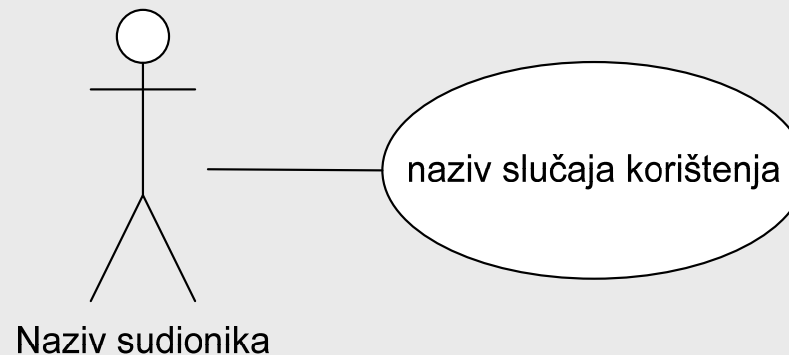
- Uloga koju ima korisnik u odnosu na sustav
- Netko ili nešto u uzajamnom djelovanju (interakciji) sa sustavom
- Proučava se da bi se odredile njegove potrebe

❑ Slučaj ponašanja (use case)

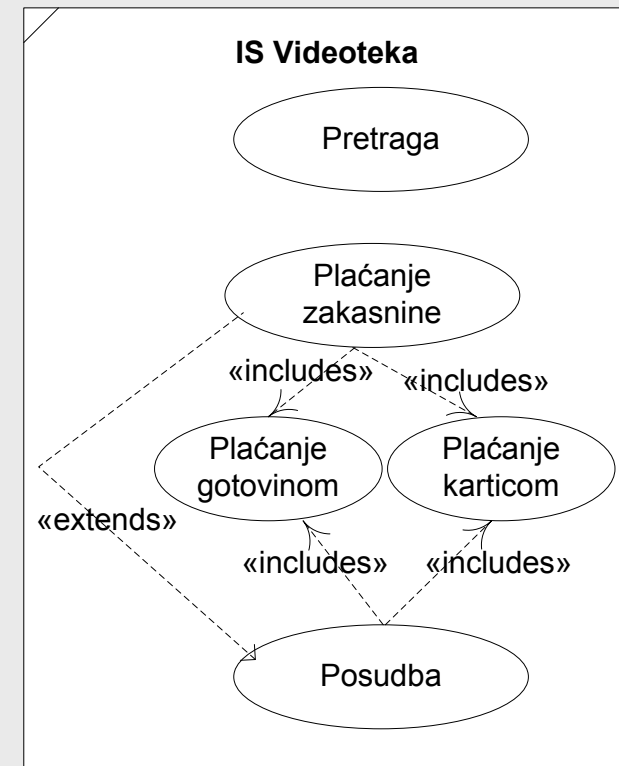
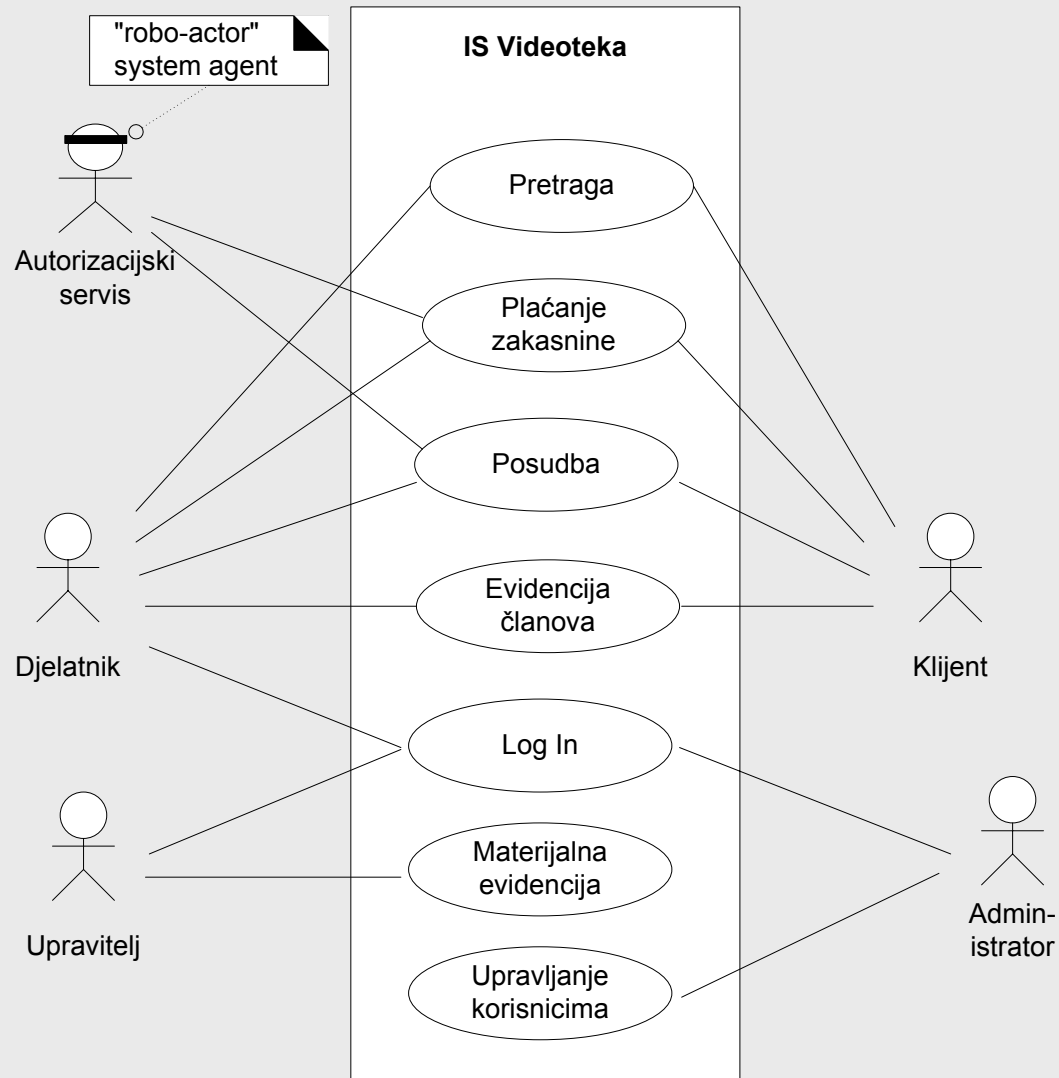
- Primjer, uzorak, obrazac ponašanja
- Slijed povezanih interakcija između sudionika i sustava

❑ UC dijagrami predložuju veze između glumaca i slučajeva

- uključivanje («include») – povezuje zajedničke dijelove ponašanja
- proširenje («extend») – varijacije normalnog ponašanja uz dodatna pravila



Primjer UC dijagrama



Dijagrami razreda

❑ Dijagram razreda (Class Diagram)

- Prikaz statičke strukture, elemenata (pr. razredi) te njihovih odnosa
- Dijagram može sadržavati i sučelja (interfaces), pakete (packages), ...
- Bolji naziv bio bi 'statički strukturni dijagram' (static structural diagram).

❑ Elementi dijagrama

- Razredi, njihova struktura i ponašanje (atributi i operacije)
- Mogućnost pristupa - vidljivost

❑ Statički odnosi

- Obične veze ili asocijacije (associations)
 - npr. "korisnik posuđuje više video-kazeta"
- Generalizacija - Zavisnost (dependency) i nasljeđivanje (inheritance)
 - npr. "bolničarka je osoba"
- Gomilanje ili udruživanje, agregacija (aggregation),
 - npr. "motor je dio aviona"

❑ Dodatne oznake

- Indikatori množine, mnogostrukosti (multiplicity) - kardinalnost
- Indikatori upravljivosti, navigabilnosti (navigability) – smjer povezivanja
- Uloge (role names)

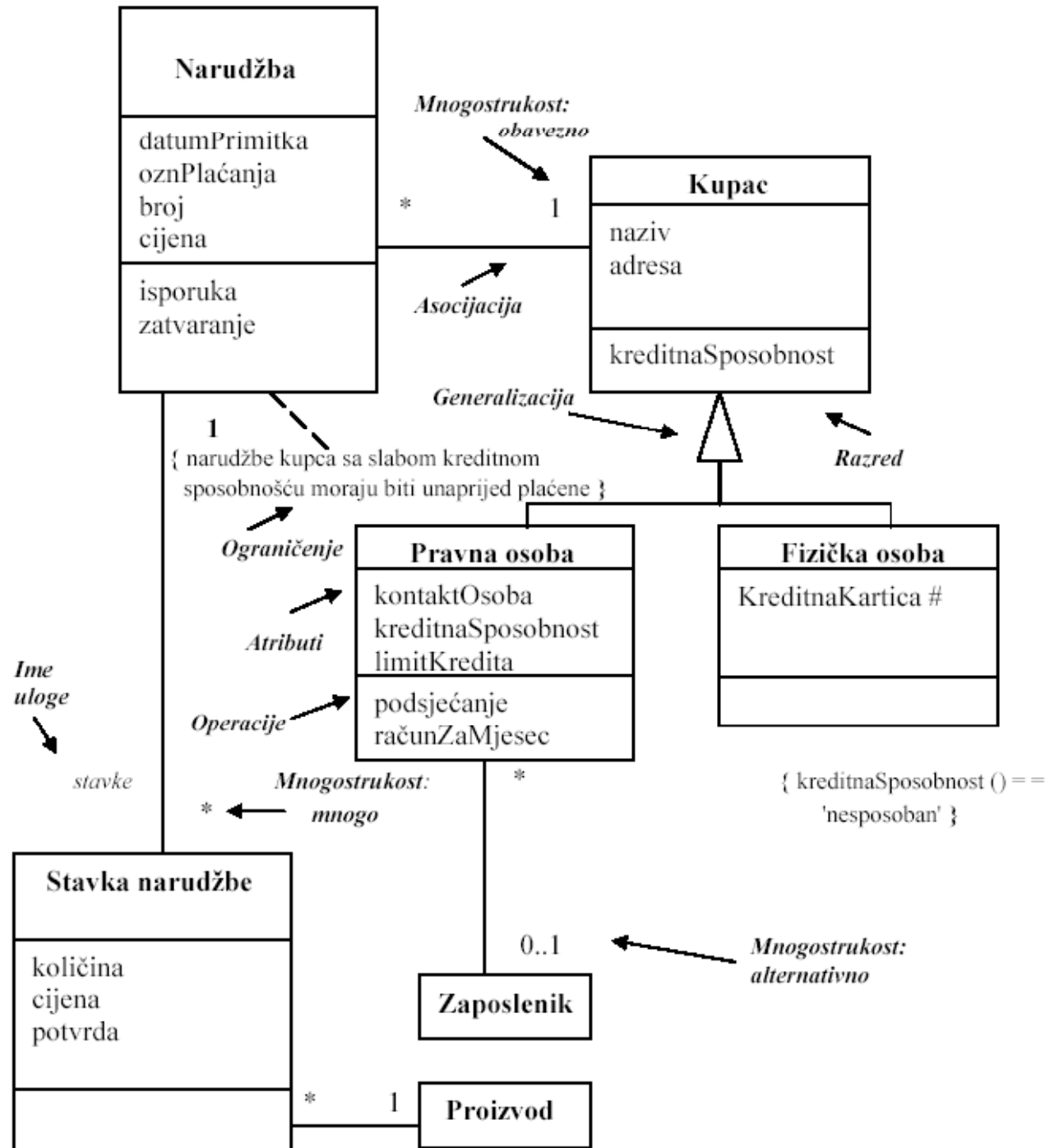
❑ Primjer dijagrama razreda

❑ Upravljivost

- ukoliko veza nije usmjerena, smatra se da je odgovornost obostrana – Narudžba sadrži pokazivač na kupca, a kupac skup (polje, listu) narudžbi
- usmjerena veza – određuje što će od toga biti ugrađeno

❑ Primjer

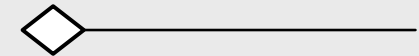
- Koje bismo veze usmjerili ?



Agregacija i kompozicija

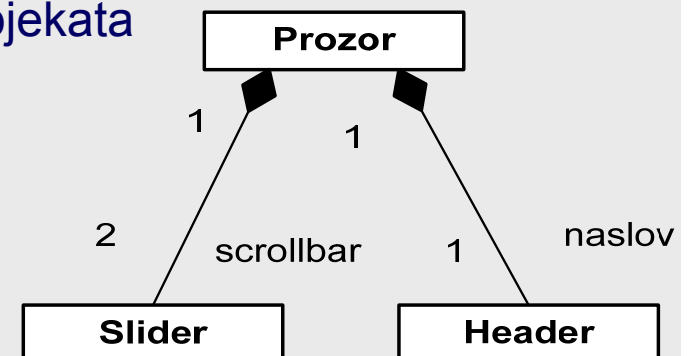
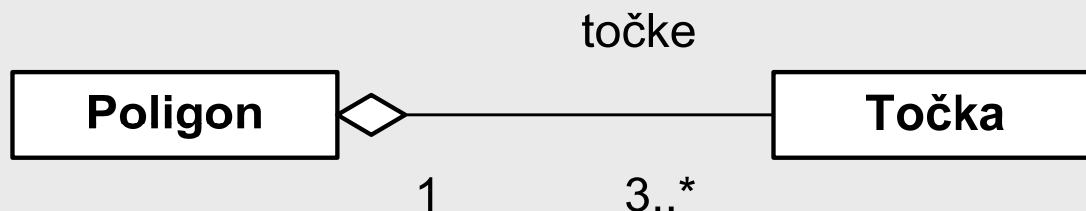
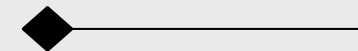
❑ Agregacija (aggregation), sadržavanje, obuhvaćanje

- poseban oblik asocijacije: veza "dio-od", "cjelina-dio"
- Komponente su dijelovi objekta-agregata, takvi da
 - dio može nastati i postojati nezavisno od cjeline
 - uništenje "cjeline" ne uništava njene dijelove
- Objekt može pripadati u više cjelina
 - istu instancu nekog objekta može dijeliti (referencirati) više agregata
 - ali objekt (instanca) ne može biti svoj sastavni dio



❑ Kompozicija (Composition)

- "stroža" varijanta agregacije
- objekt može pripadati samo jednoj cjelini
- elementi postoje tako dugo dok postoji cjelina
 - kaskadno brisanje, tj. uništenje sastavnih objekata



Primjer agregacije i rekurzije

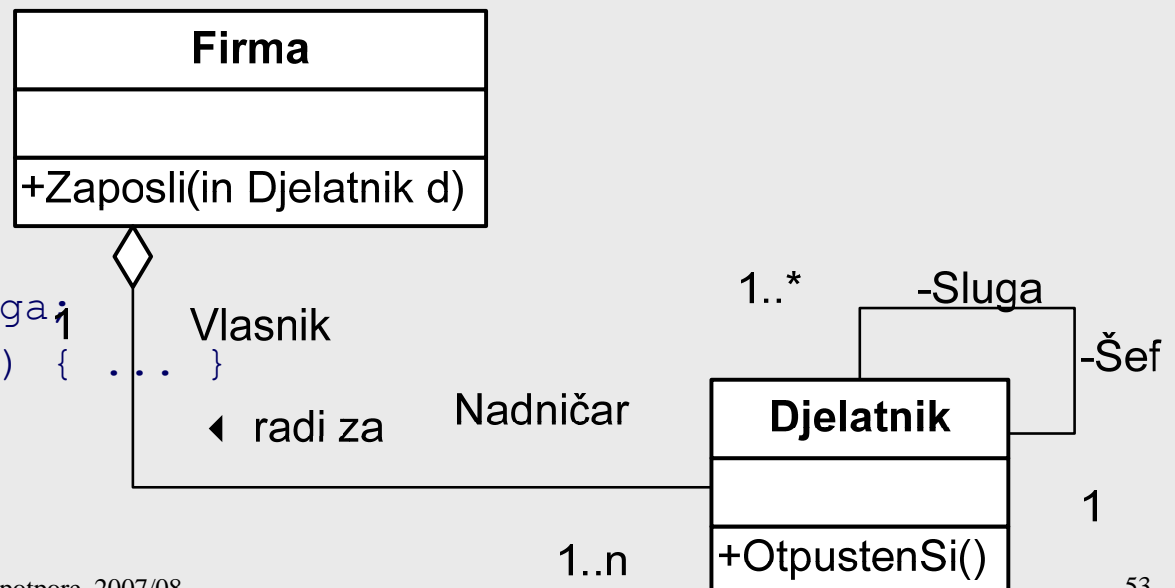
```
class Firma
{
    int brojDjelatnika = 0;
    private Djelatnik[] nadnicar = new
        Djelatnik[3];
    public void MolimPovisicu(Djelatnik d) { }

    public void Zaposli(Djelatnik d)
    {
        nadnicar[brojDjelatnika++] = d;
    }
    ...
}
```

```
class Djelatnik {
    private Firma vlasnik;
    private Djelatnik sef;
    private Djelatnik[] sluga;
    public void OtpustenSi() { ... }
}
```

Razredi\Agregacija

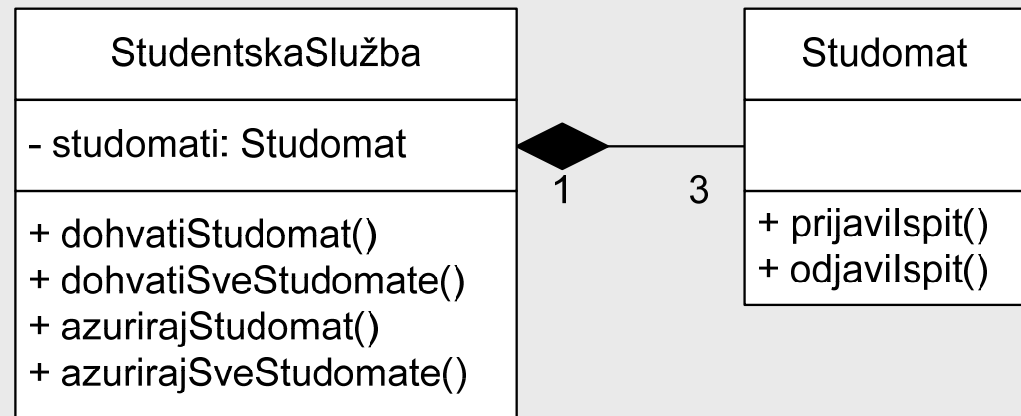
- Dijagram
- Prikaz člana i prikaz asocijacije



Primjer kompozicije

```
class StudentskaSluzba
{
    private Studomat[] studomati;
    public StudentskaSluzba() {
        studomati = new Studomat[3];
        studomati[0] =
            new Studomat();
    }
    ~StudentskaSluzba()
    {
        studomati[0] = null;
        // delete[] studomati;
    }
    void dohvatiStudomat() { }
    void dohvatiSveStudomate() { }
    void azurirajStudomat() { }
    void azurirajSveStudomate() { }
}
```

```
class Studomat{
    Studomat() {}
    ~Studomat() {}
    void prijaviIspit() {}
    void odjaviIspit() {}
};
```



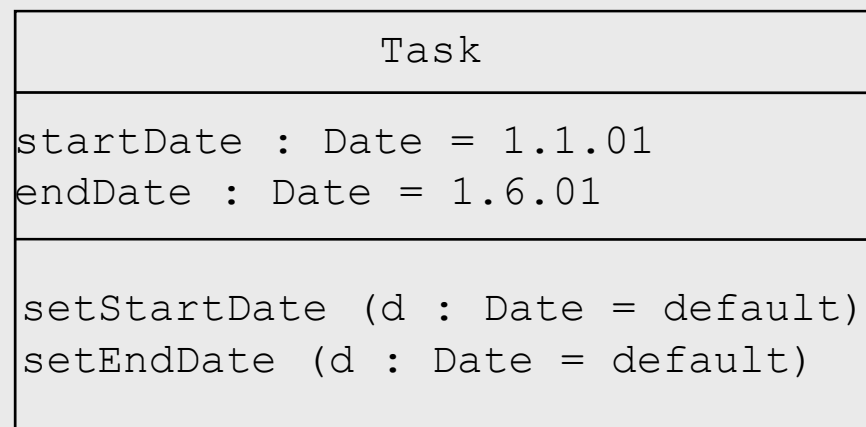
Razredi i Agregacija

- Dijagram
- Prikaz člana i prikaz asocijacije

Dijagram objekata (Object diagram)

❑ Dijagram objekata

- Koriste se za ilustraciju ili objašnjenje nekog trenutka

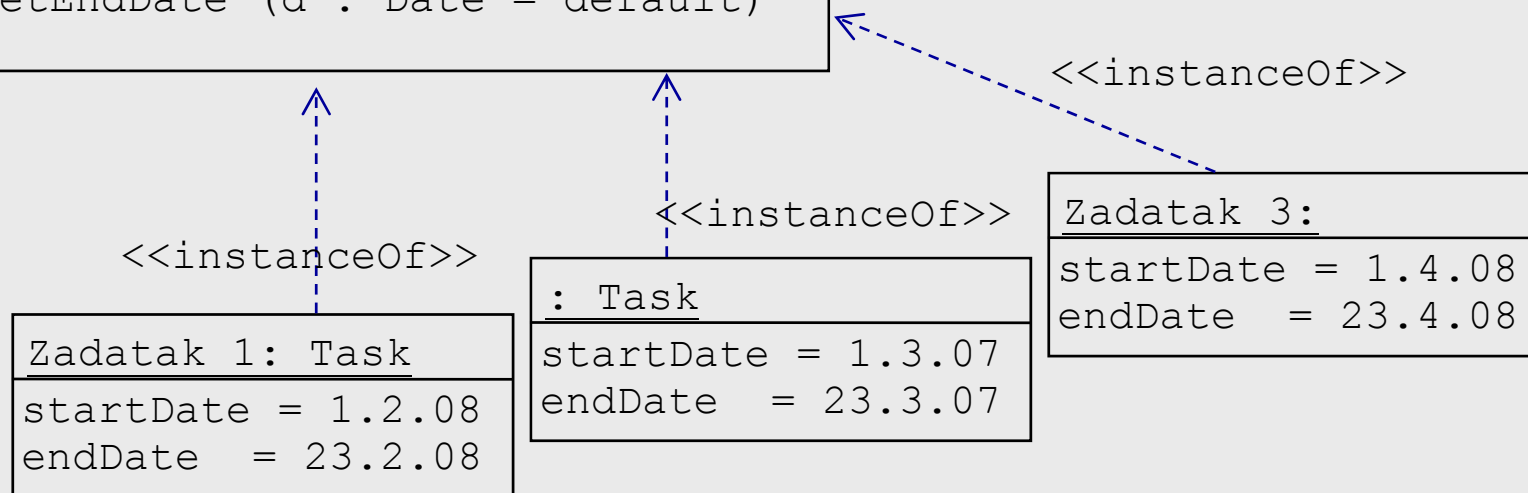


❑ Prikazuje se

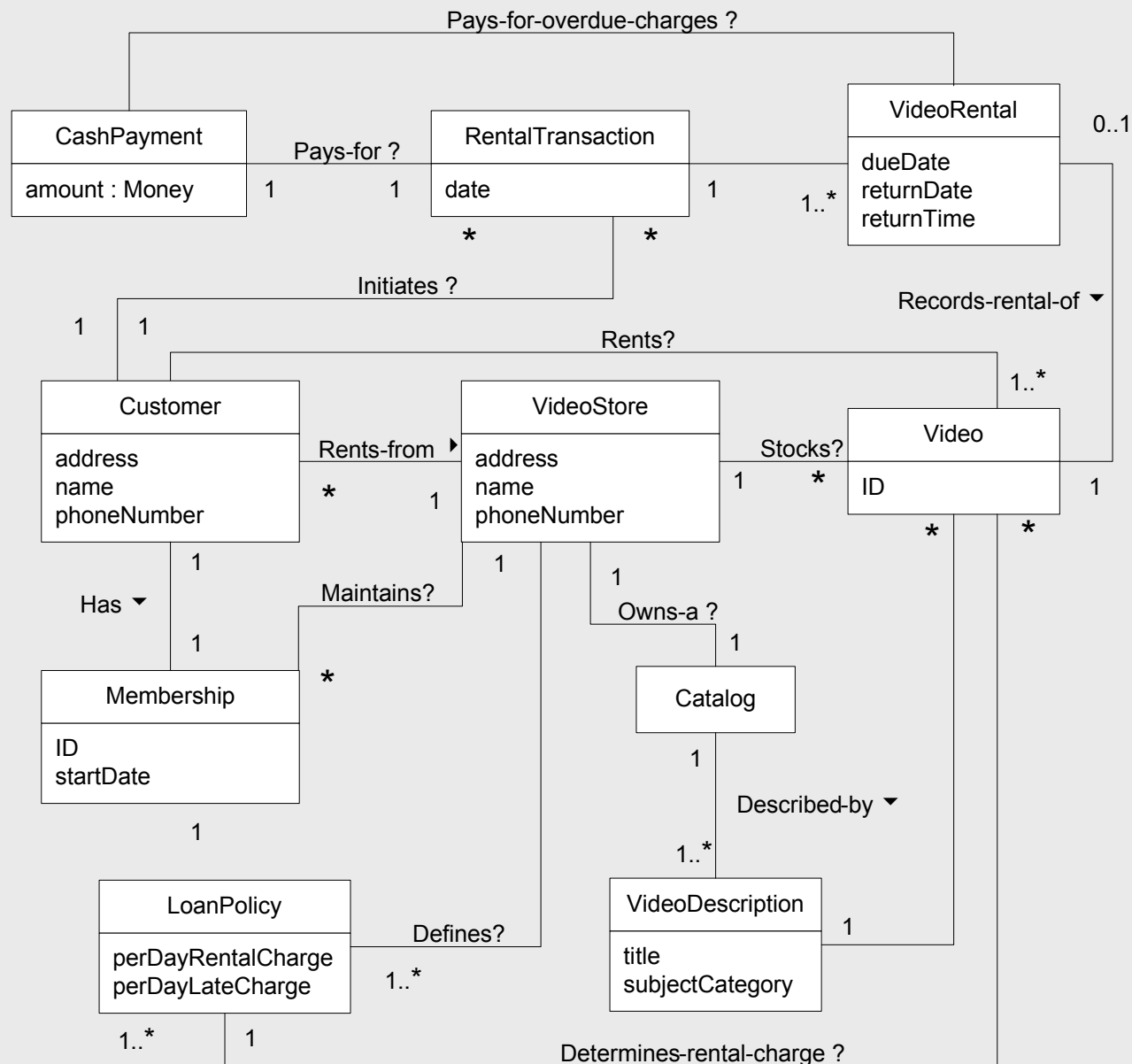
- Naziv objekta (potcrtano)
- Naziv razreda (opcionalno)
- Vrijednost atributa (opcionalno)

❑ Varijante

- imenovana instanca
- neimenovana instanca
- siroče



Zadatak za vježbu



Reference

Resursi

- UML Quick Reference Card
- UML Reference Card

UML i alati

- <http://www.omg.org/uml/>
- <http://www.intelinfo.com>
- <http://www.objectteering.com/>
- <http://www.proxysource.com/>
- <http://argouml.tigris.org/>
- <http://www.sparxsystems.com.au/>

Literatura (Google ...)

- S.W.Ambler: UML Class Diagram Guidelines, Rational, 2002.
- Pascal Roques: From the UML design model to C#
- Hans-Erik Eriksson and Magnus Penker: Design Java Apps with UML

Nasljeđivanje, višeeobličje ...

Nasljeđivanje

- ❑ **Omogućuje izvođenje (deriviranje, ne pokretanje) novog razreda na temelju postojećeg razreda**
 - `class DerivedClass: BaseClass { }`
- ❑ **Generalizacija (i pripadna specijalizacija)**
 - osnovni razred – bazni razred (base class), superrazred (superclass), roditelj
 - izvedeni razred – derivirani (derived), podrazred (subclass), dijete
- ❑ **Dijete nasljeđuje članove roditelja i definira vlastite članove**
- ❑ **Višeobličje (polymorphism)**
 - Operacija deklarirana u osnovnom razredu biva poništena u korist operacije jednako deklarirane u naslijeđenom razredu.
 - Podtip se ponaša drukčije odnosu na nadtip ili neki drugi podtip.
- ❑ **Zamjenjivost (supstitutability)**
 - Razred je podtip osnovnog razreda ako je u aplikaciji moguće zamijeniti osnovni razred podtipom, a da aplikacija normalno nastavi s radom.
 - Drugim riječima, ako postoji kôd koji se odnosi na nadtip `Kupac`, umjesto njega može se supstituirati bilo koji njegov podtip

Primjer nasljeđivanja

```
class Kupac {  
    public void PisiAdresu() { ... }  
}  
class PravOsoba: Kupac {  
    public string VrstaRacuna() { return "R-1"; }  
}  
class FizOsoba: Kupac {  
    public string VrstaRacuna() { return "R-2"; }  
}  
...  
FizOsoba f = new FizOsoba();  
f.pisiAdresu();           // inherited from base class  
  
string s = f.VrstaRacuna(); // introduced in derived class  
  
Kupac k = f;              // subclass as base class  
k.pisiAdresu();
```

Nasljeđivanje u programskom jeziku C#

- ❑ U programskom jeziku C# moguće je naslijediti samo jedan razred, a implementirati više sučelja.

```
class Osoba : Partner, IAdresa, IRazred { ... }
```

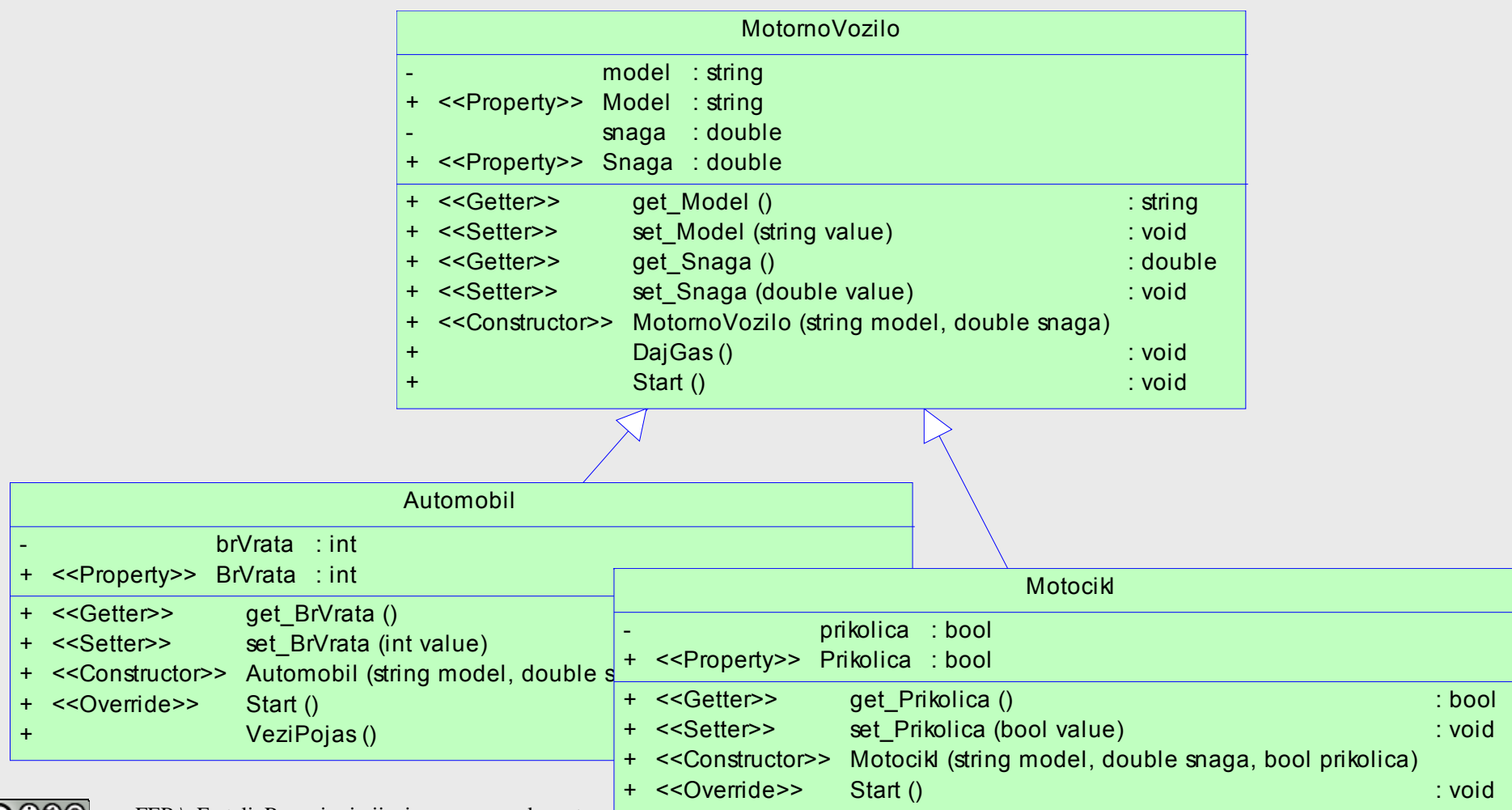
- ❑ **public** varijable i postupci se nasljeđuju
- ❑ **private** varijable i postupci ne mogu biti naslijeđeni
- ❑ **sealed** razred ne može biti naslijeđen

```
sealed class SealedPoint { ... }  
class MyPoint : SealedPoint // pogreška prevođenja
```

- ❑ **base.member** pristup nadjačanim članovima
 - Npr. `base.F()` ;
- ❑ Instanciranje izvedenog razreda izaziva poziv konstruktora osnovnog razreda
- ❑ Prvo se poziva destruktore izvedenog razreda, a zatim destruktore osnovnog razreda

Preopterećenje postupaka (method overloading)

- `virtual` deklarira virtualni postupak roditelja koji može biti nadjačan
- `override` deklarira postupak djeteta koji nadjačava, a može koristiti nadjačani postupak



Primjer nasljeđivanja

❏ Primjer Razredi\MotornoVozilo

```
class MotornoVozilo
{
    // atributi
    private string model;
    public string Model { ... }
    private double snaga;
    public double Snaga { ... }

    // konstruktor
    public MotornoVozilo(string model, double snaga) { ... }

    // postupak (zajednicki svim vozilima)
    public void DajGas() { ... }

    // postupak (koji izvedene klase implementiraju svaka na svoj
    nacin)
    public virtual void Start() { ... }
}
```

Primjer nasljeđivanja

```
class Automobil : MotornoVozilo
{
    // dodatni atribut
    private int brVrata;
    public int BrVrata { ... }

    // konstruktor
    public Automobil(string model, double snaga, int brVrata)
        : base(model, snaga) // poziv osnovnog konstruktora
    { ... }

    // nadjačavanje baznog postupka Start()
    public override void Start() { ... }

    // dodatni postupak za Automobil
    public void VeziPojas() { ... }
}
```

❑ Što sve nasljeđuje Automobil?

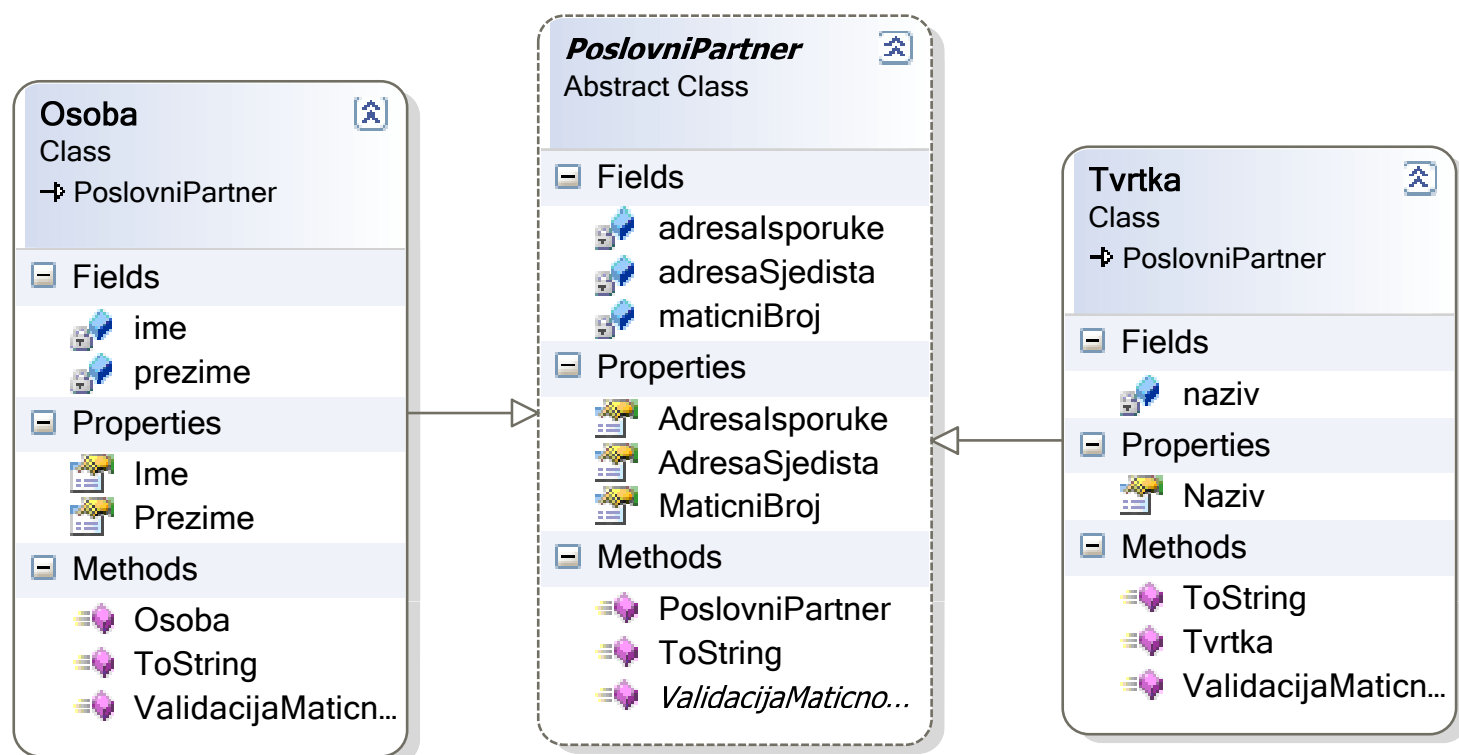
Apstraktni razredi

❑ abstract – apstraktni razred

- Osnovni razred je nedovršen
- Apstraktni postupci nisu ugrađeni
- Dijete mora u potpunosti ugraditi nedovršene postupke
- Ne može se instancirati objekt apstraktnog razreda

❑ Primjer Razredi\PoslovniPartner

```
abstract class A {  
    public abstract F();  
}  
  
class B: A {  
    public override F() { ... }  
}
```



Primjer nasljeđivanja apstraktnog razreda

❑ Razred Osoba nasljeđuje razred PoslovniPartner

▪ Konstruktor

```
public Osoba(string maticniBroj, string adresaSjedista, string
adresaIsporuke, string ime, string prezime)
    : base(maticniBroj, adresaSjedista, adresaIsporuke){
    this.ime = ime; this.prezime = prezime;
}
```

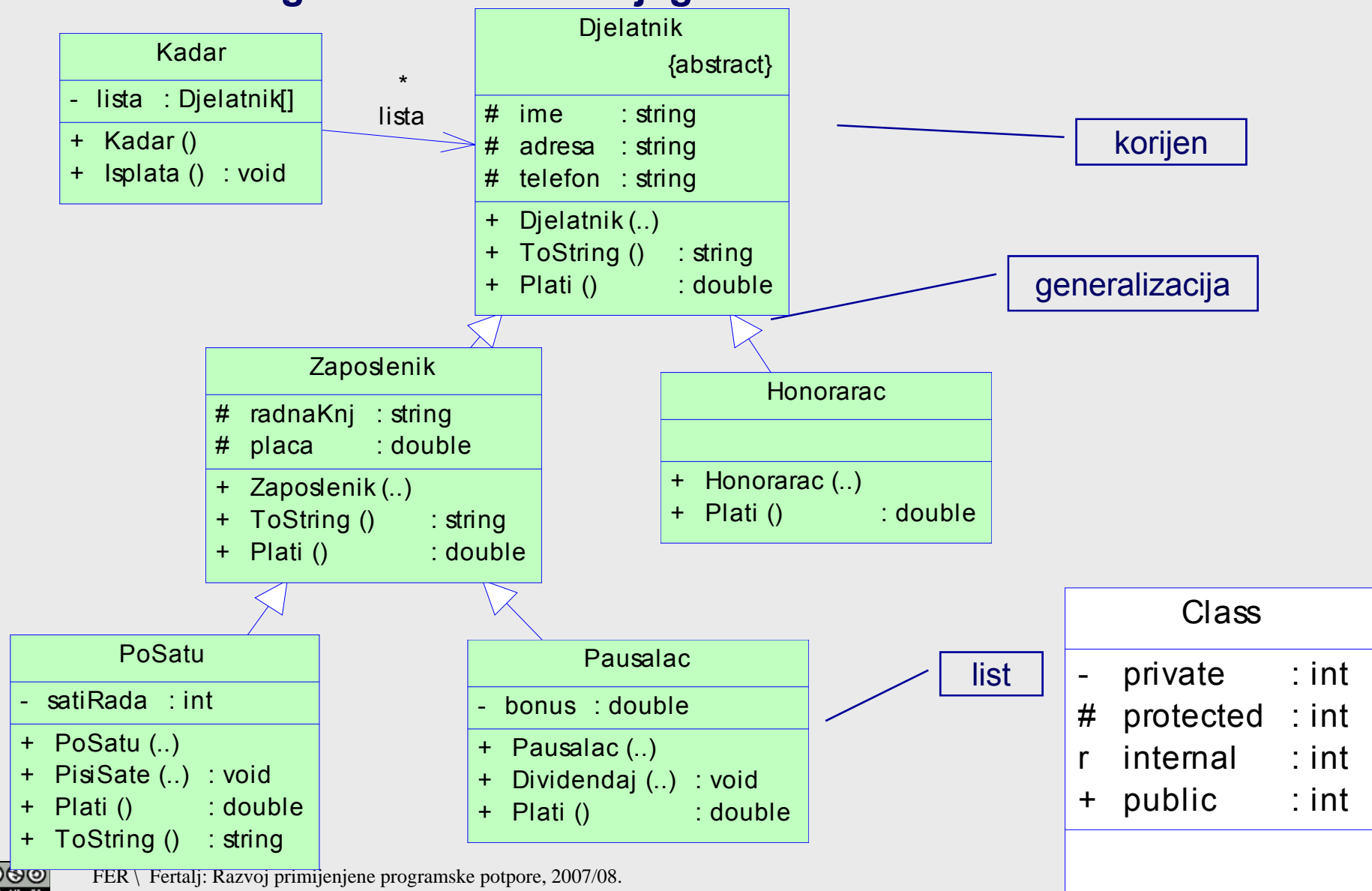
▪ Implementacija apstraktnog postupka

```
abstract class PoslovniPartner{
    public abstract bool ValidacijaMaticnogBroja();
    // zbog abstract, ovaj postupak potrebno je implementirati u
    // izvedenom razredu
}
```

```
class Osoba : PoslovniPartner{
    public override bool ValidacijaMaticnogBroja()
    { ...implementacija }
}
```

Hijerarhija nasljeđivanja

❑ Definirati i ugraditi razrede s dijagrama - 📁 Razredi\Kadrovsko



Dinamička klasifikacija

❑ Dinamičko povezivanje

- Referenca može pokazivati na različite objekte
- tip trenutnog objekta određuje postupak koji se obavlja

```
public class Kadar {
    private Djelatnik[] lista;

    public Kadar () {
        lista = new Djelatnik[6];
        lista[0] = new Pausalac ("Bobi", "Bobinje bb",
            "555-0469", "123-45-6789", 2423.07);
        ...
        ((Pausalac)lista[0]).Dividendaj (500.00); // cast
        ((PoSatu)lista[3]).PisiSate (40); // cast
    }
    public void Isplata() {
        double iznos;
        for ( int count=0; count < lista.Length; count++ ) {
            Console.WriteLine( lista[count] );
            iznos += lista[count].Plati(); // polymorphic
        }
    }
}
```

Kakva je ovo veza ?

- ❑ Umjesto nasljeđivanja, dijete može "učahuriti roditelja".

- ❑ Primjer  Razredi\Kompozicija

```
class Osoba
{
    PoslovniPartner poslovniPartner;
    private string ime;
    private string prezime;
    ...
}
```

PoslovniPartner		
-	maticniBroj	: string
-	adresaSjedista	: string
-	adresaSporuke	: string
+	<<Property>> MaticniBroj	: string
+	<<Property>> AdresaSjedista	: string
+	<<Property>> AdresaSporuke	: string
+	<<Getter>> get_MaticniBroj ()	: string
+	<<Setter>> set_AdresaSjedista (..)	: void
+	<<Getter>> get_AdresaSjedista ()	: string
+	<<Setter>> set_AdresaSporuke (..)	: void
+	<<Getter>> get_AdresaSporuke ()	: string
+	<<Constructor>> PoslovniPartner (..)	
+	ToString ()	: string

↑
poslovniPartner

Osoba		
-	poslovniPartner	: PoslovniPartner
-	ime	: string
-	prezime	: string
+	<<Property>> Ime	: string
+	<<Property>> Prezime	: string
+	<<Getter>> get_Ime ()	: string
+	<<Setter>> set_Prezime (..)	: void
+	<<Getter>> get_Prezime ()	: string
+	<<Constructor>> Osoba (..)	
+	ToString ()	: string

Sučelje (interface)

❑ Sučelje (interface) - specifikacija članova razreda bez implementacije

- Sadrži samo deklaracije operacija
- Može sadržavati postupke, svojstva, indeksere i događaje
- Svaki postupak je apstraktan (nema implementaciju)

❑ Notacija

- Standard za naziv: *INaziv*
- Proširena i skraćena (*lollipop*) notacija

❑ Sučelje definira obvezu ugradnje

- Razred koji nasljeđuje sučelje, mora implementirati sve postupke
- Sučelje može implementirati i apstraktni razred

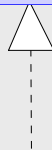
❑ Realizacija (realisation)

- veza izvornog razreda i sučelja
- sučelje može biti argument nekog postupka čime se postiže veća općenitost

Primjer sučelja i realizacije

Razredi\Sucelje

IPoint	
+ <<Property>>	x : int
+ <<Property>>	y : int
+ <<Getter>>	get_x () : int
+ <<Setter>>	set_x (int value) : void
+ <<Getter>>	get_y () : int
+ <<Setter>>	set_y (int value) : void



MyPoint	
-	myX : int
-	myY : int
+ <<Property>>	x : int
+ <<Property>>	y : int
+ <<Constructor>>	MyPoint (int x, int y)
+ <<Getter>>	get_x () : int
+ <<Setter>>	set_x (int value) : void
+ <<Getter>>	get_y () : int
+ <<Setter>>	set_y (int value) : void

InterfaceMain

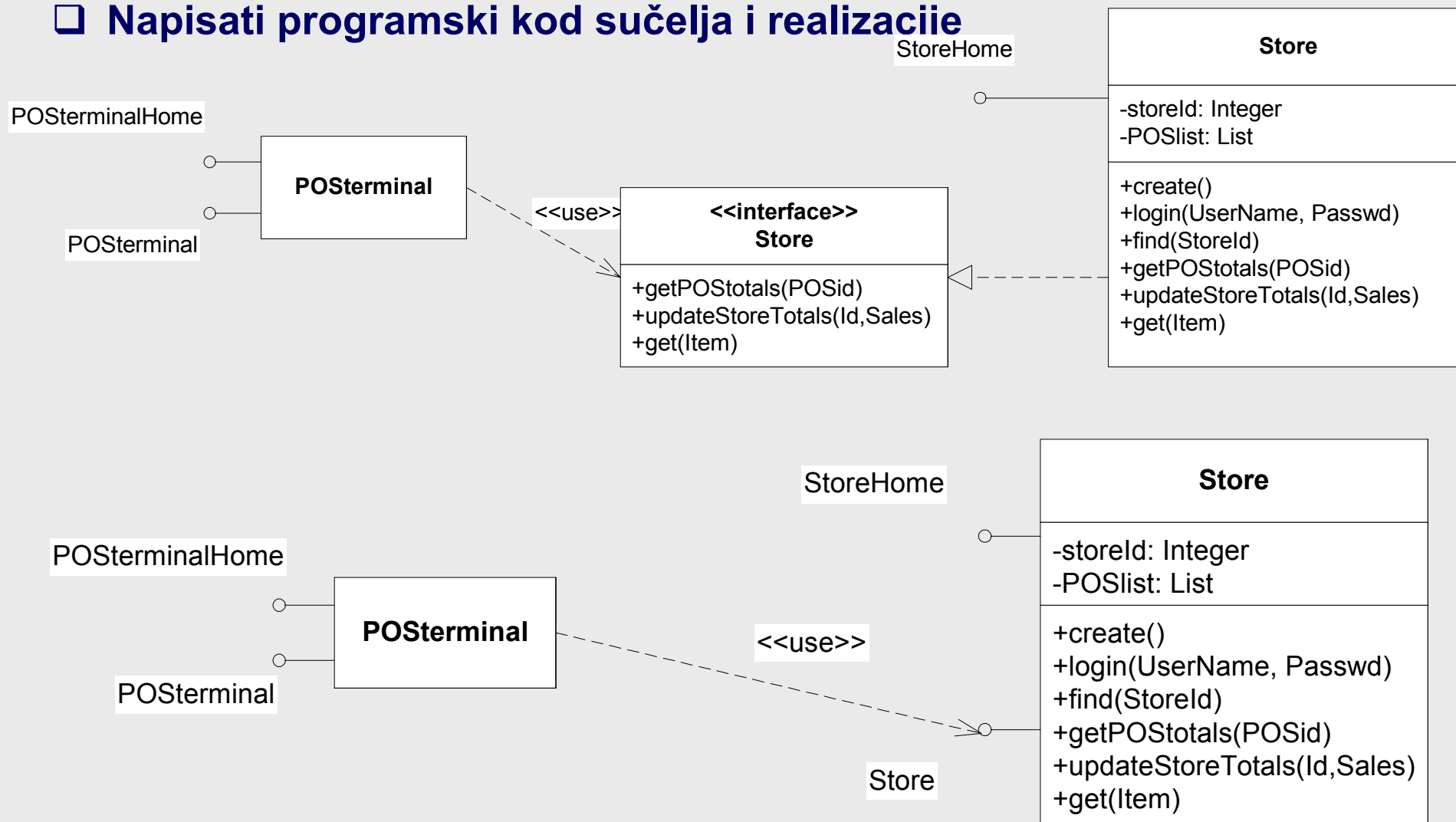
- PrintPoint (IPoint p) : void
+ Main () : void

```
interface IPoint {  
    // Property signatures:  
    int x { get; set; }  
    int y { get; set; }  
}
```

```
class MyPoint : IPoint {  
    public int x{  
        get { return myX; }  
        set { myX = value; }  
    }  
    public int y{  
        get { return myY; }  
        set { myY = value; }  
    }  
}
```

Zadatak za vježbu

- ❑ Ujedno primjer za različite notacije sučelja
- ❑ Napisati programski kod sučelja i realizacije



Imenik *System.Collections*

❑ Sadrži sučelja i razrede za rad s kolekcijama

- Kolekcija je skup povezanih objekata.


❑ Sučelja: **ICollection**, **IEnumerator**, **IEnumerable**, **IDictionary** i **IList**

- Određuju osnovne funkcionalnosti kolekcija
- Razred koji implementira jedno ili više tih sučelja naziva se kolekcija.

❑ Razredi:

ArrayList	implementira <i>IList</i> sučelje pomoću polja za pohranu objekata čija se veličina po potrebi povećava
BitArray	Radi s poljem bitova koji su predstavljeni kao <i>Boolean</i> vrijednosti (<i>true</i> predstavlja 1 a <i>false</i> 0)
Hashtable	Predstavlja kolekciju vrijednosti ključ-i-vrijednost parova (vrijednost je objekt pohranjen u kolekciji) organiziranih pomoću <i>hash</i> vrijednosti ključa
Queue	Predstavlja first-in-first-out kolekciju objekata
SortedList	Predstavlja kolekciju ključ-i-vrijednost parova poredanih (sortiranih) po ključu, uz moguć dohvat po ključu i indeksu
Stack	Predstavlja last-in-first-out kolekciju objekata

ArrayList

- ❑ **Niz čija se duljina dinamički povećava koliko je potrebno**
- ❑ **Isti niz može sadržavati različite tipove podataka**
- ❑ **Neka svojstva i postupci:**
 - `Contains` – provjerava nalazi li se zadani objekt u kolekciji
 - `Clear` – briše sve elemente iz kolekcije
 - `Insert` – ubacuje zadani objekt na zadano mjesto kolekcije
 - `Remove` – briše prvo pojavljivanje zadanog objekta iz kolekcije
 - `Add` – dodaje zadani objekt na kraj kolekcije i vraća indeks
 - `IndexOf` – vraća položaj zadanog objekta unutar kolekcije ili dijela kolekcije
 - `Item` – indekserski za vraćanje ili postavljanje objekata u kolekciju po indeksu
- ❑ **Primjer**  **Razredi\Kolekcije**

```
ArrayList lista = new ArrayList();  
lista.Add(1);  
lista.Add("abc");  
if (lista.Contains("abc")) Console.WriteLine("Lista sadrži abc");  
Console.WriteLine("Lista sadrži " + lista.Count + " elemenata");  
lista.Clear();
```

Stack, Queue

❑ **stack (stog) – Neka svojstva i postupci:**

- Push – stavlja zadani objekt na stog
- Peek – vraća objekt s vrha stoga bez brisanja
- Pop – vraća objekt s vrha stoga uz brisanje

❑ **Primjer** **Razredi\Kolekcije**

```
Stack stog = new Stack();  
stog.Push("abc");  
stog.Push(123);  
Console.WriteLine("Stog sadrži " + stog.Count.ToString()  
                  + " elemenata.");  
Console.WriteLine("S vrha stoga (pop): " + stog.Pop().ToString());  
Console.WriteLine("S vrha stoga (peek): " + stog.Peek().ToString());
```

❑ **Queue (red) – Neka svojstva i postupci**

- Enqueue – stavlja zadani objekt na kraj reda
- Peek – vraća objekt s početka reda bez brisanja
- Dequeue – vraća objekt s početka reda uz brisanje

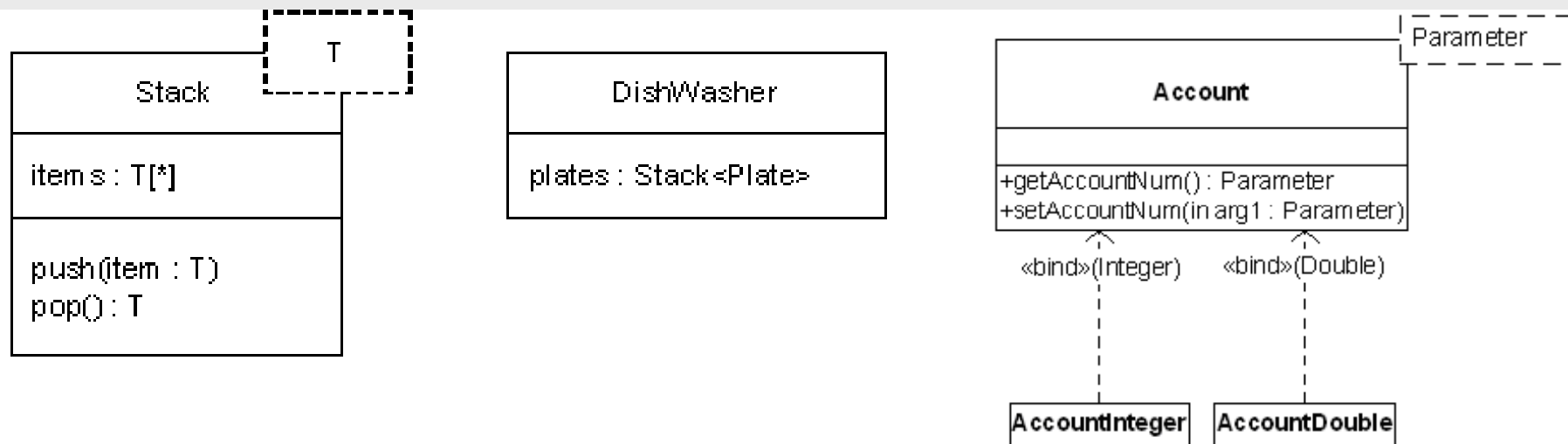
Parametrizirani razred

❑ **Generics općenito**

- Mehanizam koji omogućava definiranje općenitog programskog koda (razreda, postupka, sučelja, ...) za rukovanje različitim tipovima podataka.

❑ **Parametrizirani razred (parametrized class, template class)**

- Predložak koji definira porodicu razreda
- Konkretni razred povezuje se navođenjem aktualnog tipa



```
public class Stack <T> {
```

```
    ArrayList items;
```

```
    public void push(T item) {
```

```
public class DishWasher {
```

```
    Stack<Plate> plates;
```

System.Collections.Generic

❑ **Prostor imena `System.Collection.Generics`**

- Sadrži sučelja i razrede za generičke kolekcije
- Funkcionalnosti negeneričkih kolekcija
- Bolja tipska sigurnost (type safety) i bolje performanse od negeneričkih

❑ **Neki razredi prostora imena `System.Collections.Generics`**

- `public class List<T> : IList<T>, ICollection<T>, IEnumerable<T>, IList, ICollection, IEnumerable`
- `public class Queue<T> : IEnumerable<T>, ICollection, IEnumerable`
- `public class Stack<T> : IEnumerable<T>, ICollection, IEnumerable`
- `public class Dictionary<TKey,TValue> : IDictionary<TKey,TValue>, ICollection<KeyValuePair<TKey,TValue>>, IEnumerable<KeyValuePair<TKey,TValue>>, IDictionary, ICollection, IEnumerable, ISerializable, IDeserializationCallback`

Primjer parametriziranog razreda

❑ Primjer Razredi\Generics

```
using System.Collections.Generic;

public class Stog<T>
{
    T[] elementi;
    public void Stavi(T element) {...}
    public T Skini() {...}
}

Stog<int> stogInt = new Stog<int>();
Stog<string> stogString = new Stog<string>();
Stog<Automobil> stogAutomobil = new Stog<Automobil>();
```

System.Collections.Generic

❑ **List(T)** – parametrizirana kolekcija objekata kojima se može pristupati preko indeksa u listi

- Npr List<int>, List<string>, List<Osoba>
- Funkcionalnost razreda ArrayList

```
List<string> listaString = new List<string>();  
listaString.Add("jedan");  
...  
string trazen = ...  
if (listaString.Contains(trazen))  
    Console.WriteLine("Element postoji u listi na " +  
        listaString.IndexOf(trazen) + ". mjestu.");  
else  
    Console.WriteLine("Element ne postoji u listi!");
```

Tablice s raspršenim adresiranjem i rječnici

❑ HashTable

- System.Collection
- kolekcije ključeva/vrijednosti organizirana na temelju hash koda ključeva
- Ključ i vrijednost su tipa *object*

```
Hashtable a = new Hashtable(10);  
//10 je predviđeni max. br. elem.  
  
a.Add(100, "Jedan");  
//ključ mora biti jedinstven!  
a.Add(200, "Dva");  
  
foreach(DictionaryEntry d in a){  
    Console.WriteLine(d.Value);  
}
```

❑ Dictionary(Tkey, TValue)

- System.Collection.Generics
- Ista funkcionalnost kao Hashtable
- Ključ i vrijednost su tipizirani

```
Dictionary<int, string> a =  
    new Dictionary<int, string>(10);  
  
a.Add(100, "Jedan");  
a.Add(200, "Dva");  
  
foreach(KeyValuePair<int,string>  
        d in a){  
    Console.WriteLine(d.Value);  
}
```

❑ Primjer Razredi\HashTablice

Prednosti generičkih kolekcija

❑ Negeričke kolekcije objekte pohranjuju kao `System.Object`

- Takva pohrana je prednost sa stanovišta fleksibilnosti
- Nedostatak je obavljanje *boxinga* prilikom dodavanja `value` tipa podataka (dobivanje reference na `value` podatak da bi se s tim podatkom moglo postupati kao sa `System.Object`)
 - pad performansi za veće količine podataka
 - potrebna konverzija (*unboxing*) da se dobije smisleni podatak

❑ Generičke kolekcije su tipizirane

- Bolje performanse – ne mora se obavljati *boxing/unboxing*
- Tipska sigurnost (*type safety*) – zaštita od unosa podataka nepoželjnog tipa i pogrešaka koje pritom mogu nastati (npr. pokušaj izvršavanja naredbe neprikladne za dani tip podataka)

Zadaci za vježbu

- ❑ **Napisati razred DjelatnikCollection - kolekciju djelatnika**
 - Naslijediti IList sučelje

Reference

❑ C# School Book – Free ebook

- <http://www.programmersheaven.com/2/CSharpBook>

❑ Objects, Classes, and Structs (C# Programming Guide)

- Objects, Classes, Structs, Inheritance, ...
- <http://msdn2.microsoft.com/en-us/library/ms173109.aspx>

❑ Generics (C# Programming Guide)

- [http://msdn2.microsoft.com/en-us/library/ms379564\(vs.80\).aspx](http://msdn2.microsoft.com/en-us/library/ms379564(vs.80).aspx)