Laboratorijska Vježba 7.2
Strukturalni paterni

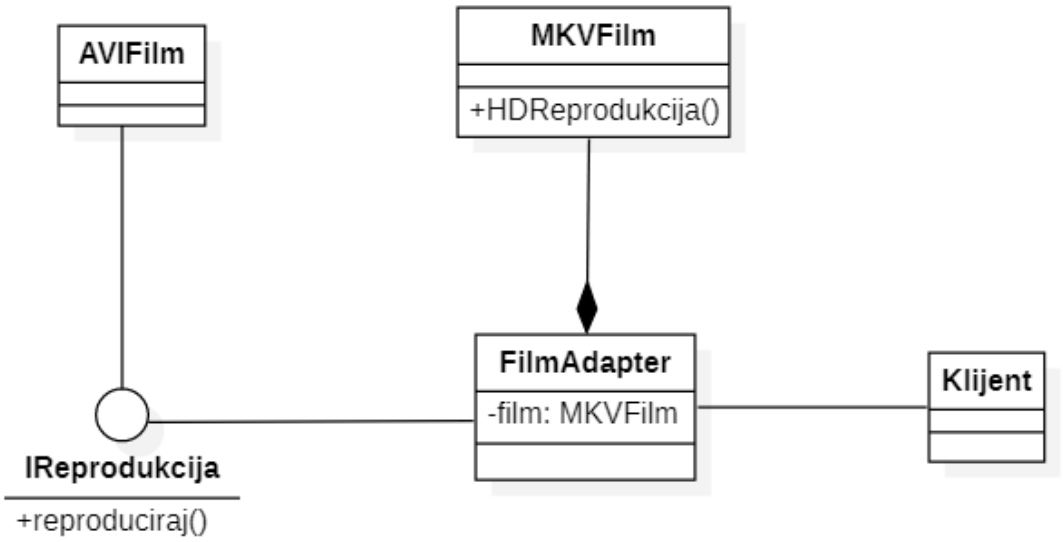
Napomena: Potrebno predznanje za vježbu 7 je gradivo obrađeno u predavanju 8.

1. Adapter patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje adapter patern.

Klijent posjeduje mogućnost gledanja filmova u AVI formatu. No, kako je ovaj format zastario, potrebno je omogućiti da klijent sada reproducira filmove u MKV formatu na istom video *player*-u.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 1.



Slika 1. Dijagram klasa sa korištenjem adapter paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public interface IReprodukcija
{
    void Reproduciraj();
}

public class AVIFilm : IReprodukcija
{
    public void Reproduciraj()
    {
        Console.Out.WriteLine("Reprodukcija u starom formatu!");
    }
}
```

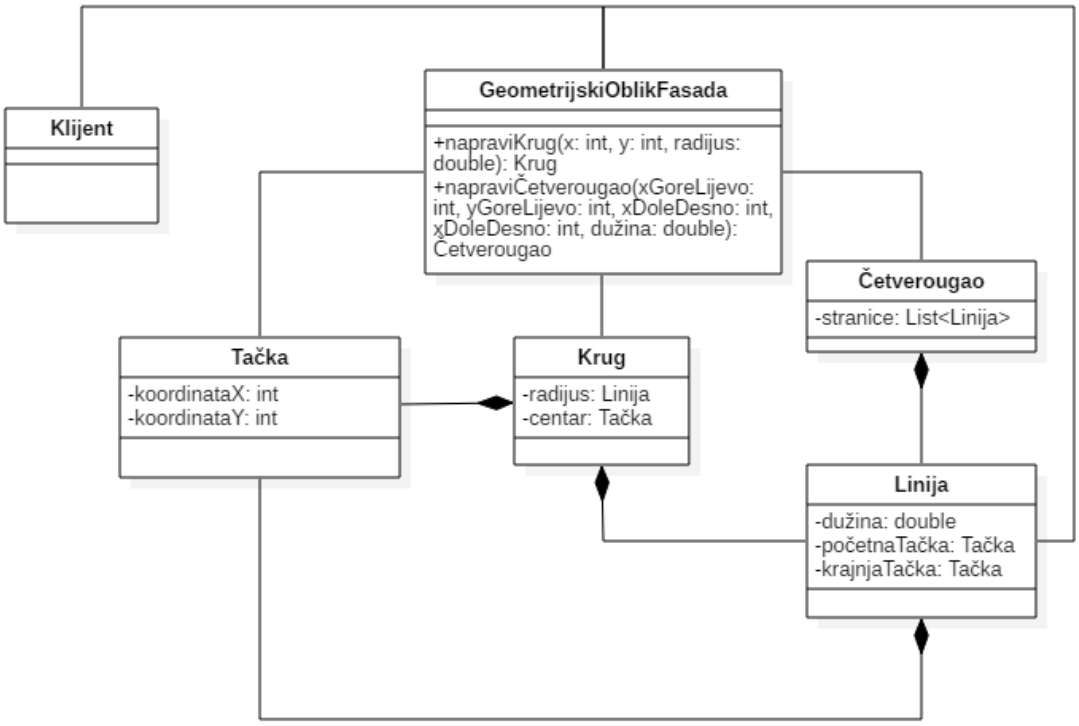
```
public class MKVFilm
{
    public void HDReprodukcija()
    {
        Console.Out.WriteLine("Reprodukcija u novom HD formatu!");
    }
}

public class FilmAdapter : IReprodukcija
{
    MKVFilm film = new MKVFilm();
    public void Reproduciraj()
    {
        film.HDReprodukcija();
    }
}

class Program
{
    static void Main(string[] args)
    {
        List<IReprodukcija> filmovi = new List<IReprodukcija>()
        {
            new AVIFilm(),
            new FilmAdapter()
        };

        foreach (var film in filmovi)
            film.Reproduciraj();
    }
}
```

2. Fasadni patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje fasadni patern.

Klijent želi mogućnost da kreira različite geometrijske oblike. Za kreiranje kruga klijentu je dovoljna jedna tačka (opisana x i y koordinatama) i jedna linija (opisana s dužinom, koja predstavlja radijus kružnice, te čije početna i krajnja tačka nisu definisane). Da bi kreirao četverougao, potrebno je spojiti četiri linije (opisane s dužinom, te krajnjim i početnim tačkama).

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 2.



Slika 2. Dijagram klasa sa korištenjem fasadnog paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public class Tačka
{
    int koordinataX, koordinataY;

    public Tačka(int x, int y)
    {
        koordinataX = x;
        koordinataY = y;
    }
}

public class Linija
{
    double dužina;
    Tačka početnaTačka, krajnjaTačka;

    public Linija(double length, Tačka start, Tačka end)
    {
        dužina = length;
        početnaTačka = start;
        krajnjaTačka = end;
    }
}

public class Krug
{
    Linija radijus;
```

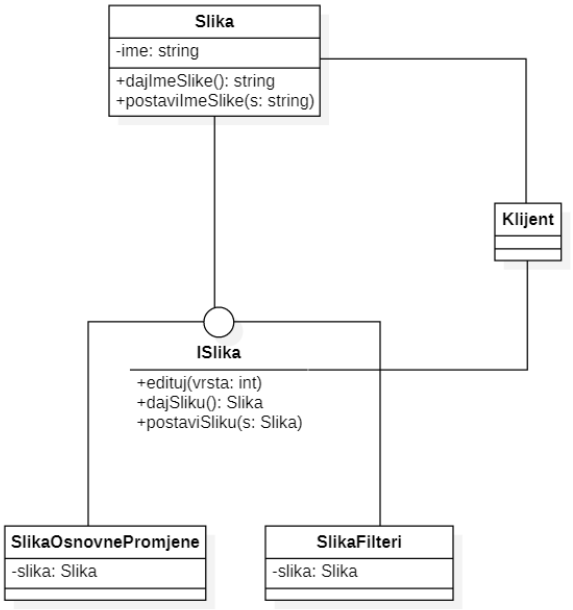
```
        Tačka centar;

        public Krug(Linija r, Tačka c)
        {
            radijus = r;
            centar = c;
        }
    }

    public class Četverougao
    {
        List<Linija> stranice = new List<Linija>();

        public Četverougao(List<Linija> sides)
        {
            stranice = sides;
        }
    }

    public class GeometrijskiOblikFasada
    {
        public Krug napraviKrug(int x, int y, double radijus)
        {
            Linija linija = new Linija(radijus, new Tačka(x, y - (int)radijus),
new Tačka(x, y + (int)radijus));
            Tačka centar = new Tačka(x, y);
            return new Krug(linija, centar);
        }
        public Četverougao napraviČetverougao(int xGoreLijevo, int yGoreLijevo,
int xDoleDesno,
         int yDoleDesno, double dužina)
        {
            Tačka goreLijevo = new Tačka(xGoreLijevo, yGoreLijevo);
            Tačka goreDesno = new Tačka(xDoleDesno, yGoreLijevo);
            Tačka doleLijevo = new Tačka(xGoreLijevo, yDoleDesno);
            Tačka doleDesno = new Tačka(xDoleDesno, yDoleDesno);
            List<Linija> stranice = new List<Linija>()
            {
                new Linija(dužina, goreLijevo, goreDesno),
                new Linija(dužina, doleLijevo, doleDesno),
                new Linija(dužina, goreLijevo, doleLijevo),
                new Linija(dužina, goreDesno, doleDesno)
            };
            return new Četverougao(stranice);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            GeometrijskiOblikFasada fasada = new GeometrijskiOblikFasada();
            Krug krug = fasada.napraviKrug(0, 0, 10);
            Četverougao četverougao = fasada.napraviČetverougao(10, 0, 0, 10, 10);
        }
    }
```

3. Dekorater patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje dekorater patern.

Klijent želi mogućnost editovanja slika. Moguće je vršenje dvije vrste modifikacija: osnovne promjene na slici (rotiranje, promjena veličine, promjena osvjetljenja) i primjena filtera (promjena intenziteta boje, izoštravanje, zamagljivanje).

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 3.



Slika 3. Dijagram klasa sa korištenjem dekorater paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public class Slika
{
    string ime;

    public string dajImeSlike()
    {
        return ime;
    }
    public void postaviImeSlike(string s)
    {
        ime = s;
    }
}

public interface ISlika
{
    void edituj(int vrsta);
    Slika dajSliku();
```

```
        void postaviSliku(Slika s);
    }

    public class SlikaOsnovnePromjene : ISlika
    {
        Slika slika;
        public void edituj(int vrsta)
        {
            string stara = slika.dajImeSlike();
            if (vrsta == 0) stara = "Rotirana " + stara;
            else if (vrsta == 1) stara = "Smanjena " + stara;
            else stara = "Svijetla " + stara;
            slika.postaviImeSlike(stara);
        }
        public Slika dajSliku()
        {
            return slika;
        }
        public void postaviSliku(Slika s)
        {
            slika = s;
        }
    }

    public class SlikaFilteri : ISlika
    {
        Slika slika;
        public void edituj(int vrsta)
        {
            string stara = slika.dajImeSlike();
            if (vrsta == 0) stara = "Crno-bijela " + stara;
            else if (vrsta == 1) stara = "Izoštrena " + stara;
            else stara = "Zamagljena " + stara;
            slika.postaviImeSlike(stara);
        }
        public Slika dajSliku()
        {
            return slika;
        }
        public void postaviSliku(Slika s)
        {
            slika = s;
        }
    }

        static void Main(string[] args)
        {
            Slika osnovnaSlika = new Slika();
            osnovnaSlika.postaviImeSlike("Slika");

            List<ISlika> dekoracije = new List<ISlika>()
            {
                new SlikaOsnovnePromjene(),
                new SlikaFilteri()
            };

            Console.Out.WriteLine("Slika prije promjena: " +
osnovnaSlika.dajImeSlike());
```

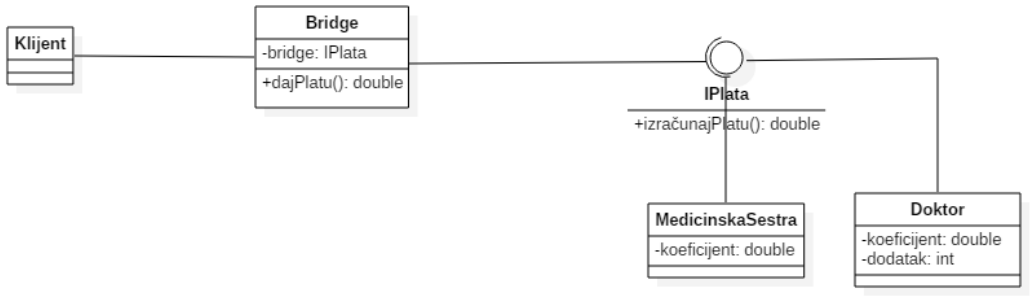
```
            dekoracije[0].postaviSliku(osnovnaSlika);
            dekoracije[0].edituj(0);
            Console.Out.WriteLine("Slika nakon osnovnih promjena: " +
dekoracije[0].dajSliku().dajImeSlike());

            dekoracije[1].postaviSliku(dekoracije[0].dajSliku());
            dekoracije[1].edituj(1);
            Console.Out.WriteLine("Slika nakon promjena sa filterima: " +
dekoracije[1].dajSliku().dajImeSlike());
        }
```

4. *Bridge* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *bridge* patern.

Klijent želi da dobije informaciju o visini plate zaposlenika u bolnici. Zaposlenici u bolnici mogu biti doktori i medicinske sestre, pri čemu se njihove plate računaju na različite načine, no sa istom osnovicom. U budućnosti je planirano uvođenje i zaposlenika po ugovoru, čije će se plate računati na osnovu ugovora o djelu.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 4.



Slika 4. Dijagram klasa sa korištenjem bridge paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public interface IPlata
    {
        double izračunajPlatu();
    }

    public class MedicinskaSestra : IPlata
    {
        double koeficijent;
        public MedicinskaSestra(double c)
        {
            koeficijent = c;
        }
```

```
        public double izračunajPlatu()
        {
            return koeficijent * 100;
        }
    }

    public class Doktor : IPlata
    {
        double koeficijent;
        int dodatak;
        public Doktor(double c, int a = 400)
        {
            koeficijent = c;
            dodatak = a;
        }
        public double izračunajPlatu()
        {
            return koeficijent * 105 + dodatak;
        }
    }

    public class Bridge
    {
        IPlata bridge;
        public Bridge(IPlata b)
        {
            bridge = b;
        }
        public double dajPlatu()
        {
            return 400 + bridge.izračunajPlatu();
        }
    }

        static void Main(string[] args)
        {
            List<IPlata> zaposlenici = new List<IPlata>()
            {
                new MedicinskaSestra(1.5),
                new Doktor(2.0)
            };
            foreach (var zaposlenik in zaposlenici)
            {
                Bridge bridge = new Bridge(zaposlenik);
                Console.Out.WriteLine("Plata za zaposlenika " +
zaposlenik.GetType().ToString().Substring(7) + " iznosi " +
bridge.dajPlatu().ToString() + " KM.");
            }
        }
```

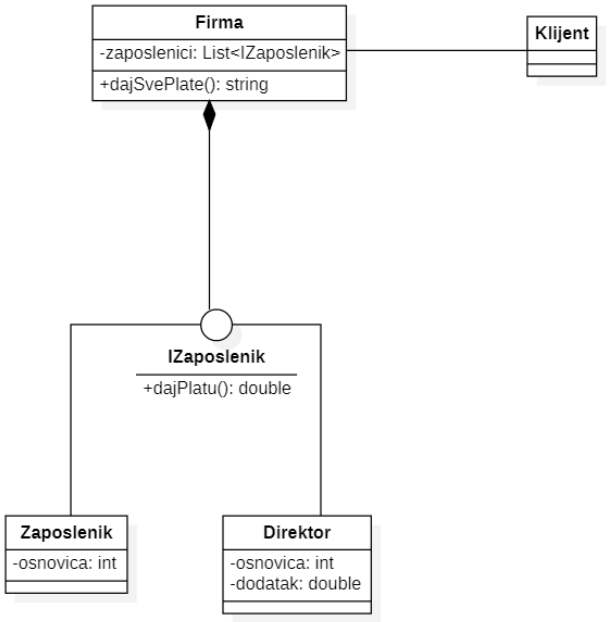
5. Kompozitni patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje kompozitni patern.

Klijent želi mogućnost pregleda plata različitih vrsta zaposlenika u nekoj firmi. Firma posjeduje dvije vrste zaposlenika – obične zaposlenike, čija plata ima samo osnovicu koja iznosi 250 KM, i koeficijent

koji iznosi 1.5, te direktore, čija plata ima osnovicu koja iznosi 500 KM, koeficijent koji iznosi 2.0, i dodatak koji iznosi 150 KM. Klijent želi da putem jedne metode dobije informacije o platama svih zaposlenika u firmi.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 5.



Slika 5. Dijagram klasa sa korištenjem kompozitnog paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public interface IZaposlenik
{
    double dajPlatu();
}

public class Zaposlenik : IZaposlenik
{
    int osnovica = 250;
    public double dajPlatu()
    {
        return (osnovica * 1.5);
    }
}

public class Direktor : IZaposlenik
{
    int osnovica = 500;
    double dodatak = 150;
    public double dajPlatu()
    {
        return (osnovica * 2 + dodatak);
    }
}
```

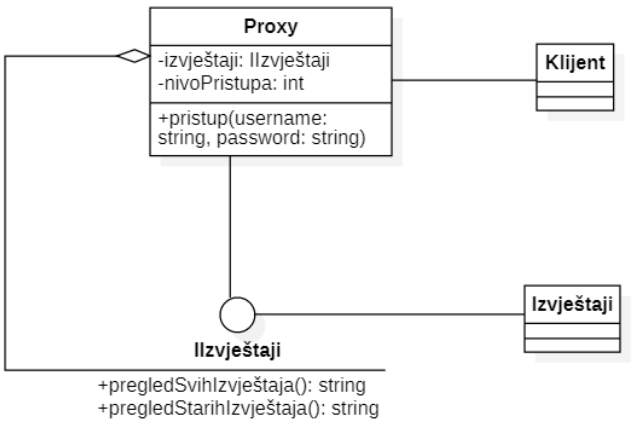
```
    public class Firma
    {
        List<IZaposlenik> zaposlenici = new List<IZaposlenik>();

        public List<IZaposlenik> Zaposlenici { get => zaposlenici; set =>
zaposlenici = value; }

        public string dajSvePlate()
        {
            string svePlate = "";
            foreach (IZaposlenik z in Zaposlenici)
            {
                svePlate += "Zaposlenik: " + z.GetType().ToString().Substring(11) +
", Plata: " + z.dajPlatu() + " KM\n";
            }
            return svePlate;
        }
    }

        static void Main(string[] args)
        {
            Firma firma = new Firma();
            firma.Zaposlenici.Add(new Direktor());
            firma.Zaposlenici.Add(new Zaposlenik());
            Console.Out.WriteLine("Sve plate zaposlenika firme:");
            Console.Out.WriteLine(firma.dajSvePlate());
        }
```

6. *Proxy* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *proxy* patern.

Klijent želi ograničiti prava pristupa za pregled finansijskih izvještaja. Finansijskim izvještajima mogu pristupiti samo korisnici čije korisničko ime završava s brojem, te čiji je *password* jednak *OOAD2019*. Ukoliko korisničko ime završava s brojem, a uneseni *password* je jednak *OOAD2018*, korisnik može pristupiti starim izvještajima.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 6.



Slika 6. Dijagram klasa sa korištenjem proxy paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public interface IIzvještaji
    {
        string pregledSvihIzvještaja();
        string pregledStarihIzvještaja();
    }

    public class Izvještaji : IIzvještaji
    {
        public string pregledSvihIzvještaja()
        {
            return "Svi izvještaji";
        }
        public string pregledStarihIzvještaja()
        {
            return "Stari izvještaji";
        }
    }

    public class Proxy : IIzvještaji
    {
        IIzvještaji izvještaji = new Izvještaji();
        int nivoPristupa = 0;
        public void pristup(string username, string password)
        {
            if (Char.IsDigit(username[username.Length - 1]) && password ==
"OOAD2021")
            {
                nivoPristupa = 1;
            }
            else if (password == "OOAD2020")
            {
                nivoPristupa = 2;
            }
            else nivoPristupa = 0;
        }
        public string pregledSvihIzvještaja()
        {
            if (nivoPristupa == 1) return izvještaji.pregledSvihIzvještaja();
            else return "PRISTUP ZABRANJEN";
        }
        public string pregledStarihIzvještaja()
        {
            if (nivoPristupa == 1 || nivoPristupa == 2) return
izvještaji.pregledStarihIzvještaja();
            else return "PRISTUP ZABRANJEN";
        }
    }

        static void Main(string[] args)
        {
            Proxy proxy = new Proxy();
            string izvještaji = proxy.pregledSvihIzvještaja();
            Console.Out.WriteLine("Svi izvještaji bez autentifikacije: \n" +
izvještaji);
            izvještaji = proxy.pregledStarihIzvještaja();
```

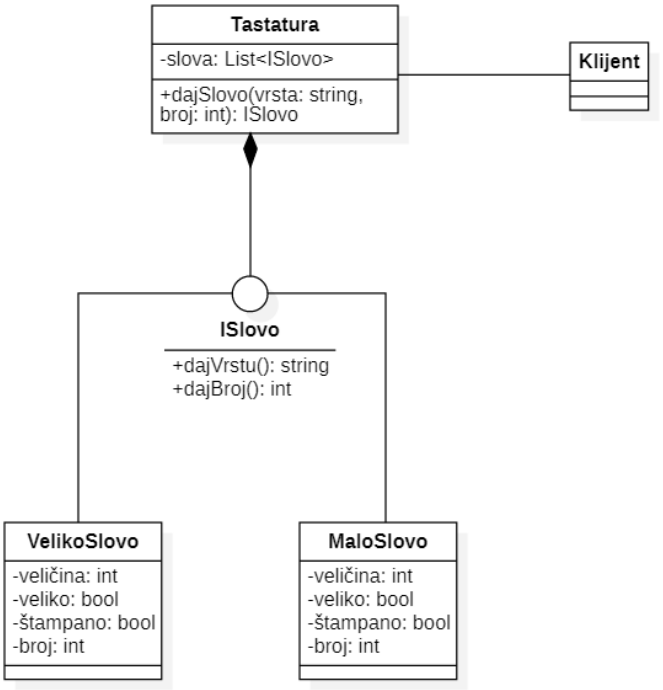
```
            Console.Out.WriteLine("Stari izvještaji bez autentifikacije: \n" +
izvještaji + "\n");

            proxy.pristup("Student1", "OOAD2020");
            izvještaji = proxy.pregledSvihIzvještaja();
            Console.Out.WriteLine("Svi izvještaji za nivo pristupa 2: \n" +
izvještaji);
            izvještaji = proxy.pregledStarihIzvještaja();
            Console.Out.WriteLine("Stari izvještaji za nivo pristupa 2: \n" +
izvještaji + "\n");

            proxy.pristup("Student1", "OOAD2021");
            izvještaji = proxy.pregledSvihIzvještaja();
            Console.Out.WriteLine("Svi izvještaji za nivo pristupa 1: \n" +
izvještaji);
            izvještaji = proxy.pregledStarihIzvještaja();
            Console.Out.WriteLine("Stari izvještaji za nivo pristupa 1: \n" +
izvještaji);
        }
```

7. *Flyweight* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *flyweight* patern.

Klijent želi mogućnost unošenja slova na tastaturi. Postoje dvije vrste slova: velika štampana veličine 12, i mala štampana veličine 10.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 7.



Slika 7. Dijagram klasa sa korištenjem flyweight paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public interface ISlovo
{
    string dajVrstu();
    int dajBroj();
}

 public class VelikoSlovo : ISlovo
{
    // bezlično stanje
    int veličina = 12;
    bool veliko = true;
    bool štampano = true;
    // specifično stanje
    int broj;

    public VelikoSlovo(int no)
    {
        broj = no;
    }

    public string dajVrstu()
    {
        return "Veliko slovo";
    }
    public int dajBroj()
    {
        return broj;
    }
}

public class MaloSlovo : ISlovo
{
    // bezlično stanje
    int veličina = 10;
    bool veliko = false;
    bool štampano = true;
    // specifično stanje
    int broj;

    public MaloSlovo(int no)
    {
        broj = no;
    }

    public string dajVrstu()
    {
        return "Malo slovo";
    }
    public int dajBroj()
    {
        return broj;
    }
}
```

```
    public class Tastatura
    {
        List<ISlovo> slova = new List<ISlovo>();

        public List<ISlovo> Slova { get => slova; set => slova = value; }

        public ISlovo dajSlovo(string vrsta, int broj)
        {
            foreach (ISlovo s in Slova)
            {
                // objekat već postoji - nema potrebe da instanciramo novi
                if (s.dajVrstu() == vrsta && s.dajBroj() == broj) return s;
            }
            ISlovo slovo;
            if (vrsta == "Veliko slovo") slovo = new VelikoSlovo(broj);
            else slovo = new MaloSlovo(broj);
            Slova.Add(slovo);
            return slovo;
        }
    }

        static void Main(string[] args)
        {
            Tastatura t = new Tastatura();

            for (int i = 0; i < 5; i++)
            {
                t.dajSlovo("Veliko slovo", i);
                t.dajSlovo("Malo slovo", i);
            }

            Console.Out.WriteLine("Broj slova u listi nakon dodavanja 5 novih
slova: " + t.Slova.Count);

            for (int i = 0; i < 5; i++)
            {
                t.dajSlovo("Veliko slovo", i);
                t.dajSlovo("Malo slovo", i);
            }

            Console.Out.WriteLine("Broj slova u listi nakon dodavanja 5 istih
slova: " + t.Slova.Count);
        }
```

8. Zadaci za samostalan rad

Programski kod primjera za sve paterne iz vježbe dostupan je u repozitoriju na sljedećem linku: https://github.com/ehlymana/OOADVjezbe, na *branchu* **lv7-2**.

Prepoznati i primijeniti odgovarajuće paterne na sljedeće opise sistema:

1. Klijent želi da autentificira pristup dosjeima radnika. Dosjeima mogu pristupiti samo oni koji unesu password *OOADAutentifikacija*. Potrebno je omogućiti i računanje plata zaposlenika, koji mogu biti direktor (s koeficijentom 1.75 i dodatkom 150 KM), sekretar (s koeficijentom 1.5 i dodatkom 100 KM) i administrativni radnik (s koeficijentom 1.25, bez dodataka). Svi zaposlenici imaju istu osnovicu koja iznosi 400 KM, na koju se dodaje koeficijent pomnožen sa 100 i dodatak (ukoliko isti postoji).

2. Klijent želi mogućnost naručivanja palačinaka u restoranu. On ne zna ništa o tome kako se palačinci prave, već samo želi mogućnost da odabere vrstu palačinka, vrstu namaza i vrstu dodatka koje želi. On također želi da naručuje palačinke preko postojećeg menu-a *StarinskiPalačinak*, s tim što taj menu omogućava samo odabir vrste i namaza palačinka, bez vrste dodataka.