4. Datu tipi un mainīgie

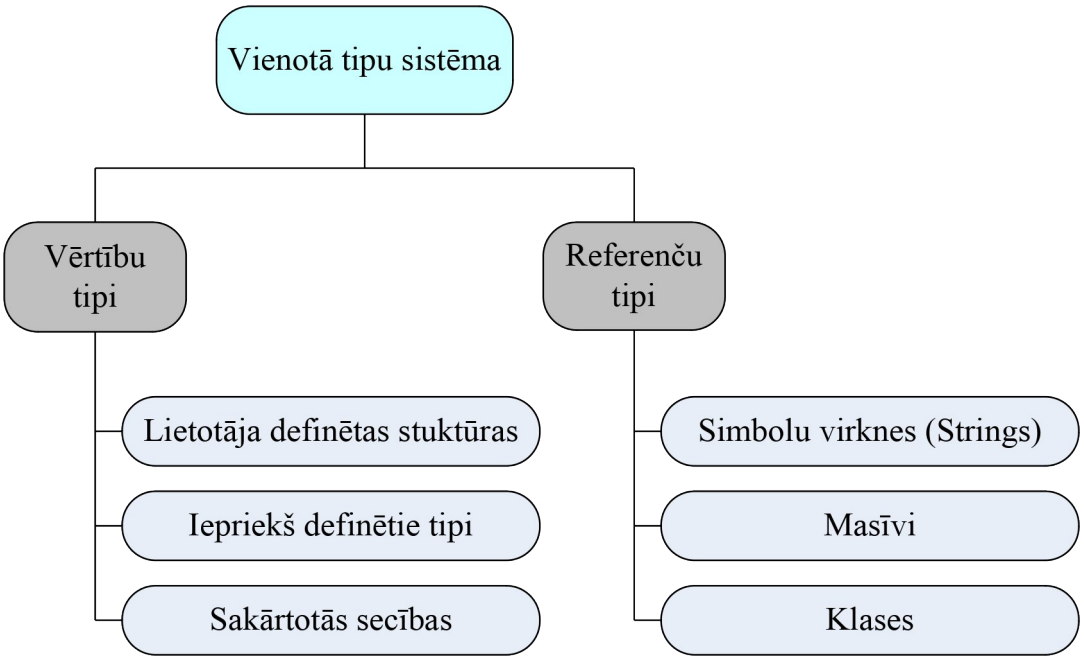
Šajā nodaļā apskatīsim .NET Framework datu tipu sistēmu: vienotā tipu sistēmas koncepcija, vērtību un referenču tipa mainīgus, lietotāja definētus datu tipus.

4.1. Ievads .NET Framework tipu sistēmā

Datu tips nosaka programmēšanas elementa datu veidu un uzglabāšanas īpatnības. Definējot (izveidojot) mainīgo jebkurā aplikācijā tam jānorāda arī atbilstošai datu tips.

Visus .NET Framework datu tipus nosaka **vienotā tipu sistēma** (*common type system*). Šī sistēma satur vērtību un referenču tipu definīcijas. Vienotā tipu sistēma nodrošina starp valodu integrāciju. Piemēram, var izveidot aplikāciju, kura uzrakstīta C# programmēšanas valodā, izmanto komponenti, kas izveidota Visual Basic un izmanto metodes Java kodā.

Visi datu tipi tiek sadalīti divās lielās grupās: vērtību un referenču tipi (4.1.attēls).



4.1.att. .NET Framework vienotā tipu sistēma

Vērtību tipiem ir raksturīgas šādas īpašības:

- tieša datu uzglabāšana;
- jābūt obligāti definētiem;
- nevar būt tukšas (jābūt norādītai kādai sākotnējai vērtībai).

Piešķirot viena vērtību tipa mainīgā vērtību citam vērtību tipa mainīgajam notiek vērtības kopēšana.

Katra mainīgā vērtību mainīšana vērtību tipiem neietekmē citu vērtību, savukārt viena referenču tipa mainīgā vērtību citam mainīgam, notiek tikai atsauces kopēšana. Šajā gadījumā viena mainīgā vērtības izmaiņas var ietekmēt citu mainīgo. Referenču tipa mainīgais var būt tukšs (*null*), kas visplašāk tiek izmantots datu bāzēs. Tālāk grāmatā referenču datu tipi tiks apskatīti sīkāk.

Iepriekš definētie vērtību datu tipi vienotā tipu sistēmā tiek iedalīti trijās grupas: skaitliskie (dažādu skaitļu reprezentācijai), simboliskie (dažādu simbolu uzglabāšanai) un specializētie (loģiskās vērtības, datumi utt.). Tabula 1 parāda biežāk lietojamus datu tipus.

Tabula 1

Biežāk lietojamie datu tipi

Nosaukums	Apraksts	Izmērs (bitos)	Vērtību intervāls
`int`	Veselie skaitļi	4	-2 147 483 648 līdz 2 147 483 648
`byte`	Bits	1	0 līdz 255
`long`	Lielākie veselie skaitļi	8	-9 223 372 036 854 775 808 līdz 9 223 372 036 854 775 808
`float`	Skaitļi ar peldošo punktu (daļskaitļi)	4	$1.5 * 10^{-45}$ līdz $3.4 * 10^{38}$ ar 7 zīmju precizitāti
`double`	Daļskaitļi ar lielāku precizitāti	8	$5*10^{-324}$ līdz $1.7*10^{308}$ ar 15 zīmju precizitāti
`decimal`	Naudas vienības	16	Skaitļi līdz 28 zīmēm
`char`	Viens simbols	2	-
`bool`	Loģiskā vērtība	1	true vai false
`DateTime`	Laika moments	8	-
`string`	Simbolu virknes	2 vienam simbolam	-
`object`	Vispārīgai objekts	4 (32-bitu) vai 8 (64-bitu)	-

Pastāv šādi ieteikumi datu tipu izmantošanai:
skaitliskiem tipiem:
- cikla skaitītājiem vai vienkāršiem veseliem skaitļiem izmantojiet **`int`** datu tipu;
- skaitļiem ar zīmēm aiz komata izmantojiet **`float`**, vai nepieciešamības gadījumā **`double`**;
- naudas vienību uzglabāšanai izmantojiet **`decimal`** datu tipu.

simboliskie tipi:
- viena simbola uzglabāšanai izmantojiet **`char`** datu tipu;
- vienas simbolu virknes uzglabāšanai izmantojiet **`string`** datu tipu;
- simbolu virkņu izveidošanai un manipulācijā ar tiem izmantojiet **`StringBuilder`** klasi;

specializētus tipus izmantojiet datuma (**`DateTime`**) un loģisko vērtību uzglabāšanai (**`bool`**).

4.2.Mainīgo definēšana un izmantošana

Mainīgai (*variable*) uzglabā aplikācijas darbībai nepieciešamas vērtības datora īslaicīgajā atmiņā. Aplikācijas izmanto šīs vērtības dažādiem aprēķiniem, datu analīzei utt.

Konstantes ir mainīgie ar fiksēto vērtību, kura nemainās programmas darbības laikā. Konstantu izmantošana padara kodu par vieglāk lasāmo, kā arī aizņem mazāk vietas atmiņā, jo ir stingri kodētās MSIL valodā.

Mainīgajam ir šādi to raksturojošie elementi:
- **nosaukums** (*name*) – unikālais identifikators mainīgā izmantošanai programmas kodā;
- **adrese** (*address*) – mainīgā uzglabāšanas vieta atmiņā;
- **datu tips** (*data type*) – mainīga veids un izmērs;
- **vērtība** (*value*) – mainīgā uzglabātā vērtība;
- **darbības apgabals** (*scope*) – nosaka programmas koda apgabalu, kurā var piekļūt un izmantot mainīgo;
- **dzīves cikls** (*lifetime*) – laika periods, kad mainīgais ir spēkā un ir pieejams izmantošanai.

Pirms mainīgā izmantošanas to ir nepieciešams definēt, norādot tās nosaukumu un pārējus parametrus. Vērtību piešķiršana notiek ar **=** simbola palīdzību:

Var definēt katru mainīgo atsevišķi, kā arī vienāda tipa mainīgus kopā:

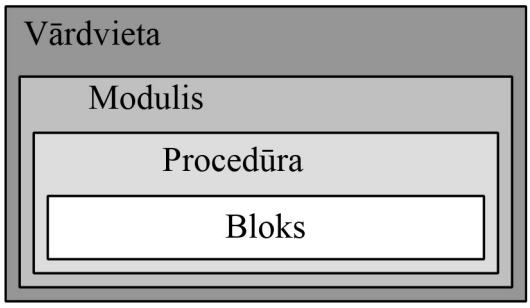
```
int k, c;
c = 2;
char u = 'a';
double b = 3.75;
```

Konstanšu deklarēšanai izmanto atslēgas vārdu **const**.

Nākošajā piemērā tiek definēta konstante PI un tālāk to izmanto apļa ar rādiusu 5 laukuma aprēķināšanai:

```
const double PI = 3.14159;
int radius = 5;
double laukums = PI * radius * radius;
MessageBox.Show("Laukuma radiuss ir " + laukums.ToString());
```

Definējot mainīgo jāpārliecinās, ka to varēs izmantot visi koda apgabali, kuriem tas ir atļauts, kā arī jāierobežo pieeja šiem mainīgajiem no citas aplikācijas vietas drošības nolūkos. Ja programma cenšas izmantot mainīgo ārpus tā darbības apgabala, kompilators izdod kļūdas paziņojumu. Darbības apgabalu līmeņi ir parādīti 4.2.attēlā.



4.2.att. Mainīgo darbības apgabalu līmeņi

Bloks – koda apgabals, kas sākas ar inicializācijas un beidzās ar pabeigšanas priekšrakstiem, mainīgais ir pieejams tikai šajā blokā;
Procedūra – mainīgais ir pieejams metodes vai īpašības iekšienē;
Modulis – mainīgais pieejams modulī, klasē vai struktūrā, kurā tās tika definēts.
Vārdvieta – mainīgais ir pieejams visas vārdvietas ietvaros.

Šaurākais apgabals ir bloks, bet plašākais ir vārdvieta.

Mainīgā darbības apgabalu var definēt arī ar speciālo atslēgas vārdu palīdzību:

public	Neierobežota pieeja, mainīgo var izmantot jebkura klase.
private	Pieeja ir ierobežota ar mainīgā saturošu koda apgabalu; mainīgo var izmantot tika klase, kur šīs mainīgais ir definēts.
internal	Pieeja ir atļauta tikai asamblejai; mainīgo var izmantot tikai konkrētās asamblejas klases.
protected	Mainīgo var izmantot tikai tā saturoša klase un klases, kas izveidotas uz dotās klases bāzes.
protected internal	Pieeja ir atļauta mainīgo saturošai klasei, klasēm, kas izveidotas uz dotās klases bāzes un dotās asamblejas klasēm, kuru iekšā ir definēta klase ar mainīgo.

Apskatīsim mainīgā definēšanas piemērus dažādos darbības apgabalos.

Mainīgā definēšana blokā (šajā gadījumā metodes iekšienē). Mainīgā „laukums" darbības apgabals ir ierobežots ar metodes „KvadrPlatiba" apgabalu, kas ierobežots ar figūriekavām; ārpus šīs metodes mainīgā izmantošana nav iespējama:

```
public int KvadrPlatiba(int garums)
{
    int laukums = 0;
    laukums = garums * garums;
    return laukums;
}
```

Nākošais piemērs parāda moduļa līmeņa mainīgo „pazinojums" izmantošanu, pie kura var griezties no jebkuras šī moduļa vietas:

```
private string pazinojums;

void NoteiktPazinojumu()
{
    pazinojums = "Sveiks!";
}

void ParaditPazinojumu()
{
    MessageBox.Show(pazinojums);
}
```

Sakārtotās secības tipus izmanto gadījumos, kad mainīgais var pieņemt vairākas iepriekš definētas vērtības. Tad ar sakārtotu secību var šo vērtību vienkāršo skaitļu vietā izmantot nosaukumus, kuri atspoguļo vērtības jēgu. Rezultātā kods ir labāk saprotams, vieglāk nepieciešamības gadījumā veikt izmaiņas (pievienot vai noņemt kādu vērtību), kā arī IntelliSense iespēja piedāvās izvēlēties no saraksta saprotamus nosaukumus:

```
enum Gadalaiki :
     int { Pavasaris = 1, Vasara = 2, Rudens = 3, Ziema = 4 };

private void button1_Click(object sender, EventArgs e)
{
     textBox1.Text = Gadalaiki.Pavasaris.ToString();
}
```

Sakārtoto secību izmantošanā jāņem vērā to, ka tās jādefinē moduļa līmenī, pretējā gadījumā lietotājs saņem paziņojumu par sintaktisko kļūdu.

4.3.Kolekciju definēšana un izmantošana

Kolekcija ir elementu grupa ar vienotu nosaukumu, kura tiek izveidota šo elementu kopīgās apstrādes nolūkā. Atšķirībā no masīviem kolekcijā var būt jebkura tipa elementi (objekti, struktūras, kā arī citas kolekcijas).

.NET Framework satur dažādas iepriekš definētas kolekcijas, kuras pieejamas **`System.Collections`** vārdvietā. Šeit ir definētas arī metodes kolekciju izveidošanai, meklēšanai vai šķirošanai kolekcijās un citas.

Masīvs ir kolekciju veids, kurš vienāda tipa elementus. Masīvu priekšrocība salīdzinājumā ar citiem kolekciju veidiem ir tās darbības ātrums.

Biežāk lietojamie kolekciju veidi ir:

`SortedList`	Kolekcija satur sašķirotus elementus ar saviem indeksiem un vērtībām. Šķirošana notiek izmantojot indeksus.
`ArrayList`	Satur elementus ar indeksiem. Ļauj patvaļīgi griezties pie jebkura elementa, kā arī mainīt savu izmēru, pievienojot vai nodzēšot elementus.
`BitArray`	Paredzēta loģisko vērtību uzglabāšanai.
`HashTable`	Elementi tiek organizēti pēc indeksa jaukšanas (*hash*) koda. Šīs kods paātrina elementu meklēšanu kolekcijā (šādu principu izmanto pārlūkprogrammas ievadīto tīmekļa lappušu adrešu apstrādei).
`Queue`	Reprezentē elementu sarakstu pēc FIFO (*first-in, first-out*) principa (pirmais iekšā pirmais ārā). Pievienojot jauno elementu kolekcijai, tas tiek ievietots kolekcijas beigās, nodzēšot kolekcijas pirmo elementu (piemēram, informācijas ierakstīšana atmiņas buferī).
`Stack`	Reprezentē elementu sarakstu pēc LIFO (*last-in, first-out*) principa (pēdējais iekšā, pirmais ārā). Pievienojot elementu, tās tiek izvietots steka pirmajā vietā, pievienojot nākošo elementu, dotais tiek nodzēsts. Var izmantot aprēķinu starp rezultātu vērtību uzglabāšanai.

Masīvi ir referenču tipa mainīgie, masīva elementi atsaucās uz masīva eksemplāru un netiek saglabāti stekā, kā vērtību mainīgie. Definējot masīvi nav obligāti norādīt tas elementu skaitu (masīva izmēru), jo to var izdarīt arī aplikācijas izpildes laikā.

Masīva definēšana notiek šādi (pirmais ir viendimensiju, otrais ir divu dimensiju masīvs):

```
int[] masivs1D = new int[5];
double [][] masivs2D=new double [2,3];
```

Masīva elementu izmantošanai jānorāda masīva nosaukums un elementa indekss (numerācija masīvos sākas no nulles):

```
masivs1D[0] = 10;
masivs2D[1,2] = 3.45;
```

Salīdzinājumā ar masīviem, citi kolekciju veidi tiek apstrādāti lēnāk, tomēr tiem ir daudz plašāks iepriekš definēto metožu un īpašību skaits. Apskatīsim kolekcijas izveidošanu uz ArrayList kolekcijas piemēra:

```
//izveidojam kolekciju
ArrayList Kolekcija = new ArrayList();
//pievienojam kolekcijai elementus
Kolekcija.Add("Sarkanais Ford");
Kolekcija.Add("Zalja Toyota");
//iesprausim elementu kolekcijas vidū
Kolekcija.Insert(1, "Melnais Lexus");
//nodzēsīsim sarkano fordu no kolekcijas
Kolekcija.Remove("Sarkanais Ford");
```

Atkarībā no kolekcijas veida, tās iepriekš definētas metodes būs atšķirīgas. Iepriekšējā piemērā tika apskatītas trīs **`ArrayList`** kolekcijas metodes **`Add`**, **`Insert`** un **`Remove`**. **`Stack`** kolekcijas gadījumā elementu pievieno ar metodi **`Pop`**, bet **`Queue`** kolekcijai elementu pievieno ar **`Enqueue`** metodes palīdzību.

4.4.Datu tipu konvertācija

Datu tipu konvertācija ir vērtību tipu nomaiņas process, piemēram, vesela skaitļa mainīgā konvertācija uz simbolu virkni. Pastāv **netiešā** (*implicit*) jeb automātiskā un **tiešā** (*explicit*) jeb manuālā konvertācijas.

Bieži vien aplikācijas laikā ir nepieciešams nodod vērtību no viena mainīgā citā, ja mainīgo tipi nesakrīt, ir nepieciešama tipu konvertācija.

Netiešā datu tipu konvertācija notiek automātiski, tā neprasa speciālo kodu. Visual C# programmēšanas valoda atbalsta tikai drošu tipu netiešo konvertāciju (bez informācijas iespējama zuduma), kad, piemēram, konvertē vērtību no **`float`** uz **`double`** datu tipu:

```
float x = 2.35;
double y = x;
```

Šāda konvertācija būs vienmēr veiksmīga un informācijas zudums nav iespējams. Automātiskā konvertācija ir spēkā pārveidojot **`int`** uz **`long`**, **`float`**, **`double`** un **`decimal`**; pārveidojot **`byte`** uz **`int`**, **`float`**, **`double`**, **`decimal`**; **`char`** uz **`int`**, **`float`**, **`double`**, **`decimal`** utt.

Tiešās konvertācijas gadījumos programmētājam jāraksta kods šī procesa nodrošināšanai. Šeit var būt dažādi varianti, piemēram, pārveidojot vērtības tipu

uz referenču un otrādi. Pirmo variantu sauc par **ievietošanu kastē** (*boxing*), bet otro par **izņemšanu no kastes** (*unboxing*):

```
//boxing
int i = 12;
object o = (object)i;

//unboxing
object o = 12;
int i = (int)o;
```

Piemērā redzams, ka datu tipu konvertācija notiek pirms mainīgā nosaukuma apaļās iekavās norādot tipu, uz kuru konvertē (*cast* priekšraksts). Veicot konvertāciju no šaurākā tipu uz plašāku (paplašināšana), datu precizitātes zuduma nav, savukārt, pretējais process (sašaurināšana) var novērst pie informācijas zuduma:

```
double y = 12.67;
int x = (int)y;
//x vērtība būs 12
```

Vēl viena datu tipu konvertācijas metode ir **`System.Convert`** klases izmantošana. Šī klase piedāvā dažādas metodes tipu pārveidošanai:

```
double y = 12.67;
string s = y.ToString();
```

Šajā piemērā mainīgajam **`s`** tiek piešķirta mainīgā y vērtība, izmantojot metodi **`ToString()`**.

Nodaļas kopsavilkums

Datu tips nosaka programmēšanas elementa datu veidu un uzglabāšanas īpatnības. Definējot (izveidojot) mainīgo jebkurā aplikācijā tam jānorāda arī atbilstošai datu tips.

Visi datu tipi tiek sadalīti divās lielās grupās: vērtību un referenču tipi. Vērtību tipiem ir raksturīgas šādas īpašības: tieša datu uzglabāšana; jābūt obligāti definētiem; nevar būt tukšas (jābūt norādītai kādai sākotnējai vērtībai).

Katra mainīgā vērtību mainīšana vērtību tipiem neietekmē citu vērtību, savukārt viena referenču tipa mainīgā vērtību citam mainīgam, notiek tikai atsauces kopēšana.

Mainīgai uzglabā aplikācijas darbībai nepieciešamas vērtības datora īslaicīgajā atmiņā. Aplikācijas izmanto šīs vērtības dažādiem aprēķiniem, datu analīzei utt. Konstantes ir mainīgie ar fiksēto vērtību, kura nemainās programmas darbības laikā. Mainīgo to raksturo: nosaukums, adrese, datu tips, vērtība, darbības apgabals un dzīves cikls.

Kolekcija ir elementu grupa ar vienotu nosaukumu, kura tiek izveidota šo elementu kopīgās apstrādes nolūkā. Atšķirībā no masīviem kolekcijā var būt jebkura tipa elementi (objekti, struktūras, kā arī citas kolekcijas). Biežāk lietojamās kolekcijas ir: ArrayList, Queue, Stack, BitArray, HashTable un SortedList.

Datu tipu konvertācija ir vērtību tipu nomaiņas process. Pastāv netiešā jeb automātiskā un tiešā jeb manuālā konvertācijas.

UZDEVUMI PAŠPĀRBAUDEI

1. Nosauciet .NET Framework biežāk lietojamus datu tipus un paskaidrojiet to izmantošanas veidus.
2. Kādi ir mainīgā darbības apgabala līmeņi un kā tos var mainīt?
3. Paskaidrojiet biežāk lietojamo kolekciju īpašības un pielietojuma situācijas.
4. Kāpēc ir nepieciešama datu tipu konvertācija; kādos gadījumos tā var novest pie informācijas zaudēšanas?

UZDEVUMI PATSTĀVĪGAJĀM DARBAM

1. Nosauciet un raksturojiet datu tipus, kuri netika nosaukti lekcijā.
2. Uzrakstiet programmu ar četriem mainīgajiem, kuru datu tipi ir int, float, double un string, ar šādām atbilstošām sākotnējām vērtībām: 10, 2.76, 34.23456 un „variable”. Definējiet visus iespējamus konvertācijas veidus starp tiem, izvadot paziņojumu logā oriģinālo un konvertēto vērtību.
3. Aprēķiniet cik daudz atmiņas ir nepieciešams otrajā punktā izveidotai programmai.
4. Uzrakstiet programmu, kurā tiek definēta sakārtotā secība (enum) ar piecām vērtībām: „Rezekne”, „Vilani”, „Malta”, „Ludza” un „Kaunata”, kā arī ArrayList tipa kolekcija, kura satur tos pašus nosaukumus. Definējiet StringBuilder objektu un ierakstiet tajā kolekcijas vērtības. Izvadiet paziņojumu logā StringBuilder saturu.