Laboratorijska Vježba 10.2
Paterni ponašanja

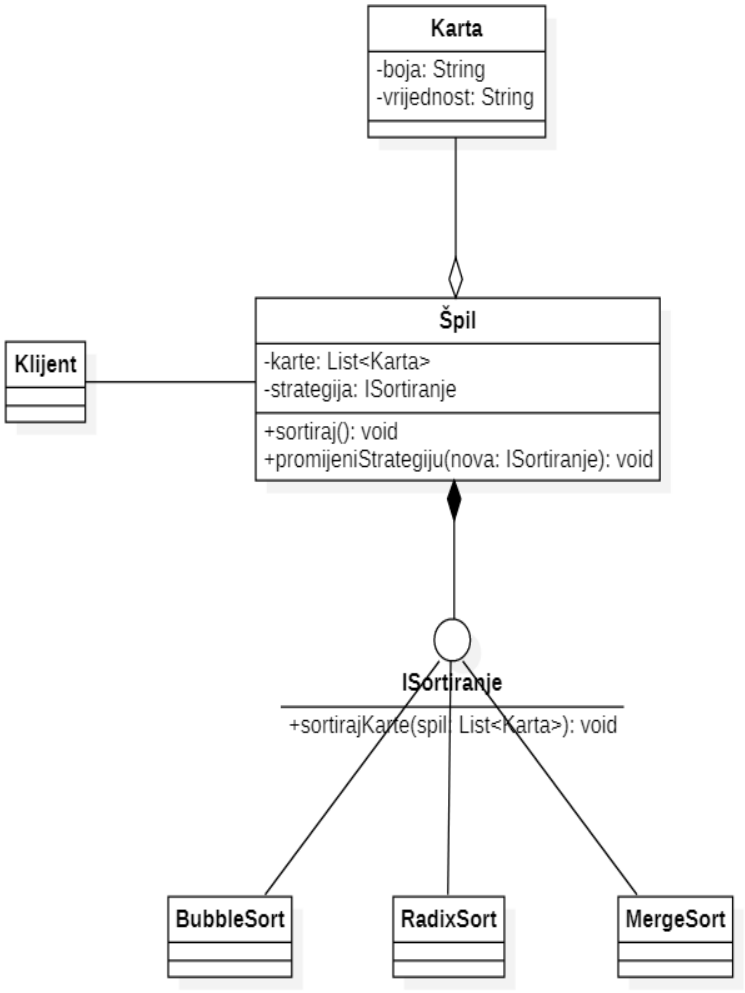
Napomena: Potrebno predznanje za vježbu 10 je gradivo obrađeno u predavanju 10-2.

1. *Strategy* patern

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

Klijent želi da sortira špil karata. Budući da ne zna koji će algoritam najbrže sortirati karte čiji se broj stalno povećava, želi isprobati različite algoritme za sortiranje te po želji promijeniti algoritam koji će se koristiti. Tri algoritma koje želi koristiti su *bubble sort, radix sort* i *merge sort*.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 1.



Slika 1. Dijagram klasa sa korištenjem strategy paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public class Karta
{
    string boja, vrijednost;
    public Karta(string b, string v)
    {
        boja = b;
        vrijednost = v;
    }
}

public interface ISortiranje
{
    public void sortirajKarte(List<Karta> spil);
}

public class BubbleSort : ISortiranje
{
    public void sortirajKarte(List<Karta> spil)
    {
        Console.Out.WriteLine("Sortiranje koristeći Bubble Sort");
    }
}

public class RadixSort : ISortiranje
{
    public void sortirajKarte(List<Karta> spil)
    {
        Console.Out.WriteLine("Sortiranje koristeći Radix Sort");
    }
}

public class MergeSort : ISortiranje
{
    public void sortirajKarte(List<Karta> spil)
    {
        Console.Out.WriteLine("Sortiranje koristeći Merge Sort");
    }
}

public class Špil
{
    List<Karta> karte = new List<Karta>();
    ISortiranje strategija;

    public Špil(List<Karta> cards)
    {
        karte = cards;
    }

    public void sortiraj()
    {
        strategija.sortirajKarte(karte);
    }

    public void promijeniStrategiju(ISortiranje novaStrategija)
    {
        strategija = novaStrategija;
    }
}
```

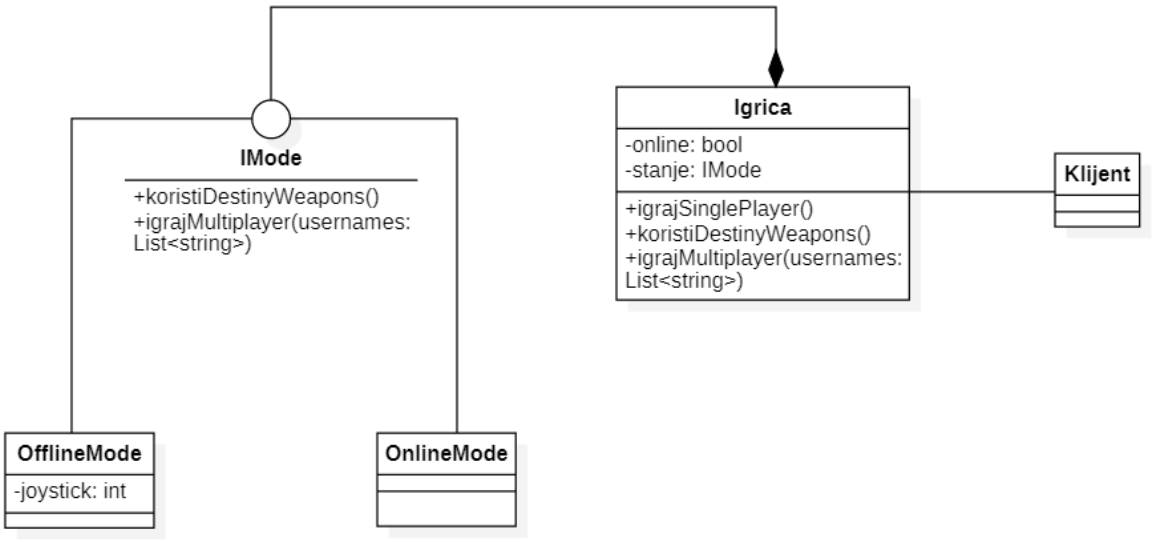
```
        static void Main(string[] args)
        {
            List<Karta> karte = new List<Karta> { new Karta("baklava", "1"), new
Karta("list", "2") };
            Špil spil = new Špil(karte);

            spil.promijeniStrategiju(new BubbleSort());
            spil.sortiraj();

            spil.promijeniStrategiju(new RadixSort());
            spil.sortiraj();

            spil.promijeniStrategiju(new MergeSort());
            spil.sortiraj();
        }
```

2. *State* patern

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

Klijent igra igricu na svom računaru. Može igrati igricu offline, u kom slučaju mu nije dopušteno koristiti *destiny weapons*, a *multiplayer mode* može igrati samo ukoliko ima dva *joysticka* konektovana na kompjuter. Ukoliko igra igricu *online*, dopušteno mu je koristiti *destiny weapons* i *multiplayer mode* sa bilo kojim igračem koji je također *online*.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 2.



Slika 2. Dijagram klasa sa korištenjem state paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public interface IMode
    {
        public void koristiDestinyWeapons();
        public void igrajMultiplayer(List<string> usernames);
    }

    public class OfflineMode : IMode
    {
        int joystick = 2;
        public void koristiDestinyWeapons()
        {
            throw new Exception("Nemoguće koristiti destiny weapons u offline
mode-u!");
        }
        public void igrajMultiplayer(List<string> usernames)
        {
            if (joystick != 2)
                throw new Exception("Konektujte dva joysticka na USB portove!");

            Console.Out.WriteLine("-------Multiplayer Igrica-------");
            Console.Out.Write("Igrači: ");
            foreach (string user in usernames)
                Console.Out.Write(user + " ");
            Console.Out.WriteLine("\n-------Kraj igrice-------");
        }
    }

    public class OnlineMode : IMode
    {
        public void koristiDestinyWeapons()
        {
            Console.Out.WriteLine("Koriste se destiny weapons!");
        }
        public void igrajMultiplayer(List<string> usernames)
        {
            Console.Out.WriteLine("-------Multiplayer Igrica-------");
            Console.Out.Write("Igrači: ");
            foreach (string user in usernames)
                Console.Out.Write(user + " ");
            Console.Out.WriteLine("\n-------Kraj igrice-------");
        }
    }

    public class Igrica
    {
        bool online = false;
        IMode stanje;

        public bool Online { get => online; set => online = value; }

        public void igrajSinglePlayer()
        {
            Console.Out.WriteLine("-------Single Player Igrica-------");
            Console.Out.WriteLine("-------Kraj igrice-------");
        }
        public void koristiDestinyWeapons()
        {
            if (Online)
                stanje = new OnlineMode();
```

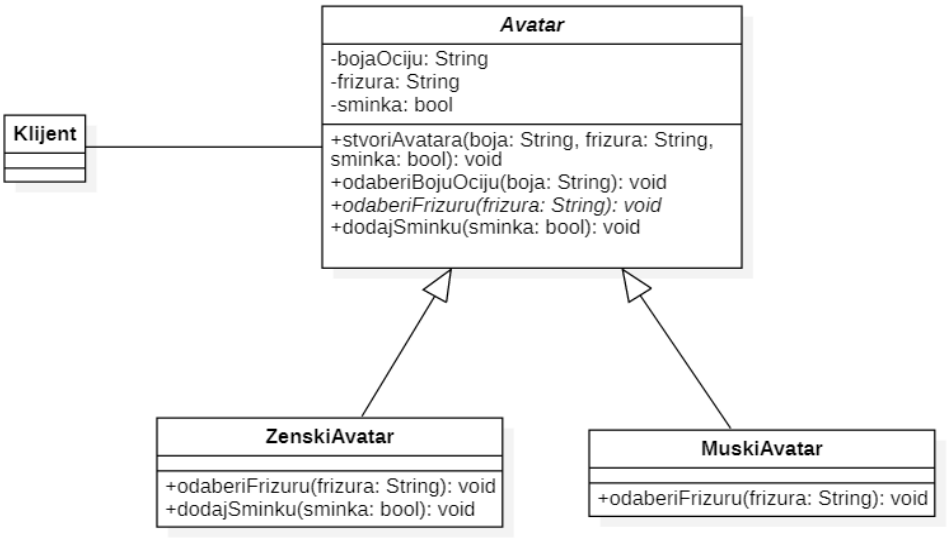
```
            else
                stanje = new OfflineMode();

            stanje.koristiDestinyWeapons();
        }
        public void igrajMultiplayer(List<string> usernames)
        {
            if (Online)
                stanje = new OnlineMode();
            else
                stanje = new OfflineMode();

            stanje.igrajMultiplayer(usernames);
        }
    }

        static void Main(string[] args)
        {
            Console.Out.WriteLine("Igrač želi da igra single player");
            Igrica igrica = new Igrica();
            igrica.igrajSinglePlayer();
            Console.Out.WriteLine();
            try
            {
                igrica.Online = false;
                Console.Out.WriteLine("Igrač je sada offline");
                igrica.koristiDestinyWeapons();
            }
            catch (Exception e)
            {
                Console.Out.WriteLine(e.Message);
                Console.Out.WriteLine();
                igrica.Online = true;
                Console.Out.WriteLine("Igrač je sada online");
                igrica.koristiDestinyWeapons();
            }
            Console.Out.WriteLine();
            igrica.igrajMultiplayer(new List<string> { "Igrač 1", "Igrač 2",
"Igrač 3" });
        }
```

3. *Template method* patern

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

Klijent želi da napravi svog avatara. Prvo se vrši odabir avatara, koji može biti muško ili žensko. Neovisno o vrsti avatara bira se boja očiju. Za ženske avatare omogućen je odabir 5 različitih frizura, a za muške avatare 3. Ženski avatari mogu imati definisanu šminku, dok muški ne mogu.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 3.



Slika 3. Dijagram klasa sa korištenjem template method paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public abstract class Avatar
    {
        protected string bojaOciju, frizura;
        protected bool sminka;

        public void odaberiBojuOciju(string boja)
        {
            if (new List<string> { "Zelena", "Plava", "Smeđa" }.Find(x => x ==
boja) != null)
                bojaOciju = boja;
            else
                throw new Exception("Nedozvoljena boja očiju!");
        }

        public abstract void odaberiFrizuru(string frizura);

        public virtual void dodajSminku(bool sminka)
        {
            throw new Exception("Nije dozvoljeno šminkanje avatara!");
        }

        public void stvoriAvatara(string boja, string frizura, bool sminka)
        {
            odaberiBojuOciju(boja);
            odaberiFrizuru(frizura);
            dodajSminku(sminka);
        }
    }

    public class MuskiAvatar : Avatar
    {
        public override void odaberiFrizuru(string frizura)
        {
            if (new List<string> { "Podignuta", "Kratka", "Ošišana na ćelavo"
}.Find(x => x == frizura) != null)
```

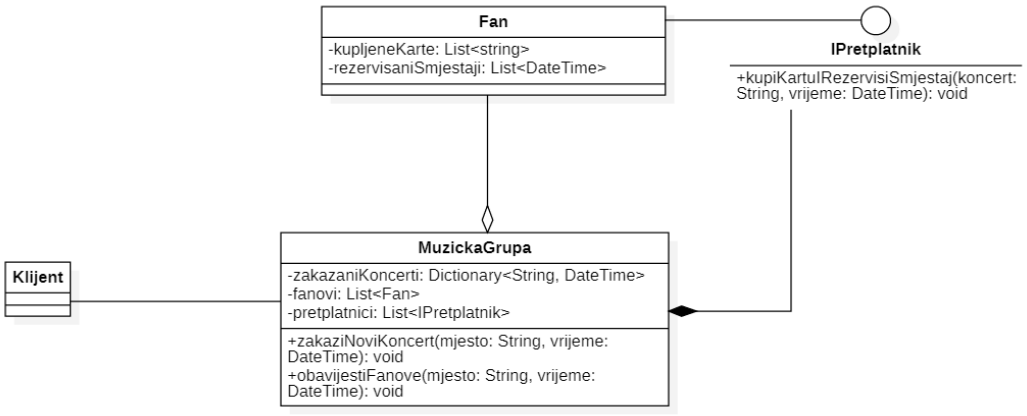
```
                this.frizura = frizura;
            else
                throw new Exception("Nedozvoljena frizura!");
        }
    }

    public class ZenskiAvatar : Avatar
    {
        public override void odaberiFrizuru(string frizura)
        {
            if (new List<string> { "Rep", "Puštena", "Pletenice", "Šiške", "Ombre"
}.Find(x => x == frizura) != null)
                this.frizura = frizura;
            else
                throw new Exception("Nedozvoljena frizura!");
        }
        public override void dodajSminku(bool sminka)
        {
            this.sminka = sminka;
        }
    }

        static void Main(string[] args)
        {
            Avatar musko = new MuskiAvatar();
            try
            {
                musko.stvoriAvatara("Zelena", "Podignuta", true);
            }
            catch (Exception e)
            {
                Console.Out.WriteLine(e.Message);
            }
            Avatar zensko = new ZenskiAvatar();
            zensko.stvoriAvatara("Smeđa", "Puštena", true);
            Console.Out.WriteLine("Šminkanje ženskog avatara uspješno!");
            try
            {
                zensko.stvoriAvatara("Plava", "Podignuta", true);
            }
            catch (Exception e)
            {
                Console.Out.WriteLine(e.Message);
            }
        }
```

4. *Observer* patern

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

Klijent želi da prisustvuje svakom koncertu muzičke grupe koju sluša. Kako bi mogao rezervisati hotel i kupiti kartu za koncert, želi dobiti obavijest o tome da je zakazan novi koncert. Samo oni klijenti koji se pretplate na primanje obavijesti biti će obaviješteni o zakazivanju koncerata.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 4.



Slika 4. Dijagram klasa sa korištenjem observer paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public interface IPretplatnik
    {
        public void kupiKartuIRezervisiSmjestaj(string koncert, DateTime vrijeme);
    }

    public class Fan : IPretplatnik
    {
        List<string> kupljeneKarte = new List<string>();
        List<DateTime> rezervisaniSmjestaji = new List<DateTime>();

        public List<string> KupljeneKarte { get => kupljeneKarte; set =>
kupljeneKarte = value; }

        public void kupiKartuIRezervisiSmjestaj(string koncert, DateTime vrijeme)
        {
            KupljeneKarte.Add(koncert);
            rezervisaniSmjestaji.Add(vrijeme);
        }
    }

    public class MuzickaGrupa
    {
        Dictionary<string, DateTime> zakazaniKoncerti = new Dictionary<string,
DateTime>();
        List<Fan> fanovi;
        List<IPretplatnik> pretplatnici;

        public List<Fan> Fanovi { get => fanovi; set => fanovi = value; }
        public List<IPretplatnik> Pretplatnici { get => pretplatnici; set =>
pretplatnici = value; }

        public void zakaziNoviKoncert(string mjesto, DateTime vrijeme)
        {
            zakazaniKoncerti.Add(mjesto, vrijeme);
            obavijestiFanove(mjesto, vrijeme);
        }
        public void obavijestiFanove(string mjesto, DateTime vrijeme)
        {
            foreach (IPretplatnik pretplatnik in Pretplatnici)
                pretplatnik.kupiKartuIRezervisiSmjestaj(mjesto, vrijeme);
```

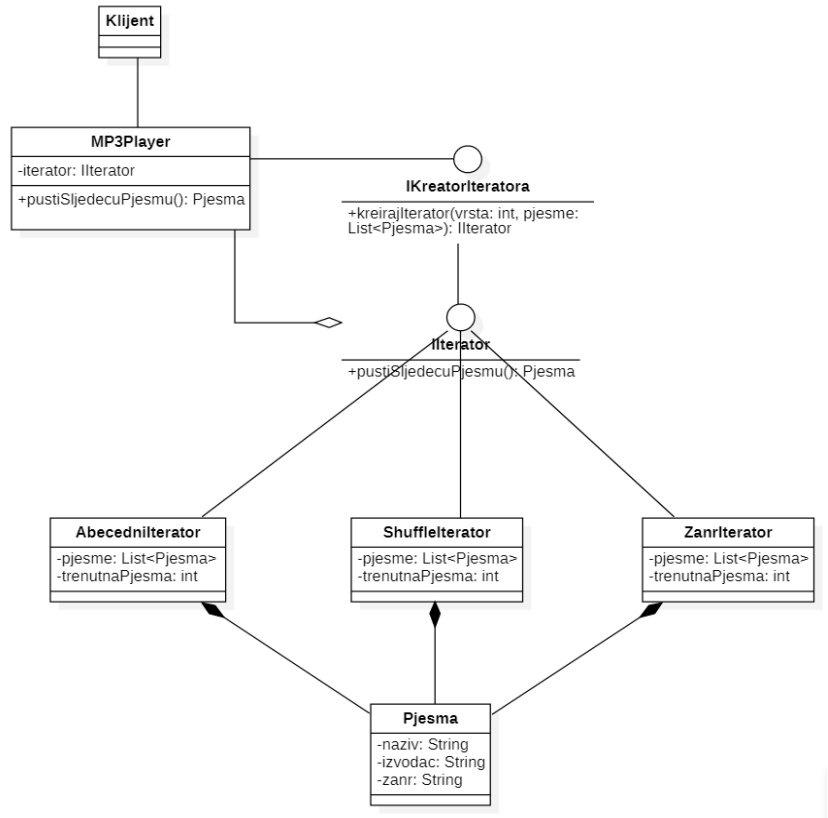
```
        }
    }

        static void Main(string[] args)
        {
            MuzickaGrupa grupa = new MuzickaGrupa();
            List<Fan> fanovi = new List<Fan> { new Fan(), new Fan() };
            grupa.Fanovi = fanovi;

            grupa.Pretplatnici = new List<IPretplatnik> { fanovi[0] };
            grupa.zakaziNoviKoncert("Sarajevo", new DateTime(2020, 12, 31));

            foreach(Fan fan in fanovi)
                Console.Out.WriteLine("Broj karata koje je fan kupio: " +
fan.KupljeneKarte.Count);
        }
```

5. Iterator patern

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

Klijent želi da sluša muziku na svom MP3 *playeru*. Osnovi način za formiranje *playliste* je redanje pjesama po abedecnom redu. Ukoliko klijent uključi *shuffle*, kroz muziku će se prolaziti koristeći neku formulu koja određuje broj sljedeće pjesme. Ukoliko klijent želi, može puštati muziku prema žanrovima, pri čemu se žanrovi određuju po nekom predefinisanom redoslijedu.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 5.



Slika 5. Dijagram klasa sa korištenjem iterator paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
public class Pjesma
{
    string naziv, izvodac, zanr;

    public string Naziv { get => naziv; set => naziv = value; }
    public string Izvodac { get => izvodac; set => izvodac = value; }
    public string Zanr { get => zanr; set => zanr = value; }

    public Pjesma(string name, string artist, string genre)
    {
        Naziv = name;
        Izvodac = artist;
        Zanr = genre;
    }
}

public interface IIterator
{
    public Pjesma pustiSljedecuPjesmu();
}

public class AbecedniIterator : IIterator
{
    List<Pjesma> pjesme;
    int trenutnaPjesma;
    public AbecedniIterator(List<Pjesma> songs)
    {
        pjesme = songs;
    }
    public Pjesma pustiSljedecuPjesmu()
    {
        Pjesma trenutna = pjesme[trenutnaPjesma];
        pjesme = pjesme.OrderBy(x => x.Naziv).ThenBy(x => x.Izvodac).ToList();
        int noviIndex = pjesme.IndexOf(trenutna);
        trenutnaPjesma = noviIndex + 1;
        if (trenutnaPjesma >= pjesme.Count)
            trenutnaPjesma -= pjesme.Count;
        return pjesme[trenutnaPjesma];
    }
}

public class ShuffleIterator : IIterator
{
    List<Pjesma> pjesme;
    int trenutnaPjesma;
    public ShuffleIterator(List<Pjesma> songs)
    {
        pjesme = songs;
    }
    public Pjesma pustiSljedecuPjesmu()
    {
        Random random = new Random();
        int noviIndex = random.Next(0, pjesme.Count - 1);
        trenutnaPjesma = noviIndex + 1;
        if (trenutnaPjesma >= pjesme.Count)
            trenutnaPjesma -= pjesme.Count;
```

```
            return pjesme[trenutnaPjesma];
        }
    }

    public class ZanrIterator : IIterator
    {
        List<Pjesma> pjesme;
        int trenutnaPjesma;
        public ZanrIterator(List<Pjesma> songs)
        {
            pjesme = songs;
        }
        public Pjesma pustiSljedecuPjesmu()
        {
            Pjesma trenutna = pjesme[trenutnaPjesma];
            pjesme = pjesme.OrderBy(x => x.Zanr).ToList();
            int noviIndex = pjesme.IndexOf(trenutna);
            trenutnaPjesma = noviIndex + 1;
            if (trenutnaPjesma >= pjesme.Count)
                trenutnaPjesma -= pjesme.Count;
            return pjesme[trenutnaPjesma];
        }
    }

    public interface IKreatorIteratora
    {
        public IIterator kreirajIterator(int vrsta, List<Pjesma> pjesme);
    }

    public class MP3Player : IKreatorIteratora
    {
        IIterator iterator;
        public IIterator kreirajIterator(int vrsta, List<Pjesma> pjesme)
        {
            if (vrsta == 0)
                iterator = new AbecedniIterator(pjesme);
            else if (vrsta == 1)
                iterator = new ShuffleIterator(pjesme);
            else
                iterator = new ZanrIterator(pjesme);

            return iterator;
        }
        public Pjesma pustiSljedecuPjesmu()
        {
            return iterator.pustiSljedecuPjesmu();
        }
    }

        static void Main(string[] args)
        {
            MP3Player player = new MP3Player();
            List<Pjesma> pjesme = new List<Pjesma>
            {
                new Pjesma("Escondida", "Spanish artist", "Pop"),
                new Pjesma("El Dorado", "Korean artist", "K-Pop"),
                new Pjesma("Galway Girl", "English artist", "Pop")
            };
            Pjesma p1 = null;
```

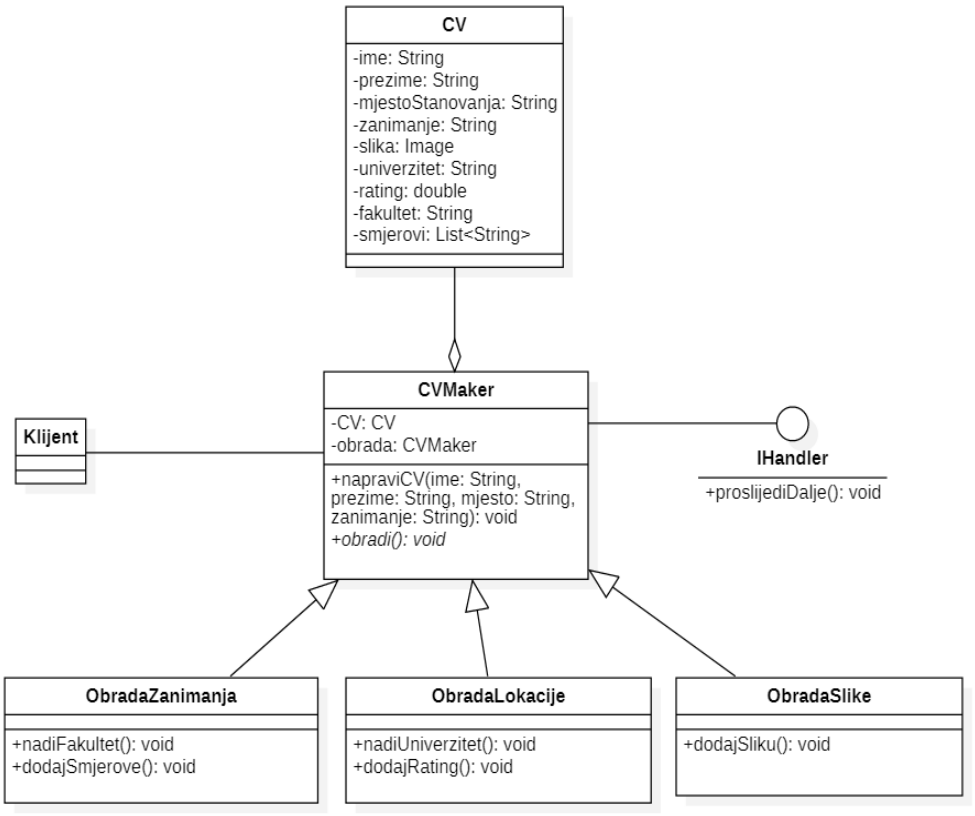
```
            player.kreirajIterator(0, pjesme);
            Console.Out.WriteLine("Trenutni iterator: Abecedni");
            for (int i = 0; i < 5; i++)
            {
                p1 = player.pustiSljedecuPjesmu();
                Console.Out.WriteLine("Trenutno je puštena pjesma: " + p1.Naziv + "
koju izvodi " + p1.Izvodac);
            }

            player.kreirajIterator(1, pjesme);
            Console.Out.WriteLine("\nTrenutni iterator: Shuffle");
            for (int i = 0; i < 5; i++)
            {
                p1 = player.pustiSljedecuPjesmu();
                Console.Out.WriteLine("Trenutno je puštena pjesma: " + p1.Naziv + "
koju izvodi " + p1.Izvodac);
            }

            player.kreirajIterator(2, pjesme);
            Console.Out.WriteLine("\nTrenutni iterator: Žanr");
            for (int i = 0; i < 5; i++)
            {
                p1 = player.pustiSljedecuPjesmu();
                Console.Out.WriteLine("Trenutno je puštena pjesma: " + p1.Naziv + "
koju izvodi " + p1.Izvodac);
            }
```

6. *Chain of responsibility* patern

Za sljedeću definiciju sistema potrebno je osmisliti dijagram klasa koji poštuje *chain of responsibility* patern.

Klijent želi automatski napraviti svoj CV. O sebi će napisati samo osnovne informacije (ime, prezime, mjesto stanovanja, zanimanje), a CVMaker će za njega dodati cve dijelove CV-a. Prvo će na internetu pronaći sliku za dato ime i prezime. Zatim će dodati univerzitet za specificirani grad i njegov rating na listi svjetskih univerziteta. Nakon toga dodati će informacije o fakultetu iz dobivenog univerziteta te smjerovima koje fakultet nudi. Klijentu će se kao rezultat dati konačni CV.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 6.



Slika 6. Dijagram klasa sa korištenjem chain of responsibility paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public class CV
    {
        string ime, prezime, mjestoStanovanja, zanimanje, univerzitet, fakultet,
slika;
        double rating;
        List<string> smjerovi = new List<string>();

        public string Ime { get => ime; set => ime = value; }
        public string Prezime { get => prezime; set => prezime = value; }
        public string MjestoStanovanja { get => mjestoStanovanja; set =>
mjestoStanovanja = value; }
        public string Zanimanje { get => zanimanje; set => zanimanje = value; }
        public string Univerzitet { get => univerzitet; set => univerzitet = value;
}
        public string Fakultet { get => fakultet; set => fakultet = value; }
        public string Slika { get => slika; set => slika = value; }
        public double Rating { get => rating; set => rating = value; }
        public List<string> Smjerovi { get => smjerovi; set => smjerovi = value; }
    }

    public interface IHandler
    {
        public void proslijediDalje();
    }
```

```
    public class CVMaker : IHandler
    {
        CV cv = new CV();
        CVMaker obrada;

        public CV Cv { get => cv; set => cv = value; }

        public void napraviCV(string ime, string prezime, string mjesto, string
zanimanje)
        {
            Cv.Ime = ime;
            Cv.Prezime = prezime;
            Cv.MjestoStanovanja = mjesto;
            Cv.Zanimanje = zanimanje;
        }
        public void proslijediDalje()
        {
            if (obrada is null)
                obrada = new ObradaSlike(Cv);
            else if (obrada is ObradaSlike)
                obrada = new ObradaLokacije(Cv);
            else if (obrada is ObradaLokacije)
                obrada = new ObradaZanimanja(Cv);
        }
        public virtual void obradi()
        {
            obrada.obradi();
            Cv = obrada.Cv;
        }

    public class ObradaSlike : CVMaker
    {
        public ObradaSlike(CV c)
        {
            Cv = c;
        }
        public void dodajSliku()
        {
            Console.Out.WriteLine("Pretraga slika s interneta...");
            Cv.Slika = "Slika s interneta";
        }
        public override void obradi()
        {
            dodajSliku();
        }
    }

    public class ObradaLokacije : CVMaker
    {
        public ObradaLokacije(CV c)
        {
            Cv = c;
        }
        public void nadiUniverzitet()
        {
            Console.Out.WriteLine("Pretraga najbližih univerziteta...");
            Cv.Univerzitet = "Najbliži univerzitet";
        }
```

```
        public void dodajRating()
        {
            Console.Out.WriteLine("Pretraga ratinga za univerzitete...");
            Cv.Rating = 10;
        }
        public override void obradi()
        {
            nadiUniverzitet();
            dodajRating();
        }
    }

    public class ObradaZanimanja : CVMaker
    {
        public ObradaZanimanja(CV c)
        {
            Cv = c;
        }
        public void nadiFakultet()
        {
            Console.Out.WriteLine("Pretraga fakulteta...");
            Cv.Fakultet = "Fakultet";
        }
        public void dodajSmjerove()
        {
            Console.Out.WriteLine("Pretraga smjerova...");
            Cv.Smjerovi = new List<string>() { "Smjer 1", "Smjer 2" };
        }
        public override void obradi()
        {
            nadiFakultet();
            dodajSmjerove();
        }
    }

        static void Main(string[] args)
        {
            CVMaker cvMaker = new CVMaker();
            cvMaker.napraviCV("Ime", "Prezime", "UNSA", "Inžinjer elektrotehnike");

            Console.Out.WriteLine("-----Prva obrada-----");
            cvMaker.proslijediDalje();
            cvMaker.obradi();
            CV cv = cvMaker.Cv;
            Console.Out.WriteLine("Trenutni CV: " + cv.Ime + ", " + cv.Prezime + ",
" + cv.MjestoStanovanja + ", "
                + cv.Univerzitet + ", " + cv.Rating + ", " + cv.Fakultet + ", " +
cv.Zanimanje + ", " + cv.Slika);

            Console.Out.WriteLine("-----Druga obrada-----");
            cvMaker.proslijediDalje();
            cvMaker.obradi();
            cv = cvMaker.Cv;
            Console.Out.WriteLine("Trenutni CV: " + cv.Ime + ", " + cv.Prezime + ",
" + cv.MjestoStanovanja + ", "
                + cv.Univerzitet + ", " + cv.Rating + ", " + cv.Fakultet + ", " +
cv.Zanimanje + ", " + cv.Slika);

            Console.Out.WriteLine("-----Treća obrada-----");
```

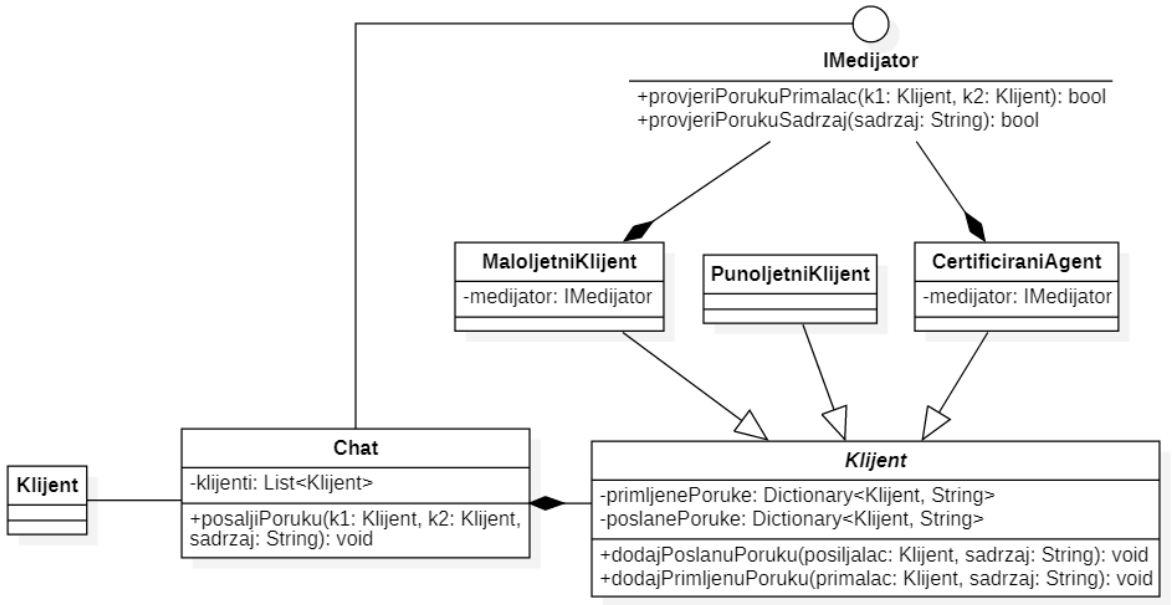
```
            cvMaker.proslijediDalje();
            cvMaker.obradi();
            cv = cvMaker.Cv;
            Console.Out.WriteLine("Trenutni CV: " + cv.Ime + ", " + cv.Prezime + ",
" + cv.MjestoStanovanja + ", "
                + cv.Univerzitet + ", " + cv.Rating + ", " + cv.Fakultet + ", " +
cv.Zanimanje + ", " + cv.Slika);
        }
```

7. Medijator patern

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

Chat platforma omogućava klijentima da šalju jedni drugima poruke. Maloljetni klijenti mogu slati poruke samo drugim maloljetnim klijentima i certificiranim agentima korisničke podrške. Poruke za certificirane agente se provjeravaju i odbacuju ukoliko imaju neprimjereni sadržaj. Punoljetni klijenti mogu međusobno razgovarati bez ikakvih provjera.

Ovako definisanom sistemu odgovara dijagram klasa prikazan na Slici 7.



Slika 7. Dijagram klasa sa korištenjem medijator paterna

U isječku koda ispod prikazan je način implementacije ovakvog sistema.

```
    public abstract class Klijent
    {
        Dictionary<Klijent, string> primljenePoruke = new Dictionary<Klijent,
string>(), poslanePoruke = new Dictionary<Klijent, string>();

        public Dictionary<Klijent, string> PoslanePoruke { get => poslanePoruke; set
=> poslanePoruke = value; }
        public Dictionary<Klijent, string> PrimljenePoruke { get => primljenePoruke;
set => primljenePoruke = value; }
```

```
        public virtual void dodajPoslanuPoruku(Klijent primalac, string sadržaj)
        {
            poslanePoruke.Add(primalac, sadržaj);
        }
        public virtual void dodajPrimljenuPoruku(Klijent pošiljalac, string
sadržaj)
        {
            primljenePoruke.Add(pošiljalac, sadržaj);
        }
    }

    public class MaloljetniKlijent : Klijent
    {
        IMedijator medijator;

        public IMedijator Medijator { get => medijator; set => medijator = value; }

        public override void dodajPoslanuPoruku(Klijent primalac, string sadržaj)
        {
            if (Medijator.provjeriPorukuPrimalac(this, primalac))
                PoslanePoruke.Add(primalac, sadržaj);
        }
        public override void dodajPrimljenuPoruku(Klijent pošiljalac, string
sadržaj)
        {
            if (Medijator.provjeriPorukuPrimalac(this, pošiljalac))
                PrimljenePoruke.Add(pošiljalac, sadržaj);
        }
    }

    public class PunoljetniKlijent : Klijent { }

    public class CertificiraniAgent : Klijent
    {
        IMedijator medijator;

        public IMedijator Medijator { get => medijator; set => medijator = value; }

        public override void dodajPrimljenuPoruku(Klijent pošiljalac, string
sadržaj)
        {
            if (Medijator.provjeriPorukuSadrzaj(sadržaj))
                PrimljenePoruke.Add(pošiljalac, sadržaj);
        }
    }

    public interface IMedijator
    {
        public bool provjeriPorukuPrimalac(Klijent k1, Klijent k2);
        public bool provjeriPorukuSadrzaj(string sadrzaj);
    }

    public class Chat : IMedijator
    {
        List<Klijent> klijenti;
        public Chat(List<Klijent> k)
        {
            foreach (var client in k)
```

```
            {
                if (client is MaloljetniKlijent)
                    ((MaloljetniKlijent)client).Medijator = this;
                else if (client is CertificiraniAgent)
                    ((CertificiraniAgent)client).Medijator = this;
            }
            klijenti = k;
        }
        public bool provjeriPorukuPrimalac(Klijent k1, Klijent k2)
        {
            bool maloljetniAgentu = (k1 is MaloljetniKlijent && k2 is
CertificiraniAgent);
            bool agentMaloljetnom = (k1 is CertificiraniAgent && k2 is
MaloljetniKlijent);
            return (maloljetniAgentu || agentMaloljetnom);
        }
        public bool provjeriPorukuSadrzaj(string sadrzaj)
        {
            return !sadrzaj.Contains("Govor mržnje");
        }
        public void posaljiPoruku(Klijent k1, Klijent k2, string sadrzaj)
        {
            k1.dodajPoslanuPoruku(k2, sadrzaj);
            k2.dodajPrimljenuPoruku(k1, sadrzaj);
        }
    }

        static void Main(string[] args)
        {
            Klijent maloljetni = new MaloljetniKlijent();
            Klijent punoljetni = new PunoljetniKlijent();
            Klijent agent = new CertificiraniAgent();
            Chat chat = new Chat(new List<Klijent> { maloljetni, punoljetni, agent
});

            chat.posaljiPoruku(punoljetni, maloljetni, "Zdravo!");
            Console.Out.WriteLine("Broj poruka u inboxu maloljetnog klijenta: " +
maloljetni.PrimljenePoruke.Count);

            chat.posaljiPoruku(maloljetni, agent, "Govor mržnje");
            Console.Out.WriteLine("Broj poruka u inboxu certificiranog agenta: " +
agent.PrimljenePoruke.Count);

            chat.posaljiPoruku(punoljetni, agent, "OK");
            Console.Out.WriteLine("Broj poruka u inboxu certificiranog agenta: " +
agent.PrimljenePoruke.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* **lv9-2**.

Prepoznati i primijeniti odgovarajuće paterne na sljedeće opise sistema:

1. Klijent želi da pušta pjesme na računaru. Svaka pjesma prvenstveno je zaustavljena, a kada se klikne na *play*, počinje reprodukcija. Pjesmu je moguće pauzirati, čime postaje zaustavljena, a ako se vrati na početak, reprodukcija počinje ispočetka. Svaki put kada ponovo počne reprodukcija, klijent treba dobiti notifikaciju da je pjesma puštena.

2. Klijent igra igricu i može odabrati odbrambenu ili napadačku taktiku. Ako odabere odbrambenu taktiku, resursi će se minimalno trošiti i pri svakoj borbi odabrati će se bijeg, a s napadačkom taktikom resursi će se maksimalno trošiti i uvijek će se odabirati borba. Svaka borba sastoji se od redova koji se iznova ponavljaju dok neko ne ostane bez *health* poena. Svaki red počinje sa odabirom magije heroja (odbrambeni i napadački heroji imaju različite magije), zatim sa postavljanjem vojnika na polje (koje je isto za sve heroje) i odabirom oružja (koje je omogućeno samo napadačkim herojima).