Objektno orijentisano programiranje 2

Tipovi podataka u C#

Klasifikacija tipova

- Osnovna podela na:
  - vrednosne (*value*) tipove
  - ukazane (*reference*) tipove
- Vrednosni tipovi:
  - jednostavni tipovi (kao što su npr. `byte`, `int`, `long`, `float`, `double`)
  - nabrajanja
  - strukture
- Ukazani tipovi:
  - klase
  - interfejsi
  - nizovi
  - delegati
- Svi tipovi (uključujući jednostavne kao što je `int`) su podtipovi `System.Object`

Vrednosni i ukazani tipovi

- Razlike između vrednosnih i ukazanih tipova:
  - alokacija memorije:
    - vrednosni tipovi – na steku, odnosno u okviru objekta, ako su članovi ukazanih tipova na hipu
    - ukazani tipovi – na hipu
  - sadržaj:
    - vrednosni tipovi – podatak
    - ukazani tipovi – pokazivač (referenca) na podatak
  - uništenje:
    - vrednosni tipovi – odmah pošto se napusti oblast definisanosti
    - ukazani tipovi – sakupljač đubreta
- Most između vrednosnih i ukazanih tipova
  - mehanizam pakovanja (*boxing*) i raspakivanja (*unboxing*)
  - efekat sličan korišćenju klasa-omotača (npr. `Integer`) na Javi

Pakovanje (*boxing*)

- Pakovanje je mehanizam koji od vrednosnog podatka pravi ukazani objekat
- Mehanizam:
  - pravi se primerak objekta na hipu u koji se kopira vrednosni podatak
- Primer – pakovanje `int` promenljive:

```
int i=10; object o=i;
System.Console.WriteLine("i="+i+", o="+o);
```

- Primer – pakovanje `long` literala:

```
object longObj = 1000L;
```

- Strukture se mogu konvertovati u tipove interfejsa koje implementiraju
- Primer – pakovanje `struct` podatka `S` koji implementira interfejs `I`:

```
S s=new S(); I i=s;
```

- Implicitno pakovanje:
  - prilikom dodele vrednosti ili inicijalizacije (kao u gornjim primerima)
  - prilikom prosleđivanja vrednosnog argumenta gde se očekuje ukazani objekat
  - prilikom pozivanja metoda strukture

Raspakivanje (*unboxing*)

- Obrnut proces od pakovanja
- Od objekta koji sadrži prethodno spakovanu vrednost se pravi podatak vrednosnog tipa
- Nije moguće raspakivanje bilo kog objekta (onog koji ne sadrži spakovanu vrednost)
- Primer – pakovanje i raspakivanje `int` promenljive:

```
int i=10; object o=i; int ii=(int)o;
System.Console.WriteLine("i="+i+", o="+o+", ii="+ii);
```

- Neophodna je eksplicitna konverzija (*cast*)
- Izvršni sistem proverava tip konverzije
  - mora da odgovara tipu spakovane vrednosti
  - ako se ne koristi odgovarajuća konverzija
    - izuzetak `System.InvalidCastException`
- Ako su performanse bitne
  - treba izbegavati pakovanje/raspakivanje, jer troši vreme

Vrednosni tipovi

- Vrednosni tipovi na C# su strukture i nabrajanja
- Java nije imala ni strukture ni nabrajanja
  - u Javi 1.5 su uvedeni tipovi nabrajanja
- Svi ugrađeni vrednosni tipovi u C# su strukture
  - čak su jednostavni tipovi: `int`,... strukture
  - bitna razlika u odnosu na Javu koja poseduje primitivne tipove (`int`,...)
    - ali poseduje i odgovarajuće klase omotača (`Integer`,...)
  - važno za realizaciju mehanizma pakovanja ugrađenih tipova
- Vrednosni tipovi su izvedeni iz `System.ValueType`
  - koji je izveden iz `System.Object`

Strukture

- Slične klasama, ali su vrednosni, a ne ukazani tipovi
- Razlike u odnosu na klase:
  - primerci se alociraju na steku, ili u okviru objekta na hipu, ako je struktura član
  - memorija alocirana primerku strukture sadrži članove podatke, ne referencu
  - zauzeta memorija se oslobađa odmah nakon napuštanja dosega, mimo sakupljača đubreta
- Definicija:
  `[<atributi>][<modifikatori>] struct <identifikator> [:<interfejsi>]{<metodi i polja>}`
- Nasleđivanje:
  - iz njih se ne može izvoditi niti se one izvode (osim implicitno iz `System.ValueType`)
  - mogu da implementiraju interfejse

Jednostavni tipovi

- Implementirani kao strukture (vrednosni tipovi)
- Jezik definiše ključne reči (`int`, ...)
  kao sinonime za ugrađene jednostavne tipove
- Na primer:
  `int` je sinonim za `System.Int32`,
  `float` je sinonim za `System.Single`
- Odgovaraju primitivnim tipovima na Javi, ali je skup bogatiji
  - postoje kardinalni (neoznačeni, *unsigned*) celobrojni tipovi
  - tip `decimal` za preciznu aritmetiku u fiksnom zarezu
    (128 bita, 28 značajnih cifara)
- Na raspolaganju su:
  `bool`, `sbyte`, `byte`, `short`, `ushort`, `int`, `uint`, `long`, `ulong`,
  `float`, `double`, `decimal`, `char`

Nabrajanja

- Tip definiše skup imenovanih celobrojnih konstanti
- Definicija:
  `[<atributi>][<modifikatori>]enum<ident>[:<osnova>]{<telo>}`
- Osnova može biti bilo koji celobrojni tip, podrazumevano je `int`
- Telo sadrži listu imena članova sa opcionim vrednostima
- Imena članova moraju biti jedinstvena, ali više članova može imati istu vrednost
- Podrazumevano, prvi član ima vrednost `0`, a naredni članovi veće vrednosti redom
- Primer – boje karata u preferansu:
```
public enum BojeKarata:byte
{Pik=2,Karo,Herc,Tref,Min=Pik,Max=Tref}
```
- Korišćenje: `BojeKarata.Herc`
- Celobrojna konstanta `0` je implicitno konvertibilna u bilo koji tip nabrajanja
  - može se uvek proslediti kao argument tamo gde se očekuje podatak tipa nabrajanja
- Nasleđivanje:
  - iz njih se ne može izvoditi niti se oni izvode
  - izvedeni su implicitno iz `System.Enum` koja je izvedena iz `System.ValueType`

Ukazani tipovi

- Podaci ukazanih tipova (*reference types*):
  - stvaraju se na hipu
  - uklanjaju se automatski od strane sakupljača đubreta
- U ukazane tipove spadaju:
  - klase
  - interfejsi
  - nizovi
  - delegati
- Klase, interfejsi i nizovi su poznati koncepti, na jeziku C# je slična podrška kao u Javi
- Delegati – novi koncept
  - bezbedno pozivanje metoda po referenci

Klase

- Različita sintaksa za izvođenje i implementaciju interfejsa nego u Javi
  - koristi se, kao u jeziku C++, simbol `:`
- Prvo se eventualno navodi bazna klasa (može biti samo jedna), pa interfejsi
- Primer:

```
public class IzvedenaKlasa:OsnovnaKlasa,
                           Interfejs1, Interfejs2 {...}
```

- Na C# nema lokalnih ni anonimnih klasa
- Apstraktna klasa mora da naznači
  da je metod interfejsa koji nije realizovan apstraktan
- Podrazumevani konstruktor je uvek javni,
  osim za apstraktnu klasu, gde je zaštićen
- Postoji metod `Object.Finalize()` koji se ne poziva i ne redefiniše
- Metod `Object.Finalize()` se implicitno poziva iz destruktora
  - ista sintaksa destruktora kao u C++

Promenljive i parametri metoda

- Kao i Java, C# je strogo tipiziran jezik
  - zahteva se da svaka promenljiva ima tip
- Postoji 7 vrsta promenljivih, od kojih su 3 vrste parametara:
  - statičko polje (promenljiva klase)
  - nestatičko polje (promenljiva objekta)
  - element niza
  - lokalna promenljiva
  - parametar
    - po vrednosti
    - po referenci
    - izlazni
- Podrazumevano – parametri se prenose po vrednosti, isto kao u Javi
  - pravi se kopija stvarnog argumenta
  - ako se kopija menja to ne utiče na stvarni argument

Parametri `ref` i `out`

- Za prenos po referenci se koristi ključna reč `ref`, a za izlazni `out`
- Ove ključne reči se koriste i u definiciji metoda i na mestu poziva
- Prenos parametara po referenci omogućava bočne efekte metoda nad njima:
  - ne pravi se kopija stvarnog argumenta već se izmene vrše nad njim
- Izlazni parametri ne moraju da budu inicijalizovani pre prosleđivanja metodu
- Ako se izlaznom parametru ne pridruži vrednost u metodu – greška
- Primer:

```
public class C{
  public static void M(out int x){x=5;}
  public static void Main(){int x; C.M(out x);}
}
```

Delegati

- Bezbedan mehanizam za prosleđivanje “reference na metod(e)” kao parametra
  - prosleđivanje pokazivača na metod i poziv metoda preko pokazivača ne bi bili bezbedni
- Koriste se primarno za obradu događaja i povratne pozive (*callback*)
- Primerci (objekti, instance) delegata sadrže reference na jedan ili više metoda
  - formira se lista poziva (*invocation list*)
  - metodi se stavljaju na listu poziva
    - u konstruktoru delegata ili
    - operatorima `+`, `+=`
  - uklanjaju sa liste poziva
    - operatorima `-`, `-=`
  - metodi sa liste pozivaju se preko delegata
- Definicija:
  `[<atributi>][<modifikatori>]delegate<tip><identifikator>(<parametri>);`
- Tip i parametri definišu povratni tip i potpis za metode sa liste poziva
- Delegati se mogu definisati
  - kao tip najvišeg nivoa ili
  - kao ugnežđeni u klasu/strukturu

Primerci delegata

- Primerci delegata se prave navođenjem imena metoda koji odgovara deklaraciji
  - bilo koji (statički ili nestatički) metod koji odgovara deklaraciji može da se koristi
- Dodela jednog delegata drugom kopira listu poziva iz izvorišnog u odredišni
- Operatori + i += rezultuju u kombinovanim listama poziva
  - reference na metode seređaju redosledom dodavanja
  - dodavanje jednog metoda na listu dva puta rezultuje u dupliranim referencama na metod
- Operatori – i `-=` rezultuju u uklanjanju referenci na metode sa liste poziva delegata
  - u slučaju višestrukih referenci na metod – uklanja se poslednja referenca sa liste
  - pokušaj uklanjanja reference na metod koji nije na listi ne izaziva grešku
- Delegat se poziva po imenu sa parametrima kao da je metod koji odgovara deklaraciji
  - poziv izaziva izvršavanje svih metoda sa liste poziva redom, prosleđujući im parametre
  - ako je parametar objekat ili `ref` parametar, promene iz jednog metoda vidi naredni metod
  - povratna vrednost je ona koju vrati poslednji metod sa liste

Primer delegata

```
//neka su: int MetodX(int i, string s){} gde je X∈{1,2,3}
//MetodX je statički ili metod objekta klase

public delegate int MojDelegat(int i, string s);
...
MojDelegat d1 = new MojDelegat(Metod1);
MojDelegat d2 = new MojDelegat(Metod2);
MojDelegat d3 = d1+d2;          //lista: Metod1, Metod2
d3 += new MojDelegat(Metod3); //lista: Metod1, Metod2, Metod3
d3 -= d1;                       //lista: Metod2, Metod3
d3 -= new MojDelegat(Metod3); //lista: Metod2
int i = d3(5, "Proba");
```

Članovi

- Članovi su elementi programa koji se sadrže u:
  - prostorima imena, klasama, strukturama i interfejsima
- Dele se u tri kategorije:
  - funkcionalni (izvršni kod),
  - podaci (konstantni i promenljivi) i
  - tipovi (ugnežđene definicije)
- Funkcionalni članovi:
  - statički i konstruktori primeraka, destruktori, metodi, operatori, svojstva, indekseri, događaji
- Članovi podaci:
  - konstante, polja
- Tipovi:
  - klase, interfejsi, strukture, nabrajanja, delegati
- Statički članovi nisu pristupačni preko referenci na objekte, već samo preko odgovarajućih tipova

Prava pristupa članovima

- Modifikatori za određivanje prava pristupa članu:
  - `public` – javni pristup bez ograničenja
  - `protected` – zaštićeni pristup, samo članovi klase i članovi izvedenih klasa
  - `private` – privatni pristup, samo članovi klase
  - `internal` – interni pristup, samo elementi koji su u okviru istog sklopa
  - `protected internal` – zaštićen interni pristup, unija zaštićenog i internog pristupa
- Ne postoji pravo pristupa vezano za prostor imena
  - pandan paketskom pravu pristupa u Javi
- Podrazumevano pravo pristupa članova klasa i struktura je privatno
  - za strukture različito nego u C++ (u C++ je bilo javno)
- Dozvoljeni modifikatori za određivanje prava pristupa članu prostora imena:
  - `public`
  - `internal`
- Podrazumevano pravo pristupa članova prostora imena je interno (`internal`)

Nasleđivanje članova

- U Javi :
  - metode se implicitno definišu kao virtuelne
  - metod u izvedenoj klasi sa istim potpisom i povratnim tipom redefniše nasleđeni metod
  - poziv metoda objekta pokreće onaj koji odgovara “najbližoj redefiniciji” u hijerarhiji klasa
- Problemi pri izmeni bazne klase (pisanju nove verzije bazne klase):
  - ako se u baznoj klasi napiše metod koji potpuno odgovara nekom metodu izvedene klase
    - pozivaće se metod izvedene klase, iako programer to neće očekivati
  - ako se u baznoj klasi napiše metod sa istim potpisom, ali različitim povratnim tipom
    - izvedena klasa više neće moći da se prevede
  - opisani problemi su verovatni ako osnovnu i izvedenu klasu ne proizvodi ista firma
- C# prevazilazi probleme nasleđivanja
  - eksplicitnim redefinsanjem ili
  - sakrivanjem

Redefinisanje i sakrivanje nasleđa

- Redefinisanje ima istu prirodu kao u Javi, ali nije podrazumevano
- Član bazne klase koji treba da bude redefinisan, deklariše se kao virtuelni (`virtual`)
- Član izvedene klase koji definiše novu implementaciju nasleđenog člana
  - deklariše se kao redefinisan (`override`)
  - bez modifikatora `override`, član u izvedenoj klasi skriva član u baznoj (uz upozorenje)
  - ako se stavi modifikator `override` za član koji u baznoj nije `virtual`, dobija se greška
- Za eksplicitno sakrivanje člana bazne klase, u izvedenoj se koristi modifikator `new`
  - na taj način se izbegava upozorenje prevodioca
- Sakrivanje raskida polimorfno ponašanje virtuelnih članova
- Kombinacija `virtual` i `new`: nova početna tačka specijalizacije
  - polimorfno ponašanje sa članovima potomaka, ali ne i predaka
- Za dohvatanje nasleđa iz bazne klase:
  - ključna reč `base` (odgovara `super` u Javi)

Zapečaćeni članovi

- U Javi se koristi modifikator `final`
  da označi da se metod ne može redefinisati
- U C# se koristi kombinacija modifikatora `override` i `sealed`
  - primenljivo samo na nasleđene virtuelne članove
  - razlika u odnosu na Javu – na C# se ne može koristiti odmah u osnovnoj klasi
  - nije ni potrebno, jer metodi u baznoj klasi podrazumevano nisu virtuelni
- Ako se klasa označi kao `sealed`, iz nje se ne može izvoditi

Svojstva (1)

- Svojstvo (*property*) omogućava direktan pristup stanju objekta:
  - notacijski kao da se pristupa polju
  - pristup se (u pozadini) ostvaruje preko metoda
- Definicija:
  `[<atributi>][<modifikatori>]<tip><identifikator>`
  `[{<pristupnici>}]`
- Tip može biti bilo koji vrednosni ili ukazani tip:
  - specificira tip podataka koji se dodeljuje svojstvu, odnosno koje svojstvo vraća
- Pristupnici (*accessors*) - pristupne definicije:
  - definišu `set` i `get` funkcije za pristup (promenu/čitanje) stanju svojstva
  - mogu sadržati proizvoljan kod
- Automatski definisana promenljiva `value` sadrži vrednost koja se dodeljuje svojstvu

Svojstva (2)

- Primer:

```
public class Osoba{
   private int godine;
   public int Starost{get {return godine;} set {godine=value;}}
}
Osoba o=new Osoba();
o.Starost=21; int g=o.Starost;
o.Starost++; o.Starost+=5;
```

- Ako svojstvo ima samo `get` ili `set`, reč je o svojstvu koje se samo čita ili samo menja
- Ako se želi upotreba operatora `++`, `--`, ili kombinovanih dodela (npr. `+=`)
  - potrebno je realizovati oba pristupnika (`set` i `get`)
- Interfejsi mogu da sadrže deklaraciju svojstva, bez tela pristupnika `get` i `set`
- Primer:

```
public interface Iosoba{
    int Starost{get;set;}
    string MaticniBroj{get;}
}
```

Indekseri (1)

- Indekseri omogućavaju korišćenje sintakse indeksiranja za klase/strukture koje ih sadrže
- Pogodno je za klase koje sadrže kolekcije podataka, za pristup elementima kolekcije
- Slični svojstvima
  - u smislu da pružaju način pristupa stanju objekta preko funkcija
- Nije potreban identifikator indeksera (kao što je potreban za svojstvo)
  - jer se elementu kolekcije pristupa preko imena objekta kolekcije
  - u definiciji se koristi ključna reč `this` umesto identifikatora indeksera
- Definicija:

```
[<atributi>][<modifikatori>]<tip> this [<parametri>]
{<pristupnici>}
```

Indekseri (2)

- Mora postojati bar jedan parametar (indeks), moguće više
- Indekseri sa više parametara odgovaraju multidimenzionalnim nizovima
- Tip parametra može biti proizvoljan vrednosni ili ukazani tip
- Primer:

```
public string this
[int indeks1, byte indeks 2, string indeks3]{...}
```

- Moguće je definisati i više indeksera u klasi/strukturi
- Da bi se razlikovali moraju imati različite tipove parametara
- Kao i svojstva, indekseri mogu da imaju samo `set` ili `get` pristupnike
  - moraju imati oba pristupnika ako se žele koristiti `++, --, +=,...`
- Mogu biti članovi interfejsa (`set` i `get` bez tela)
  - tada se mogu koristiti samo preko referenci na interfejse

Primeri indeksera

```
public class Naj10{
  private string[] najbolji=new string[10];
  public string this[int i]{
    get{ if(i>0&&i<11){ return najbolji[i-1];} else {return null;}}
    set{ if(i>0&&i<11){ najbolji[i-1]=value; }}
  }
}
public static void Main(){
  Naj10 studenti=new Naj10();
  studenti[1]= "Pera"; studenti[2]= "Mika"; studenti[3]= "Laza";...
}

public interface I{string this [int i]{get;set;}}
public class C:I{string I.this [int i]{get{...} set{...}}}
C a=new C(); string s1=a[1];      // pogresno
C b=new C(); string s2=((I)b)[1]; // ispravno
I i=new C(); string s3=i[3];      // ispravno
```

Događaji

- Formalizacija opšteg mehanizma za obaveštavanje o događajima
  - skup registrovanih slušalaca se obaveštava kada se desi događaj
- Mehanizam koristi delegate za sistem obaveštavanja
  - slušaoci registruju odgovarajućeg delegata kod izvora događaja
  - izvor događaja izvršava metode svih registrovanih delegata kada se događaj desi
- Definicija:
  `[<atributi>][<modifikatori>]event<tip><identifikator>`
  `[{<pristupnici>}]`
- Tip događaja (`<tip>`) je tip već definisanog delegata
- Pristupnici omogućavaju posebnu funkcionalnost dodavanja i uklanjanja slušalaca
  - ako nedostaju, prevodilac obezbeđuje podrazumevanu funkcionalnost
- Događaj se može izazvati jedino unutar tipa u kojem se definiše
  - čak i ako je `protected`, ne može se izazvati iz izvedene klase
- Događaj se izaziva pozivom po imenu sa stvarnim argumentima koji odgovaraju tipu
- Za registrovanje/deregistrovanje slušalaca se koriste operatori `+=` i `-=`

Primer obrade događaja (1)

- Obrada događaja promene temperature
  - više slušalaca se registruje kod više izvora promene temperature

```
using System;

public delegate void ObradaDogadjajaPromeneT(string izvor, int t);

public class IzvorDogadjajaT{
  private string ime; private int t=0;
  public event ObradaDogadjajaPromeneT dogadjajPromenaT;
  public IzvorDogadjajaT(string imeIzvora){ime=imeIzvora;}
  public void Promena(int novaT){
    t=novaT;
    if (dogadjajPromenaT != null) dogadjajPromenaT(ime,t);
  }
  public void otkaciSlusaoce(){dogadjajPromenaT=null;}
}
```

Primer obrade događaja (2)

```
public class Slusalac{
  private string ime;
  public Slusalac(string imeSlusaoca, IzvorDogadjajaT[] izvori){
    ime=imeSlusaoca;
    foreach (IzvorDogadjajaT t in izvori)
      t.dogadjajPromenaT+=new ObradaDogadjajaPromeneT(this.TPromenjena);
  }                                                    //this se podrazumeva
  public void TPromenjena(string izvor, int t){
    Console.WriteLine(ime + ": t=" + t + "C na lokaciji " + izvor);
  }
  public static void Main(){
    IzvorDogadjajaT d = new IzvorDogadjajaT("dvoriste");
    IzvorDogadjajaT f = new IzvorDogadjajaT("frizider");
    new Slusalac("Slusalac 1", new IzvorDogadjajaT[]{d,f});
    new Slusalac("Slusalac 2", new IzvorDogadjajaT[]{d,f});
    d.Promena(35); f.Promena(10);
  }
}
```

Pristupnici događaja

- Funkcije koje se pozivaju kada se koriste `+=` i `-=` za dodavanje/uklanjanje slušalaca
- U većini slučajeva podrazumevana funkcionalnost je dovoljna
- Ključna reč `add` za definisanje funkcionalnosti dodavanja slušalaca
- Ključna reč `remove` za definisanje funkcionalnosti uklanjanja slušalaca
- Jedini parametar unutar blokova `add` i `remove`:
  - implicitni parametar `value`
  - sadrži referencu na primerak delegata koji se dodaje/uklanja
- Primer:

```
public event RutinaZaObradu Dogadjaj{
  add { /* funkcionalnost za dodavanje slusaoca */ }
  remove { /* funkcionalnost za uklanjanje slusaoca */}
}
```

Standardni delegat za obradu događaja

- .NET standardizuje potpis delegata koji se koriste za obradu događaja
- Konkretna implementacija u `System.EventHandler` sa potpisom:

```
public delegate void EventHandler(object izvor, EventArgs arg);
```

- Izvor je referenca na objekat koji izaziva događaj
- Klasa `EventArgs` nema specijalizovanu funkcionalnost, iz nje se izvode argumenti