

Laslo Kraus
Igor Tartalja

**ZBIRKA ZADATAKA
IZ
PROJEKTOVANJA
SOFTVERA**

AKADEMSKA MISAO
Beograd, 2009.

Laslo Kraus, Igor Tortalja

ZBIRKA ZADATAKA
IZ PROJEKTOVANJA SOFTVERA
Drugo, dopunjeno izdanje

Recenzenti
Dr Milo Tomašević
Dr Vladimir Blagojević

Izдавач
AKADEMSKA MISAO
Beograd

Lektor
Andelka Kovačević

Dizajn naslovne strane
Zorica Marković, akademski slikar

Štampa
Planeta print, Beograd

Tiraž
300 primeraka

ISBN 978-86-7466-368-4

NAPOMENA: Fotokopiranje ili umnožavanje na bilo koji način ili ponovno objavljivanje ove knjige - u celini ili u delovima - nije dozvoljeno bez prethodne izričite saglasnosti i pismenog odobrenja izdavača.

Predgovor

Ova zbirka sadrži sve zadatke koji su u toku školske 2008/09. godine rađeni na vežbama iz predmeta **Projektovanje softvera** na četvrtoj godini *Odseka za računarsku tehniku i informatiku*, odnosno na trećoj godini *Odseka za softversko inženjerstvo Elektrotehničkog fakulteta Univerziteta u Beogradu*. Pored toga, uključeni su zadaci i pitanja sa većine kolokvijuma i ispita. Iako je prvenstveno namenjena studentima *Elektrotehničkog fakulteta* koji prate predmet **Projektovanje softvera**, zbirka može biti od koristi i drugima, koji uče da projektuju softver modelovanjem na jeziku UML, uz primenu projektnih uzoraka.

U odnosu na prvo izdanje zbirke dodati su zadaci sa kolokvijuma *Odseka za računarsku tehniku i informatiku* {K}, prvog i drugog kolokvijuma *Odseka za softversko inženjerstvo* {K1, K2} i ispita oba odseka {I}. Dva ispitna zadatka su nastala na osnovu ideje kolege Žarka Stanisavljevića {Ž.S.}.

Prvi deo zbirke sadrži rešene zadatke, organizovane na način da prate auditorne vežbe iz predmeta, i većina njih se radi na vežbama u toku semestra. Svrha ovog dela zbirke je da studentima olakša praćenje nastave. Zadaci su priloženi u zatečenom obliku u kojem se koriste u toku izvođenja vežbi. To znači da je uz pojedina rešenja priloženo vrlo malo dopunske objašnjenje i to prvenstveno u obliku slika ili formula. Zbog toga se studentima izričito preporučuje redovno pohađanje vežbi u toku nastave, gde će dobiti usmena objašnjenja za lakše razumevanje pojedinih zadataka.

U drugom delu zbirke nalaze se ispitni zadaci bez rešenja. Namjenjeni su za samostalno rešavanje u toku priprema za ispite.

U trećem delu zbirke nalaze se pitanja za proveru znanja, sa ispit i kolokvijuma u periodu od 2006/07. do 2008/09. školske godine. Pitanja su uređena tematski, ali se napominje da ona ne čine skupove pitanja koji potpuno pokrivaju odgovarajuće teme. Odgovori na pitanja koji se očekuju od studenata na kolokvijumima i ispitima treba da budu jezgrovit i precizni. Primer odgovora na nekoliko pitanja, priložen je na kraju ovog dela.

Iako su problemi koncipirani i formulisana na način da rešenja ne zavise od konkretnog alata za projektovanje softvera, za njihovo rešavanje korišćen je alat za modelovanje *StarUML 5.0.2*, koji može da se preuzme sa adrese <http://staruml.sourceforge.net/> i besplatno koristi. Skreće se pažnja na to da *StarUML* nije potpuno kompatibilan sa jezikom *UML* verzije 2, kao i da ima programskih grešaka (naročito u generisanju koda), ali je u kategoriji besplatnih alata jedan od najudobnijih za korišćenje i najbolje pokriva standard *UML2*.

Svoja zapažanja čitaoci mogu da upute autorima elektronskom poštom na adrese kraus@etf.rs i tortalja@etf.rs.

Beograd, septembar 2009.

Laslo Kraus
i Igor Tortalja

Sadržaj

Predgovor	3
Sadržaj	4
Preporučena literatura	6
Zadaci.....	7
Zadatak 1 Skladište predmeta (osnove dijagrama <i>klasa</i>)	8
Zadatak 2 Tačka i figure u ravni (nasleđivanje i apstraktne klase).....	12
Zadatak 3 Uređivači uporedivih stvari (interfejsi).....	16
Zadatak 4 Proizvod, skladište i mehanizmi (dijagram <i>paketa</i> ; aktivna klasa)	20
Zadatak 5 Samoposluga (generisanje modela – <i>Java</i>)	24
Zadatak 6 Proizvodi, maštine i radnik (generisanje modela – <i>C++</i>)	28
Zadatak 7 Pravila, student, studentski odsek (projektni uzorak <i>Unikat</i>).....	30
Zadatak 8 Predmeti i radnici (projektni uzorci <i>Prototip</i> i <i>Sastav</i>).....	32
Zadatak 9 Predmeti i radnici (dijagrami <i>objekata</i> i <i>interakcije</i>).....	34
Zadatak 10 Otpornici (<i>KI, 10.11.2006.</i>)	36
Zadatak 11 Proizvod, skladište, mehanizmi i motor	38
Zadatak 12 Vektor, predmeti i orijentisani predmeti (<i>KI, 01.11.2007.</i>)	40
Zadatak 13 Simbol, font, vektor, figure, platno i aplikacija (<i>KI, 30.10.2008.</i>)	44
Zadatak 14 Samoposluga	48
Zadatak 15 Distributeri, klijenti, pošiljke i raspodele (projektni uzorak <i>Posmatrač</i>)	54
Zadatak 16 Boja, tačka, figure i iteratori (projektni uzorak <i>Iterator</i>)	62
Zadatak 17 Grafički statistički pokazatelji (projektni uzorak <i>Dopuna</i>)	72
Zadatak 18 Prodavnica, fakultet i pošta (dijagram <i>slučajeva korišćenja</i>)	76
Zadatak 19 Projektni uzorci <i>Strategija</i> i <i>Šablonska metoda</i>	78
Zadatak 20 Atomi, znak, piksel, tekst, slika i dokument (<i>K, 01.12.2006.</i>)	82
Zadatak 21 Artikli, zbirke, obilasci, osobe, načini plaćanja, samoposluga (<i>K, 16.11.2008.</i>).....	86
Zadatak 22 Polje, tabla, figure, igrači i igre (<i>K, 30.11.2007.</i>)	90
Zadatak 23 Samoposluga (dijagrami <i>aktivnosti</i> i <i>stanja</i>)	94
Zadatak 24 Predmeti i akteri	96
Zadatak 25 Predmet, skladište, pokazivač, nadzornici (<i>K2, 15.12.2006.</i>)	102
Zadatak 26 Roba, artikal, porez, popust, skladište, redosledi i izveštaj (<i>K2, 06.12.2007.</i>)	104
Zadatak 27 Vozila, mesto, parking, redosledi i semafor (<i>K2, 04.12.2008.</i>)	108
Zadatak 28 Dijagram stanja i aktivnosti niti	112
Zadatak 29 Projektni uzorci u AWT-u	113
Zadatak 30 Gradska saobraćaj (projektni uzorak <i>Stanje</i>)	114
Zadatak 31 Figure u ravni (projektni uzorak <i>Muva</i>)	120
Zadatak 32 Predmeti, maštine i radnik (projektni uzorak <i>Proizvodna metoda</i>)	126
Zadatak 33 Igre (projektni uzorci <i>Podsetnik</i> i <i>Komanda</i>)	130
Zadatak 34 Biblioteka (dijagram <i>komponenata</i> , projektni uzorci <i>Most</i> i <i>Apstraktna fabrika</i>).....	136

Zadatak 35 Uređaj, operacije i API (projektni uzorak <i>Fasada</i>) {I, 14.02.2007.}	144
Zadatak 36 Fabrika, radnici, skladište i automobili (projektni uzorak <i>Graditeљ</i>) {I, 06.02.2009.}	152
Zadatak 37 Tačka, boja, elementi, crtači, komponente, akteri i program {I, 05.02.2007.}	158
Zadatak 38 Vozila, saobraćajnice i semafor {I, 23.01.2008.}	164
Zadatak 39 Tačka, boja, duži, telo, scena, crtači, preslikavanje, dugme, radnja i program {I, 28.01.2009.}	170

Ispitni zadaci

Zadatak I Izdavaštvo {I, 21.02.2007.}	176
Zadatak II Videoteka {I, 28.03.2007.}	177
Zadatak III Medicinski dokumenti {I, 04.07.2007.}	178
Zadatak IV Testovi {I, 05.09.2007.}	179
Zadatak V Šalterska služba {I, 29.02.2008.}	180
Zadatak VI Softverska kompanija {I, 29.02.2008. – Ž.S.}	181
Zadatak VII Upravljanje nitima i procesima {I, 03.09.2008.}	182
Zadatak VIII Igre na tabli, šah {I, 17.09.2008.}	183
Zadatak IX Prodaja računara {I, 17.09.2008. – Ž.S.}	184
Zadatak X Čoveče ne ljuti se {I, 15.06.2009.}	185

Ispitna pitanja

i UML – uvod	188
ii UML – dijagrami klasa	188
iii UML – dijagrami paketa	188
iv UML – dijagrami interakcije	188
v UML – dijagrami slučajeva korišćenja	189
vi UML – dijagrami aktivnosti	189
vii UML – dijagrami stanja	189
viii UML – dijagrami klasa, napredniji pojmovi	189
ix UML – dijagrami složene strukture	190
x UML – dijagrami komponenata	190
xi UML – dijagrami raspoređivanja	190
xii Projektni uzorci – Uvod	190
xiii Projektni uzorci stvaranja	190
xiv Projektni uzorci strukture	190
xv Projektni uzorci ponašanja	191
xvi Primeri odgovora na pitanja	192

Preporučena literatura

- [1] Grady Booch, James Rumbaugh, Ivar Jacobson: **The Unified Modeling Language User Guide**, 2nd Edition, Addison Wesley, 2005. (Prevod na srpski jezik: Booch,G., Rumbaugh,J., Jacobson,I., UML Vodič za korisnike, *prevod prvog izdanja*, CET, Beograd, 2001. – zastareo; obrađuje UML 1)
- [2] James Rumbaugh, Ivar Jacobson, Grady Booch, **Unified Modeling Language Reference Manual**, 2nd Edition, Addison-Wesley, 2004.
- [3] **Unified Modeling Language: Infrastructure i Unified Modeling Language: Superstructure – Object Management Group**, 2009., <http://www.omg.org/spec/UML/2.2/>
- [4] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: **Design Patterns: Elements of Reusable Object-Oriented Software**, Addison-Wesley Professional Computing Series, Addison-Wesley Publishing, Inc., Reading, Massachusetts, 1995. (Prevod na srpski jezik: **Gotova rešenja: Elementi objektno orijentisanog softvera**, CET, Beograd, 2002.)
- [5] Alan Shalloway, James R. Trott: **Design Patterns Explained: A New Perspective on Object-Oriented Design**, 1st Edition, Addison Wesley Professional, 2002. (Prevod na srpski jezik: **Projektni obrasci – Nove tehnike objektno orijentisanog projektovanja**, Mikro knjiga, Beograd, 2004.)
- [6] Laslo Kraus: **Rešeni zadaci iz programskog jezika Java**, drugo izdanje, Akademska misao, Beograd, 2007.
- [7] Laslo Kraus: **Rešeni zadaci iz programskog jezika C++**, drugo izdanje, Akademska misao, Beograd, 2006.

Zadaci

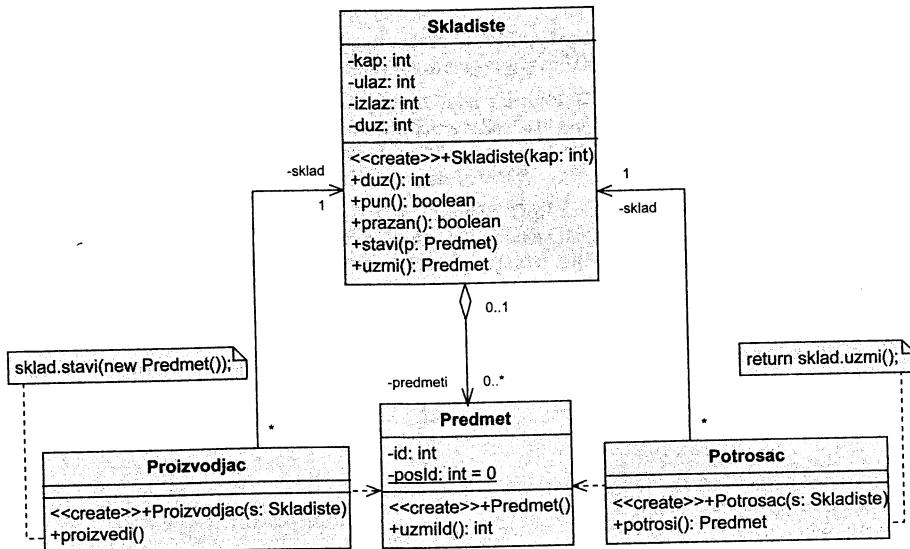
Zadatak 1 Skladište predmeta (osnove dijagrama klasa)

Nacrtati dijagram klasa na jeziku UML sledećeg sistema klasa:

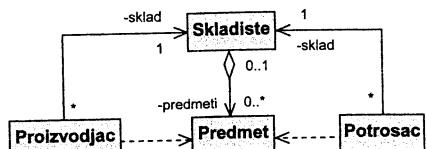
- **Predmet** ima jedinstven, automatski generisan identifikacioni broj koji može da se dohvati.
- **Skladište** može da sadrži zadat broj predmeta. Stvara se prazno, posle čega predmeti mogu da se stavljaju i vade jedan po jedan. Predmeti se vade po redosledu stavljanja. Može da se dohvati broj predmeta u skladištu i da se ispita da li je skladište puno i da li je prazno.
- **Proizvođač** može da napravi jedan predmet i da ga stavi u skladište koje se zadaje prilikom stvaranja proizvođača.
- **Potrošač** može da uzme jedan predmet iz skladišta koje se zadaje prilikom stvaranja potrošača.

Rešenje:

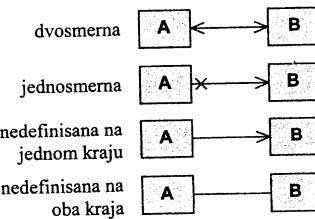
a) Detaljni dijagram



b) Sažeti dijagram



c) Navigabilnost po UML2 (StarUML ne podržava)



d) Kôd koji je na jeziku Java generisao StarUML

```

// Generated by StarUML(tm) Java Add-In
//
// @ Project : PS
// @ File Name : Predmet.java
// @ Date : 16.8.2007
// @ Author :

public class Predmet {
    private int id;
    private static int posId = 0;
    public void Predmet() {
    }

    public int uzmiId() {
    }
}
  
```

```

// @ File Name : Skladiste.java
public class Skladiste {
    private int kap;
    private int ulaz;
    private int izlaz;
    private int duz;
    private Predmet predmeti;
    public void Skladiste(int kap) { }
    public int duz() { }
    public boolean pun() { }
    public boolean prazan() { }
    public void stavi(Predmet p) { }
    public Predmet uzmi() { }
}
  
```

```

// @ File Name : Proizvodjac.java
public class Proizvodjac {
    private Skladiste sklad;
    public void Proizvodjac(Skladiste s) { }
    public void proizvedi() { }
}
  
```

```

// @ File Name : Potrosac.java
public class Potrosac {
    private Skladiste sklad;
    public void Potrosac(Skladiste s) { }
    public Predmet potrosi() { }
}
  
```

e) Kôd koji je na jeziku C# generisao StarUML

```

// Generated by StarUML(tm) C# Add-In
// @ Project : PS
// @ File Name : Predmet.cs
// @ Date : 16.8.2007
// @ Author :

public class Predmet {
    private int id;
    private static int posId = 0;
    public Predmet(){}
    public int uzmiId(){
    }
}

// @ File Name : Skladiste.cs
public class Skladiste {
    private int kap;
    private int ulaz;
    private int izlaz;
    private int duz;
    private Predmet predmeti;
    public Skladiste(int kap){}
    public int duz(){}
    public boolean pun(){}
    public boolean prazan(){}
    public void stavi(Predmet p){}
    public Predmet uzmi(){}
}

```

```

// @ File Name : Proizvodjac.cs
public class Proizvodjac {
    private Skladiste sklad;
    public Proizvodjac(Skladiste s){}
    public void proizvedi(){}
}

```

```

// @ File Name : Potrosac.cs
public class Potrosac {
    private Skladiste sklad;
    public Potrosac(Skladiste s){}
    public Predmet potrosi(){}
}

```

f) Kôd koji je na jeziku C++ generisao StarUML

```

// Generated by StarUML(tm) C++ Add-In
// @ Project : PS
// @ File Name : Predmet.h
// @ Date : 16.8.2007
// @ Author :

#ifndef _PREDMET_H
#define _PREDMET_H

class Predmet {
public:
    Predmet();
    int uzmiId();
private:
    int id;
    static int posId = 0;
};

#endif // _PREDMET_H

```

```

// @ File Name : Proizvodjac.h
#ifndef _PROIZVODJAC_H
#define _PROIZVODJAC_H
#include "Skladiste.h"
class Proizvodjac {
public:
    Proizvodjac(Skladiste s);
    void proizvedi();
private:
    Skladiste *sklad;
};

#endif // _PROIZVODJAC_H

```

```

// @ File Name : Potrosac.h
#ifndef _POTROSAC_H
#define _POTROSAC_H
#include "Skladiste.h"
#include "Predmet.h"
class Potrosac {
public:
    Potrosac(Skladiste s);
    Predmet potrosi();
private:
    Skladiste *sklad;
};

#endif // _POTROSAC_H

```

```

// @ File Name : Predmet.cpp
#include "Predmet.h"
Predmet::Predmet() {}
int Predmet::uzmiId() {}


```

```

// @ File Name : Skladiste.h
#ifndef _SKLADISTE_H
#define _SKLADISTE_H
#include "Predmet.h"
class Skladiste {
public:
    Skladiste(int kap);
    int duz();
    boolean pun();
    boolean prazan();
    void stavi(Predmet p);
    Predmet uzmi();
private:
    int kap;
    int ulaz;
    int izlaz;
    int duz;
    Predmet *predmeti;
};

#endif // _SKLADISTE_H

```

```

// @ File Name : Skladiste.cpp
#include "Skladiste.h"
#include "Predmet.h"
Skladiste::Skladiste(int kap) {}
int Skladiste::duz() {}
boolean Skladiste::pun() {}
boolean Skladiste::prazan() {}
void Skladiste::stavi(Predmet p) {}
Predmet Skladiste::uzmi() {}


```

```

// @ File Name : Proizvodjac.cpp
#include "Proizvodjac.h"
#include "Skladiste.h"
Proizvodjac::Proizvodjac(Skladiste s){}
void Proizvodjac::proizvedi() {}


```

```

// @ File Name : Potrosac.cpp
#include "Potrosac.h"
#include "Skladiste.h"
#include "Predmet.h"
Potrosac::Potrosac(Skladiste s) {}
Predmet Potrosac::potrosi() {}


```

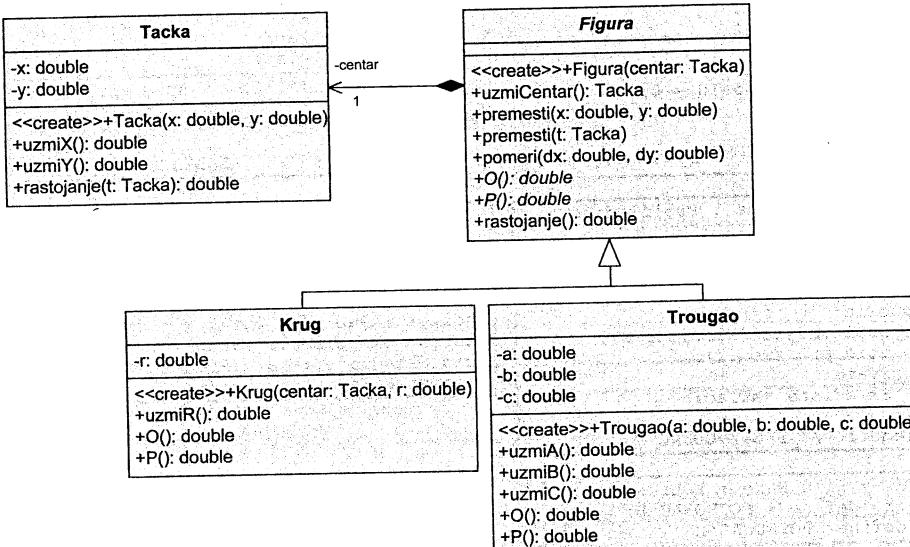
Zadatak 2 Tačka i figure u ravni (nasleđivanje i apstraktne klase)

Nacrtati dijagram klasa na jeziku UML sledećeg sistema klasa:

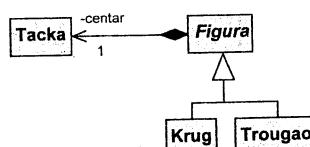
- **Tacka** u ravni se zadaje realnim koordinatama koje mogu da se dohvate. Može da se izračuna rastojanje do zadate tačke.
- Apstraktna **figura** u ravni se zadaje tačkom koja predstavlja centar figure i koja može da se dohvati. Figura može da se premesti na novo mesto i da se pomeri za zadati pomak duž koordinatnih osa. Može da se izračuna obim i površina figure i da se odredi rastojanje od centra figure do centra zadata figure.
- **Krug** u ravni je figura zadata poluprečnikom koji može da se dohvati.
- **Trougao** u ravni je figura zadata dužinama stranica koje mogu da se dohvate.

Rešenje:

a) Detaljni dijagram



b) Sažeti dijagram



c) Kôd koji je na jeziku Java generisao StarUML

```

// @ File Name : Tacka.java
public class Tacka {
    private double x;
    private double y;
    public void Tacka(double x, double y) { }
    public double uzmiX() { }
    public double uzmiY() { }
    public double rastojanje(Tacka t) { }
}
  
```

```

// @ File Name : Figura.java
public abstract class Figura {
    private Tacka centar;
    public void Figura(Tacka centar) { }
    public Tacka uzmiCentar() { }
    public void premesti(double x, double y) { }
    public void premesti(Tacka t) { }
    public void pomeri(double dx, double dy) { }
    public abstract double O(); 
    public abstract double P(); 
    public double rastojanje() { }
}
  
```

```

// @ File Name : Krug.java
public class Krug extends Figura {
    private double r;
    public void Krug(Tacka centar, double r) { }
    public double uzmiR() { }
    public double O() { }
    public double P() { }
    public double O() { }
    public double P() { }
}
  
```

```

// @ File Name : Trougao.java
public class Trougao extends Figura {
    private double a;
    private double b;
    private double c;
    public void Trougao(double a, double b, double c) { }
    public double uzmiA() { }
    public double uzmiB() { }
    public double uzmiC() { }
    public double O() { }
    public double P() { }
    public double O() { }
    public double P() { }
}
  
```

d) Kôd koji je na jeziku C# generisao StarUML

```
// @ File Name : Tacka.cs
public class Tacka {
    private double x;
    private double y;
    public Tacka(double x, double y){ }
    public double uzmiX(){ }
    public double uzmiY(){ }
    public double rastojanje(Tacka t){ }
}
```

```
// @ File Name : Figura.cs
public abstract class Figura {
    private Tacka centar;
    public Figura(Tacka centar){ }
    public Tacka uzmiCentar(){ }
    public void premesti(double x, double y){ }
    public void premesti(Tacka t){ }
    public void pomeri(double dx, double dy){ }
    public abstract double O();
    public abstract double P();
    public double rastojanje(){ }
}
```

```
// @ File Name : Krug.cs
public class Krug : Figura{
    private double r;
    public Krug(Tacka centar, double r){ }
    public double uzmiR(){ }
    public double O(){ }
    public double P(){ }
}
```

```
// @ File Name : Trougao.cs
public class Trougao : Figura{
    private double a;
    private double b;
    private double c;
    public Trougao(double a, double b, double c){ }
    public double uzmiA(){ }
    public double uzmiB(){ }
    public double uzmiC(){ }
    public double O(){ }
    public double P(){ }
}
```

e) Kôd koji je na jeziku C++ generisao StarUML

```
// @ File Name : Tacka.h
#ifndef _TACKA_H
#define _TACKA_H
class Tacka {
public:
    Tacka(double x, double y);
    double uzmiX();
    double uzmiY();
    double rastojanje(Tacka t);
private:
    double x;
    double y;
};
```

```
#endif // _TACKA_H
```

```
// @ File Name : Figura.h
#ifndef _FIGURA_H
#define _FIGURA_H
#include "Tacka.h"
class Figura {
public:
    Figura(Tacka centar);
    Tacka uzmiCentar();
    void premesti(double x, double y);
    void premesti(Tacka t);
    void pomeri(double dx, double dy);
    virtual double O() = 0;
    virtual double P() = 0;
    double rastojanje();
private:
    Tacka centar;
};
```

```
#endif // _FIGURA_H
```

```
// @ File Name : Krug.h
#ifndef _KRUG_H
#define _KRUG_H
#include "Figura.h"
#include "Tacka.h"

class Krug : public Figura {
public:
    Krug(Tacka centar, double r);
    double uzmiR();
    double O();
    double P();
    double O();
    double P();
private:
    double r;
};
```

```
#endif // _KRUG_H
```

```
// @ File Name : Tacka.cpp
#include "Tacka.h"
Tacka::Tacka(double x, double y) { }
double Tacka::uzmiX() { }
double Tacka::uzmiY() { }
double Tacka::rastojanje(Tacka t) { }
```

```
// @ File Name : Figura.cpp
#include "Figura.h"
#include "Tacka.h"
Figura::Figura(Tacka centar) { }
Tacka Figura::uzmiCentar() { }
void Figura::premesti(double x, double y) { }
void Figura::premesti(Tacka t) { }
void Figura::pomeri(double dx, double dy) { }
double Figura::rastojanje() { }
```

```
// @ File Name : Krug.cpp
#include "Krug.h"
#include "Tacka.h"
Krug::Krug(Tacka centar, double r) { }
double Krug::uzmiR() { }
double Krug::O() { }
double Krug::P() { }
double Krug::O() { }
double Krug::P() { }
```

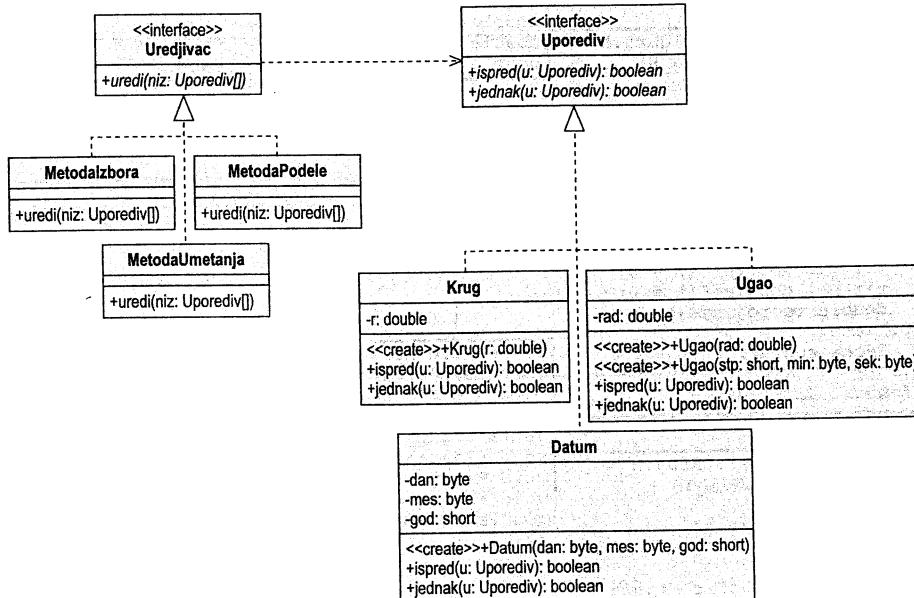
Zadatak 3 Uređivač uporedivih stvari (interfejsi)

Nacrtati dijagram klasa na jeziku UML sledećeg sistema tipova:

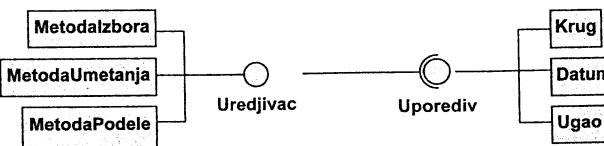
- Za apstraktnu **uporedivu** stvar može da se ispita da li se nalazi ispred i da li je jednaka drugoj uporedivoj stvari.
- Krug, datum i ugao** su uporedive stvari.
- Apstraktan **uređivač** može da uređi niz uporedivih stvari.
- Metoda izbora, metoda umetanja i metoda podele** su uređivači.

Rešenje:

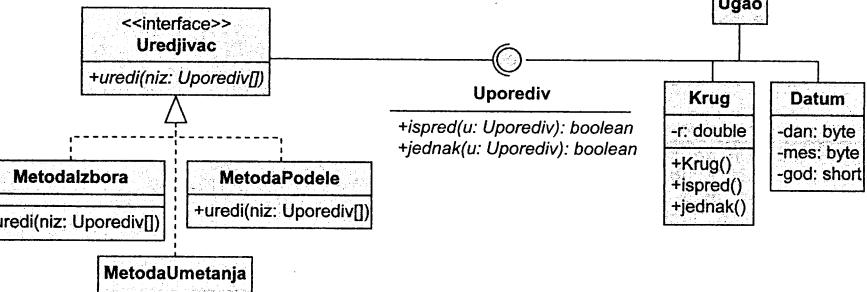
a) Detaljni dijagram



b) Sažeti dijagram



c) Mešoviti dijagram



d) Kôd koji je na jeziku Java generisao StarUML

```
// @ File Name : Uporediv.java
public interface Uporediv {
    public abstract boolean ispred(Uporediv u);
    public abstract boolean jednak(Uporediv u);
}
```

```
// @ File Name : Krug.java
public class Krug implements Uporediv {
    private double r;
    public void Krug(double r) { }
    public boolean ispred(Uporediv u) { }
    public boolean jednak(Uporediv u) { }
    public boolean ispred(Uporediv u);
    public boolean jednak(Uporediv u);
```

```
// @ File Name : Uredjivac.java
public interface Uredjivac {
    public abstract void uredi(Uporediv[] niz);
}
```

```
// @ File Name : Metodalzbora.java
public class Metodalzbora implements Uredjivac {
    public void uredi(Uporediv[] niz) { }
    public void uredi(Uporediv[] niz);
```

e) Kôd koji je na jeziku C# generisao StarUML

```
// @ File Name : Uporediv.cs
public interface Uporediv {
    abstract boolean ispred(Uporediv u);
    abstract boolean jednak(Uporediv u);
}
```

```
// @ File Name : Krug.cs
public class Krug : Uporediv{
    private double r ;
    public Krug(double r){ }
    public boolean ispred(Uporediv u){ }
    public boolean jednak(Uporediv u){ }
}
```

```
// @ File Name : Uredjivac.cs
public interface Uredjivac {
    abstract void uredi(Uporediv[] niz);
}
```

```
// @ File Name : MetodaIzbora.cs
public class MetodaIzbora : Uredjivac{
    public void uredi(Uporediv[] niz){ }
}
```

f) Kôd koji je na jeziku C++ generisao StarUML

```
// @ File Name : Uporediv.h
#ifndef _UPOREDIV_H
#define _UPOREDIV_H
class Uporediv {
public:
    virtual boolean ispred(Uporediv u) = 0;
    virtual boolean jednak(Uporediv u) = 0;
};
#endif // _UPOREDIV_H
```

```
// @ File Name : Krug.h
#ifndef _KRUG_H
#define _KRUG_H
#include "Uporediv.h"
class Krug : public Uporediv {
public:
    Krug(double r);
    boolean ispred(Uporediv u);
    boolean jednak(Uporediv u);
    boolean ispred(Uporediv u);
    boolean jednak(Uporediv u);
private:
    double r;
};
#endif // _KRUG_H
```

```
// @ File Name : Krug.cpp
#include "Krug.h"
#include "Uporediv.h"
void Krug::Krug(double r) { }
boolean Krug::ispred(Uporediv u) { }
boolean Krug::jednak(Uporediv u) { }
boolean Krug::ispred(Uporediv u) { }
boolean Krug::jednak(Uporediv u) { }
```

```
// @ File Name : Uredjivac.h
#ifndef _UREDJIVAC_H
#define _UREDJIVAC_H
class Uredjivac {
public:
    virtual void uredi(Uporediv[] niz) = 0;
};
#endif // _UREDJIVAC_H
```

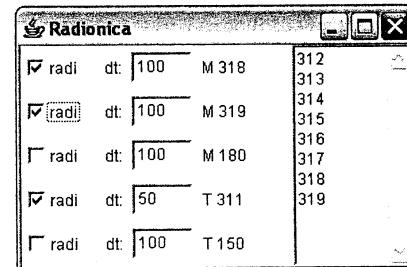
```
// @ File Name : MetodaIzbora.h
#ifndef _METODAIZBORA_H
#define _METODAIZBORA_H
#include "Uredjivac.h"
class MetodaIzbora : public Uredjivac {
public:
    void uredi(Uporediv[] niz);
    void uredi(Uporediv[] niz);
};
#endif // _METODAIZBORA_H
```

```
// @ File Name : MetodaIzbora.cpp
#include "MetodaIzbora.h"
void MetodaIzbora::uredi(Uporediv[] niz) { }
void MetodaIzbora::uredi(Uporediv[] niz) { }
```

Zadatak 4 Proizvod, skladište i mehanizmi (dijagram paketa; aktivna klasa)

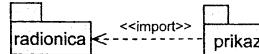
Nacrtati dijagram paketa i dijagram klasa na jeziku UML sledećeg sistema tipova:

- Proizvod** ima jednoznačan, automatski generisan identifikacioni broj. Tekstualni oblik proizvoda sadrži samo njegov identifikacioni broj.
- Skladište** proizvoda može da smesti zadati broj proizvoda. Stvara se prazno, posle čega može da se dodaje i uzima po jedan proizvod. Pri pokušaju stavljanja u puno skladište, odnosno uzimanja iz praznog skladišta, niti izvršioča radnje privremeno se blokira. Tekstualni oblik skladišta sastoji se od niza tekstuálnih oblika sadržanih proizvoda, jedan proizvod po redu.
- Astraktan **mehanizam** predviđa izvođenje neke radnje.
- Mašina** je mehanizam čija se radnja sastoji od stvaranja jednog proizvoda i njegovog stavljanja u skladište koje je zadato prilikom stvaranja mašine. Tekstualni oblik sadrži slovo **M** i tekstualni oblik proizvoda koji je upravo stavljen u skladište.
- Pokretna **traka** je mehanizam čija se radnja sastoji od uzimanja jednog proizvoda iz skladišta koje je zadato prilikom stvaranja trake. Tekstualni oblik sadrži slovo **T** i tekstualni oblik proizvoda koji je upravo uzet iz skladišta.
- Aktivan **motor** ponavlja radnju jednog mehanizma koji se zadaje prilikom stvaranja motora u slučajnim vremenskim intervalima $\Delta t \pm 20\%$ ms (na početku je $\Delta t = 100$ ms i može da se promeni za vreme života motora). Rad motora može da se zaustavi, da se nastavi dalje i da se prekine.
- Grafičko **skladište** je skladište koje se inicijalizuje grafičkim tekstualnim prostorom (TextArea) na kome prikazuje svaku promenu stanja skladišta.
- Grafički **motor** je motor koji se inicijalizuje grafičkom pločom (Panel) koju popunjava komponentama tako da korisnik može zaustaviti i nastaviti rad motora, promeniti vremenski interval Δt (promena ima efekta odmah) i može da se prikaže tekstualni oblik pridruženog mehanizma.
- Radionica** je program koji na grafičkoj korisničkoj površi prema slici prikazuje rad s jednim skladistom kapaciteta 20, tri mašine i dve pokretnе trake.

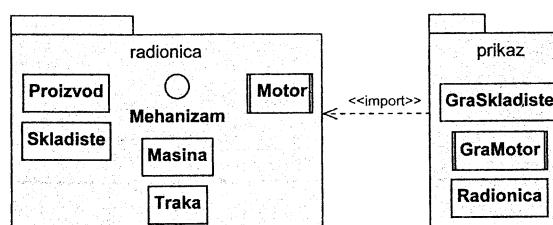


Rešenje:

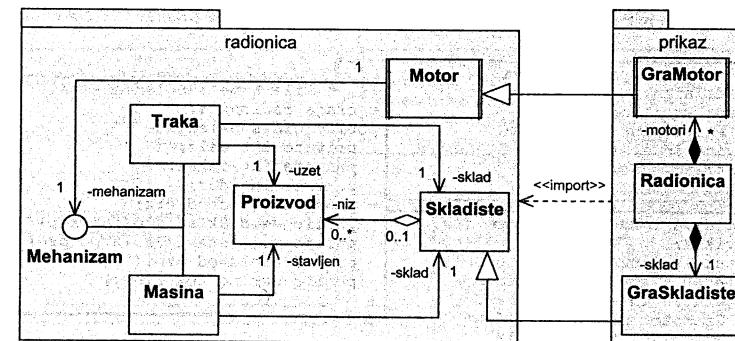
a) Dijagram paketa



b) Dijagram paketa sa sadržajem



c) Odnosi među klasama



d) Detalji klasa

radionica::Proizvod	radionica::Mehanizam	radionica::Motor
-posId: int = 0 -id: int = +posId +toString(): String	+radnja()	-radi: boolean -dt: int <<create>>+Motor(m: Mehanizam) +postaviDt(dt: int) #radnja() +run() +kreni() +stani() +zavrsi()
	radionica::Skladište	
-uatz: int -izlaz: int -duz: int <<create>>+Skladište(kap: int) +stavi(p: Proizvod) +uzmi(): Proizvod +toString(): String	<<create>>+Masina(s: Skladište) +radnja() +toString(): String	
	radionica::Traka	
	<<create>>+Traka(s: Skladište) +radnja() +toString(): String	
	prikaz::GraSkladište	prikaz::GraMotor
-prikaz: TextArea <<create>>+GraSkladište(kap: int, prikaz: TextArea) +stavi(p: Proizvod) +uzmi(): Proizvod	-potRadi: Checkbox -oznMehanizam: Label -ploca: Panel <<create>>+GraMotor(m: Mehanizam, plo: Panel) #radnja()	
		prikaz::Radionica
		-BR_M: int = 3 -BR_T: int = 2 <<create>>+Radionica() +popuniPrzor() +main(varargin: String[])

- e) Kôd koji je na jeziku Java generisao StarUML
- paketi su stavljeni u odgovarajuće potfascikle
 - klase iz drugih paketa koriste se punim imenom

```
// @ File Name : Proizvod.java
package radionica;
public class Proizvod {
    private static int posId = 0;
    private int id = ++posId;
    public String toString() { }
}

// @ File Name : Radionica.java
package prikaz;
public class Radionica {
    private static int BR_M = 3;
    private static int BR_T = 2;
    private GraMotor motori;
    private GraSkladiste sklad;
    public void Radionica() { }
    private void popuniProzor() { }
    public static void main(String[] varg) { }
}

// @ File Name : Skladiste.java
package radionica;
public class Skladiste {
    private int ualz;
    private int izlaz;
    private int duz;
    private Proizvod niz;
    public void Skladiste(int kap) { }
    public void stavi(Proizvod pro) { }
    public Proizvod uzmi() { }
    public String toString() { }
}

// @ File Name : GraSkladiste.java
package prikaz;
public class GraSkladiste extends radionica.Skladiste {
    private TextArea prikaz;
    public void GraSkladiste(int kap, TextArea prikaz) { }
    public void stavi(radionica.Proizvod p) { }
    public radionica.Proizvod uzmi() { }
}
```

- f) Kôd koji je na jeziku C# generisao StarUML
- paketi su stavljeni u odvojene prostore imena i potfascikle
 - upotreba klasa iz drugog paketa nije korektna

```
// @ File Name : Proizvod.cs
namespace radionica{
    public class Proizvod {
        private static int posId = 0;
        private int id = ++posId;
        public String toString() { }
    }
}

// @ File Name : Radionica.cs
namespace prikaz{
    public class Radionica {
        private static int BR_M = 3;
        private static int BR_T = 2;
        private GraMotor motori;
        private GraSkladiste sklad;
        public Radionica() { }
        private void popuniProzor() { }
        public static void main(String[] varg) { }
    }
}

// @ File Name : Skladiste.cs
namespace radionica{
    public class Skladiste {
        private int ualz ;
        private int izlaz ;
        private int duz ;
        private Proizvod niz;
        public Skladiste(int kap){ }
        public void stavi(Proizvod pro){ }
        public Proizvod uzmi(){ }
        public String toString() { }
    }
}

// @ File Name : GraSkladiste.cs
namespace prikaz{
    public class GraSkladiste : Skladiste{
        private TextArea prikaz ;
        public GraSkladiste(int kap, TextArea prikaz) { }
        public void stavi(Proizvod p) { }
        public Proizvod uzmi() { }
    }
}
```

- g) Kôd koji je na jeziku C++ generisao StarUML

```
// @ File Name : Proizvod.h
#ifndef _PROIZVOD_H_
#define _PROIZVOD_H_
namespace radionica {
    class Proizvod {
    public:
        String toString();
    private:
        static int posId = -0;
        int id = ++posId;
    };
#endif // _PROIZVOD_H_

// @ File Name : Skladiste.h
#ifndef _SKLADISTE_H_
#define _SKLADISTE_H_
#include "Proizvod.h"
namespace radionica {
    class Skladiste {
    public:
        Skladiste(int kap);
        void stavi(Proizvod pro);
        Proizvod uzmi();
        String toString();
    private:
        int ualz;
        int izlaz;
        int duz;
        Proizvod *niz;
    };
#endif // _SKLADISTE_H_

// @ File Name : GraSkladiste.h
#ifndef _GRASKLADISTE_H_
#define _GRASKLADISTE_H_
#include "Skladiste.h"
#include "Proizvod.h"
namespace prikaz {
    class GraSkladiste : public Skladiste {
    public:
        GraSkladiste(int kap, TextArea prikaz);
        void stavi(Proizvod p);
        Proizvod uzmi();
    private:
        TextArea prikaz;
    };
#endif // _GRASKLADISTE_H_

// @ File Name : Radionica.h
#ifndef _RADIONICA_H_
#define _RADIONICA_H_
#include "GraMotor.h"
#include "GraSkladiste.h"
namespace prikaz {
    class Radionica {
    public:
        Radionica();
        static void main(String[] varg);
    private:
        static int BR_M = -3;
        static int BR_T = -2;
        void popuniProzor();
        GraMotor motori;
        GraSkladiste sklad;
    };
#endif // _RADIONICA_H_

// @ File Name : Proizvod.cpp
#include "Proizvod.h"
namespace radionica {
    String Proizvod::toString() { }

// @ File Name : Skladiste.cpp
#include "Skladiste.h"
#include "Proizvod.h"
namespace radionica {
    Skladiste::Skladiste(int kap) { }
    void Skladiste::stavi(Proizvod pro) { }
    Proizvod Skladiste::uzmi() { }
    String Skladiste::toString() { }

// @ File Name : GraSkladiste.cpp
#include "GraSkladiste.h"
#include "Proizvod.h"
namespace prikaz {
    GraSkladiste::GraSkladiste(int kap, TextArea prikaz) { }
    void GraSkladiste::stavi(Proizvod p) { }
    Proizvod GraSkladiste::uzmi() { }

// @ File Name : Radionica.cpp
#include "Radionica.h"
namespace prikaz {
    Radionica::Radionica() { }
    void Radionica::main(String[] varg) { }
    void Radionica::popuniProzor() { }
}
```

Zadatak 5 Samoposluga (generisanje modela – Java)

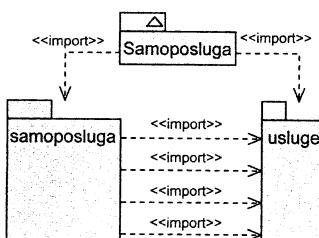
Nacrtati dijagram paketa i dijagrame klase za sledeći gotov sistem klasa napisan na jeziku Java (videti zadatak 8.14 u zbirci zadataka [6]) koji simulira rad samoposluge:

- Aktivan **ulaz** samoposluge ima jedinstven, automatski generisan identifikacioni broj koji može da se dohvati. Ulaz može da se otvori, zatvari i uništi. Kada je ulaz otvoren kupci ulaze u slučajnim vremenskim intervalima. Najduže vreme između ulazaka kupaca je parametar ulaza. Prikaz ulaza u polju grafičke korisničke površi sadrži identifikacioni broj ulaza. Ako je ulaz otvoren prikaz sadrži i identifikacioni broj kupca koji je poslednji ušao.
- Aktivan **kupac** ima jedinstven, automatski generisan identifikacioni broj koji može da se dohvati. Po ulasku u samoposlužu neko slučajno vreme bira robu (najduže vreme je parametar kupca), potom stane u red ispred slučajno odabранe kase. Posle plaćanja napušta samoposlužu.
- Red** kupaca ispred kase je neograničenog kapaciteta. Kupci se dodaju na kraj reda, a uzimaju sa početka reda. Prikaz reda u polju grafičke korisničke površi sadrži identifikacione brojeve kupaca u redu.
- Aktivna **kasa** ima jedinstven, automatski generisan, identifikacioni broj koji može da se dohvati. Ispred kase postoji jedan red kupaca. Kasa može da se otvori, zatvori i uništi. Kada je kasa otvorena naplaćuju od kupaca koji stoje u redu ispred nje. Naplaćivanje traje slučajno vreme. Najduže vreme naplaćivanja je parametar kase. Prikaz kase u polju grafičke korisničke površi sadrži identifikacioni broj kase. Ako je naplaćivanje u toku, prikaz sadrži i identifikacioni broj kupca koji se opslužuje.
- Samoposluga** ima jedinstven, automatski generisan, identifikacioni broj koji može da se dohvati. U samoposluži postoji zadati broj ulaza i kasa. U početku ulazi su zatvoreni, a kase su spremne za rad. Samoposluža može da se otvori, zatvori i uništi. Pri otvaranju svi ulazi odmah se otvore. Pri zatvaranju svi ulazi odmah se zatvore. Zatvaranje se smatra završenim kada i poslednji kupac napusti samoposlužu. Prikaz samoposluge u polju grafičke korisničke površi sadrži broj kupaca u samoposluži.

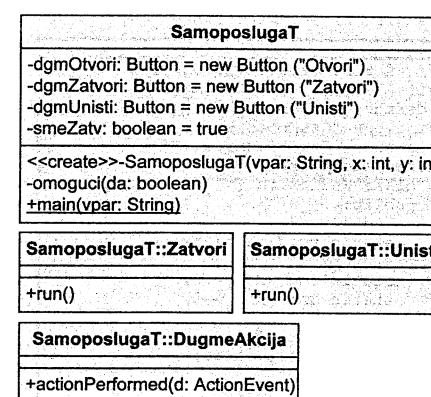
Rešenje:

Sledeći model je generisan pomoću StarUML.

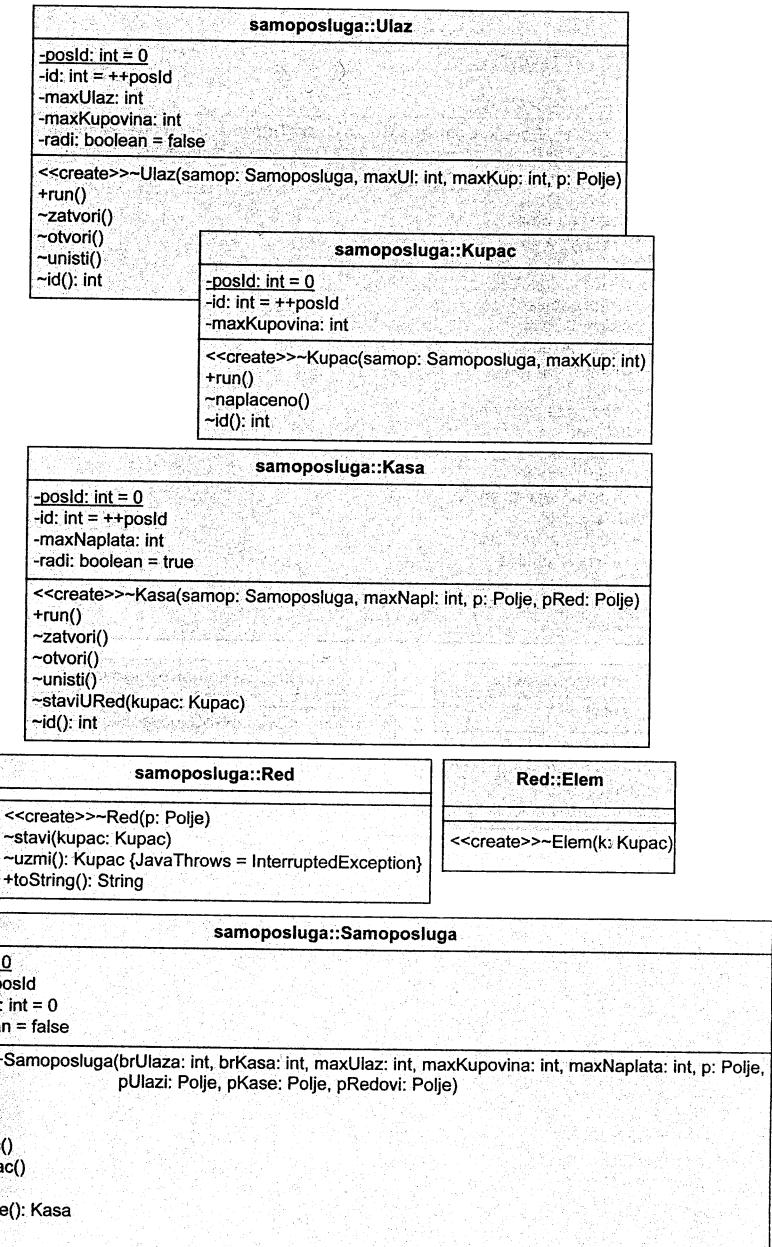
a) Dijagram paketa



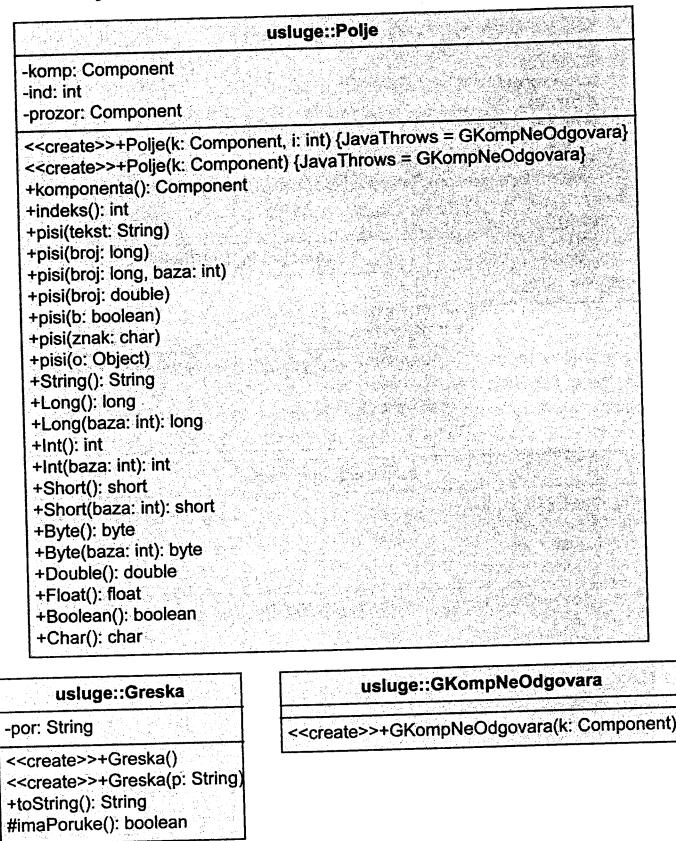
b) Klase u bezimenom paketu



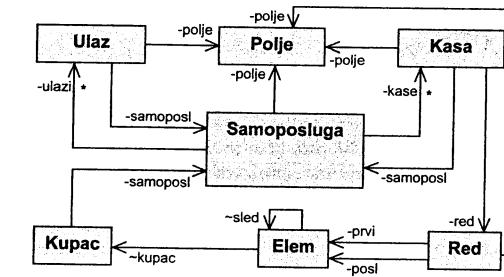
c) Klase u paketu samoposluga



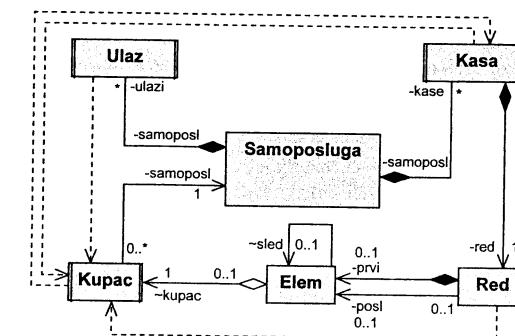
d) Klase u paketu usluge



e) Generisani dijagram klasa



f) Poboljšani dijagram klasa (bez klase Polje)



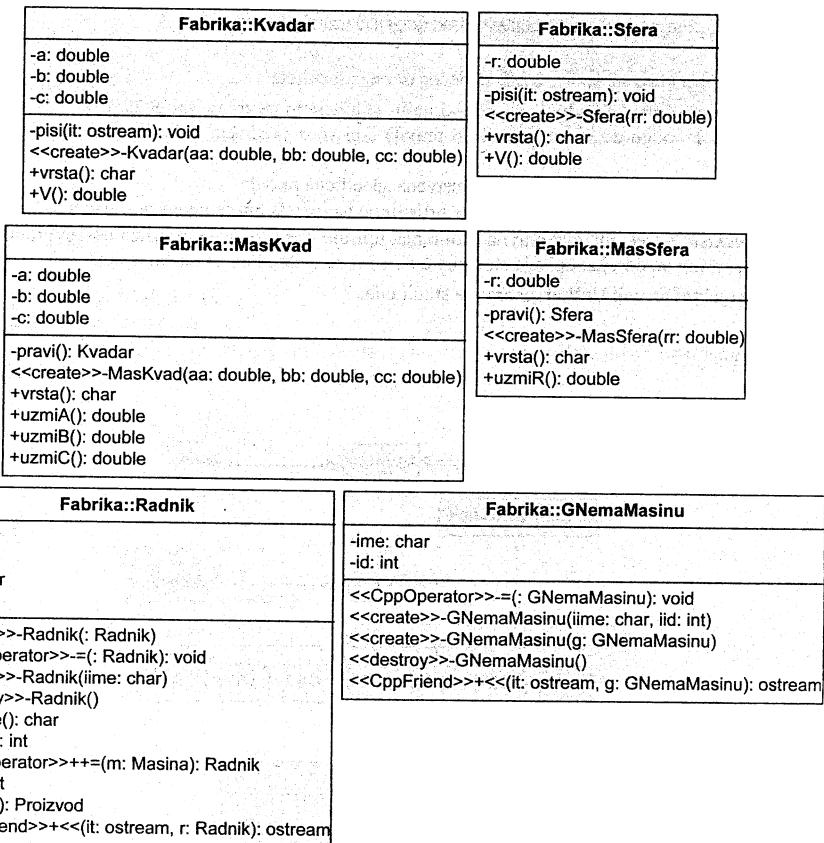
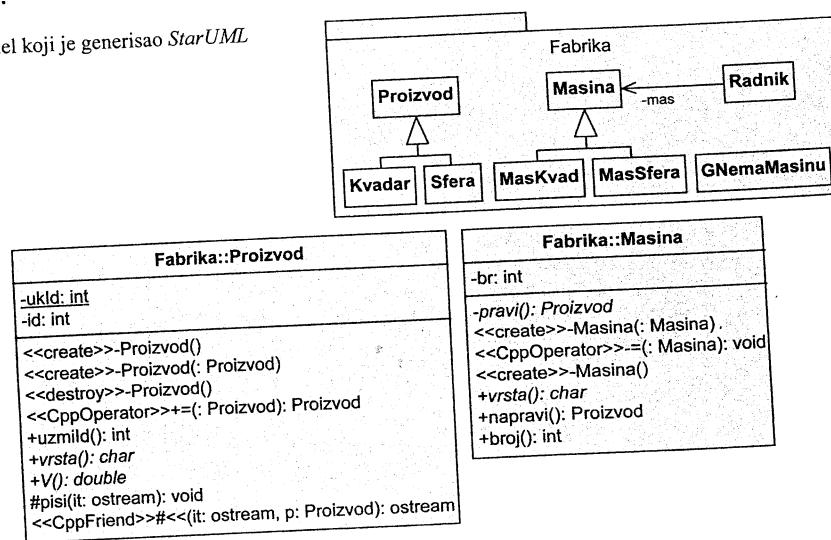
Zadatak 6 Proizvodi, mašine i radnik (generisanje modela – C++)

Nacrtati dijagram paketa i dijagrame klase za sledeći gotov sistem klasa napisan na jeziku C++ (videti zadatak 5.9 u zbirci zadataka [7]):

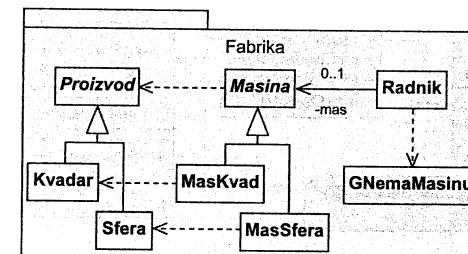
- Apstraktan **proizvod** ima jednoslovnu oznaku vrste i jedinstven, automatski generisan identifikacioni broj. Može da se dohvati vrsta proizvoda i identifikacioni broj i da se izračuna zapremina. Pri pisanju u izlazni tok (`it << p`) piše se vrsta i identifikacioni broj proizvoda.
- Apstraktna **mašina** može da proizvodi apstraktan proizvod i zna se koliko je proizvoda ta mašina proizvela. Može da se dohvati oznaka vrste proizvoda koje data mašina proizvodi i broj proizvedenih proizvoda. Ne može da se stvara kopija maštine.
- Kvadar** je proizvod zadat dužinama ivica. Oznaka vrste proizvoda je **K**. Pri pisanju navode se i dimenzije kvadra.
- Sfera** je proizvod zadat poluprečnikom. Oznaka vrste proizvoda je **S**. Pri pisanju navodi se i poluprečnik sfere.
- Mašina za kvadre** je mašina koja može da proizvodi kvadre zadatih dimenzija. Parametri maštine (dimenzije kvadara koje proizvodi) ne mogu da se promene, ali mogu da se dohvate.
- Mašina za sfere** je mašina koja može da proizvodi sfere zadatog poluprečnika. Parametar maštine (poluprečnik sfere koje proizvodi) ne može da se promeni, ali može da se dohvati.
- Radnik** ima ime i jedinstven, automatski generisan identifikacioni broj. Izrađuje proizvode određene vrste pomoću odgovarajuće maštine. Mašina na kojoj radnik radi može da se promeni (`r+=m`). Zna se koliko je proizvoda izradio na maštini na kojoj trenutno radi. Radnik može da ne bude rasporenjen ni na jednu mašinu. U tom slučaju pokušaj izrade proizvoda ili dohvatanja broja izrađenih reden na jednu mašinu. U slučaju da je radnik rasporenjen na neku mašinu piše se i vrsta i identifikacioni broj radnika. U slučaju da je radnik rasporenjen na neku mašinu piše se i vrsta i identifikacioni broj proizvoda koji je izradio na toj maštini od početka rada na proizvoda koje trenutno izrađuje i broj proizvoda koje je izradio na toj maštini od početka rada na njoj.

Rešenje:

a) Model koji je generisao StarUML



b) Poboljšani model



Zadatak 7 Pravila, student, studentski odsek (projektni uzorak **Unikat**)

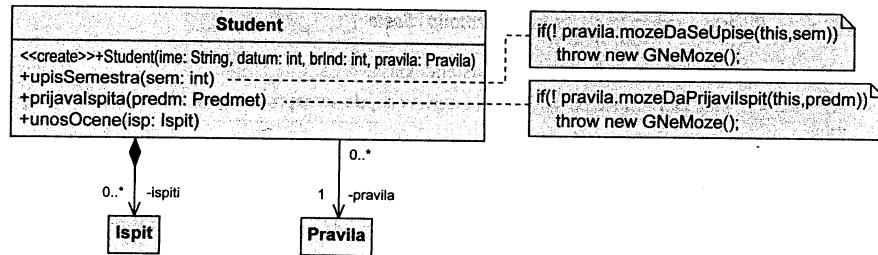
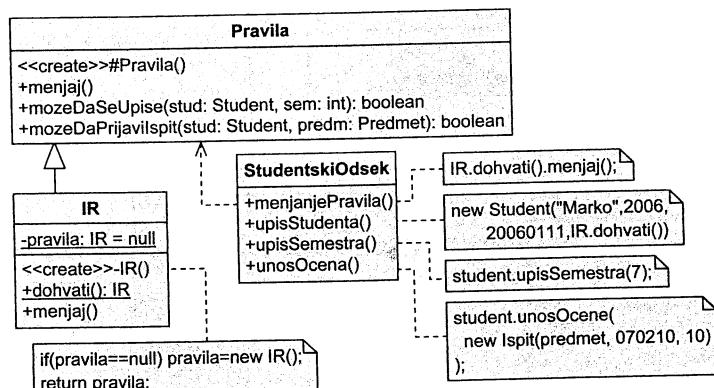
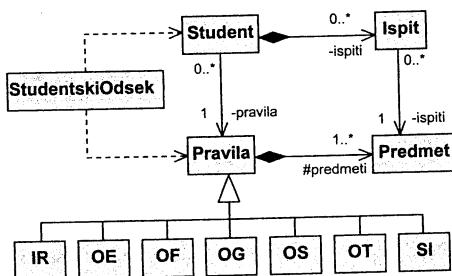
Nacrtati dijagram klasa na jeziku UML sledećeg modela fakulteta:

- **Pravila** studiranja određuju uslove pod kojima student može da upiše semestar i da prijavljuje ispiske. Pravila mogu da se menjaju. Deo pravila čini skup predmeta iz kojih student treba da polaže ispiske.
- Za svaki odsek fakulteta postoje jedinstvena specifična pravila.
- **Student** može da upisuje semestar, da prijavljuje ispiske i da mu se unose ocene ispita.
- **Studentski odsek** održava pravila studiranja, upisuje nove studente pridružujući im pravila studiranja zavisno od odseka, upisuje studente u naredne semestre i unosi ocene ispita.

Koristiti projektni uzorak **Unikat** za pravila studiranja.

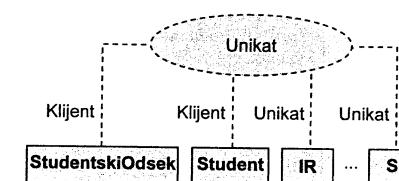
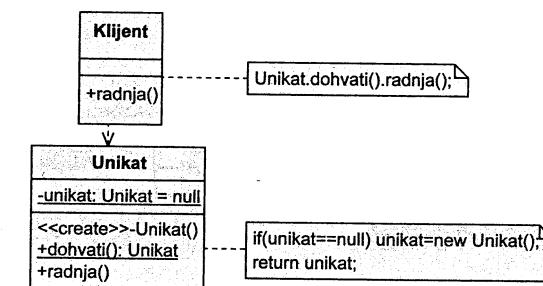
Rešenje:

a) Dijagrami klase



b) Projektni uzorak **Unikat (Singleton)**

- objektni uzorak stvaranja
- obezbediti da postoji samo jedan primerak klase čije se stvaranje odlaže do prvog pristupa objektu
- klijent pristupa unikatnom objektu isključivo pristupnom metodom



• Alternativno rešenje:

- samo zajednički atributi i metode
- ne pravi se nijedan objekat
 - privatni konstruktor koga niko ne poziva
- ne može da se obezbedi polimorfno ponašanje

Zadatak 8 Predmeti i radnici (projektni uzorci **Prototip** i **Sastav**)

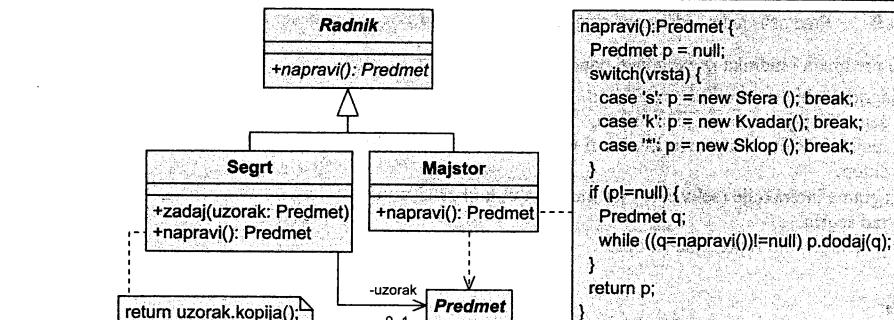
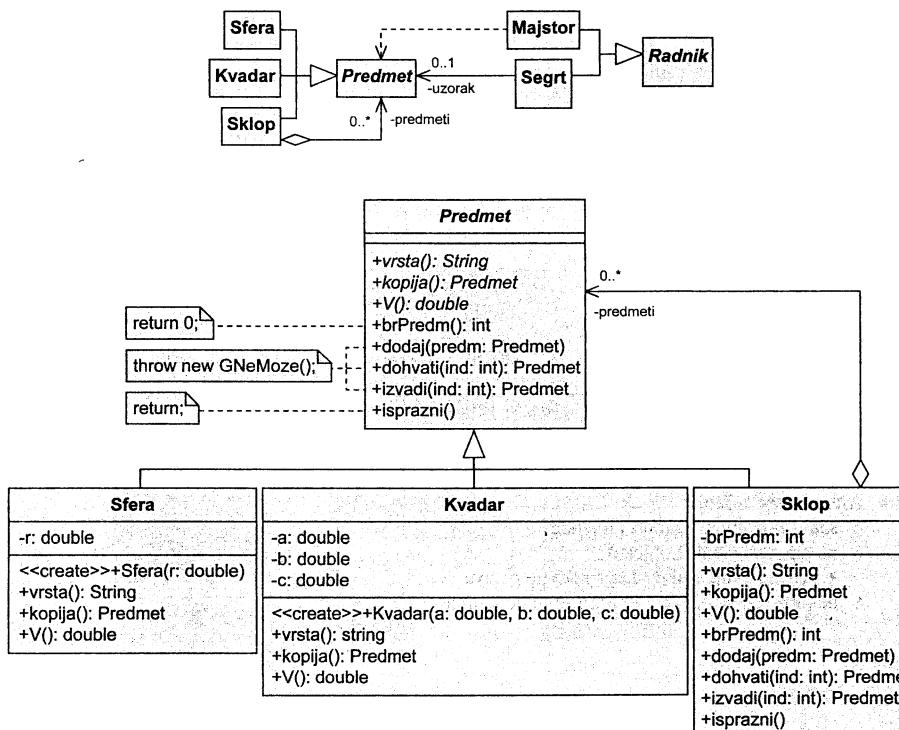
Nacrtati dijagram klasa na jeziku UML sledećeg sistema klasa:

- Apstraktnom **predmetu** može da se dohvati naziv vrste, da se napravi kopija i da se izračuna zapremina.
- Sfera** i **kvadar** su predmeti zadati poluprečnikom, odnosno dužinama ivica.
- Sklop** je predmet koji može da sadrži proizvoljan broj predmeta proizvoljne vrste. Stvara se prazan, posle čega se predmeti dodaju jedan po jedan. Može da se dohvati broj predmeta u sklopu, da se dohvati i da se izvadi predmet sa zadatim rednim brojem i da se sklop isprazni.
- Apstraktan **radnik** može da napravi jedan predmet.
- Segrt** je radnik koji predmete pravi kao verne kopije drugog predmeta koji mu se daje kao uzorak. Uzorak može da se zameni drugim predmetom.
- Majstor** je radnik koji može da napravi predmet po svom nahođenju.

Koristiti projektnе uzorce **Prototip** i **Sastav**.

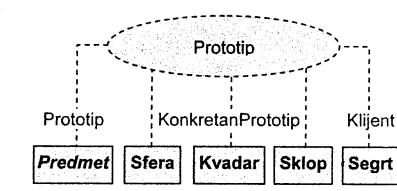
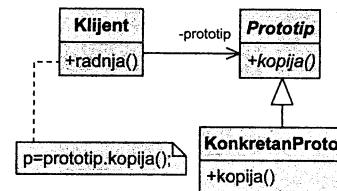
Rešenje:

a) Dijagrami klasa



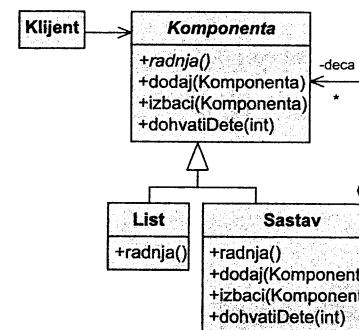
b) Projektni uzorak **Prototip** (*Prototype*)

- objektni uzorak stvaranja
- polimorfno kopiranje objekata na osnovu zadatog originala
- klijent traži od originala da napravi svoju kopiju



c) Projektni uzorak **Sastav** (*Composite*)

- objektni uzorak strukture
- hijerarhijska struktura delova i sklopova čiji elementi mogu da se obrađuju uniformno, nezavisno da li su čvorovi ili listovi
- klijent objekte koristi kroz zajednički interfejs komponente



• Alternativno rešenje

- rukovanje sastavom ne nalazi se u komponenti već samo u sastavu
 - mana**: nije isti interfejs za listove i za sastav
 - prednost**: bezbednije je jer se na listove ne mogu primeniti neprimerene radnje

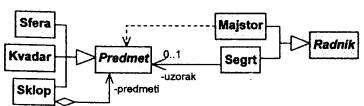
Zadatak 9 Predmeti i radnici (dijagrami objekata i interakcije)

Za model predmeta i radnika iz zadatka 8 nacrtati na jeziku UML:

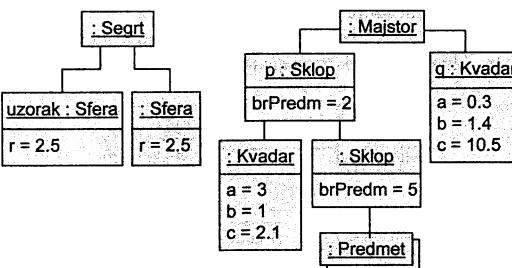
- dijagrame objekata koji prikazuju:
 - šegrt koji je napravio jednu sferu,
 - majstora koji je napravio kvadar u toku pravljenja sklopa koji već sadrži jedan kvadar i jedan sklop;
- dijagrame interakcije (sekvene i komunikacije) koji prikazuju:
 - rad šegrt,
 - izračunavanje zapremine sklopa.

Rešenje:

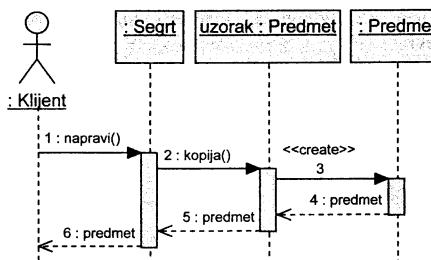
a) Ponovljeni dijagram klasa



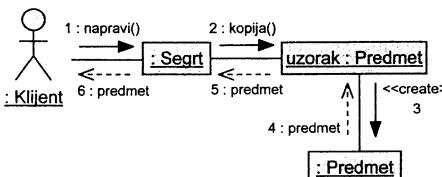
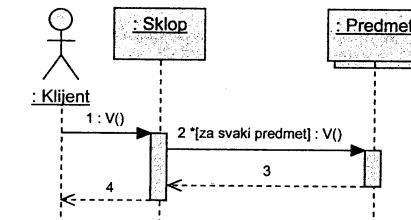
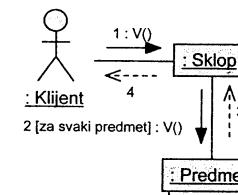
b) Dijagrami objekata



c) Dijagram sekvene rada šegrt-a



d) Dijagram komunikacije rada šegrt-a

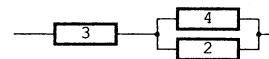

e) Dijagram sekvene računanja zapremine sklopa

f) Dijagram komunikacije računanja zapremine sklopa


Zadatak 10 Otpornici {K1, 10.11.2006.}

Apstraktnom otporniku može da se odredi otpornost i da se napravi njegova kopija. Prostom otporniku se zna vrednost njegove otpornosti. Apstraktan složen otpornik je otpornik koji se sastoji od proizvoljnog broja otpornika. Stvara se prazan, posle čega mogu da mu se dodaju komponentni otpornici. Redna veza otpornika je složen otpornik čija je otpornost jednak zbiru otpornosti sadržanih otpornika. Paralelna veza otpornika je složen otpornik čija je otpornost jednaka recipročnoj vrednosti zbiru recipročnih vrednosti sadržanih otpornika.

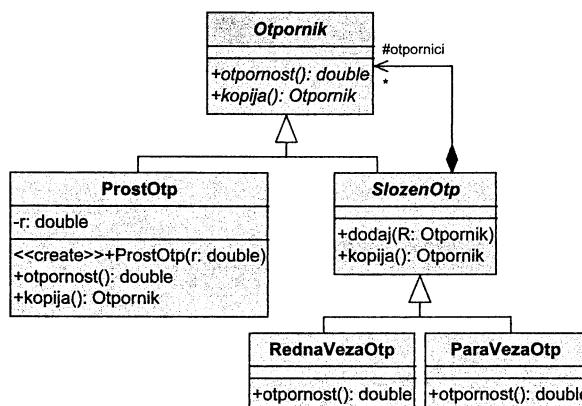
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram objekata koji predstavlja vezu otpornika sa slike,
- dijagram sekvene za određivanje otpornosti složenog otpornika.

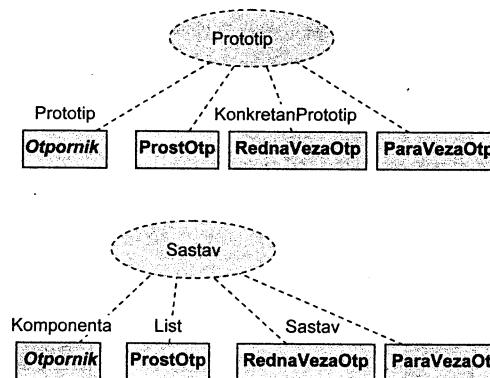


Rešenje:

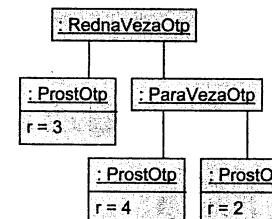
a) Dijagram klasa i projektni uzorci



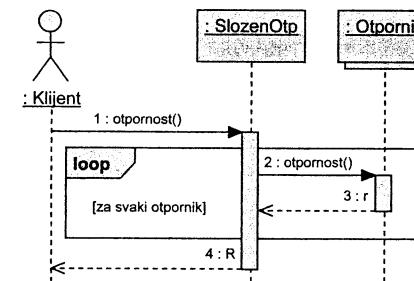
b) Projektni uzorci



c) Dijagram objekata



d) Dijagram sekvene računanja otpornosti



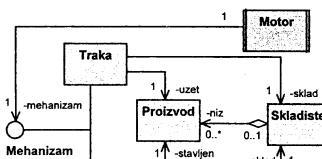
Zadatak 11 Proizvod, skladište, mehanizmi i motor

Za model proizvoda, mehanizma, skladišta i motora iz zadatka 4 nacrtati na jeziku UML:

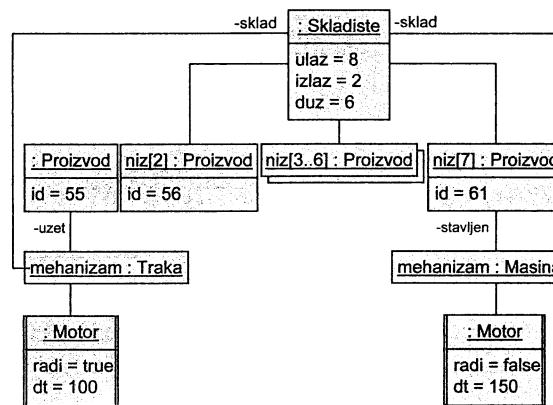
- dijagram objekata koji prikazuje skladište s nekoliko proizvoda, jedan motor koji pokreće mašinu i jedan motor koji pokreće traku;
- dijagrame interakcije (sekvence i komunikacije) koji prikazuju:
 - stvaranje mašine i izvršavanje radnje maštine,
 - stvaranje motora koji pokreće traku i rad motora.

Rešenje:

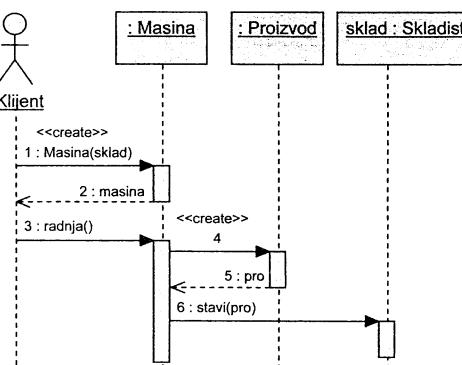
a) Ponovljeni dijagram klasa



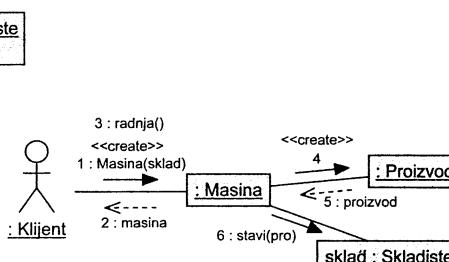
b) Dijagram objekata



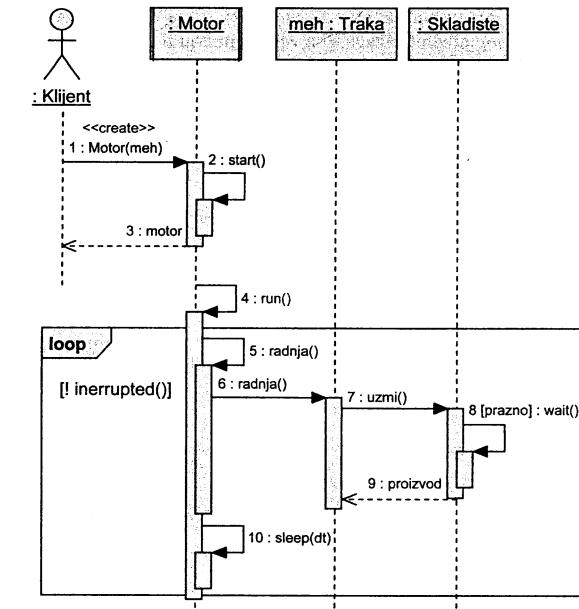
c) Dijagram sekvenca korišćenja maštine



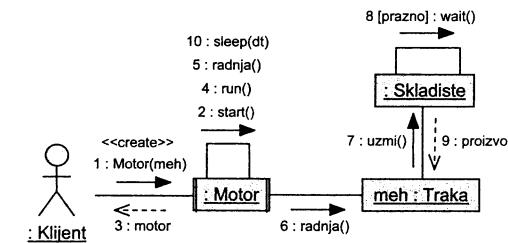
d) Dijagram komunikacije korišćenja maštine



e) Dijagram sekvenca korišćenja motora



f) Dijagram komunikacije korišćenja motora



Zadatak 12 Vektor, predmeti i orijentisani predmeti {K1, 01.11.2007.}

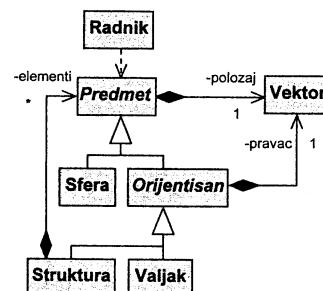
Vektor u prostoru zadaje se realnim koordinatama koje mogu da se dohvate. Vektoru može da se doda drugi vektor. Položaj apstraktog predmeta zadaje se vektorom koji može da se dohvati. Može da se izračuna zapremina predmeta. Orijentisan predmet je predmet koji ima i pravac u prostoru određen drugim vektorom koji može da se dohvati. Sfera je predmet zadatog poluprečnika koji može da se dohvati. Valjak je orijentisan predmet zadat poluprečnikom osnove i visinom koji mogu da se dohvate. Struktura je orijentisan predmet koji može da sadrži proizvoljan broj predmeta. Stvara se prazna posle čega može da joj se doda po jedan predmet. Radnik ima ime koje može da se dohvati. Radnik može da napravi jedan predmet po svom nahođenju.

Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram objekata koji predstavlja radnika koji pravi strukturu koja već sadrži jednu sferu i jedan valjak i upravo želi da strukturi doda još jednu sferu,
- dijagram sekvene za izradu jedne strukture koja sadrži samo sfere i valjke.

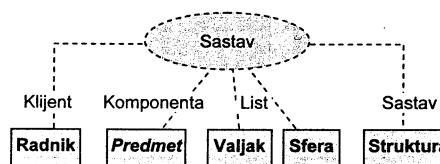
Rešenje:

a) Dijagram klasa

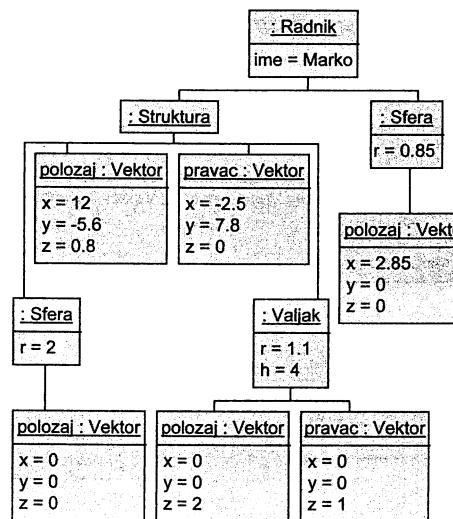


Vektor	
-x: double	
-y: double	
-z: double	
<<create>>+Vektor(a: double, b: double, c: double)	
+uzmiX(): double	
+uzmiY(): double	
+uzmiZ(): double	
+dodaj(vekt: Vektor): Vektor	

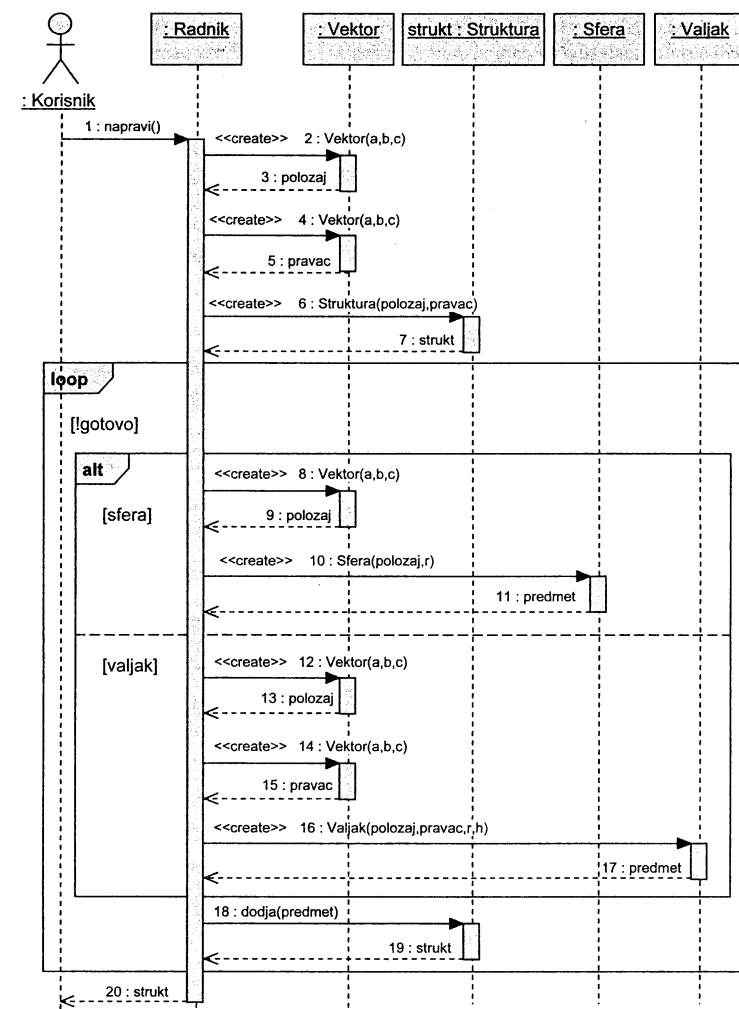
Predmet	Orijentisan
<<create>>#Predmet(polozaj: Vektor) +uzmiPolozaj(): Vektor +zapremina(): double	<<create>>#Orijentisan(polozaj: Vektor, pravac: Vektor) +uzmiPravac(): Vektor
Sfera	
-r: double	
<<create>>+Sfera(polozaj: Vektor, r: double) +uzmiR(): double +zapremina(): double	
Valjak	
-r: double	
-h: double	
<<create>>+Valjak(polozaj: Vektor, pravac: Vektor, r: double, h: double) +uzmiR(): double +uzmiH(): double +zapremina(): double	
Struktura	
<<create>>+Struktura(polozaj: Vektor, pravac: Vektor) +dodja(predmet: Predmet): Struktura +zapremina(): double	
Radnik	
-ime: String	
<<create>>+Radnik(ime: String) +uzmilme(): String +napravi(): Predmet	

b) Projektni uzorak *Sastav*

c) Dijagram objekata



d) Dijagram sekvene

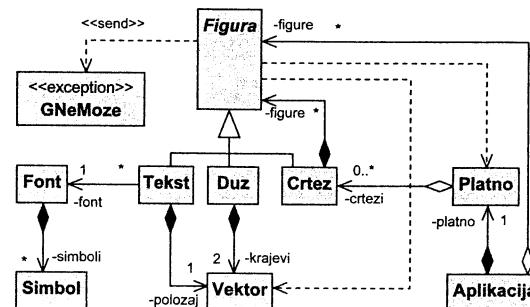


Zadatak 13 Simbol, font, vektor, figura, platno i aplikacija {K1, 30.10.2008.}

Simbol sadrži jedan znak i odnos širine i visine pri ispisivanju znaka. Može da se dohvati sadržani znak i da se odredi širina ispisivanja za datu visinu. Font sadrži proizvoljan broj simbola. Stvara se prazan posle čega se simboli dodaju jedan po jedan. Može da se dohvati broj simbola u fontu i da se dohvati simbol koji sadrži zadati znak. Vektor u ravni zadaje se realnim komponentama u pravcu koordinatnih osa. Može da se vektoru doda drugi vektor. Apstraktna figura u ravni može da se pomeri za zadati vektor pomaka, da se napravi kopija figure i da se figura prikaže na zadatom platnu. Duž je figura koja sadrži dva vektora položaja krajnjih tačaka. Tekst je figura koja sadrži niz znakova koji se iscrtava simbolima zadate visine, primenom zadatog fonta sa zadatim vektorom položaja donjem levog temena prvog simbola. Visina i font mogu da se promene. Crtič je figura koja sadrži proizvoljan broj figura. Stvara se prazan posle čega se figure dodaju jedna po jedna. Aplikacija ima jedinstveno platno koje može da iscrtava sadržane crteže. Crtiči mogu da mu se dodaju i uklanjuju jedan po jedan.

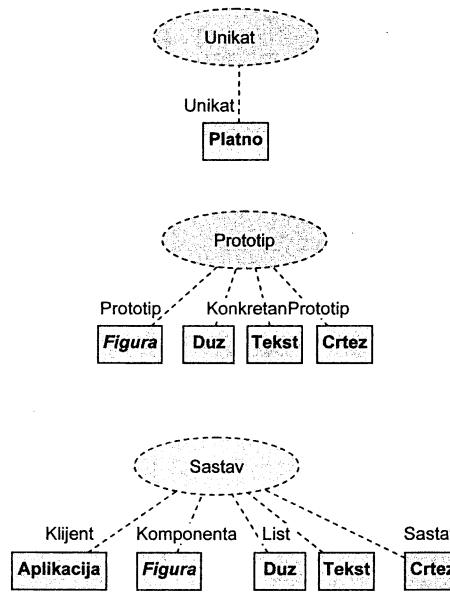
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram objekata koji prikazuje crtež s jednom duži i jednim tekstrom,
- dijagram sekvence za jedno prikazivanje platna u najopštijem slučaju.

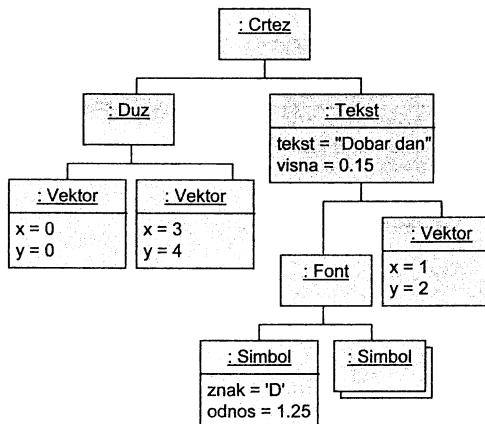
Rešenje:
a) Dijagram klasa


Simbol	Font
-znak: char -odnos: double +dodaj(s: Simbol) +brojSimbola(): int +dohvati(znak: char): Simbol +sirina(visina: double): double	+dodaj(s: Simbol) +brojSimbola(): int +dohvati(znak: char): Simbol
Vektor	
-x: double -y: double +dodaj(v: Vektor): Vektor	
Figura	Duz
+dodaj(f: Figura) raises GNeMoze +pomeri(pomak: Vektor): Figura +kopija(): Figura +prikazi(p: Platno)	+dodaj(a: Vektor, b: Vektor) +pomeri(pomak: Vektor): Figura +kopija(): Figura +prikazi(p: Platno)
Tekst	
+tekst: String +visina: double +dodaj(tekst: String, f: Font, visina: double, polozaj: Vektor) +pomeri(pomak: Vektor): Figura +kopija(): Figura +prikazi(p: Platno) +postaviVisinu(visina: double) +postaviFont(f: Font)	
Platno	Aplikacija
-primerak: Platno = null +dohvatiPlatno(): Platno +dodaj(c: Crtic) +ukloni(c: Crtic) +prikazi()	+main(varg: String[])

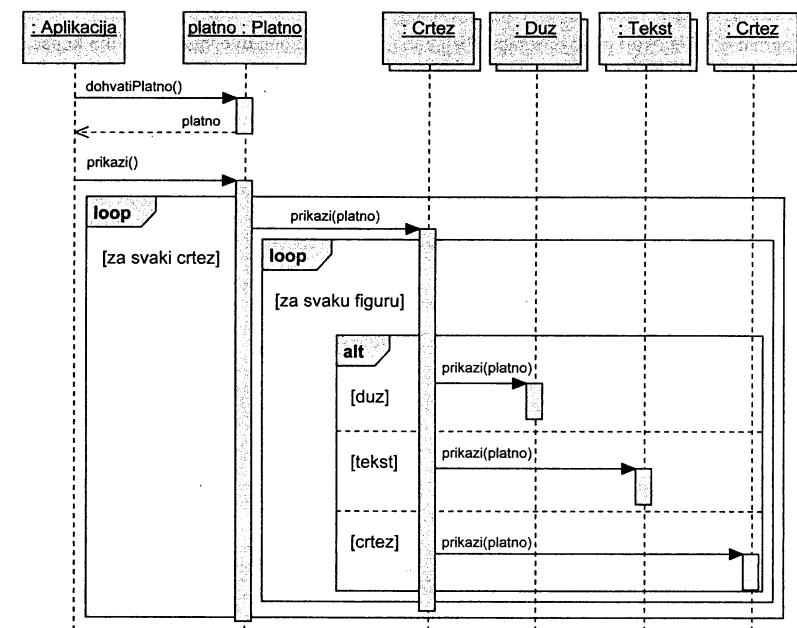
b) Projektni uzorci



c) Dijagram objekata



d) Dijagram sekvenca prikazivanja platna



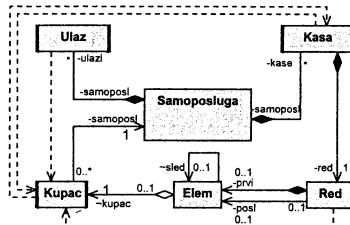
Zadatak 14 Samoposluga

Za model samoposluge iz zadatka 5 nacrtati na jeziku UML:

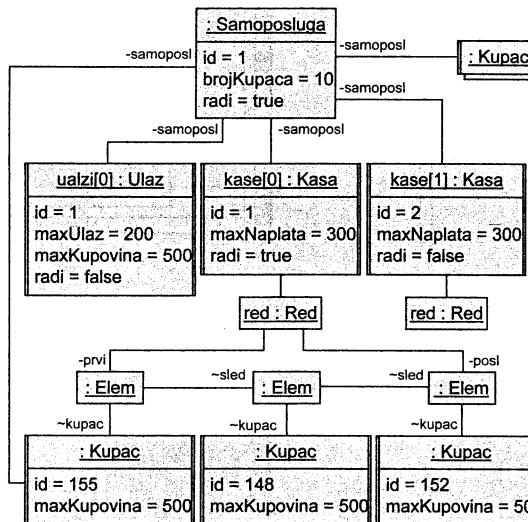
- dijagram objekata koji prikazuje samoposlugu s jednim ulazom, dve kase, nekoliko kupaca od kojih neki stoje u redu ispred jedne od kasa;
- dijagrame interakcije (sekvencije i komunikacije) koji prikazuju:
 - stvaranje samoposluge,
 - otvaranje samoposluge,
 - zatvaranje samoposluge,
 - uništavanje samoposluge,
 - životni vek kupca.

Rešenje:

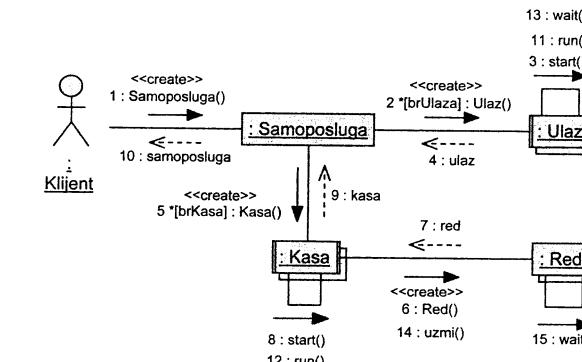
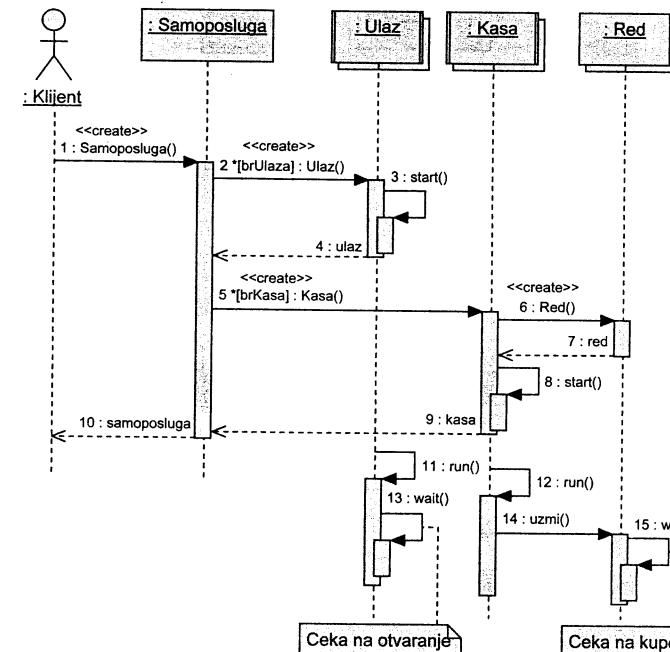
a) Ponovljeni dijagram klasa



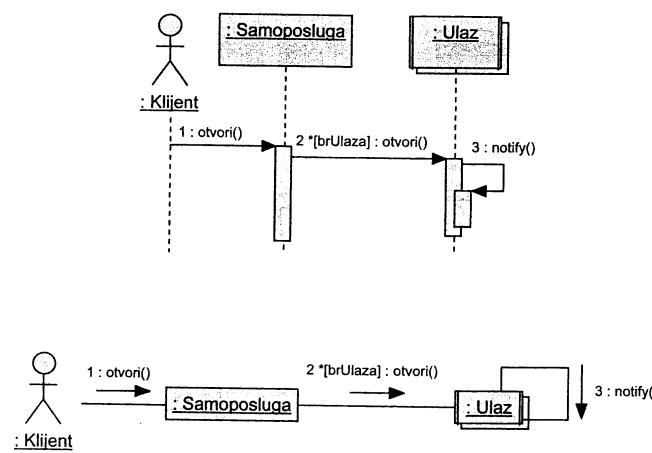
b) Dijagram objekata



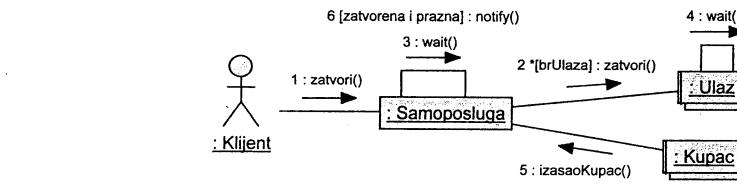
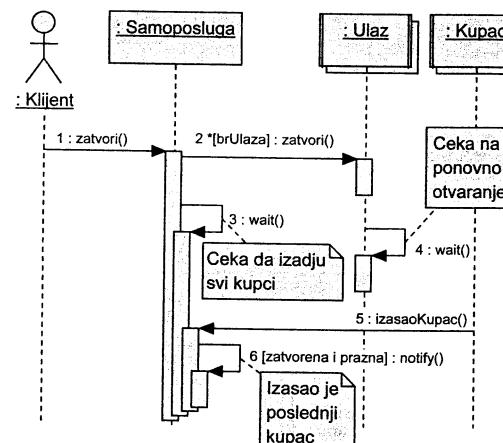
c) Dijagrami interakcije stvaranja samoposluge



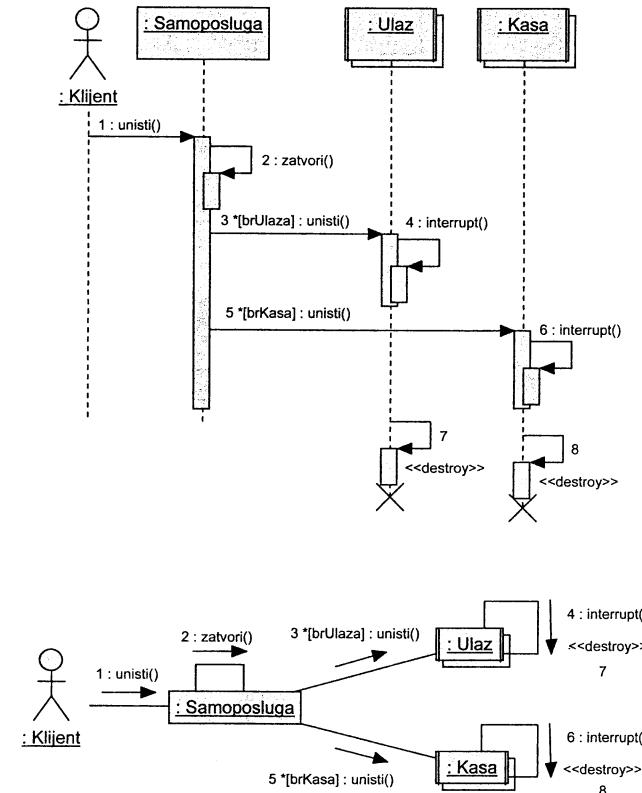
d) Dijagrami interakcije otvaranja samoposluge



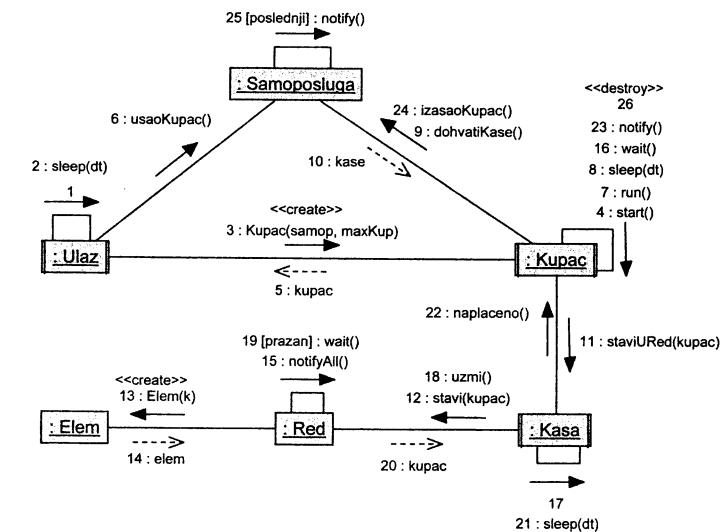
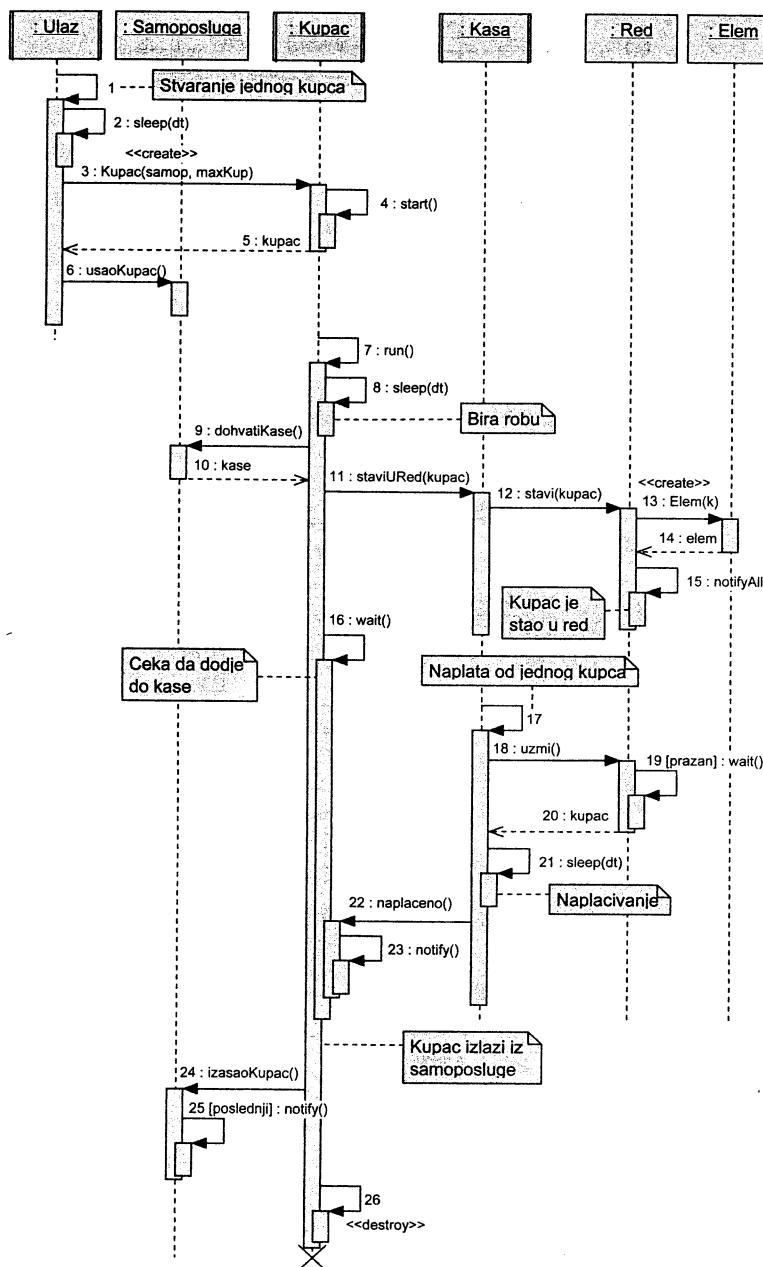
e) Dijagrami interakcije zatvaranja samoposluge



f) Dijagrami interakcije uništavanja samoposluge



g) Dijagrami interakcije životnog veka kupca



Zadatak 15 Distributeri, klijenti, pošiljke i raspodele (projektni uzorak **Posmatrač**)

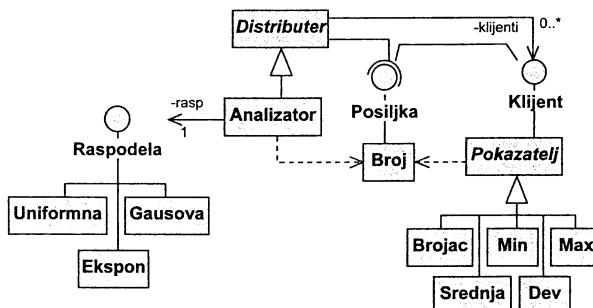
Apstraktan distributer može da isporučuje apstraktnu pošiljku svakom apstraktnom klijentu koji mu se prijavljuje radi prijema pošiljki. Klijenti mogu da se odjavljuju ako više nisu zainteresovani za prijem pošiljki. Apstraktan generator slučajnih brojeva na svaki zahtev daje jedan pseudoslučajan realan broj po nekoj raspodeli. Moguće raspodele su: uniformna, Gausova i eksponencijalna. Analizator je distributer koji na zahtev generiše zadati broj pseudoslučajnih brojeva. Te brojeve isporučuje, jedan po jedan, svim apstraktnim statističkim pokazateljima koji predstavljaju njegove klijente. Raspodela generisanih brojeva može da se postavlja po želji. Apstraktan statistički pokazatelj je klijent koji obrazuje neki statistički pokazatelj primljenih brojeva u pošiljkama. Može da se dovodi u početno stanje radi početka nove analize i da se dohvati rezultat analize primljenih brojeva do momenta zahteva. Mogući statistički pokazatelji su: broj uzorka, najmanja i najveća vrednost među uzorcima, srednja vrednost i standardna devijacija uzorka.

Projektovati na jeziku UML i napisati na jeziku Java prethodni sistem. Priložiti:

- dijagrame klasa,
- dijagram sekvence rada analizatora,
- prikaz korišćenih projektnih uzoraka,
- programsku realizaciju.

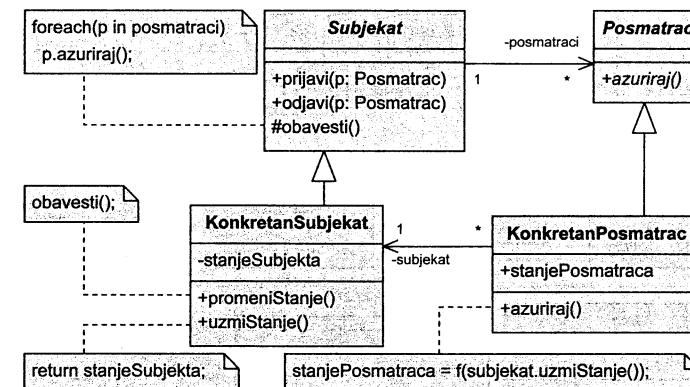
Rešenje:

a) Dijagram klasa



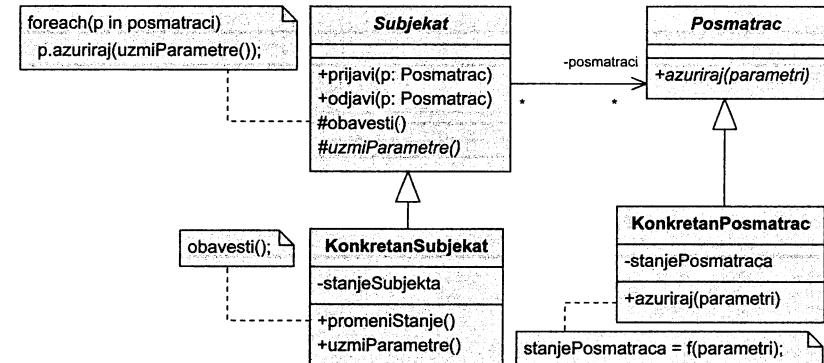
b) Projektni uzorak **Posmatrač** (*Observer*)

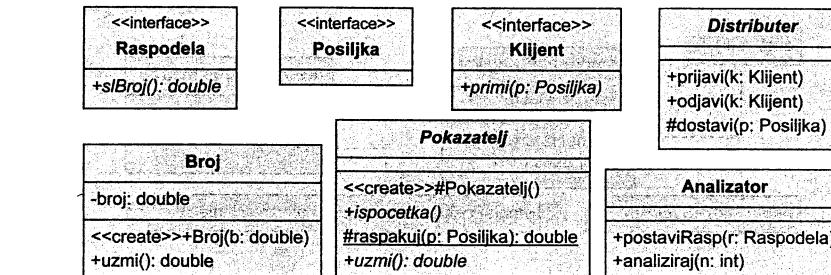
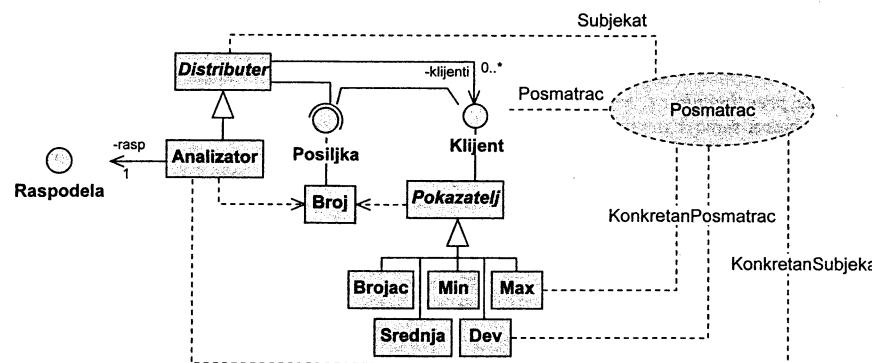
- objektni uzorak ponašanja
- distribucija informacija većem broju nezavisnih zainteresovanih klijenata
- subjekat obaveštava svoje posmatrače o promeni svog stanja da bi i posmatrači ažurirali svoja stanja
- model **preuzimanja**:
 - subjekat samo signalizira da je došlo do promene stanja, posle čega posmatrači treba da traže podatke o toj promeni
 - mana: posmatrači treba da znaju za detalje subjekta da bi znali kako i koje podatke mogu da traže
 - prednost: lako uvođenje novih vrsta posmatrača sa novim potrebama



• model **isporučivanja**:

- subjekat dostavlja podatke o promeni svog stanja
- prednost: posmatrači ne treba da znaju ništa o detaljima subjekta
- mana: subjekat treba da zna za potrebe svojih posmatrača i teško se prilagođava pojavi novih potreba



c) Primer uzorka *Posmatrač* (model isporučivanja)

```
// Raspodela.java
public interface Raspodela {
    public double slBroj();
}
```

```
// Posiljka.java
public interface Posiljka {}
```

```
// Klijent.java
public interface Klijent {
    public void primi(Posiljka p);
}
```

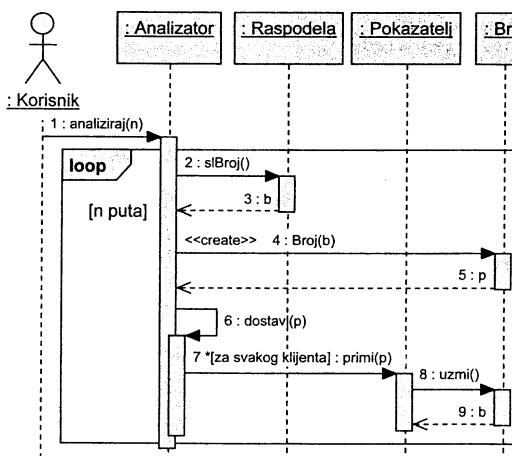
```
// Distributer.java
public abstract class Distributer {
    private class Elem {
        Klijent klijent; Elelem sled;
        Elelem(klijent k) {
            klijent = k; sled = null;
            if (prvi == null) prvi = this; else posl.sled = this;
            posl = this;
        }
    }
    private Elelem prvi, posl;
    public void prijavi(Klijent k) { new Elelem(k); }
    public void odjavi(Klijent k) {
        Elelem tek = prvi, pret = null;
        while (tek!=null & tek.klijent!=k) { pret = tek; tek = tek.sled; }
        if (tek != null) {
            if (pret == null) prvi = tek.sled; else pret.sled = tek.sled;
            if (tek == posl) posl = pret;
        }
    }
    protected void dostavi(Posiljka p) {
        for (Elelem tek=prvi; tek!=null; tek=tek.sled) tek.klijent.primi(p);
    }
}
```

```
// Broj.java
public class Broj implements Posiljka {
    private double broj;
    public Broj(double b) { broj = b; }
    public double uzmi() { return broj; }
}
```

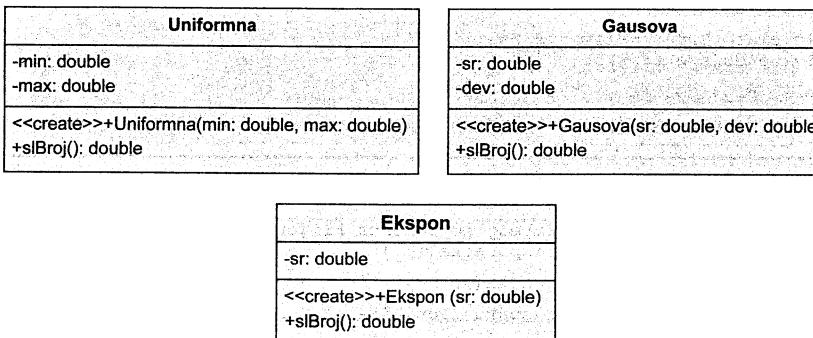
```
// Pokazatelj.java
public abstract class Pokazatelj implements Klijent {
    protected Pokazatelj() { ispocetka(); }
    public abstract void ispocetka();
    protected static double raspakuj(Posiljka p) {
        return ((Broj)p).uzmi();
    }
    public abstract double uzmi();
}
```

```
// Analizator.java
public class Analizator extends Distributer {
    private Raspodela rasp;
    public void postaviRasp(Raspodela r) { rasp = r; }
    public void analiziraj(int n) {
        for (int i=0; i<n; i++)
            dostavi(new Broj(rasp.slBroj()));
    }
}
```

d) Dijagram sekvencije rada analizatora



e) Ostale klase



```

// Uniformna.java
public class Uniformna implements Raspodela {
    private double min;
    private double max;

    public Uniformna(double min, double max) {
        this.min = min; this.max = max;
    }

    public double slBroj() { return min + (max - min) * Math.random(); }
}

```

```

// Gausova.java
public class Gausova implements Raspodela {
    private double sr;
    private double dev;

    public Gausova(double sr, double dev) { this.sr = sr; this.dev = dev; }

    public double slBroj() {
        double s = 0;
        for (int i=0; i<12; i++) s += Math.random();
        return (s - 6) * dev + sr;
    }
}

```

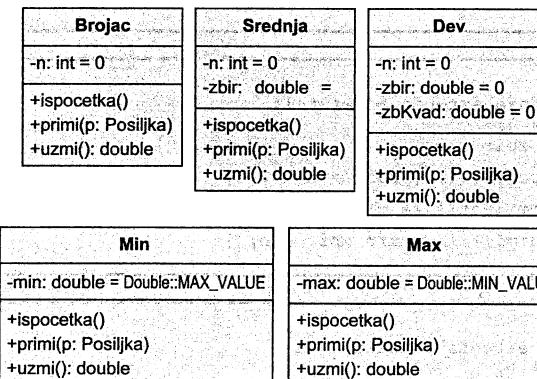
```

// Ekspon.java
public class Ekspon implements Raspodela {
    private double sr;

    public Ekspon(double sr) { this.sr = sr; }

    public double slBroj() { return - sr * Math.log (1 - Math.random()); }
}

```



```

// Brojac.java
public class Brojac extends Pokazatelj {
    private int n = 0;

    public void ispocetka() { n = 0; }

    public void primi(Posiljka p) { n++; }

    public double uzmi() { return n; }
}

```

```
// Min.java
public class Min extends Pokazatelj {
    private double min = Double.MAX_VALUE;
    public void ispocetka() { min = Double.MAX_VALUE; }
    public void primi(Posiljka p) {
        double b = raspakuj(p);
        if (b < min) min = b;
    }
    public double uzmi() { return min; }
}
```

```
// Max.java
public class Max extends Pokazatelj {
    private double max = Double.MIN_VALUE;
    public void ispocetka() { max = Double.MIN_VALUE; }
    public void primi(Posiljka p) {
        double b = raspakuj(p);
        if (b > max) max = b;
    }
    public double uzmi() { return max; }
}
```

```
// Srednja.java
public class Srednja extends Pokazatelj {
    private int n = 0;
    private double zbir = 0;
    public void ispocetka() { zbir = n = 0; }
    public void primi(Posiljka p) { n++; zbir += raspakuj(p); }
    public double uzmi() { return zbir / n; }
}
```

```
// Dev.java
public class Dev extends Pokazatelj {
    private int n = 0;
    private double zbir = 0;
    private double zbKvad = 0;
    public void ispocetka() { zbKvad = zbir = n = 0; }
    public void primi(Posiljka p) {
        double b = raspakuj(p);
        n++; zbir += b; zbKvad += b * b;
    }
    public double uzmi() {
        double sr = zbir / n;
        return Math.sqrt(zbKvad/n - sr * sr);
    }
}
```

f) Program za ispitivanje klase statističkih pokazatelja

```
// Test.java
public class Test {
    private static Pokazatelj[] pok = {
        new Brojac(), new Min(), new Max(),
        new Srednja(), new Dev()
    };
    private static String[] vrs = { "n", "min", "max", "sr", "dev" };
    public static void main(String[] varg) {
        Analizator analiz = new Analizator();
        for (int i=0; i<pok.length; i++) analiz.prijavi(pok[i]);
        analiz.postaviRasp(new Uniformna(-10, 10));
        analiz.analiziraj(10000);
        for (int i=0; i<pok.length; i++)
            System.out.printf("%3s = %g\n", vrs[i], pok[i].uzmi());
    }
}
```

```
n = 10000.0
min = -9.99998
max = 9.99790
sr = 0.0170178
dev = 5.78494
```

Zadatak 16 Boja, tačka, figure i iteratori (projektni uzorak *Iterator*)

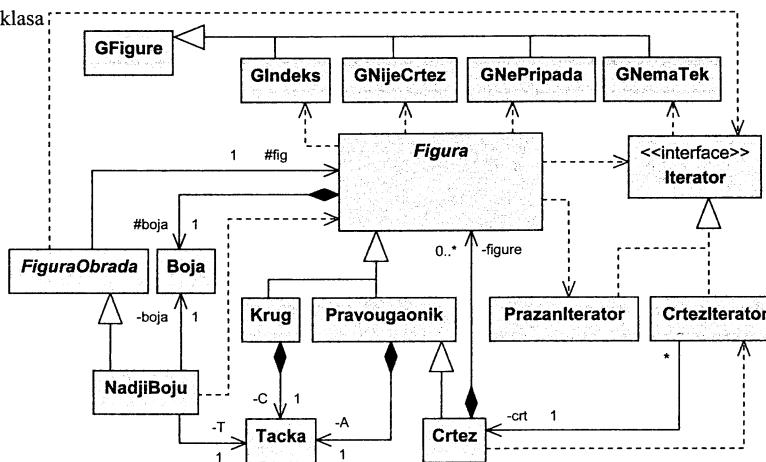
Boja se zadaje celobrojnim komponentama crvene, zelene i plave boje u opsegu od 0 do 255. Tačka u ravni se zadaje realnim koordinatama i može da se izračuna rastojanje do zadate druge tačke. Popunjena apstraktna figura u ravni ima jedinstven, automatski generisan identifikacioni broj. Može da se napravi kopija figure, da se dohvati boja podloge figure i boja zadate tačke unutar figure. Krug i pravougaonik su figure kod kojih su sve tačke iste boje. Crtež je pravougaonik koji može da sadrži proizvoljan broj figura proizvoljne vrste. Boju tačke u crtežu određuje figura koja sadrži tu tačku i poslednja je dodata crtežu. Ako nema takve figure boja tačke jednaka je boji podloge. Sve vrste figura, bez obzira da li mogu da sadrže druge figure ili ne, obraduju se na isti način.

Projektovati na jeziku UML i napisati na jeziku Java prethodni sistem. Priložiti:

- dijagrame klase,
- prikaz korišćenih projektnih uzoraka,
- dijagram sekvence nalaženja boje tačke unutar figure,
- programsku realizaciju.

Rešenje:

a) Dijagram klasa:



b) Pomoćne klase

Boja
-c: byte
-z: byte
-p: byte
<<create>>+Boja(c: byte, z: byte, p: byte)
<<create>>+Boja()
+c(): byte
+z(): byte
+p(): byte

Tacka
-x: double
-y: double
<<create>>+Tacka(x: double, y: double)
<<create>>+Tacka()
+x(): double
+y(): double
+d(T: Tacka): double

c) Projektni uzorak *Sastav*

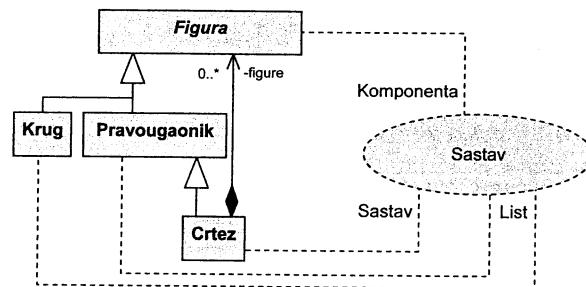


Figura
-ukld: int = 0
-id: int = ++ukld
<<create>>+Figura(b: Boja)
<<create>>+Figura()
+clone(): Object
+id(): int
+P(): double
+pripada(T: Tacka): boolean
+boja(): Boja
+boja(T: Tacka): Boja
+dodaj(f: Figura)
+dohvati(ind: int): Figura
+brisi(ind: int)
+brojFigura(): int
+iterator(): Iterator

Pravougaonik
#sir: double
#vis: double
<<create>>+Pravougaonik(buja: Boja, A: Tacka, sir: double, vis: double)
<<create>>+Pravougaonik(A: tacka, sir: double, vis: double)
<<create>>+Pravougaonik(buja: Boja)
<<create>>+Pravougaonik()
+A(): Tacka
+sir(): double
+vis(): double
+pripada(T: Tacka): boolean
+P(): double

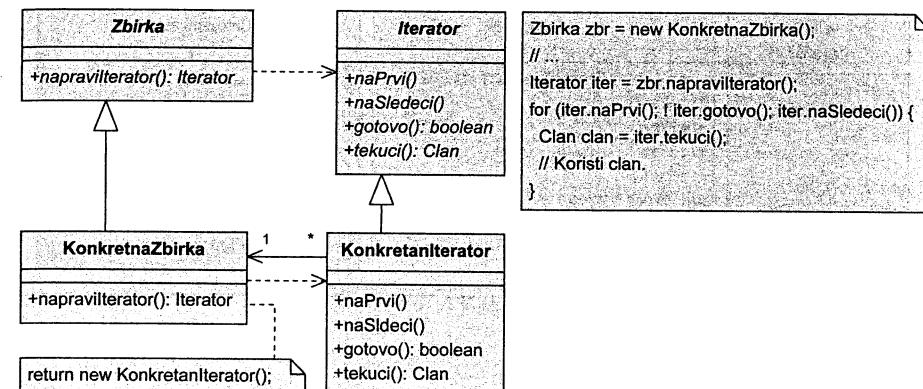
Krug
-r: double
<<create>>+Krug(buja: Boja, r: double, C: Tacka)
<<create>>+Krug(r: double, C: Tacka)
<<create>>+Krug(buja: Boja)
<<create>>+Krug()
+r(): double
+C(): Tacka
+pripada(T: Tacka): boolean
+P(): double

Crtez
-brojFigura: int
<<create>>+Crtez(buja: Boja, A: Tacka, sir: double, vis: double)
<<create>>+Crtez(A: Tacka, sir: double, vis: double)
<<create>>+Crtez(buja: Boja)
<<create>>+Crtez()
+clone(): Object
+boja(T: Tacka): Boja
+dodaj(fig: Figura)
+dohvati(ind: int): Figura
+brisi(ind: int)
+brojFigura(): int
+iterator(): Iterator

```
// Figura.java - Apstraktna klasa figura.
public abstract class Figura implements Cloneable {
    private static int ukId = 0;
    private int id = ++ukId;
    protected Boja boja;
    public Figura(Boja b) { boja = b; }
    public Figura() { boja = new Boja(); }
    public Object clone () {
        try { return super.clone(); }
        catch (CloneNotSupportedException g) { return null; }
    }
    public int id() { return id; }
    public abstract double P();
    public abstract boolean pripada(Tacka T);
    public Boja boja() { return boja; }
    public Boja boja(Tacka T) throws GNePripada {
        if (! pripada(T)) throw new GNePripada();
        return boja;
    }
    // Podrazumevane radnje za proste figure.
    public void dodaj(Figura f) throws GNijeCrtez {
        throw new GNijeCrtez();
    }
    public Figura dohvati(int ind) throws GNijeCrtez, GIindeks {
        throw new GNijeCrtez();
    }
    public void brisi(int ind) throws GNijeCrtez, GIindeks {
        throw new GNijeCrtez();
    }
    public int brojFigura() { return 0; }
    public Iterator iterator()
        { return new PrazanIterator(this); }
}
```

d) Projektni uzorak **Iterator** (*Iterator*)

- objektni uzorak ponašanja
- sekvensijalan obilazak članova zbirke na uniforman način
- iterator sprovodi strategiju obilaska zbirke i dohvata u zastopne članove
 - zbirka treba da podržava rad iteratora
 - iterator je često prijateljska ili čak unutrašnja klasa zbirke
- u isto vreme može više obilazaka biti u toku nad istom zbirkom čak i po različitim kriterijumima

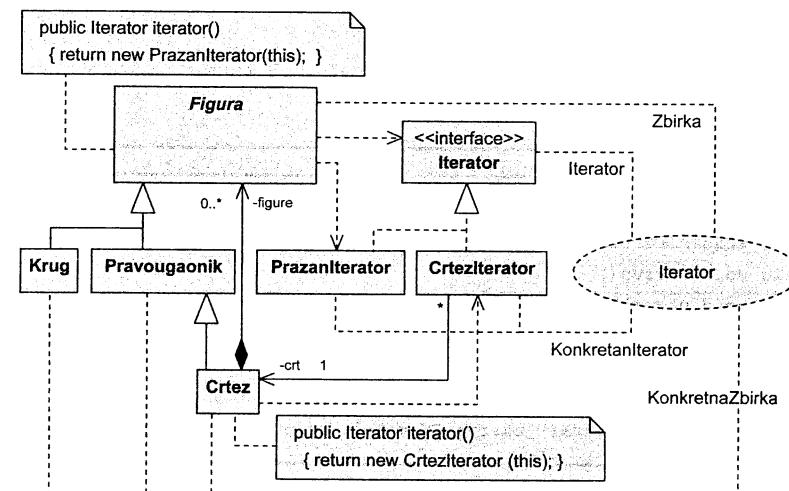


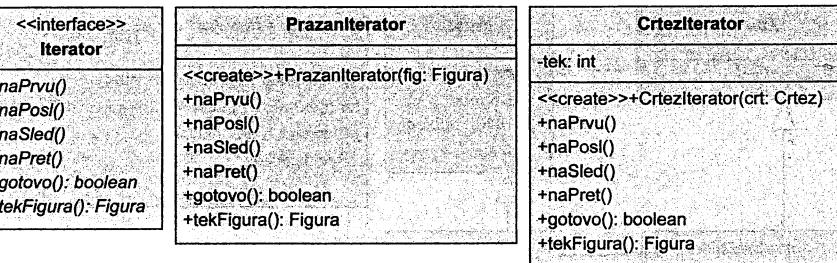
```

Zbirka zbr = new KonkretnaZbirka();
// ...
Iterator iter = zbr.napraviliterator();
for (iter.naPrvi(); !iter.gotovo(); iter.naSledec()) {
    Clan clan = iter.tekuci();
    // Koristi clan.
}
  
```

- **prazan iterator**
 - ne radi ništa i uvek se smatra završenim
 - obezbeđuje ujednačenu obradu listova i sastava u uzorku *Sastav*
- **spoljašnji iterator**
 - tok iteracija kontroliše korisnik
- **unutrašnji iterator**
 - kontroliše tok iteracija
 - na svakog člana zbirke primenjuje određenu radnju
 - može da koristi neki spoljašnji iterator
 - može da vrši selekciju među elementima
 - tok iteracija može da se pre vremena prekine na zahtev korisnika

e) Primer spoljašnjeg iteratora





```

// Iterator.java
public interface Iterator {
    public void naPruv();
    public void naPosl();
    public void naSled();
    public void naPret();
    public boolean gotovo();
    public Figura tekFigura() throws GNemaTek;
}
  
```

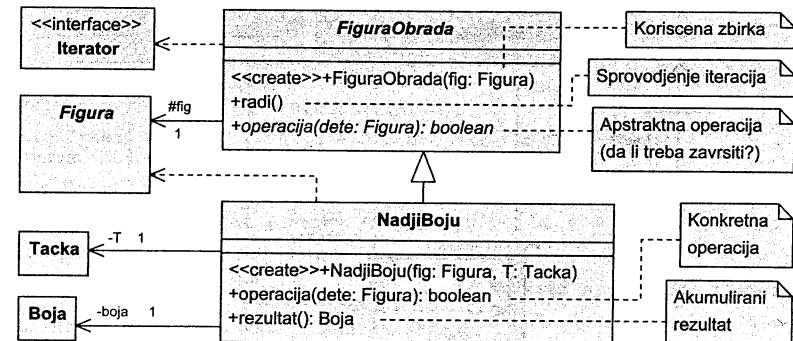
```

// CrtezIterator.java
public class CrtezIterator implements Iterator {
    private int tek;
    private Crtez crt;
    public CrtezIterator(Crtez crt) { this.crt = crt; tek = 0; }
    public void naPruv() { tek = 0; }
    public void naPosl() { tek = crt.brojFigura()-1; }
    public void naSled() { tek++; }
    public void naPret() { tek--; }
    public boolean gotovo() { return tek<0 || tek>=crt.brojFigura(); }
    public Figura tekFigura() throws GNemaTek {
        if (gotovo()) throw new GNemaTek();
        try { return crt.dohvati (tek); }
        catch (GIndeks g) { return null; }
    }
}
  
```

```

// PrazanIterator.java
public class PrazanIterator implements Iterator {
    public PrazanIterator(Figura fig) {}
    public void naPruv() {}
    public void naPosl() {}
    public void naSled() {}
    public void naPret() {}
    public boolean gotovo() { return true; }
    public Figura tekFigura() throws GNemaTek { throw new GNemaTek(); }
}
  
```

f) Primer unutrašnjeg iteratora



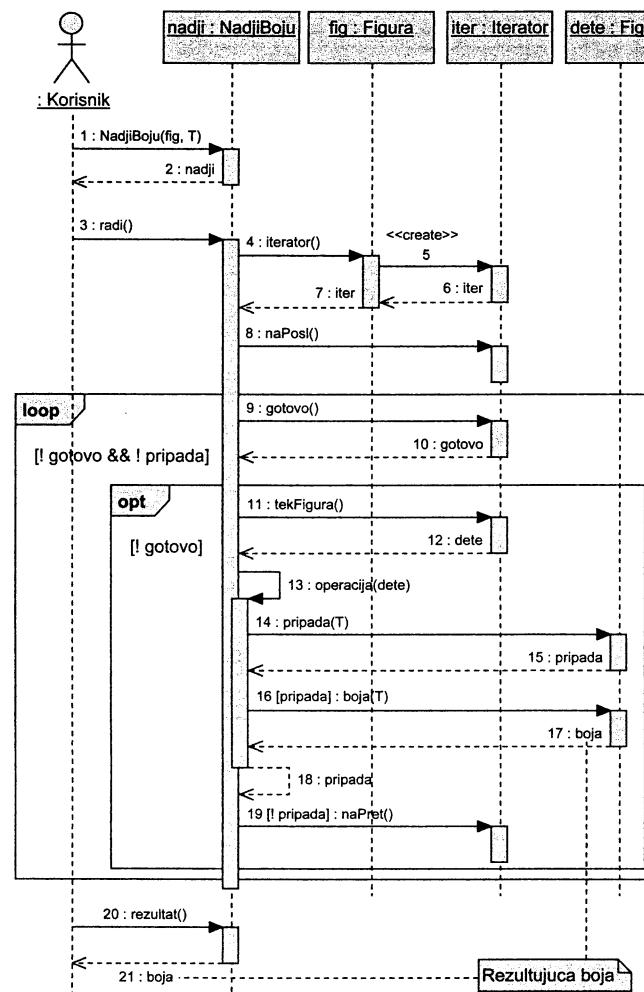
```

// FiguraObrada.java
public abstract class FiguraObrada {
    protected Figura fig;
    public FiguraObrada(Figura fig) { this.fig = fig; }
    public void radi() {
        Iterator iter = fig.iterator();
        for (iter.naPosl(); !iter.gotovo(); iter.naPret())
            try { if (operacija(iter.tekFigura())) break; }
            catch (GNemaTek g) {}
    }
    protected abstract boolean operacija(Figura dete);
}
  
```

```

// NadjiBoju.java
public class NadjiBoju extends FiguraObrada {
    private Boja boja;
    private Tacka T;
    public NadjiBoju(Figura fig, Tacka T)
    { super(fig); this.T = T; }
    public boolean operacija(Figura dete) {
        if (dete.pripada(T)) {
            try { boja = dete.boja(T); } catch(GNePripada g) {}
            return true;
        } else return false;
    }
    public Boja rezultat() { return boja; }
}
  
```

g) Dijagram sekvence određivanja boje tačke



h) Upotreba iteratora

```

// Crtez.java

public class Crtez extends Pravougaonik {
    private int brojFigura;
    private Figura[] figure = new Figura [5];
    public Crtez(Boja boja, Tacka A, double sir, double vis)
        { super(boja, A, sir, vis); }
    public Crtez(Tacka A, double sir, double vis) { super (A, sir, vis); }
    public Crtez(Boja boja) { super (boja); }
    public Crtez() {}

    public Object clone () { // Koristi spoljašnji iterator.
        Crtez crt = (Crtez) super.clone();
        Iterator iter = new CrtezIterator(this);
        for (iter.naPrvu(); !iter.gotovo(); iter.naSled())
            try { crt.dodaj((Figura)iter.tekFigura().clone()); }
            catch (GNemaTek g) {}
        return crt;
    }

    public Boja boja(Tacka T) throws GNePripada { // Koristi unutrašnji iterator.
        if (! pripada(T)) throw new GNePripada(); // iterator.
        Tacka T2 = new Tacka(T.x()-A.x(), T.y()-A.y());
        NadjiBoju nadji = new NadjiBoju(this, T2);
        nadji.radi ();
        if (nadji.rezultat() != null) return nadji.rezultat();
        else return boja;
    }

    public void dodaj(Figura fig) {
        if (brojFigura == figure.length) {
            Figura[] nove = new Figura [figure.length+5];
            for (int i=0; i<brojFigura; nove[i]=figure[i++]);
            figure = nove;
        }
        figure[brojFigura++] = fig;
    }

    public Figura dohvati(int ind) throws GIindeks {
        if (ind<0 || ind>=brojFigura) throw new GIindeks();
        return figure[ind];
    }

    public void brisi(int ind) throws GIindeks {
        if (ind<0 || ind>=brojFigura) throw new GIindeks();
        for (int i=ind; i<brojFigura-1; figure[i]=figure[+i]);
        brojFigura--;
        if (figure.length-brojFigura > 5) {
            Figura[] nove = new Figura [figure.length-5];
            for (int i=0; i<brojFigura; nove[i]=figure[i++]);
            figure = nove;
        }
    }

    public int brojFigura() { return brojFigura; }
    public Iterator iterator() { return new CrtezIterator (this); }
}
  
```

i) Ostale klase

```
// Boja.java - Klasa boja.
public class Boja {
    private byte c; private byte z; private byte p;
    public Boja(byte c, byte z, byte p) {this.c = c; this.z = z; this.p = p;}
    public Boja() { c = z = p = (byte)255; }
    public byte c() { return c; }
    public byte z() { return z; }
    public byte p() { return p; }
}
```

```
// Tacka.java - Klasa tacaka.
public class Tacka {
    private double x; private double y;
    public Tacka(double x, double y) { this.x = x; this.y = y; }
    public Tacka() { x = y = 0; }
    public double x() { return x; }
    public double y() { return y; }
    public double d(Tacka T) {
        return Math.sqrt(Math.pow(x-T.x,2) + Math.pow(y-T.y,2));
    }
}
```

```
// Krug.java - Klasa krugova.
public class Krug extends Figura {
    private double r; private Tacka C;
    public Krug(Boja boja, double r, Tacka C)
        { super (boja); this.r = r; this.C = C; }
    public Krug(double r, Tacka C) { this.r = r; this.C = C; }
    public Krug(Boja boja) { this (boja, 1, new Tacka()); }
    public Krug() { this (new Boja()); }
    public double r() { return r; }
    public Tacka C() { return C; }
    public boolean pripada(Tacka T) { return C.d(T) <= r; }
    public double P() { return 4 * r * r * Math.PI; }
}
```

```
// Pravougaonik.java
public class Pravougaonik extends Figura {
    protected double sir; protected double vis; private Tacka A;
    public Pravougaonik(Boja boja, Tacka A, double sir, double vis) {
        super(boja); this.A = A; this.sir = sir; this.vis = vis;
    }
    public Pravougaonik(Tacka A, double sir, double vis)
        { this(new Boja(), A, sir, vis); }
    public Pravougaonik(Boja boja) { this (boja, new Tacka(), 1, 1); }
    public Pravougaonik() { this(new Boja()); }
    public Tacka A() { return A; }
    public double sir() { return sir; }
    public double vis() { return vis; }
    public boolean pripada(Tacka T) {
        return A.x()<=T.x() && T.x()<=A.x()+sir &&
               A.y()<=T.y() && T.y()<=A.y()+vis;
    }
    public double P() { return sir * vis; }
}
```

```
// GFigure.java
public class GFigure extends Exception {}
```

```
// GIindeks.java
public class GIindeks extends GFigure {}
```

```
// GNepripada.java
public class GNepripada extends GFigure {}
```

```
// GNematek.java
public class GNematek extends GFigure {}
```

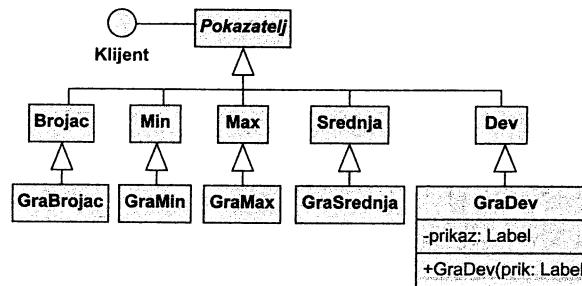
```
// GNijeCrtez.java
public class GNijeCrtez extends GFigure {}
```

Zadatak 17 Grafički statistički pokazatelji (projektni uzorak *Dopuna*)

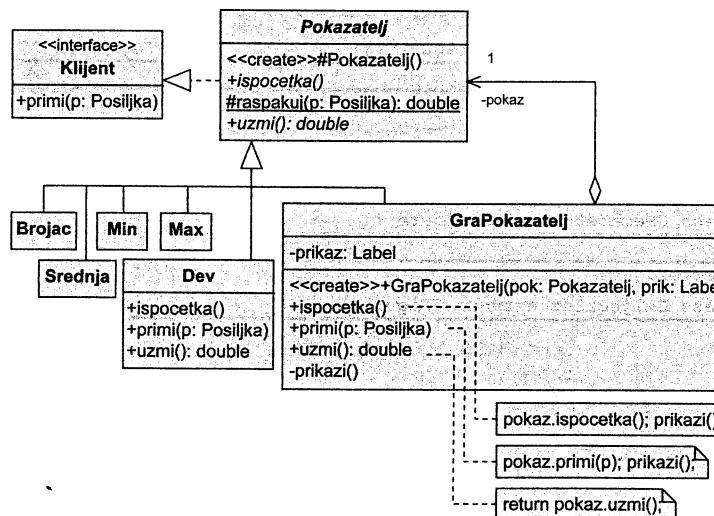
Ospozibiti statističke pokazatelje iz zadatka 14 za prikazivanje svojih stanja na grafičkoj korisničkoj površi prilikom svake promene stanja.

Rešenje:

a) Loše rešenje

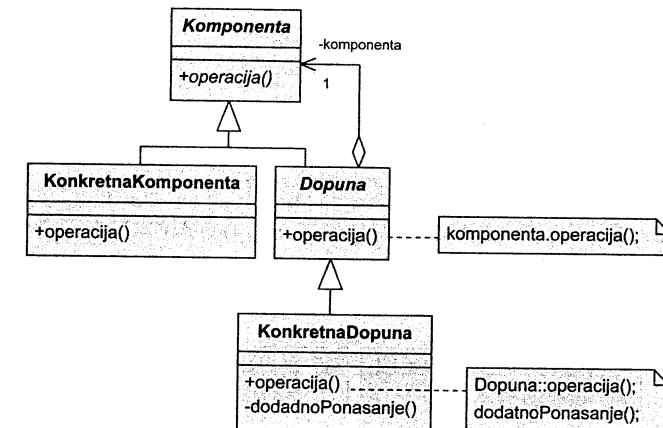


b) Dijagram klasa statističkih pokazatelja

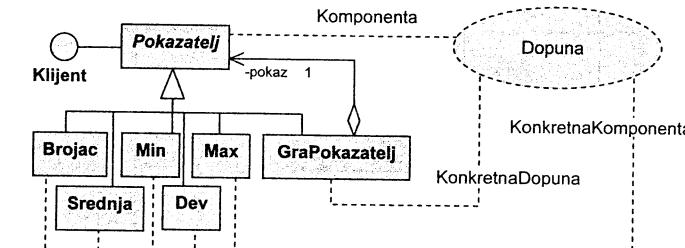


c) Projektni uzorak *Dopuna* (*Decorator*)

- objektni uzorak strukture
- dinamičko dodavanje ponašanja objektu
- dopuna prosledjuje zahteve dopunjenoj objektu i izvršava dodatne radnje pre ili posle prosledjivanja zahteva

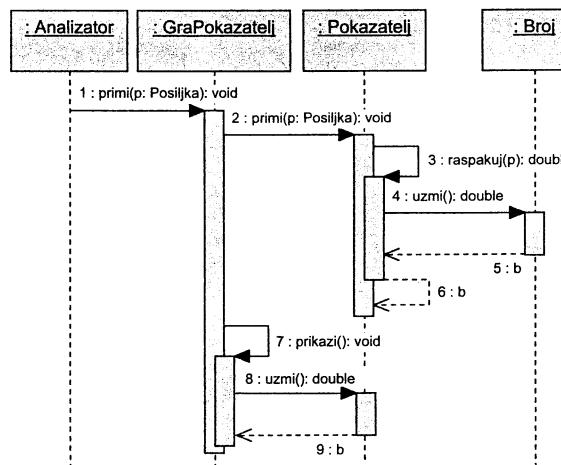


d) Primer uzorka *Dopuna*



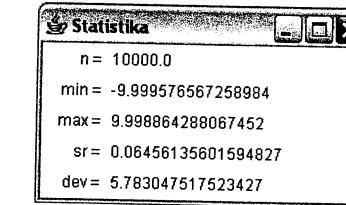
```
// GraPokazatelj.java
import java.awt.Label;
public class GraPokazatelj extends Pokazatelj {
    private Pokazatelj pokaz;
    private Label prikaz;
    public GraPokazatelj(Pokazatelj pok, Label prik) {
        pokaz = pok; prikaz = prik;
    }
    public void ispocetka() {
        if (pokaz != null) { pokaz.ispocetka(); prikazi(); }
    }
    public void primi(Posiljka p) {
        pokaz.primi(p); prikazi();
    }
    public double uzmi() { return pokaz.uzmi(); }
    private void prikazi () {
        prikaz.setText(Double.toString(pokaz.uzmi()));
    }
}
```

e) Dijagram sekvence rada grafičkog statističkog pokazatelja



f) Program za ispitivanje klasa grafičkih statističkih pokazatelja

```
// Test.java
import java.awt.*;
import java.awt.event.*;
public class Test extends Frame {
    private Pokazatelj[] pok = { new Brojac(), new Min(), new Max(),
        new Srednja(), new Dev() };
    private String[] vrs = { "n", "min", "max", "sr", "dev" };
    private Analizator analiz = new Analizator();
    analiz.postaviRasp(new Uniformna(-10, 10));
    private Test () {
        super ("Statistika");
        setSize (250, 150);
        addWindowListener (new WindowAdapter() {
            public void windowClosing(WindowEvent d) { dispose(); }
        });
        popuniProzor();
        setVisible(true);
        analiz.analiziraj(10000);
    }
    private void popuniProzor() {
        Panel p1 = new Panel(new GridLayout(0,1)); add(p1, "West");
        Panel p2 = new Panel(new GridLayout(0,1)); add(p2, "Center");
        for (int i=0; i<pok.length; i++) {
            p1.add(new Label(vrs[i] + " = ", Label.RIGHT));
            Label ozn = new Label(); p2.add(ozn);
            analiz.prijavi(new GraPokazatelj(pok[i], ozn));
        }
    }
    public static void main(String[] varg) { new Test(); }
}
```



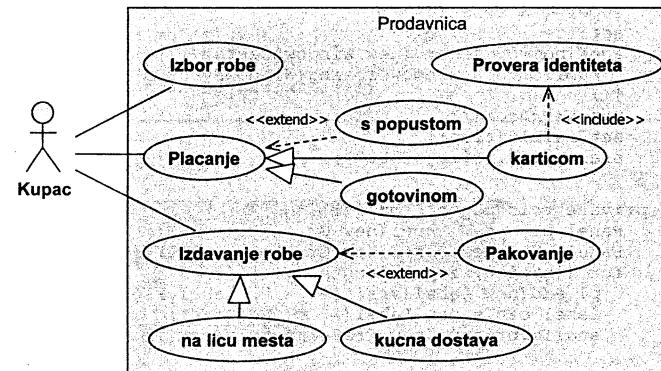
Zadatak 18 Prodavnica, fakultet i pošta (dijagram slučajeva korišćenja)

Nacrtati na jeziku UML dijagrame slučajeva korišćenja za:

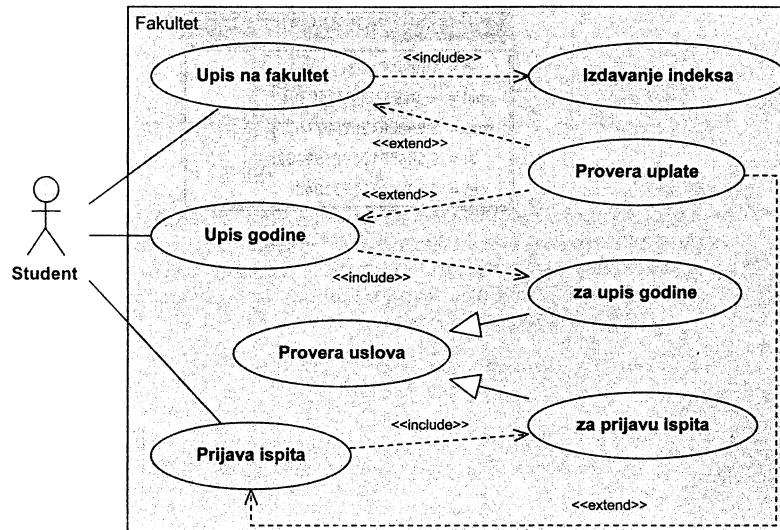
- kupovinu u prodavnici,
- rad studentske službe,
- rad poštanskog šaltera.

Rešenje:

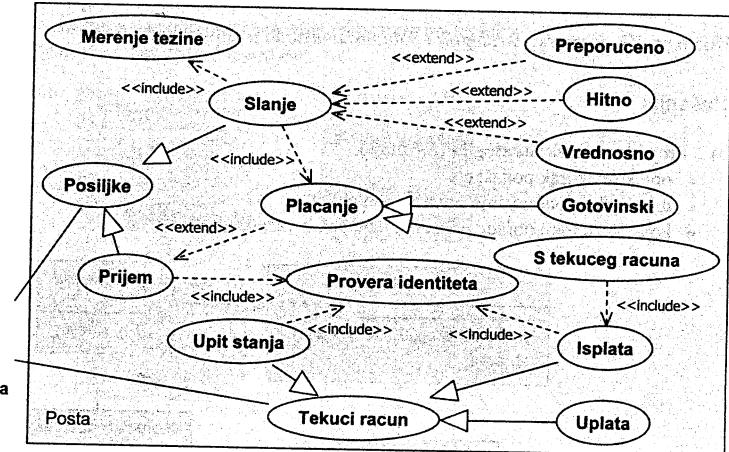
a) Kupovina u prodavnici



b) Rad studentske službe



c) Rad poštanskog šaltera



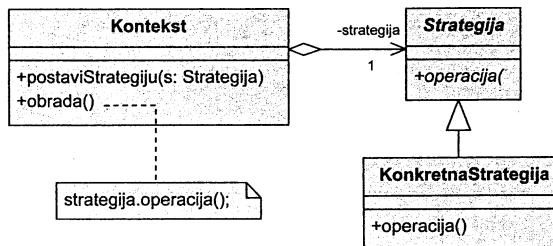
Zadatak 19 Projektni uzorci *Strategija* i *Šablonska metoda*

Uočiti projektnе uzorce *Strategija* i *Šablonska metoda* u ranijim zadacima.

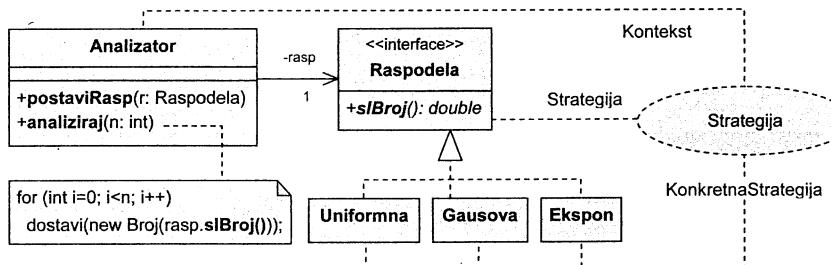
Rešenje:

a) Projektni uzorak *Strategija* (*Strategy*)

- objektni uzorak ponašanja
- dinamičko podešavanje ponašanja objekta
- kontekst u toku obrade primenjuje trenutno važeću strategiju

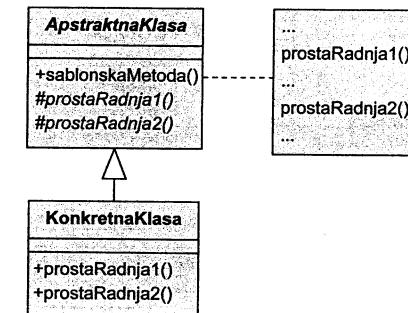


b) Primer uzorka *Strategija* u zadatu 14

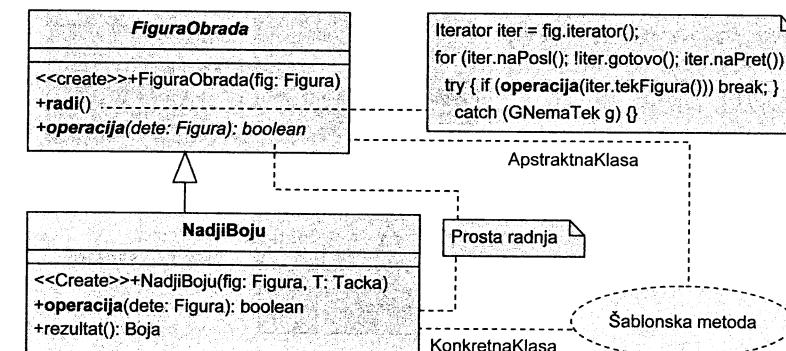


c) Projektni uzorak *Šablonska metoda* (*Template Method*)

- klasni uzorak ponašanja
- statičko određivanje varijacija na određenim mestima složenog postupka
- konkretna klasa ostvaruje korake apstraktne klase

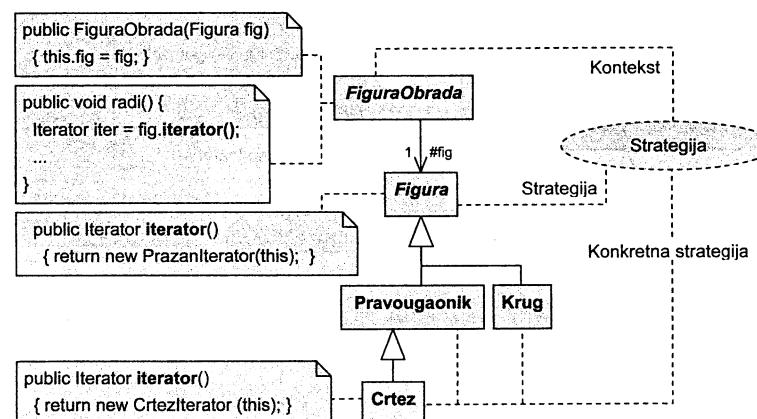
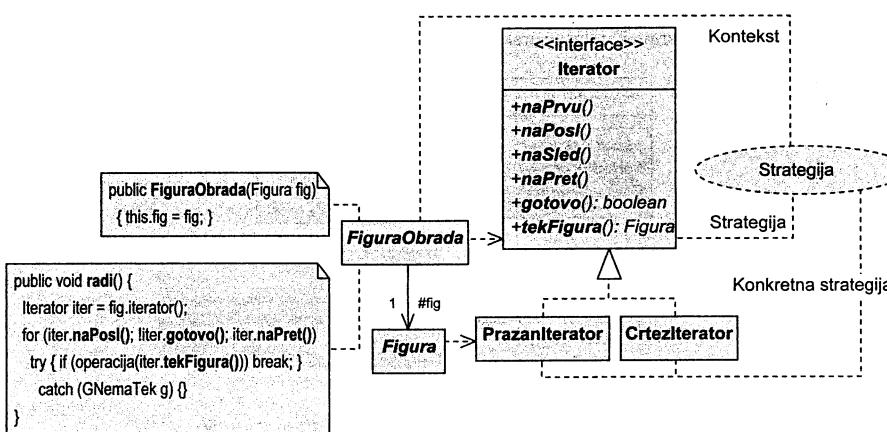
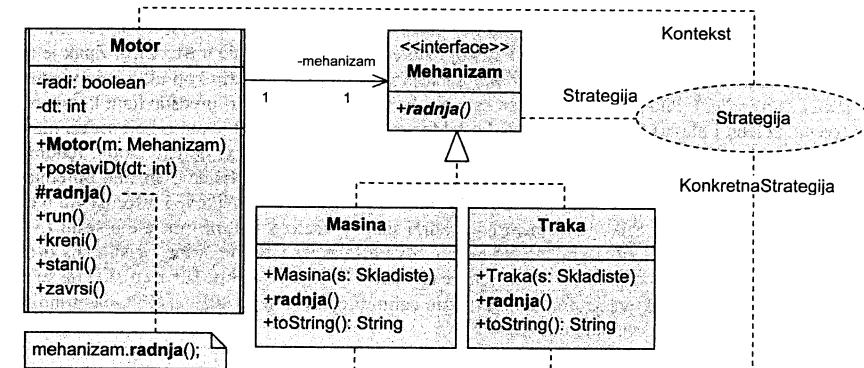
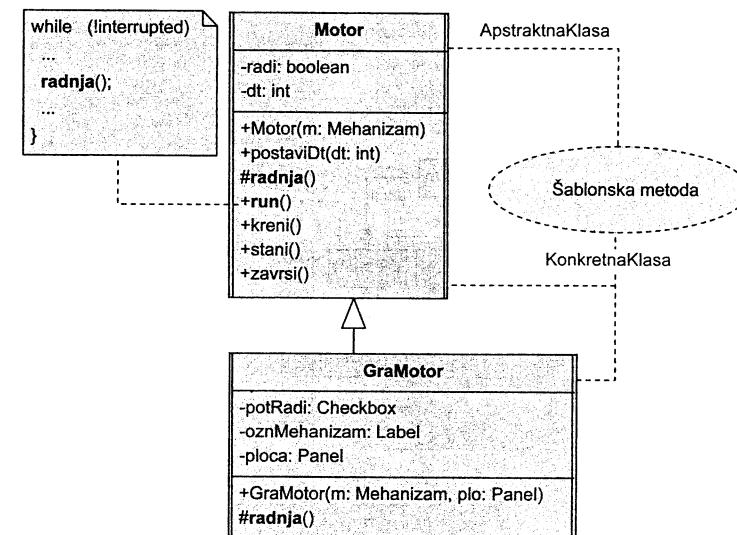


d) Primer uzorka *Šablonska metoda* u zadatu 15



e) Zaključak

- uzorci *Strategija* i *Šablonska metoda* ostvaruju razlike u ponašanju objekata
- *Strategija* omogućava dinamičko menjanje (za vreme izvršavanja programa) ponašanja objekata
 - dati objekat u različitim trenucima može da se ponaša različito
- *Šablonska metoda* statički određuje (za vreme pisanja klasa) razlike u ponašanju pojedinih potklasa
 - svi objekti iste klase uvek se ponašaju na isti način

f) Primeri uzorka *Strategija* u zadatku 15g) Primer uzorka *Strategija* u zadatku 4h) Primer uzorka *Šablonka metoda* u zadatku 4

Zadatak 20 Atomi, znak, piksel, tekst, slika i dokument {K, 01.12.2006.}

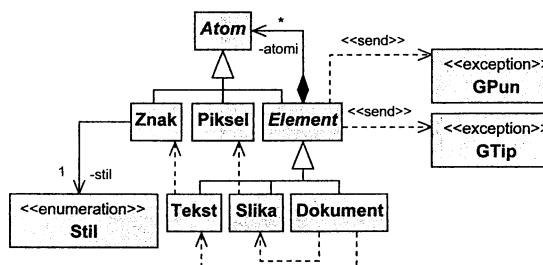
Svakom apstraktnom atomu može da se napravi kopija (klon), da se izračuna celobrojna veličina koja predstavlja broj bajtova koje bi atom zauzeo u nekoj datoteci i da se konvertuje u `String`. Znak je atom koji je opisan sa dva podatka: kodom znaka (`char`), i stilom znaka (nabrojivi tip koji uključuje vrednosti: *bold*, *italic* i *underline*). Piksel je atom opisan sa četiri bajta, od kojih prva tri predstavljaju komponente boje (crvenu, zelenu i plavu), a četvrti predstavlja faktor prozirnosti. Element je atom koji sadrži niz atoma ograničenog kapaciteta. Stvara se prazan zadatog kapacitet, a onda mu se dodaju atomi, redom posebnom metodom. Prekoračenje kapaciteta niza atoma izaziva izuzetak. Veličina elementa određuje se izračunavanjem veličina sastavnih atoma. Tekst je element koji sadrži samo znakove i informaciju o dužini sadržaja (kratak ceo broj). Slika je element koji sadrži samo piksele, kao i informacije o širini i visini slike, izražene u broju piksela (dva kratka cela broja). Stvara se prazna zadate širine i visine, na osnovu kojih se određuje kapacitet niza piksela. Dokument je element koji ima svoje ime (`String`) i čija se veličina meri zauzećem niza znakova sa dodatnim kratkim celim brojem za opis dužine. Dokument može da sadrži samo tekstove, slike i druge dokumente.

Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram sekvenце koji prikazuje scenario stvaranja jednog dokumenta sastavljenog od tekstova, slika i drugih dokumenata.

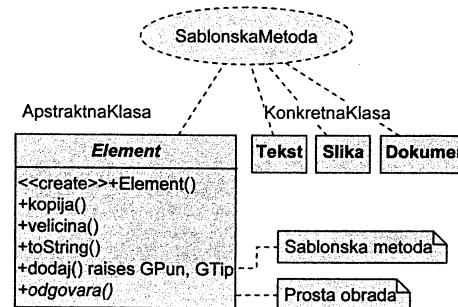
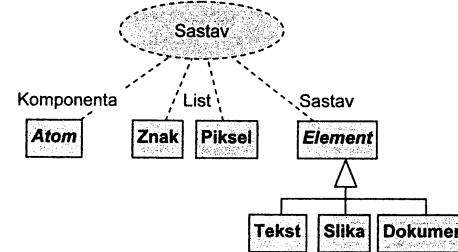
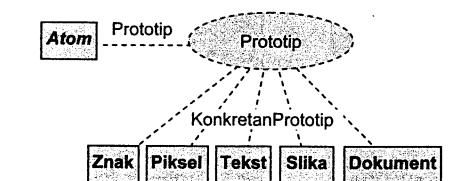
Rešenje:

a) Dijagram klasa

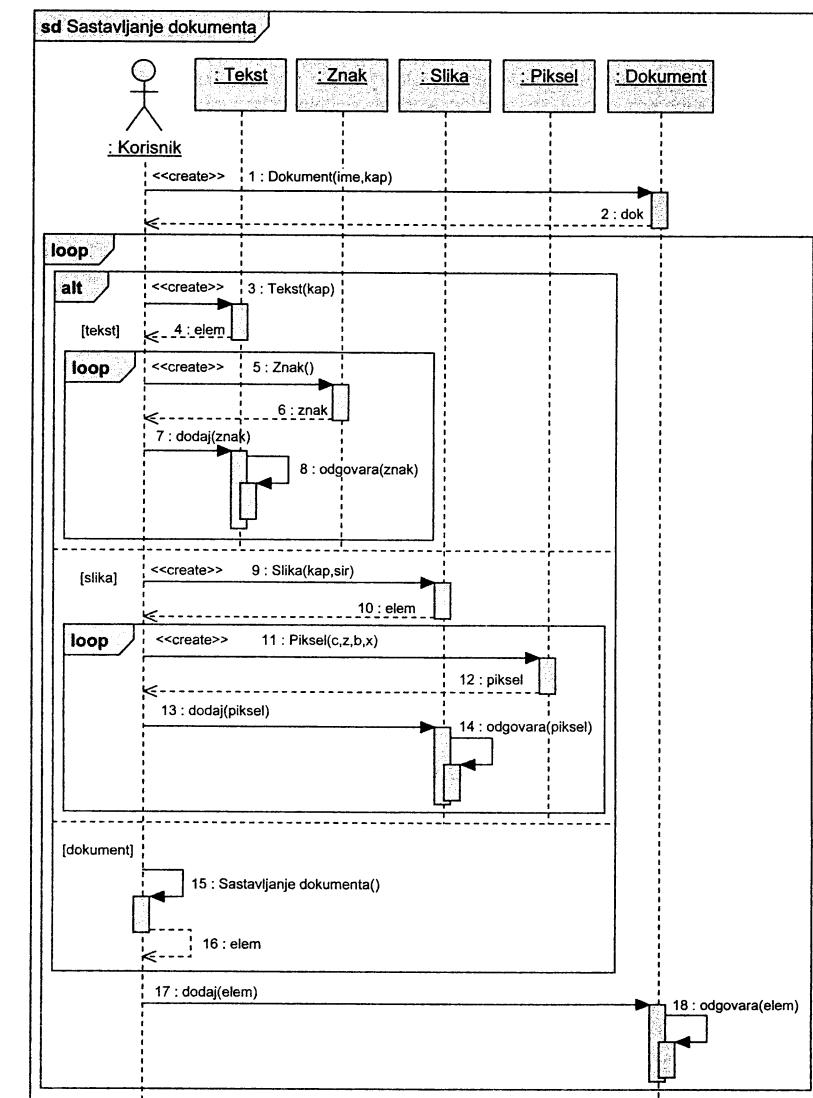


Atom	<<enumeration>> Stil
+kopija(): Atom +velicina(): int +toString(): String	+BOLD +ITALIC +UNDERLINE
Znak	Piksel
-kod: char <<create>>+Znak(kod: char, stil: Stil) +kopija(): Atom +velicina(): int +toString(): String	-crvena: byte -zelena: byte -plava: byte -prozimost: byte <<create>>+Piksel(c: byte, z: byte, p: byte, x: byte) +kopija(): Atom +velicina(): int +toString(): String
Element	<<exception>> GPun
+kap: int +brAtoma: int <<create>>+Element(kap: int) +kopija(): Atom +velicina(): int +toString(): String +dodaj(a: Atom) raises GPun, GTip +odgovara(a: Atom): boolean	<<exception>> GTip
Tekst	Slika
-duz: short <<create>>+Tekst(kap: int) +kopija(): Atom +velicina(): int +toString(): String +odgovara(a: Atom): boolean	+sir: short +vis: short <<create>>+Slika(kap: int, sir: short) +kopija(): Atom +velicina(): int +toString(): String +odgovara(a: Atom): boolean
Dokument	
-ime: String -duz: int <<create>>+Dokument(ime: String, kap: int) +kopija(): Atom +velicina(): int +toString(): String +odgovara(a: Atom): boolean	

b) Projektni uzorci



c) Dijagram sekvenčne izrade strukture



Zadatak 21 Artikli, zbirke, obilasci, osobe, načini plaćanja, samoposluga {K, 16.11.2008.}

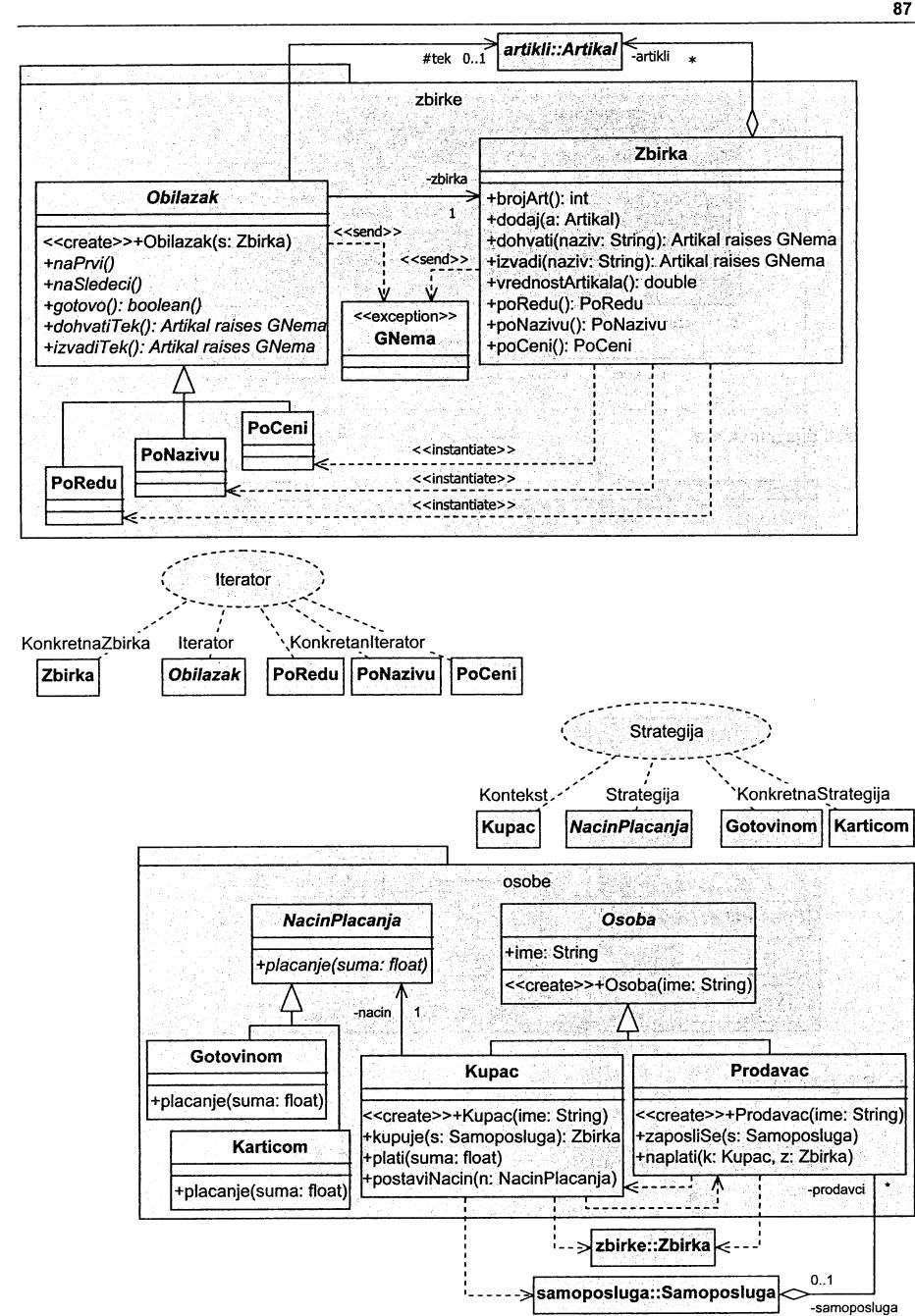
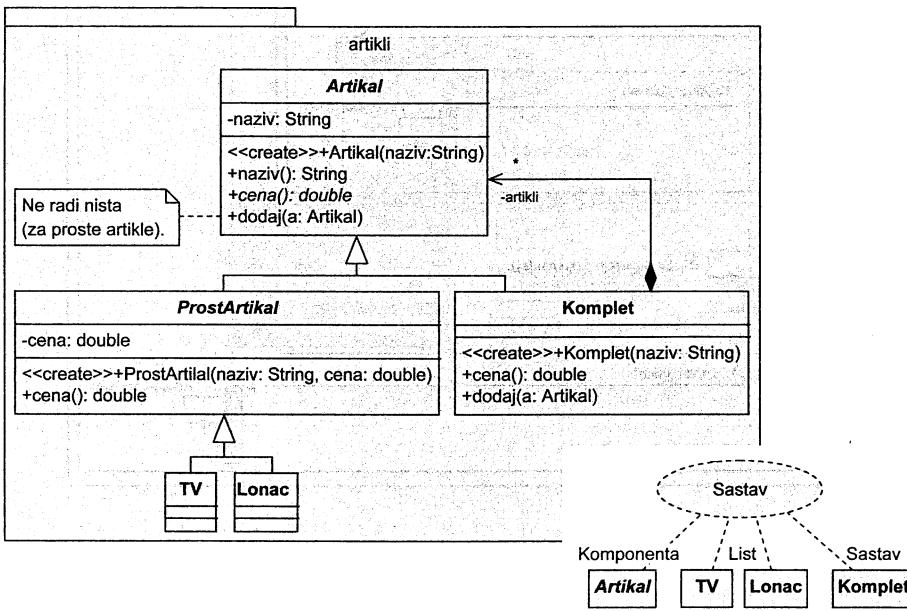
Artikal ima naziv i može da mu se odredi cena. Prost artikal je artikal koji ima zadatu cenu. Televizor i lonac su prosti artikli. Komplet je artikal koji može da sadrži proizvoljan broj artikala. Stvara se prazan posle čega se artikli dodaju jedan po jedan. Cena kompleta jednaka je zbiru cena sadržanih artikala. Zbirka artikala može da sadrži proizvoljan broj artikala. Može da se dohvati broj artikala u zbirki, da se zadati artikal doda zbirki, da se artikal sa zadatim nazivom dohvati ili izvadi iz zbirke i da se odredi ukupna vrednost svih artikala u zbirki. Obilazak zbirke radi pristupa artiklima može se obaviti po redosledu smerštanja, naziva ili cene artikala. Osoba ima ime. Kupac je osoba koja može da kupuje u zadatoj samoposluži i da plati zadatu sumu. Rezultat kupovine je zbirka kupljenih artikala. Postoje dva načina plaćanja: gotovinom ili kreditnom karticom. Prodavac je osoba koja može da se zaposli u zadatoj samoposluži i može da naplati artikle koje je zadati kupac odabro. Samoposluga sadrži zbirku artikala koje prodaje. Može da zaposli ili otpusti zadatog prodavca, da vrati niz trenutno zaposlenih prodavaca, da odjednom nabavi više artikala, da dohvati kao i da izbaci artikal sa zadatim nazivom. Kamion samoposluge je zbirka sa tekstualnim registarskim brojem.

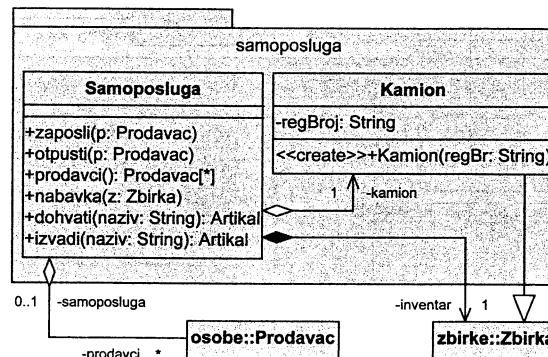
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa razvrstanih u pakete,
- prikaz korišćenih projektnih uzoraka,
- dijagram sekvencije koji prikazuje prijem artikala u samoposluži iz natovarenog kamiona,
- dijagram komunikacije koji prikazuje prodaju artikala u samoposluži jednom kupcu.

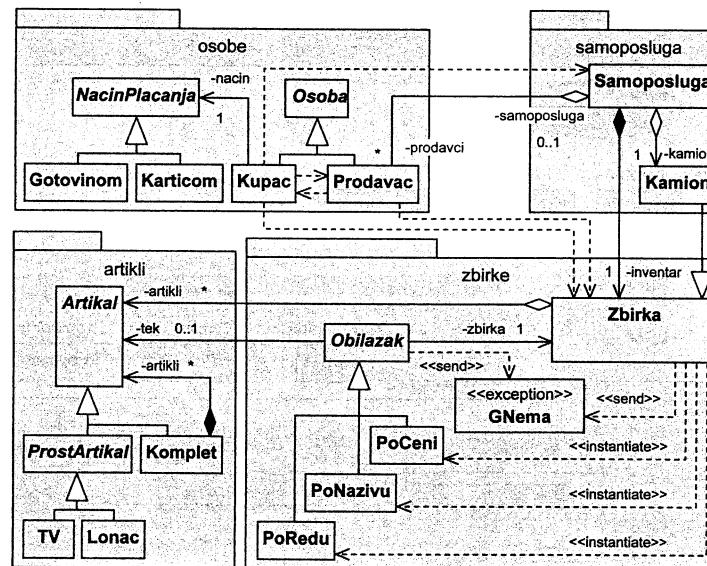
Rešenje:

a) Dijagrami klasa i projektni uzorci

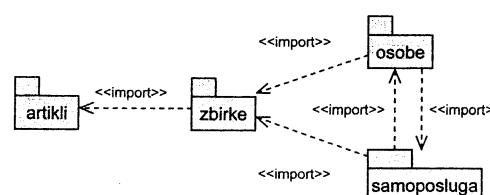




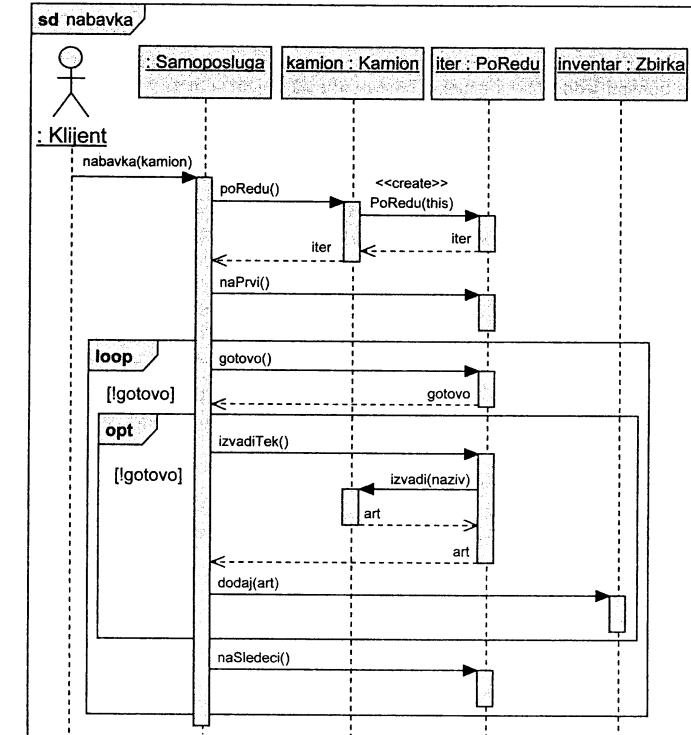
b) Sažeti dijagram klasa



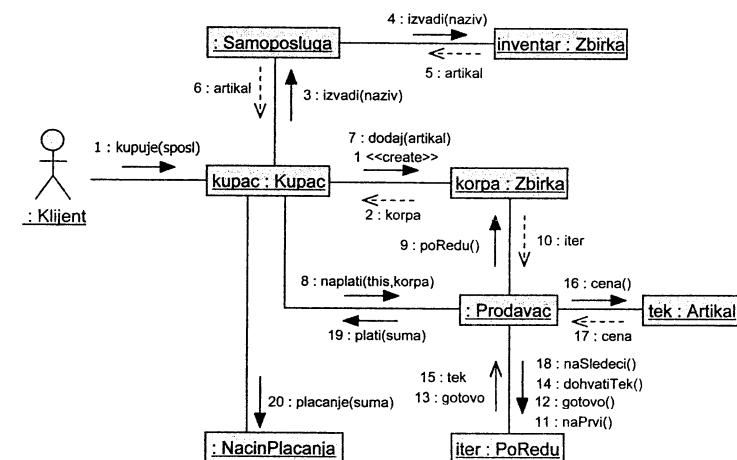
c) Dijagram paketa



d) Dijagram sekvence nabavke artikala



e) Dijagram komunikacije prodaje artikala



Zadatak 22 Polje, tabla, figure, igrači i igre {K, 30.11.2007.}

Radni okvir za igre na tabli ne zavisi od konkretnе igre već obuhvata apstrakcije koje bi se koristile u raznim igrama. Figura za igre ima boju (jedno slovo) i informaciju o polju table na kojem se trenutno nalazi. Može da se dohvati jednoslovna oznaka, boja i polje na kojem se nalazi, kao i da se figura stavi na neko polje i da se premesti na drugo polje na tabli. Dužnost je figure da proveri da li je prema pravilima igre i trenutnom stanju na tabli traženo premeštanje dozvoljeno. Polja dvodimenzionalne table za igre mogu da budu prazna ili da sadrže po jednu figuru. Mogu da se dohvate koordinate polja na tabli, da se dohvati tabla kojoj pripada polje i da se dohvati figura na polju. Tabla sadrži zadati broj vrsta i kolona polja. Može da se postavi figura na dato mesto na tabli, da se dohvati figura s datog mesta, da se figura premesti s jednog na drugo mesto, da se ukloni figura s datog mesta i da se dohvati polje s datim koordinatama. Apstraktan igrač igra figurama zadate boje na zadatoj tabli. Može da odigra potez pomerajući jednu svoju figuru na drugo mesto na tabli. Ako potez nije dozvoljen igrač treba da ponovi potez. Igra sadrži jednu tablu za igru i nekoliko igrača. Stvara se s praznom tablom zadatih dimenzija i popunjena nizom igrača. Može da se postavi početno stanje igre, da se proveri da li je igra završena i da se odigra partija. U toku partije igračima se ciklički omogućava da odigraju potez.

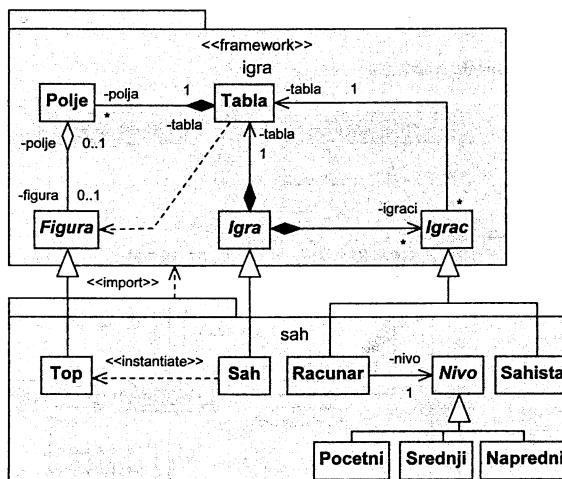
Šah je igra koju igraju dva igrača na tabli od 8×8 polja sa figurama bele i crne boje. Koriste se figure za šah od kojih je jedan top. Šahista je igrač koji vuče poteze preko tastature. Računar je igrač koji automatski vuče poteze na početnom, srednjem ili naprednom nivou.

Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa razvrstanih u pakete,
- prikaz korišćenih projektnih uzoraka,
- dijagram sekvene koji prikazuje odvijanje partije.

Rešenje:

a) Dijagram klasa



igra::Figura	igra::Polje
#boja: char	-x: int
<<create>>+Figura(boja: char)	-y: int
+oznaka(): char	<<create>>+Polje(t: Tabla, x: int, y: int)
+boja(): char	+x(): int
+stavi(na: Polje)	+y(): int
+pomeri(na: Polje)	+postavi(f: Figura)
+polje(): Polje	+dohvatiTablu(): Tabla
+moze(na: Polje): boolean	+dohvatiFiguru(): Figura

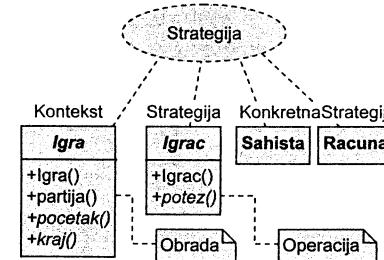
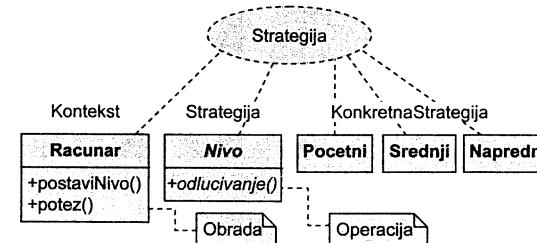
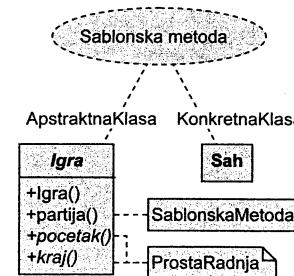
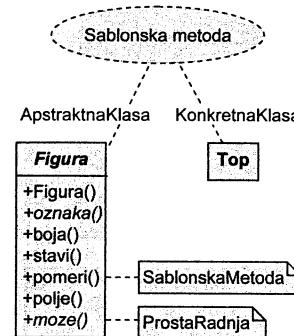
igra::Tabla
-sir: int
-vis: int
+Tabla(sir: int, vis: int)
+sir(): int
+vis(): int
+staviFiguru(f: Figura, x: int, y: int)
+dohvatiFiguru(x: int, y: int): Figura
+premetiFiguru(x1: int, y1: int, x2: int, y2: int)
+ukloniFiguru(x: int, y: int)
+dohvatiPolje(x: int, y: int): Polje

igra::Igrac
#boja: char
<<create>>+Igrac(boja: char, t: Tabla)
+potez()

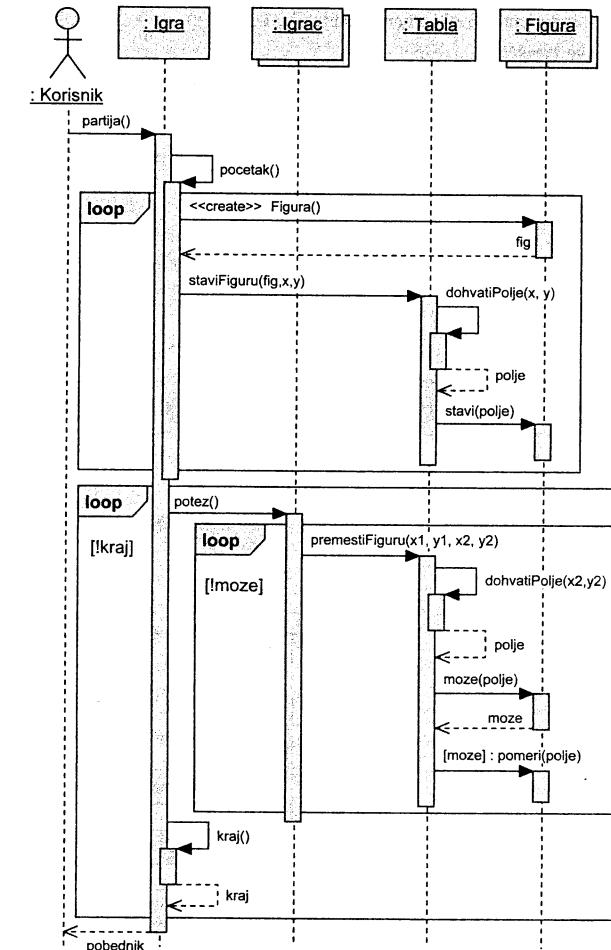
igra::Igra
<<create>>+Igra(t: Tabla,igr: Igra[*])
+partija(): int
+pocetak()
+kraj(): boolean

sah::Sah	sah::Top	sah::Sahista	sah::Racunar
+pocetak()	<<create>>+Top(boja: char)	+potez()	+postaviNivo(n: Nivo)
+kraj(): boolean	+oznaka(): char		+potez()
	+moze(naPolje): boolean		
sah::Nivo	sah::Pocetni	sah::Srednji	sah::Napredni
+odlucivanje()	+odlucivanje()	+odlucivanje()	+odlucivanje()

b) Projektni uzorci



c) Dijagram sekvenca igranja partije



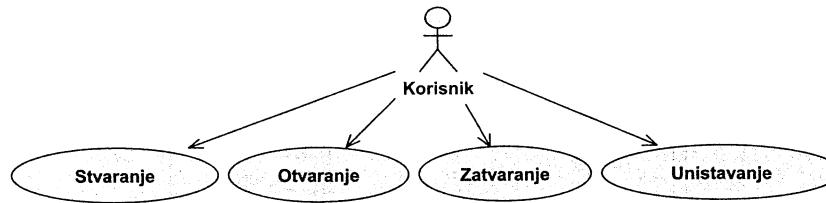
Zadatak 23 Samoposluga (dijagrami aktivnosti i stanja)

Za model samoposluge iz zadatka 5 i 11 nacrtati na jeziku UML:

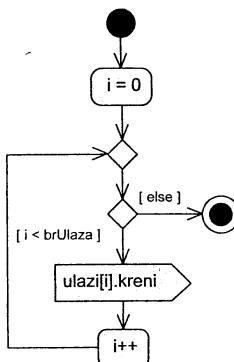
- dijagram slučajeva korišćenja,
- dijagrame aktivnosti otvaranja samoposluge, ulaza, kupca, kase i zatvaranja samoposluge,
- dijagrame stanja kupca i samoposluge.

Rešenje:

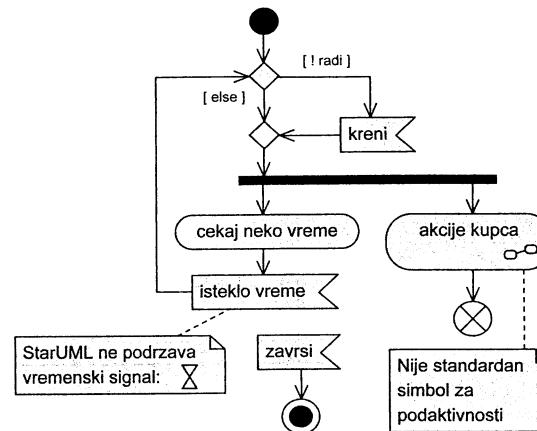
a) Dijagram slučajeva korišćenja



b) Dijagram aktivnosti otvaranja samoposluge



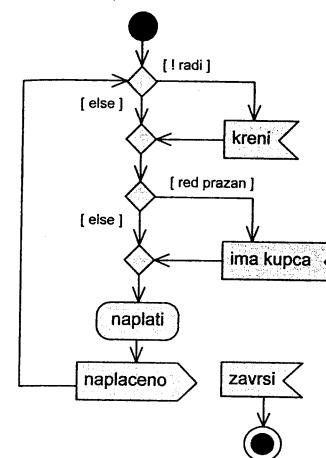
c) Dijagram aktivnosti ulaza



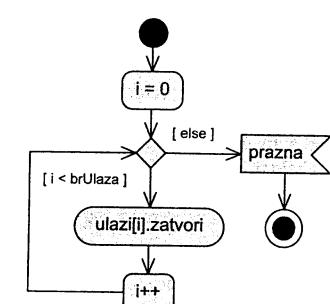
d) Dijagram aktivnosti kupca



e) Dijagram aktivnosti kase



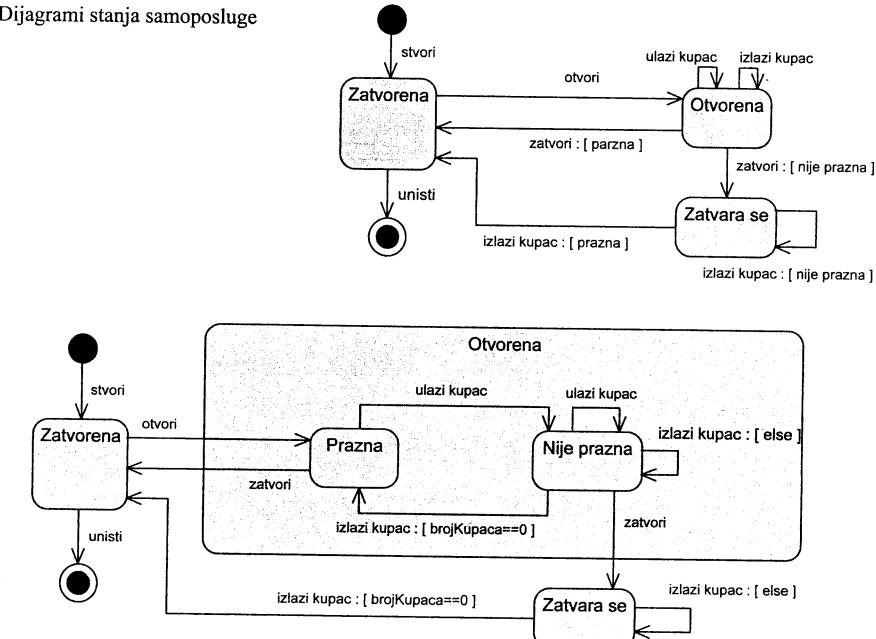
f) Dijagram aktivnosti zatvaranja samoposluge



g) Dijagram stanja kupca



h) Dijagrami stanja samoposluge



Zadatak 24 Predmeti i akteri

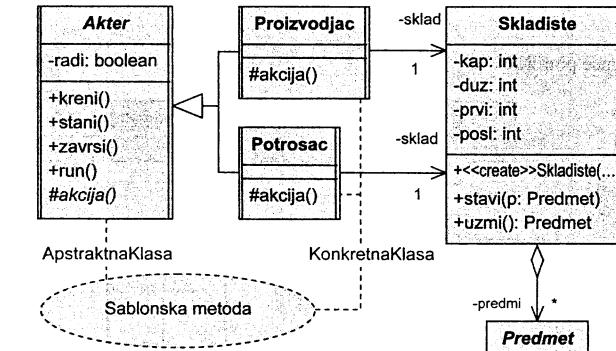
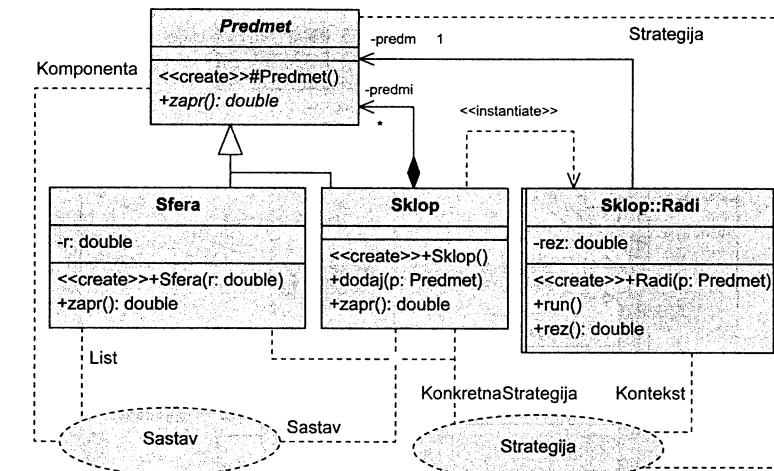
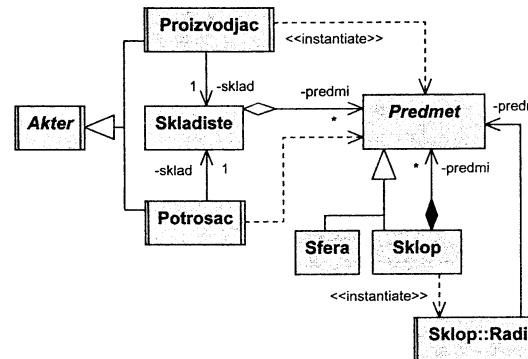
Apstraktom predmetu može da se izračuna zapremina. Sfera je predmet. Sklop je predmet koji može da sadrži proizvoljan broj predmeta. Zapremine delova sklopa računaju se konkurentno. Skladište može da sadrži zadati broj predmeta. Predmeti se stavljuju i uzimaju jedan po jedan. Pokušaj stavljanja predmeta u puno skladište ili uzimanja iz praznog skladišta dovodi do privremenog blokiranja niti izvršiocu. Apstrakstan aktivan akter ponavlja neku apstraktну akciju. Rad aktera može da se zaustavi, da se nastavi dalje i da se završi. Proizvođač je akter čija se akcija sastoji od pravljenja i stavljanja u zadato skladište po jednog predmeta. Potrošač je akter čija se akcija sastoji od uzimanja po jednog predmeta iz zadatog skladišta i izračunavanja zapremine tog predmeta.

Projektovati na jeziku UML prethodni sistem. Priložiti:

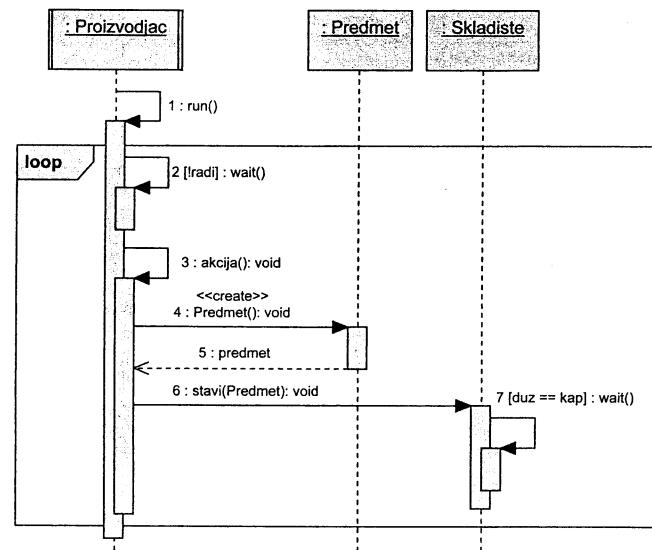
- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagrame interakcije rada proizvođača i potrošača,
- dijagrame aktivnosti proizvođača i potrošača,
- dijagrame aktivnosti računanja zapremine sklopa,
- dijagrame stanja skladišta, proizvođača i potrošača.

Rešenje:

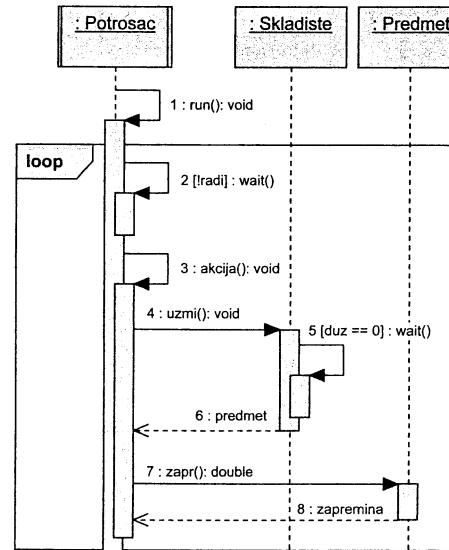
a) Dijagrami klasa i projektni uzorci



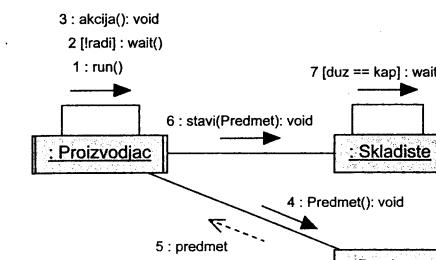
b) Dijagram sekvence proizvođača



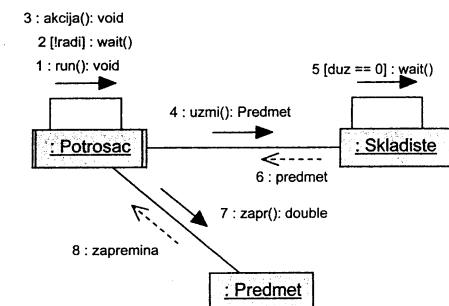
c) Dijagram sekvence potrošača



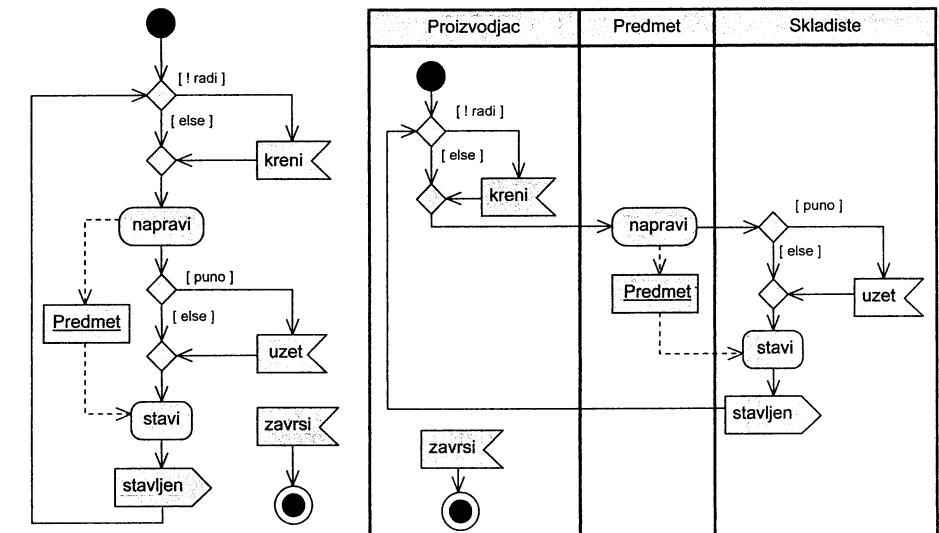
d) Dijagram komunikacije proizvođača



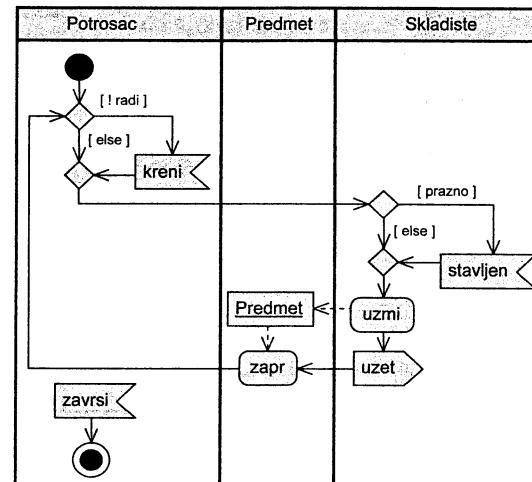
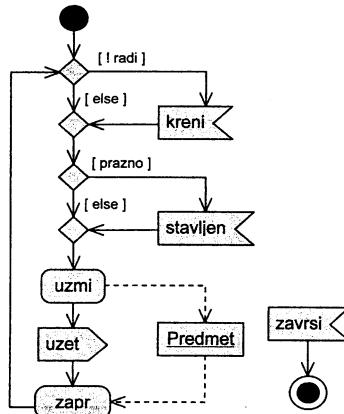
e) Dijagram komunikacije potrošača



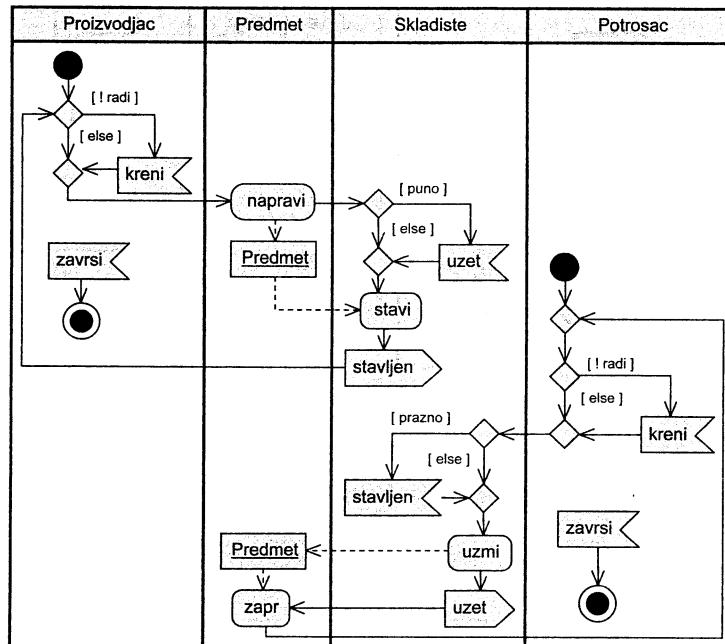
f) Dijagrami aktivnosti proizvođača



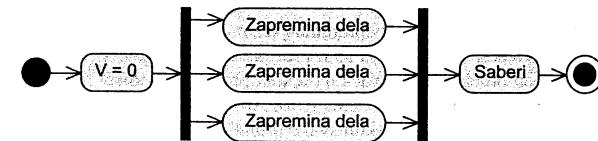
g) Dijagrami aktivnosti potrošača



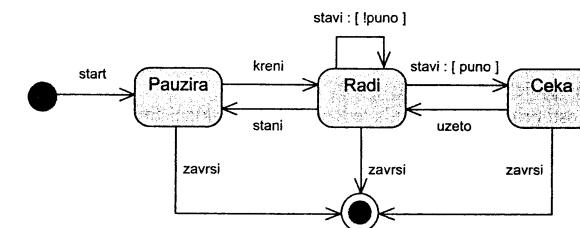
h) Zajednički dijagrami aktivnosti proizvođača i potrošača



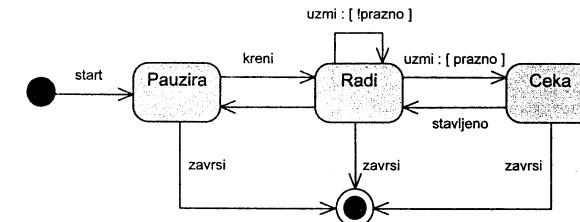
i) Dijagram aktivnosti izračunavanja zapremine sklopa



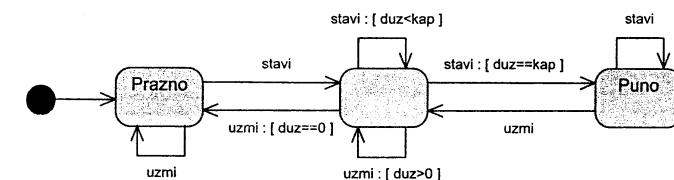
j) Dijagram stanja proizvođača



k) Dijagram stanja potrošača



l) Dijagram stanja skladišta



Zadatak 25 Predmet, skladište, pokazivač, nadzornici {K2, 15.12.2006.}

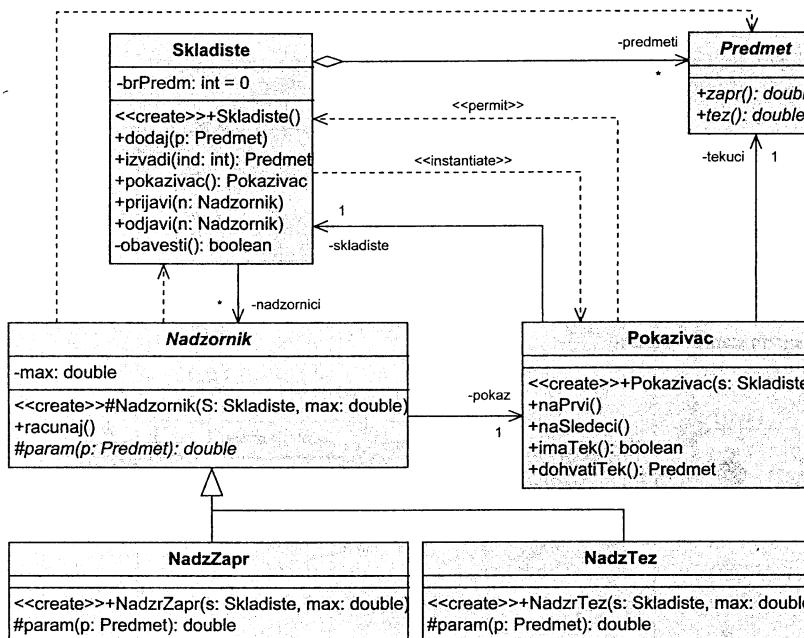
Apstraktnom predmetu može da se izračuna zapremina i težina. Skladište može da sadrži proizvoljan broj predmeta. Može da se doda jedan predmet i da se izvadi predmet sa zadatim rednim brojem (predmet se uklanja iz skladišta). Pomoću objekta pokazivača na tekući predmet skladišta mogu da se dohvataju uzaštopni predmeti iz skladišta (predmeti ostaju u skladištu). Apstraktan nadzornik nadgleda određeni realan parametar skladišta. Parametar se određuje nakon svake promene u skladištu kao zbir odgovarajućih parametara svih predmeta u skladištu. Ukoliko je parametar veći od neke zadate veličine, prekida se tekuća prozivka nadzornika i ispiše se poruka. Nadzornici mogu jedan po jedan da se prijavljuju i odjavljaju od skladišta. Nadzornik zapremine i nadzornik težine prate ukupnu zapreminu, odnosno težinu predmeta u skladištu.

Projektovati na jeziku UML prethodni sistem. Priložiti:

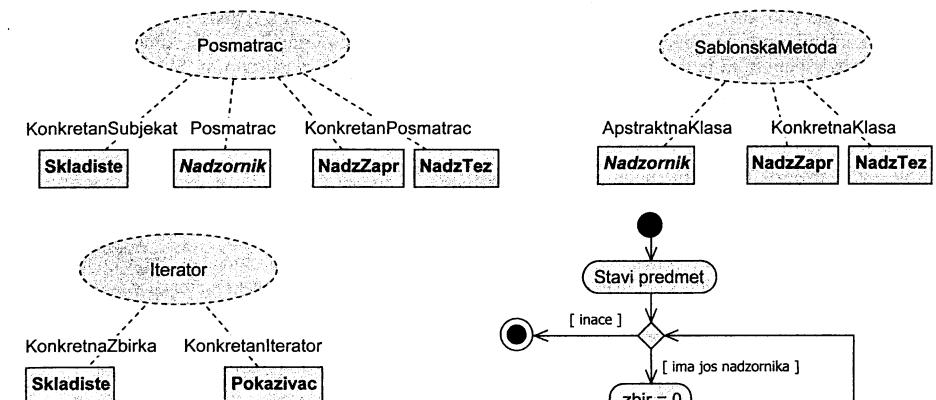
- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram aktivnosti prilikom dodavanja jednog predmeta u skladištu.

Rešenje:

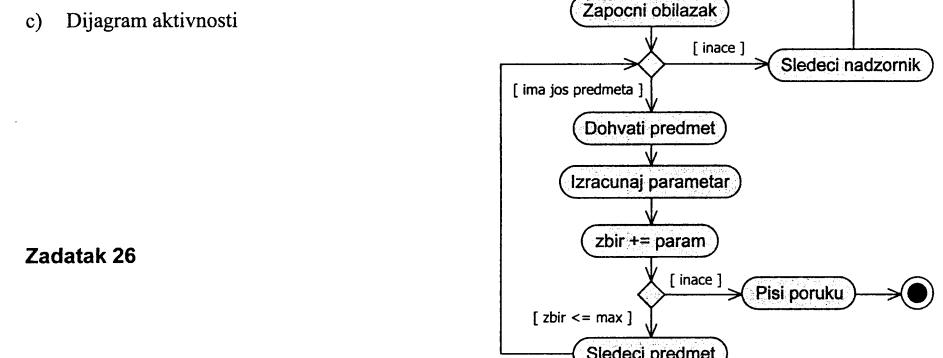
a) Dijagram klasa



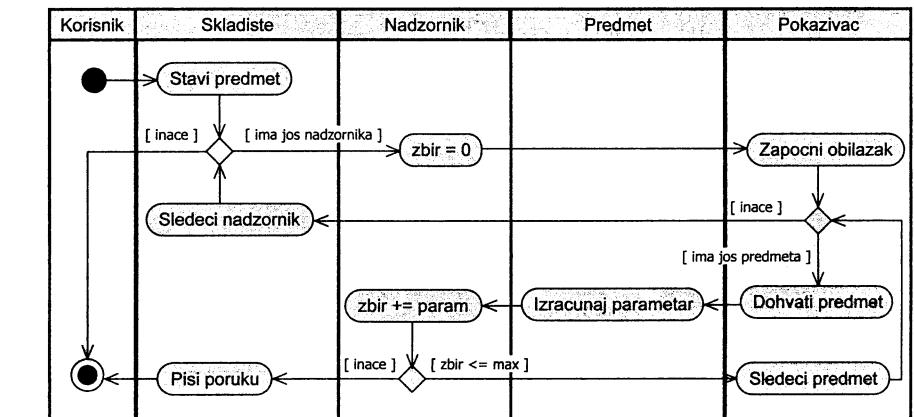
b) Projektni uzorci



c) Dijagram aktivnosti



Zadatak 26



Roba, artikal, porez, popust, skladište, redosled i izveštaj {K2, 06.12.2007.}

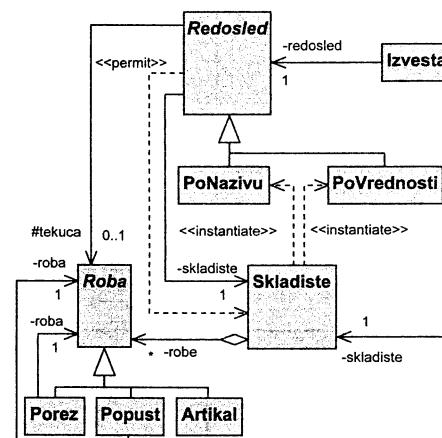
Roba predviđa dohvatanje naziva, cene i količine i izračunavanje vrednosti. Artikal je roba koja sadrži naziv, jediničnu cenu i količinu. Roba s porezom je roba koja na cenu pridružene robe zaračunava zadati porez. Roba s popustom je roba koja od cene pridružene robe odbija zadati popust. Skladište može da sadrži proizvoljno mnogo robe. Stvara se prazno posle čega može da se doda i da se izvadi zadata roba. Robe u skladištu mogu da se dohvate po redosledu naziva i po redosledu vrednosti. Izveštaj može da ispiše sadržaj zadatog skladišta. Može da se postavlja redosled prikazivanja robe u skladištu.

Projektovati na jeziku UML prethodni sistem. Priložiti:

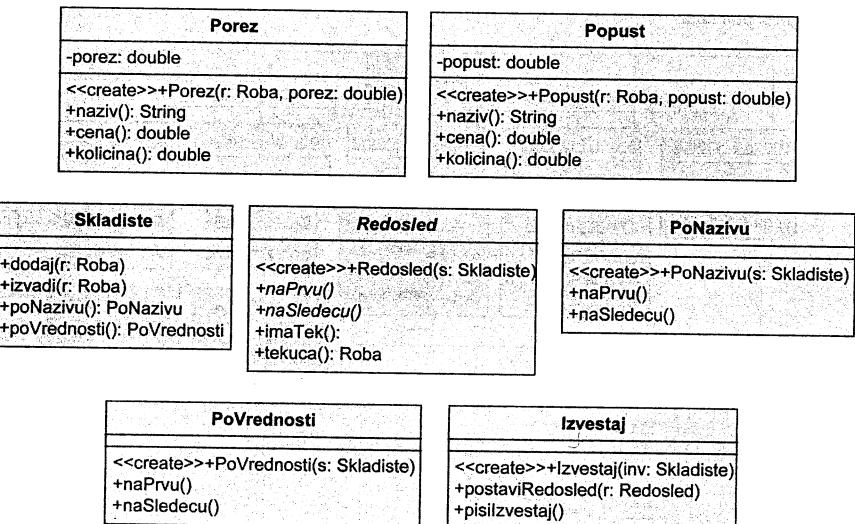
- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram objekata koji prikazuje skladište koje sadrži po jedan primerak robe bez poreza i popusta, samo s porezom, samo s popustom, s porezom posle čega se odbija popust i s popustom posle čega se dodaje porez,
- dijagrame sekvence i aktivnosti sastavljanja izveštaja za slučaj kada se na svaku robu prvo dodaje porez i posle tогa odbija popust.

Rešenje:

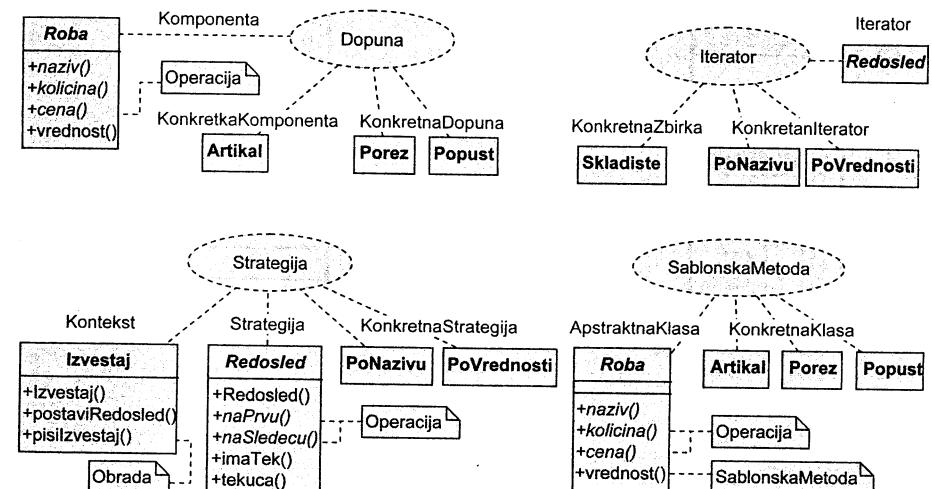
a) Dijagram klasa



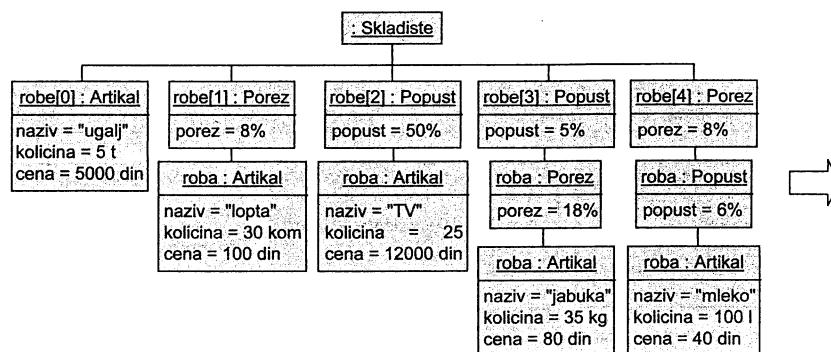
Roba	Artikal
<code>+naziv(): String</code> <code>+kolicina(): double</code> <code>+cena(): double</code> <code>+vrednost(): double</code>	<code>-naziv: String</code> <code>-cena: double</code> <code>-kolicina: double</code> <code><<create>>+Artikal(naziv: String, cena: double, kolicina: double)</code> <code>+naziv(): String</code> <code>+cena(): double</code> <code>+kolicina(): double</code>



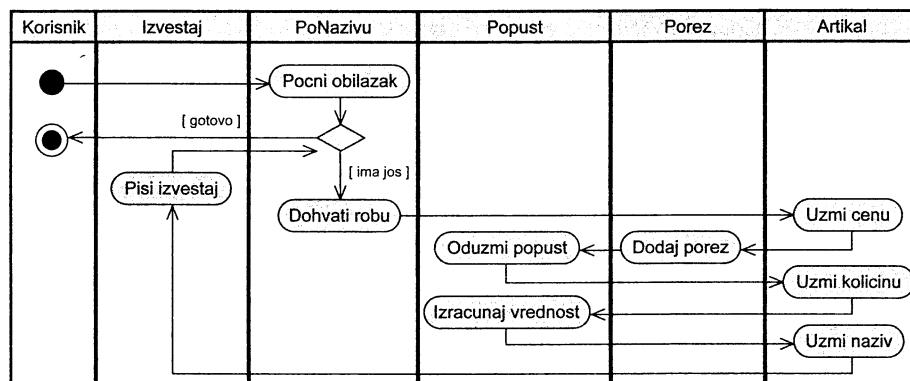
b) Projektni uzorci



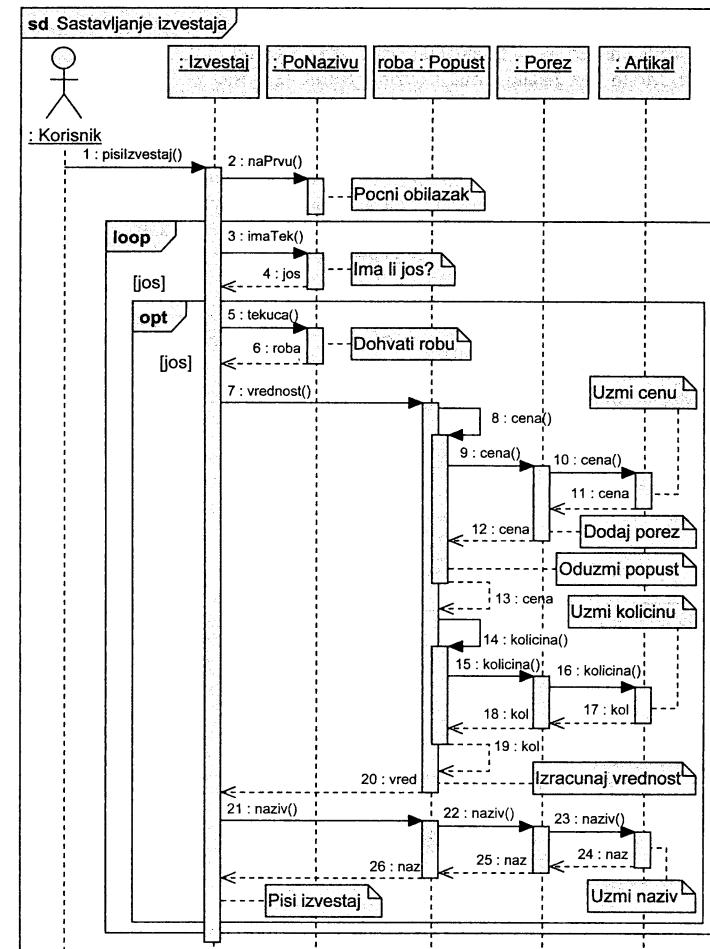
c) Dijagram objekata



e) Dijagram aktivnosti



d) Dijagram sekvenca



Zadatak 27 Vozila, mesto, parking, redosled i semafor {K2, 04.12.2008.}

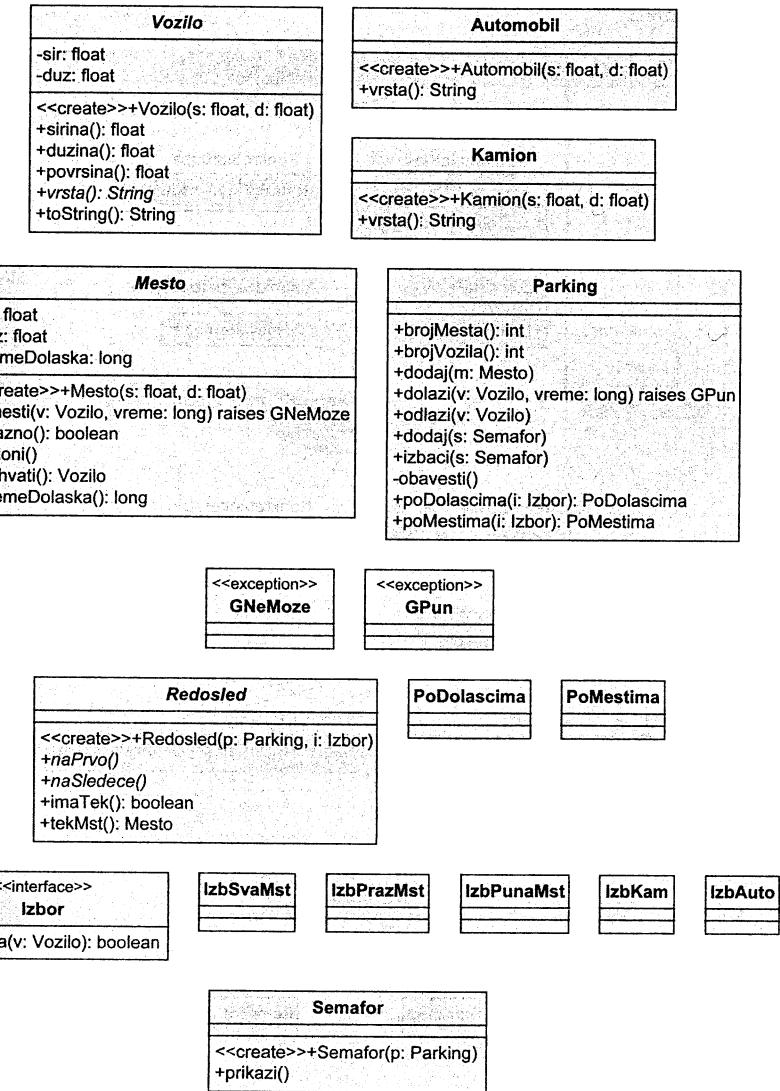
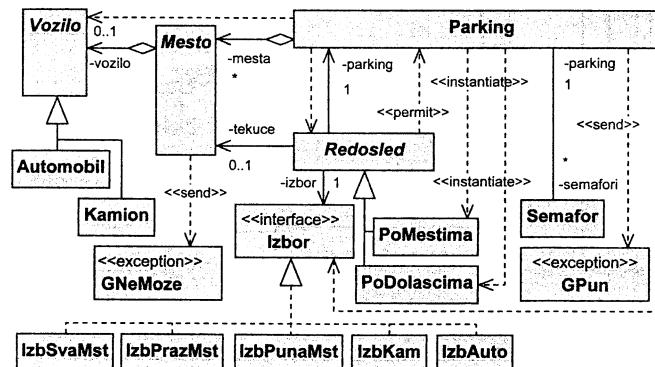
Vozilo ima zadatu širinu i dužinu koje mogu da se dohvate. Može da se odredi površina koju zauzima, da se dohvati vrsta vozila i da se sastavi tekstualni opis koji se sastoji od vrste i površine vozila. Automobil i kamion su vozila. Mesto ima zadatu širinu i dužinu. Može da se smesti jedno vozilo na mesto, da se vozilo dohvati ili ukloni i da se ispita da li je mesto prazno. Greška je ako vozilo ne može da se smesti na datu mesto. Parking ima određen broj mesta različite veličine. Stvara se prazan posle čega mesta mogu da se dodaju jedno po jedno. Na parking vozila mogu da dolaze i sa njega odlaze jedno po jedno. Greška je ako vozilo ne može da nađe slobodno mesto koje mu odgovara. Parking može da se obide po redosledu mesta i po redosledu dolazaka vozila, a pri tome da se uzmu u obzir samo mesta sa vozilima jedne vrste, samo popunjena mesta, samo prazna mesta ili sva mesta. Postoji proizvoljan broj semafora koji prikazuju informacije o broju raspoloživih i zauzetih mesta na parkingu.

Projektovati na jeziku UML prethodni sistem. Priložiti:

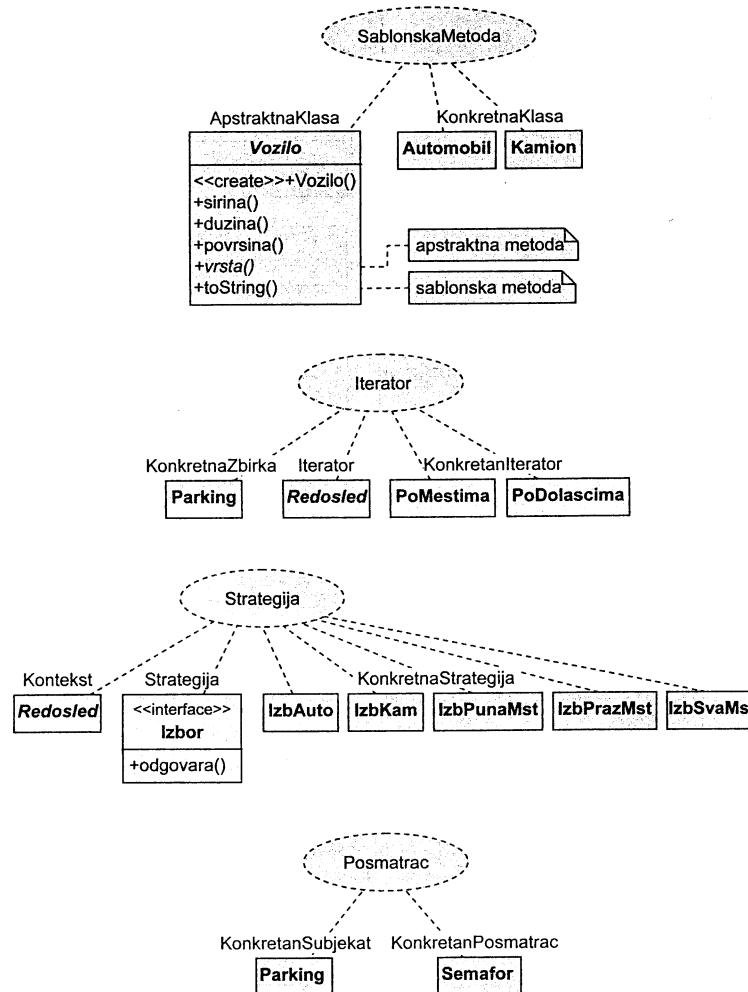
- dijagram klasa,
- prikaz korišćenih projektnih uzoraka,
- dijagram aktivnosti pri dolasku jednog vozila na parking,
- dijagram stanja parkinga.

Rešenje:

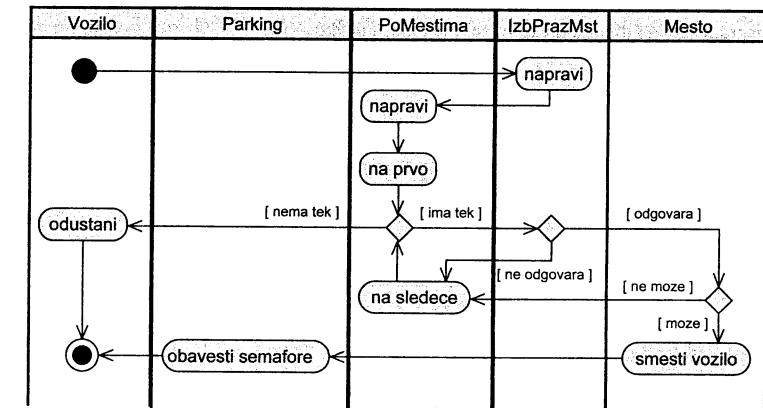
a) Dijagram klasa



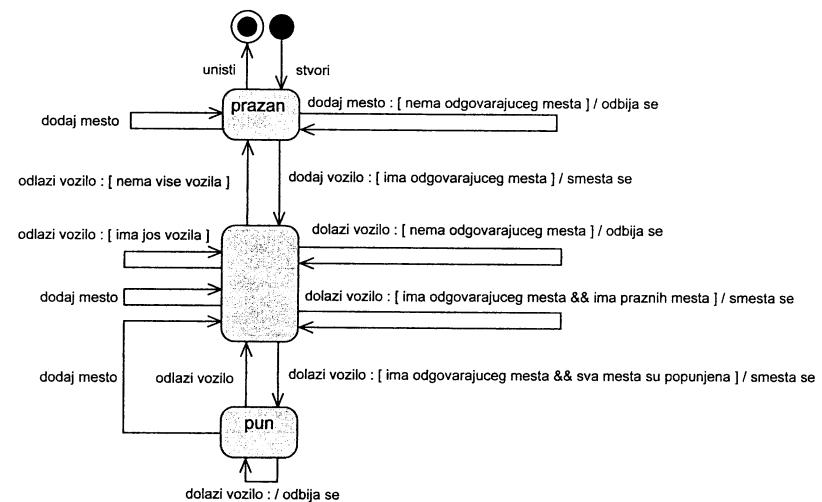
b) Projektni uzorci



c) Dijagram aktivnosti dolaska vozila



d) Dijagram stanja parkinga

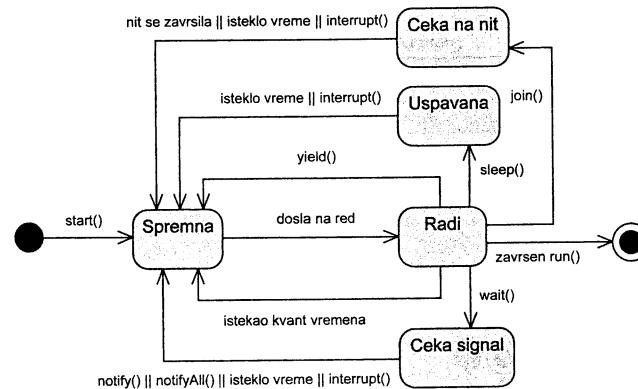


Zadatak 28 Dijagram stanja i aktivnosti niti

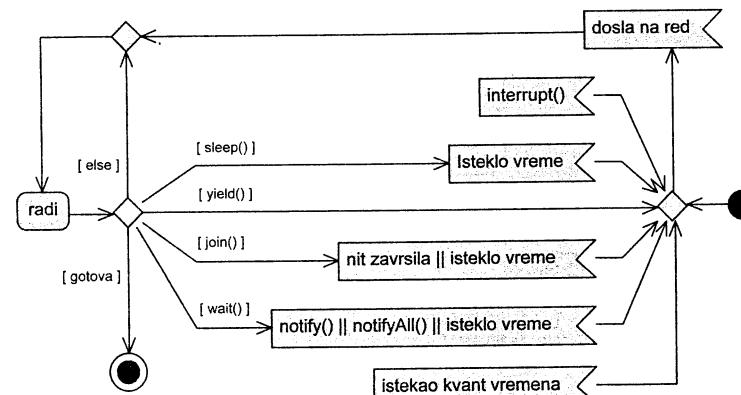
Nacrtati dijagram stanja i dijagram aktivnosti rada niti u jeziku Java.

Rešenje:

a) Dijagram stanja

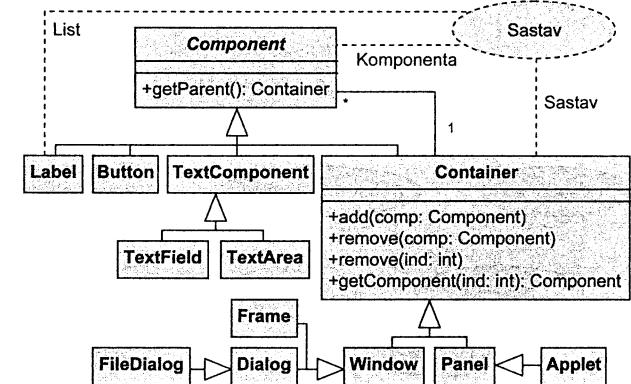
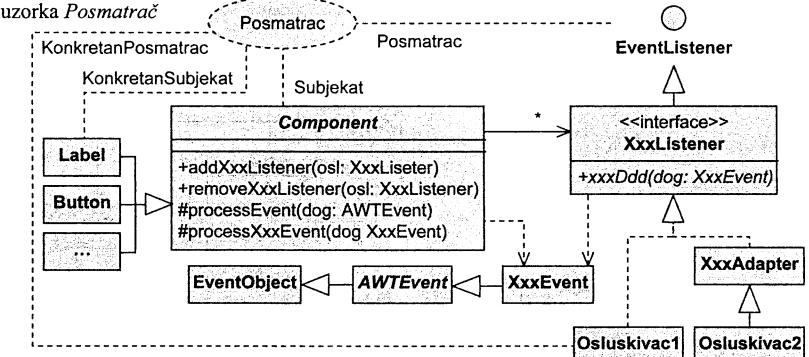
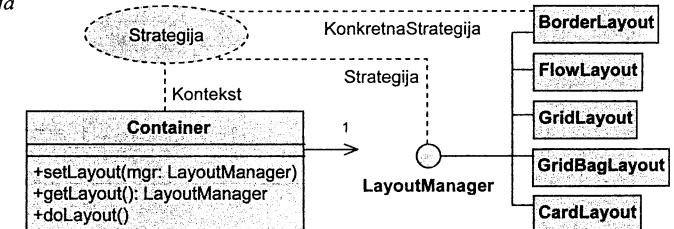


b) Dijagram aktivnosti

**Zadatak 29** Projektni uzorci u AWT-u

Naći primere projektnih uzoraka u AWT-u jezika Java.

Rešenje:

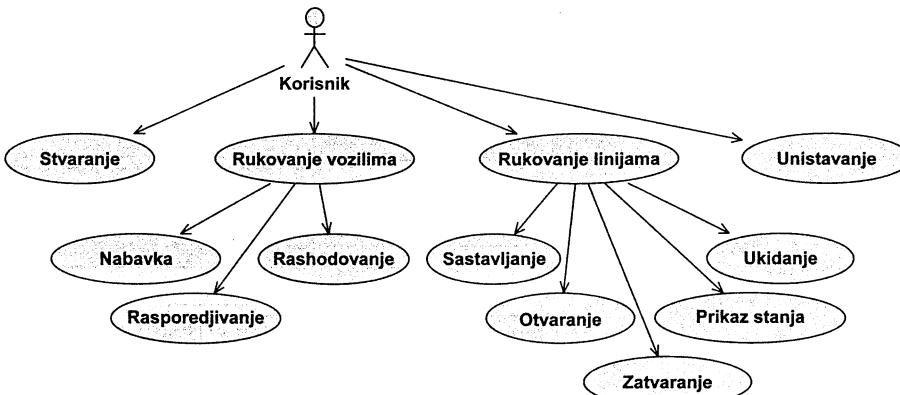
a) Primer uzorka *Sastav*b) Primer uzorka *Posmatrač*c) Primer uzorka *Strategija*

Zadatak 30 Gradska saobraćaj (projektni uzorak *Stanje*)

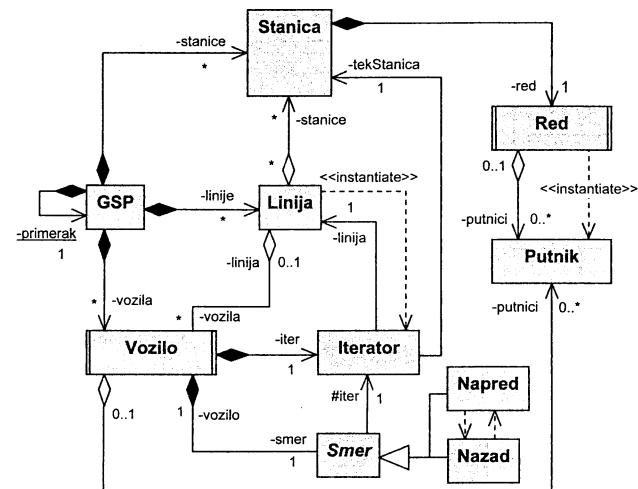
Projektovati na jeziku UML model sistema za simuliranje rada gradskog saobraćaja.

Rešenje:

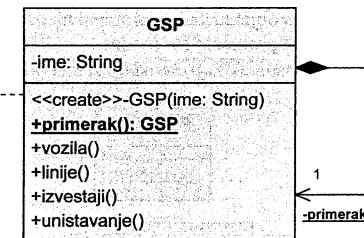
a) Dijagram slučajeva korišćenja



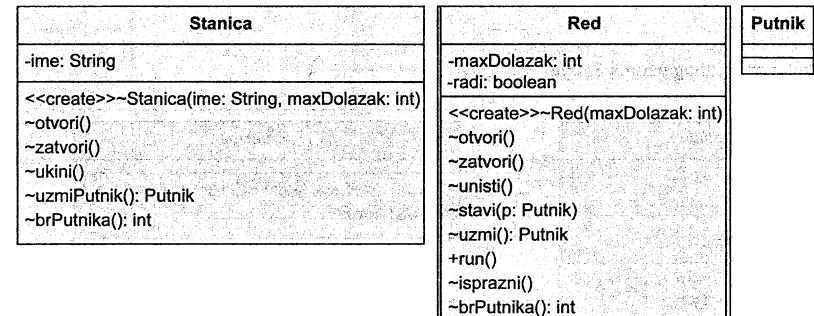
b) Dijagram klasa



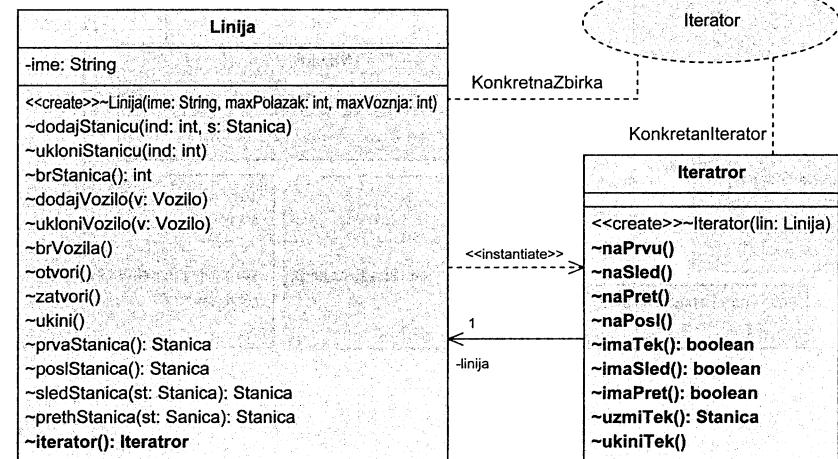
c) Primer projektnog uzorka *Unikat*



d) Klase *Stanica*, *Red* i *Putnik*

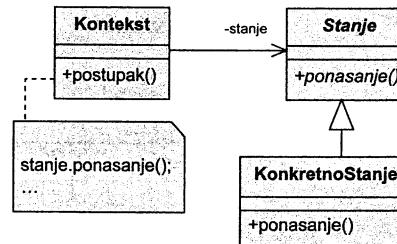
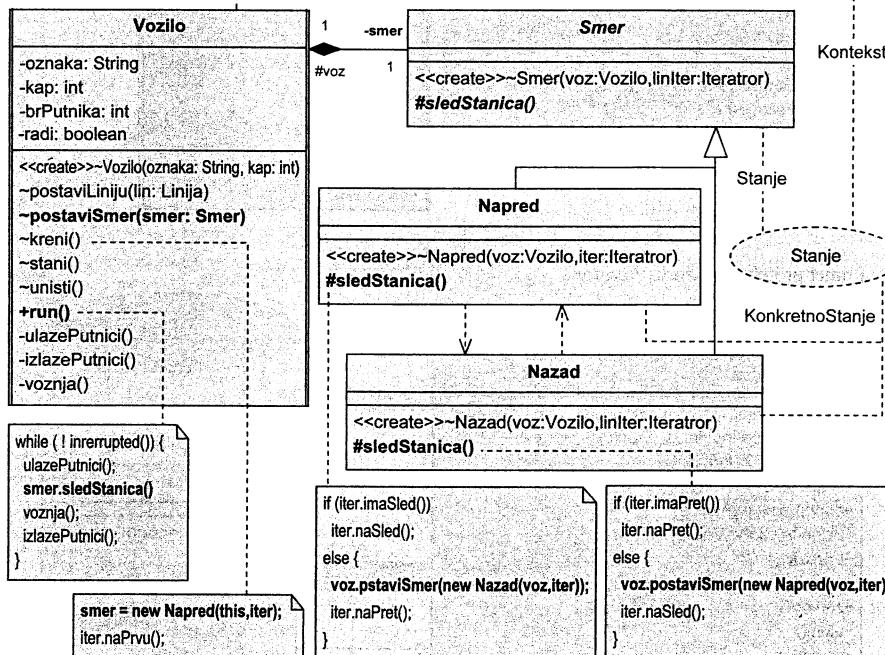


e) Primer projektnog uzorka *Iterator*

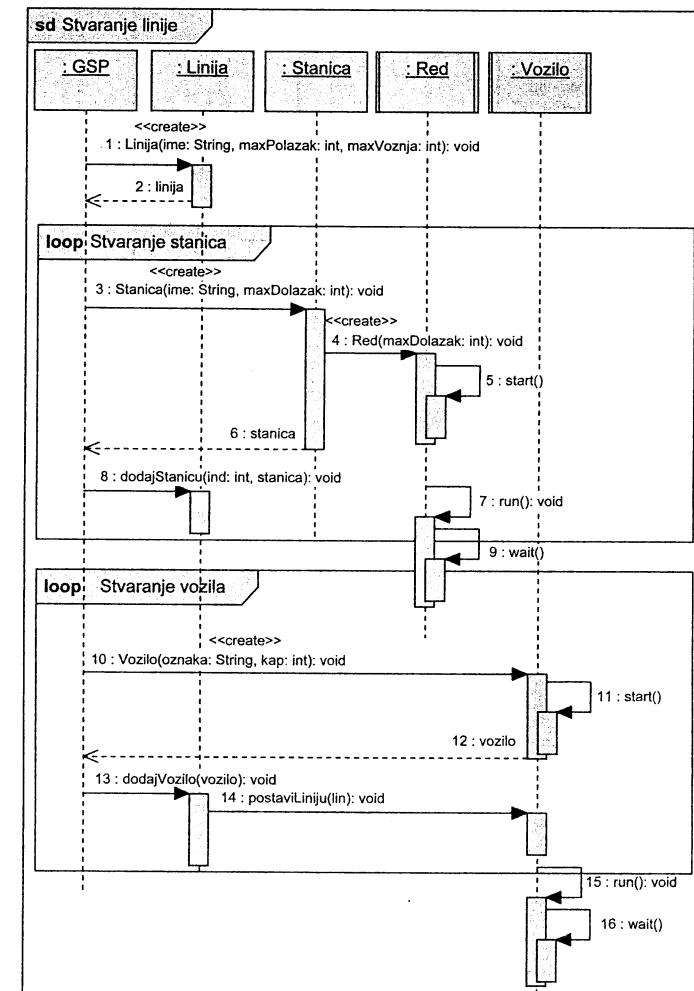


f) Projektni uzorak *Stanje (State)*

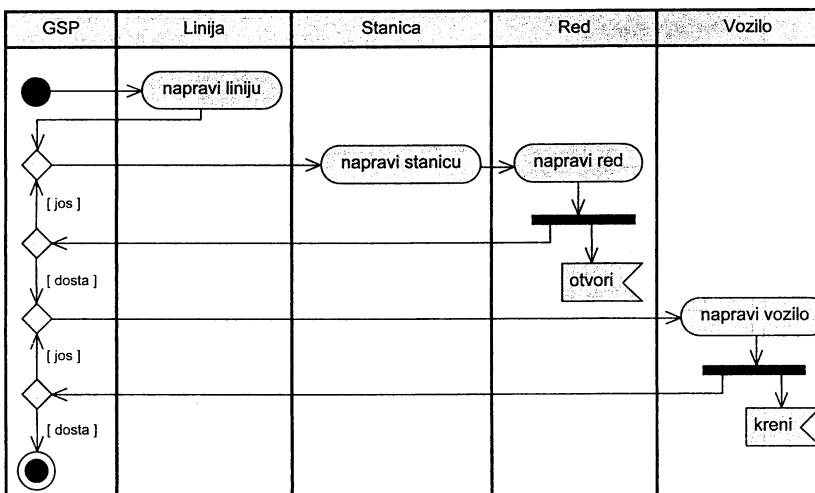
- objektni uzorak ponašanja
- menjanje ponašanja objekta kad se promeni njegovo unutrašnje stanje
 - kao da je objekat promenio svoju klasu
- kontekst obavlja postupak koji zavisi od stanja konteksta

g) Primer projektnog uzorka *Stanje*

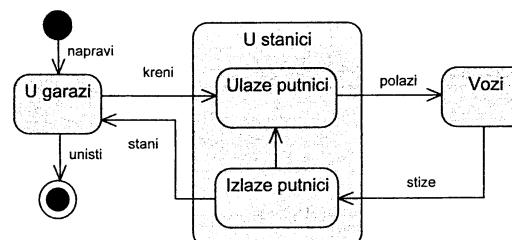
h) Dijagram sekvencije sastavljanja linije



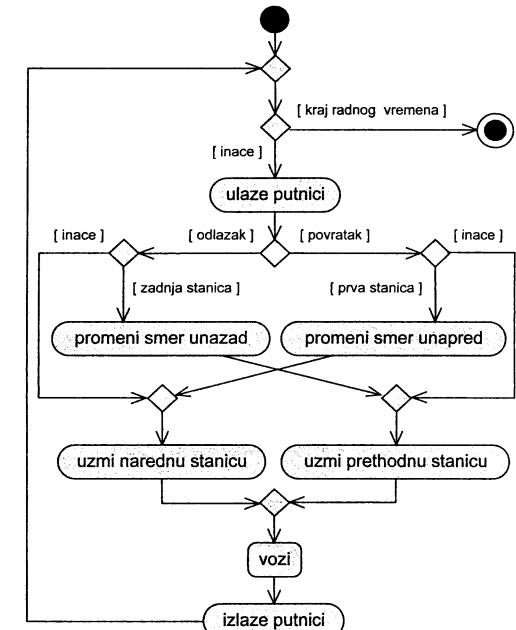
i) Dijagram aktivnosti sastavljanja linije



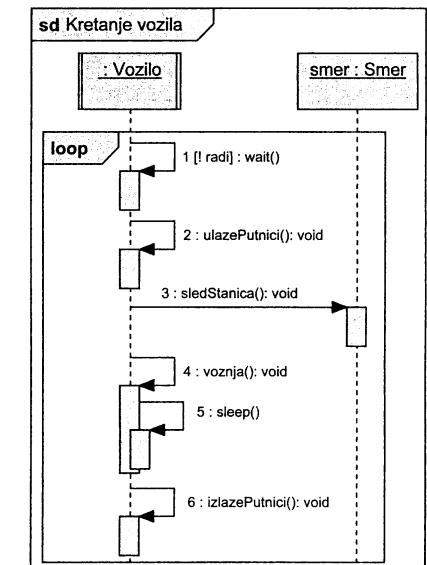
j) Dijagram stanja vozila



k) Dijagram aktivnosti kretanja vozila



l) Dijagram sekvene kretanja vozila



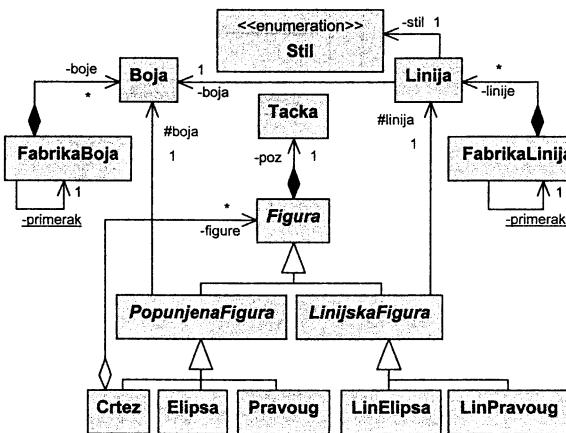
Zadatak 31 Figure u ravni (projektni uzorak Muva)

Projektovati na jeziku UML model sledećeg problema:

Apskratna figura može da se crta na grafičkoj korisničkoj površi unutar pravougaonog prostora zadatih dimenzija sa gornjim levim uglom u zadatoj tački. Postoje popunjene figure koje se popunjavaju zadatom bojom i linijske figure koje se crtaju linijom zadatog stila (puna, isprekidana, tačkasta i crta-tačka), debljine i boje. Pravougaoni i elipse mogu da se crtaju popunjeno i linijski. Crtež je popunjena figura koja može da sadrži proizvoljan broj figura proizvoljne vrste. Prepostaviti da se tipično koristi mali broj različitih boja i različitih vrsta linija.

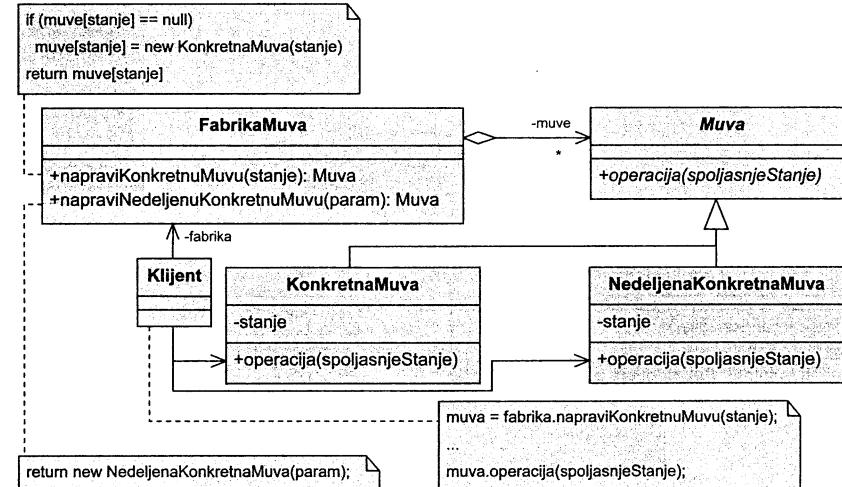
Rešenje:

a) Dijagram klasa

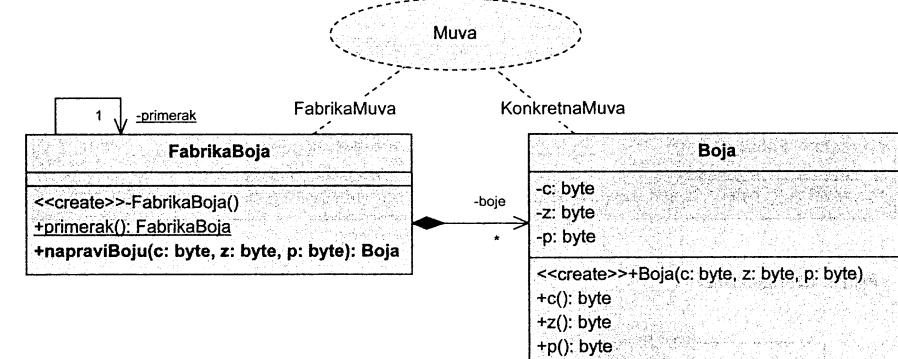


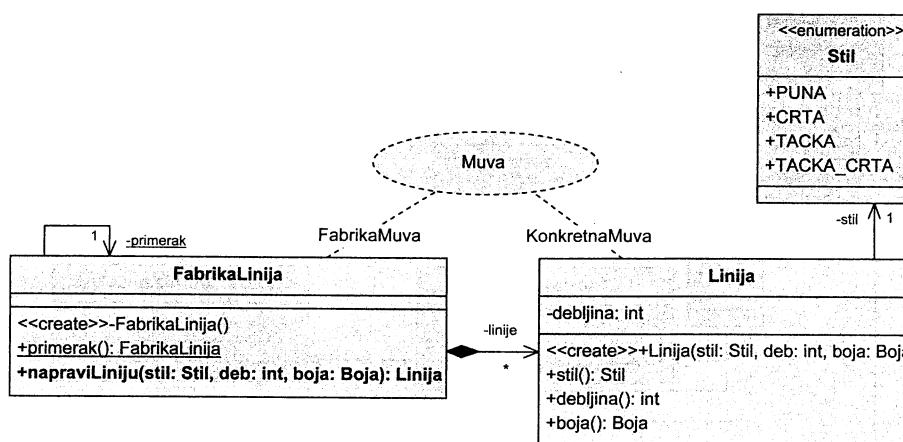
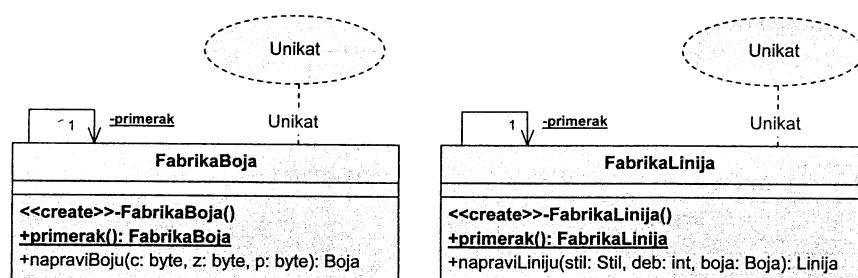
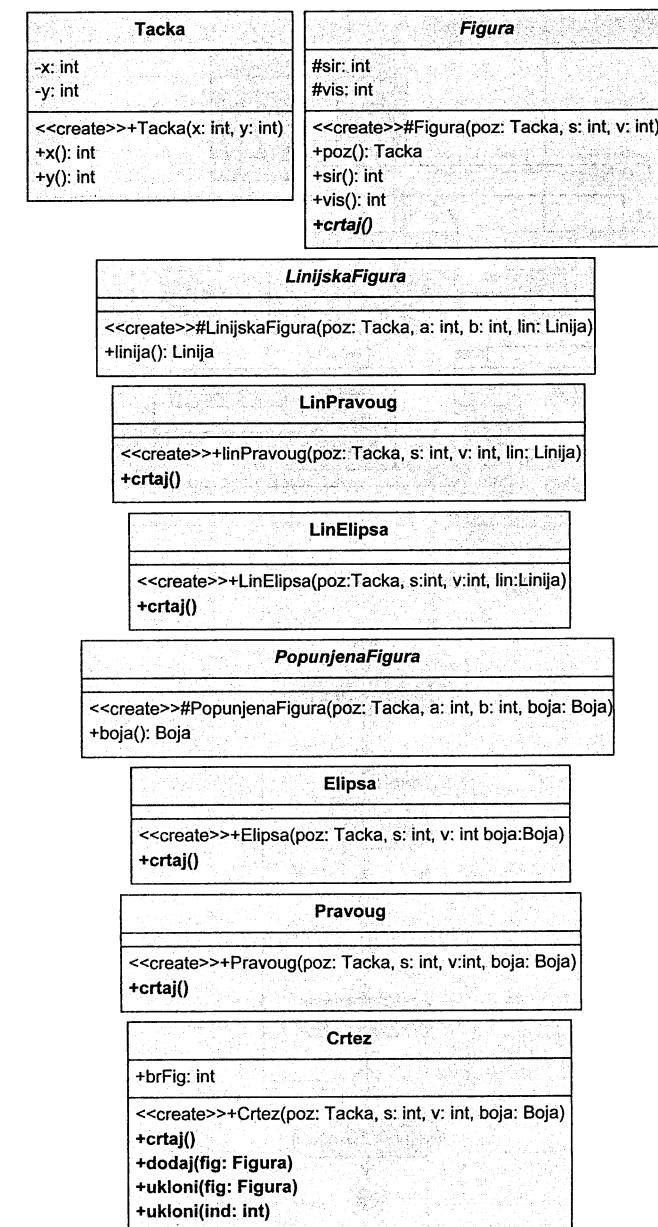
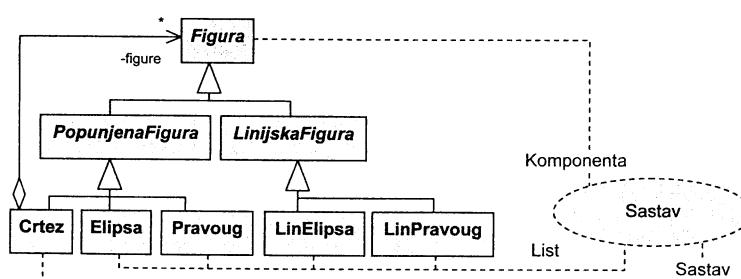
b) Projektni uzorak **Muva** (*Flyweight*)

- objektni uzorak strukture
- deljenje objekata radi izbegavanja hiperprodukcije objekata
 - objekti ili nemaju stanja ili im stanje ne zavisi od konteksta korišćenja (ne menja se posle stvaranja)
- klijent stvara objekte isključivo posredstvom fabrike objekata koja vodi računa o tome da ne postoje dva objekta s istim *unutrašnjim* stanjem, a pri korišćenju navodi *spoljašnje* stanje zavisno od konteksta korišćenja

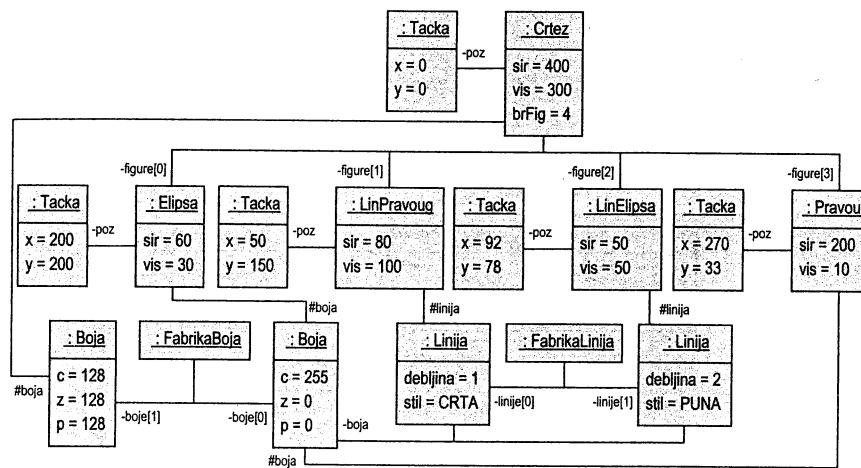


c) Primeri uzorka **Muva**

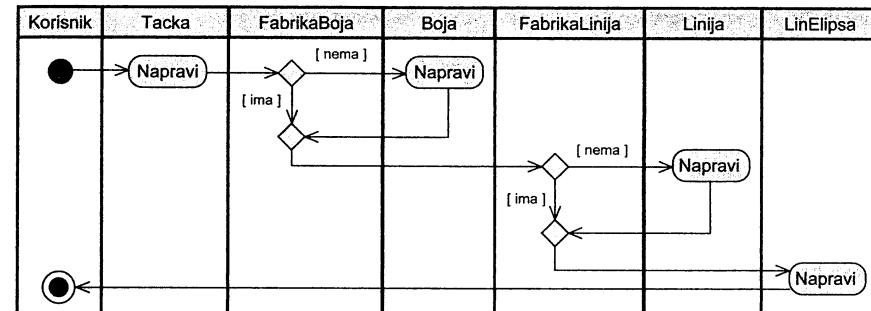
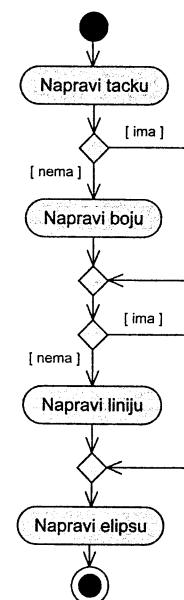


d) Primeri uzorka *Unikat*e) Primer uzorka *Sastav*

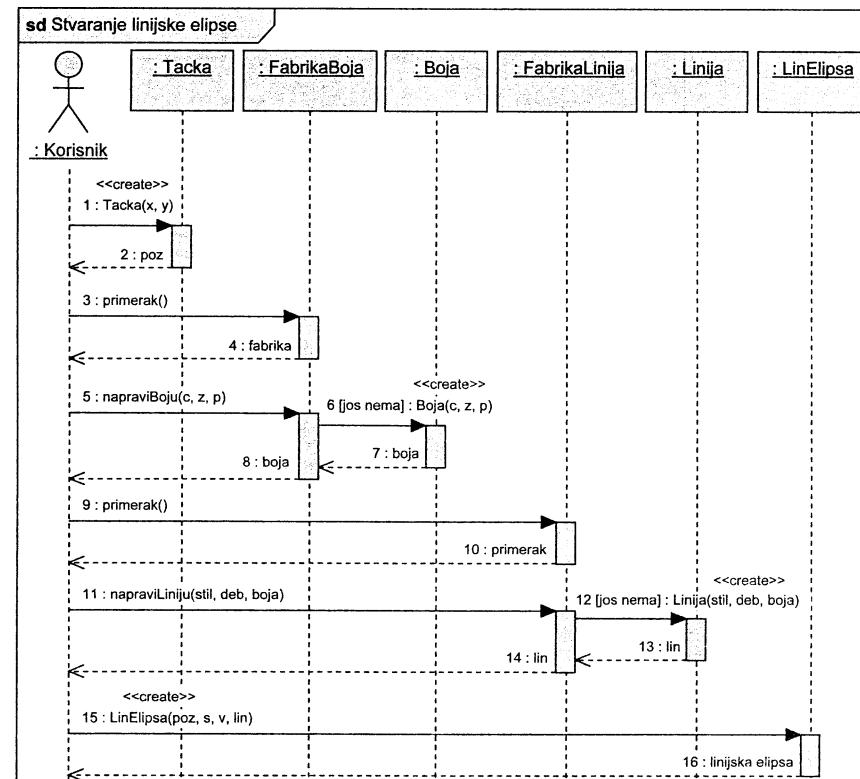
f) Dijagram objekata



g) Dijagram aktivnosti stvaranja linijske elipse



h) Dijagram sekvence stvaranja linijske elipse



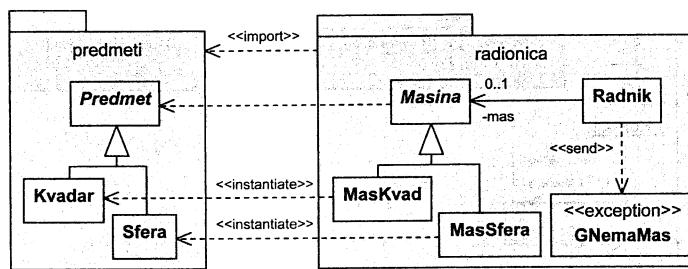
Zadatak 32 Predmeti, mašine i radnik (projektni uzorak *Proizvodna metoda*)

Projektovati na jeziku UML model sledećeg problema:

Apskratna mašina može da proizvodi apstraktan predmet od materijala zadate specifične težine i zna se koliko je predmeta od datog materijala proizvela ta mašina. Može da se dohvati oznaka vrste proizvoda koje dada mašina proizvodi i broj proizvedenih predmeta. Može da se promeni materijal od čega se prave predmeti i da se dohvati specifična težina korišćenog materijala. Mašina za kvadre i mašina za sfere su mašine koje mogu da proizvode kvadre, odnosno sfere zadatih dimenzija. Radnik ima ime i jedinstven, automatski generisan identifikacioni broj. Izrađuje predmete određene vrste pomoću odgovarajuće mašine. Mašina na kojoj radnik radi može da se promeni. Zna se koliko je predmeta radnik izradio na mašini na kojoj trenutno radi. Radnik može da ne bude raspoređen ni na jednu mašinu. U tom slučaju pokušaj izrade predmeta ili dohvatanja broja izrađenih predmeta je greška. Tekstualni oblik radnika sadrži ime i identifikacioni broj radnika. U slučaju da je radnik raspoređen na neku mašinu dodaje se i vrsta predmeta koju trenutno izrađuje i broj proizvoda koje je izradio na toj mašini od početka rada na njoj.

Rešenje:

a) Dijagrami klase



predmeti::Predmet	predmeti::Sfera
- <i>spTez</i> : double <<create>># <i>Predmet</i> (<i>spTez</i> : double) <<create>># <i>Predmet</i> () + <i>vr()</i> : char + <i>V()</i> : double + <i>Q()</i> : double + <i>toString()</i> : String	+ <i>VR</i> : char = 'S' {frozen} - <i>r</i> : double <<create>>+ <i>Sfera</i> (<i>spTez</i> : double, <i>r</i> : double) <<create>>+ <i>Sfera</i> () + <i>vr()</i> : char + <i>V()</i> : double + <i>toString()</i> : String

predmeti::Kvadar
+ <i>VR</i> : char = 'K' {frozen} - <i>a</i> : double - <i>b</i> : double - <i>c</i> : double <<create>>+ <i>Kvadar</i> (<i>spTez</i> : double, <i>a</i> : double, <i>b</i> : double, <i>c</i> : double) <<create>>+ <i>Kvadar</i> () + <i>vr()</i> : char + <i>V()</i> : double + <i>toString()</i> : String

radionica::Masina	radionica::MasSfera
# <i>spTez</i> : double # <i>napravila</i> : int <<create>># <i>Masina</i> (<i>s</i> : double) + <i>vrsta()</i> : char + <i>napravila()</i> : int + <i>uzmiSpTez()</i> : double + <i>postaviSpTez</i> (<i>s</i> : double) + <i>napravi()</i> : Predmet	- <i>r</i> : double <<create>>+ <i>MasSfera</i> (<i>spTez</i> : double, <i>r</i> : double) + <i>r()</i> : double + <i>vrsta()</i> : char + <i>napravi()</i> : Predmet

radionica::MasKvad	radionica::GNemaMas
-a: double -b: double -c: double <<create>>+ <i>MasKvad</i> (<i>spTez</i> : double, <i>a</i> : double, <i>b</i> : double, <i>c</i> : double) + <i>a()</i> : double + <i>b()</i> : double + <i>c()</i> : double + <i>vrsta()</i> : char + <i>napravi()</i> : Predmet	<<exception>> - <i>ime</i> : String - <i>id</i> : int <<create>>+ <i>CNemaMas</i> (<i>ime</i> : String, <i>id</i> : int) + <i>ime()</i> : String + <i>id()</i> : int + <i>toString()</i> : String

b) Kôd na jeziku Java koji je generisao StarUML

```

// @ File Name : Predmet.java
package predmeti;
public abstract class Predmet {
    private double spTez;
    protected void Predmet(double spTez) {}
    public abstract char vr();
    public abstract double V();
    public final double Q() {} ← Potvrđena je opcija IsLeaf.
    public String toString() {}
}
  
```

```
// @ File Name : Sfera.java
package predmeti;
public class Sfera extends Predmet {
    public static final char VR = 'S'; ← Odabrano je: Changeability=FROZEN.
    private double r;
    public void Sfera(double spTez, double r) {}
    public void Sfera() {}
    public char vr() {}
    public double V() {}
    public String toString() {}
    public char vr() {}
    public double V() {}
}
```

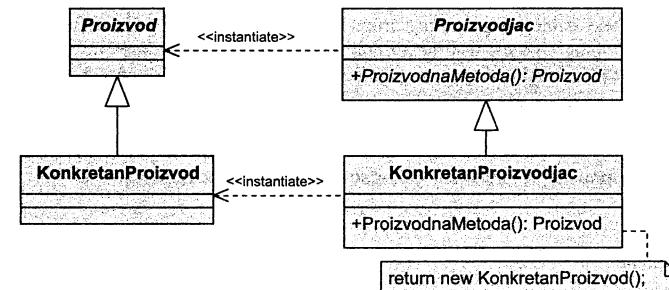
```
// @ File Name : Masina.java
package radionica;
import predmeti.*;
public abstract class Masina {
    protected double spTez;
    protected int napravila;
    protected void Masina(double s) {}
    public abstract char vrsta();
    public int napravila() {}
    public double uzmiSpTez() {}
    public void postaviSpTez(double s) {}
    public abstract Predmet napravi();
}
```

```
// @ File Name : MasSfera.java
package radionica;
import predmeti.*;
public class MasSfera extends Masina {
    private double r;
    public void MasSfera(double spTez, double r) {}
    public double r() {}
    public char vrsta() {}
    public Predmet napravi() {}
    public char vrsta() {}
    public Predmet napravi() {}
}
```

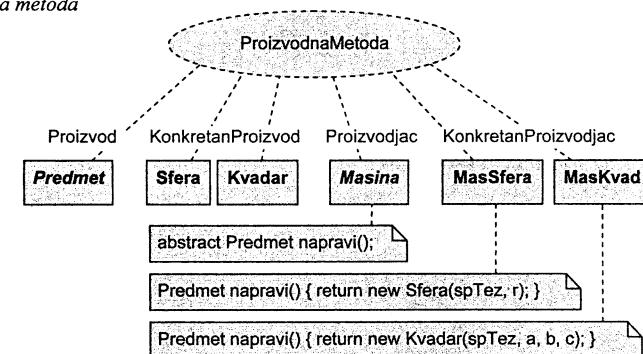
```
// @ File Name : Radnik.java
package radionica;
import predmeti.*;
public class Radnik {
    private static int posId;
    private int id = ++posId;
    private String ime;
    private int napravio;
    private Masina mas;
    public void Radnik(String ime) {}
    public int id() {}
    public String ime() {}
    public Radnik rasporedi(Masina m) {}
    public Masina dohvatiMas() {}
    public int napravio() throws GNemaMas {} ← Ne generiše se klasa za izuzetke,
    public Predmet napravi() throws GNemaMas {} ali se dodaje klauzula throws:
    public String toString() {}
}
```

c) Projektni uzorak **Proizvodna metoda (Factory Method)**

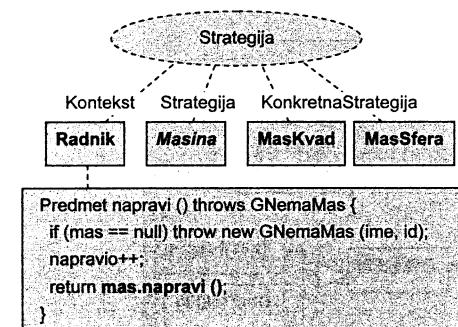
- klasni uzorak stvaranja
- predviđa interfejs za stvaranje objekata
- potklase proizvodača definišu metodu koja proizvodi konkretnе objekte



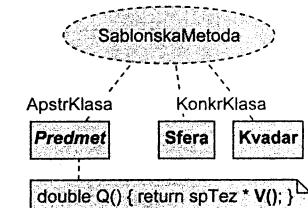
d) Primer uzorka **Proizvodna metoda**



e) Primer uzorka **Strategija**



f) Primer uzorka **Šablonska metoda**



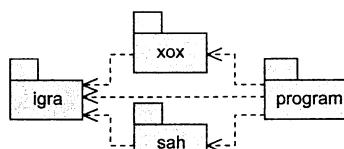
Zadatak 33 Igre (projektni uzorci Podsetnik i Komanda)

Projektovati na jeziku UML model sledećeg problema:

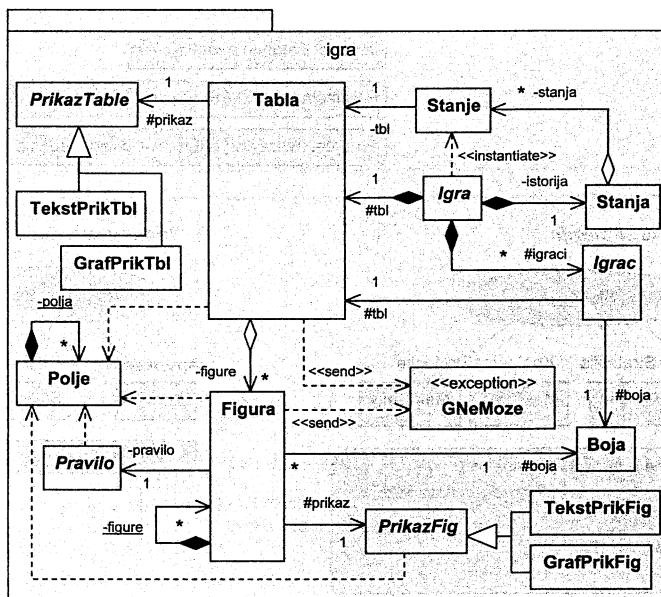
Figura za igre ima boju i može da se premešta na novo mesto na tabli za igre, uz proveravanje ispravnosti poteza u zavisnosti od igre i prirode figure. Polja dvodimenzionalne table za igre mogu biti prazna ili da sadrže jednu figuru. Figure na tabli mogu da se postavljaju, premeštaju i uklanjanju. Sadržaj table može da se prikazuje pomoću tekstualnog ili grafičkog prikazivača. Apstraktan igrač igra figurama zadate boje na zadatoj tabli. Može da odigra potez. Kod nekih igara ponekad može da dobije pravo na još jedan potez. Apstraktne igre sadrži jednu tablu za igru i nekoliko igrača. Stvara se s praznom tablom i popunjениm nizom igrača. Može da se odigra jedna partija koja se sastoji od postavljanja početnog stanja igre i povlačenja poteza igrača po cikličnom redosledu do završetka igre. Partija može pre vremena da se prekine i da se napravi korak unazad proizvoljan broj puta. XoX i šah su igre koje igraju dva igrača na tabli 3×3 , odnosno 8×8 . Igram se upravlja programom koji koristi grafičku korisničku površ.

Rešenje:

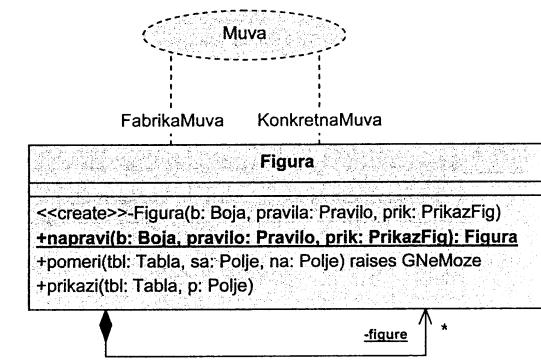
a) Dijagram paketa



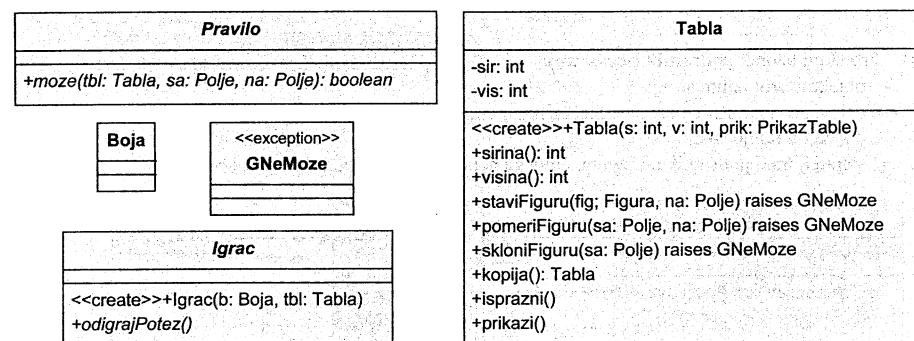
b) Dijagram klasa paketa igra



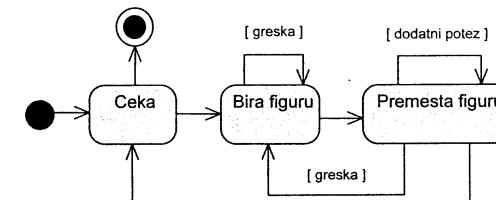
c) Primeri projektnog uzorka Muva



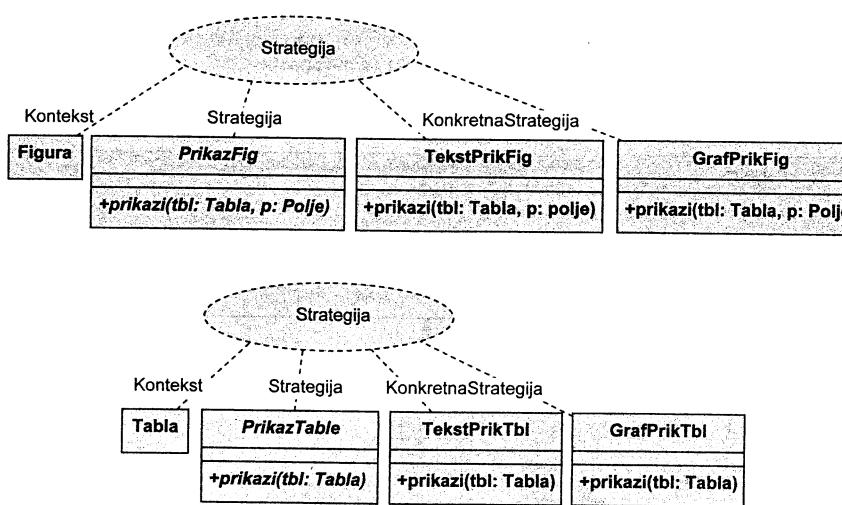
d) Ostale klase paketa igra



e) Dijagram stanja igrača

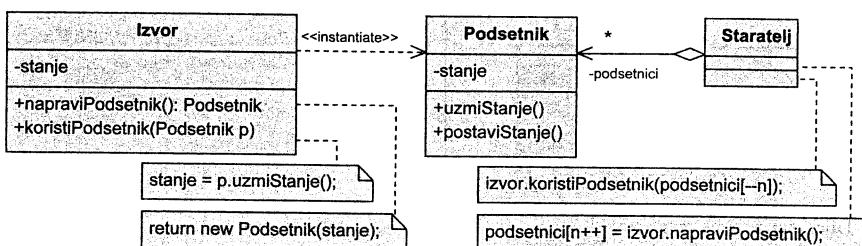


f) Primeri uzorka Strategija

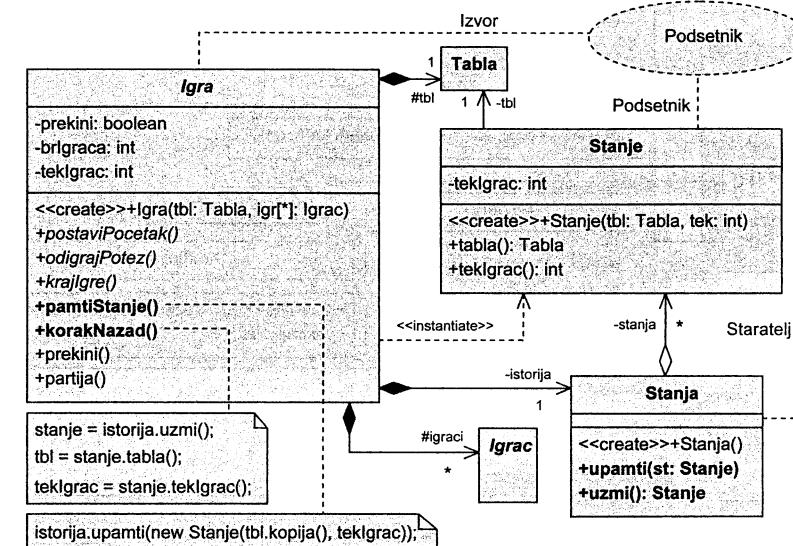


g) Projektni uzorak podsetnik (memento)

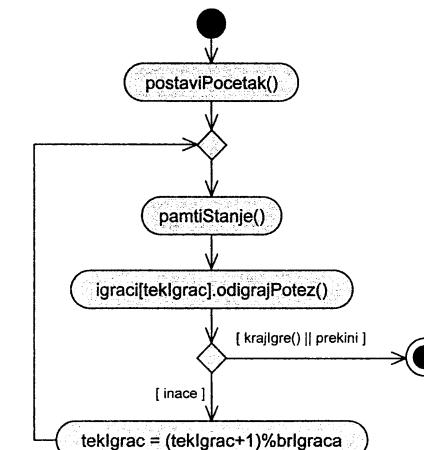
- objektni uzorak ponašanja
- snimanje unutrašnjeg stanja objekta bez narušavanja učaurivanja podataka radi kasnijeg vraćanja objekta u ranije stanje
- staratelj traži podsetnik od izvora, čuva ga neko vreme i po potrebi vraća izvoru



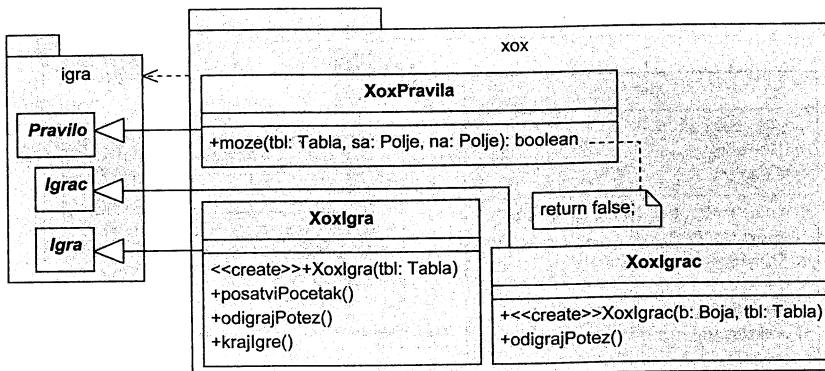
h) Primer uzorka Podsetnik



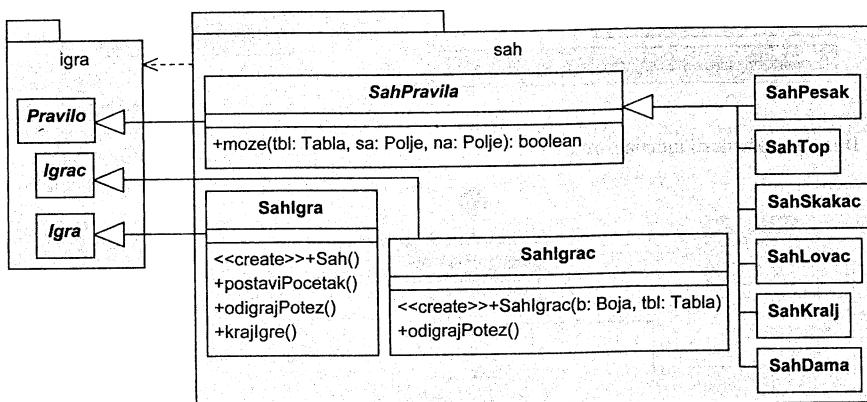
i) Dijagram aktivnosti igranja partie



j) Paket xox

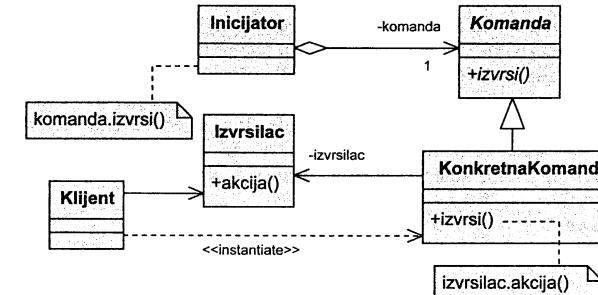


k) Paket sah

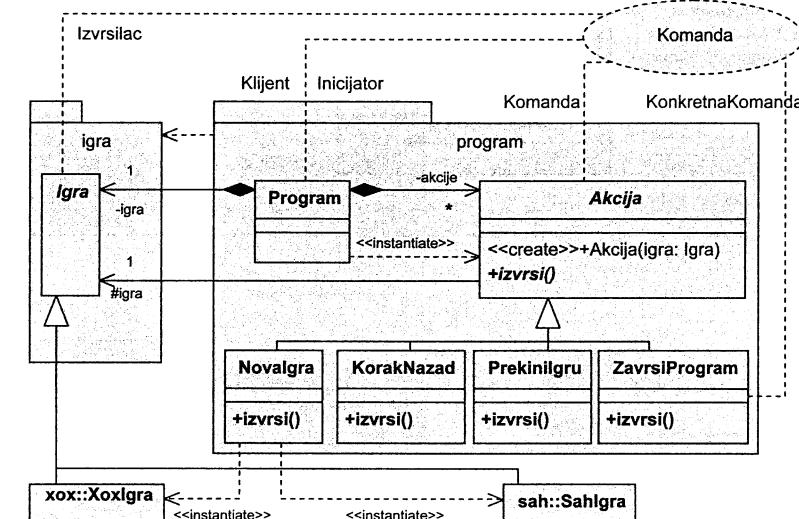


l) Projektni uzorak Komanda (Command)

- objektní vzorok ponašanja
- razdvájí izvršioca radnje od inicijatora omogućujući parametrizaciju klijenta, stavljanje zahteva u redove, vođenje dnevnika i poništavanje dejstva operacija
- klijent pravi objekat komande koji prosleduje zahtev inicijatoru izvršiocu



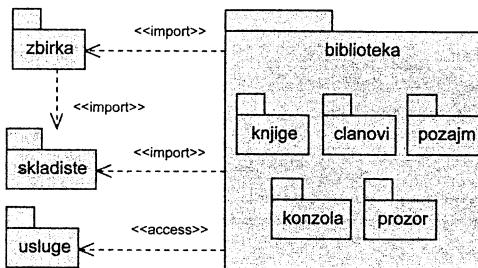
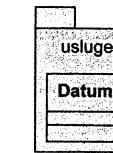
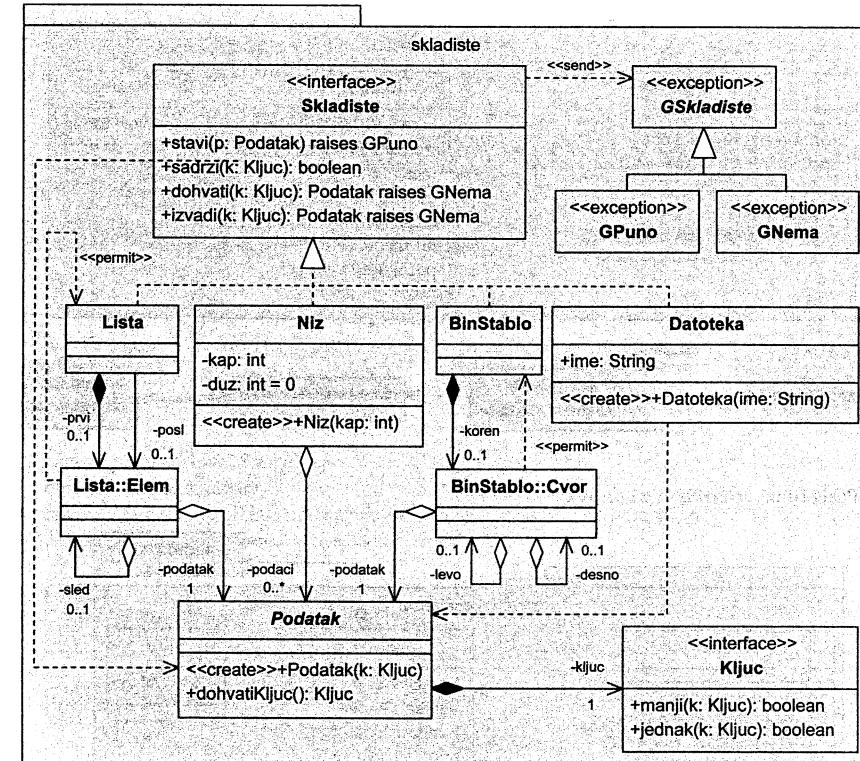
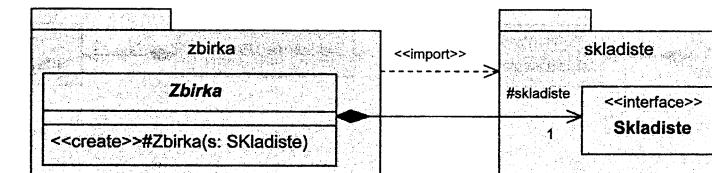
m) Primer uzorka Komanda u paketu program



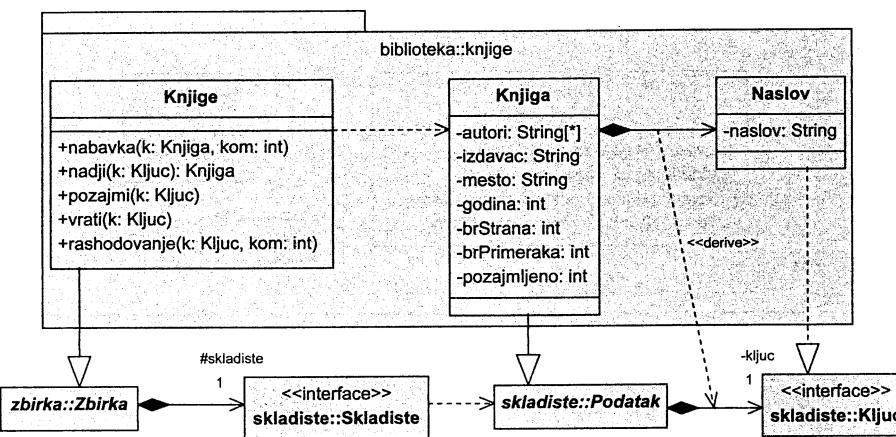
Zadatak 34 Biblioteka (dijagram komponenata, projektni uzorci *Most* i *Abstraktna fabrika*)

Projektovati na jeziku UML model sledećeg problema:

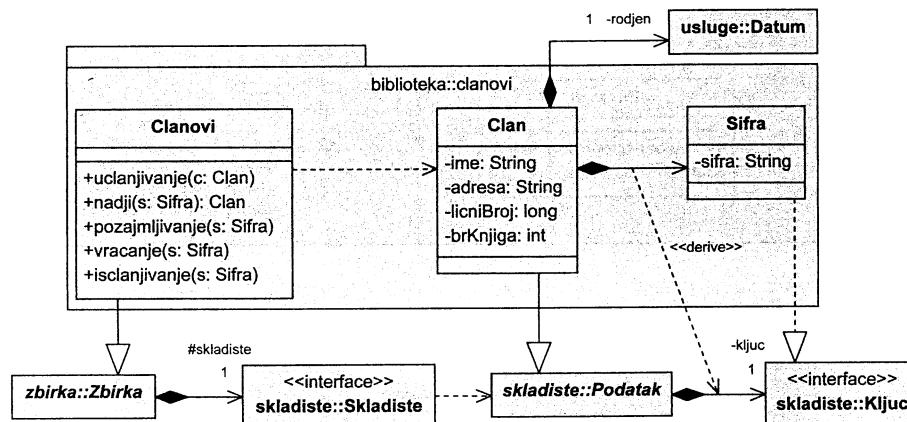
Apstraktno skladište može da sadrži apstraktne podatke koji se identifikuju pomoću apstraktнog ključa. Podaci u skladištu uređeni su po veličini ključeva. Može da se stavi neki podatak u skladište, da se ispita da li postoji podatak s datim ključem, da se podatak s datim ključem dohvati i da se izvadi. Niz ograničenog kapaciteta, lista, binarno stablo i datoteka su skladišta. Apstraktna zbirka sadrži skladište čiji sadržaj obrađuje. Biblioteka sadrži zbirku knjiga, zbirku članova i zbirku pozajmljivanja. Knjige se identifikuju na osnovu naslova. Za svaki naslov postoji jedna stavka u kojoj, pored opštih podataka o knjizi, postoji broj primeraka koje biblioteka poseduje i koliko je od njih trenutno pozajmljeno. Članovi se identifikuju pomoću šifre članske karte. Stavke o pozajmljivanju se identifikuju jedinstvenim automatski generisanim identifikacionim brojem i sadrže datum uzimanja knjige i rok za vraćanje u danima. Postoje grupe obrada za knjige, članove i pozajmljivanja koje mogu da se izvode u obliku dijaloga u tekstualnom režimu ili korišćenjem prozora u grafičkom režimu.

Rešenje:
a) Dijagram paketa

b) Paket usluge

c) Paket skladiste

d) Paket zbirka


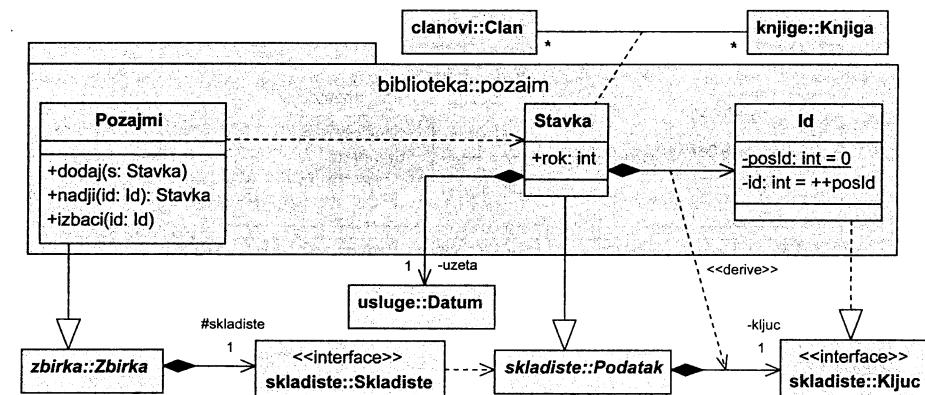
e) Paket biblioteka::knjige



f) Paket biblioteka::clanovi

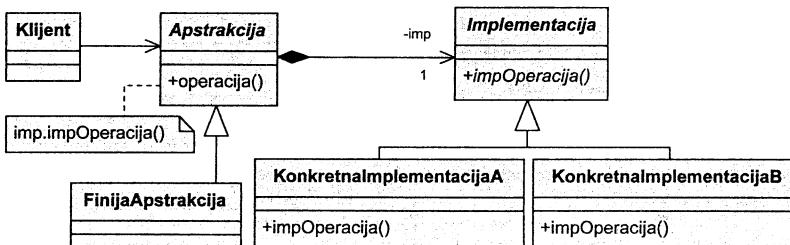


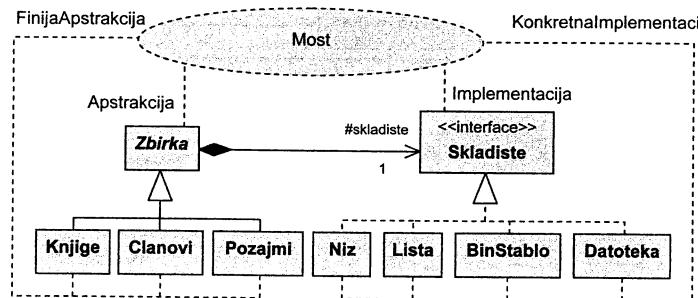
g) Paket biblioteka::pozajm



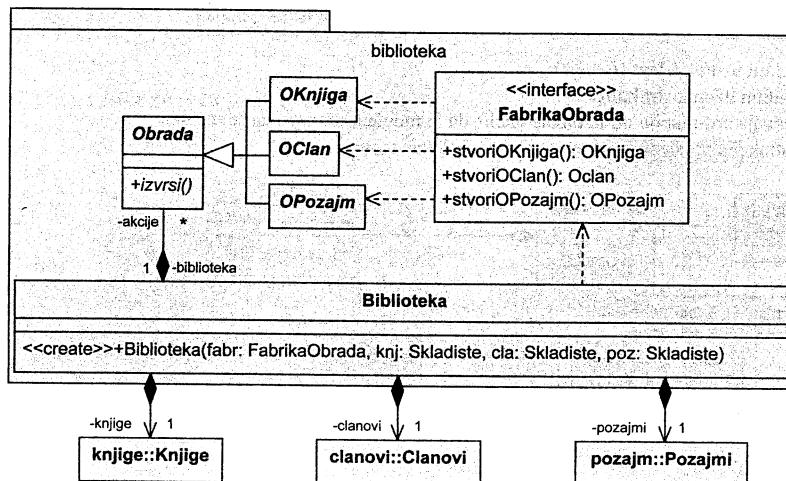
h) Projektni uzorak Most (Bridge)

- objektni uzorak strukture
- razdvaja apstrakciju od implementacije da bi mogle nezavisno da se menjaju
- apstrakcija prosleđuje zahteve klijenta implementaciji

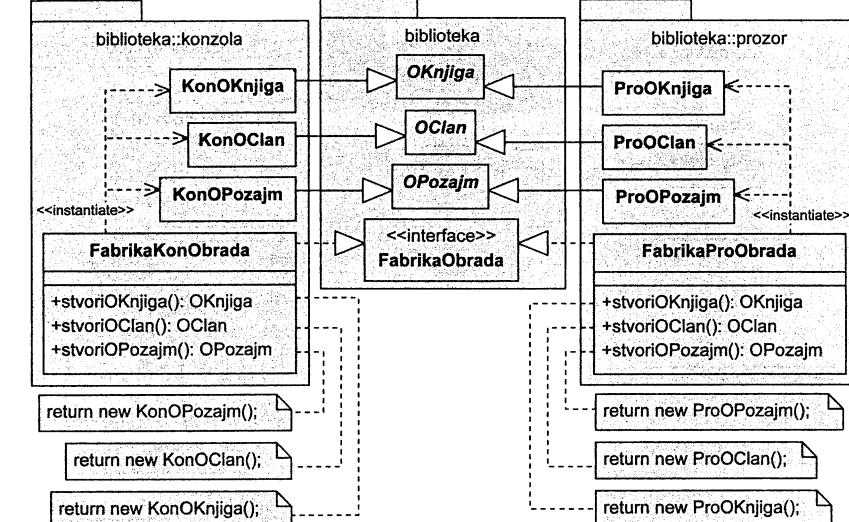


i) Primer uzorka *Most*

j) Paket biblioteka



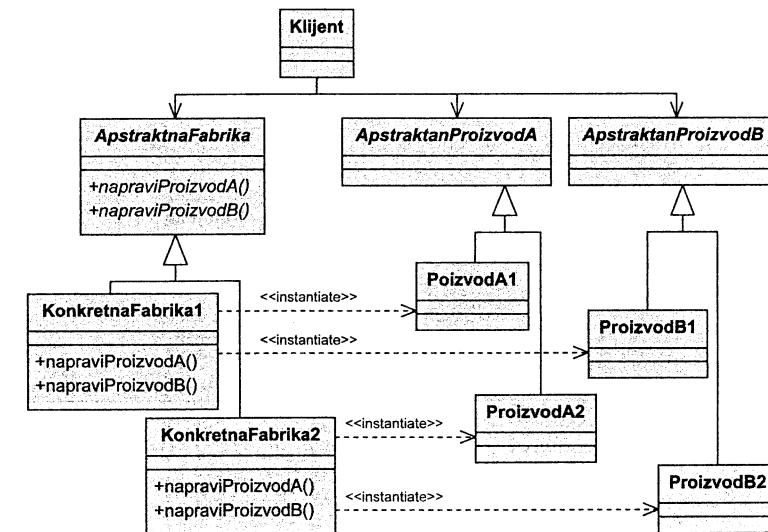
k) Paket biblioteka::konzola

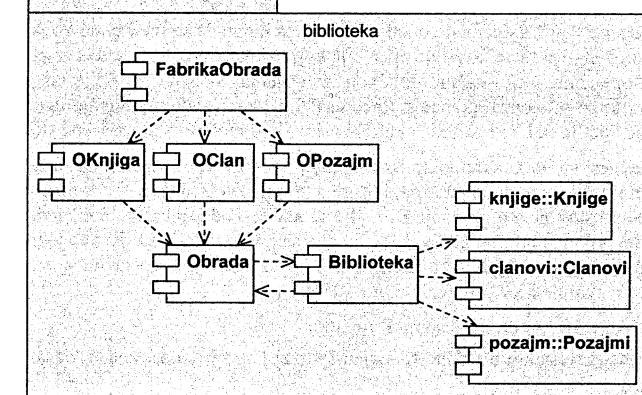
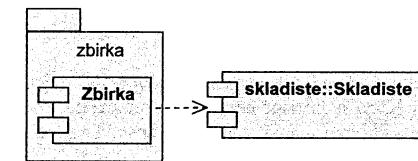
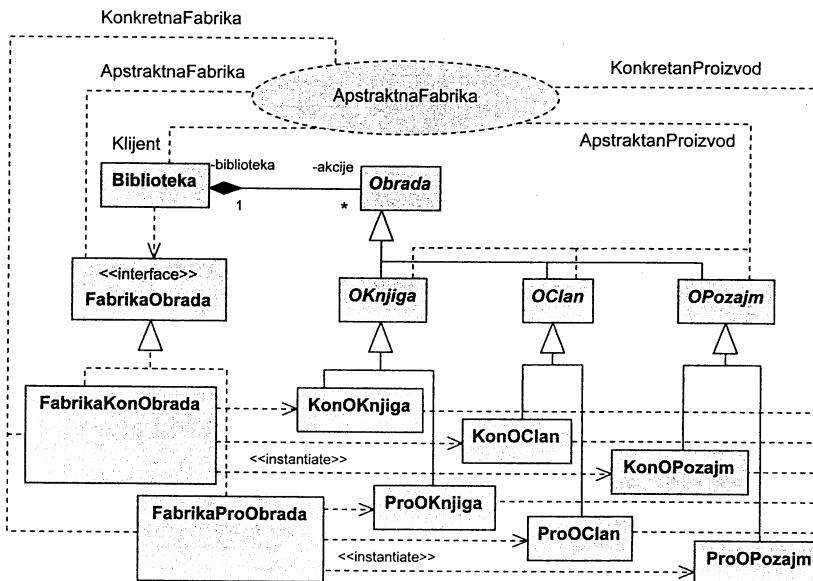


l) Paket biblioteka::prozor

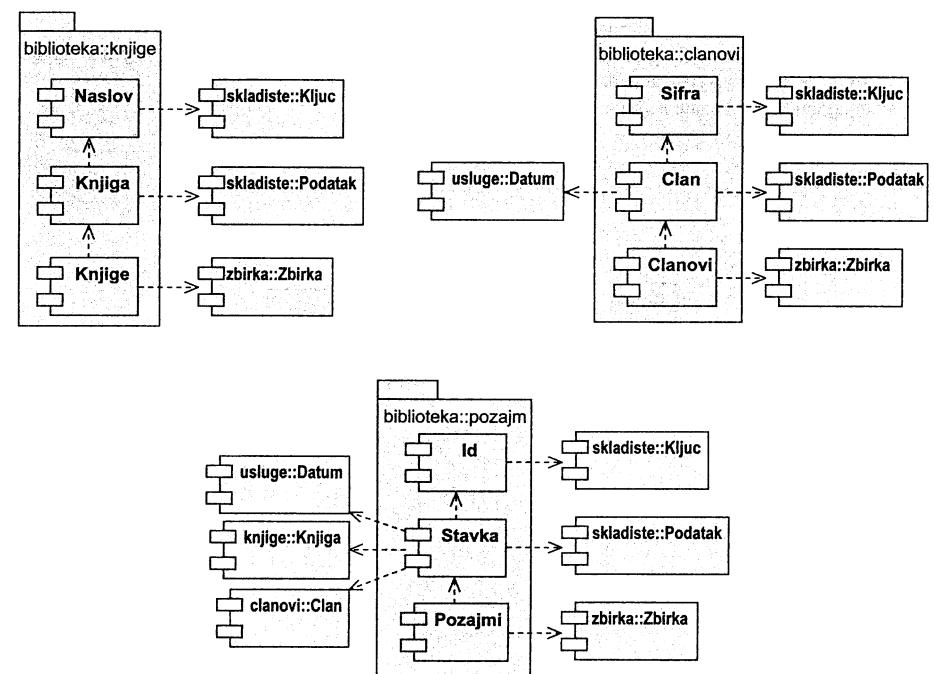
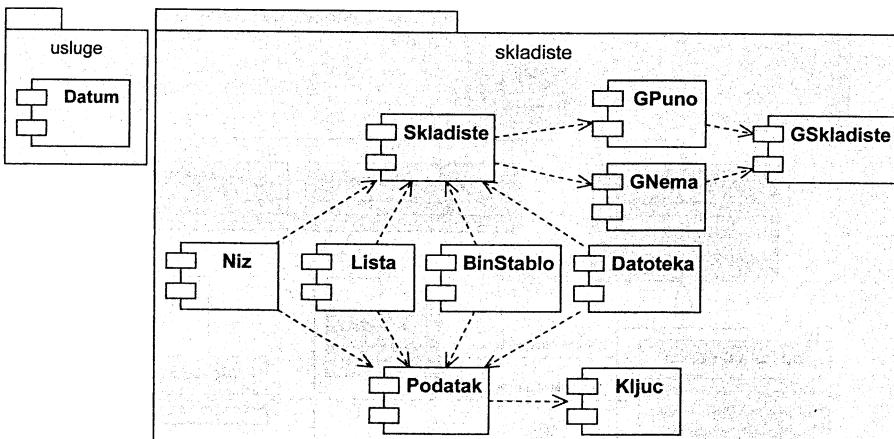
m) Projektni uzorak *Apstraktna fabrika* (*Abstract Factory*)

- objektni uzorak stvaranja
- omogućuje stvaranje objekata određene grupe tipova po izboru
- klijent pravi proizvode iz određene grupe vrsta proizvoda pomoću odgovarajuće fabrike



n) Primer uzorka *Apstraktna fabrika*

o) Dijagrami komponenata



Zadatak 35 Uredaj, operacije i API (projektni uzorak *Fasada*) {1, 14.02.2007.}

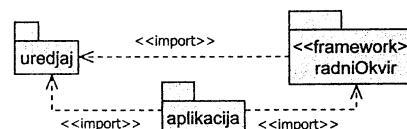
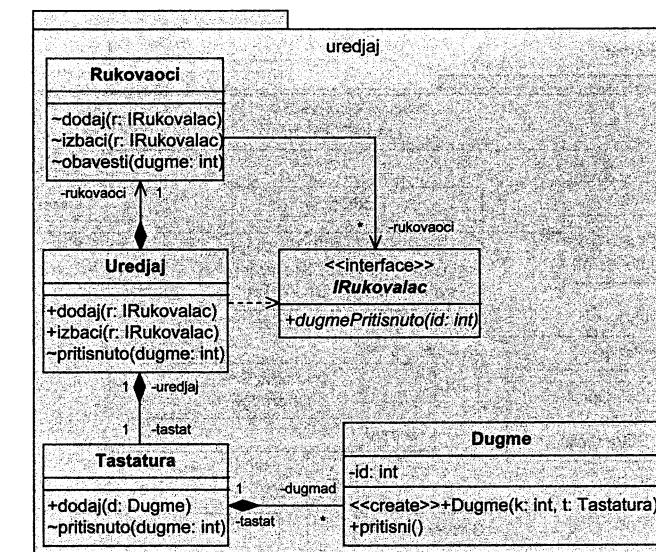
Uredaj sadrži tastaturu funkcijskih dugmadi gde svako dugme ima pridruženu posebnu operaciju u svakoj aplikaciji (režimu rada uređaja). Kada se pritisne neko dugme uređaj obavesti sve registrovane rukovaoce događaja. Rukovaoci implementiraju interfejs koji predviđa dostavljanje celobrojnog identifikatora pritisknutog dugmeta.

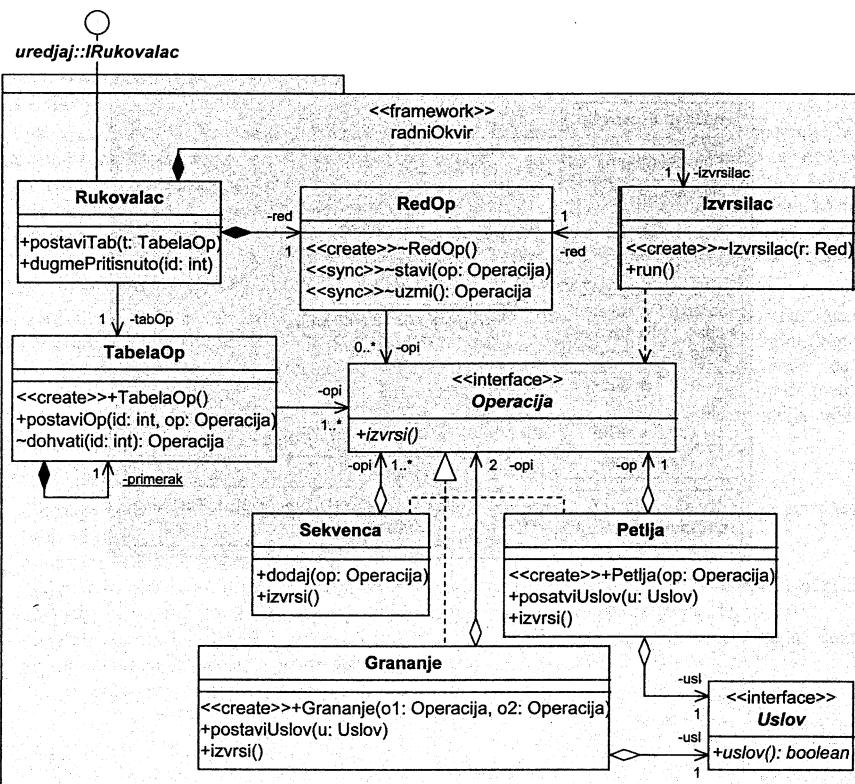
Rukovalac na osnovu celobrojnog identifikatora dugmeta i tabele preslikavanja određuje operaciju koju treba izvršiti i smešta je u red čekanja. Operacije koje je smestio jedan rukovalac izvršavaće se sekvencialno, po redosledu smeštanja, a operacije raznih rukovalaca mogu da se izvršavaju konkurentno. Svakom redu operacija pridružen je aktivan izvršilac operacija koji uzima operacije iz reda i izvršava ih ne znajući ništa o prirodi tih operacija, niti o načinu njihovog izvršavanja. Složene operacije nisu zavisne od konkretnih aplikacija i to su sekvenca, grananje i petlja. Petlja sadrži jednu, grananje dve, a sekvenca više operacija. Grananje i petlja sadrže i po jedan logički uslov, koji može biti proizvoljne složenosti.

Proste operacije zavisne su od konkretnе aplikacije, tako da se mora obezbediti da se za jednu aplikaciju kreira konzistentan skup operacija od interesa, koji se u vreme izvršavanja može zameniti drugim skupom operacija. Konkretan uslov nekog grananja ili petlje je zavisan od konkretnе aplikacije, a može i da se promeni u vreme izvršavanja programa. Svaka operacija se izvršava tako što poziva jednu ili više metoda aplikativnog programskega interfejsa. Aplikativni programski interfejs predstavlja skup operacija aplikativne logike koja se ostvaruje kroz proizvoljan broj klasa.

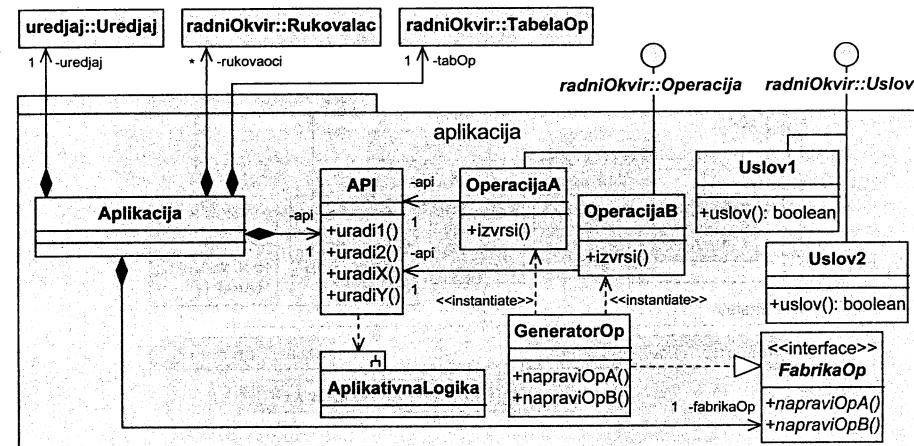
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa razvrstanih u pakete (detaljan sadržaj klasa i odnose među klasama na potrebnom broju slika),
- prikaz korišćenih projektnih uzoraka,
- dijagram objekata koji prikazuje uređaj sa dva dugmeta, red koji sadrži jednu sekvencu s nekoliko operacija i izvršioca koji izvršava petlju koja sadrži grananje s dve proste operacije,
- dijagram sekvence i dijagram aktivnosti za ceo scenario koji usledi nakon pritiska na jedno dugme kojim se zahteva izvršavanje proste operacije,
- dijagram komponenata u vreme izvršenja.

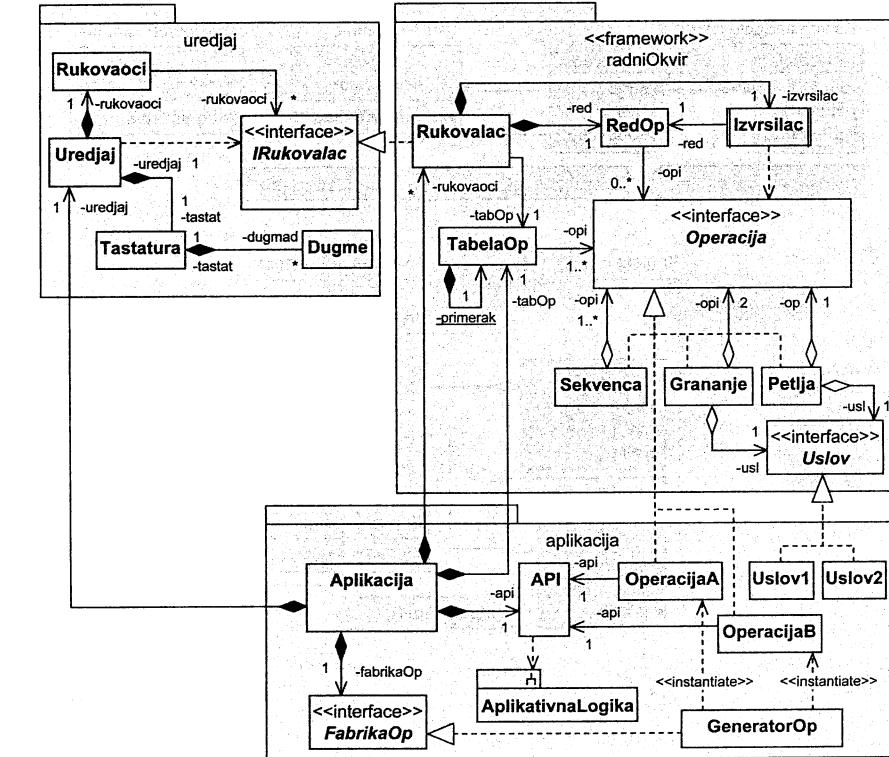
Rešenje:
a) Dijagram paketa

b) Dijagram klasa paketa *uredaj*


c) Dijagram klasa paketa `radniOkvir`

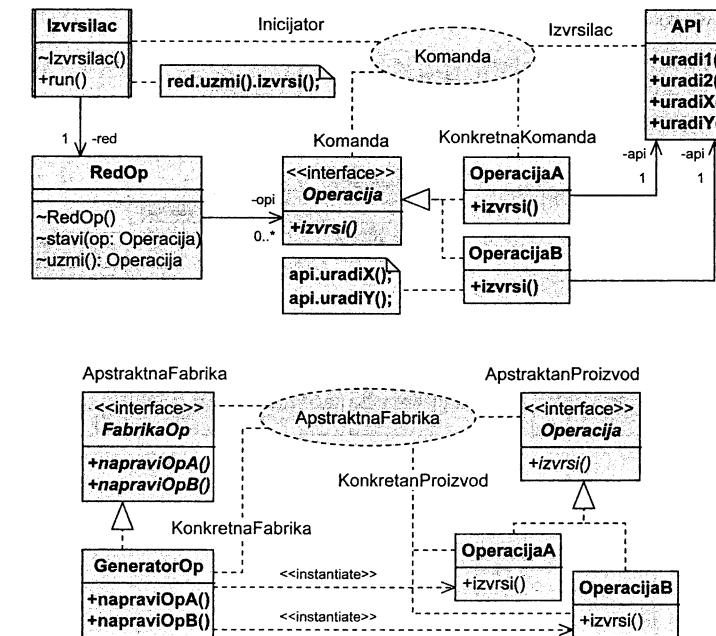
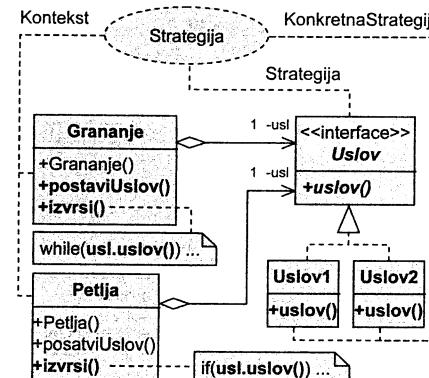
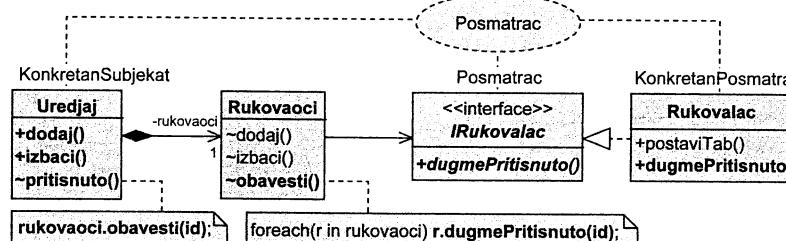
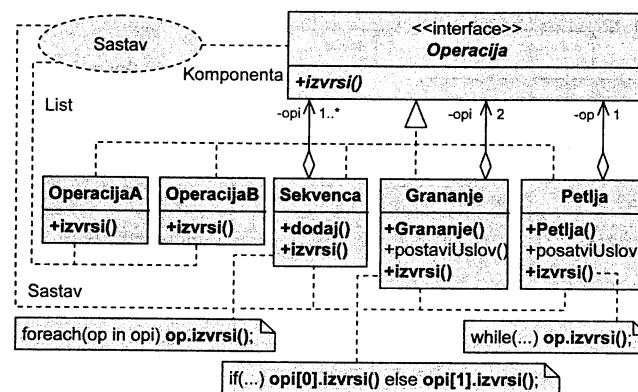
d) Dijagram klasa paketa aplikacija



e) Pregledni dijagram klasa

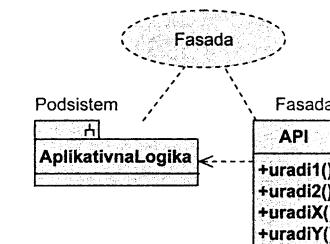
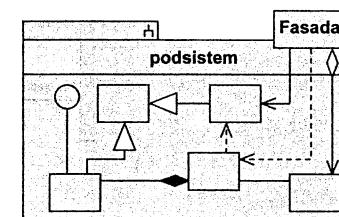


f) Projektni uzorci

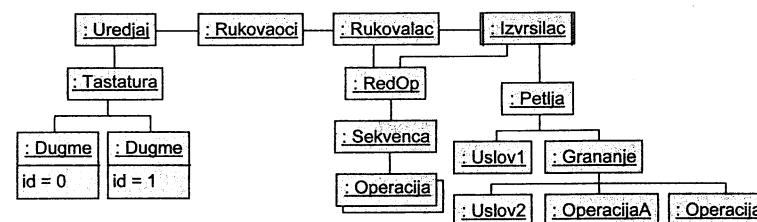


g) Projektni uzorak Fasada (Facade)

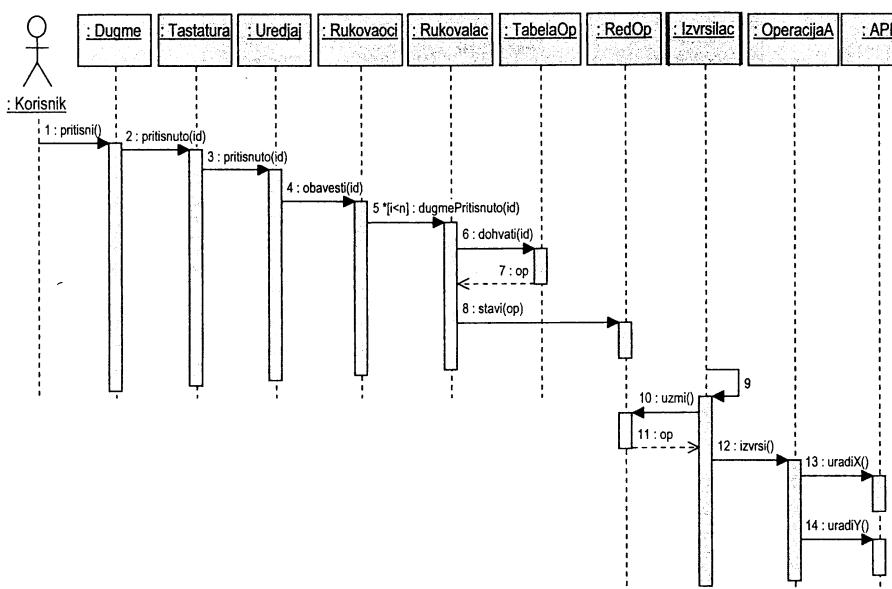
- objektni uzorak strukture
- daje interfejs višeg nivoa za skup interfejsa jednog podistema
- klijent podistem koristi preko fasade



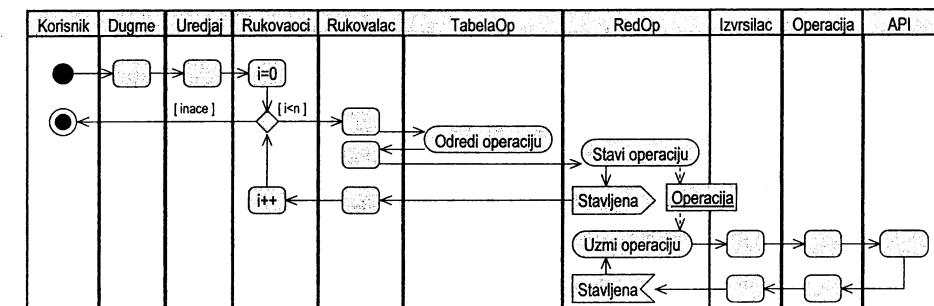
h) Dijagram objekata



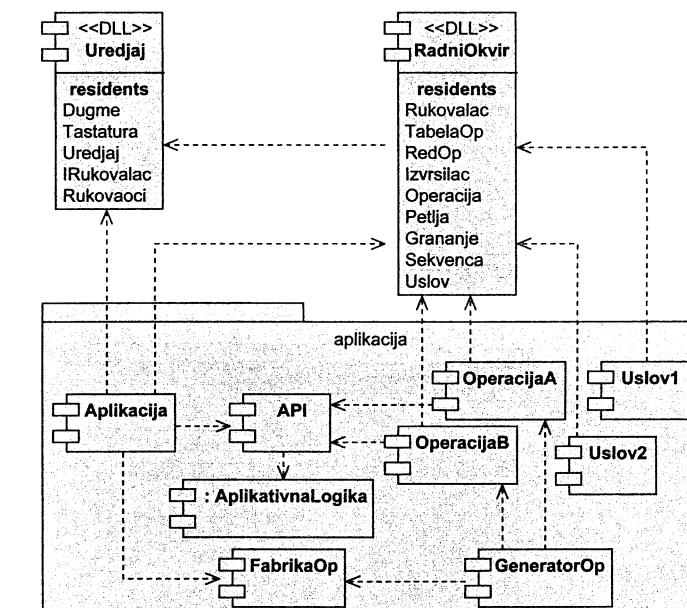
i) Dijagram sekvencije prilikom pritiska na dugme



j) Dijagram aktivnosti prilikom pritiska na dugme



k) Dijagram komponenta u vreme izvršenja



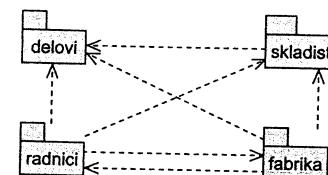
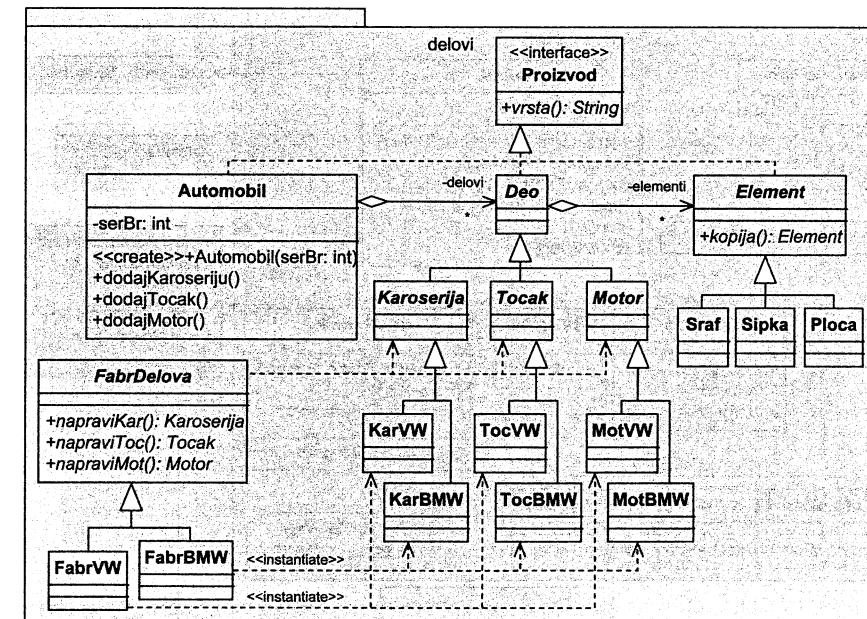
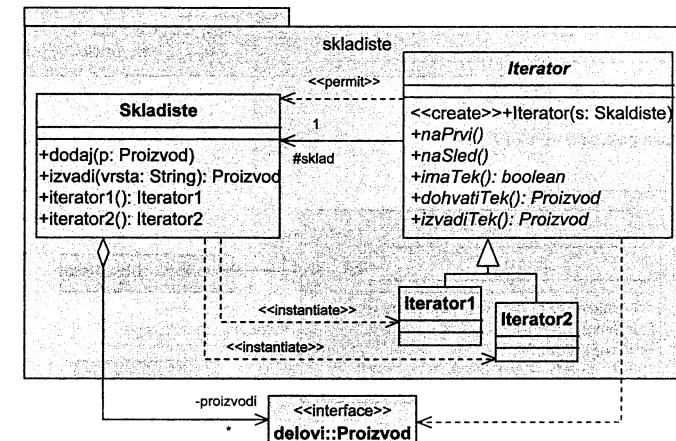
Zadatak 36 Fabrika, radnici, skladište i automobili (projektni uzorak *Graditelj*) {1, 06.02.2009.}

Proizvodu može da se dohvati vrsta. Automobil, deo i element su proizvodi. Automobil se sastoji od delova, a delovi od elemenata. Automobil ima serijski broj. Karoserija, točak i motor su delovi. Šraf, šipka i ploča su elementi. Skladište sadrži proizvoljan broj proizvoda. Može da se dodaje jedan proizvod i da se proizvod zadate vrste izvadi iz skladišta. Sadržaj skladišta može da se pretražuje na više načina da bi se pronašla željena vrsta proizvoda. Aktivan radnik ima ime, može da mu se dohvati kvalifikacija, može da se zaposli u nekoj fabriči i može da proizvodi proizvode koje stavlja u zadato skladište. Postoje nekvalifikovani, kvalifikovani i visokokvalifikovani radnici. Nekvalifikovani radnik proizvodi elemente na osnovu zadatog originala. Kvalifikovani radnik proizvodi delove od elemenata iz skladišta. Specijalizovani kvalifikovani radnici proizvode karoserije, točkove ili motore. Delovi se različito proizvode za različite marke automobila. Visokokvalifikovani radnik proizvodi automobile tako što im ugradi karoseriju, četiri točka i motor iz skladišta. Fabrika zapošjava radnike, poseduje skladište i proizvodi automobile marke koja se određuje prilikom stvaranja fabrike.

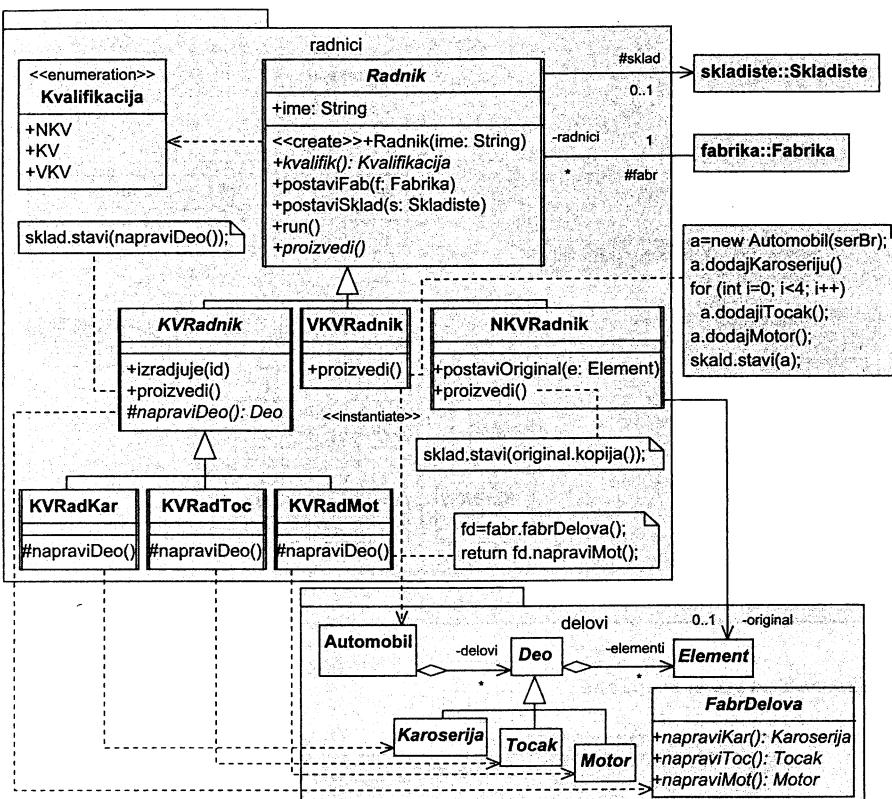
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klase razvrstavši ih u pakete (detaljan sadržaj klasa i odnose među klasama na potrebnom broju slika),
- prikaz korišćenih projektnih uzoraka,
- dijagram sekvence za proizvodnju automobila,
- dijagram komponenata stavljući klase koje čine logičke celine u istu komponentu.

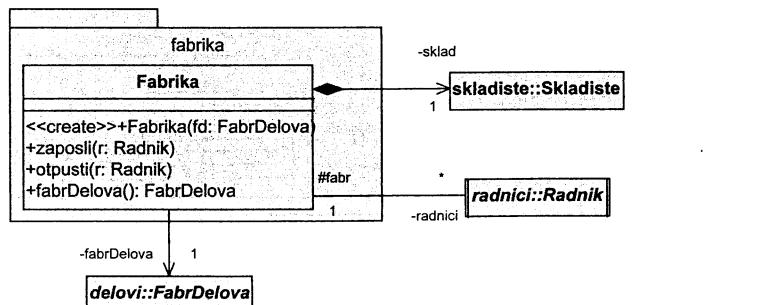
Rešenje:

a) Dijagram paketa

b) Dijagram klasa paketa delovi

c) Dijagram klasa paketa skadiste


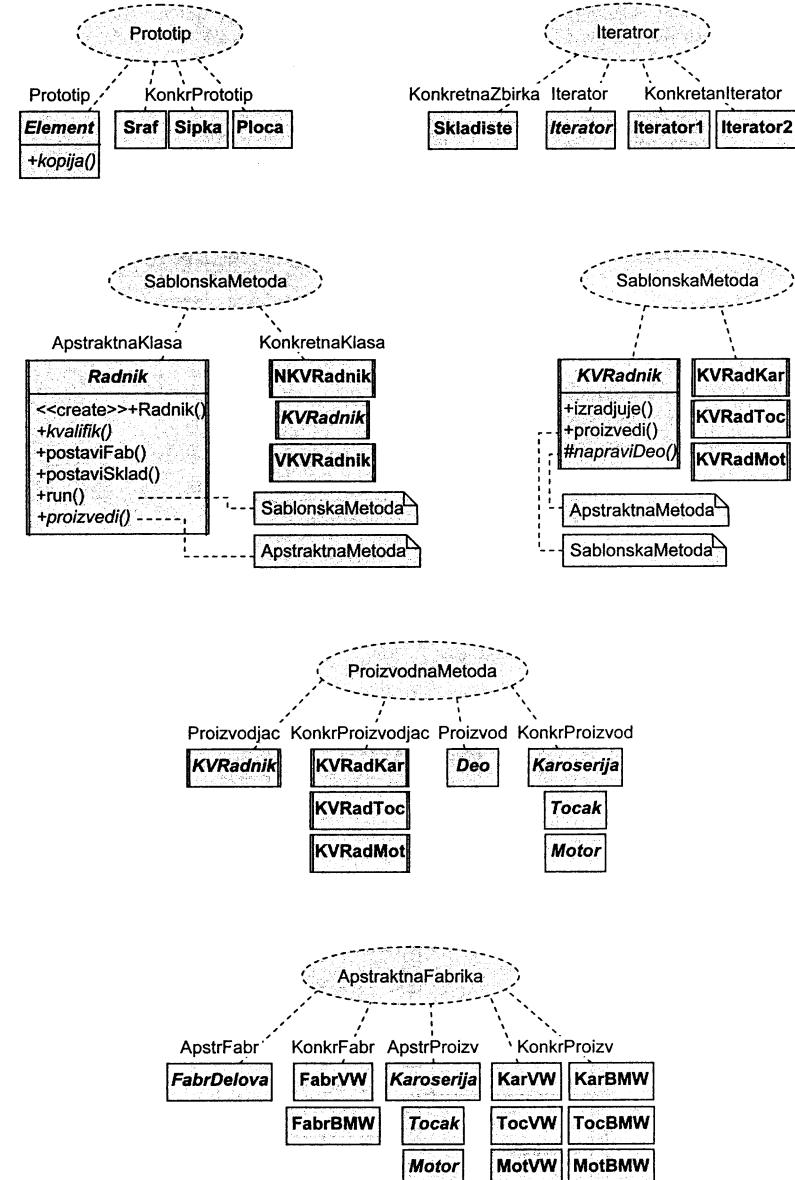
d) Dijagram klasa paketa radnici



e) Dijagram klasa paketa fabrika

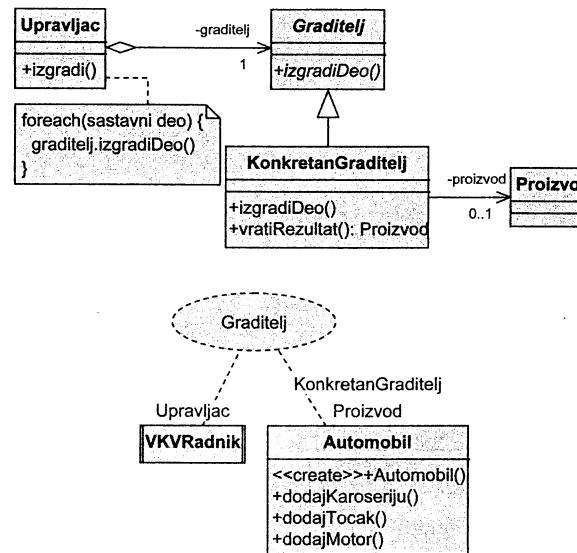


f) Projektni uzorci

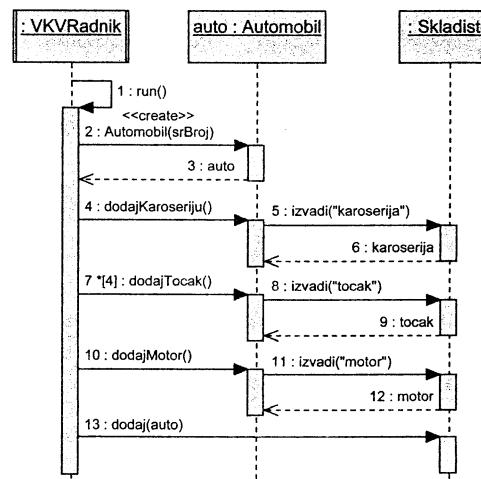


g) Projektni uzorak **Graditelj (Builder)**

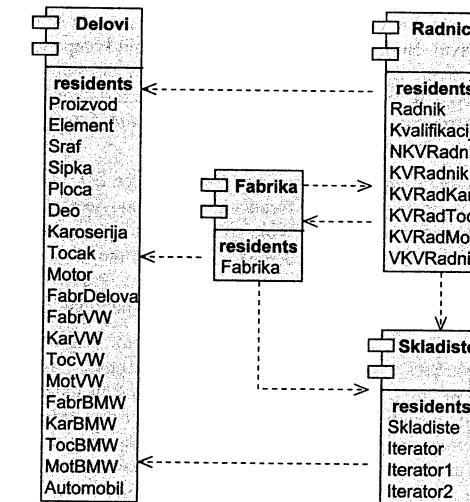
- objektni uzorak stvaranja
- razdvaja izgradnju složenog objekta od reprezentacije
- graditelj konstruiše proizvod korak po korak pod kontrolom upravljača



h) Dijagram sekvencije izgradnje automobila



i) Dijagram komponenata



Zadatak 37 Tačka, boja, elementi, crtači, komponente, akteri i program {l, 05.02.2007.}

Tačka u ravni zadaje se pomoću dve celobrojne koordinate koje mogu da se dohvate. Tačka može da se premesti na drugo mesto u ravni. Boja se zadaje pomoću celobrojnih komponenata crvene, zelene i plave boje u opsegu od 0 do 255. Crtač predviđa postavljanje boje pera za crtanje i pravolinjsko pomeranje pera iz trenutnog položaja do zadate tačke bez crtanja i s crtanjem. Ciljna tačka postaje nova trenutna tačka. Konkretni crtači ostvaruju crtanje pod konkretnim operativnim sistemom, koji može biti *Windows* ili *Linux*.

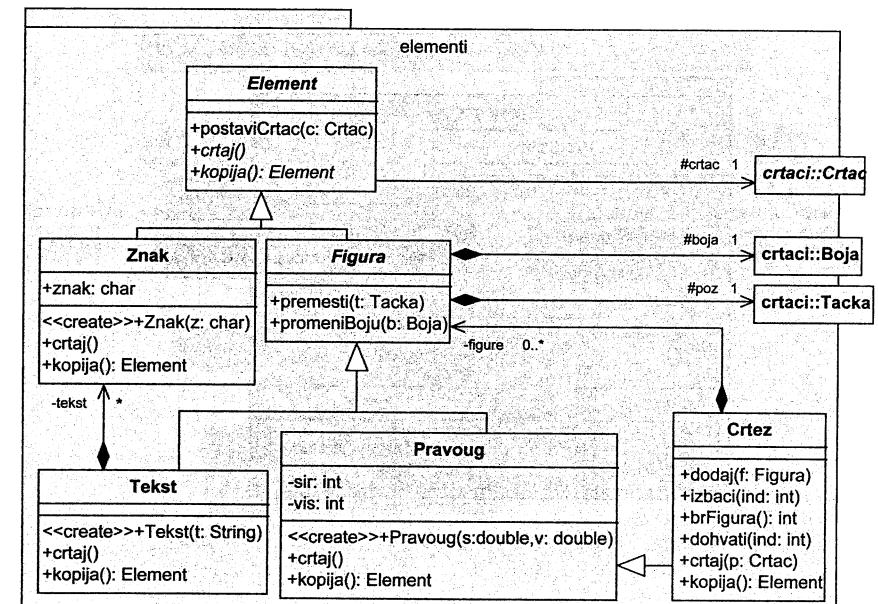
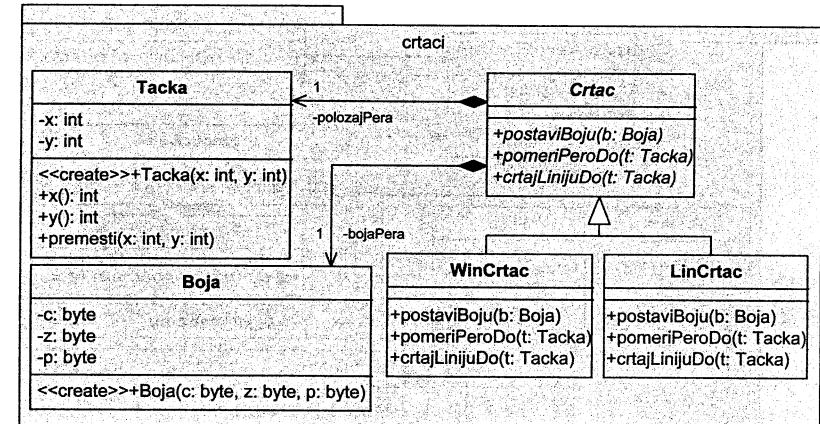
Grafički element predviđa crtanje elementa i pravljenje kopije elementa. Crtanje se ostvaruje pomoću objekta crtača koji može da se postavlja u toku života elementa. Znak je element koji sadrži podatak tipa char. Figura je element koji sadrži boju i tačku koja predstavlja položaj gornjeg levog temena. Stvara se bela sa temenom u koordinatnom početku, posle čega ti parametri mogu da se promene. Tekst je figura koja sadrži niz znakova. Stvara se na osnovu podatka tipa *String*. Pravougaonik je figura zadate celobrojne širine i visine. Crtež je pravougaonik koji može da sadrži proizvoljan broj figura.

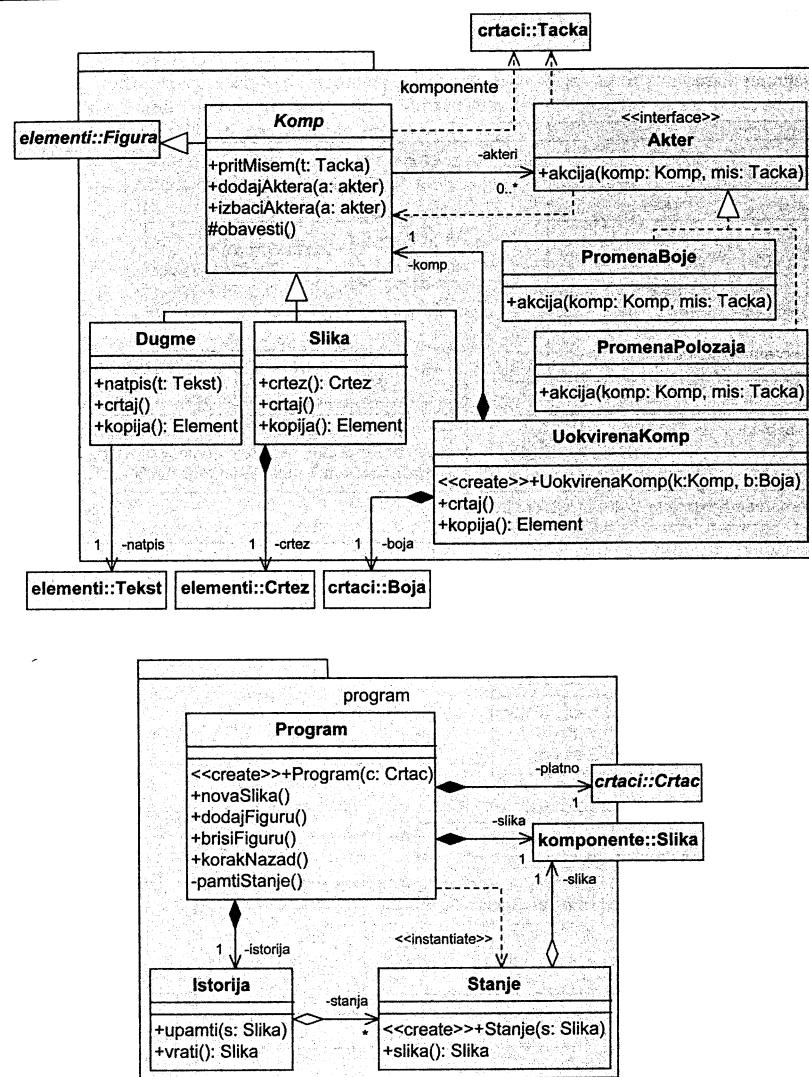
Komponenta je figura koja može da obradi događaj pritiska mišem. Dugme je komponenta koja sadrži tekst koji čini njegov natpis. Stvara se s praznim natpisom koji posle može da se postavlja. Slika je komponenta koja sadrži jedan crtež. Stvara se s praznim crtežom koji može da se dohvati radi kasnije dorade. Komponente mogu da se uokvire linijom zadate boje. Akter predviđa izvršavanje neke akcije kad se neka komponenta pritisne mišem. Tom prilikom komponenta saopštava akteru koja je komponenta pritisnuta i dostavlja tačku koja predstavlja položaj miša. Akteri mogu da se prijavljuju i odjavljaju kod komponenata radi dobijanja obaveštenja o pritiscima mišem. Promena boje i promena položaja komponente jesu mogući akteri. Ne treba razrađivati detalje izvođenja ovih akcija.

Program obezbeđuje interaktivno crtanje jedne slike korišćenjem crtača koji se zadaje prilikom stvaranja programa. Program podržava početak sastavljanja nove slike, dodavanje figure, izbacivanje figure i vraćanje proizvoljan broj koraka unazad u postupku sastavljanja slike.

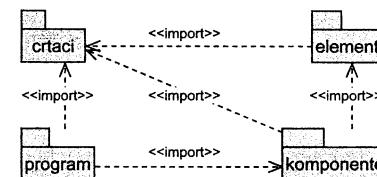
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa razvrstavi ih u pakete (detaljan sadržaj klasa i odnose među klasama na potrebnom broju slika),
- prikaz korišćenih projektnih uzoraka,
- dijagram objekata koji prikazuje jedan crtež koji sadrži jedan uokviren pravougaonik i jedan tekst od nekoliko znakova,
- dijagram aktivnosti koji prikazuje crtanje uokvirene slike koja sadrži pravougaonike i tekstove,
- dijagram slučajeva korišćenja pri radu s programom,
- dijagram komponenata stavljajući klase koje čine logičke celine u istu komponentu.

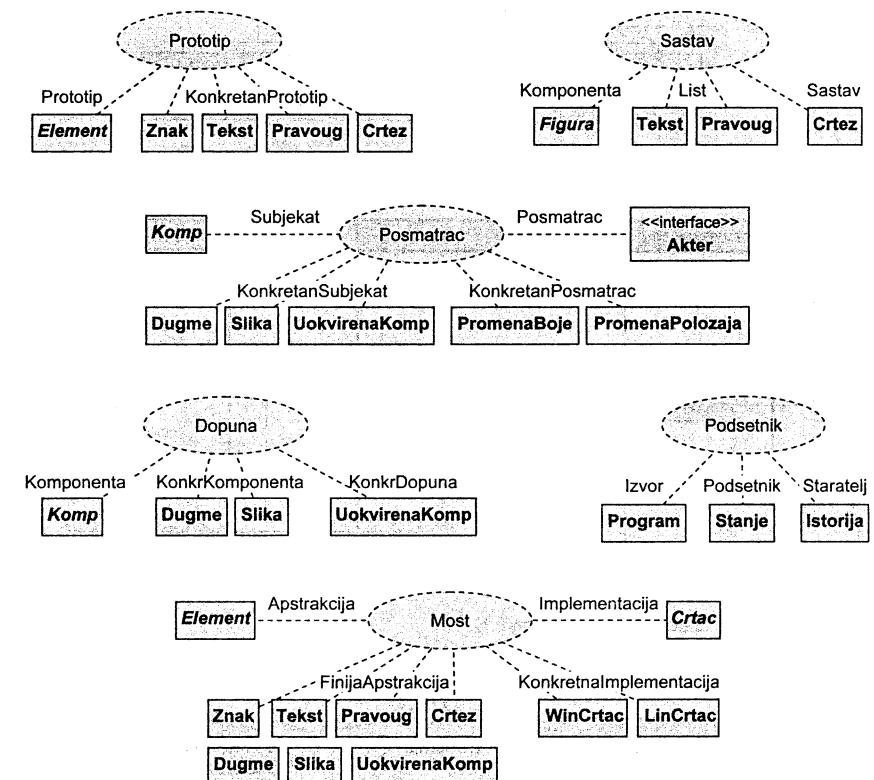
Rešenje:
a) Dijagrami klasa




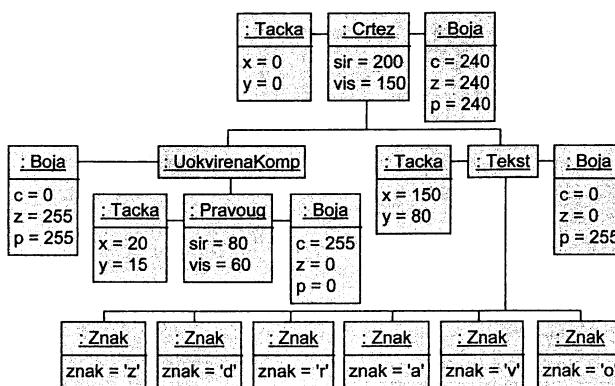
b) Dijagram paketa



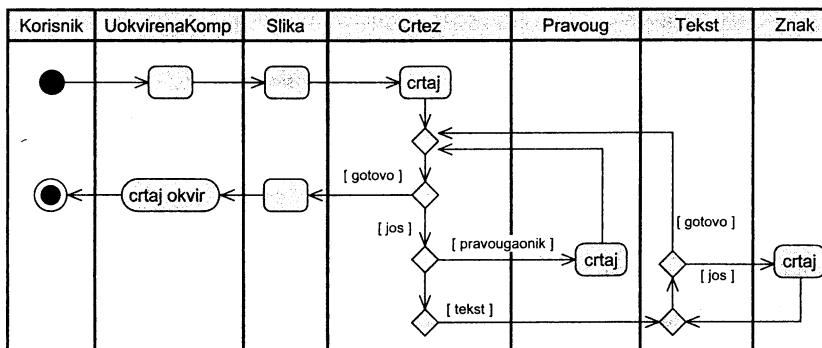
c) Projektni uzorci



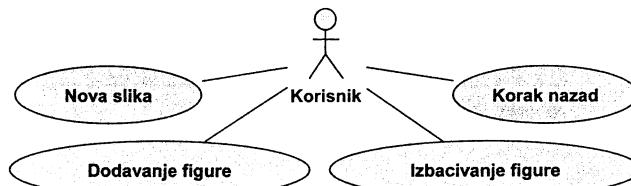
d) Dijagram objekata crteža



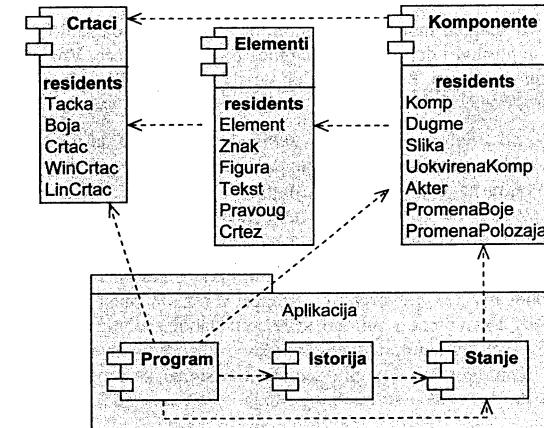
e) Dijagram aktivnosti crtanja uokvirene slike



f) Dijagram slučajeva korišćenja programa



g) Dijagram komponenata



Zadatak 38 Vozila, saobraćajnice i semafor {l, 23.01.2008.}

Aktivno vozilo je opisano snagom motora i trenutnom brzinom. Može da se dohvati trenutna brzina vozila, da se prikaže i da se postavi i dohvati saobraćajnica po kojoj se kreće. Vozilo ima apstraktan model vozila čiji naziv može da se dohvati. Pojedini modeli vozila imaju specifične geometrije i druge osobine. Postoji relativno mali broj različitih modela vozila i veliki broj vozila na saobraćajnicama.

Saobraćajnica može biti povezana sa više drugih saobraćajnica, može da joj se odredi dužina i da se prikaže. Vozila koja se kreću po saobraćajnici mogu da se dodaju i uklone jedno po jedno. Prosta saobraćajnica ima dužinu. Traka, raskrsnica i kružni tok su proste saobraćajnica. Ulica se sastoji od više traka. Složena saobraćajnica sastoji se od više povezanih saobraćajnica, a dužina joj se određuje kao zbir dužina saobraćajnica od kojih se sastoji. Jedinstvena maketa sadrži jednu složenu saobraćajnicu.

Semafor sadrži crveno, žuto i zeleno svetlo koje može biti uključeno ili ne. Može da radi u režimu jakog i slabog saobraćaja i da bude otvoren, zatvoren ili da trepće. U režimu jakog saobraćaja na zahtev promene stanja iz otvorenog prelazi u zatvoreno i obrnuto. U režimu slabog saobraćaja stalno je u stanju treptanja. Radom semafora na jednoj raskrsnici upravlja aktivi kontroler. Prilikom stvaranja semafor se dodeljuje zaduženom kontroleru. Kontroler šalje svojim semaforima signal za promenu stanja, a svaki semafor određuje svoje naredno stanje na osnovu trenutnog stanja i potrebnih informacija o prohodnim pravcima od kontrolera.

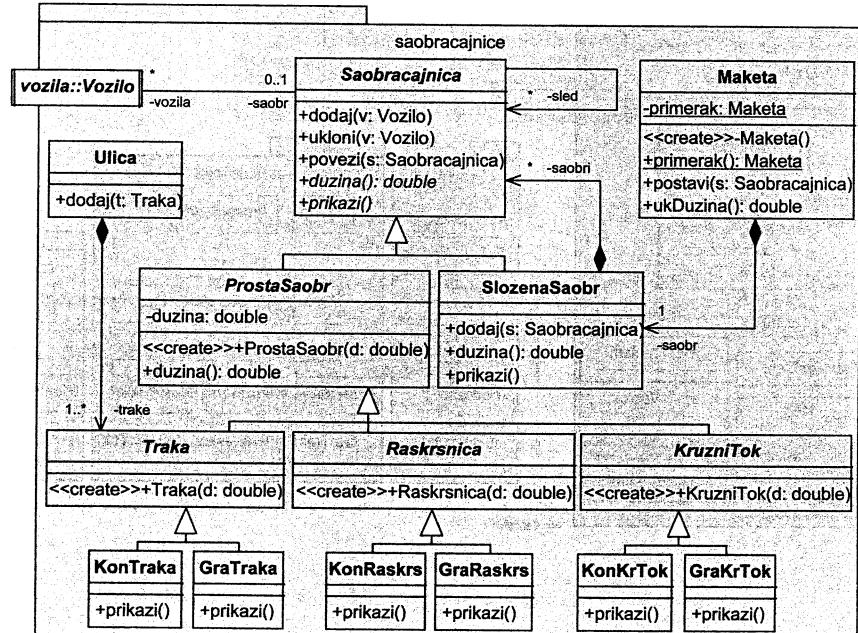
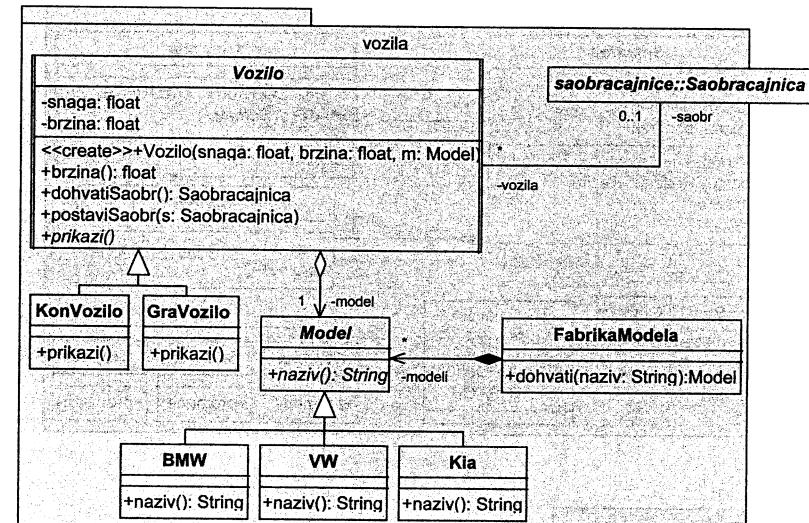
Interaktivni simulator saobraćaja sadrži maketu i može da radi u konzolnom i grafičkom režimu, o čemu se odlučuje prilikom stvaranja simulatora. U konzolnom režimu zahtevi korisnika se primaju preko tastature, a stanja elemenata makete ispisuju u tekstualnom obliku. U grafičkom režimu celokupna komunikacija sa korisnikom odvija se preko grafičke korisničke površi.

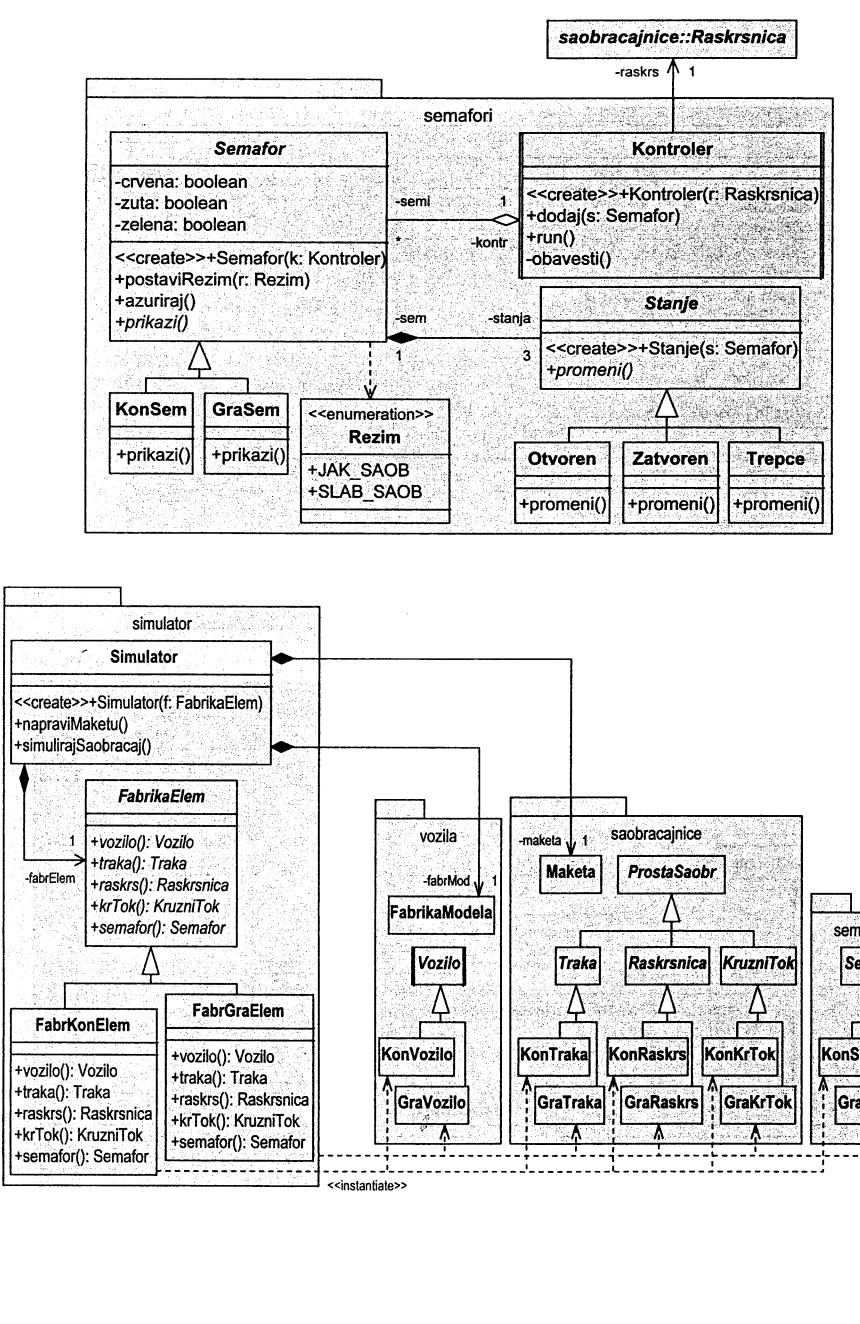
Projektovati na jeziku UML prethodni sistem. Priložiti:

- dijagram klasa razvrstanih u pakete (detaljan sadržaj klasa i odnose među klasama na potrebnom broju slika),
- prikaz korišćenih projektnih uzoraka,
- dijagram sekvence za određivanje ukupne dužine saobraćajnica jedne makete,
- dijagram stanja semafora,
- dijagram komponenata.

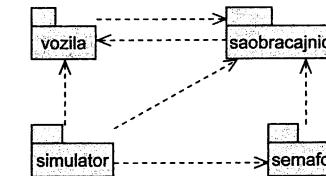
Rešenje:

- a) Dijagrami klase

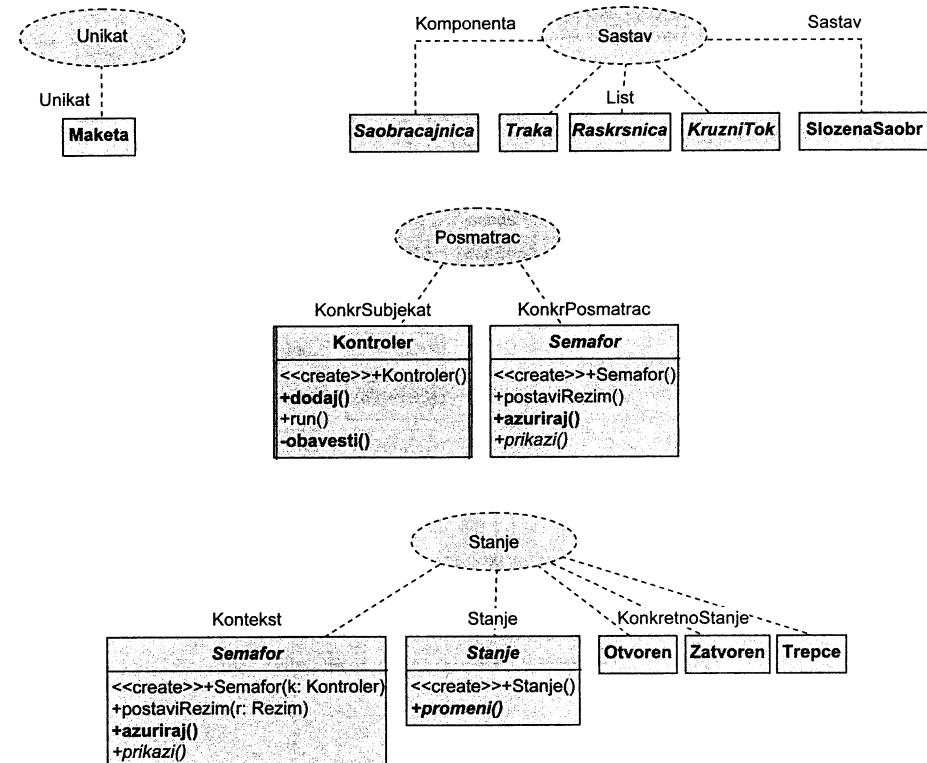


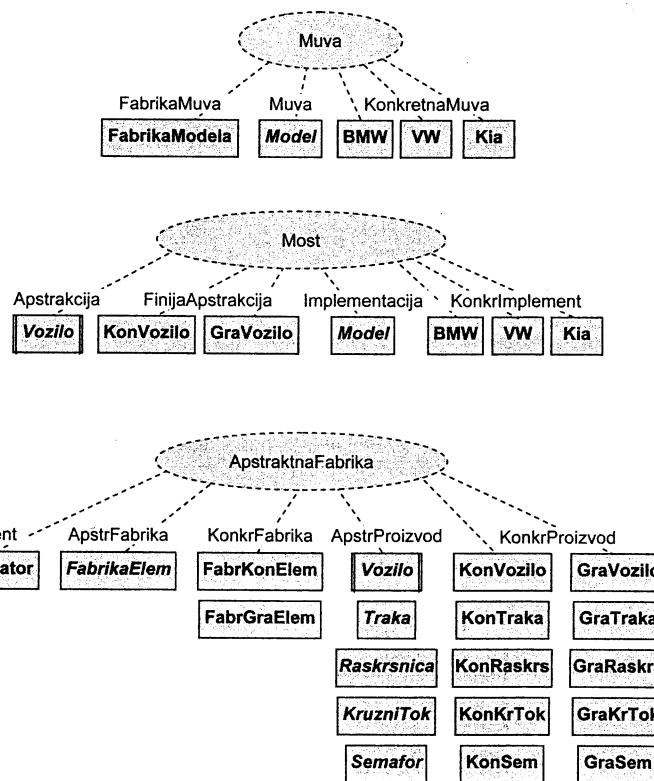


b) Dijagram paketa

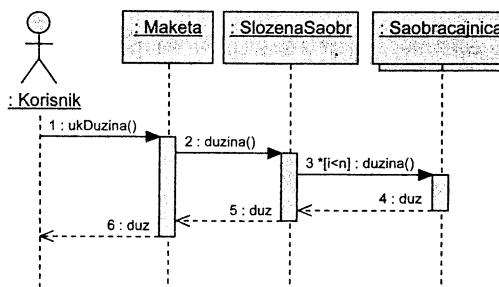


c) Projektni uzorci

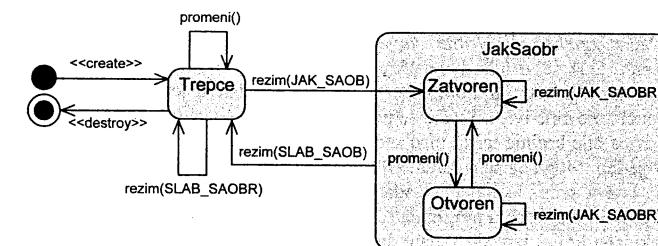




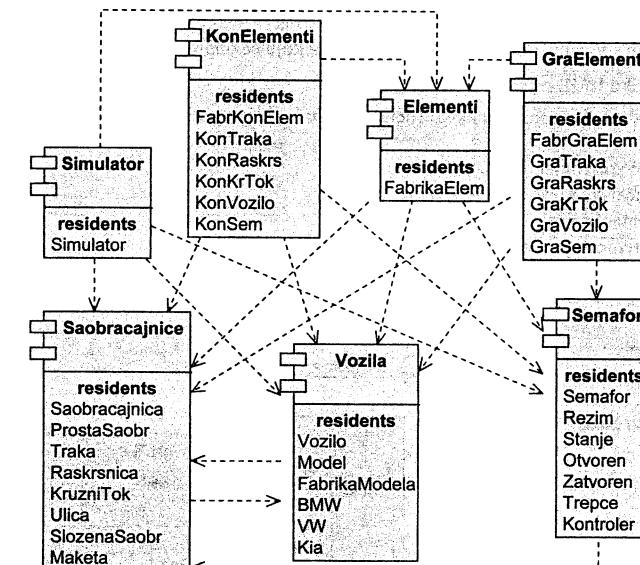
d) Dijagram sekvencije određivanja dužine saobraćajnice



e) Dijagram stanja semafora



f) Dijagram komponenta



Zadatak 39 Tačka, boja, duž, telo, scena, crtači, preslikavanje, dugme, radnja i program {l, 28.01.2009.}

Tačka u prostoru zadaje se realnim koordinatama koje mogu da se dohvate. Boja se zadaje celobrojnim komponentama crvene, zelene i plave boje u opsegu od 0 do 255 koje mogu da se dohvate. Broj različitih tačaka i boja je mali. Duž se zadaje dvema krajnjim tačkama i može da se crta zadatom bojom. Duž punom linijom i duž isprekidanom linijom su vrste duži. Telo zadate boje sadrži skup tačaka koje čine temena tela i skup duži koje čine ivice. Stvara se prazno poslužega se dodaju temena i ivice. Greška je ako se pokuša dodati ivica čije krajnje tačke nisu sadržane u telu. Scena sadrži tela koja se mogu dodavati i uklanjati jedno po jedno. Mogu da se dohvate sva sadržana tela odjednom, da se sadržaj scene zameni zadatam skupom tela i da se scena iscrtava na zadatom crtaču. Crtanje svih duži vrši se na istom crtaču koji može da se promeni u toku izvršenja programa. Crtač može da nacrtava pravu liniju zadatom bojom i stilom (puna linija, isprekidana linija) između zadate dve tačke u trodimenzionalnom prostoru uz projektovanje na dvodimenzionalnu prikaznu površ. Za dobijanje celobrojnih koordinata tačke na prikaznoj površi crtač koristi preslikavanje čije parametre ne treba razmatrati. Ekran i štampač su crtači. Interaktivni program korišćenjem grafičke korisničke površi omogućava obradu jedne scene. Pomoću dugmadi, od programa mogu da se zahtevaju radnje kao što su pravljenje nove scene, dodavanje i uklanjanje po jednog tela, crtanje na ekranu i štampanje na štampaču scene, vraćanje stanja scene proizvoljan broj koraka unazad i završetak rada. Prilikom stvaranja dugmeta definiše se radnja koju će dugme izazvati.

Projektovati na jeziku UML prethodni sistem. Priložiti:

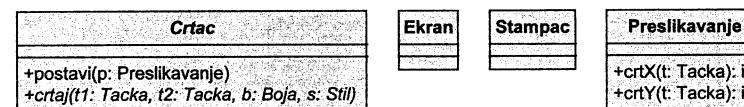
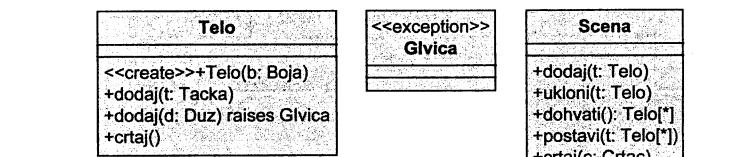
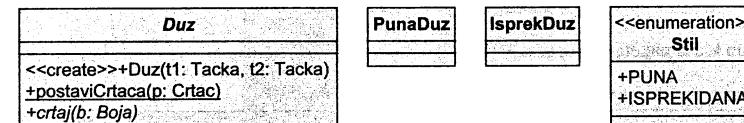
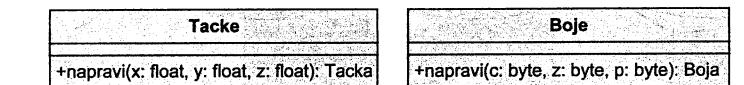
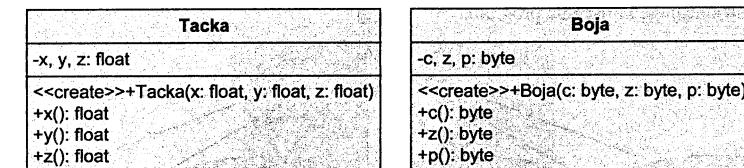
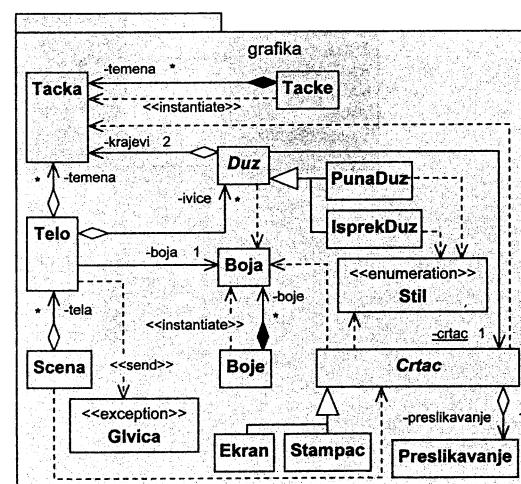
- dijagram klase razvrstanih u pakete (detaljan sadržaj klasa i odnose među klasama na potrebnom broju slika),
- prikaz korišćenih projektnih uzoraka,
- dijagram slučajeva korišćenja programa,
- dijagram komponenata,
- dijagram aktivnosti pri dodavanju novog tela sceni u programu.

Rešenje:

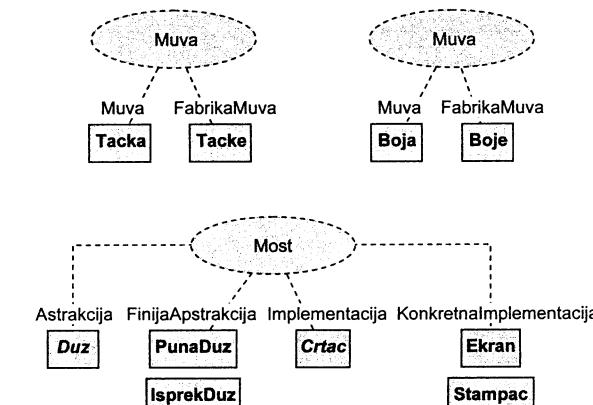
a) Dijagram paketa



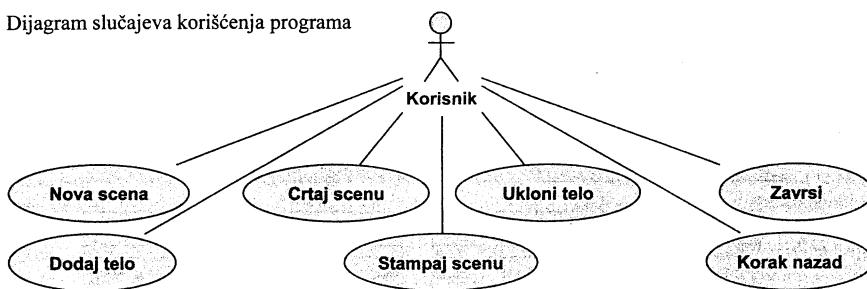
b) Dijagram klasa paketa *grafika*



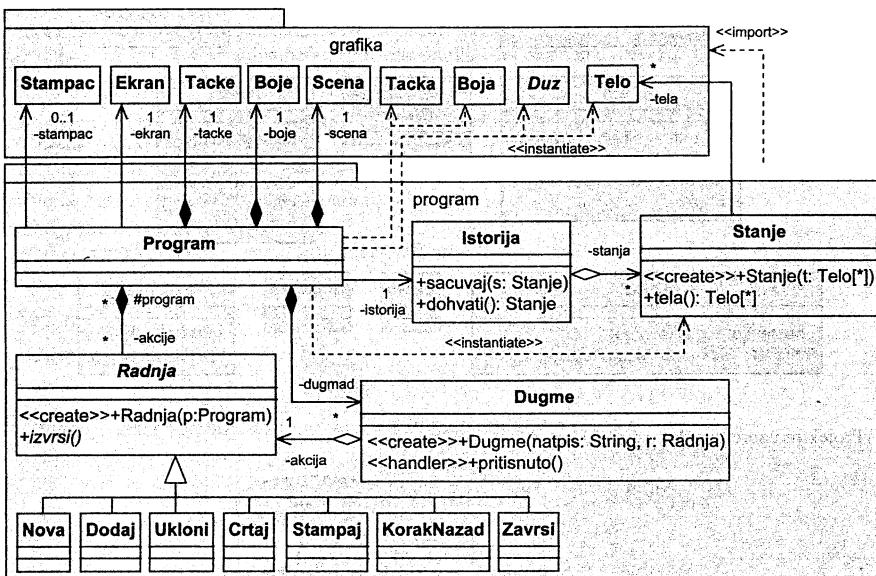
c) Projektni uzorci u paketu *grafika*



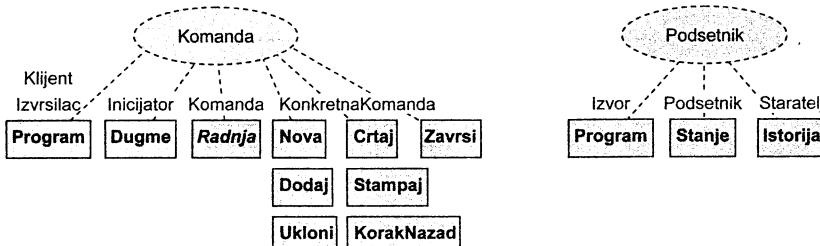
d) Dijagram slučajeva korišćenja programa



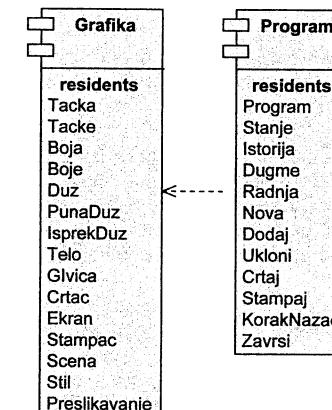
e) Dijagram klasa paketa program



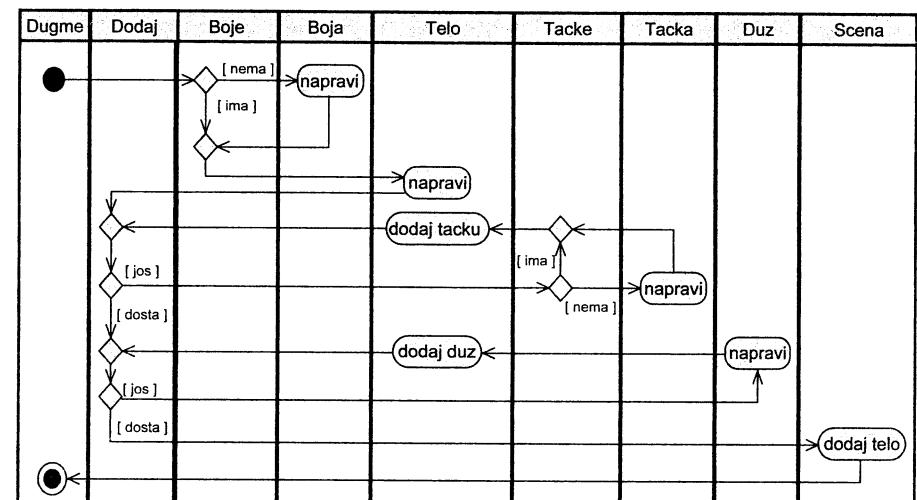
f) Primeri projektnih uzoraka u paketu program



g) Dijagram komponenata



h) Dijagram aktivnosti pri dodavanju novog tela sceni u programu



Ispitni zadaci

Zadatak I Izdavaštvo {I, 21.02.2007.}

Osoba ima ime. Autori, recenzenti i lektori su osobe. Autorsko delo ima autore. Roman, članak i pesma su autorska dela. Izdanje sadrži jedno ili više autorskih dela, i ima recenzente i lektora. Knjige i časopisi su izdanja. Zahtev za tiraž jednog izdanja sadrži broj primeraka i stil tog izdanja. Stil izdanja je određen načinom korišćenja i vrstom slova. Način korišćenja i vrsta slova moraju biti istovremeno generisani za ceo tiraž. Izdavač štampa primerke izdanja. Aktivna odeljenja izdavača su: priprema za štampu, štamparija, knjigoveznica, magacin, transport i knjižare. Izdanje se štampa tako što se zahtev za tiraž preda u odeljenju za pripremu, koje ga prosleđuje odeljenju štamparije, koje ga zatim prosleđuje knjigoveznici, koja smešta primerke izdanja u magacin. Priprema radi prelom izdanja primenjujući odgovarajuću vrstu slova. Štamparija štampa primerke izdanja umnožavanjem šablona izdanja. Knjigoveznica primenjuje razne tehnike korišćenja. Svaka tehnika korišćenja ima sledeće korake: obrada listova, izrada korica, spajanje. Na primer, za spiralni povez sa mekim koricama listovi se buše, korice se buše i vrši se spajanje spiralom, dok se za tvrdo korišćenje listovi ušivaju, na koricama se vrši štampa i korice se lepe. U knjižarama se nalazi veći broj primeraka istog izdanja. Aktivni kupci kupuju u knjižarama primerke izdanja.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke. Priložiti dijagram klasa sa naznakom korišćenih projektnih uzoraka (na potrebnom broju slika), dijagram sekvence za scenario proizvodnje jednog tiraža izdanja i dijagram stanja za jedno izdanje.

Zadatak II Videoteka {I, 28.03.2007.}

Osobe imaju ime i prezime i jedinstveni broj. Režiseri, glumci i fotografi su osobe. Delo ima naziv. Film je delo koje ima režisera i glumce. Fotografija je delo koja ima fotografa. Apstraktni medijum može sadržati jedno ili više dela. Kaseta i disk su apstraktni medijumi, pri čemu film može biti zapisan na kaseti ili disku, a fotografija samo na disku. Formatirani medijum nije vrsta medijuma već konkretnije apstraktni medijum, pri čemu formatirana kaseta i formatirani disk, kao vrste formatiranih medijuma, konkretnije odgovarajuće vrste apstraktnih medijuma. Konkretnije formatirane kasete mogu biti VHS ili MiniDV. Konkretni formatirani diskovi mogu biti CD ili DVD. Izdanje je formatirani medijum koji sadrži konkretni formatirani medijum u kutiji sa omotom. Videoteka sadrži police na kojima su originali izdanja. Magacin videoteka sadrži police na kojima se čuva dozvoljen broj kopija originala, koje se prave čim original stigne u videoteku. Korisnici su aktivne osobe koje pozajmљuju izdanja iz videoteka. Videoteka ima jedinstvenu evidenciju (čak i u slučaju više videoteka) u kojoj se vode zapisi o pozajmicama. Pri prvoj pozajmici kopije nekog izdanja u evidenciji se formira kompletan zapis o tom izdanju, a pri sledećim pozajmicama samo se u spisku pozajmica dodaje stavka koja sa referiše na taj zapis i na korisnika koji je pozajmio kopiju izdanja. Radnik je aktivna osoba koja iznajmljuje izdanja i vodi evidenciju iznajmljivanja.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke. Priložiti dijagram klasa raspoređenih u pakete vezane za stvaralaštvo, proizvodnju i korišćenje izdanja, sa naznakom korišćenih projektnih uzoraka (na potrebnom broju slika), dijagram paketa, i dijagram sekvence za životni vek jedne kopije uz zanemarivanje vođenja evidencije.

Zadatak III Medicinski dokumenti {/, 04.07.2007.}

Datum sadrži 3 cela broja (dan, mesec i godina). Osoba sadrži jedinstveni matični broj građana (JMBG), ime i prezime i datum rođenja. Pacijent je osoba koja ima broj zdravstvene legitimacije. Medicinska osoba ima broj radne knjižice i godine staža. Lekar je medicinska osoba koja može imati specijalizaciju. U slučaju promene podataka lekara (npr. prezime ili specijalizacija) ne smjeu se promeniti ovi podaci u ranoj formiranim dokumentima koji sadrže podatke o lekaru. Medicinski tehničar je medicinska osoba. Medicinsko osoblje je skup medicinskih osoba. Medicinski dokument ima jedinstveni identifikacioni broj, vezan je za konkretnog pacijenta i može da se pretražuje po zadatom uzorku teksta. Medicinski zapis sadrži datum, tekstualni opis i lekara koji beleži zapis. Dijagnoza sadrži šifru bolesti i tekstualni opis bolesti. Šifarnik oboljenja sadrži zbirku mogućih dijagnoza. U slučaju promene opisa neke dijagnoze u šifarniku mora se automatski promeniti njen opis u medicinskim dokumentima u kojima se ona već pojavi. Anamneza je medicinski dokument koji sadrži jedan medicinski zapis (datum prijema u bolnicu, prijemu, lekaru kroz razgovor sa pacijentom prilikom prijema, lekar koji vrši prijem) i dijagnozu na prijemu. Dekurzus je medicinski dokument koji sadrži zbirku dnevnih medicinskih zapisova o stanju pacijenta za vreme boravka u bolnici. Operativna lista je medicinski dokument koji sadrži jedan medicinski zapis (datum operacije, opis operacije, glavnog lekara koji potpisuje listu), operativnu dijagnozu i medicinsko osoblje koje je obavilo operaciju. Otpusna lista je medicinski dokument koji sadrži jedan medicinski zapis (datum otpusta, tekst predložene terapije, podatak o lekaru koji vrši otpust) i otpusnu dijagnozu. Istorija bolesti je medicinski dokument koji sadrži zbirku medicinskih dokumenata jednog pacijenta koji se kreiraju za vreme jednog lečenja, ali ne može da sadrži druge istorije bolesti. Istorija bolesti može da sadrži samo po jednu anamnezu, dekurzus i otpusnu listu, a operativnih lista može biti proizvoljan broj. Arhiva je zbirka istorija bolesti jedne ustanove.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke ili njihove modifikacije, jasno ih naznačiti i diskutovati odstupanja od originalnih uzoraka. Priložiti dijagram klasa raspoređenih u pakete, dijagram paketa, i dijagram sekvence za utvrđivanje broja istorija bolesti u kojima se razlikuju prijemne i otpusne dijagnoze pacijenata koji su lečeni u nekom periodu u dатој ustanovi.

Zadatak IV Testovi {/, 05.09.2007.}

Opis može biti tekst, slika ili složeni opis sastavljen od drugih opisa. Problem ima celobrojnu šifru, realnu težinu i opis problema. Zadatak je problem koji sadrži opis rešenja zadatka. Pitanje je problem koji sadrži skup ponuđenih odgovora od kojih jedan predstavlja tačan odgovor. Odgovor ima oznaku, opis i faktor kojim se množi težina pitanja ukoliko ga odabere ispitanik. Oznaka sadrži jedno slovo. Zbirka problema sadrži proizvoljan broj problema. Moguće je dodavanje, pronalaženje, menjanje i izbacivanje pojedinačnih problema. Baza problema je zbirka svih raspoloživih problema. Test je zbirka odabranih problema koja sadrži datum generisanja. Može da mu se odredi težina kao srednja težina sadržanih problema. Generator testa dobija kao ulazne podatke broj zadataka, broj pitanja i željenu težinu testa, a zatim nekim izabranim postupkom bira iz zbirke odgovarajuće probleme da zadovolji uslove, te sastavlja tekst testa. Program za interaktivno testiranje poziva generator testa koji stvara jedan test, prikazuje tekst testa korisniku, čeka da korisnik predlaže rešenje, a zatim poziva pregledača koji dodeljuje bodove i prikazuje osvojene bodove korisniku. Rešenje korisnika sadrži odgovore na sva pitanja i opis rešenja za svaki problem. Pregledač ima dve realizacije. U obe realizacije pregledač dohvata svaki odgovor korisnika i odgovarajući problem iz zbirke, stvara ocenjivač problema i upošljava ih. U prvoj realizaciji pregledač razvrstava pitanja i zadatke, stvara dovoljan broj ocenjivača pitanja i ocenjivača zadataka, pa pitanja prosledjuje ocenjivačima pitanja, a zadatke ocenjivačima zadataka. Ocenjivači pitanja i ocenjivači zadataka rade konkurentno. Druga realizacija pregledača stvara samo po jedan primerak ocenjivača pitanja i ocenjivača zadatka i prosledjuje svaki problem ocenjivaču pitanja, a ocenjivač pitanja pregleda pitanja i prosledjuje probleme ocenjivaču zadatka. Ocenjivač pitanja budiće odgovore na pitanja množeći težinu pitanja sa faktorom odgovora. Ocenjivač zadataka dohvata rešenje korisnika i opis rešenja odgovarajućeg zadatka iz zbirke, a zatim poređi nekom izabranom tehnikom rešenje korisnika sa opisom rešenja iz zbirke i dodeljuje bodove.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke ili njihove modifikacije, jasno ih naznačiti i diskutovati odstupanja od originalnih uzoraka. Priložiti dijagram klasa raspoređenih u pakete, dijagram slučajeva korišćenja sistema i dijagram sekvence za postupak testiranja.

Zadatak V Šalterska služba {l, 29.02.2008.}

Opis službenika ima automatski dodeljen identifikacioni broj, ime i prezime i funkciju (tekst). Službenici imaju opis i hijerarhijski su organizovani. Upravnik je aktivan službenik koji ima svoje podređene službenike, a sam može biti podređen svom upravniku. Radnik je službenik koji ne može imati podređene službenike. Radnik izvršava neku osnovnu obradu zahteva koji mu je dat. Posebno zaduženim radnicima, bez obzira na vrstu, mogu biti dodata i neka zaduženja. U zavisnosti od dnevnog režima rada (pre podne, posle podne) jedan korak u osnovnoj obradi zahteva, koji je nezavisan od vrste radnika, menja se dok se dodatna zaduženja ne menjaju. Šalterski radnik, kurir i referent aktivni su radnici koji obavljaju različite osnovne obrade u šalterskoj službi. Šalterska služba ima izvestan broj šaltera na kojima je po jedan šalterski radnik. Iza svakog šaltera postoji red od nekoliko referenata. Neki referenti su u drugim prostorijama. Šalterski radnik prima zahtev klijentu i predaje ga najbližem referentu u pozadini šaltera. Referent u pozadini šaltera, ako je slobodan, rešava zahtev, a ako ne obradi zahtev prosledjuje ga narednom najbližem referentu. Poslednji referent u redu iza šaltera, ukoliko sam ne obradi zahtev, pravi pošiljku u koju stavlja zahtev i koju adresira nekom referentu, a zatim je ubacuje u poštansko sanduče za kurira koji ih raznosi referentima u drugim prostorijama. Dodatno zaduženje radnika "brojača" u šalterskoj službi može biti brojanje zahteva.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke. Priložiti dijagram klasa smeštenih u pakete koji čine logičke celine sa naznakom korišćenih projektnih uzoraka (na potrebnom broju slika), dijagram objekata (bez navođenja atributa) za šaltersku službu s jednim šalterom i radnicima svih vrsta od interesa u trenutku kad šalterski radnik obraduje zahtev i kurir nosi zahtev nekom referentu, dijagram sekvene i dijagram aktivnosti obrade jednog zahteva koji biva potpuno obraden tek od strane referenta u nekoj prostoriji i dijagram komponenata stavljajući klase koje čine logičku celinu u istu komponentu.

Zadatak VI Softverska kompanija {l, 29.02.2008 – Ž.S.}

Softverskom proizvodu može da se dohvati vrsta. Informacioni sistem, aplikacija i softverski modul jesu softverski proizvodi. Informacioni sistem sastoji se od aplikacija, a aplikacije od softverskih modula. Informacioni sistem ima jedinstveni identifikacioni broj. Web aplikacija, klijent/server aplikacija i desktop aplikacija su aplikacije. Modul za poslovnu logiku, modul za rad s bazom podataka i modul za prikaz podataka su softverski moduli. Kolekcija softverskih rešenja sadrži proizvoljan broj softverskih proizvoda. Može da se dodaje original jednog softverskog proizvoda i da se kopija softverskog proizvoda zadate vrste dohvati iz kolekcije softverskih rešenja. Sadržaj kolekcije softverskih rešenja može da se pretražuje na više načina da bi se pronašla željena vrsta softverskog proizvoda. Neki od načina pretraživanja kolekcije softverskih rešenja jesu pretraživanje po datumu i pretraživanje po veličini. Softverski inženjer ima ime, može da mu se dohvati status, može da se zaposlji u nekoj softverskoj kompaniji i može da proizvodi softverske proizvode koje stavlja u zadatu kolekciju softverskih rešenja. Mlađi programer, stariji programer i vodeći projektant su statusi softverskih inženjera. Mlađi programer kreira softverske module. Stariji programer programira aplikacije pomoću softverskih modula iz kolekcije softverskih rešenja. Specijalizovani stariji programeri programiraju web aplikacije, klijent/server aplikacije ili desktop aplikacije. Aplikacije se različito programiraju za različite klijente. Smatratи da su moduli dovoljno opšti, odnosno da nisu različiti za različite klijente. Vodeći projektant integriše informacione sisteme tako što povezuje web aplikacije, klijent/server aplikacije i desktop aplikacije iz kolekcije softverskih rešenja. Softverska kompanija zapošjava softverske inženjere, poseduje kolekciju softverskih rešenja i proizvodi informacione sisteme za različite klijente. Neki od klijenata su banka i kladionica.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke. Priložiti dijagram klasa sa naznakom korišćenih projektnih uzoraka (na potrebnom broju slika), dijagram sekvene i dijagram aktivnosti za proizvodnju informacionog sistema. Akcijama se smatraju programiranje modula, programiranje aplikacija i integriranje informacionog sistema.

Zadatak VII Upravljanje nitima i procesima {/, 03.09.2008.}

Akter ima ime, identifikacioni broj, prioritet i kvant vremena koliko najduže može bez prekida da koristi procesor. Može da bude u jednom od stanja: *nov*, *spreman*, *aktivran*, *blokirani*, *uspavan* i *završen*. U aktivnom stanju izvršava se metoda `radi()`. Jedinstveni dispečer upravlja dodelom procesora spremnim akterima. Izbor aktera može biti na osnovu redosleda dolaženja u stanje *spreman* ili po opadajućim prioritetima. Jedinstveni časovnik po isteku postavljenog kvanta vremena obaveštava dispečera o tome. Akteri mogu da se samouspavljaju u zadatom trajanju, kada se prijavljuju časovniku radi obaveštavanja o protoku vremena. Brigu o isteku vremena spavanja vode sami akteri. Objekat uslova predviđa ispitivanje da li je neki uslov ispunjen. Semafor služi za sinhronizaciju rada aktera. Akter može da zatraži čekanje na semafor do ispunjavanja zadatog uslova. Ako uslov nije ispunjen akter se blokira. Akteri obaveštavaju semafor da se nešto desilo i da je potrebno da on preispita uslove blokiranih aktera. Tom prilikom se svi akteri u nizu blokiranih aktera, čiji su uslovi ispunjeni, prebacuju u stanje *spreman*. Proses je akter koji izvršava metodu `main()`. Niti su akteri koje stvara i čiji je vlasnik neki proces. Upravljač je proces koji čita naredbe operativnog sistema i stvara procese kojima se realizuju te naredbe. Jedinstveni operativni sistem sadrži dispečer, časovnik, upravljač i izvestan broj procesa.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke. Priložiti dijagram klase sa naznakom korišćenih projektnih uzoraka (na potrebnom broju slika), dijagrame sekvence koji opisuju scenarije raspoređivanja aktera i to gubljenja i dobijanja procesora u slučajevima isticanja kvanta vremena, čekanja na semaforu i spavanja i dijagram stanja aktera od stvaranja do završetka.

Zadatak VIII Igre na tabli, šah {/, 17.09.2008.}

Apstraktna aktivna igra sadrži apstraktну dvodimenzionalnu tablu i apstraktne igrače. Može da se postavi početno stanje igre, da se sproveđe igra, kad se igračima ciklički zahteva povlačenje poteza, i da se provjeri da li je igra završena, kad se proglašava i pobednik igre. Tabla sadrži polja koja mogu da budu prazna ili da sadrže po jednu apstraktnu figuru. Na tabli može da se postavi figura na prazno polje, da se ukloni figura s nepraznog polja, da se premesti figura s jednog na drugo polje (eventualna figura na određenom polju se uklanja) i da se prikaže sadržaj table. Apstraktna figura zna kom igraču pripada, može da se prenesti s jednog mesta na drugo na tabli, uz proveru ispravnosti poteza, i da se prikaže. Apstraktan igrač ima boju, igra na zadatoj tabli i može da vuče poteze. Apstraktan računar-igrač je igrač koji automatski vuče potez. Apstraktan korisnik-igrač je igrač koji ručno vuče potez. Igra može da se odvija preko konzole (tastatura i ekran) ili preko grafičke korisničke površi (GUI), pa postoje konkretnе table, figure i korisnici-igrači za oba načina igranja, pri čemu je način igranja parametar igre. Šah je konkretna igra. Šahovske figure (kralj, kraljica, ...) jesu figure od kojih svaka zna svoja pravila kretanja. Računar-šahista je računar-igrač u igri šaha. Korisnik-šahista je korisnik-igrač u igri šaha.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke. Priložiti dijagram klase sa naznakom korišćenih projektnih uzoraka (na potrebnom broju slika), dijagram komunikacije (saradnje) za jednu igru šaha u kojoj učestvuju računar-igrač i korisnik-igrač.

Zadatak IX Prodaja računara {l, 17.09.2008. – Ž.S.}

Računarska komponenta ima model i serijski broj koji mogu da se dohvate. Model je određen vrstom, nazivom i proizvođačem komponenata koji mogu da se dohvate. Procesor, matična ploča, operativna memorijska jedinica i hard disk su računarske komponente odgovarajućih vrsta. Može da se dohvati vrsta komponente i da se odredi cena komponente pomoću jedinstvenog cenovnika. Cenovnik računarskih komponenata za svaki model sadrži stavku sa cenom tog modela. Cena u stavci može da se dohvati i da se promeni. Cenovnik može da se pregleda po vrsti, po modelu i po ceni komponenata. Proizvođač računarskih komponenata ima naziv i proizvodi računarske komponente koje stavlja u pridruženo skladište. Postoje proizvođači procesora, matičnih ploča, operativnih memorija i diskova. Skladište može da sadrži proizvoljan broj komponenata. Stvara se prazno posle čega se komponente dodaju jedna po jedna. Može da se odredi broj komada zadatog modela u skladištu i da se iz skladišta odjednom izvadi zadati broj komponenata zadatog modela. Greška je ako se u skladištu ne nalazi traženi broj komponenata. Prodavnica računara sadrži skladište računarskih komponenata i zapošljava i otpušta radnike. Radnik ima ime, može da se zaposli u zadatoj prodavnici i da obradi zadati radni nalog. Prodavac, tehničar i dostavljač su radnici. Prodavnica može da nabavi odjednom veći broj komponenata i da prodaje jedan računar na osnovu narudžbenice primljene od datog kupca. Narudžbenica sadrži spisak modela koji treba da se ugrade u računar. Kupac ima ime i može da naruči računar u dатоj prodavnici na osnovu date narudžbenice, da plati zadatu sumu i da preuzme računar. Prodaja počinje stvaranjem radnog naloga na osnovu narudžbenice i podatka o kupcu. Nalog se potom prosledi prodavcu koji proveri da li u skladištu postoje sve potrebne računarske komponente. Ako je sve u redu, izračuna cenu računara, naplati od kupca i prosledi nalog tehničaru. Tehničar sastavlja računar i prosledi dostavljaču koji gotov računar isporučuje kupcu.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke ili njihove modifikacije, jasno ih naznačiti i diskutovati odstupanja od originalnih uzoraka. Priložiti dijagram klasa raspoređenih u pakete, dijagram sekvencije i dijagram aktivnosti za jedan scenario koji obuhvata sve korake kupoprodaje računara.

Zadatak X Čoveče ne lјuti se {l, 15.06.2009.}

Prikazivo je nešto što može da se prikaže. Prikazivi element igre ima pridružen prikazivač. Elementi igre Čoveče ne lјuti se su: kocka, mesto, staza, kućica, figura i tabla (videti dole). Za svaki element može da se sastavi tekstualni opis. Apstraktan prikazivač može da prikazuje stanje zadatog elementa. Prikazivač može biti konzolni (za tekstualni prikaz) ili grafički (za prikaz na grafičkoj površi). Kada se elementi prikazuju na grafičkoj površi moguće je odjednom menjati stil prikaza (na primer: 2D ili 3D) svih elemenata. Bačena kocka vraća pseudoslučajan broj od 1 do 6. Mesto može da (privremeno) sadrži figuru. Može da se proveri da li je mesto prazno, da se postavi figura na mesto, da se dohvati i da se ukloni figura s mesta. Polje je mesto koje sadrži i pokazivač na naredno polje. Skretница je polje koje sadrži i dodatni pokazivač na prvo polje završne kućice. Staza sadrži kružnu listu od zadatog broja polja (n), od kojih je svako ($n/4$)-to polje skretница. Apstraktna kućica ima jednu od 4 boje (crvena, plava, zelena, žuta) koja može da se dohvati. Polazna kućica je kućica koja sadrži 4 mesta i pokazivač na polje na stazi na kojoj se izlazi iz kućice. Završna kućica je kućica koja sadrži 4 polja. Figura ima boju i pokazivač na mesto na kojem se nalazi. Figura može da se izvede na početno polje staze za datu boju (kada pri bacanju kocke zadati broj k od 1 do 6, može da se pomeri na odgovarajuće polje, i to: (1) za k polja unapred na stazi, ako se medu njima ne nađe na polje sa skretnicom na kućicu iste boje; (2) ako se nađe na skretnicu na kućicu iste boje skreće se na prvo polje završne kućice i napreduje prema kraju kućice (ako je moguće, inače se figura ne pomera); (3) ako se na odredišnom polju nalazi figura druge boje ta figura se vraća u polaznu kućicu za njenu boju; (4) ako se na odredišnom polju nalazi figura iste boje figura se ne pomera. Tabla sadrži stazu, 4 polazne i 4 završne kućice koje se postavljaju pri stvaranju, a kasnije se mogu dohvati. Apstraktan igrač igra na zadatoj tabli zadatom kockom, ima boju, 4 figure, polaznu i završnu kućicu (sve u boji igrača). Na početku igrač smešta svoje figure u polaznu kućicu, a kasnije igra poteze tako što baca kocku i vuče figuru na tabli. Ukoliko pri bacanju kocke dobije 6 igrač igra dodatni potez. Rezultat poteza je informacija da li je igrač završio igru. Računar je igrač koji automatski bira i vuče figuru primenjujući znanje određenog nivoa. Početno znanje je nivo znanja kojim se na relativno naivan način bira figura kojom će se igrati. Ekspertsko znanje je znanje izbora figure, koje se zasniva na heurističkim pravilima koja primenjuju dobri igrači. Korisnik je igrač koji interaktivno saopštava figuru koju vuče. Jedinstvena igra Čoveče ne lјuti se ima tablu, kocku i niz od 4 igrača, a igra se tako što igrači nai-zmenično vuku poteze, sve dok jedan igrač ne završi. Posle svakog poteza prikazuju se kocka i tabla.

Projektovati na jeziku UML model opisanog sistema. Koristiti poznate projektne uzorke ili njihove modifikacije, jasno ih naznačiti i diskutovati odstupanja od originalnih uzoraka. Priložiti dijagram klasa raspoređenih u pakete, dijagram sekvencije za igru na konzoli s jednim korisnikom i dijagram aktivnosti za pomeranje figure.

Ispitna pitanja

i UML – uvod

1. Koje gradivne blokove definiše UML i čemu je namenjena koja vrsta gradivnih blokova?
2. Koje vrste dijagrama za opis statičkih aspekata modela definiše UML?
3. Kako se pravi razlika u notaciji apstrakcija i njihovih pojava (instanci) na jeziku UML? Navesti primere.

ii UML – dijagrami klasa

4. Navesti preciznu sintaksu operacije klase.
5. Navesti simbole za deklarisanje prava pristupa (vidljivost) klasnim članovima na UML dijagramu klasa i značenje svakog simbola.
6. Koje osobine (*properties*) vezane za ponašanje u konkurentnom okruženju može posedovati UML operacija? Šta koja osobina znači?
7. Šta označava i kako se predstavlja asocijacija na dijagramu klasa? Koji su ukrasi asocijacije mogući? Samo navesti nazive i primere.
8. Šta označava relacija asocijacije na dijagramu klasa i kakav je njen odnos sa vezom (*link*) na dijagramu objekata?
9. Koje vrste sadržanja podržava UML, koje su razlike i koja je notacija?
10. Koje vrednosti za multiplikativnost strana asocijacije su dozvoljene? Dati primere.
11. Šta označava relacija zavisnosti? Navesti najčešće primere zavisnosti između klasa.
12. Koja je osnovna razlika između relacija asocijacije i zavisnosti?
13. Kako se na dijogramima klasa i dijogramima paketa može naznačiti ugnezdenje?

iii UML – dijagrami paketa

14. Kojoj grupi "stvari" pripada, čemu je namenjen paket, koji je njegov simbol i šta se predstavlja na dijagramu paketa?
15. Na kojim dijogramima se pojavljuju, kako se obeležavaju i šta označavaju stereotipi <<access>> i <<import>>?
16. Šta označavaju pojmovi javnog i privatnog uvoza paketa i kojim se grafičkim simbolima predstavljaju na jeziku UML?
17. Šta je radni okvir i kako se predstavlja na jeziku UML?

iv UML – dijagrami interakcije

18. Koje vrste dijagrama interakcije postoje u jeziku UML 2?
19. U čemu je osnovna razlika između dijagrama sekvence i dijagrama komunikacije?
20. Kako se na dijogramima interakcije predstavljaju asinhronе poruke i kakva notacija se koristi za označavanje rednog broja asinhronе poruke? Navesti primer.
21. Šta označava natpis na strelici poruke na dijagramu komunikacije: G4 . 2?
22. Kakve vrste poruka se pojavljuju na dijogramima sekvene i komunikacije? Priložiti odgovarajuće grafičke UML simbole.
23. Navesti operatore okvira interakcije i njihovo značenje.
24. Šta označavaju fragmenti interakcije sa simbolima par, opt i alt?

v UML – dijagrami slučajeva korišćenja

25. Šta je slučaj korišćenja (*use case*), a šta scenario?
26. Šta je to akter, koji UML simbol se koristi, na kojim dijogramima se pojavljuje i kojim relacijama se povezuje sa drugim stvarima na datim dijogramima?
27. Šta označava puna linija na dijogramima slučajeva korišćenja? Ko su učesnici u odgovarajućoj relaciji i šta oni predstavljaju?
28. Kako se grafički predstavljaju, na kojoj vrsti dijagrama i šta označavaju relacije uključivanja (*include*) i proširivanja (*extend*)?
29. Na kojoj vrsti UML dijagrama se koristi stereotip relacije zavisnosti <<extend>> i šta on označava? Priložiti realan primer.
30. Na kojem UML dijagramu se koristi stereotip zavisnosti <<import>>, a na kojem <<include>> i šta odgovarajuće relacije označavaju?

vi UML – dijagrami aktivnosti

31. Kako se na jeziku UML predstavlja složena aktivnost sa ulaznim i izlaznim parametrima?
32. Nabrojati čvorove i pseudočvorove dijagrama aktivnosti. Za svaki priložiti grafički simbol i navesti šta označava.
33. Navesti notaciju i objasniti razliku između završnog čvora i čvora kraja toka.
34. Kako se na jeziku UML modelira tok objekata na dijogramima aktivnosti?
35. Kako se na dijogramima aktivnosti označavaju paralelni tokovi kontrole? Dati primer.
36. Kako se na dijogramima aktivnosti označavaju slanje signala, prihvatanje događaja, prihvatanje vremenskog događaja, bacanje i prihvatanje (obrada) izuzetka?

vii UML – dijagrami stanja

37. Navesti UML sintaksu prelaza (tranzicija) na dijagramu stanja. Priložiti primer.
38. Koja je osnovna razlika između pojmove samostalnog prelaza (*self-transition*) i unutrašnjeg prelaza stanja na jeziku UML?
39. Na kojim UML dijogramima se pojavljuju i u čemu je razlika između samostalne tranzicije i unutrašnje tranzicije (prelaza)?
40. Šta označava (navesti termin i značenje) simbol (H), a šta simbol (H*)?
41. Šta su to konkurentna podstanja i kako se modeliraju na jeziku UML?

viii UML – dijagrami klasa, napredniji pojmovi

42. Kako se modeliraju aktivne klase i objekti? Koja je razlika između niti i procesa?
43. Šta predstavlja klasa asocijacije? Priložiti primer koristeći UML notaciju.
44. Šta je klasa aocijacije? Navesti njen grafički simbol na nekom primeru.
45. Šta je klasa asocijacije, kako se grafički predstavlja na jeziku UML, i sa kojim pojmom u relacionim bazama se može uporediti?
46. Definisati pojam i na primeru ilustrovati *n*-arnu asocijaciju.
47. Šta označava kvalifikator? Priložiti primer.
48. Šta su zahtevani, a šta realizovani interfejsi i kako se oni označavaju na jeziku UML?
49. Šta je "generalizacioni skup", na kojem UML dijagramu se pojavljuje i koja notacija se koristi? Priložiti primer.

ix UML – dijagrami složene strukture

50. Šta je saradnja (*collaboration*)? Navesti UML2 notaciju saradnje sa ulogama, događanja saradnje (*collaboration occurrence*) i priložiti primer saradnje sa događanjima saradnje.

x UML – dijagrami komponenata

51. Šta predstavlja, na kojim dijagramima se pojavljuje i koji je UML simbol za artefakt?
 52. Na kojim dijagramima se sreću i koji je odnos između komponenata, čvorova i artefakata?
 53. Kojom relacijom su povezani artefakt i odgovarajuća komponenta? Skicirati primer.

xi UML – dijagrami raspoređivanja

54. Definisati pojmove jezika UML: artefakt (*artifact*) i čvor (*node*)? Na kojim dijagramima se pojavljuju? Navesti odgovaraće simbole.

xii Projektni uzorci – Uvod

1. Navesti i obrazložiti načine klasifikacije projektnih uzoraka.
 2. Kako se dele projektni uzorci prema kriterijumu namene, a kako prema kriterijumu domena?
 3. Kako su klasifikovani projektni uzorci? Definisati klase uzorka.

xiii Projektni uzorci stvaranja

4. Koja je motivacija i kada treba primeniti uzorak Unikat (*Singleton*)? Priložiti dijagram klasa koji opisuje uzorak, adekvatnu UML notaciju saradnje za ovaj uzorak i implementaciju uzorka na jezicima Java ili C++
 5. Navesti klasifikaciju i namenu uzorka Fabrički (proizvodni) metod (*Factory Method*). U okviru kojih drugih projektnih uzorka se često koristi i fabrički metod?
 6. Navesti namenu projektnog uzorka Graditelj (*Builder*) i uopštenu klasnu strukturu uzorka.
 7. Navesti namenu i priložiti uopšteni klasni dijagram projektnog uzorka Apstraktna fabrika (*Abstract Factory*).

xiv Projektni uzorci strukture

8. Nacrtati dijagram klasa koji opisuju klasnu strukturu projektnog uzorka Dekorater (Dopuna, *Decorator*).
 9. Navesti klasifikaciju i namenu projektnog uzorka Dekorater (Dopuna, *Decorator*). Priložiti objektni dijagram primera primene uzorka.
 10. Da li klijent treba da bude svestan postojanja objekta dopune (uzorak *Decorator*) i zašto?
 11. Uporediti po nameni i strukturi uzorke Kompozicija (Sklop, Sastav, *Composite*) i Dekorater (Omočač, Dopuna, *Decorator*).
 12. Koje vrste Adaptera postoje? Priložiti odgovaraće klasne dijagrame strukture sa naznačenim ulogama klasa u projektnom uzorku.
 13. Navesti klasifikaciju i strukturu projektnog uzorka Muva (*Flyweight*).
 14. Navesti uslove kada se uzorak Muva (*Flyweight*) može primeniti.
 15. Nacrtati klasni dijagram uzorka Muva (*Flyweight*) i opisati uloge klasa u uzorku.
 16. Šta označavaju pojmovi unutrašnjeg i spoljašnjeg stanja objekta kod projektnog uzorka Muva (*Flyweight*)? Koje uloge u uzorku pante odgovarajuća stanja?
 17. Kako je klasifikovan i koja je namena projektnog uzorka Fasada (*Facade*)?

18. Šta je motivacija projektnog uzorka Fasada (*Facade*) i da li projektni uzorak Fasada omogućava direktni pristup metodama objekata podsistema?
 19. Navesti klasifikaciju i namenu projektnog uzorka Most (*Bridge*). Nacrtati uopšteni klasni dijagram tog uzorka.

xv Projektni uzorci ponašanja

20. Koja je namena i koje su uloge klasa u projektnom uzorku Posmatrač (*Observer*)? Priložiti dijagram klasa koji uopšteno opisuje strukturu uzorka Posmatrač.
 21. Priložiti dijagram sekvene koji opisuje osnovni scenario projektnog uzorka Posmatrač (*Observer*).
 22. Koji projektni uzorak koristi mehanizam delegirane obrade događaja u paketu AWT jezika Java? Odrediti uloge klasa u datom projektnom uzorku.
 23. Koje vrste iteratora prema subjektu koji kontroliše iteraciju postoje i čime se odlikuju?
 24. Koja je klasifikacija, namena i kada treba primeniti uzorak Strategija (*Strategy*)? Priložiti dijagram klasa koji opisuje uzorak i adekvatnu UML notaciju saradnje za ovaj uzorak.
 25. Priložiti klasni dijagram strukture projektnog uzorka Strategija (*Strategy*).
 26. Navesti klasifikaciju, namenu i klasnu strukturu sa ulogama, za uzorak Stanje (*State*).
 27. Nacrtati uopšteni klasni dijagram uzorka Stanje (*State*). Koji uzorak ima praktično istu strukturu dijagrama klasa?
 28. Koje vrste (prema nameni) projektnog uzorka Predstavnik (*Proxy*) postoje?
 29. Navesti namenu projektnog uzorka Podsetnik (*Memento*), klasnu strukturu i dijagram sekvene koji opisuje ponašanje uloga u uzorku.
 30. Navesti klasifikaciju, namenu i klasnu strukturu uzorka Komanda (*Command*).
 31. Da li dodavanje novih komandi zahteva izmenu klasa koje učestvuju u projektnom uzorku Komanda (*Command*) i zašto? Sa kojim uzorcima je povezan uzorak Komanda?
 32. Opisati motivaciju za korišćenje uzorka Posrednik (*Mediator*) na nekom primeru.
 33. Kako je klasifikovan i koja je namena projektnog uzorka Lanac odgovornosti (*Chain of Responsibility*)?
 34. Koja je namena uzorka Lanac odgovornosti (*Chain of responsibility*) i na koji način se može primeniti zajedno sa uzorkom Kompozicija (Sklop, Sastav, *Composite*)?
 35. Navesti namenu i priložiti klasni dijagram (sa naznačenim metodama) koji opisuje opštu strukturu projektnog uzorka Posetilac (*Visitor*).

xvi Primeri odgovora na pitanja

- Koje gradivne blokove definiše UML?

Odgovor: UML definiše stvari (osnovne apstrakcije modela), relacije (koje povezuju stvari) i dijagrame (koji opisuju pojedine aspekte modela prikazujući grupisane stvari povezane relacijama).

- Navesti simbole za deklarisanje prava pristupa (vidljivost) klasnim članovima na UML dijagramu klasa i značenje svakog simbola.

Odgovor: + javni, - privatni, # zaštićeni, ~ paketski.

- Koje vrste sadržanja podržava UML, koje su razlike i koja je notacija?

Odgovor: UML podržava sadržanje tipa agregacije i kompozicije. Agregacija predstavlja odnos celina-deo u kojem celina nije odgovorna za životni vek dela, deo može postojati i bez celine i deo može biti sastavni deo više celina, dok kompozicija predstavlja odnos celina-deo u kojem je celina odgovorna za životni vek dela, deo ne može postojati bez celine i samo jedna celina može sadržati dati deo. Notacija:



- Navesti klasifikaciju i namenu uzorka Fabrički (proizvodni) metod (*Factory Method*). U okviru kojih drugih projektnih uzorka se često koristi i fabrički metod?

Odgovor: Fabrički metod je klasni uzorak stvaranja, a namena mu je delegiranje stvaranja objekta konkretnih klasa potklasi, jer se u apstraktnoj klasi fabrike ne zna za konkretni tip proizvoda. Koristi se često u okviru Šablonskog metoda i Apstraktne fabrike.

- Da li klijent treba da bude svestan postojanja objekta dopune (uzorak Decorator) i zašto?

Odgovor: Ne, klijent ne treba da bude svestan postojanja objekta dopune, jer dopuna pruža isti interfejs klijentu kao i dopunjeni (dekorisani) objekat. Razlog je što se na taj način objekat može dopunjavati proizvoljnim brojem dopuna, a da klijent o tome ne zna ništa.

Izdavač

AKADEMSKA MISAO

Bul. kralja Aleksandra 73, Beograd
tel./fax: (+381 11) 3218 354

office@akademiska-misao.rs

www.akademiska-misao.rs

CIP - Каталогизација у публикацији
Народна библиотека Србије, Београд



004. 41 (075.8) (076)

КРАУС, Ласло Л., 1949-

Zbirka zadataka iz projektovanja softvera
/ Laslo Kraus, Tartaљa Igor. - 2. dopunjeno
izd. - Beograd : Akademска мисао, 2009
(Београд : Planeta print). - 192 str. :
илустр. ; 24 cm

Tiraž 300. - Preporučena literatura: str. 6.

ISBN 978-86-7466-368-4

1. Тарталја, Игор, 1959 - [автор]
а) Софтвер - Пројектовање - Задаци
COBISIS.SR-ID 169818124
