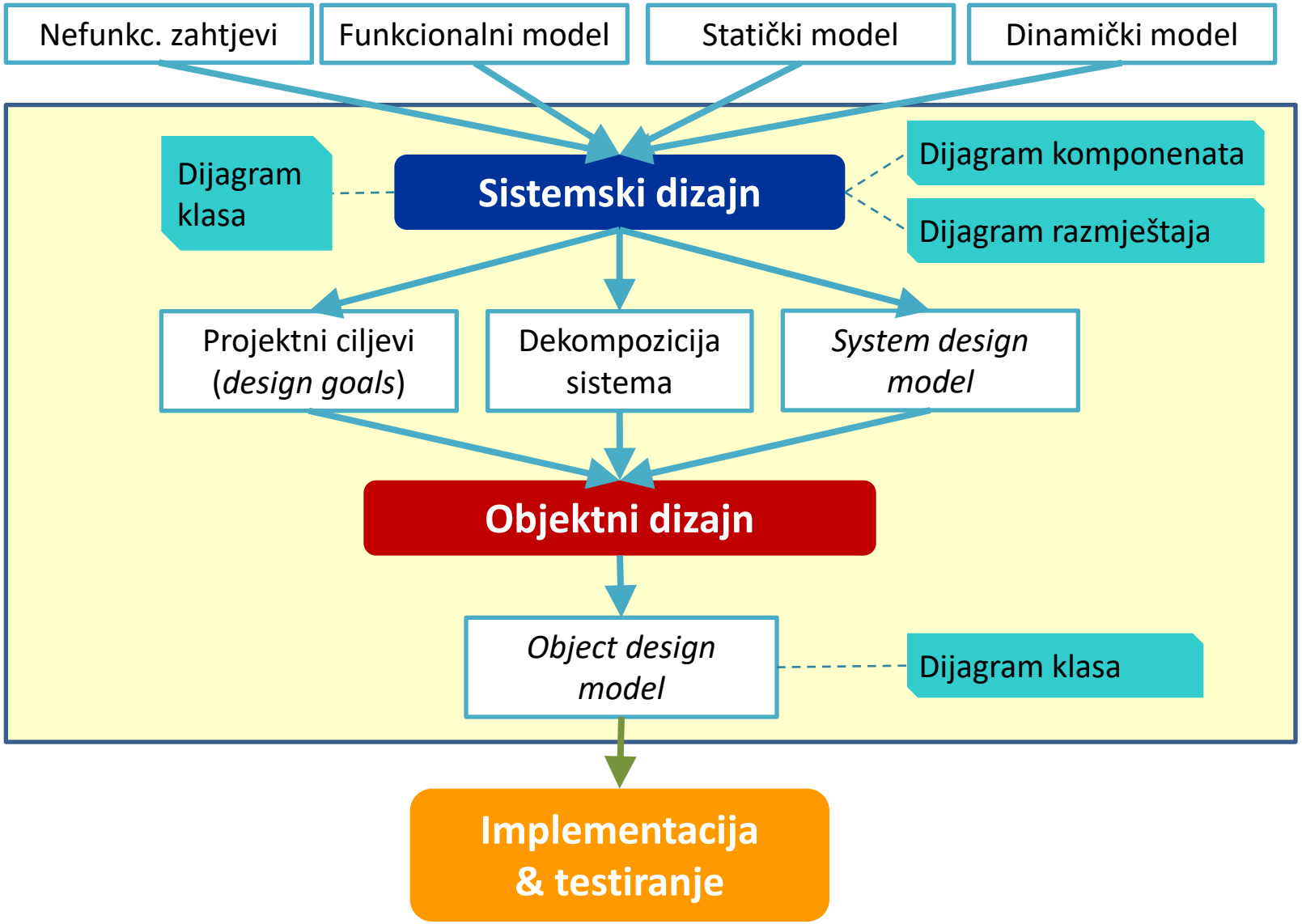
UNIVERZITET U BANJOJ LUCI
ELEKTROTEHNIČKI FAKULTET

Prof. dr Dražen Brđanin

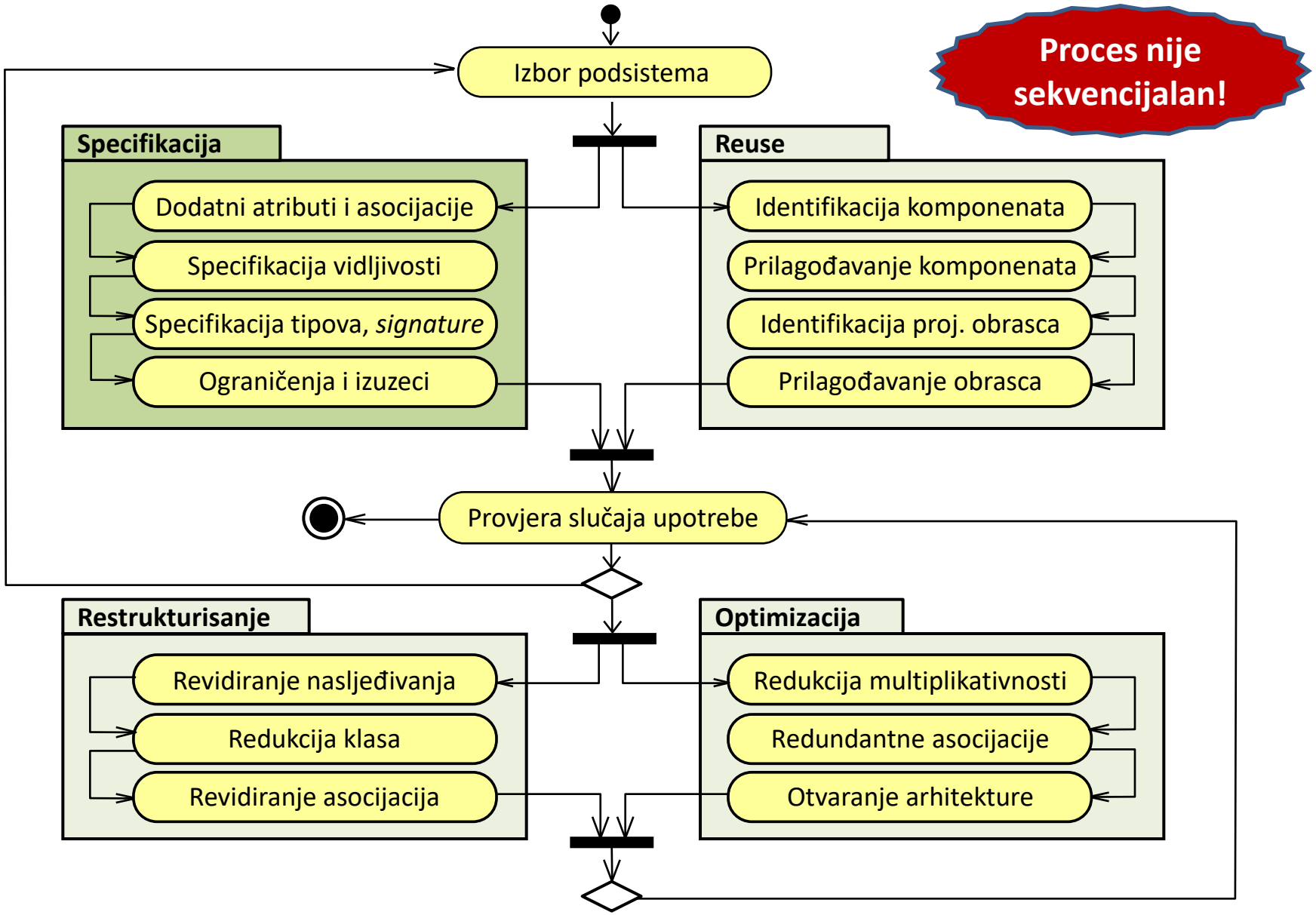
OBJEKTNO-ORIJENTISANI DIZAJN
/specifikacija interfejsa/

Banja Luka
2024.

Objektno-orijentisano projektovanje



Aktivnosti u objektnom dizajnu



Specifikacija interfejsa

Specifikacija interfejsa

- Precizne specifikacije interfejsa za svaku komponentu (**component API**) i za svaku klasu
- **Osnovni ciljevi:**
  - smanjenje stepena sprege između podsistema/klasa,
  - specifikacija što jednostavnijeg interfejsa koji je razumljiv za svakog pojedinačnog programera

OOA:

- Identifikacija atributa i operacija bez specifikacije tipova i argumenata

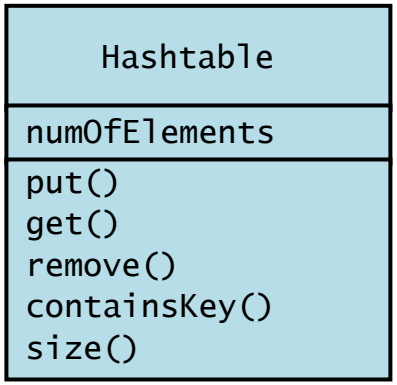
OOD:

- **Specifikacija tipova i argumenata (*signature*)**
- **Specifikacija vidljivosti** (*public*, *protected*, *private*)
- **Specifikacija ograničenja (*contracts*)**
  - uslovi koji moraju da budu ispunjeni:
    - za svaku instancu klase,
    - prije pozivanja metode,
    - poslije izvršavanja metode
  - specifikacija rezultata koji metoda vraća kao rezultat izvršavanja

Specifikacija tipova/argumenata (*signature*)

OOA:

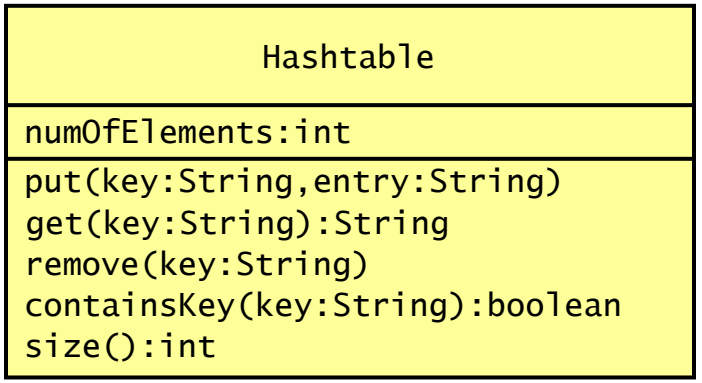
Identifikacija atributa i operacija bez specifikacije tipova i argumenata

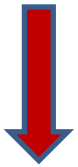


Inicijalna specifikacija

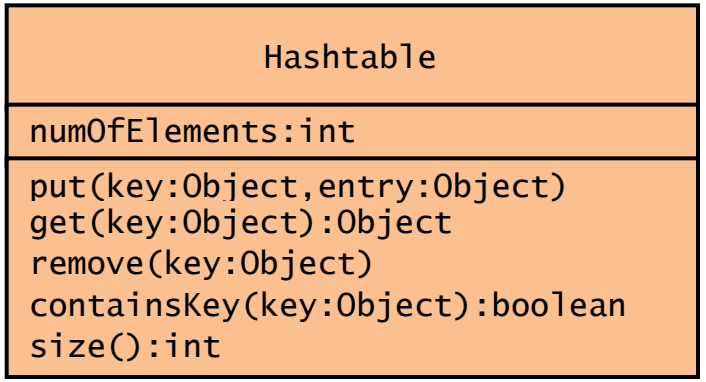
OOD:

Specifikacija tipova i argumenata





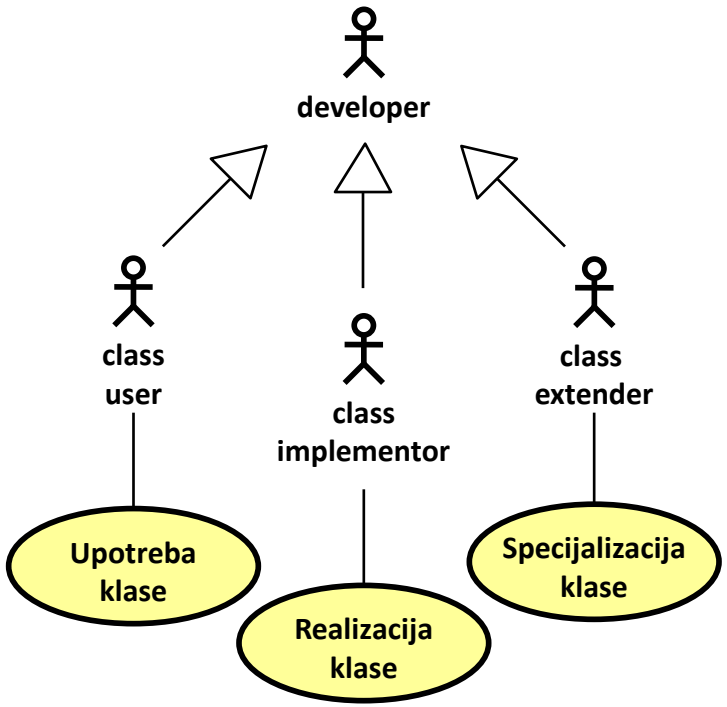
Naknadna specifikacija uključuje uopštavanja (generalizacije)



Specifikacija vidljivosti

Specifikacija interfejsa iz perspektive developera

Uloge *developera* u odnosu na klasu:



class implementor

- odgovoran za realizaciju (implementaciju) klase
- specifikacija interfejsa za njega predstavlja radni zadatak (implementira operacije)

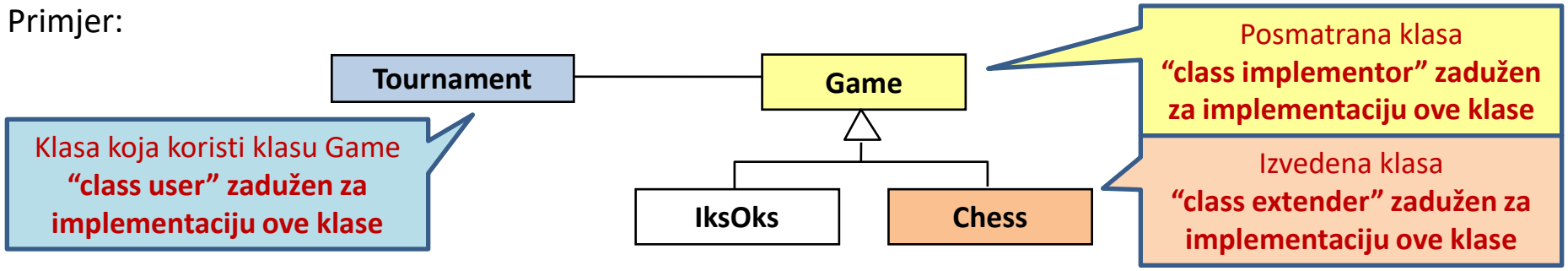
class user

- koristi klasu za realizaciju aplikativnog koda ili za realizaciju druge klase koja agregira objekte date klase ili poziva operacije date klase
- specifikacija interfejsa za njega predstavlja specifikaciju onog što mu je dostupno iz klase, onog što mora da ima u vidu kad koristi datu klasu

class extender

- specijalizuje posmatranu klasu
- fokusira se na postojeću specifikaciju interfejsa posmatrane klase, jer je ona osnov za specifikaciju interfejsa specijalizovane klase

Primjer:



Specifikacija vidljivosti

Nivoi vidljivosti članova klase

Private (-)

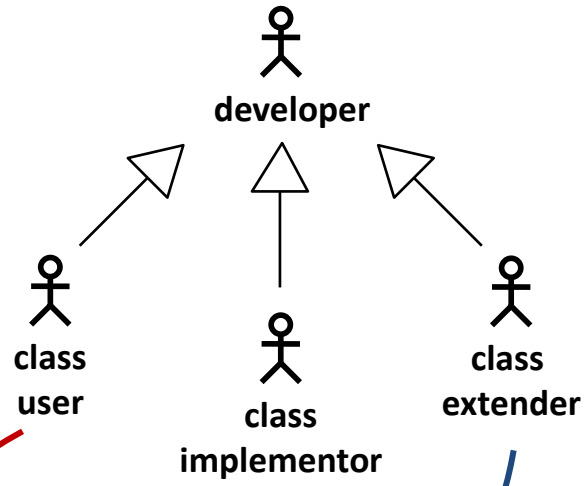
- Privatni atributi su **dostupni samo iz unutrašnjosti klase** kojoj pripadaju (može da im se pristupi samo iz operacija koje pripadaju istoj klasi).
- Privatne operacije su **dostupne samo iz unutrašnjosti klase** kojoj pripadaju (mogu da ih pozivaju samo operacije koje su definisane u istoj klasi).
- Privatni atributi i operacije **nisu dostupni izvan klase** kojoj pripadaju i ne može da im se pristupi iz drugih klasa.

Protected (#)

- Zaštićeni atributi i operacije **dostupni su iz unutrašnjosti klase** kojoj pripadaju (može da im se pristupi iz drugih operacija koje pripadaju istoj klasi), ali **i iz klasa koje su izvedene iz date klase**.

Public (+)

- Za javne članove klase **ne postoje ograničenja u dostupnosti** (dostupni su i spolja, iz unutrašnjosti klase, kao i svih izvedenih klasa).



Poseban značaj za class extendera imaju protected članovi

Korisnici klase imaju mogućnost pristupa samo javnim članovima klase

Specifikacija vidljivosti

Primjer (Implementacija vidljivosti):

```
public class Tournament
{
    protected int maxNumPlayers;
    protected List players;
    public Tournament(int maxPlayers) {// ...}
    public int getMaxNumPlayers() {// ...}
    public List getPlayers() {// ...}
    public boolean addPlayer(Player p) {// ...}
    public void removePlayer(Player p) {// ...}
    private boolean isAcceptable(Player p) {// ...}
}
```

Specifikacija vidljivosti

Heuristike za skrivanje informacija

- **Skrivanje informacija** nije predmet razmatranja tokom analize, ali je **veoma bitan zahtjev u projektovanju**!
- Interfejse treba definisati veoma oprezno (**što manji broj argumenata**), bilo da je riječ o klasama ili podsistemima (treba nastojati da se primjenjuje **FASADA/INTERFEJSNI OBJEKAT**)
- **Primjena principa "TREBA DA ZNA"**
  - član klase treba da je vidljiv samo ako neko "treba da zna"
- **Korisnik klase treba da zna što manje detalja o klasi (BLACK BOX)**
  - tada je lakše mijenjati datu klasu, jer je manja sprega sa drugim klasama
  - izmjene na datoj klasi imaće manji uticaj na spregnute klase
- **"Trade-off": SKRIVANJE INFORMACIJA ↔ EFIKASNOST**
  - pristup privatnim članovima usporava rad (kritično u *real-time* sistemima, igrama, ...)

Specifikacija ograničenja

Contracts

- **Kontrakt specifikuje ograničenja koja korisnik klase mora da zadovolji da bio koristio klasu, kao i ograničenja koja *class implementor* i *class extender* moraju da obezbijede u implementaciji klase.**
- **Kontrakt uključuje tri tipa ograničenja**:
- **invarijanta** (eng. *invariant*)
  - invarijanta je predikat koji je istinit za sve instance date klase
  - invarijante su ograničenja vezana za klase (interfejse)
  - invarijante se koriste za specifikaciju ograničenja između atributa
- **preduslov** (eng. *precondition*) – "**PRAVA**"
  - preduslov je predikat koji mora biti istinit prije poziva operacije
  - preduslov je vezan za konkretnu operaciju
  - preduslov se koristi za specifikaciju ograničenja koje korisnik klase mora da zadovolji prije poziva konkretne operacije
- **postuslov** (eng. *postcondition*) – "**OBAVEZE**"
  - postuslov je predikat koji mora biti istinit po završetku izvršenja operacije
  - postuslov je vezan za konkretnu operaciju
  - postuslov se koristi za specifikaciju ograničenja koje *class implementor/extender* mora da obezbijedi po završetku izvršenja operacije

Specifikacija ograničenja

Ako su invarijante, preduslovi i postuslovi jednoznačni, tada se kontrakt naziva FORMALNA SPECIFIKACIJA.

Primjeri ograničenja

invarijanta

Maksimalan broj igrača na turniru mora biti pozitivan (ako se kreira instanca u kojoj je maxNumPlayers=0, biće narušen kontrakt)

$\forall$**t : typeof(t)=Tournament** $\Rightarrow$ **t.getMaxNumPlayers()>0**

preduslov

Da bi se novi igrač mogao priključiti turniru, moraju da budu zadovoljena dva uslova:

1. dati igrač se još ne nalazi na listi uključenih igrača,
2. još nije dostignut maksimalan broj igrača

$\forall$**t : typeof(t)=Tournament** $\wedge$ $\forall$**p : typeof(p)=Player**

$\Rightarrow$ **!t.isAccepted(p)** $\wedge$

t. getNumPlayers()<t. getMaxNumPlayers()

postuslov

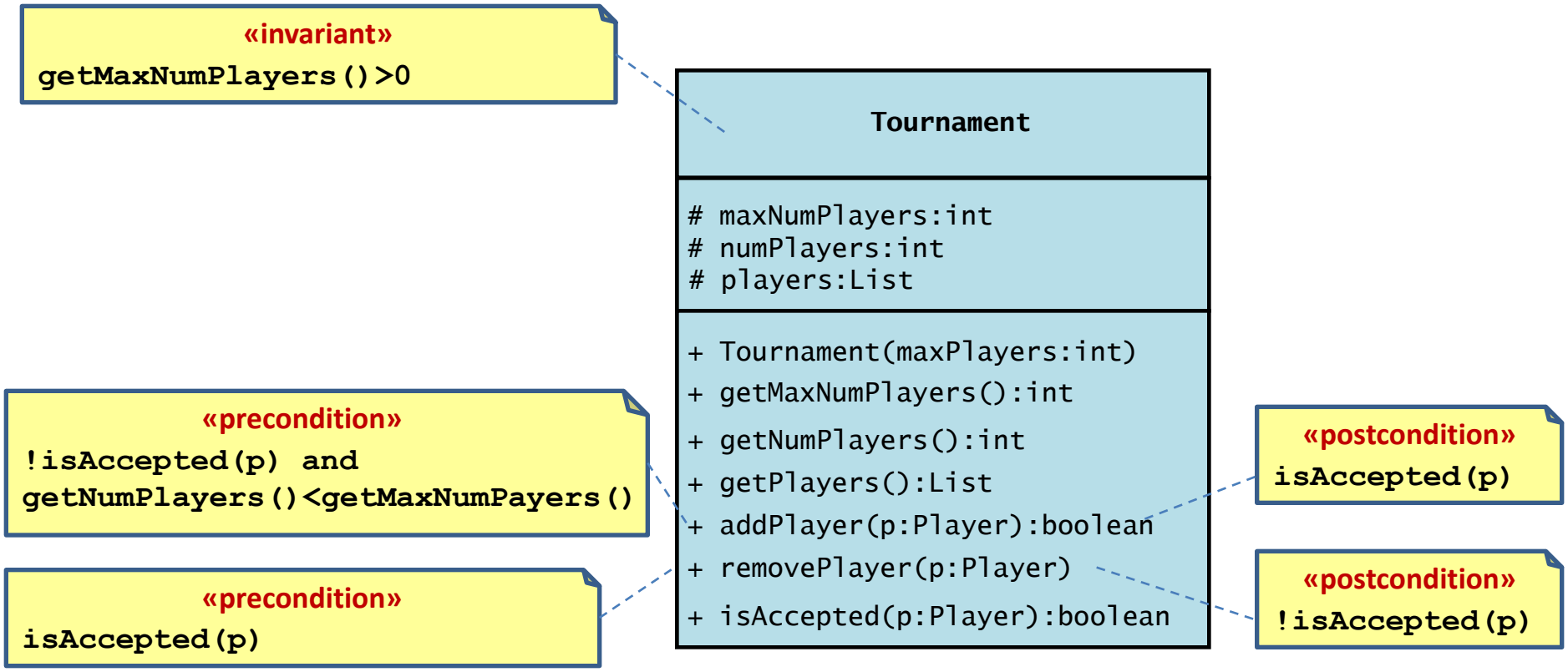
Ako se novi igrač uspješno uključi u turnir, tada se taj igrač nalazi u listi igrača:

t.addPayer(p) $\Rightarrow$ **t. isAccepted(p)**

Specifikacija ograničenja

Reprezentacija ograničenja u UML modelima

Ograničenje može da se prikaže kao *note* (vezan za odnosni UML koncept) koji sadrži logički izraz.



ALTERNATIVA:

UML specifikacija uključuje OCL (*Object Constraint Language*) za specifikaciju ograničenja

OCL

OCL (Object Constraint Language)

- **OCL je formalni jezik za specifikaciju ograničenja na skupu objekata i njihovim atributima**
- Dio standardnog UML2 (od 2006. godine)
- Koristi se za specifikaciju ograničenja koja ne mogu da se specifikuju standardnom UML notacijom
- **Osnovne karakteristike: deklarativni jezik (bez bočnih efekata i kontrole toka)**
- **Osnovni koncepti: OCL izrazi**
  - vraćaju TRUE ili FALSE
  - evaluiraju se u specifikovanom kontekstu (KLASA / OPERACIJA)
  - sva ograničenja primjenjuju se na sve instance

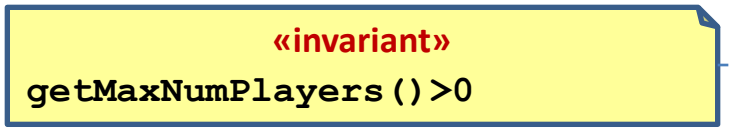
Opšti oblik OCL ograničenja

```
context  <scope>  inv:|pre:|post:  <expr>
```

- `context` — Ključna riječ za specifikaciju OCL ograničenja
- `<scope>` — Klasa ili operacija na koju se odnosi ograničenje
- `inv:|pre:|post:` — **inv:** ≡ «invariant», **pre:** ≡ «precondition», **post:** ≡ «postcondition»
- `<expr>` — OCL izraz

OCL

Primjer OCL specifikacije INVARIJANTE:



```
context Tournament inv:
    self.getMaxNumPlayers()>0
```

self
Ima istu semantiku kao this u OO programskim jezicima.
Ovdje označava sve instance klase Tournament (za svaku instancu).

Interpretacija OCL pravila:
"MAKSIMALAN BROJ IGRAČA NA SVAKOM TURNIRU TREBA DA BUDE POZITIVAN"

OCL

Primjer OCL specifikacije PREDUSLOVA:

Kontekst ograničenja (operacija)
`Tournament::removePlayer(p:Player)`

```
context Tournament::removePlayer(p:Player)
        pre: isAccepted(p)
```

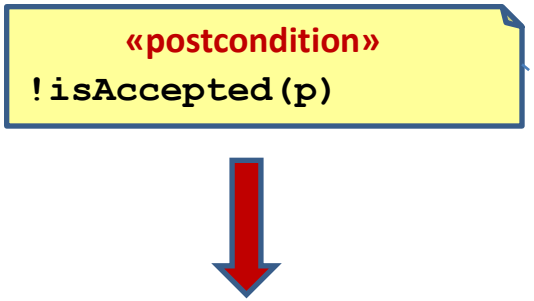
«precondition»
`isAccepted(p)`

Interpretacija OCL pravila:
"OPERACIJA `Tournament::removePlayer(p:Player)`
MOŽE BITI POZVANA SAMO AKO IGRAČ p UČESTVUJE NA TURNIRU"

Alternativno je moglo: `self.isAccepted(p)`
Self ne mora (a može) da se koristi ako nema nedvosmislenosti, kao što je slučaj sa preduslovima i postuslovima (zna se da je kontekst operacija za objekat za koji se poziva).

OCL

Primjer OCL specifikacije POSTUSLOVA:




```
context Tournament::removePlayer(p:Player)
        post: not isAccepted(p)
```

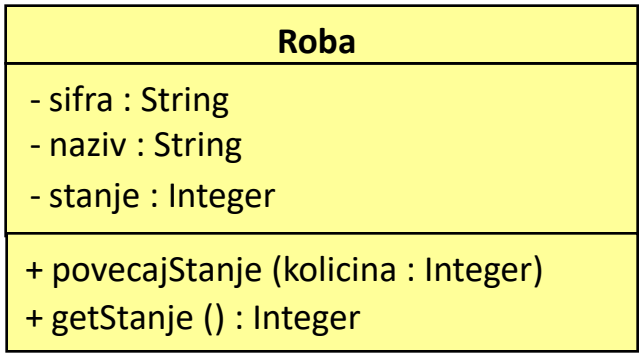
Alternativno pravilo:
Nakon isključivanja jednog igrača, ukupan broj igrača manji je za jedan nego prije isključivanja datog igrača.

```
context Tournament::removePlayer(p:Player)
        post: getNumPlayers() = self.getNumPlayers@pre()-1
```

self@pre označava stanje objekta (Tournament) prije izvršenja operacije Tournamnet::removePlayer()

OCL

Primjer:




```
context Roba::povecajStanje (kolicina:Integer)
pre:    kolicina > 0
post:   stanje = self.stanje@pre + kolicina

context Roba::getStanje ()
post:   result = stanje
```

result reprezentuje rezultat izvršavanja operacije

OCL

Višestruka ograničenja:

- Za isti kontekst mogu da se specifikuju višestruka ograničenja
- Sva ograničenja iste vrste moraju biti zadovoljena
  - npr. operacija ne može da se pozove ako nisu svi preduslovi zadovoljeni

KONTRAKT za
`Tournament::addPlayer(p:Player)`

```
context
  Tournament::addPlayer(p:Player)
pre:
  not isAccepted(p)

context
  Tournament::addPlayer(p:Player)
pre:
  getNumPlayers() < getMaxNumPlayers()
```

```
context
  Tournament::addPlayer(p:Player)
post:
  isAccepted(p)

context
  Tournament::addPlayer(p:Player)
post:
  getNumPlayers() = getNumPlayers@pre()+1
```

```
context
  Tournament::addPlayer(p:Player)
pre:
  not isAccepted(p) and
  getNumPlayers() < getMaxNumPlayers()
```

```
context
  Tournament::addPlayer(p:Player)
post:
  isAccepted(p) and
  getNumPlayers()=getNumPlayers@pre()+1
```

OCL

OCL kontrakt kao JavaDoc anotacije:

- Alati za automatsko generisanje koda na osnovu UML modela imaju mogućnost generisanja JavaDoc anotacija na osnovu specifikovanih OCL kontrakta
- Specifični JavaDoc tagovi: **@invariant**, **@pre** i **@post**
- Anotacije **omogućavaju automatsku validaciju predikata** (može da usporava rad sistema)

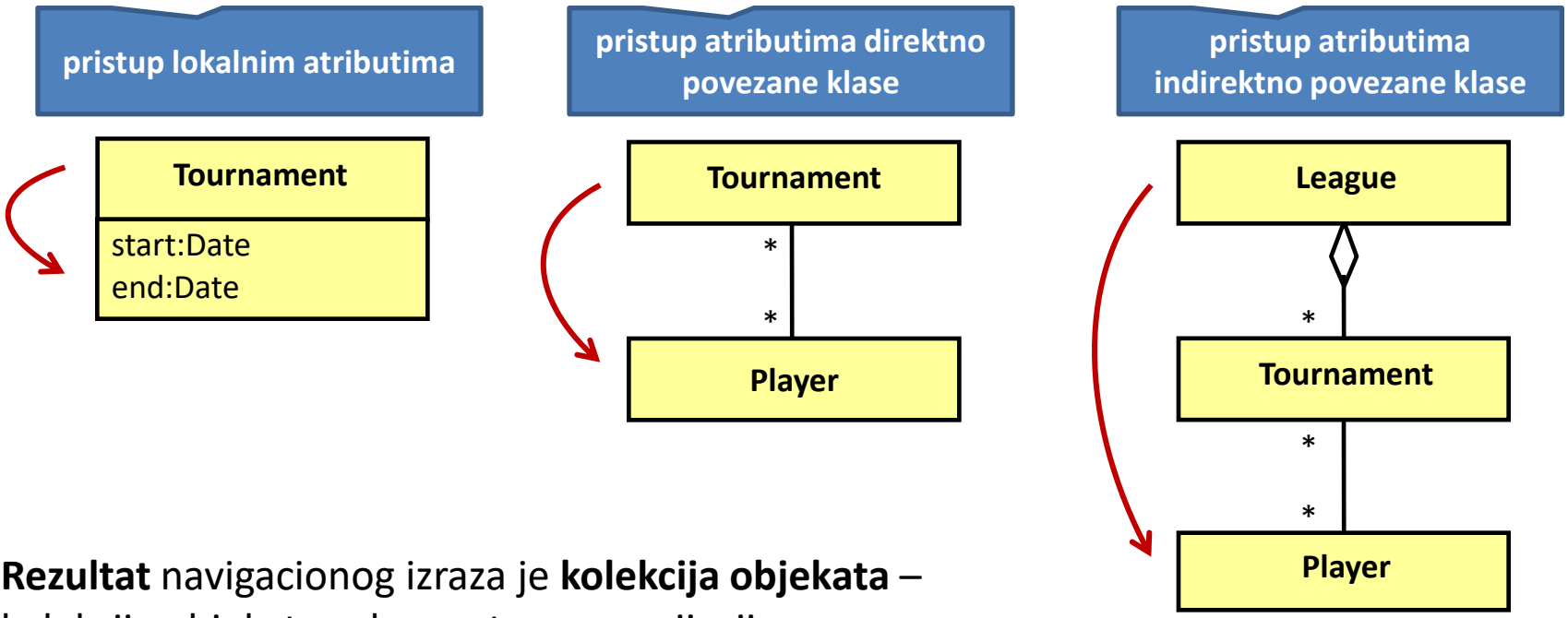
```
context Tournament inv:
  self.maxNumPlayers > 0

context Tournament::addPlayer(p:Player) pre:
  not isAccepted(p)

context Tournament::addPlayer(p:Player) pre:
  getNumPlayers() < getMaxNumPlayers()

context Tournament::addPlayer(p:Player) post:
  isAccepted(p)

context Tournament::addPlayer(p:Player) post:
  getNumPlayers() = getNumPlayers@pre()+1
```

```
/* @invariant maxNumPlayers > 0  */
public classTournament
{
  private int maxNumPlayers;

  private List players;

  public int getNumPlayers() {//...}
  public int getMaxNumPlayers() {//...}

  /**
  * @pre !isAccepted(p)
  * @pre getNumPlayers() < getmaxNumPlayers()
  * @post isAccepted(p)
  * @post getNumPlayers() =
  *         self.getNumPlayers@pre() + 1
  */
  public void addPlayer (Player p) {//...}

  /* ostale metode */
}
```

OCL

OCL ograničenja koja uključuju više klasa:

- Dijagram klasa predstavlja kolekciju povezanih klasa – moguća (i često potrebna) **navigacija** kroz dijagram i referenciranje drugih klasa i njihovih atributa/operacija = **navigacioni OCL izrazi**
- **tri tipa navigacije kroz model:**



- **Rezultat** navigacionog izraza je **kolekcija objekata** – kolekcija objekata s druge strane asocijacije (broj objekata u kolekciji zavisi od multiplikativnosti druge strane asocijacije)

OCL

OCL kolekcije:

- OCL tip `Collection` je generička klasa koja reprezentuje kolekcije objekata tipa T
  - **primitivni**: Integer, String, Boolean, Real, ...
  - **korisnički definisan**: Tournament, Player, ...
- Podržane OCL kolekcije (potklase klase Collection):
  - **Set** – neuređena kolekcija bez duplikata,
  - **Bag** – neuređeni multiset (neuređena kolekcija sa mogućim duplikatima),
  - **Sequence** – uređeni multiset (uređena kolekcija sa mogućim duplikatima).
- Primjeri kolekcija:
  - Set(Integer), Bag(Player), Sequence(Integer)
- Generalizacija svih tipova podataka: `OclAny`
- OCL raspolaže velikim brojem različitih operacija koje mogu da se primjenjuju nad kolekcijama

  `kolekcija->operacija(argumenti)`

OCL operacije nad kolekcijama:

`size() : Integer`
ukupan broj elemenata u kolekciji

`includes(o:OclAny) : Boolean`
TRUE, ako je objekat "o" sadržan u kolekciji

`count(o:OclAny) : Integer`
ukupan broj pojavljivanja objekta "o" u kolekciji

`isEmpty() : Boolean`
TRUE, ako je kolekcija prazna

`notEmpty() : Boolean`
TRUE, ako kolekcija nije prazna

-- operacije koje vraćaju kolekciju --

`union(c:Collection)`
unija sa kolekcijom "c"

`intersection(c:Collection)`
presjek sa kolekcijom "c"

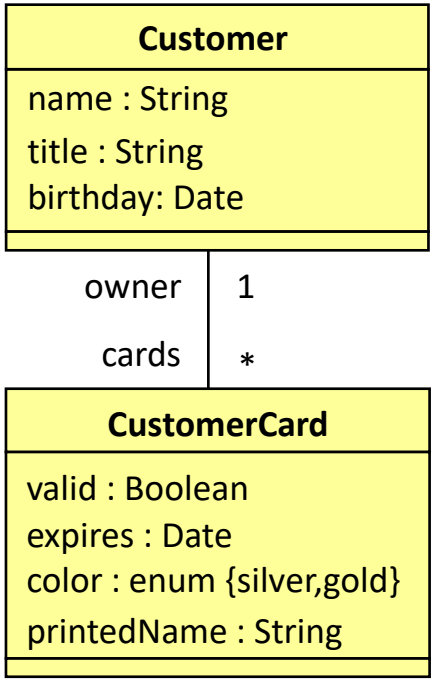
`including(o:OclAny)`
dodaje objekat "o" u kolekciju

`select(expr:OclExpression)`
podskup kolekcije koji zadovoljava izraz "expr"

OCL

Navigacija kroz asocijaciju “1:*”

- Navigacija kroz asocijaciju “1:*” kao rezultat daje **Set**
- Navigacija kroz {ordered} asocijaciju “1:*” kao rezultat daje **Sequence**

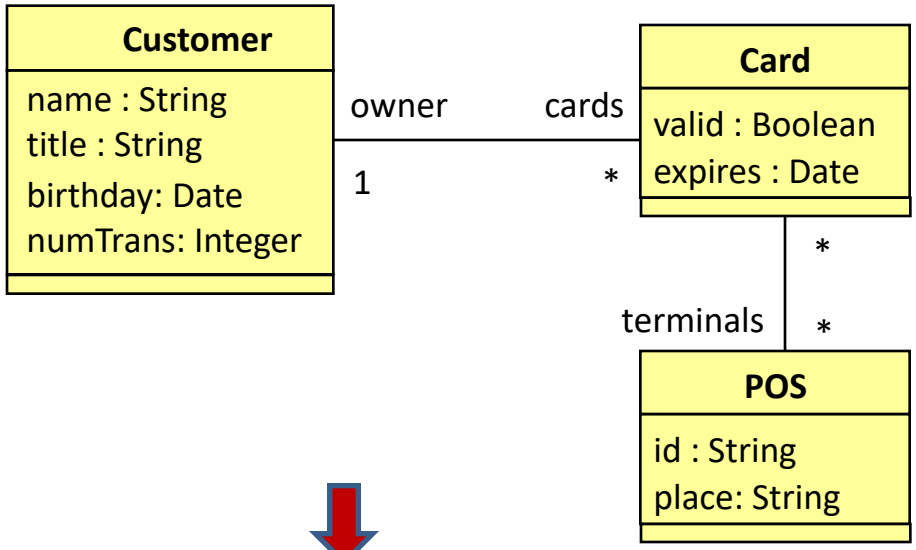



```
context Customer inv:
  self.cards->size() < 4
```

Customer može da ima najviše tri kartice

Navigacija kroz više “1:*” asocijacija

- Navigacija kroz više “1:*” asocijacija kao rezultat daje **Bag (multiset)**

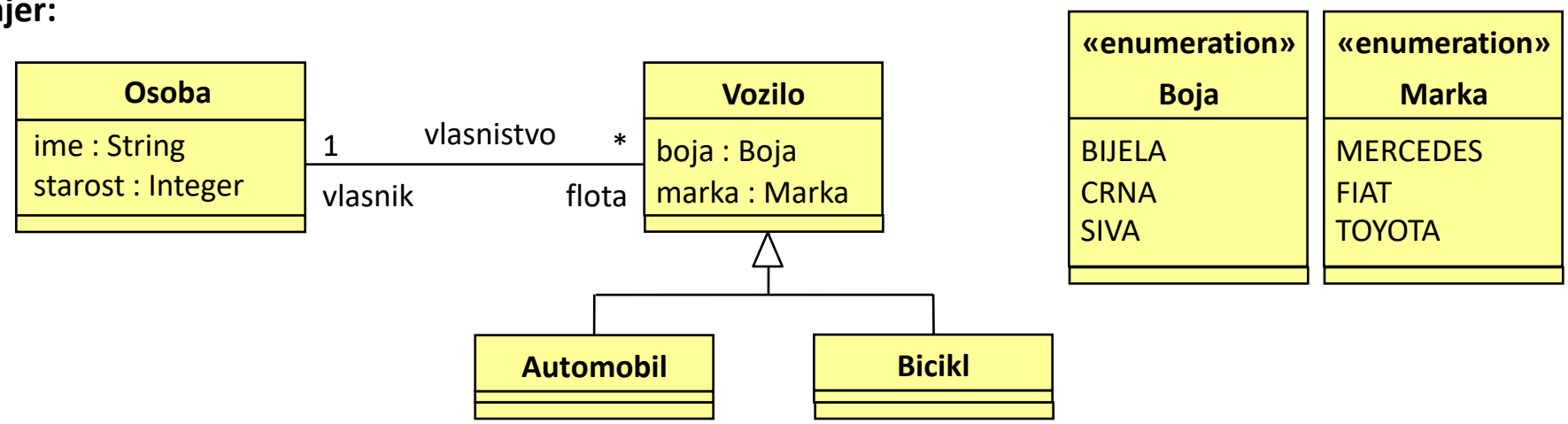



```
context Customer inv:
  self.numTrans = self.cards.terminals->size()
```

Ukupan broj transakcija koje je ostvario customer

OCL

Primjer:



Odaberi odgovarajući upit:

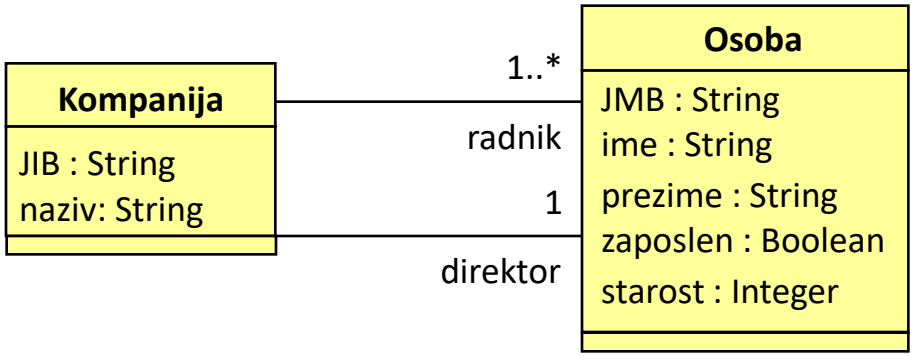
❶
```
context Osoba
inv: self.starost >= 18
```
✘ Sve osobe moraju da imaju bar 18 godina

❷
```
context Vozilo
inv: self.vlasnik.starost >= 18
```
✘ Vlasnik svakog vozila mora da ima bar 18 godina

❸
```
context Automobil
inv: self.vlasnik.starost >= 18
```
✓ Vlasnik svakog automobila mora da ima bar 18 godina

OCL

Primjer:




```
context Kompanija

inv: self.direktor.zaposlen = TRUE

inv: self.radnik->notEmpty()

inv: self.radnik->
     select(r | r.starost>=65)->isEmpty()
```

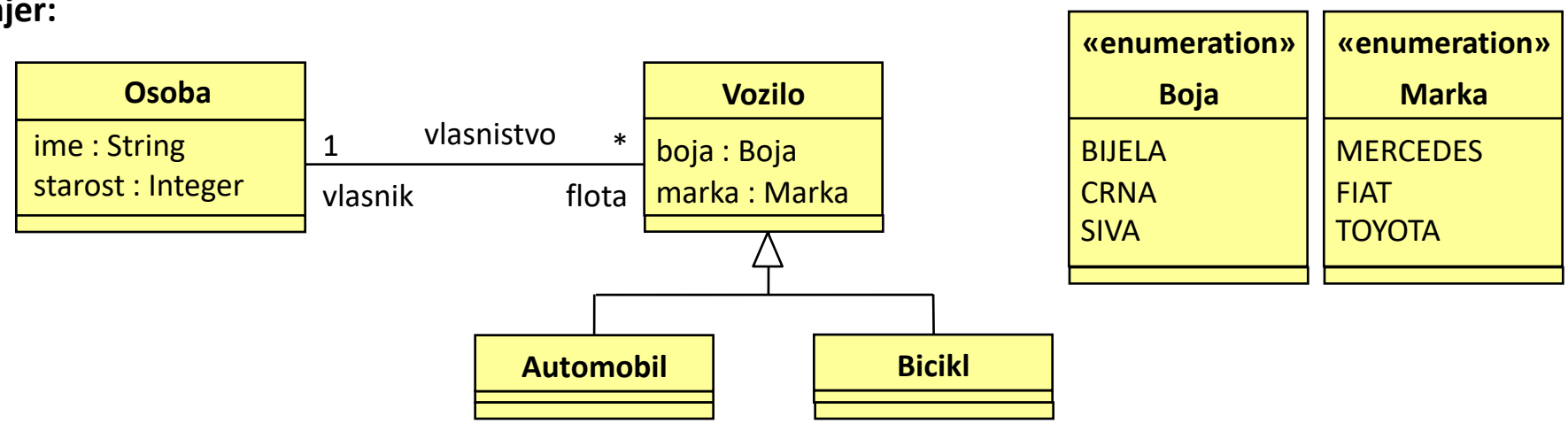
Multiplikativnost asocijacije na strani Osobe je “1”, pa rezultat upita jedan objekat tipa Osoba

Multiplikativnost asocijacije na strani Osobe je “1..*”, pa je rezultat upita skup objekata tipa Osoba

Svi stariji od 65 godina moraju u penziju!

OCL

Primjer:




```
context Osoba
inv: self.flota->size() < 4
```

Svaka osoba može da ima najviše tri vozila

```
context Osoba
inv: self.flota->select (v | v.boja=#CRNA)->size() < 3
```

Svaka osoba može da ima najviše dva crna vozila

```
context Osoba
inv: self.flota->select (v | v.oclIsKindOf(Automobil))->size() < 3
```

Svaka osoba može da ima najviše dva automobila

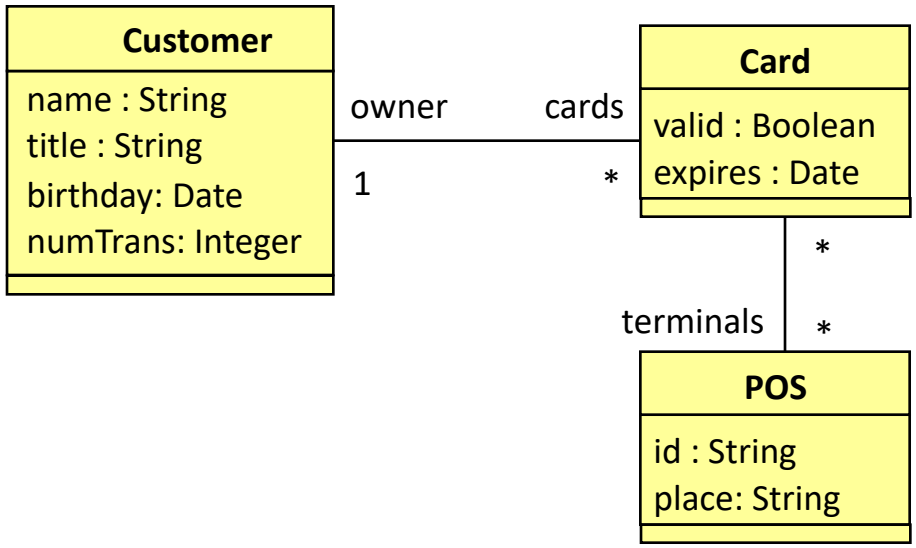
OCL

Konverzije OCL kolekcija

`asSet()`
konvertuje Sequence ili Bag kolekciju u Set

`asSequence()`
konvertuje Set ili Bag kolekciju u Sequence

`asBag()`
Konvertuje Set ili Sequence kolekciju u Bag



Ukupan broj transakcija koje je ostvario customer

```
cards.terminals->size()
```

asSet() izbacuje duplikate, pa u setu imamo samo sve različite terminale na kojima su ostvarene transakcije

```
cards.terminals->asSet()->size()
```

OCL

OCL kvantifikatori

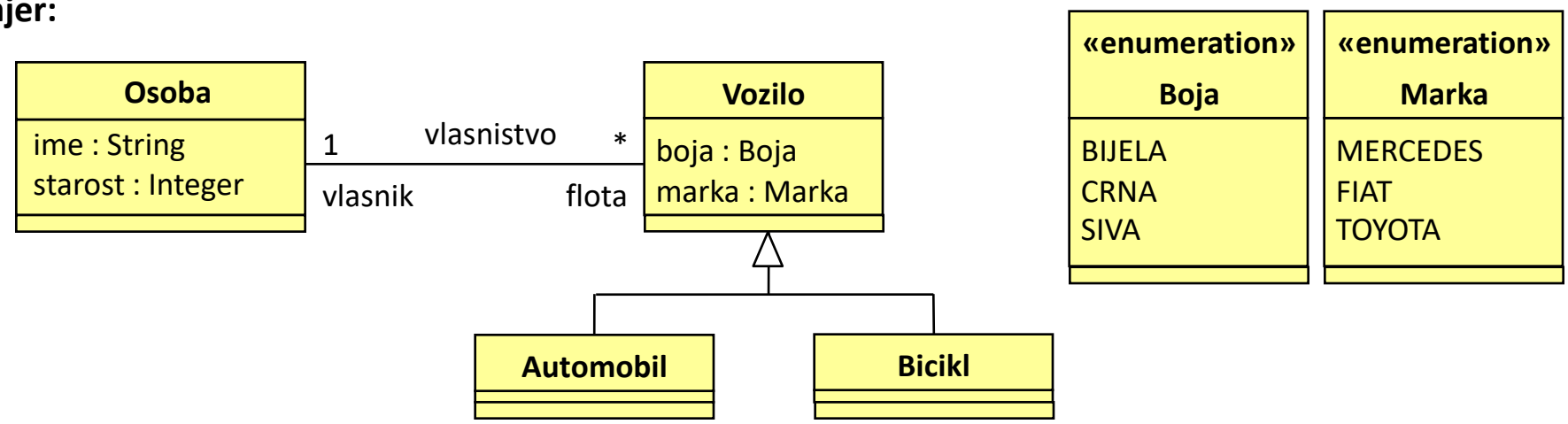
`forAll( <var> | <expr> ) : Boolean`

TRUE, ako je <expr> istinit za sve elemente u kolekciji

`exists( <var> | <expr> ) : Boolean`

TRUE, ako je <expr> istinit bar za jedan element u kolekciji

Primjer:




```
context Osoba
inv: self.flota->forAll(v | v.boja=#CRNA)
```

Osoba mora da ima sva crna vozila

```
context Osoba
inv: self.flota->exists(v | v.boja=#CRNA)
```

Osoba mora da ima bar jedno crno vozilo

OCL

Primjeri:

```
-- nijedan racun ne smije biti u minusu
context Racun inv:
    Racun.allInstances()->forAll( r | r.stanje >= 0 )

-- svi radnici u kompaniji moraju biti mladji od 65 godina
context Kompanija inv:
    self.radnik->forAll( r:Osoba | r.starost < 65 )

-- ne mogu da postoje dva studenta sa istim brojem indeksa
context Student inv:
    Student.allInstances()->
            forAll( s1,s2 | s1<>s2 implies s1.indeks<>s2.indeks )

-- mora da postoji bar jedno vozilo crne boje
context Vozilo inv:
    Vozilo.allInstances()->exists( v | v.boja = #CRNA )

-- za svaki predmet mora da postoji bar jedan zaduzeni nastavnik
context Predmet inv:
    self.nastavnik->exists( z:Zaduzenje | z.predmet = self.naziv )
```

Heuristike za specifikaciju ograničenja

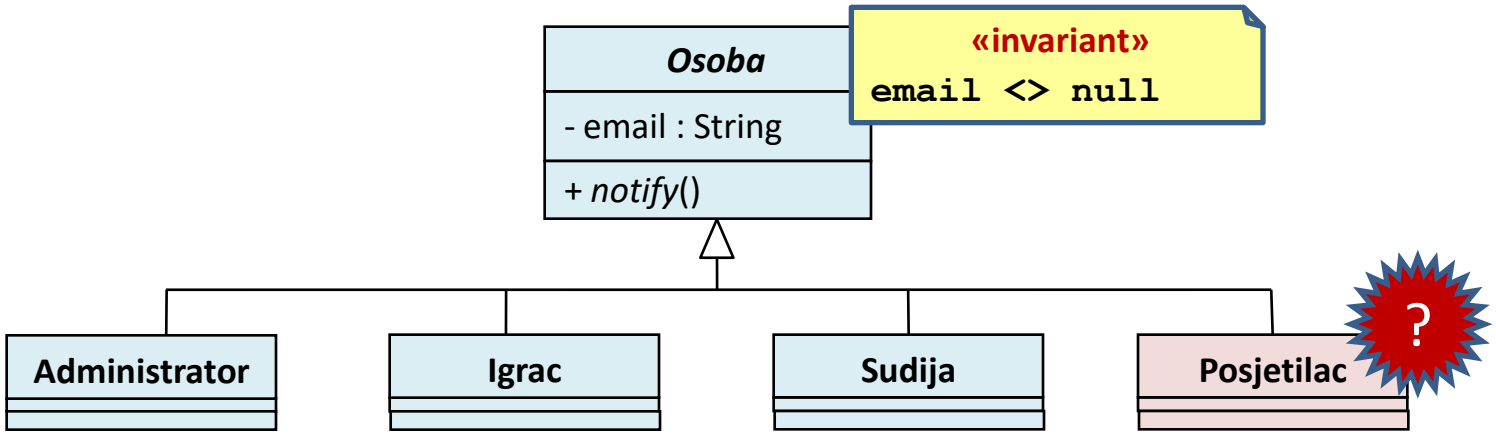
Neke heuristike za specifikaciju ograničenja:

- **fokusirati se na životni ciklus klase**
  - ograničenja koja su specifična za operacije ili neka stanja objekata (ulazak u stanje/izlazak iz stanja) bolje je specifikovati kao preduslove ili postuslove
- **obratiti pažnju na specifične vrijednosti svakog atributa**
  - null vrijednosti, jedinstvene vrijednosti, vrijednosti koje zavise od vrijednosti drugih atributa...
- **identifkovati specifične slučajeve asocijacija**
  - multiplikativnosti krajeva u nekim asocijacijama ne mogu adekvatno da se specifikuju standardnom notacijom, npr. podskupovi, uslovna multiplikativnost, ...
- **definisati pomoćne metode za specifična ograničenja**
  - nekad je bolje uvesti novu metodu (helper) zaduženu za implementaciju specifičnih ograničenja, nego specifikovati komplikovana OCL ograničenja, npr. preklapanje događaja, ...
- **minimizovati broj asocijacija uključenih u neko OCL ograničenje**
  - veći broj uključenih asocijacija smanjuje čitljivost OCL ograničenja, povećava mogućnost greške, povećava broj klasa u multisetovima, ...

Ograničenja u kontekstu nasljeđivanja

***Contract inheritance*:**

- Striktno nasljeđivanje (nasljeđivanje u skladu sa pravilom supstitucije) korisniku hijerarhije klasa daje za pravo da smatra da ograničenja koja važe na superklasi, važe i na potklasama.



Operacija **notify()** služi za slanje elektronske poruke osobi.

Invarijanta za klasu **Osoba** specifikuje da **svaka osoba ima e-adresu**.

Striktno nasljeđivanje daje korisniku za pravo da smatra da invarijanta važi i za potklase, tj. da i administrator i igrači i sudije imaju elektronske adrese te da im je moguće poslati elektronsku poruku.

Posjetioci su specifičan slučaj: najvjerovatnije nemamo e-adrese svih posjetilaca!

RJEŠENJE: Ili klasu Posjetilac isključiti iz hijerarhije ili promijeniti ograničenje!

Ograničenja u kontekstu nasljeđivanja

Pravila za nasljeđivanje kontrakta

- **invarijante**:
  - Sve invarijante superklase moraju da važe i za potklase.
  - Za svaku potklasu **mogu da se specifikuju i dodatne invarijante**.
  - Npr. **Niz** ⇦ **SortiraniNiz** (sve što važi za niz, važi i za sortirani niz, uključujući i uređeni poredak).
- **preduslovi**: (*constraint weakening*)
  - Metoda u potklasi **ima pravo da "oslabi" preduslov** za metodu koju redefiniše, tj. preduslovi za metodu u potklasi su "manje strogi" nego za metodu u natklasi.
  - Npr. redefinisana metoda u potklasi može da obrađuje više različitih slučajeva u odnosu na odnosnu metodu iz superklase, pa može da oslabi preduslove iz natklase.
- **postuslovi**: (*constraint strenghtening*)
  - Metoda u potklasi **mora da obezbijedi iste postuslove** kao metoda u superklasi, a **može i da ih "pooštri"**, tj. postuslovi za redefinisanu metodu u potklasi mogu da budu "više strogi" nego u natklasi.