Klasa, objekat

Posmatrajmo nastavni proces u školi. Recimo da u učionici 1 učenik IVa, Pera, radi zadatak iz matematike po zahtevu profesora Laze, a u učionici 2 profesor Mika predaje svim učenicima odeljenja IVb novu lekciju iz fizike. Sasvim je moguće da sutradan uloge profesora budu promenjene, pa da profesor Mika u IVa ispituje ili predaje novu lekciju iz fizike, a profesor Laza u IVb predaje novu lekciju iz matematike.

Već u ovom uprošćenom prikazu nastavnog procesa možemo uočiti osnovne **objekte** koji učestvuju u njemu (Pera, Mika, Laza, IVa, IVb, matematika, fizika, učionica 1…).

Lako se može uočiti da Mika i Laza koordiniraju nastavnim procesom, da upoznaju učenike sa novim gradivom, da vrše proveru znanja učenika, ocenjuju ga. Oni imaju i zajedničke **atribute** (ime, stručna sprema, predmet koji predaju, godine staža). Samim tim oni pripadaju jednoj od **klasa** učesnika u nastavnom procesu, klasi *Profesor*.

Na sličan način možemo uočiti i klase *Ucenik* (Pera), *Predmet* (matematika, fizika) , *Odeljenje* (IVa, IVb)…

Do klasa dolazimo polazeći od pojedinačnih objekata. Posmatranjem svih učenika uočavamo za sve njih zajednička svojstva. Na primer, za sve učenike je potrebno pratiti ime, prezime, datum rođenja, razred, odeljenje. Ta svojstva nazivamo **atributima.** Takođe svim učenicima možemo pridružiti iste „akcije“: učenik odgovara i dobija ocenu, učenik menja odeljenje ... Te akcije u okviru klase nazivamo **metodama** klase, i njima se opisuje funkcionalnost objekata te klase.

Klasa *Ucenik* je uopštenje (**apstrakcija**) pojedinačnih učenika. Svaki pojedinačni učenik je primerak (**instanca**) klase *Ucenik*-***objekat***, i opisan je skupom vrednosti atributa klase. Vrednost atributa ime za konkretnog učenika je na primer “Pera“, a vrednost atributa razred je 4. "Pera" i 4 su pojedinačni podaci.

Svakom aplikacijom obrađujemo podatke. Svi podaci čuvaju se u memoriji računara u obliku nizova 0 i 1, bez obzira da li podatak predstavlja broj, reč, sliku ili zvuk. Različite vrste podataka registruju se na različite načine, takođe nad različitim vrstama podataka su dozvoljene različite operacije. U zavisnosti od vrste podatka mora se odvojiti određeni prostor za njegovo registrovanje i obezbediti izvršavanje određenih akcija nad tim podatkom. Računaru moramo dati informaciju koja vrsta podatka je registrovana, da bi ga na pravi

način tumačio i obrađivao. Potrebne informacije saopštavamo računaru **tipom podataka**.

Tipom podatka definisan je

- **način registrovanja podatka u memoriji**
- **skup mogućih vrednosti tog podatka**
- **skup mogućih akcija nad podatkom.**

Možemo reći da vrednost atributa ime, razred je podatak na najnižem nivou apstrakcije i za njihovo definisanje koristimo primitivne tipove podataka. Pošto klasa opisuje skup objekata koji imaju zajedničke karakteristike i jednako se ponašaju (funkcionalnost), klasa je takođe tip podatka na višem nivou apstrakcije.

U analizi procesa za koji kreiramo program uočavamo objekte koji učestvuju u procesu. U nastavnom procesu možemo uočavati predmete, učenike, profesore, odeljenja... Neke od tih objekata opisujemo istim svojstvima i nad njima možemo izvoditi iste akcije. Na osnovu njih pravimo šablon za takve objekte. Taj šablon zovemo klasom. Svaka klasa ima atribute i metode.

Klasu definišemo navođenjem rezervisane reči *class* iza koje sledi identifikator klase (ime klase). Posle imena klase, ako formiramo klasu koja nasleđuje neku prethodno definisanu klasu navodimo ':' a zatim sledi ime klase iz koje je izvedena. Zatim u vitičastim zagradama {} definišemo članove klase (atribute i metode).

```
class <imeKlase> :<imePrethodnoDefinisaneKlase>
{
opis /definicija članova klase
}
```

Na primer, klasu učenika iz uvodnog primera možemo intuitivno definisati na sledeći način:

```
class Ucenik
{     String ime, prezime;
      DateTime datumRodjenja;
      int razred;
      uint SkolskaGodina;
      char odeljenje;
      int [] ocene;
      double prosek()
      {
      …
      }
     void uci(Lekcija X)
     {
      …
      }
}
```

ATRIBUTI

METODE

Atributima opisujemo određenu **osobinu objekta** (ime, prezime, datumRodjenja, razred, odeljenje). Najčešće, različiti objekti iste klase imaju različite vrednosti atributa. Često kažemo da vrednosti atributa definišu stanje objekta. Pri opisu atributa moramo navesti tip kome taj atribut pripada (celobrojni, realni, znakovni, Form, Button) i ime atributa. Pri tome navedeni tip mora biti prethodno definisan (ugrađen u sistem ili definisan od strane programera).

<TIP ATRIBUTA> <IME ATRIBUTA> - OPIS ATRIBUTA
int razred

int a,b; //celobrojni atributi a i b
Button dugme;// atribut dugme, objekat ugrađene klase Button

Školska godina je statički atribut, ista je za sve objekte te klase
Ako je atribut definisan kao static onda se atributu pristupa
<imeKlase><imeAtributa>
Ucenik.SkolskaGodina

Pristup atributima objekta u C# realizujemo navođenjem imena objekta, znaka tačka ('.'), pa imena atributa. Na primer ako je *X* objekat klase *Ucenik* atributu *razred* pristupamo ***X.razred*.**

IME OBJEKTA.IME ATRIBUTA –PRITUP ATRIBUTU

X.razred

Metodom opisujemo **ponašanje objekta** u određenoj situaciji i pod određenim uslovima (*uci*), ali i određujemo nove vrednosti na osnovu osobina koje objekat poseduje (*prosek*). Na taj način **opisujemo funkcionalnost objekta.**

Metod klase je imenovani blok naredbi koji se sastoji se iz zaglavlja metode i tela metode. U zaglavlju navodimo povratni tip (ako metod ne proizvodi vrednost koju vraća navodimo rezervisanu reč ***void***), zatim ime metode za kojim sledi u malim zagradama spisak parametara te metode. Za svaki parametar navodi se tip kome taj parametar pripada kao i ime parametra. Posle zaglavlja metode u vitičastim zagradama navodimo telo metode koje se sastoji iz odgovarajućih iskaza C#.

<povratni tip> <ime metode>(<lista parametara>)
{ <telo metode> }

Poziv metode objekta u C# realizujemo na sledeći način:

imeObjekta.imeMetode(lista stvarnih parametara)

Na primer ako je *x* objekat klase *Ucenik* metodu *prosek* pozivamo

x.prosek(),

a metodu *uci*

x.uci(L)

gde je *L* objekat klase *Lekcija*.

Enkapsulacija podrazumeva da korisnici nekog objekta nemogu menjati unutrašnja stanja i metode tog objekta. Jedan od osnovnih koncepata objektno orijentisanog programiranja jeste mogućnost objekta da kontroliše pristup svojim članovima. Za svaki član klase (atribute i metode) moramo definisati nivo pristupa (vidljivosti, dostupnosti, zaštite). Na taj način definišemo koliko je taj član klase otvoren prema „spoljnjem“ svetu.

U C# postoje više nivoa pristupa a mi ćemo, u početku, koristiti **privatni** (*private*) i **javni** (*public*) pristup.

Neki članovi klase mogu da se koriste interno, samo u klasi, i van nje im se ne može pristupiti. Za takve članove klase kažemo da su privatni i pri njihovom definisanju navodimo rezervisanu reč *private*. **Privatnim** članovima klase mogu pristupati samo metode te klase i to je najveći nivo zaštite.

Pri opisu **javnih** članove klase navodimo rezervisanu reč *public*. Takvi članovi klase su dostupni svetu van klase i pristup tim članovima je potpuno slobodan.

Podrazumevani (*default*) pristup članovima klase je privatni, što znači da su članovi klase za koje nije naveden nivo pristupa, privatni članovi. Uobičajeno je da su atributi privatni, a metode javni članovi klase.

Kada se u toku izvršavanja aplikacije javi potreba za uvođenjem konkretnog objekta određene klase prostor za njegovo registrovanje se odvaja u memoriji računara van aplikacije (*heap - dinamičkoj memoriji*), a aplikacija sa njim komunicira preko njegove adrese (*reference*) pa zbog toga kažemo da je klasa **referentni** tip podataka.

Strukture

Obzirom da je **objekat primerak klase a klasa u sebi sadrži atribute koji mogu biti objekti neke druge klase jasno je da „ispod“ objekta moraju postojati neki osnovni (bazični) podaci na koje se oslanja objekat a samim tim i aplikacija.**

U C# osnovni podaci su predstavljeni **strukturama** (*struct*). Strukture su postojale još u programskom jeziku C, ali tada su služile isključivo za objedinjavanje podataka koji su opisivali karakteristike nekog objekta, ali nisu sadržale funkcionalnosti tog objekta. Nastankom objektno orijentisanog programiranja strukture, poput klasa, sadrže funkcionalnosti. U C# strukture se definišu na sledeći način:

```
struct imeStrukture
{
 definicija atributa i metoda strukture
}
```

Članovima strukture kao i kod klase možemo definisati nivo pristupa. Strukturu, za razliku od klasa, ne možemo izvesti iz druge strukture, niti se iz nje može izvesti druga struktura ili klasa.

Pri kreiranju određenog strukturnog podatka prostor za njegovo registrovanje odvaja se u memoriji računara u prostoru aplikacije koji je rezervisan za registrovanje podataka pa se njegovoj vrednosti pristupa direktno. Zato kažemo da je struktura **vrednosni** tip podataka. Podsetimo se da instancama klasa pristupamo preko adrese, pa klase zovemo **referentnim** tipom podataka.

C# kao i ostali programski jezici sadrži ugrađene (sistemske) biblioteke klasa i struktura. U okviru tih biblioteka definisani su, kao strukture, i osnovni tipovi podataka celi brojevi, brojevi u pokretnom zarezu, znakovni i logički tip. Obzirom da su realizovani preko struktura svi osnovni tipovi su vrednosni.

Osnovni tipovi podataka
Celobrojni tipovi

Tip	dozvoljene vrednosti	veličina u bajtovima
Sbyte	od -128 do 127 $(-2^7, 2^7-1)$	1 (8-bita)
byte	od 0 do 255 $(0, 2^8-1)$	1 (8-bita)
short	od -32,768 do 32,767 $(-2^{15}, 2^{15}-1)$	2 (16-bita)
ushort	od 0 do 65,535 $(0, 2^{16}-1)$	2 (16-bita)
int	od -2,147,483,648 do 2,147,483,647 $(-2^{31}, 2^{31}-1)$	4 (32-bita)
uint	od 0 do 4,294,967,295 $(0, 2^{32}-1)$	4 (32-bita)
long	od -9,223,372,036,854,775,808 do 9,223,372,036,854,775,807 $(-2^{63}, 2^{63}-1)$	8 (64 bita)
ulong	Od 0 do 18,446,744,073,709,551,615 $(0, 2^{64}-1)$	8 (64-bita)

tip	dozvoljene vrednosti	Veličina u bajtima

Znak **'u'** ispred naziva nekih od tipova ukazuje da su dozvoljene vrednosti za taj tip neoznačeni (unsigned) celi brojevi, tj. da taj tip **ne sadrži negativne cele brojeve.**

Osnovni celobrojni tip podataka *int* realizovan je strukturom *System.Int32*. Ta struktura sadrži *static* atribut *MinValue* i *MaxValue* koje sadrži najmanju i najveću moguću vrednost. Ostali celobrojni tipovi su takođe realizovani odgovarajućim strukturama.

Celi brojevi se pišu na uobičajen način. Ako želimo da naglasimo da je neki ceo broj tipa long dodamo mu sufiks l ili L (100L, 100l), za tip uint dodajemo sufiks u ili U (2340U) , za ulong sufikse u(U) i l (L) (23456lU).

Realni tipovi

Brojevi sa pokretnim zarezom, odnosno realni brojevi u C# se predstavljaju preko tri tipa podataka, float, double i decimal. Oni se razlikuju po skupu dozvoljenih vrednosti kao i po broju značajnih cifara (preciznosti). Osnovne karakteristike tipova podataka za brojeve u pokretnom zarezu date su sledećom tabelom.

tip	dozvoljene vrednosti	preciznost	Veličina u bajtima
float	od $\pm 1.5 \times 10^{-45}$ do $\pm 3.4 \times 10^{38}$	7 cifara	4 (32 bita)
double	od $\pm 5.0 \times 10^{-324}$ do $\pm 1.7 \times 10^{308}$	15-16 cifara	8 (64 bita)
decimal	od $\pm 1.0 \times 10^{-28}$ do $\pm 7.9 \times 10^{28}$	28-29 cifara	16 (128bita)

Analiziranjem prethodne tabele možemo primetiti da tip decimal ima veću preciznost, pa se stoga često primenjuje u finansijskom računanju, tj. pri radu sa novčanim vrednostima.

Broj zapisan u pokretnom zarezu se tumači kao podatak tipa double. Ako želimo da naglasimo da je broj u pokretnom zarezu float podatak dodajemo sufiks f ili F (1.65f), za double tip sufiks d ili D, a za tip decimal sufiks m ili M (2.123m).

Znakovni i logički tip

char	pojedinačni Unicode znak ceo broj od 0 do 65535	2 (16 bita)
bool	false, true	1
tip	dozvoljene vrednosti	Veličina u bajtima
char	pojedinačni Unicode znak ceo broj od 0 do 65535	2 (16 bita)
bool	false, true	1

Konstante char tipa pišu se navođenjem znaka između apostrofa ('A', 's', '3',...) ili kao escape sekvence kao što je navedeno u tabeli.

\'	jednostruki navodnik
\"	dvostruki navodnik
\\	obrnuta kosa crta
\a	zvučni signal
\n	novi red
\r	povratak na početak reda
\b	Backspace
\t	horizontalni tab

Kao što smo ranije napomenuli osnovni tipovi podataka realizovani su sistemskim strukturama što je navedeno u tabeli:

tip	sinonimna struktura
sbyte	*System.Sbyte*
byte	*System.Byte*
short	*System.Int16*
ushort	*System.UInt16*
int	*System.Int32*
uint	*System.UInt32*
long	*System.Int64*
ulong	*System.UInt64*
float	*System.Single*
double	*System.Double*
decimal	*System.Decimal*
char	*System.Char*
Bool	*System.Bool*

Svaka od navedenih struktura ima polja *MinValue* i *MaxValue* u kojima su redom smeštene minimalne i maksimalne vrednosti za svaki tip. Takođe, one sadrže i razne metode (za konverziju i slično... kasnije opširnije o tome).

U aplikaciji je svejedno da li koristimo *int* ili *System.Int32*, za prevodilac to su sinonimi.

String

Vrlo često u osnovne tipove C# ubraja se i string koji zapravo predstavlja niz karaktera, i za razliku od ostalih navedenih osnovnih tipova realizovan

je referentnim tipom. Za broj znakova koji čine string ne postoji gornja granica, pa je količina memorije koju zauzima jedan string promenljiva.

Konstante string tipa se navode između znakova " " (navodnici). Na primer, stringovi su:

"Ovo je string", "" (ovo je prazan string), "Beograd".

Tip string je realizovan sistemskom klasom *System.String*.

Deklaracija i definicija promenljive

U centru pažnje objektno orijentisane aplikacije su objekti koji *nastaju*, *egzistiraju*, međusobno *komuniciraju* i *nestaju* tokom aplikacije. Svaki objekat je instanca neke klase, ugrađene ili prethodno definisane od strane programera. Svaki objekat moramo imenovati da bi aplikacija mogla komunicirati sa njim. Takođe da bi mogli da koristimo njegove metode moramo znati kojoj klasi pripada. Na taj način definišemo način registrovanja objekta u memoriji, skup mogućih vrednosti tog objekta i skup mogućih operacija nad objektom.

Promenljiva je ime (identifikator) memorijske lokacije u kojoj aplikacija čuva vrednost.

Imena (identifikatori) koja programer definiše grade se na osnovu sledećeg pravila:

ime je niz slova (velikih i malih) engleske abecede, znaka za podvlačenje (_) i cifara. Ime mora da počne slovom ili znakom za podvlačenje.

Prema tome, imena: nazivPredmeta, ucenik1, Rezultat, radni_dan, _x su ispravna dok imena: naziv Predmeta, godisnje-doba, 1dan nisu ispravna.

Važno je napomenuti da C# razlikuje velika i mala slova tako da nisu ista sledeća dva imena, GodisnjeDoba i godisnjeDoba.

U C#, promenljive su podeljene u dve kategorije, **vrednosne** i **referentne** promenljive. Memorijska lokacija imenovana promenljivom vrednosnog tipa direktno sadrži vrednost, dok lokacija imenovana promenljivom referentnog tipa sadrži adresu lokacije u dinamičkoj memoriji gde se čuva informacija. Za podatke osnovnih tipova (tj. sistemskih struktura) u C# (int, double, char, bool,…), samo ime promenljive predstavlja njihovu vrednost, a za objekte sistemskih i od strane programera definisanih klasa, ime promenljive predstavlja adresu lokacije u dinamičkoj memoriji na kojoj su ti objekti kreirani i gde postoje, pa se njima pristupa indirektno.

Referentnim promenljivim ćemo dodeljivati objekte (primerke klase) ili nizove. Referentna promenljiva čuva memorijsku adresu objekta, tj. referencira na pravu vrednost.

Da bi smo koristili podatke u aplikaciji neophodno ih je imenovati i naznačiti kom tipu pripadaju. To postižemo deklaracijom promenljivih na sledeći način:

<ime tipa> <ime promenljive>, …;

Deklaracijom promenljive vrednosnog tipa u memoriji se odvaja prostor neophodan za registrovanje podatka zadatog tipa, a pri deklaraciji promenljive referentnog tipa odvaja se prostor za registrovanje adrese memorijske lokacije u kojoj se nalazi (ili će se nalaziti) objekat.

```
int x,y;
char p,q;
Ucenik ucenik1,ucenik2;
```

Samom deklaracijom ne dodeljuje se vrednost promenljivoj vrednosnog tipa. Ako želimo da dodelimo vrednost promenljivoj koristimo operator = (operator dodele) na sledeći način:

<promenljiva>=<izraz>

gde se tip vrednosti izraza poklapa sa tipom promenljive (ili se implicitnom konverzijom može konvertovati u tip promenljive)[1].

```
int x,y;
x=5;
y=-145+x;
char p;
p='A';
```

Deklaracijom promenljive referentnog tipa ne zauzima se memorija za objekat već samo za referencu na njega (tj. za adresu objekta). Da bi napravili objekat upotrebljavamo izraz oblika

```
new <ime klase>(<lista parametara>)
mojaKlasa x;
x= new mojaKlasa();
Ucenik  a;
a=new Ucenik("Pera", "Peric", 15);
```

Korišćenjem operatora *new* odvaja se u dinamičkoj memoriji dovoljno prostora za objekte klase *mojaKlasa* i *Ucenik*. Operator *new* vraća adresu dodeljenog prostora koju onda pridružujemo promenljivoj referentnog tipa *x*, odnosno *a*. Metode *mojaKlasa()* i *Ucenik("Pera", "Peric", 15)* kreiraju objekte odgovarajuće klase u prostoru odvojenom korišćenjem operatora new.

Metode koje kreiraju objekte zovemo **konstruktorima.** Konstruktori su sastavni deo svake klase i nose njeno ime. Pozivom

[1] Detaljnije o izrazima i konverzijama podataka ??? (naziv poglavlja)

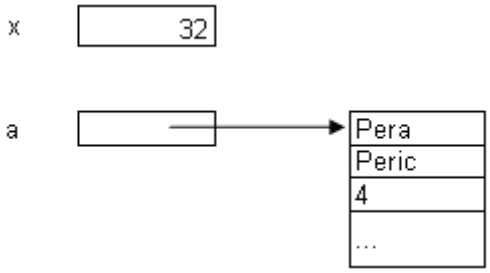
konstruktora objekat počinje svoj život. Klasa može imati jedan ili više konstruktora, koji se razlikuju po listi parametara. Lista parametara najčešće sadrži vrednosti kojima inicijalizujemo svojstva objekta, a može biti i prazna.

Možemo zaključiti da se definicija promenljive odvija u dva koraka.

- **deklaracija promenljive**
  - **promenljiva vrednosnog tipa se pri deklaraciji ne inicijalizuje ukoliko to nije eksplicitno navedeno**
  - **promenljiva referentnog tipa se pri deklaraciji inicijalizuje nulom**
- **inicijalizacija promenljive**
  - **eksplicitno navodimo vrednost promenljive**
  - **operator new odvaja u dinamičkoj memoriji dovoljno prostora za objekat odgovarajuće klase a konstruktor ga kreira u tom prostoru**

U C# je data mogućnost objedinjavanja ove dva koraka, što vrlo često koristimo.

```
int x=32, y;
Ucenik  a=new Ucenik("Pera", "Peric", 15);
Form1 F= new Form1();
Pen olovka=new Pen(Color.Blue);
```



Koristeći prethodno navedene postupke, deklarišemo i definišemo atribute klase, ali i pomoćne (lokalne) promenljive koje su neophodne za uspešnu realizaciju određenih metoda. Svi atributi konkretnog objekta dostupni su i vidljivi svim metodama tog objekta i traju dok traje objekat. Za razliku od njih, promenljive deklarisane u okviru neke od metoda, nastaju izvršavanjem te metode, vidljive su samo u okviru nje, i gase se završetkom te metode. Zato, ako je potrebno da koristite vrednost u okviru više metoda jedne klase (ili u više izvršavanja jedne metode), neophodno je da je definišete kao atribut klase.

Promenljiva vrednosnog tipa direktno sadrži podatak, dok promenljiva referentnog tipa sadrži referencu na objekat koji je smešten u posebnom memorijskom prostoru. Svaka promenljiva vrednosnog tipa sadrži sopstvenu kopiju podatka, pa operacije nad jednom promenljivom nemaju efekta na drugu promenljivu. Dve referentne promenljive mogu sadržati adresu istog objekta, pa operacije nad jednom referentnom promenljivom mogu imati efekat na objekat na koji referiše druga promenljiva.