Elektrotehnički fakultet Sarajevo

Odsjek za računarstvo i informatiku

Razvoj programskih rješenja

Primjer izrade ispitnog zadatka

Napomena: ovaj materijal ne treba služiti kao *manual* za pripremu ispita već isključivo kao primjer jednog od načina na koje je moguće pristupiti konkretnom problemu.

1. Tekst problema

Napisati konzolnu aplikaciju koja će omogućiti knjižari M iznajmljivanje knjiga. Knjige se opisuju nazivom, autorom, godinom izdavanja i identifikacijskom brojem, tipom knjige. Za sada postoje dva tipa knjiga Naučna i Zabavna. Ukoliko je knjiga Naučna zapisuje se i grana nauke, a ukoliko je Zabavna dodatno se zapisuje za koju starosnu dob osoba je namijenjena. Knjige naučnog tipa se mogu iznajmiti do 10 dana za cijenu od 5 KM, a za svaki dodatni dan se plaća 0.5 KM ukoliko je godina izdavanja knjige prije 1990, a ako je poslije 1990 plaća se 0.3KM. Knjige zabavnog tipa se mogu iznajmiti do 5 dana za cijenu od 4 KM, a za svaki dodatni dan se plaća 2% od osnovne cijene ukoliko je starosnu dob iznad 18 godina.

Program treba omogućiti unos podataka o knjigama preko odgovarajućeg menija i obračun na zahtjev osobe koja želi iznajmiti knjigu. Osoba navodi koju knjigu želi i koliko dana je želi zadržati a program ispisuje koliko je potrebno platiti za iznajmljivanje te knjige. Definirajte klase i veze između klasa koje će odgovarati opisanom problem.

S obzirom da se očekuje da će knjižara M uvesti i nove tipove knjiga i obračune, da bi se lakše modifikovao napisani program dizajnirati hijerarhiju klasa tako da je na vrhu hijerarhije apstraktna klasa.

Potrebno je primjenom programske dekompozicije napisati program, raditi u programskom jeziku C# i obavezno koristiti polimorfizam. Testirati program tako da se evidentira po 3 knjige za svaki tip (voditi računa o godini izdavanja i starosnoj dobi) i nakon toga da se i vrši njihove iznajmljivanje na 3 i 7 dana.

2. Analiza problema i dizajn rješenja

U prvom koraku iz postavke problema identificiramo klase i njihove atribute. Imenice koje se spominju su: knjižara, knjiga (u raznim varijantama) i osoba. Osoba otpada kao potencijalna klasa zbog činjenice da se od programskog rješenja nigdje u postavci ne traži unos klijenata i njihova pohrana, iako u izračunima figurira starost osobe. Štaviše, mehanizam zaduživanja knjiga svodi se na obavljanje izračuna, dok se nigdje ne bilježi raspoloživost (količinska) određenog izdanja neke knjige. Dakle, imamo knjižaru kao kolekciju raznih knjiga, te knjigu u njene dvije varijante.

Knjige se specijaliziraju u naučne i zabavne knjige što je očigledno mjesto gdje koristimo naslijeđivanje, dok je knjižara sada polimorfna kolekcija. Imajući u vidu i eksplicitne zahtijeve da je knjiga apstraktna klasa, formalno definišemo klase i njihove atribute:

+ Knjižara (lista knjiga: lista objekata tipa knjiga)

+ Knjiga (ID: int, autor:string, naziv:string, godina izdavanja:int/datetime)

+ NaučnaKnjiga: Knjiga (grana nauke: string)

+ ZabavnaKnjiga: Knjiga (ciljana dob: int)

Sada razmatramo mehanizam naplate: podaci koji se koriste prilikom naplate iznajmljivanja knjige su: osobine osobe koja iznajmljuje knjigu (starosna dob) i broj dana na koje se želi iznajmiti knjigu (u općem slučaju). Pošto je ovaj slučaj krajnje pojednostavljen, koristi se samo jedan atribut osobe tj. starost osobe.

Pošto postoje različiti načini obračuna za različite tipove knjiga, potrebna nam je apstraktna metoda izracunajIznosZaNaplatu koju će ona svaka specijalizacija knjige implementirati na sebi svojstven način.

Sada je preostalo još protumačiti način naplate:

- za naučne knjige, if brojDana≤10 cijena=5 else { if izdanje≤1990 cijena=5+prekoračenje*0.5 else cijena=5+prekoračenje*.3}
- za zabavne knjige, if brojDana≤5 || dob≤18 cijena =4 else if dob≥18 cijena=4+prekoračenje*0.08

3. Implementacija rješenja

Očigledno, potrebno je kreirati konzolni meni koji će sadržavati minimalno tri opcije: registriranje zabavne knjige, naučne knjige i izračun iznosa za naplatu (eventualno i izlaz iz aplikacije).

Prvo implementiramo klase.

Primijetiti da nije implementirana provjera novih vrijednosti koje se dodjeljuje atributima u setterima - npr. da ne bude moguće postaviti null naziv ili godinu izdanja iz budućnosti i slično. Navedene provjere se ne traže eksplicitno zadatkom ali se u izradi stvarnih aplikacija podrazumijevaju. Studentima se ostavlja za vježbu da implementiraju i odgovarajuće provjere.

```
    public abstract class Knjiga
    {
        public int ID { get; set; }
        public String Naziv { get; set; }
        public String Autor { get; set; }
        public DateTime GodinaIzdanja { get; set; }

        public Knjiga(int ID, String Naziv, String Autor, DateTime GodinaIzdanja)
        {
            this.ID = ID;
            this.Naziv = Naziv;
            this.Autor = Autor;
            this.GodinaIzdanja = GodinaIzdanja;
        }

        abstract public Decimal izracunajIznosZaNaplatu(int brojDanaNajma, int
starostOsobe);
    }



    public class KnjigaNaucna : Knjiga
    {
        public String GranaNauke { get; set; }

        private static int maxDanaBezDodatneNaplate = 10;
        private static Decimal baznaCijena = 5;
        private static int granicnaGodinaIzdanja = 1990;
        private static Decimal koeficijentZaStarijeOdGranice = 0.5m;
        private static Decimal koeficijentZaMladjeOdGranice = 0.3m;
```

```
        public KnjigaNaucna(int ID, String Naziv, String Autor, DateTime GodinaIzdanja,
String GranaNauke) :
            base(ID, Naziv, Autor, GodinaIzdanja)
        {
            this.GranaNauke = GranaNauke;
        }

        override public Decimal izracunajIznosZaNaplatu(int brojDanaNajma, int
starostOsobe)
        {
            if (brojDanaNajma<=0)
                throw new ArgumentException("Ne moze se iznajmiti na negativan broj
dana!");
            if (starostOsobe<=0)
                throw new ArgumentException("Osoba ne moze imati negativnu starost!");

            int diff = brojDanaNajma-maxDanaBezDodatneNaplate;
            if (diff<=0)
                return baznaCijena;
            else
            {
                return baznaCijena + diff * ((GodinaIzdanja.Year <= granicnaGodinaIzdanja)
? koeficijentZaStarijeOdGranice : koeficijentZaMladjeOdGranice);
            }
        }
    }

    public class KnjigaZabavna : Knjiga
    {
        public int CiljanaDob { get; set; }

        private static int maxDanaBezDodatneNaplate = 5;
        private static Decimal baznaCijena = 4;
        private static Decimal koeficijentZaPrekoracenje = 0.08m;
        private static int granicnaDobOsobe = 18;

        public KnjigaZabavna(int ID, String Naziv, String Autor, DateTime GodinaIzdanja,
int CiljanaDob) :
            base(ID, Naziv, Autor, GodinaIzdanja)
        {
            this.CiljanaDob = CiljanaDob;
        }

        override public Decimal izracunajIznosZaNaplatu(int brojDanaNajma, int
starostOsobe)
        {
            if (brojDanaNajma <= 0)
                throw new ArgumentException("Ne moze se iznajmiti na negativan broj
dana!");
            if (starostOsobe <= 0)
                throw new ArgumentException("Osoba ne moze imati negativnu starost!");

            int diff = brojDanaNajma - maxDanaBezDodatneNaplate;
            if (diff <= 0 || starostOsobe<=granicnaDobOsobe)
                return baznaCijena;
            else
            {
                return baznaCijena + diff * koeficijentZaPrekoracenje;
```

```
            }
        }
    }
```

Sada kad smo napravili osnovne podatkovne modele, razvijamo kontejnersku klasu *Knjižara*.

```
    public class Knjizara
    {
        private List<Knjiga> knjige;

        public Knjizara()
        {
            knjige = new List<Knjiga>();
        }

        public void registrirajKnjigu(Knjiga novaKnjiga)
        {
            if (novaKnjiga == null)
                throw new ArgumentException("Parametar ne smije biti null");

            // provjera da li već postoji ista knjiga
            if (knjige.Any(postojecaKnjiga => postojecaKnjiga.ID == novaKnjiga.ID))
            {
                throw new ArgumentException("Knjiga već unesena");
            }

            knjige.Add(novaKnjiga);
        }

        public Decimal iznosNajmaKnjige(int IDKnjige, int brojDana, int starostOsobe)
        {
            Knjiga registriranaKnjiga = knjige.FirstOrDefault(knjiga => knjiga.ID ==
IDKnjige);
            if (registriranaKnjiga == null)
                throw new ArgumentException("Ne postoji knjiga sa datim ID!");
            return registriranaKnjiga.izracunajIznosZaNaplatu(brojDana, starostOsobe);
        }
    }
```

Sada možemo kreirati glavni program. Napomena: glavni program se sigurno može bolje dizajnirati, a nisu ni implementirane validacije ulaznih podataka (npr. ako se umjesto očekivanog integera unese slovo i slično). Validacije nisu eksplicitno tražene ali se u stvarnim projektima podrazumijevaju. Ovdje nisu rađene radi jednostavnosti rješenja.

```
class Program
    {
        static void Main(string[] args)
        {
            knjizara = new Knjizara();
            String izabranaOpcija;
            do
            {
                ispisiMeni();
                izabranaOpcija = Console.ReadLine();
                if (izabranaOpcija == "1")
                    unosNaucneKnjige();
                else if (izabranaOpcija == "2")
                    unosZabavneKnjige();
                else if (izabranaOpcija == "3")
                    izracunIznosa();
                else if (izabranaOpcija == "4")
                    Console.WriteLine("Byee!");
                else
                    Console.WriteLine("Unrecognized input.");
            } while (izabranaOpcija != "4");
        }

        static Knjizara knjizara;

        static void ispisiMeni()
        {
            Console.WriteLine("1 - unos naučne knjige" + Environment.NewLine +
                "2 - unos zabavne knjige" + Environment.NewLine +
                "3 - izracun najma" + Environment.NewLine +
                "4 - izlaz");
            Console.Write("Odaberite opciju i pritisnite ENTER: ");
        }

        static void unosNaucneKnjige()
        {
            Console.WriteLine(Environment.NewLine + "----- sada unosite naucnu knjigu ----
");
            Console.WriteLine("Unesite redom: ID, naziv knjige, autora knjige, godinu
izdanja u formatu YYYY i naucnu oblast, iza svakog ENTER");
            int id;
            Int32.TryParse(Console.ReadLine(), out id);
            String naziv = Console.ReadLine();
            String autor = Console.ReadLine();
            int godinaizdanja;
            Int32.TryParse(Console.ReadLine(), out godinaizdanja);
            String oblast = Console.ReadLine();

            KnjigaNaucna kn = new KnjigaNaucna(id, naziv, autor, new
DateTime(godinaizdanja, 1, 1), oblast);

            try
            {
                knjizara.registrirajKnjigu(kn);
            }
            catch (Exception ex)
            {
                Console.WriteLine("--- error ---" + Environment.NewLine + ex.Message);
```

```
            }

            Console.WriteLine("----- zavrsen unos naucne knjige ----");
        }

        static void unosZabavneKnjige()
        {
            Console.WriteLine(Environment.NewLine + "----- sada unosite zabavnu knjigu ---
-");
            Console.WriteLine("Unesite redom: ID, naziv knjige, autora knjige, godinu
izdanja u formatu YYYY i ciljanu dob, iza svakog ENTER");
            int id;
            Int32.TryParse(Console.ReadLine(), out id);
            String naziv = Console.ReadLine();
            String autor = Console.ReadLine();
            int godinaizdanja;
            Int32.TryParse(Console.ReadLine(), out godinaizdanja);
            int ciljanaDob;
            Int32.TryParse(Console.ReadLine(), out ciljanaDob);

            KnjigaZabavna kz = new KnjigaZabavna(id, naziv, autor, new
DateTime(godinaizdanja, 1, 1), ciljanaDob);

            try
            {
                knjizara.registrirajKnjigu(kz);
            }
            catch (Exception ex)
            {
                Console.WriteLine("--- error ---" + Environment.NewLine + ex.Message);
            }

            Console.WriteLine("----- zavrsen unos zabavne knjige ----");
        }

        static void izracunIznosa()
        {
            Console.WriteLine("----- sada zahtijevate izracun iznosa -----");
            Console.WriteLine("Unesite ID knjige, broj dana najma i starosnu dob osobe,
svako praceno enterom");
            int idKnjige, danaNajma, starost;
            Int32.TryParse(Console.ReadLine(), out idKnjige);
            Int32.TryParse(Console.ReadLine(), out danaNajma);
            Int32.TryParse(Console.ReadLine(), out starost);

            try
            {
                Decimal result = knjizara.iznosNajmaKnjige(idKnjige, danaNajma, starost);
                Console.WriteLine("Za naplatiti: " + result);
            }
            catch (Exception ex)
            {
                Console.WriteLine("--- error ---" + Environment.NewLine + ex.Message);
            }
        }
    }
```

Diskusija:

U ovom zadatku od studenata se moglo tražiti da u rješenju obavezno iskoriste i interfejs. Razmotrimo način na koji se to moglo ostvariti.

Podsjetimo se na činjenicu da interfejsi služe kao ugovor kojeg klase mogu prihvatiti (implementirati) i onda je klasa obavezna imati metode (i propertije i evente) specificirane interfejsom. U ovom slučaju mogli smo imati interfejs iNaplativo koji bi imao metodu naplati, međutim njeni parametri nam predstavljaju problem. Naime, iNaplativo je toliko općenito da se odnosi na bilo što što je moguće naplatiti tj. uvjeti pod kojima se obavlja naplata knjiga u postavljenom zadatku je izuzetno specifična (traži isključivo broj dana najma i starost osobe). Interfejs u takvim uvjetima nudi tek blago poopćenje i ne možemo ga nazvati iNaplativo već nešto poput iNaplativoNaOsnovuStarostiOsobeIBrojaDanaNajma. Tako bi interfejs se mogao odnositi i npr. na rentanje stanova i slično.

Implementacija je u svakom slučaju (bez obzira na smislenost takvog zahtijeva) veoma jednostavna - deklarira se interfejs sa metodom naplati koja prima potrebne parametre (dva integera) i vraća decimalni broj (iznos za naplatu), a potom Knjiga implementira interfejs i deklarira apstraktnu metodu naplati iz interfejsa koja je public i onda se u izvedenim klasama vrši override metode dok se kôd metoda suštinski ne mijenja. Može se poopćiti i knjižaru tako da ne sadrži listu knjiga već listu iNaplativih objekata, ali uz određena ograničenja.

```
    public interface iNaplativoPosebno
    {
        Decimal izracunajIznosZaNaplatu(int brojDanaNajma, int starostOsobe);
    }

    public abstract class Knjiga : iNaplativoPosebno
    {
        public int ID { get; set; }
        public String Naziv { get; set; }
        public String Autor { get; set; }
        public DateTime GodinaIzdanja { get; set; }

        public Knjiga(int ID, String Naziv, String Autor, DateTime GodinaIzdanja)
        {
            this.ID = ID;
            this.Naziv = Naziv;
            this.Autor = Autor;
            this.GodinaIzdanja = GodinaIzdanja;
        }

        abstract public Decimal izracunajIznosZaNaplatu(int brojDanaNajma, int
starostOsobe);
    }
```

Nikakve dalje promjene nisu potrebne što se tiče knjiga. U knjižari se može držati lista iNaplativo objekata, međutim problem nastaje zbog činjenice da ih je potrebno porediti po ID jer se kroz interfejs vidi samo metoda izračunavanja iznosa za naplatu. U toj situaciji postoje dvije varijante: ili vršiti pretvorbu između tipova (lošija) ili interfejs proširiti metodom :

Boolean uporedi(iNaplativo saKojimSePoredi)

Navedena metoda bi opet bila apstraktna u Knjizi, dok bi specijalizacije knjige overrideale istu i obavljali vlastito poređenje. Studentima se ostavlja da sami implementiraju navedeno rješenje.

Napomena: Na ispitu ne treba raditi validaciju, ako eksplicitno nije naglaseno podaci u listu se mogu unijeti preko konstruktora.!