



MUHAMEDIYEVA D.T., VARLAMOVA L.P.,
BAXROMOV S.A.

“DASTURIY INJINIRING”
C#-ДАСТУРЛАШ АСОСЛАРИ

О‘QUV QO‘LLANMA



MUHAMEDIYEVA D.T., VARLAMOVA L.P.,
BAXROMOV S.A.

**O‘ZBEKISTON RESPUBLIKASI
OLIV VA O‘RTA MAXSUS TA‘LIM
VAZIRLIGI**

**MIRZO ULUG‘BEK NOMIDAGI O‘ZBEKISTON
MILLIY UNIVERSITETI**

**Muhamediyeva D.T., Varlamova L.P.,
Baxromov S.A.**

**“Dasturiy injiniring”
C#-дастурлаш асослари**

o‘quv qo‘llanma

Toshkent-2023

“Dasturiy injiniring” kursi uchun o‘quv qo‘llanma “Amaliy matematika va intellektual tizimlar” fakulteti talabalari uchun dasturlash tillaridan foydalanish, dasturiy ta‘minot ishlab chiqish va ma‘lumotlar bazasini boshqarish tizimlarini o‘rgatish maqsadida namunaviy va ishchi dastur asosida tuzilgan. . Ushbu kurs talabalarga uch semestr davomida o‘qitiladi. Birinchi semestr sinflarga kirishni boshqarish, rekursiya, inkapsulyatsiya va meros, C# dasturlash tili asosida interfeyslar va kolleksiyalarni yaratish masalalarini qamrab oladi.

The textbook for the course "Software Engineering" is compiled on the basis of the exemplary and working program for students of the faculty of "Applied Mathematics and Intelligent Systems" with the aim of teaching the use of programming languages, software development and database management systems. This course is taught to students for three semesters. The first semester covers issues of access control to classes, recursion, encapsulation and inheritance, creating interfaces and collections based on the C # programming language.

Mualliflar:

D.T.Muhamediyeva - Toshkent irrigatsiya va qishloq xo‘jaligini mexanizatsiyalash muhandislari instituti milliy tadqiqot universiteti professori, t.f.d.

L.P.Varlamova - Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti, Hisoblash matematikasi va axborot tizimlari kafedrasida professori, t.f.d.

S.A. Baxromov - Mirzo Ulug‘bek nomidagi O‘zbekiston Milliy universiteti, Hisoblash matematikasi va axborot tizimlari kafedrasida dotsenti, t.f.n.

Taqrizchilar:

Matyaqubov A.S – O‘zMU, Amaliy matematika va computer taxlili kafedrasida mudiri

Yakubov M.S. – TATU, Axborot texnologiyalari kafedrasida professori

O‘quv qo‘llanma O‘zbekiston Respublikasi Oliy va o‘rta maxsus ta‘lim vazirligining 2022 yil “09” sentyabrdagi “302” –sonli buyrug‘iga asosan nashrga tavsiya etilgan. Ro‘yxatga olish raqami 302-0383.

Mundarija

Kirish.....	4
1. Sinflarni boshqarish.....	5
2. Ref va out parametrlardan foydalanish, metoddan obe'ktni qaytarish, majburiy bo'lmagan argumentlar.....	21
3. Rekursiya.....	28
4. Static kalit so'zini ishlatish, statik klasslar.....	33
5. Indeksatorlar va xususiyatlar.....	38
6. Merosxurlik.....	52
7. Interfeyslar, strukturalar va ro'yxatlar.....	59
8. Kolleksiya va iteratorlar.....	71
Xulosa.....	85
Foydalanilgan adabiyotlar.....	85

Kirish

Dasturiy injenering fani zamonaviy axborot-kommunikatsiya texnologiyalarining eng muhim va asosiy tarmoqlaridan biri bo'lib, ushbu fan doirasida ko'plab model masalalar o'rganiladiki, bu mazkur fanni chuqur o'rgangan har bir bakalavr olgan bilim va ko'nikmalarini amaliy ish faoliyatida, ilmiy-tadqiqot ishlarida, shuningdek, talim tizimida samarali foydalanishi imkonini beradi. Ilm-fanning ushbu sohalaridagi zamonaviy hamda muhim masalalarini hal etishda keng o'rin tutadi.

“Dasturiy injiniring” predmetining asosiy maqsadi – talabalarda dasturiy injiniring sohasida bilim va ko'nikmalarni hosil qilib ularni amalda qo'llash namunalari bayon etishdan iborat.

Ushbu kurs dasturiy injiniring tadbiq qilish jarayoniga keng ma'noda kirish hisoblanadi.

“Dasturiy injiniring” o'quv predmeti 5330100 – Axborot tizimlarining matematik va dasturiy ta'minoti ta'lim yo'nalishining “Dasturlash tillari”, “Matematika”, “Berilganlar tuzilmalari” fanlari bilan uzviy bog'liq va ushbu fanlarni ayrim boblarini bilish zarur. “Dasturiy injiniring” o'quv predmetidan olingan bilimlar “Kompyuter tizimlari xavfsizligi”, “Berilganlar bazalari va axborot tizimlari”, “Berilganlar bazalari tizimlarida zamonaviy texnologiyalar”, “Intellectual tizimlar”, “Boshkaruv axborot tizimlari”, “Axborot tizimi auditori”, “Biznes-dasturiy ilovalari uchun dasturiy ta'minotni ishlab chikish”, “Mobil dasturiy ilovalarini ishlab chikish” fanlarni o'zlashtirishda zarur bo'ladi.

Dasturiy injiniring har xil predmet sohaları dasturlash loyihalarida samarali ishlatiladi.

Mazkur dasturga ko'ra ushbu fan doirasida ko'plab model masalalar o'rganiladiki, bu mazkur fanni chuqur o'rgangan har bir bakalavr olgan bilim va ko'nikmalarini amaliy ish faoliyatida, ilmiy-tadqiqot ishlarida, shuningdek, talim tizimida samarali foydalanishi imkonini beradi.

Ushbu uslubiy qo'llanma “Dasturiy injiniring” fanining o'quv predmeti 5330100 – Axborot tizimlarining matematik va dasturiy ta'minoti yo'nalishining davlat ta'lim standartiga mos bilim va ko'nikmalarni hosil qilishni ta'minlaydi, dunyoqarash va tizimli fikrlashni shakllantirishga ko'maklashadi va shu sohadagi mutaxassislariga dasturiy tizimlarini qo'llanilish sohalarini o'rgatadi.

1. SINFLARNI BOSHQARISH

Reja:

1. Sinf, maydon tushunchasi.
2. Sinfga kirishni nazorat qilish usullari.
3. Nazorat savollari.

Sinf, maydon tushunchasi

Sinflar ob'ekt shaklini belgilab beruvchi namunadir. Unda ma'lumotlar va shu ma'lumotlar ustida ish ko'ruvchi kod ko'rsatiladi. C# da sinf namunalari hisoblangan ob'ektlarni qurish uchun sinf spetsifikasidan foydalaniladi. Demak, sinf umuman olganda ob'ektni qurishga oid bir qator sxematik ta'riflarni o'z ichiga oladi. Bunda, sinf mantiqiy abstraksiya hisoblanishini qayd etish lozim. Sinfning jismoniy tavsifi operativ xotirada mazkur sinfning ob'ekti yaratilgandan keyingina akslantiriladi.

Sinf va tuzilmalar- umuman olganda ob'ektlarni yaratish mumkin bo'lgan namunalardir. Har bir ob'ekt berilganlar ustida kerakli amallarni bajaruvchi usullarni o'z ichiga oladi.

Sinfning umumiy berilish usuli

Sinfning berilishida tarkibiga kiruvchi ma'lumotlar, hamda shu ma'lumotlar ustida amallar bajaruvchi kod e'lon qilinadi. Agar eng sodda sinflar faqat kod yoki faqat ma'lumotlarni o'z ichiga olsa, ko'pgina haqiqiy sinflar har ikkalasini o'z ichiga oladi.

Umuman olganda, ma'lumotlar sinftomonidan beriluvchi ma'lumot a'zolarida, kod esa- funksiya-a'zolarida saqlanadi. Qayd etish joizki, C# da ma'lumot a'zolari va funksiya-a'zolarining bir nechta turlari hisobga olingan:

Sinf	
Ma'lumot-a'zolar	Funksiya-a'zolar
Maydonlar	Usullar
O'zgarmaslar	Xususiyatlar
Hodisalar	Konstruktorlar
	Finalizatorlar
	Amallar
	Indeksatorlar

Ma'lumot-a'zolar

Ma'lumot-a'zolar- bu sinf ma'lumotlari o'z ichiga olgan a'zolardir. Ma'lumot-a'zolar statik(static) bo'lishi mumkin. Sinf a'zosi static deb e'lon qilinmagan bo'lsa namuna a'zosi hisoblanadi. Ushbu ma'lumot turlarini o'rganib chiqaylik.

Maydonlar(field)

Bu sinf bilan o'xshash deb topilgan har qanday o'zgaruvchilardir

O'zgarmaslar

O'zgarmlar o'zgaruvchilar singari o'xshash usulda moslashtirilishi
Sinf class kalit o'zi yordamida yaratiladi. Quyida sodda sinfni aniqlashning umumiy shakli keltirilgan bo'lib, unda faqat namuna va usul o'zgaruvchilari saqlanadi:

```
class sinf_nomi {  
    // Namuna o'zgaruvchilarini e'lon  
    qilish_ murojaat o'zgaruvchi1 turi;  
    murojaat_o'zgaruvchi2 turi;  
    //...  
    murojaat_o'zgaruvchiN turi;  
  
    // Usullarni e'lon qilish_  
    murojaat_qaytariluvchi 1-tur (parametrlar) {  
        // usul tanasi  
    }  
    murojaat_qaytariluvchi 2-tur (parametrlar) {  
        // usul tanasi  
    }  
    //...  
    murojaat_qaytariluvchi N-tur (parametrlar) {  
        // usul tanasi  
    }  
}
```

Qayd etish joizki, o'zgaruvchi va usul e'lon qilinishidan avval murojaat ko'rsatib o'tiladi. Bu murojaat spetsifikatoridir, masalan public mazkur sinf a'zosiga murojaat tartibini belgilaydi. Sinf a'zolari sinf doirasida yopiq (private), ochiq (public), ya'ni murojaat uchun mos bo'lishi mumkin. Murojaat spetsifikatori ruxsat berilgan murojaat turini belgilaydi. Murojaat spetsifikatorini ko'rsatib o'tish shart emas, ammo u bo'lmasa, u holda e'lon qilinadigan a'zo sinf doirasida yopiq deb hisoblanadi. YOpiq murojaatli a'zolar sinfdagi boshqa a'zolar tomonidan tadbiiq etilishi mumkin.

```
Foydalanuvchining tavsiflarini ifodalovchi sinfni yaratishga misol qaraylik:  
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```
namespace ConsoleApplication1  
{  
    class UserInfo  
    {  
        // Sinf maydonlari  
        public string Name, Family, Adress;  
        public byte Age;
```



```

    // Konsolga kontak ma'lumotlarni chiqaruvchi usul
    public void writeInConsoleInfo(string name, string family, string
adress, byte age)
    {
        Console.WriteLine("Ismi: {0}\nFamiliyasi: {1}\nJoylashgan o'rni:
{2}\nYOshi: {3}\n", name, family, adress, age);
    }
}

class Program
{
    static void Main(string[] args)
    {
        // Sozdaem ob'ekt tipa UserInfo
        UserInfo myInfo = new UserInfo();

        myInfo.Name = "Alexandr";
        myInfo.Family = "Erohin";
        myInfo.Adress = "ViceCity";
        myInfo.Age = 26;

        // UserInfo sinfining yangi nusxasi yaratildi
        UserInfo myGirlFriendInfo = new UserInfo();

        myGirlFriendInfo.Name = "Elena";
        myGirlFriendInfo.Family = "Korneeva";
        myGirlFriendInfo.Adress = "ViceCity";
        myGirlFriendInfo.Age = 22;

        // Ma'lumotni konsolga chiqaramiz
        myInfo.writeInConsoleInfo(myInfo.Name,myInfo.Family,
myInfo.Adress, myInfo.Age);

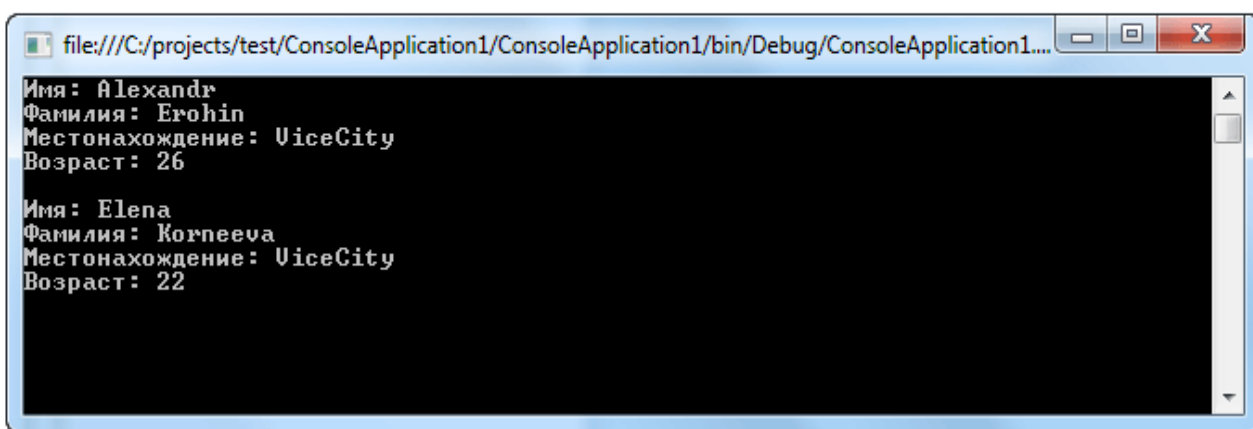
myGirlFriendInfo.writeInConsoleInfo(myGirlFriendInfo.Name,myGirlFriendInfo.
Family,myGirlFriendInfo.Adress,myGirlFriendInfo.Age);

        Console.ReadLine();
    }
}
}

```


Berilgan misolda ochiq hisoblangan (ya'ni murojaat modifikatori public bo'lgan) 4 ta maydon va 1 ta usuldan iborat yangi foydalanuvchilik sinfi UserInfo aniqlanadi. Main() usulida mazkur sinfnig ikkita nusxasi yaratiladi: myInfo va myGirlFriendInfo. So'ngra nusxalarning ma'lumot maydonlari initsiallashtiriladi va writeInConsoleInfo() usul chaqiriladi.

Oldinga qarab siljishdan avval, quyidagi asosiy tamoyilni o'rganaylik: har bir ob'ektning sinfda aniqlangan nusxa o'zgaruvchilari bo'ladi. Demak, bitta ob'ektdagi o'zgaruvchi tarkibi boshqa ob'ektdagi tarkibdan farq qilishi mumkin. Ikkala ob'ekt o'rtasida hech qanday aloqa yo'q, umumiy jihati faqatgina bir turdagi ob'ekt ekanligidir. CHunonchi, agar UserInfo turiga mansub ikkita ob'ekt berilgan bo'lsa, u holda har birida Name, Family, Age va Adress nusxalari saqlanadi, ikkala ob'ektdagi tarkibi esa farqlanishi mumkin:



```
file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
Имя: Александр
Фамилия: Ершин
Местонахождение: ViceCity
Возраст: 26

Имя: Елена
Фамилия: Корнеева
Местонахождение: ViceCity
Возраст: 22
```

1-Rasm

Object sinfi

C# --- C# bo'yicha yo'riqnoma --- Object sinfi

C# da maxsus object sinfi hisobga olingan bo'lib, u qolgan barcha sinf va turlar uchun bazaviy sinf hisoblanadi. Boshqa so'z bilan aytganda, qolgan barcha turlar object dan hosil bo'ladi. Xususan, object havolali turga mansub o'zgaruvchi har qanday boshqa turdagi ob'ektga havola qilishi mumkin. Bundan tashqari, object turiga mansub o'zgaruvchi har qanday massivga havola qilishi mumkin, chunki C# da massivlar ob'ekt sifatida joriy qilinadi. SHartli ravishda object nomi C# System.Object sinfnig .NET Framework muhiti uchun sinflar kutubxonasiga kiruvchi yana bir belgilanishidir.

Buning amaliy ahamiyati shundaki, usul va xususiyatlardan tashqari, Object sinfida aniqlangan himoyalangan va ommabop usul-a'zolarga murojaat huquqi beriladi. Bu usullar ko'pgina aniqlanuvchi sinflarda ishtirok etadi.

System.Object usullar

Quyida berilgan sinfnig barcha usullari sanab o'tilgan:

ToString()

ToString() usuli o'zi chaqiriluvchi ob'ekt ta'rifini o'z ichiga olgan belgili satrni qaytaradi. Bundan tashqari ToString() usuli ob'ekt tarkibini WriteLine() usuli yordamida chaqirganda avtomatik ravishda qaytariladi. Bu usul ko'pgina

sinflarda avvaldan e'lon qilinadi, bu esa mazkur sinflarda yaratiluvchi ma'lum turdagi ob'ektlarga moslashtirish imkonini beradi.

Bu usulni ob'ekt tarkibi to'g'risida tasavvurga ega bo'lish uchun tadbiriq eting. U ma'lumotlarni formatlashning chegaralangan vositalarini taqdim etadi. Masalan, sanalar juda ham ko'p formatlarda taqdim etilishi mumkin, ammo `DateTime.ToString()` bu borada hech qanday tanlov qoldirmaydi. Agar o'rnatilgan afzalliklar, mahalliy standartlarni e'tiborga olish kerak bo'lib, murakkab satrli ifoda kerak bo'lsa, u holda `Iformattable` interfeysni joriy etish lozim.

`GetHashCode()`

Bu usul ob'ekt xarita(`map`) sifatida tanilgan, xesh-jadval yoki lug'at deb ataluvchi ma'lumotlar tuzilmasiga joylashtirilganda tadbiriq etiladi. Ushbu tuzilmalar ustida amallar bajaruvchi sinflar tomonidan tadbiriq etiladi. Agar siz sinfdan lug'at kaliti sifatida foydalanmoqchi bo'lsangiz, u holda `GetHashCode()` ni oldindan belgilashingiz lozim. O'ta yuklanishni joriy etish borasida qat'iy talablar mavjud. yoyo

Xesh-kodni saqlanuvchi ob'ektlarga murojaat etish vositasi sifatida tadbiriq etiluvchi har qanday algoritmda tadbiriq etish mumkin. Ammo, shuni yodda tutish kerakki, `GetHashCode()` usulning standart realizatsiyasi barcha holatlarda ham o'rinli bo'lavermaydi.

`Equals()` i `ReferenceEquals()`

Kelishuvga binoan `Equals (object)` usuli chaqiruvchi ob'ekt argument sifatida ko'rsatilgan ob'ekt singari huddi shu ob'ektga havola qilishini belgilaydi, ya'ni har ikkala havolaning bir xil ekanligini tekshiradi. `Equals (object)` usuli taqqoslanuvchi ob'ektlar bir xil bo'lsa `true` mantiqiy qiymatini, aks holda `false` mantiqiy qiymatini qaytaradi. SHuningdek u yaratiluvchi sinflarda avvaldan aniqlanishi mumkin. Bu yaratiluvchi sinf uchun ob'ektlarning tengligi nima ekanligini aniqlashga imkon beradi. Masalan `Equals (object)` usuli shunday aniqlanishi mumkinki, unda ikkita ob'ektlarning tarkibi taqqoslanishi mumkin.

Ob'ektlarni taqqoslashning uchta usuli mavjud ekanligini hisobga olsak,.NET muhiti ob'ektlarning ekvivalentligini aniqlash bo'yicha anchagina murakkab sxemadan foydalanadi. SHu uchta usul va `==` taqqoslash amali o'rtasidagi farqlarni hisobga olish lozim. `Equals()` ning virtual versiyasini avvaldan aniqlashning tartib qoidalariga oid cheklovlar mavjud-chunki `System.Collections` nomlar fazosidan olingan ayrim bazaviy sinflar bu usulni chaqiradi va undan ma'lum bir reaksiyani kutadi.

`Finalize()`

Bu usulning C# dagi vazifasi S++ destruktorga mos keladi va u havolali ob'ekt tomonidan band qilingan resursni tozalash davomida chiqindilarni yig'ish uchun chaqiriladi. `Object` dan `Finalize()` ning amaliyotga joriy etilishi aslida hech qanday amal bajarmaydi va chiqindi yig'uvchi tomonidan e'tiborga olinmaydi. Odatda `Finalize()` ni ob'ekt boshqarib bo'lmaydigan resurslarga ega bo'lganda hisobga olish zarur. CHiqindi tashuvchi buni to'g'ridan-to'g'ri amalga oshira olmaydi, chunki u faqat boshqariluvchi resurslar to'g'risida ma'lumotga ega, shuning uchun u siz tomondan belgilangan yakunga tayanadi.

GetType()

Bu usul System.Type dan vorislikka olingan sinf nusxasini qaytaradi. Ushbu ob'ekt a'zosi ob'ekt bo'lgan sinf to'g'risidagi katta hajmdagi ma'lumotni, masalan bazaviy tur, usullar, xususiyatlar va h.k. larni taqdim etadi. System.Type .NET refleksiya texnologiyasining tayanch nuqtasidir.

Clone()

Bu usul ob'ekt nusxasini yaratadi va bu nusxaning havolasini qaytaradi. Qayd etish joizki, bunda chuqurmas nusxalash bajariladi, ya'ni sinfdagi barcha qiymat turlari nusxalanadi. Agar sinflar havolali turga mansub a'zolari o'z ichiga olsa, u holda o'zi ishora qiladigan ob'ektlar emas, havolalar nusxalanadi. Bu usul himoyalangandir, shuning uchun tashqi ob'ektlarni nusxalash uchun chaqirila olmaydi. SHuningdek, u virtual emas, shuning uchun uning joriy etilishini avvaldan aniqlab bo'lmaydi.

Mazkur usullardan ayrimlarining aniq misolda joriy etilishini ko'raylik:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

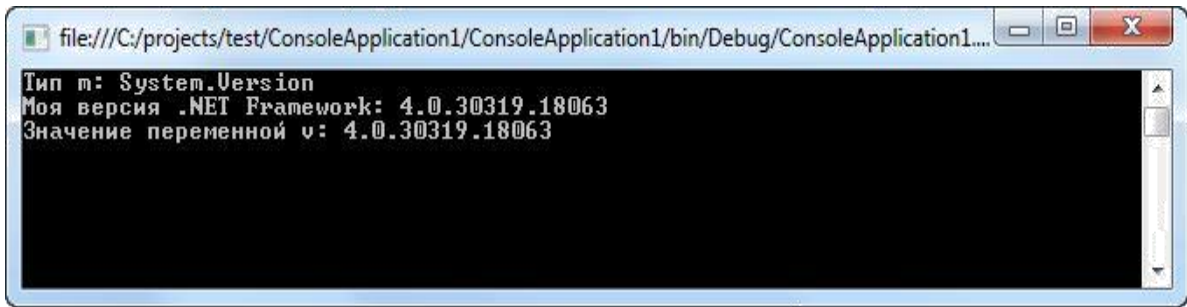
namespace ConsoleApplication1
{
    class Program
    {
        static void Main(string[] args)
        {
            var m = Environment.Version;

            Console.WriteLine("m tur: "+m.GetType());

            string s = m.ToString();
            Console.WriteLine(".NET Framework ning mening versiyam: " + s);

            Version v = (Version)m.Clone();
            Console.WriteLine("v o'zgaruvchining qiymati: "+v);

            Console.ReadLine();
        }
    }
}
```



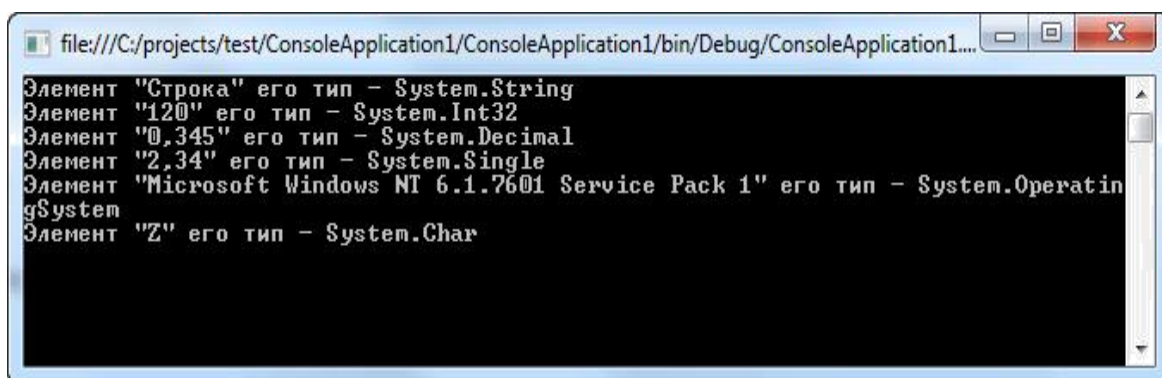
2-Rasm

object sinfi universal ma'lumotlar turi sifatida

Agar object qolgan barcha turlar uchun bazaviy sinf hisoblansa va sodda turlar qiymatlarining joriy etilishi avtomatik ravishda amalga oshirilsa, u holda object sinfidan "universal" ma'lumotlar turi sifatida foydalanish mumkin. Misol tariqasida object turidagi massiv yaratilib, uning elementlariga har xil turdagi ma'lumot qiymatlari beriladigan dasturni qaraylik:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```
namespace ConsoleApplication1  
{  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            var myOS = Environment.OSVersion;  
            object[] myArr = { "Stroka", 120, 0.345m, 2.34f, myOS, 'Z' };  
            foreach (object obj in myArr)  
                Console.WriteLine("Element \"{0}\" uning turi -  
{1}",obj,obj.GetType());  
            Console.ReadLine();  
        }  
    }  
}
```



```
file:///C:/projects/test/ConsoleApplication1/ConsoleApplication1/bin/Debug/ConsoleApplication1...
Элемент "Строка" его тип - System.String
Элемент "120" его тип - System.Int32
Элемент "0,345" его тип - System.Decimal
Элемент "2,34" его тип - System.Single
Элемент "Microsoft Windows NT 6.1.7601 Service Pack 1" его тип - System.Operatin
gSystem
Элемент "Z" его тип - System.Char
```

3-Rasm

Berilgan misoldan ko‘rinib turibdiki, object sinfiga mansub ob‘ektga havolasi orqali ixtiyoriy turdagi ma‘lumotlarga murojaat etish mumkin, chunki havolali turga mansub o‘zgaruvchida qolgan barcha turdagi ma‘lumotlarga havolalarni saqlash mumkin. Demak, dasturda ko‘rilgan object turdagi massivda ixtiyoriy turdagi ma‘lumotlarni saqlab qo‘yish mumkin. Bu g‘oyani rivojantirib object sinf ob‘ektlariga havola qiluvchi stek sinfini hech qanday qiyinchiliklarsiz yaratish mumkin. Bu stekda ixtiyoriy turdagi ma‘lumotlarni saqlashga imkon berar edi.

Object sinfining mukammal xarakteri ayrim holatlarda muvaffaqiyatli tadbiiq etilishiga qaramay, bu sinf yordamida C# da qat‘iy belgilangan tur nazoratini aylanib o‘tishning iloji yo‘q edi. Umuman olganda butun qiymatni int turidagi o‘zgaruvchida, satrni- havolali turga mansub string o‘garuvchida saqlash maqsadga muvofiqdir.

Eng muhimi, C# ning 2.0 versiyasidan boshlab asli umumlashgan ma‘lumot turlari- birlashmalar paydo bo‘ldi. Birlashmalarning joriy etilishi har xil turga mansub ma‘lumotlarni avtomatik ravishda qayta ishlovchi sinf va algoritmlarni odatiy xavfsizlikka amal qilgan holda aniqlash imkonini berdi. Umumlashmalar hisobiga object sinfidan yangi kodni yaratish davomida universal ma‘lumot turi sifatida foydalanish ehtiyoji qolmadi. Ushbu sinfning universal xarakteridan alohida holatlardagina foydalanish maqsadga muvofiqdir.

Ob‘ektlarni yaratish

Ixtiyoriy turga mansub ob‘ektни e‘lon qilish uchun quyidagi tuzilmadan foydalaniladi:

<sinf turi> o‘zgaruvchi nomi = new <sinf turi>();

Masalan:

InfoUser myinfo = new InfoUser();

Mazkur e‘lon satri uchta funksiyani bajaradi. Birinchidan, InfoUser sinfiga mansub myinfo o‘zgaruvchisi e‘lon qilinadi. Bu o‘zgaruvchining o‘zi ob‘ekt hisoblanmaydi, u ob‘ektga havola qiluvchi o‘zgaruvchi xolosdir. Ikkinchidan, ob‘ektning aniq, jismoniy nusxasi yaratiladi. Bu new operatori yordamida amalga oshiriladi. Va nihoyat, myinfo o‘zgaruvchiga mazkur ob‘ektga havola beriladi. SHu tariqa, tahlil qilinuvchi satr bajarilgandan so‘ng e‘lon qilingan myinfo o‘zgaruvchisi InfoUser turiga mansub ob‘ektga havola qiladi.

new operatori dinamik ravishda (ya‘ni bajarilish davomida) ob‘ekt uchun xotirani taqsimlaydi va unga havolani qaytaradi, so‘ngra u o‘zgaruvchida saqlab

qolinadi. Natijada, C# da barcha sinflarga mansub ob'ektlar uchun xotira dinamik ravishda qayta taqsimlanadi.

Sinf ob'ektlariga havola orqali murojaat etish mumkin bo'lganligi uchun, sinflarning nima sababdan havolali turlar deb nomlanishini izohlaydi. Qiymatli turlarning havolali turlardan asosiy farqi har bir turga mansub o'zgaruvchining tarkibiga nima kirishidadir. Binobarin, qiymat turiga mansub o'zgaruvchi aniq qiymatni, havolali o'zgaruvchi esa ob'ektning o'zini emas, unga bo'lgan havolani o'z ichiga oladi.

Havolali turga mansub o'zgaruvchilar va o'zlashtirish

O'zlashtirish amalida havolali turga mansub o'zgaruvchilar qiymat turiga mansub o'zgaruvchi, masalan int turidan boshqacha ta'sir ko'rsatadi. Qiymat turiga mansub o'zgaruvchi boshqasiga o'zlashtirilsa, vaziyat ancha osonlashadi. O'zlashtirish operatoridan chap tomonda turgan o'zgaruvchi mazkur operatoridan o'ng tomonda joylashgan o'zgaruvchi qiymatining nusxasini qabul qiladi.

Ob'ektga havola o'zgaruvchisining biri boshqasiga o'zlashtirilsa vaziyat biroz murakkablashadi, chunki bunday o'zlashtirish operatorning chap qismida joylashgan o'zgaruvchi operatorning o'ng tomonida turgan o'zgaruvchi singari bir xil ob'ektga havola qiladi. Ob'ektning o'zi esa nusxalanmaydi. Havolali turga mansub o'zgaruvchilarni o'zlashtirishdagi mazkur farq hisobiga kutilmagan natijalar kelib chiqadi.

O'zlashtirishdan so'ng birinchi o'zgaruvchida ikkinchisining qiymati saqlanadi	
1- qiymat turidagi o'zgaruvchi	2-qiymat turidagi o'zgaruvchi
1-havolali o'zgaruvchi	2-havolali o'zgaruvchi
Birinchi o'zgaruvchi ikkinchisiga havola qiladi. Bunda agar ikkinchi o'zgaruvchining qiymati o'zgarsa, birinchining ham qiymati o'zgaradi	

Misol tariqasida quyidagi kod qismini ko'raylik:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class autoCar
    {
        public string marka;
    }

    class Program
    {
        static void Main(string[] args)
    }
}

```

```

    {
        autoCar Car1 = new autoCar();
        autoCar Car2 = Car1;

        Car1.marka = "Renault";
        Console.WriteLine(Car1.marka);
        Console.WriteLine(Car2.marka);

        Console.ReadLine();
    }
}
}
}

```



4-Rasm

Car1 o'zgaruvchisi Car2 o'zgaruvchiga o'zlashtirilsa, oxir oqibat Car2 o'zgaruvchi Car1 kabi ob'ektga havola qiladi xolos. Natijada, bu ob'ektni Car1 yoki Car2 o'zgaruvchilari yordamida boshqarish mumkin. Ikkala o'zgaruvchi, Car1 va Car2 bir xil ob'ektga havola qilishiga qaramay, ular o'zaro bog'liq emas.

Ob'ektlar initsializatori

Ob'ekt initsializatorlari ob'ektni yaratish usulini hamda maydon va xususiyatlar initsializatsiyasini taqdim etadi. Agar ob'ekt initsializatorlaridan foydalanilsa, u xolda klass konstruktorining oddiy chaqirilishi o'rniga, birinchi bo'lib kiritilgan qiymatlar orqali initsiallashtiriluvchi maydon yoki xususiyat nomlari orqali beriladi. Natijada, initsializator sintaksisi sinf konstruktorini oshkor ravishda chaqirilishning o'rnini bosadi. Ob'ekt initsializatori sintaksisi LINQ-ifodada mavhum turlarni yaratishda tadbiq etiladi. Ob'ekt initsializatoridan nomlangan sinflarda foydalanish mumkinligi uchun, quyida ob'ekt initsializatsiyasi to'g'risidagi asosiy holatlar keltirilgan.

Quyida ob'ekt initsializatsiyasi sintaksisining umumiy shakli keltirilgan:

```
new klass-nomi { nomi = ifoda, ism-ifoda, ... }
```

bu erda nom maydon yoki xususiyat, ya'ni ochiq sinf a'zosi nominini ifodalaydi. Ifoda esa turi maydon yoki xususiyat turiga mos bo'lishi kerak bo'lgan initsiallashtiriluvchi ifodani belgilaydi.

Ob'ekt initsializatorlari nomlangan sinflarda tadbiq etilmaydi, ammo buning amalda iloji bor. Umuman olganda, nomlangan sinflar bilan ish tutganda oddiy konstruktorni chaqirish sintaksisidan foydalaniladi.

Ob'ekt initsializatorlaridan foydalanishga misollar:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
class autoCar
{
public string marka;
public short year;
}
class Program
{
static void Main(string[] args)
{
// initsializatoridan foydalanamiz
autoCar myCar = new autoCar { marka = "Renault", year = 2004 };
Console.ReadLine();
}
}
}
```

Usullar

Qayd etish joizki, C# ning rasmiy atamasi funksiya va usullar o'rtasida farq o'rnatadi. Mazkur atamaga ko'ra, "funksiya-a'zo" nafaqat usullarni, balki ma'lumot hisoblanmaydigan boshqa a'zolari ham o'z ichiga oladi. Bu erga indeksatorlar, amallar, konstruktorlar, destruktorlar, balki kutilmaganda-xususiyatlar kiradi. Ular ma'lumot-a'zolar: maydonlar, o'zgarmaslar va hodisalar bilan hamohang keladi.

Usullarni e'lon qilish

C# da usullarning aniqlanishi ixtiyoriy modifikatorlardan, qaytariluvchi qiymat turi, so'ngra usul nomi, argumentar ro'yxati dumaloq qavslarda, so'ngra usul tanasi figur qavslardadan tashkil topadi:

```
[modifikatorlao] qaytarilish-turi UsulNomi([parametrlar])
{
// Usul tanasi
}
```

Har bir parametr parametr turi va unga usul tanasida murojaat etish mumkin bo'lgan nomdan tashkil topadi. SHunga qo'shimcha qilib, agar usul qiymat qaytarsa, u holda chiquvchi nuqtani ko'rsatish uchun qaytariluvchi qiymat bilan birgalikda return qaytariluvchi operatoridan foydalanish mumkin.

Agar usul hech qanday qiymat qaytarmasa, u holda qaytariluvchi tur sifatida **void** ko'rsatiladi, chunki qaytarilish turini mutlaqo ko'rsatmaslikning iloji yo'q. Agarda u argument qabul qilmasa, u holda usul nomidan so'ng nima bo'lganda ham bo'sh qavslar turishi lozim. Bunda usul tanasiga qaytarish operatorini kiritish shart emas- usul yopilgan figur qavsga etgandan so'ng avtomatik ravishda boshqaruvni qaytaradi.

Usuldan qaytish va qiymat qaytarish

Umuman olganda, usuldan qaytish ikkita sharoitda sodir bo'lishi mumkin. Birinchidan, usul tanasini yopgan figur qavsga duch kelganda. Va ikkinchidan, **return** operatori bajarilganda. Return operatorining ikkita shakli mavjud: birinchisi —void turidagi usullar uchun (*usuldan qaytish*), ya'ni qiymat qaytarmaydigan usullar, boshqasi esa — aniq qiymat qaytaruvchi usullar uchun (*qiymat qaytarish*).

Quyidagi misolni ko'raylik:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
```

```
namespace ConsoleApplication1
{
    class MyMathOperation
    {
        public double r;
        public string s;

        // Doira yuzini qaytaradi
        public double sqrCircle()
        {
            return Math.PI * r * r;
        }
        // Aylana uzunligini qaytaradi
        public double longCircle()
        {
            return 2 * Math.PI * r;
        }
        public void writeResult()
        {
            Console.WriteLine("YUza yoki uzunlik hisoblansinmi? s/l:");
            s = Console.ReadLine();
            s = s.ToLower();
            if (s == "s")
```

```

    {
        Console.WriteLine("Doira yuzi {0:#.###}",sqrCircle());
        return;
    }
    else if (s == "l")
    {
        Console.WriteLine("Aylana uzunligi {0:#.##}",longCircle());
        return;
    }
    else
    {
        Console.WriteLine("Siz boshqa belgi kiritdingiz");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Radiusni kiriting: ");
        string radius = Console.ReadLine();
        MyMathOperation newOperation = new MyMathOperation { r =
double.Parse(radius) };
        newOperation.writeResult();
        Console.ReadLine();
    }
}
}

```



5-Rasm

Parametrlardan foydalanish

Usulni chaqirishda unga bir yoki bir nechta qiymatni uzatish mumkin. Usulga uzatiladigan qiymat argument deb ataladi. Argument qabul qiladigan

o'zgaruvchi shartli parametr, yoki shunchaki parametr deb ataladi. Parametrlar qavs ichida usul nomidan keyin e'lon qilinadi. Parametrlarni e'lon qilish sintaksisi o'zgaruvchilarniki kabidir. Parametrlarning ta'sir doirasi esa usul tanasidir. Argumentlarni usulga uzatishning o'ziga xos holatlaridan tashqari qolgan hollarda parametrlar qolgan barcha o'zgaruvchilar singari ish tutadi.

Umumiy holda parametrlar usulga yoki qiymat orqali, yoki havola bo'yicha uzatilishi mumkin. O'zgaruvchi havola orqali uzatilsa, chaqiriluvchi usul o'zgaruvchining o'zini qabul qiladi, shuning uchun usul ichida duch kelinadigan har qanday o'zgarishlar yakunlanganidan so'ng o'z kuchini yo'qotmaydi. Ammo, agar o'zgaruvchi qiymat orqali uzatilsa, chaqiriluvchi usul o'zgaruvchining nusxasini qabul qiladi, ya'ni undagi har qanday o'zgarishlar usul yakunlanganidan so'ng yo'qoladi. Murakkab ma'lumot turlari uchun havola orqali uzatish nusxalash kerak bo'ladigan katta hajmdagi ma'lumot hisobiga samaraliroqdir.

Quyidagi misolni qaraylik:

```
using System;
```

```
using System.Collections.Generic;
```

```
using System.Linq;
```

```
using System.Text;
```

```
namespace ConsoleApplication1
```

```
{
```

```
  class myClass
```

```
  {
```

```
    public void someMethod(double[] myArr, int i )
```

```
    {
```

```
      myArr[0] = 12.0;
```

```
      i = 12;
```

```
    }
```

```
  }
```

```
  class Program
```

```
  {
```

```
    static void Main(string[] args)
```

```
    {
```

```
      double[] arr1 = { 0, 1.5, 3.9, 5.1 };
```

```
      int i = 0;
```

```
      Console.WriteLine("Usul chaqirilishidan avval arr1 massivi: ");
```

```
      foreach (double d in arr1)
```

```
        Console.Write("{0}\t",d);
```

```
      Console.WriteLine("\n O'zgaruvchi i = {0}\n",i);
```

```
      Console.WriteLine("someMethod ... usulining chaqirilishi");
```

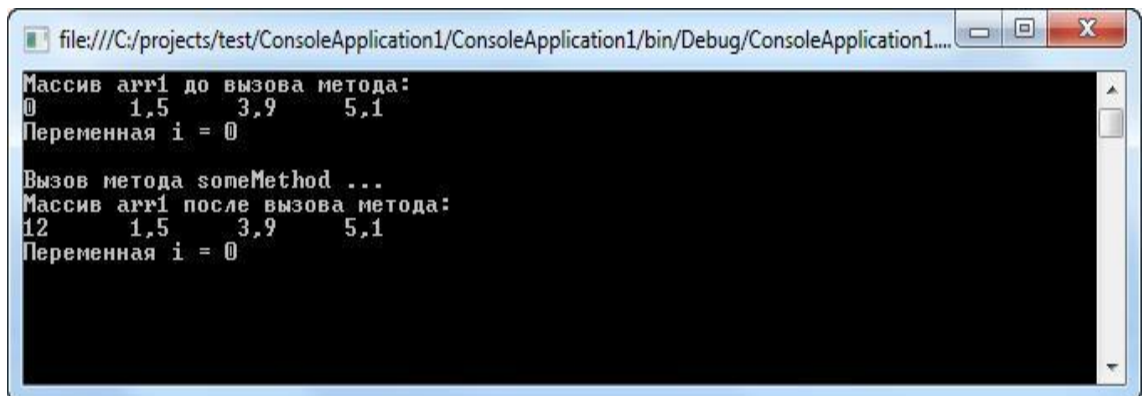
```
      myClass ss = new myClass();
```

```
      ss.someMethod(arr1,i);
```

```

        Console.WriteLine("Usul chaqirilishidan so'ng arr1 massiv");
        foreach (double d in arr1)
            Console.Write("{0}\t",d);
        Console.WriteLine("\nO'zgaruvchi i = {0}\n",i);
        Console.ReadLine();
    }
}
}

```



6-Rasm

SHunisiga alohida e'tibor berish kerakki, `i` qiymat o'zgarib qoldi, ammo `myArr` da o'zgartirilgan qiymatlar ham `arr1` boshlang'ich massivda o'zgardi, chunki massivlar havolali tur bo'lib hisoblanadi.

Satrlarning hatti-harakati ham farq qiladi. Gap shundaki, satrlar o'zgarib hisoblanadi (satr qiymatining o'zgarishi mutlaqo yangi satrning yaratilishiga olib keladi), shuning uchun satrlar havolali turlar uchun harakterli hisoblangan xatti-harakatlarni namoyon qilmaydi. Usul ichidagi satr ichida amalga oshirilgan har qanday o'zgarishlar boshlang'ich satrga ta'sir ko'rsatmaydi.

Konstruktorlar

Konstruktor ob'ektni yaratilish paytida initsiallashtiradi. Konstruktorning nomi klassniki kabidir, sintaksis nuqtai nazaridan esa u usulga o'xshashdir. Ammo konstruktorlarning oshkor ravishda ko'rsatiladigan qaytariluvchi turi yo'q. Quyida konstruktorning umumiy shakli keltirilgan:

```

murojaat klass-nomi(parametrlar-ro'yxati) {
    // konstruktor tanasi
}

```

Odatda, konstruktor sinfda aniqlangan namuna o'zgaruvchilarining boshlang'ich qiymatini berish uchun qo'llaniladi. Bundan tashqari, murojaat `public` turidagi murojaat modifikatoridir, chunki konstruktorlar ko'pincha klass ichida chaqiriladi. Parametrlar-ro'yxati esa ham bo'sh, ham bir yoki undan ortiq parametrlardan tashkil topishi mumkin.

C# ning har bir sinfi kelishuvga binoan konstruktor bilan ta'minlanadi, zaruratga ko'ra u avvaldan aniqlanishi mumkin. Ta'rifga ko'ra konstruktor argumentlar hech qachon qiymat qabul qilmaydi. Xotiraga yangi ob'ekt joylashtirilgandan so'ng, konstruktor kelishuvga binoan barcha maydonlarning mos standart qiymatlarga o'rnatilishini kafolatlaydi. Agar siz kelishuvga binoan

bunday o'zlashtirishlar binoan qanoatlanmasangiz, kelishuvga binoan konstruktorni o'z ehtiyojlaringizdan kelib chiqqan holda o'rnatishingiz mumkin.

Konstruktor bir yoki bir nechta parametr qabul qilishi mumkin. Konstruktorga parametrlar usul kabi kiritiladi. Buning uchun ularni konstruktor nomidan so'ng qavs ichida e'lon qilish etarli.

Konstruktorlarning amalda qo'llanishiga misollar ko'raylik:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class MyClass
    {
        public string Name;
        public byte Age;
        //Parametrli konstruktorni yaratamiz
        public MyClass(string s, byte b)
        {
            Name = s;
            Age = b;
        }
        public void reWrite()
        {
            Console.WriteLine("Ism: {0}\nYOsh: {1}", Name, Age);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            MyClass ex = new MyClass("Alexandr", 26);
            ex.reWrite();
            Console.ReadLine();
        }
    }
}
```

Ko'rib turganingizdek, mazkur misolda ex sinf nusxasining maydonlari konstruktor yordamida kelishuvga binoan initsiallashtiriladi.

Nazorat savollari

1. Sinflar va ob'ektlardan foydalanishning asosiy usullarini keltiring va tavsiflang?
2. Usulni qaytarish va qiymatni qaytarish o'rtasidagi farqlar qanday?
3. Parametr konstruktor qanday ishlaydi?
4. Yopiq va ochiq darslarga tavsif bering
5. Topshiriq qanday bajariladi?

2. REF VA OUT PARAMETRLARDAN FOYDALANISH, METODDAN OBE'KTNI QAYTARISH, MAJBURIY BO'LMAGAN ARGUMENTLAR

Reja

1. Ref va tashqaridan foydalanish tushunchasi.
2. Usul yarating va ob'ektni qaytaring.
3. Nazorat savollari.

Usullar parametrlar qabul qilishi yoki qabul qilmasligi, hamda chaqiriluvchi tomonga qiymat qaytarishi yoki qaytarmasligi mumkin. C# da usulning aniqlanishi anchagina tushunarli bo'lsada, argumentlarni qiziqtirgan usulga uzatish usulini boshqarish mumkin bo'lgan bir nechta kalit so'z mavjud:

Ko'pincha usullar uzatilgan argumentlarni boshqarishga majbur bo'ladi. Bunga yaqqol misol, o'z argumentlarining o'rnini almashtiruvchi Swap() usulidir. Ammo oddiy turdagi argumentlar qiymat bo'yicha uzatilgani uchun, C# da argumentni parametrga uzatish uchun qiymat orqali chaqirish me'anizmi orqali int turidagi ikkita argumentning o'rnini almashtiruvchi usulni yozib bo'lmaydi. Bu qiyinchilikni ref modifikatori hal etadi.

Sizga ma'lumki, qiymat dasturning chaqiriluvchi qismidagi usuldan return operatori yordamida qaytariladi. Ammo usul bir vaqtning o'zida faqatgina bitta qiymatni bir vaqtning o'zida qaytara oladi xolos. Usuldan ikki yoki undan ortiq ma'lumot fragmentini, masalan suzuvchi nuqtali son qiymatning butun va kasr qismini qaytarish kerak bo'lsachi? Bunday usulni out modifikatoridan foydalangan holda yozish mumkin.

YUqorida qayd qilingan kalit so'zlarning har biri egallagan o'rnini ko'rib chiqaylik.

Ref modifikatori

Parametr modifikatorlari

Parametr modifikatori	Ta'rif
------------------------------	---------------

(mavjud emas)	Agar parametr modifikator bilan hamroxlikda kelmasa, u qiymat orqali uzatiladi deb faraz qilinadi, ya'ni chaqiriluvchi usuo boshlang'ich ma'lumotlarning nusxasini qabul qiladi
Out	CHiquvchi parametrlar chaqiriluvchi usul yordamida o'zlashtiriladi (demak, havola orqali uzatiladi). Agar out parametriga chaqiriluvchi usulda qiymat berilmagan bo'lsa, kompilyator xatolik to'g'risida xabar beradi
ref	Bu qiymat avvaliga chaqiriluvchi kod tomonidan o'zlashtiriladi va istakka ko'ra chaqiriluvchi usulda takroran o'zlashtirilishi mumkin (chunki ma'lumotlar ham havola orqali uzatiladi). Agar chaqiriluvchi usulda ref parametrlarga qiymat berilmagan bo'lsa, kompilyator hech qanday xatolikni generatsiyalamaydi
params	Bu modifikator o'zgaruvchan sondagi argumentlarni bitta mantiqiy parametr ko'rinishida uzatishga imkon beradi. Har bir usulda faqatgina bitta params modifikatori ishtirok etishi mumkin va u albatta parametrlar ro'yxatida oxirgi bo'lib ko'rsatiladi. Aslida params modifikatoridan foydalanish zarurati har doim ham tug'ilavermaydi, ammo u bazaviy sinflarning kutubxonalarida ichidagi ko'plab usullarda tadbiiq etiladi.

ref parametr modifikatori qiymat orqali emas, havola orqali chaqirilishni majburiy ravishda tashkil etadi. Mazkur modifikator ham e'lon qilinishda, ham usul chaqirilishida ham ko'rsatib o'tiladi.

Bunday modifikator bilan hamoxanglikda keluvchi parametrlar havolali deyiladi va usulga amallar bajarish kerak bo'lganda, chaqiriluvchi kodda (masalan, saralash yoki almashtirish protsedurasida) e'lon qilinuvchi turli xil ma'lumot elementlarini o'zgartirishda qo'llaniladi. Havolali va chiquvchi parametrlar o'rtasidagi quyidagi farqlarga e'tibor bering:

CHiquvchi parametrlar

Bu parametrlarni usulga uzatishdan avval initsializatsiya qilish shart emas. Sababi shundaki, usulning o'zi chiquvchi parametrlarga chiqishdan avval qiymat berishi lozim.

Havolali parametrlar

Ushbu parametrlarni usulga uzatishdan avval albatta initsializatsiya qilish kerak. Sababi shundaki, u mavjud o'zgaruvchiga havolani uzatishni nazarda tutadi.

Agar boshlang'ich qiymat unga berilmagan bo'lsa, bu initsializatsiya qilinmagan lokal o'zgaruvchi ustida amal bajarish bilan tengdir.

Ref modifikatoridan foydalanishga misol ko'raylik:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace ConsoleApplication1
{
    class Program
    {
        // O'zining argumentini o'zgartiruvchi usul
        static void myCh(ref char c)
        {
            c = 'A';
        }

        // Argumentlarning o'rnini almashtiruvchi usul
        static void Swap(ref char a, ref char b)
        {
            char c;
            c = a;
            a = b;
            b = c;
        }
        static void Main()
        {
```

chaqirilishda avval c
h

n keyin c
h

chaqirilgandan

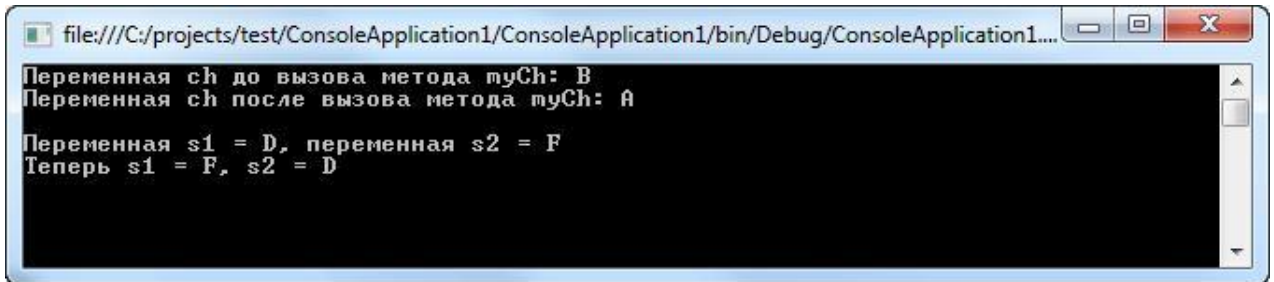
```
        char ch = 'B', s1 = 'D', s2 = 'F';
        Console.WriteLine("myCh    usul
o'zgaruvchisi: {0}",ch);
        myCh(ref ch);
        Console.WriteLine("myCh    usul
o'zgaruvchisi: {0}", ch);
```

```

Console.WriteLine("\n s1 o'zgaruvchi= {0}, s2          o'zgaruvchi= {1}",
s1, s2);
Swap(ref s1, ref s2);
Console.WriteLine("Endilikda s1 = {0}, s2 = {1}",s1,s2);

        Console.ReadLine();
    }
}
}

```



7-Rasm

Ref modifikatorga nisbatan esa quyidagini nazarda tutish kerak. Mazkur modifikator yordamida havola orqali uzatiluvchi argumentga usul chaqirilishidan avval qiymat berilishi kerak. Gap shundaki, bunday argumentni parametr sifatida qabul qiluvchi usulda parametr haqiqiy qiymatga havola qilishi nazarda tutiladi. Demak, ref modifikatordan foydalanganda usulda argumentning boshlang'ich qiymatini berib bo'lmaydi.

Out modifikatori

out parametr modifikatori ref modifikatoriga o'xshash bo'lib, ammo uning bitta farqi bor: u usul tashqarisiga qiymat uzatish uchungina xizmat qiladi. SHuning uchun out parametri sifatida tadbiiq etiluvchi o'zgaruvchiga hech qanday qiymat berish kerak emas (buning foydasi ham yo'q). Bundan tashqari, usulda out parametri initsializatsiya qilinmagan hisoblanadi, ya'ni unda boshlang'ich qiymat yo'q deb faraz qilinadi. Bu degani qiymat mazkur parametrga usulda yakuniga etguncha berilishi lozim. Demak, usul chaqirilgandan so'ng out parametri ma'lum bir qiymatga ega bo'ladi.

Misol:

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

```

```

namespace ConsoleApplication1
{

```

```

    class Program
    {

```

```

        // Sonning butun va kasr qiymatini
        // sonning kvadrati va ildizini qaytaruvchi usul
        static int TrNumber(double d, out double dr, out double sqr, out double

```

```

sqr)

```

```

    {
        int i = (int)d;
        dr = d - i;
        sqr = d * d;
        sqrt = Math.Sqrt(d);

        return i;
    }

    static void Main()
    {
        int i;
        double myDr, mySqr, mySqrt, myD = 12.987;
        i = TrNumber(myD, out myDr, out mySqr, out mySqrt);

        Console.WriteLine("Boshlang'ich son: {0}\nSonning butun qismi:
{1}\nSonning kasr qismi: {2}\nSonning kvadrati: {3}\nSonning kvadrat ildizi:
{4}",myD,i,myDr,mySqr,mySqrt);

        Console.ReadLine();
    }
}

```

8-Rasm

E'tibor berish joizki, mazkur misolda out modifikatoridan foydalanish uuldan bir vaqtning o'zida to'rtta qiymat qaytarishga imkon beradi.

Params modifikatori

C# da params kalit so'zidan foydalanish hisobiga parametrlar massividan foydalanishga ruxsat beriladi. Params kalit so'zi usulga bitta turdagi o'zgaruvchan sondagi argumentlarni yagona mantiqiy parametr ko'rinishda uzatisha imkon beradi. Params kalit so'zi bilan belgilangan argumentlar chaqiriluvchi kod qat'iy turdagi massivni yoki vergul bilan ajratilgan elementlar ro'yxatini uzatganda qayta ishlanishi mumkin.

Parametrlar massividagi elementlar soni usulga uzatiluvchi argumentlar soniga teng bo'ladi. Argumentlarni qabul qilish uchun esa dasturda mazkur massivga murojaat qilinadi:

```

using System;
using System.Collections.Generic;

```

```

using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        static void MaxArr(ref int i, params int[] arr)
        {
            // Avvaldan massivning bo'sh emasligiga oid majburiy tekshiruvni
            amalga oshirish lozim
            // na nepustotu massiva
            if (arr.Length == 0)
            {
                Console.WriteLine("Bo'sh massiv!");
                i=0;
                return;
            }
            else
            {
                if (arr.Length == 1)
                {
                    i = arr[0];
                    return;
                }
            }

            i = arr[0];
            // Maksimumni qidiramiz
            for (int j = 1; j < arr.Length; j++)
                if (arr[j] > i)
                    i = arr[j];
        }

        static void Main()
        {
            int result = 0;

            int[] arr1 = new int[8];
            int[] arr2 = new int[5];
            Random ran = new Random();
            // Ikkita massivni tasodifiy sonlar bilan to'ldiramiz
            for (int i =

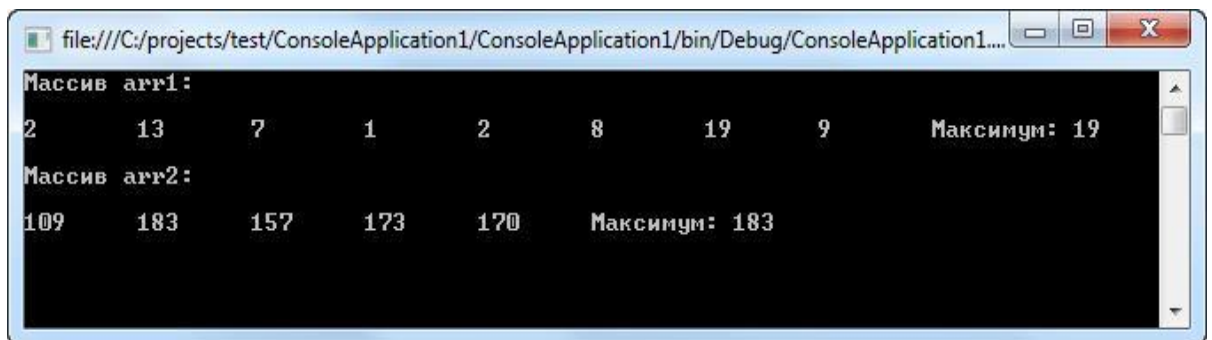
```

```
0; i < 8; i++)
```

```
    arr1[i] = ran.Next(1, 20);  
for (int i = 0; i < 5; i++)  
    arr2[i] = ran.Next(100, 200);
```

```
Console.WriteLine("Massiv arr1: \n");  
foreach (int i in arr1)  
    Console.Write("{0}\t",i);  
MaxArr(ref result, arr1);  
Console.WriteLine("Maksimum: {0}",result);
```

```
Console.WriteLine("\nMassiv arr2: \n");  
foreach (int i in arr2)  
    Console.Write("{0}\t", i);  
MaxArr(ref result, arr2);
```



9-Rasm

```
Console.WriteLine("Maksimum: {0}", result);
```

```
    Console.ReadLine();  
    }  
    }  
}
```

Usulning parametrlari oddiy hamda params turdagi uzunlikka ega bo'lgan uchchala holda ham, u mazkur usulning parametrlar ro'yxatida oxirgi bo'lib ko'rsatilishi lozim. Ammo nima bo'lganda ham, params turidagi parametr yagona bo'ladi.

2.3. Nazorat savollari

1. Ref va chiqish usullaridan foydalanish tushunchasini tavsiflang.
2. Ref modifikatori yordamida metodning qiymati qanday tayinlanadi?
3. Qaysi hollarda chiqish usulidan foydalanish parametr qiymatining bir xil parametrga berilishini bildiradi?
4. Params modifikatori yordamida kodni joyida qayta ishlash mumkinmi?

3. REKURSIYA

Reja:

1. Rekursiya tushunchasi, rekursiv usullar bilan ishlash.
2. Rekursiya mavzusiga oid masalalar yechishga misollar.
3. Test savollari.

Rekursiya

C# da usul o'z o'zini chaqirishi mumkin. Bu jarayon rekursiya, o'z-o'zini chaqiruvchi usul esa rekursiv usul deyiladi. Umuman olganda, rekursiya o'z-o'zini aniqlovchi jarayondir. Bu borada u siklik ta'rifni eslatadi. Rekuriv usulda sikldan farqli o'laroq, usul o'zichini chaqiradigan operator saqlanadi. Rekursiya dasturni boshqarishning samarali mexanizmidir.

Rekursiyaga klassik misol sonning faktorialini hisoblashdir:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Program
    {
        // Rekursivный метод
        static int factorial(int i)
        {
            int result;

            if (i == 1)
                return 1;
            result = factorial(i - 1) * i;
            return result;
        }

        static void Main(string[] args)
        {
            label1:
            Console.WriteLine("Sonni kiriting: ");
            try
            {
                int i = int.Parse(Console.ReadLine());
                Console.WriteLine("{0}! = {1}", i, factorial(i));
            }
            catch (FormatException)
            {
            }
        }
    }
}
```

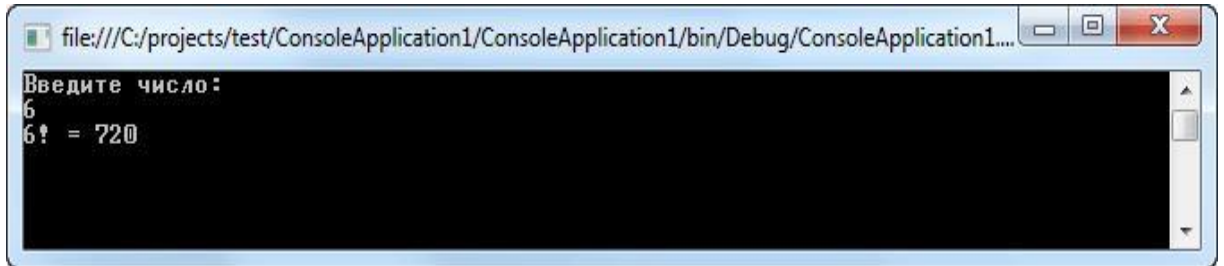


```

        Console.WriteLine("Son noto'g'ri");
        goto label1;
    }

    Console.ReadLine();
}
}
}

```



10-Rasm

SHunga alohida e'tibor berish lozimki, factorial rekursiv usuli o'z-o'zini chaqiradi, jumladan bunda i o'zgaruvchi har chaqirilishda 1 ga kamayadi.

Ko'plab protseduralarning rekursiv variantlari o'zining iteratsion ekvivalentlariga nisbatan usulning ko'p marotaba chaqirilishi hisobiga sekinroq bajarilishi mumkin. Agarda bunday chaqirilishlar juda ham ko'p bo'lsa, u holda oxir-oqibat tizimli stek to'lib ketishi mukinyu Rekursiv usulning parametrlari va lokal o'zgaruvchilari tizimli stekda saqlanib, mazkur uskl chaqirilishda ularning yangi nusxasi yaratilgani bois, qandaydir vaqtga kelib stek butkul sarf qilingan

bo'ladi. Bunday holatda istisno holat vujudga keladi va umumiy ijro etuvchi muhit istisnoni generatsiyalaydi. Ammo bu haqda rekursiv protsedura noto'g'ri bajarilgandagina qayg'urish lozim.

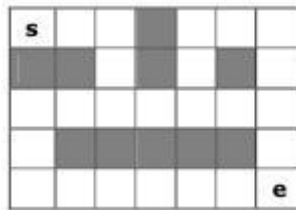
Rekursiyaning asosiy afzalligi shundaki, u iteratsion usulga qaraganda ayrim algoritmlarni aniqroq va soddaroq ko'rinishda amaliyotga joriy etish imkonini beradi. Masalan, tezkor saralash algoritmini iteratsion usulda amaliyotga joriy etish qiyin. Ammo, suniy intellekt singari ayrim masalalar aynan rekursiv yechimni talab qiladi.

Rekursiv usullarni yozishda mos o'rinda usuldan rekursiyasiz qaytishni tashkil etish uchun if shartli operatori ko'rsatib o'tish lozim. Aks holda bir marotaba chaqirilgan rekursiv usuldan umuman qaytmaslik ham mumkin. Bunday turdagi xatolik dasturlash amaliyotida rekursiyani tashkil etish uchun xosdir. Bunday holatda rekursiv usulda sodir bo'layotgan jarayonlarni kuzatish va xatolik vujudga kelganda jarayonni to'xtatish uchun WriteLine() usulni chaqiruvchi operatorlardan foydalanish tavsiya etiladi.

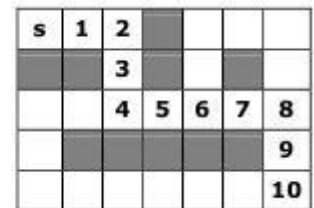
Labirintda yo'llarni izlash - Misol Bizga $N * M$ kvadratlardan iborat, to'rtburchak shaklida labirint berilgan. Har bir kvadrat kesishuvchi yoki kesishmaydigan bo'ladi. (kirish)labirintga uning chap burchagidankiradi va labirentning pastki o'ng burchakdagi chiqish yo'liga borishi lozim. (chiqish). Har

bir tavakkalchi har burilishda yuqoriga, pastga, chapga yoki o'ngga bir holatda harakatlanishiga to'g'ri keladi chunki labirint to'siqlari to'g'ridan to'g'ri tashqariga chiqib ketishiga yo'l qo'ymaydi. Bir xil holatda ikkita kesishmani kesib o'tish mumkin emas. aks holda tavakkalchi bir necha burilishni amalga oshirgach yana oldin bo'lgan joyga qaytib keladi va adashgan hisoblanadi. Labirintning kirish qismidan chiqishigacha bo'lgan barcha mumkin bo'lgan yo'llarni aks ettiruvchi kompyuter dasturini yozing.

Bu qayta takrorlash yo'li bilan oson yechiladigan masalaga oddiy misol, qayta takrorlash bilan yechim yanada murakkab va qiyinlashadi. Masalani tasavvur qilish va yechimini topish uchun bitta namuna sifatida labirintni chizaylik:



Boshlang'ich holatdan oxirigacha topshiriqning talablariga javob beruvchi 3 ta turli yo'l mavjudligini ko'rishingiz mumkin (harakat faqat kesuvchi kvadratlarga mumkin).



Yuqoridagi tasvirda 1 dan 14 gacha bo'lgan belgilangan raqamlar orqali yo'lakka mos burilishlarni ko'rsatadi.

Labirintdagi yo'llar –rekursiv algoritm

Bu masalani qanday yechishimiz mumkin? Labirintning oxiriga olib boruvchi yo'lni izlash holatini quyidagidek takroriy jarayon sifatida ko'rib chiqishimiz mumkin:

- Labirintdagi joriy holat qator yoki ustun ko'rinishida bo'lsin. Avval boshlang'ich holatdan boshlaymiz (0, 0).
- Joriy holat izlangan holat bo'lsa (N-1, M-1), yo'lni topgan bo'lamiz va uni chop etishimiz kerak. Agar ushbu holat kesishmaydigan bo'lsa, biz orqaga qaytamiz (uni bosib o'tish mumkin emas).
- Agar joriy holat allaqachon yuz bergan bo'lsa, biz orqaga qaytamiz (uni ikki marta bosib o'tishga haqli emasmiz).
- Aks holda biz 4 ta mumkin bo'lgan yo'nalishda yechim izlaymiz.
- Biz labirintdan chiqish yo'lini barcha mumkin yo'nalishlardan borishga harakatlanish orqali takror (xuddi shu algoritm bilan) izlaymiz:

- CHap tomondan: holat (qator, ustun-1);
- CHap tomondan: holat (qator, ustun-1);
- YUqoridan: holat (qator-1, ustun);
- O'ng tomondan: holat (qator, ustun+1);
- Pastdan: holat (qator+1; ustun).

Ushbu algoritmik yechimni topish uchun biz qayta o'ylaymiz. Bizga "berilgan holatdan chiqish yo'lini topish" masalasi berilgan. Bu quyidagi 4 ta vazifalarni keltirib chiqaradi:

- Ayni holatdan chiqishgacha chap tomondan yo'l izlash;
- Ayni holatdan chiqishgacha yuqori tomondan yo'l izlash;
- Ayni holatdan chiqishgacha o'ng tomondan yo'l izlash;
- Ayni holatdan chiqishgacha past tomondan yo'l izlash;

Biz har bir mumkin bo'lgan holatdan 4 ta yo'nalishni tekshiramiz va bosib o'tilgan yo'llardan harakatlanmaymiz, biz baribir yo'lni topishimiz kerak. Bu vaqtda takrorlanish oldingi masalalardek oddiy emas. har bir qadamda biz chiqish yoki taqiqlangan holatda ekanligimizni tekshirishimizga to'g'ri keladi, shundan so'ng bosib o'tilgan holatni belgilashimiz va takroran to'rta yo'nalishda izlanishni davom ettirishimiz kerak.

Takroriy harakatlardan qaytgach, boshlang'ich nuqtani bosib o'tilmagan yo'l sifatida belgilab olish lozim. Informatikada bunday sekin harakatlanish chekinish orqali izlash deb yuritiladi.

```
static char[,] lab =
```

```
{
    { " , , , * , , , " }, { ' * , ' * , ' ' , ' * , ' ' , ' * , ' ' },
    { " , , , , , , " }, { ' ' , ' * , ' * , ' * , ' * , ' * , ' ' }, { ' ' , ' ' , ' ' , ' ' , ' ' , ' ' , ' e ' },
};
```

```
static void FindPath(int row, int col)
```

```
{
    if ((col < 0) || (row < 0) ||
        (col >= lab.GetLength(1)) || (row >= lab.GetLength(0)))
    {
        // We are out of the labyrinth return;
    }

    // Check if we have found the exit if
    (lab[row, col] == 'e')
    {
        Console.WriteLine("Found the exit!");
    }

    if (lab[row, col] != ' ')
```

```

{
    // The current cell is not free return;
}

// Mark the current cell as visited lab[row,
col] = 's';

// Invoke recursion to explore all possible directions FindPath(row, col -
1); // left

FindPath(row - 1, col); // up FindPath(row,
col + 1); // right FindPath(row + 1, col); //
down

// Mark back the current cell as free
lab[row, col] = ' ';
}

static void Main()
{
    FindPath(0, 0);
}

```

Amaliy topshiriqlar

1. Barcha k sinf elementlarini qo'llab-quvvatlovchi dastur tuzing:

Kiruvchi ma'lumot: $n=3$ $k=2$

CHiquvchi ma'lumot: (1 1), (1 2), (1 3), (2 1), (2 2), (2 3), (3 1), (3 2), (3 3). Tahlil qiling va algoritmi shu kabi masalalarga qayta ishleng.

2. Barcha K element va n elementga bog'lanishidan hosil bo'lgan duplikatlarni kombinatsiyasini qo'llab quvvatlovchi va ekranga chiqaruvchi dastur tuzing.

$n=3$ $k=2$ (1 1),(1 2),(1 3),(2 2),(2 3),(3 3)

Tahlil qiling va algoritmi shu kabi masalalarga qayta ishleng.

Nazorat savollari.

1. Rekursiya tushunchasiga ta'rif bering.
2. Rekursiv usulning asosi sifatida faktorial qanday hisoblanadi?
3. Rekursiyani baholashda shartli if iborasi qachon ishlatiladi?

4. STATIC KALIT SO‘ZINI ISHLATISH, STATIK KLASSLAR

Reja

1. Statik sinflar bilan ishlash.
2. Statik sinflar.
3. Test savollari.

Static kalit so‘zi

Gohida berilgan klassning qolgan barcha ob’ektlaridan mustaqil ravishda tadbiiq etiluvchi klass a’zosini aniqlashga to’g’ri keladi. Odatda sinf a’zoriga mazkur sinf ob’ekti yordamida murojaat etiladi, lekin ayni paytda ob’ektning aniq nusxasiga havolasiz mustaqil ravishda tadbiiq etish uchun klass a’zosini yaratish mumkin. Bunday sinf a’zosini yaratish uchun eng boshida static so‘zini ko‘rsatib o‘tish etarlidir.

Agar sinf a’zosi static deb e’lon qilinsa, u ixtiyoriy ob’ektlarni yaratishdan avval qandaydir ob’ektga havola etmasdan turib ochiq bo‘lib qoladi. static kalit so‘zi yordamida ham o‘zgaruvchilarni, ham usullarni e’lon qilish mumkin. static turidagi a’zoga yaqqol misol Main() usuli bo‘lib, uning bunday e’lon qilinishina sabab, u ijro etiluvchi dasturning eng avvalida operatsion tizim tomonidan chaqirilishi kerak.

static turidagi a’zodan sinf tashqarisida foydalanish uchun mazkur sinf nomini nuqta operatori orqali ko‘rsatish etarlidir. Ammo buning uchun ob’ekt yaratish shart emas. Aslida static turiga mansub a’zo ob’ektga havola orqali emas, sinf nomi orqali ochiq bo‘ladi.

Static deb e’lon qilinadigan o‘zgaruvilar global hisoblanadi. Ob’ektlar o‘zining sinfida e’lon qiliganda static turidagi o‘zgaruvchi nusxai yaratilmaydi. Buning o‘rniga sinfning barcha nusxalari static turiga mansub bir xil o‘zgaruvchidan foydalanadi. Bunday o‘zgaruvchi sinfda tadbiiq etilishidan avval initsializatsiya qilinadi.

Static kalit so‘zidan foydalanishga misol:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

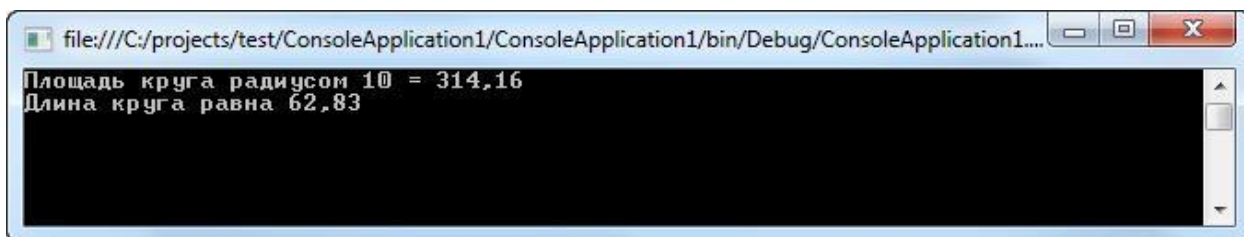
namespace ConsoleApplication1
{
    class myCircle
    {
        // aylana yuzi va uzunligini qaytaruvchi 2 ta
        usul public static double SqrCircle(int radius)
        {
            return Math.PI * radius * radius;
        }
    }
}
```

```

    }
    public static double LongCircle(int radius)
    {
        return 2 * Math.PI * radius;
    }
}
class Program
{
    static void Main(string[] args)
    {
        int r = 10;
        // Mazkur sinfning ob'ekt nusxasini yaratmasdan turib
        // Usulni boshqa sinfdan chaqirish
        Console.WriteLine("{0} radiusli aylana yuzi="
{1:###}",r,myCircle.SqrCircle(r));
        Console.WriteLine("Aylana uzunligi
{0:###}",myCircle.LongCircle(r));

        Console.ReadLine();
    }
}
}

```



11-Rasm

Static turidagi usullardan foydalanishga bir qator cheklovlar qo'yiladi:

static turidagi usulda this havolasi bo'lishi mumkin emas, chunki bunday usul biror-bir ob'ektga nisbatan bajarilmaydi.

Static turidagi usulda static turiga mansub faqat boshqa usullar bevosita chaqirilishi mumkin, ammo huddi shu klassdan nusxa usulini chaqirib bo'lmaydi. Gap shundaki, nusxa usullari aniq ob'ektlar bilan ish ko'radi, static turiga mansub usul esa ob'ekt uchun chaqirilmaydi. Demak, bunday usulning boshqarib bo'ladigan ob'ektlari yo'q.

SHunga o'xshash cheklovlar static turiga mansub ma'lumotlarga nisbatan ham qo'yiladi. Static turiga mansub usul uchun sinfdan aniqlangan static turidagi boshqa ma'lumotlar ochiq bo'ladi xolos. Xususan u o'z sinfining nusxasi bilan ish ko'ra olmaydi, chunki uning boshqarib bo'ladigan ob'ektlari bo'lmaydi

Statik konstruktorlar

Konstruktorni ham static deb e'lon qilish mumkin. Statik konstruktorlar alohida olingan ob'ektga emas, butun sinfga nisbatan tadbiq etiluvchi komponentlarni initsializatsiya qilish uchun qo'llaniladi. SHuning uchun sinf a'zolari qandaydir ob'ektlarni yaratishdan avvaol statik konstruktor tomonidan initsializatsiya qilinadi:

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;  
namespace ConsoleApplication1  
{class MyClass  
    {  
        public static int a;  
        public int b;  
  
        // Statik konstruktor  
        static MyClass()  
        {  
            a = 10;  
        }  
        // Oddiy konstruktor  
        public MyClass()  
        {  
            b = 12;  
        }  
    }  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Console.WriteLine("a sinf nusxasiga murojaat: " + MyClass.a);  
  
            MyClass obj = new MyClass();  
            Console.WriteLine("b sinf nusxasiga murojaat: " + obj.b);  
  
            Console.ReadLine();  
        }  
    }  
}
```


SHunga e'tiborimizni qaratish lozimki, static turiga mansub konstruktor sinf birinchi marotaba yuklanganida, jumladan nusxa konstruktoridan avval avtomatik tarzda chaqiriladi. Bunday umumiy xulosa chiqarish mumkin; statik konstruktor ixtiyoriy nusxa konstruktoridan avval bajarilishi lozim. Bundan tashqari, statik konstruktorlarda murojaat modifikatorlari yo'q- ular kelishuvga binoan murojaatdan foydalanadi, demak ularni dasturdan turib chaqirib bo'lmaydi.

Statik sinflar

Sinfni static deb e'lon qilish mumkin. Statik sinf ikkita asosiy xususiyatga ega. Birinchidan, statik sinf ob'ektlarini yaratish mumkin emas. Ikkinchidan, statik sinf faqat statik a'zolaridan tashkil topishi lozim. Statik sinf static kalit so'zi yordamida ko'rinishi o'zgartirilgan quyida keltirilgan shakl bo'yicha yaratiladi.

static class sinf nomi { // ...

Statik sinflar asosan ikki xolda tadbqiq etiladi. Birinchidan statik sinf usul yartilganda kengaytmalarni talab qiladi. Kengaytma usullari asosan LINQ tili bilan bog'langan. Va ikkinchidan, statik sinf o'zaro bog'langan statik usullar majmuini saqlash uchun xizmat qiladi:

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

namespace ConsoleApplication1

{Berilgan sinfda eng sodda amallarni bajaruvchi statik usullar inkapsulyasiya qilinadi

static class MyMath

{

// Sonning butun qismi

static public int round(double d)

{

return (int)d;

}

// Sonning kasr qismi

static public double doub(double d)

{

return d - (int)d;

}

// Sonning kvadrati

static public double sqr(double d)

{

eturn d * d;

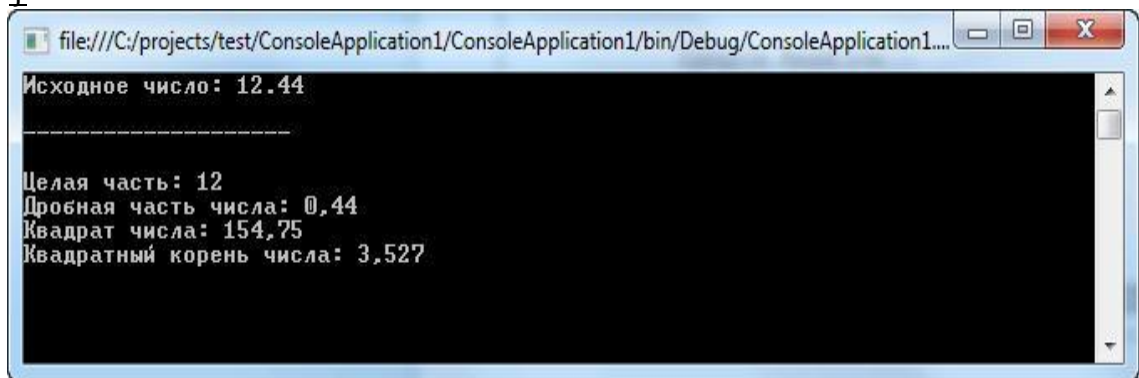
}

```

    // Sonning kvadrat ildizi
    static public double sqrt(double d)
    {
        return Math.Sqrt(d);
    }
}
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Boshlang'ich son: 12.44\n\n-----
\n");
        Console.WriteLine("Butun qismi: {0}", MyMath.round(d: 12.44));
        Console.WriteLine("Sonning kasr qismi: {0}", MyMath.doub(d:
12.44));
        Console.WriteLine("Sonning kvadrati: {0:###}", MyMath.sqr(d:
12.44));
        Console.WriteLine("Sonning kvadrat ildizi:
{0:#####}", MyMath.sqrt(d: 12.44));

        Console.ReadLine();
    }
}

```



12-Rasm

Qayd etish joizki, statik sinf nusxasining konstruktori bo'lmaydi, uning statik konstruktori bo'la oladi xolos.

Nazorat savollari.

1. Sinf tanlash tartibini aytib bering.
2. Statik o'zgaruvchilarni global deb e'lon qilish mumkinmi?
3. Statik o'zgaruvchilardan foydalanishga misollar keltiring.
4. Konstruktorga statik tip qanday beriladi?
5. Statik sinfdan foydalanish tartibini aytib bering.

5. INDEKSATORLAR VA XUSUSIYATLAR

Reja:

1. Maxsus kirish usullari.
2. Kirish modifikatorlari.
3. Inkapsulyatsiya.
4. Indekslovchilar.
5. Nazorat savollari

Xususiyatlar va inkapsulyasiya

Oddiy usullardan tashqari, C# tilida **xususiyat** deb ataluvchi murojaat usullari nazarda tutiladi. Ular sinf maydonlariga oddiy murojaatni ta'minlaydi, qiymatlarini bilib beradi yoki o'rnatish ishlarini amalga oshiradi.

Xususiyatning standart ta'rifi quyidagi sintaksisga ega:

```
1 [murojaat-modifikatori] qaytariluvchi-tur ixtiyoriy-nom
2 {
3     // xususiyat kodi
4 }
```

Masalan:

```
1 class Person
2 {
3     private string name;
4
5     public string Name
6     {
7         get
8         {
9             return name;
10        }
11
12        set
13        {
14            name = value;
15        }
16    }
17 }
```

Bu erda name yopiq maydon va ochiq Name xususiyat mavjud. Ularning registrini hisobga olmasa, deyarli bir xil nomi bo'lsa ham, bu stil xolosdir, ularning nomi ixtiyoriy bo'lib, ustma-ust tushishi shart emas.

Buxususiyatorqalibiz name _____ o'zgaruvchiga murojaatni boshqarishimiz mumkin. Xususiyatning standart ifodasi **get** va **set** bloklardan iborat bo'ladi. get blokida maydon qiymati qaytarilsa, set blokida bu qiymat o'rnatiladi. value parametri uzatiladigan qiymatni ifodalaydi.

Biz berilgan xususiyatdan quyidagi ko'rinishda foydalanishimiz mumkin:

```
1 Person p = new Person();
2 // Xususiyatni o'rnatamiz - Set blok ishga tushib ketadi
3 // "Tom" qiymat xususiyatga uzatiluvchi value dir
4 p.Name = "Tom";
5
6 // Xususiyatning qiymatini qabul qilamiz va uni o'zgaruvch - Getbloki ishga
7 tushadi
8 string personName = p.Name;
```

Bu erda sinfning oddiy maydonlaridan foydalanish mumkin bo'lsa, u xolda hossalari nima uchun kerak degan savol tug'ilishi mumkin. Lekin xususiyatlar sinf o'zgaruvchisiga ma'lum bir qiymat berish uchun kerak bo'ladigan qo'shimcha mantiqni biriktirish imkonini beradi. Masalan bizga yosh bo'yicha tekshiruvni amalga oshirish kerak bo'lsin:

```
1 class Person
2 {
3     private int age;
4
5     public int Age
6     {
7         set
8         {
9             if (value < 18)
10            {
11                Console.WriteLine("YOsh 17 dan katta bo'li
12            }
13            else
14            {
15                age = value;
16            }
17        }
18        get { return age; }
```

```

19     }
20 }

```

set va get bloklari xususiyatda bir vaqtning o'zida ishtirok etishi shart emas. Agar xususiyat faqatgina get blokini aniqlasa, u holda bu xususiyatni faqat o'qish uchun ochish mumkin- biz uning qiymatini olishimiz mumkin, lekin uni o'rnatish olmaymiz. Va aksincha, agarda xususiyat faqat set blokiga ega bo'lsa, u holda bu xususiyatdan faqat yozish uchun foydalanish mumkin- qiymatni faqat o'rnatish mumkin, lekin uni olib bo'lmaydi:

```

class Person
1  {
2      private string name;
3      //faqat o'qish uchun mo'ljallangan xususiyat
4      public string Name
5      {
6          get
7          {
8              return name;
9          }
10     }
11
12     private int age;
13     // faqat yozish uchun mo'ljallangan xususiyat
14     public int Age
15     {
16         set
17         {
18             age = value;
19         }
20     }
21 }

```

Murojaat modifikatorlari

Biz murojaat modifikatorlarini nafaqat butun xususiyatga nisbatan, balki alohida bloklar - yoki get, yoki set ga nisbatan tadbqiq etishimiz mumkin:

```

1  class Person
2  {
3      private string name;
4
5      public string Name
6      {

```

```

7         get
8         {
9             return name;
10        }
11
12        private set
13        {
14            name = value;
15        }
16    }
17    public Person(string name, int age)
18    {
19        Name = name;
20        Age = age;
21    }
22 }

```

Endilikda yopiq blok set dan faqatgina berilgan sinfda- uning usullari, xususiyatlari, konstruktorlarida tadbiiq etishimiz mumkin, lekin boshqa sinflarda tadbiiq etishning iloji yo‘q:

```

1 Person p = new Person("Tom", 24);
2
3 // Xato - set private modifikatori orqali e‘lon qilingan
4 //p.Name = "John";
5
6 Console.WriteLine(p.Name);

```

Xususiyatlarda modifikatorlardan foydalanishda bir qator cheklovlarni hisobga olish lozim:

- set yoki get bloki uchun modifikatorini xususiyat ikkala blokka (ham set, ham get) ega bo‘lgandagina o‘rnatish mumkin
- Faqatgina bitta blok set yoki get murojaat modifikatoriga ega bo‘lishi mumkin, lekin ikkalasi bir vaqtning o‘zida ega bo‘la olmaydi
- Set yoki get blok modifikatori xususiyat murojaati modifikatoriga qaraganda chegaralovchi bo‘lishi kerak. Masalan, agar xususiyat public modifikatoriga ega bo‘lsa, u holda set/get bloki faqatgina protected internal, internal, protected, private modifikatoriga ega bo‘lishi mumkin.

Inkapulyatsiya.

YUqorida biz xususiyat orqali sinfnig xususiyy o‘zgaruvchilariga murojaat o‘rnatilishini ko‘rib chiqdik. Sinf holatini tashqaridan aralishishlardan yashirish **inkapsulyasiya** mexanizmi deb atalib, u ob‘ektga-yo‘naltirilgan dasturlashning

asosiy qoidalaridan birini ifodalaydi. (Qayd etish joizki, inkapsulyasiya tushunchasining o'zi har doim ham o'zaro kesishmaydigan har xil talqinda qabul qilinadi). private turidagi murojaat modifikatorlaridan foydalanish o'zgaruvchini tashqi murojaatdan himoyalaydi. Murojaatni boshqarish uchun ko'pgina dasturlash tillarida maxsus usullar, getterlar va setterlardan foydalaniladi. C# da ularning vazifasini odatda xususiyatlar bajaradi.

Misol tariqasida ma'lum bir Account sinfini olaylik, unda sum maydon aniqlangan bo'lib, u quyidagi yig'indini ifodalasin:

```
1 class Account
2 {
3     public int sum;
4 }
```

sum o'zgaruvchisi ommaviy bo'lgani uchun, dasturning ixtiyoriy joyida unga murojaat etishimiz, o'zgartirishimiz, qandaydir qabul qilinishi mumkin bo'lmagan qiymat, masalan manfiy son berishimiz mumkin. Bunday hatti-harakat ma'qullanmasa kerak. SHuning uchun sum o'zgaruvchiga murojaatni cheklash hamda sinf ichida uni yashirish uchun inkapsulyasiyadan foydalaniladi:

```
1 class Account
2 {
3     private int sum;
4     public int Sum
5     {
6         get {return sum;}
7         set
8         {
9             if (value > 0)
10            {
11                sum=value;
12            }
13        }
14    }
15 }
```

Avtomatik xususiyatlar

Xususiyatlar sinf maydonlariga murojaatni boshqaradi. Ammo, agar bizda o'n va undan ortiq maydon berilgan bo'lsa, u holda har bir maydonni aniqlash va u uchun bir turga mansub xususiyatni yozish juda ko'p vaqtni olar edi. SHuning uchun .NET freymvorkda avtomatik xususiyatlar qo'shilgan edi. Ular qisqartirilgan e'longa ega:

```
1 class Person
```

```

2    {
3        public string Name { get; set; } public int Age
4        { get; set; }
5
6        public Person(string name, int age)
7        {
8            Name = name;
9            Age = age;
10       }
11    }

```

Aslida bu erda xususiyatlar uchun ham maydonlar yaratiladi, faqat ularni dasturchi kod ichida emas, kompilyator avtomatik ravishda kompilyasiya jarayonida yaratadi.

Avto xususiyatlarning afzalligi nimada, ular avtomatik ravishda yaratiladigan o‘zgaruvchiga murojaat etadi xolos, o‘zgaruvchiga avtoxususiyatlarsiz murojaat etib bo‘lmaydimi? Gap shundaki, ixtiyoriy vaqt momentida zarurat tug‘ilganda avtoxususiyatni oddiy xususiyatga aylantiriimiz, unga ma’lum mantiq kiritishimiz mumkin.

SHuni alohida qayd etish joizki, avtomatik xususiyatni standart xususiyatlar kabi faqat yozish uchun yaratib bo‘lmaydi.

Avtoxususiyatlarga kelishuvga binoan qiymat berish mumkin (avtoxususiyatlarni ishga tushirish):

```

1    class Person
2    {
3        public string Name { get; set; } = "Tom";
4        public int Age { get; set; } = 23;
5    }
6
7    class Program
8    {
9        static void Main(string[] args)
10       {
11           Person person = new Person();
12           Console.WriteLine(person.Name); // Tom
13           Console.WriteLine(person.Age); // 23
14
15           Console.Read();
16       }
17    }

```


Agarda biz Person ob'ekt uchun Name va Age xususiyat qiymatlarini ko'rsatib o'tmasak, u holda kelishuvga binoan qiymat ishga tushadi.

Avtoxususiyatlar ham murojaat modifikatorlariga ega bo'lishi mumkin:

```
1 class Person
2 {
3     public string Name { private set; get; }
4     public Person(string n)
5     {
6         Name = n;
7     }
8 }
```

Biz set blokini olib tashlab, avtoxususiyatni faqat o'qish uchun ochishimiz mumkin. Bu holda xususiyatning qiymatini saqlash uchun readonly modifikatorli maydon yaratiladi, shuning uchun bu kabi get-xususiyatlarni sinf konstruktoridan yoki xususiyatni e'lon qilishda o'rnatish mumkinligini hisobga olish lozim:

```
1 class Person
2 {
3     public string Name { get; } = "Tom"
4 }
```

Xususiyatlarning qisqartirilgan yozuvi

Usullar singari, xususiyatlarni ham qisqartirishimiz mumkin. Masalan:

```
1 class Person
2 {
3     private string name;
4
5     // public string Name { get { return name; } } ga ekvi
6     public string Name => name;
7 }
```

Indeksatorlar

Indeksatorlar ob'ektlarni indekslash va ma'lumotlarga indekslar orqali murojaat etish imkonini beradi. Umuman olganda indeksatorlar yordamida ob'ektlar bilan massivlar kabi ishlash mumkin. SHaklan ular qiymat qaytaruvchi va o'zlashtiruvchi get va set standart blokli xususiyatlarni eslatadi.

Indeksatorning shartli ravishda aniqlanishi:

```
1 Qaytariluvchi-tur this [Tip parametr1, ...]
2 {
```

```
3 get { ... }
4 set { ... }
5}
```

Xususiyatlardan farqli o'laroq indeksatorlarning nomi bo'lmaydi. Uning o'rniga **this kalit so'zi**, so'ngra kvadrat qavslarda parametrlar keladi. Indeksator kamida bitta parametrga ega bo'lishi kerak.

Misolda qaraylik. Bizda insonni ifodalovchi Person sinf va odamlar guruhini ifodalovchi People sinf berilgan bo'lsin. People sinfini aniqlash uchun indeksatorlardan foydalanaylik:

```
1 class Person
2 {
3     public string Name { get; set; }
4 }
5 class People
6 {
7     Person[] data;
8     public People()
9     {
10        data = new Person[5];
11    }
12    // indeksator
13    public Person this[int index]
14    {
15        get
16        {
17            return data[index];
18        }
19        set
20        {
21            data[index] = value;
22        }
23    }
24}
```

public Person this[int index] tuzilma indeksatorni ifodalaydi. Bu erda qaytariluvchi yoki o'zlashtiriluvchi ob'ekt, ya'ni Person tur aniqlanadi. Ikkinchidan, int index parametr orqali elementlarga murojaat usulini aniqlaymiz.

Umuman olganda barcha Person ob'ektlar Person sinf ichida data massivida saqlanadi. Ularni indeks bo'yicha olish uchun indeksatorida get bloki aniqlanadi:

```
1get
```

```

2{
3  return data[index];
4}

```

Indeksator Person turida bo‘lgani uchun , get blokida shu turga mansub ob’ektni return operator yordamida qaytarish lozim. Bu erda xilma–xil mantiq aniqlanishi mumkin. Bizning holimizda data massividan ob’ektni qaytaramiz xolos.

set blokida **value** parametri orqali uzatilgan Person ob’ektni qaytaramiz va uni massivga indeks bo‘yicha saqlaymiz.

```

1set
2{
3  data[index] = value;
4}

```

SHundan so‘ng biz People ob’ekti bilan Person ob’ektlar to‘plami singari ishlashimiz mumkin:

```

1 class Program
2   {
3   static void Main(string[] args)
4   {
5       People people = new People();
6       people[0] = new Person { Name = "Tom" };
7       people[1] = new Person { Name = "Bob" };
8
9       Person tom = people[0];
10      Console.WriteLine(tom?.Name);
11
12      Console.ReadKey();
13  }
14}

```

Indeksator kutilganidek indekslar tuzilishini parametrlar ko‘rinishida qabul qiladi. Ammo indekslar int turida bo‘lishi shart emas. Masalan, biz ob’ektni xususiyatlar ombori sifatida qabul qilishimiz va atribut nomini satr ko‘rinishida uzatishimiz mumkin:

```

1 class User
2 {
3   string name;
4   string email;
5   string phone;
6

```

```

7   public string this[string propname]
8   {
9       get
10      {
11          switch (propname)
12          {
13              case "name": return "Mr/Ms. " + name;
14              case "email": return email;
15              case "phone": return phone;
16              default: return null;
17          }
18      }
19      set
20      {
21          switch (propname)
22          {
23              case "name":
24                  name = value;
25                  break;
26              case "email":
27                  email = value;
28                  break;
29              case "phone":
30                  phone = value;
31                  break;
32          }
33      }
34  }
35}
36class Program
37{
38    static void Main(string[] args)
39    {
40        User tom = new User();
41        tom["name"] = "Tom";
42        tom["email"] = "tomekvilmovskiy@gmail.ru";
43
44        Console.WriteLine(tom["name"]); // Mr/Ms. Tom
45
46        Console.ReadKey();
47    }
48}

```

Bir nechta parametrlarni tadbiq etish

SHuningdek indeksatorlar bir nechta parametrni qabul qilishi mumkin. Bizda ombor ikki o'lchovli massiv yoki matritsa ko'rinishida aniqlangan sinf berilgan bo'lsin:

```
1 class Matrix
2 {
3     private int[,] numbers = new int[,] { { 1, 2, 4}, { 2, 3, 6 }, { 3, 4, 8 } };
4     public int this[int i, int j]
5     {
6         get
7         {
8             return numbers[i,j];
9         }
10    set
11    {
12        numbers[i, j] = value;
13    }
14 }
15}
```

Endi indeksatorni aniqlash uchun ikkita i va j indeksdan foydalaniladi. Endi dasturda biz ob'ektga ikkita indeksdan foydalangan holda murojaat etishimiz kerak:

```
1 Matrix matrix = new Matrix();
2 Console.WriteLine(matrix[0, 0]);
3 matrix[0, 0] = 111;
4 Console.WriteLine(matrix[0, 0]);
```

SHuni alohida qayd etish joizki, indeksator statik bo'la olmaydi va sinf nusxasiga nisbatan tadbiiq etiladi xolos. Lekin bunda ideksatorlar virtual va abstrakt bo'la oladi va hosilaviy sinflarda aniqlanishi mumkin.

get va set bloklari

Xususiyatlardagi kabi, indeksatorlarda zarurat bo'lmasa get yoki set blokini yozmasa ham bo'ladi. Masalan, set blokini olib tashlab, indeksatorni faqat o'qish uchun ochamiz:

```
1 class Matrix
2 {
3     private int[,] numbers = new int[,] { { 1, 2, 4}, { 2, 3, 6 }, { 3, 4, 8 } };
4     public int this[int i, int j]
5     {
6         get
```

```

7     {
8     return numbers[i,j];
9     }
10  }
11}

```

SHuningdek, murojaat modifikatorlaridan foydalanib, get va set bloklariga murojaatni cheklashimiz mumkin. Masalan, set blokini xususiylashtiraylik:

```

1 class Matrix
2 {
3     private int[,] numbers = new int[,] { { 1, 2, 4 }, { 2, 3, 6 }, { 3, 4, 8 } };
4     public int this[int i, int j]
5     {
6         get
7         {
8             return numbers[i,j];
9         }
10    private set
11    {
12        numbers[i, j] = value;
13    }
14 }
15}

```

Indeksatorlarning o‘ta yuklanishi

Usullar singari, indeksatorlarni o‘ta yuklash mumkin. Bunda indeksatorlar soni, turi yoki parametrlarning kelish tartibiga ko‘ra farq qilishi mumkin. Masalan:

```

class Person
{
    public string Name { get; set; }
    public int Age { get; set; }
}
class People
{
    Person[] data;
}

```

```

public People()
{
    data = new Person[5];
}

public Person this[int index]
{
    get
    {
        return data[index];
    }
    set
    {
        data[index] = value;
    }
}

public Person this[string name]
{
    get
    {
        Person person = null;
        foreach(var p in data)
        {
            if(p?.Name == name)
            {
                person = p;
                break;
            }
        }
        return person;
    }
}
}

```

```

}
class Program
{
    static void Main(string[] args)
    {
        People people = new People();
        people[0] = new Person { Name = "Tom" };
        people[1] = new Person { Name = "Bob" };
        Console.WriteLine(people[0].Name);    // Tom
        Console.WriteLine(people["Bob"].Name); // Bob
        Console.ReadKey();
    }
}

```

Bu holda People sinfi ikkita indeksator versiyasidan tashkil topadi. Birinchi versiya Person ob'ektini indeks bo'yicha qabul qiladi va o'rnatadi, ikkinchisi esa Person ob'ektini nomi bo'yicha qabul qiladi.

Nazorat savollari

1. Maxsus kirish usullarini bering va tavsiflang.
2. Get va set bloklari qanday hollarda ishlatiladi?
3. Dasturiy ta'minotni yaratishda kirish modifikatorlaridan foydalanishni asoslang.
4. Inkapsulyatsiya jarayonini tavsiflang. Uning maqsadi nima?
5. Get va set bloklari bilan ishlashda indeksarlardan foydalanish mumkinmi?

6. MEROSXURLIK

Reja:

1. OOPda meros tushunchasi.
2. Olingan sinfdan asosiy sinf a'zolariga kirish.
3. Ixtiyoriy sinflardagi konstruktorlar.
4. Nazorat savollari.

OOPda meros tushunchasi.

Vorislik(inheritance) OMD ning asosiy jihatlaridan biri hisoblanadi. Vorislik tufayli bitta sinf boshqa sinfning funksionallagini o'zlashtirishi mumkin.

Bizga alohida olingan shaxsni ta'riflovchi Person sinfi berilgan bo'lsin:

```
1 class Person
2 {
3     private string _name;
4
5     public string Name
6     {
7         get { return _name; }
8         set { _name = value; }
9     }
10    public void Display()
11    {
12        Console.WriteLine(Name);
13    }
14 }
```

Lekin bizga korxonada xodimini ta'riflovchi sinf – Employee sinfi kerak bo'lib qoldi. Bu sinf xodim- ta'riflangan shaxsning o'zi bo'lgani uchun Person singari funksionalni bajargani tufayli, Employee sinfini Person sinfining xosilasi (yoki vorisi, yoqi qism sinfi) sifatida qabul qilish maqsadga muvofiqdir, bu sinf esa o'z navbatida asosiy sinf yoki avlod (yoki superklass) deyiladi:

```
1 class Employee : Person
2 {
3
4 }
```

Ikki nuqtadan so'ng biz berilgan sinf uchun asosiy sinfni ko'rsatamiz. Employee sinfi uchun Person bazaviy hisoblanadi, shuning uchun Employee sinfi Person da bor xususiyat, usul, maydonlarni meros qilib oladi. Vorislikda uzatilmaydigan yagona narsa bazaviy sinfning konstruktoridir.

SHunday qilib, vorislik **is-a** (bo'ladi) munosabatni amaliyotga joriy etadi, Employee sinf ob'ekti Person sinfining ham ob'ekti hisoblanadi:

```
1 static void Main(string[] args)
2 {
3     Person p = new Person { Name = "Tom" };
4     p.Display();
5     p = new Employee { Name = "Sam" };
6     p.Display();
7     Console.Read();
8 }
```

Employee ob'ekt Person ning ham ob'ekti bo'lgani bois, o'zgaruvchini quyidagicha aniqlashimiz mumkin: Person p = new Employee().

Kelishuvga binoan barcha sinflar vorislik oshkor ravishda o'rnatilmasa ham **Object** asosiy sinfdan vorislanadi. SHuning uchun yuqorida aniqlangan Person va Employee sinflari

shaxsiy usullaridan tashqari Object sinf usullarini ham o'zlashtiradi: ToString(), Equals(), GetHashCode() i GetType().

Barcha sinflar kelishuvga binoan vorislanishi mumkin. Lekin bu erda qator cheklovlar mavjud:

- Ko'plik vorislik qo'llab-quvvatlanmaydi, sinf faqat bitta sinfdan vorislanishi mumkin.
- Hosilaviy sinfni yaratishda asosiy sinfga murojaat turini hisobga olish lozim-hosilaviy sinfga murojaat turi asosiy sinf kabidir va undanda qat'iydir. YA'ni, agar bazaviy sinf **internal** murojaat turiga ega bo'lsa, u holda hosilaviy sinf **internal** yoki **private** murojaat turiga ega bo'ladi, lekin **public** bo'la olmaydi.
- Agar sinf **sealed** modifikatori yordamida e'lon qilingan bo'lsa, u holda bu sinfdan hosilaviy sinflarni vorislab bo'lmaydi va yaratib ham bo'lmaydi. Masalan quyidagi sinf vorislarni yaratishga yo'l qo'ymaydi:

```
1 sealed class Admin
2 {
3 }
```

Statik sinfdan sinfni vorislab bo'lmaydi.

Asosiy sinf a'zolariga voris sinfdan murojaat qilish

Person va Employee sinflarimizga qaytaylik. Employee bor funksionalini Person sinfdan vorislasada, quyidagi holatda nima sodir bo'lishini ko'raylik:

```
1 class Employee : Person
2 {
3     public void Display()
4     {
5         Console.WriteLine(_name);
6     }
7 }
```

Bu ishga tushmaydi va xato chiqaradi, chunki kod _name o'zgaruvchisi modifikatori bilan e'lon qilingan, shuning uchun unga Person sinf private murojaat eta oladi. Lekin Person sinfdan ommaviy Name sinf e'lon qilingan bo'lib, biz undan foydalana olamiz, shuning uchun quyidagi kod normal ravishda ishlaydi:

```
1 class Employee : Person
2 {
3     public void Display()
4     {
5         Console.WriteLine(Name);
6     }
7 }
```

SHu tariqa, keltirilgansinf **private protected** (agar asosiy yoki hosilaviy sinf bitta to'plamda joylashgan bo'lsa), **public**, **internal**, **protected** i **protected internal** modifikatorlari orqali e'lon qilingan sinf a'zolarigagina murojaat etishi mumkin.

Base kalit so'zi

Endi sinflarimizga konstruktorlarni qo'shamiz:

```
1 class Person
2 {
3     public string Name { get; set; }
4
5     public Person(string name)
6     {
7         Name = name;
```

```

8      }
9
10     public void Display()
11     {
12         Console.WriteLine(Name);
13     }
14 }
15
16 class Employee : Person
17 {
18     public string Company { get; set; }
19
20     public Employee(string name, string company)
21         : base(name)
22     {
23         Company = company;
24     }
25 }

```

Person sinfi Name xususiyatsini oʻrnatuvchi konstruktorga ega. Employee sinfi Name xususiyatsini meros qilib olgani va oʻrnatgani uchun, oʻrnatiluvchi kodni yuz martalab yozmasdan, Person sinfining mos qismini chaqirish maqsadga muvofiqdir. SHuningdek, asosiy sinfning konstruktorida oʻrnatilishi kerak boʻlgan xususiyat va parametrlar anchagina koʻp boʻlishi mumkin.

Base kalit soʻzi yordamida biz asosiy sinfga murojaat etishimiz mumkin. Bizning holimizda Employee sinf konstruktorida ism va tashkilotni koʻrsatishimiz lozim. Ammō ismni biz asosiy sinfning konstruktoriga, yaʼni Person sinfining konstruktoriga base(name) ifodasi yordamida uzatamiz.

```

1  static void Main(string[] args)
2  {
3      Person p = new Person("Bill");
4      p.Display();
5      Employee emp = new Employee ("Tom", "Microsoft");
6      emp.Display();
7      Console.Read();
8  }

```

Hosilaviy sinfdagi konstruktorlar

Konstruktorlar vorislikda hosilaviy sinfga uzatilmaydi. Va agar asosiy sinfda kelishuvga binoan parametrsiz konstruktor aniqlanmasdan, faqatgina parametrli

konstruktorlar aniqlangan bo'lsa (Person bazaviy sinf kabi), u holda hosilaviy sinfda bu konstruktorlarning birini base kalit so'zi orqali chaqirishimiz lozim. Masalan, Employee sinfidan konstruktorning aniqlanishini olib tashlaymiz:

```
1 class Employee : Person
2 {
3     public string Company { get; set; }
4 }
```

Mazkur holatda xato chiqariladi, chunki Employee sinfi Person sinfiga mos kelmaydi, aniqrog'i asosiy sinfning konstruktorini chaqirmaydi. Agar biz huddi shu xususiyatlarni o'rnatuvchi qandaydir konstruktorni qo'shsak ham, baribir xatoga duch kelamiz:

```
1 public Employee(string name, string company)
2 {
3     Name = name;
4     Company = company;
5 }
```

YA'ni Employee sinfida **base** kalit so'zi orqali Person sinfining konstruktorini oshkor ravishda chaqirishimiz lozim:

```
1 public Employee(string name, string company)
2     : base(name)
3 {
4     Company = company;
5 }
```

YOki muqobil ravishda asosiy sinfda paramsiz konstruktorni aniqlashimiz mumkin:

```
1 class Person
2 {
3     // sinfning qolgan kodi
4     // kelishuvga binoan konstruktor
5     public Person()
6     {
7         FirstName = "Tom";
8         Console.WriteLine("Paramsiz konstruktorni chaqirish");
9     }
10 }
```

U holda hosilaviy sinfning asosiy sinfning konstruktoriga murojaat bo‘lmagan ixtiyoriy konstruktoriga baribirga ham huddi shu konstruktor chaqirilar edi. Masalan, quyidagi konstruktor

```
1 public Employee(string company)
2 {
3     Company = company;
4 }
```

Umuman olganda quyidagi konstruktorga ekvivalent bo‘lar edi:

```
1 public Employee(string company)
2     :base()
3 {
4     Company = company;
5 }
```

Konstruktorlarni chaqirish tartibi

Sinf konstruktorini chaqirishda birinchi bo‘lib asosiy sinflarning konstruktorlari qayta ishlanadi, shundan keyingina hosilaviy sinf konstruktorlariga o‘tiladi. Misol tariqasida quyidagi sinflarni olaylik:

```
1 class Person
2 {
3     string name;
4     int age;
5
6     public Person(string name)
7     {
8         this.name = name;
9         Console.WriteLine("Person(string name)");
10    }
11    public Person(string name, int age) : this(name)
12    {
13        this.age = age;
14        Console.WriteLine("Person(string name, int age)");
15    }
16 }
17 class Employee : Person
18 {
19     string company;
20
21     public Employee(string name, int age, string company) : base(name, age)
22     {
23         this.company = company;
```

```

24         Console.WriteLine("Employee(string name, int age, string
           company)");
25     }
26 }

```

Employee ob'ektini yaratishda:

```

1 Employee tom = new Employee("Tom", 22, "Microsoft");

```

Quyidagi konsol natijaga ega bo'lamiz:

Person(string name)

Person(string name, int age)

Employee(string name, int age, string company)

Natijada quyidagi amallar ketma-ketligini hosil qilamiz

1. Birinchi bo'lib Employee(string name, int age, string company) konstruktoriga chaqiriladi. U ijro vakolatini Person(string name, int age) —konstruktoriga uzatadi
2. Person(string name, int age) konstruktoriga chaqiriladi, u o'zi bajarilmasdan, ijroni Person(string name) konstruktoriga uzatadi
3. Person(string name) konstruktoriga chaqiriladi, u ijroni System.Object sinf konstruktoriga uzatadi, chunki u Person uchun asosiy sinf hisoblanadi.
4. System.Object.Object() konstruktoriga ijro etiladi, so'ngra ijro Person(string name) konstruktoriga qaytariladi
5. Person(string name) konstruktoriga tanasi bajariladi, so'ng ijro Person(string name, int age) konstruktoriga qaytariladi
6. Person(string name, int age) konstruktoriga tanasi bajariladi, so'ng ijro Employee(string name, int age, string company) konstruktoriga qaytariladi
7. Employee(string name, int age, string company) konstruktoriga tanasi bajariladi. Natijada Employee ob'ekt yaratiladi

Nazorat savollari.

1. Ob'ektga yo'naltirilgan dasturlashda meros tushunchasini qanday izohlay olasiz?
2. Meroslangan sinfdan asosiy sinf a'zolariga kirish tartibini aytib bering.
3. Konstruktorni qanday chaqirish mumkin?
4. Ixtiyoriy sinflarda konstruktordan foydalanishda farqlar bormi?
5. Konstruktorni chaqirish tartibini ayting.

7. INTERFEYSLAR, STRUKTURALAR VA RO‘YXATLAR

Reja:

1. Interfeyslar bilan tanishtirish.
2. Interfeyslarning bir nechta amalga oshirilishi.
3. Interfeyslar va sinf turini o'zgartirish.
4. Nazorat savollari.

Interfeyslarga kirish

Interfeyslar havolali turga mansub bo‘lib, usul va xususiyatlar majmuini belgilaydi, lekin ularni amalga oshirmaydi. So‘ngra bu funksional ushbu interfeyslarni tadbiiq etuvchi sinf fa tuzilmalarni amaliyotga joriy etadi.

Interfeysni aniqlash uchun **interface** kalit so‘zidan foydalaniladi. Odatda, C# interfeys nomi **I** katta harfdan boshlanadi, masalan, IComparable, IEnumerable (venger talqinida), ammo bu majburiy talab emas, ko‘proq dasturlash uslubidir. Masalan IMovable interfeysi:

```
1 interface IMovable
2 {
3     void Move();
4 }
```

Interfeysda usul va xususiyatlar amaliyotga joriy etilmadi, shu jihatdan ular abstrakt(mavhum) sinflarning abstrakt usullariga yaqinlashadi. Bu holatda interfeys ma’lum bir harakatni ifodalovchi Move usulini aniqlaydi. U hech qanday parametr qabul qilmaydi va hech narsa qaytramaydi.

Interfeyslarni e’lon qilishda yana bir jihat: uning barcha a’zolari- usul va xususiyatlar murojaat modifikatorlariga ega emas, lekin umuman olganda murojaat **public**, chunki

interfeysning maqsadi- funksionalni klass tomonidan joriy etish uchun aniqlash. SHuning uchun butun funksional joriy etilish uchun ochiq bo‘lishi kerak.

Umuman olganda interfeyslar quyidagi mohiyatlarni aniqlaydi:

- Usullar
- Xususiyatar
- Indeksatorlar
- Hodisalar

Ammo interfeyslar statik hadlarni, o‘zgaruvchilarni, o‘zgarmaslarni aniqlay olmaydi.

So‘ngra qandaydir sinf yoki tuzilma mazkur interfeysni tadbiq eta oladi:

```
1 // interfeysni sinfda tadbiq etish
2 class Person : IMovable
3 {
4     public void Move()
5     {
6         Console.WriteLine("Odam yurmoqda");
7     }
8 }
9 // tuzilmada interfeysni tadbiq etish
10 struct Car : IMovable
11 {
12     public void Move()
13     {
14         Console.WriteLine("Mashina yurmoqda");
15     }
16 }
```

Interfeysni tadbiq etishda, vorislik kabi, sinf yoki tuzilma nomidan keyin ikki nuqta qo‘yilib, so‘ngra tadbiq etiluvchi interfeyslarning nomi keladi. Bunda sinf barcha usul va xususiyatlarni tadbiq etishi lozim. Bunda, interfeysning barcha usul va xususiyatlari ommaviy bo‘lgani uchun, bu usul va xususiyatlarni sinfda joriy etishda ularga nisbatan public modifikatorini tadbiq etish mumkin xolos. SHuning uchun, agar sinf boshqa modifikatorli, masalan protected usulga ega bo‘lishi kerak bo‘lsa, u holda interfeys bu kabi usulni aniqlash uchun mos kelmaydi.

Agar sinf va tuzilma qandaydir usul yoki xususiyatlarni amaliyotga joriy etmasa, u holda kompilyasiya bosqichida xatoga yuzlanamiz.

Interfeysni dasturda tadbiq etish:

```
1 using System;
2
3 namespace HelloApp
4 {
```

```

5     interface IMovable
6     {
7         void Move();
8     }
9     class Person : IMovable
10    {
11        public void Move()
12        {
13            Console.WriteLine("Odam yurmoqda");
14        }
15    }
16    struct Car : IMovable
17    {
18        public void Move()
19        {
20            Console.WriteLine("Mashina yurmoqda");
21        }
22    }
23    class Program
24    {
25        static void Action(IMovable movable)
26        {
27            movable.Move();
28        }
29        static void Main(string[] args)
30        {
31            Person person = new Person();
32            Car car = new Car();
33            Action(person);
34            Action(car);
35            Console.Read();
36        }
37    }
38 }

```

Mazkur dasturda Action() usul aniqlangan bo‘lib, u parametr sifatida IMovable interfeysining ob‘ektini qabul qiladi. Kodni yozish paytida biz uning qanday ob‘ekt-qandaydir sinf yoki tuzilma bo‘lishi mumkinligini bilmaymiz. Biz ishonch bilan ayta oladigan yagona narsa, bu ob‘ekt albatta Move usulini joriy etadi va biz bu usulni chaqira olamiz.

Boshqa so‘z bilan aytganda, interfeys bu ma’lum bir tur albatta ma’lum bir funksionalni amaliyotga joriy etishi to‘g‘risidagi shartnomadir.

:

Odam yurmoqda

Mashina yurmoqda

Agar sinf interfeysni tadbiq etsa, bu sinf barcha usul va xususiyatlarni amaliyotga joriy etishi kerak. Ammo usullarni abstrakt holatga keltirib joriy etmasdan, bu huquqni hosilaviy sinflarga topshirish mumkin:

```
1 interface IMovable
2 {
3     void Move();
4 }
5 abstract class Person : IMovable
6 {
7     public abstract void Move();
8 }
9 class Driver : Person
10 {
11     public override void Move()
12     {
13         Console.WriteLine("Haydovchi mashinani boshqarmoqda");
14     }
15 }
```

Qayd etish joizki Visual Studio da yangi interfeysni alohida faylda qo‘shish uchun maxsus tashkil etuvchi mavjud. Interfeysni loyihaga qo‘shish uchun loyihaning ustiga sichqonchani o‘ng tugmasiga bosib, paydo bo‘lgan kontekst menyudan **Add-> New Item...** ni tanlab, yangi tashkil etuvchini qo‘shuvchi muloqot oynasida **Interface** bandini tanlash mumkin:

Interfeysni amalga oshirishda asosiy sinfdan vorislikka o‘tgan usul va xususiyatlar ham hisobga olinadi. Masalan:

```
1 interface IAction
2 {
3     void Move();
4 }
5 class BaseAction
```

```

6  {
7      public void Move()
8      {
9          Console.WriteLine("Move in BaseAction");
10     }
11 }
12 class HeroAction : BaseAction, IAction
13 {
14 }

```

Bu erda HeroAction sinfi IAction interfeysini amaliyotga joriy etadi, ammo Move usulini amaliyotga joriy etish uchun BaseAction bazaviy sinfdan vorislikka o'tgan interfeysdan Move usuli qo'llaniladi. SHu tariqa HeroAction sinfi Move usulini amaliyotga joriy etmasligi ham mumkin, chunki bu usul BaseAction bazaviy sinfdan vorislikka o'tgan.

Qayd etish joizki, agar sinf bir vaqtning o'zida boshqa sinfni vorislikka olsa va HeroAction sinfi kabi interfeysni amaliyotga joriy etsa, u holda bazaviy sinfning nomi joriy etiluvchi sinflardan oldin ko'rsatilishi lozim: class HeroAction : BaseAction, IAction

Interfeyslarning to'plamli joriy etilishi

Interfeyslar yana bir muhim funksiyaga ega: C# da, masalan S++ dan farqli o'laroq to'plamli vorislik qo'llab-quvvatlanmaydi, ya'ni biz sinfni faqatgina bitta sinfdan meros qilib olishimiz mumkin. Interfeyslar bu cheklovlarni qisman aylanib o'tishga imkon beradi, chunki C# da sinf bir vaqtning o'zida bir necha interfeysni joriy etishi mumkin. Barcha interfeyslar vergul orqali ko'rsatiladi:

```

1  myClass: myInterface1, myInterface2, myInterface3, ...
2  {
3
4  }

```

Misol ko'raylik

```

1  using System;
2
3  namespace HelloApp
4  {
5      interface IAccount
6      {

```

```

7         int CurrentSum { get; } // Hisobdagi joriy summa
8         void Put(int sum); // Pulni hisob raqamiga qo'yish
9         void Withdraw(int sum); // Hisobdan yechib olish
10    }
11    interface IClient
12    {
13        string Name { get; set; }
14    }
15    class Client : IAccount, IClient
16    {
17        int _sum; // Yig'indini saqlovchi o'zgaruvchi
18        public string Name { get; set; }
19        public Client(string name, int sum)
20        {
21            Name = name;
22            _sum = sum;
23
24
25            public int CurrentSum { get { return _sum; } }
26
27            public void Put(int sum) { _sum += sum; }
28
29            public void Withdraw(int sum)
30            {
31                if (_sum >= sum)
32                {
33                    _sum -=
34                    sum;
35                }
36            }
37    class Program
38    {
39        static void Main(string[] args)
40        {
41            Client client = new Client("Tom", 200);
42            client.Put(30);
43            Console.WriteLine(client.CurrentSum); //230
44            client.Withdraw(100);
45            Console.WriteLine(client.CurrentSum); //130
46            Console.Read();
47        }

```

```
48     }  
49 }
```

Berilgan holda ikkita interfeys aniqlangan. IAccount interfeysi ayni paytda hisob raqamidagi pulni CurrentSum xususiyat va ikkita Put va Withdraw usul pulni hisobga qo'yish va hisobdan olish uchun qo'llaniladi. IClient interfeysi mijoz nomini saqlash uchun xususiyatni belgilaydi.

CurrentSum va Name xususiyatlari interfeyslarda avtoxususiyatlarga o'xshashdir, lekin bu avtoxususiyatlar emas. Amaliyotga joriy etishda biz ularni to'liq xususiyatlarga aylantirishimiz yoki avtoxususiyatlarga aylantirib yuborishimiz mumkin.

Client sinfi ikkala interfeysni amaliyotga joriy etadi so'ngra dasturda tadbiiq etiladi.

Turlarni o'zgartirishdagi interfeyslar

Turlarning o'zgarishiga nisbatan aytilgan barcha mulohazalar interfeyslarga ham xosdir.

Client sinfi IAccount interfeysni amaliyotga joriy etgani bois, IAccount turidagi o'zgaruvchi

Client turidagi ob'ektga havolani saqlashi mumkin:

```
1 // Barcha Client ob'ektlari IAccount ob'ektlari hisoblanadi  
2 IAccount account = new Client("Tom", 200);  
3 account.Put(200);  
  
4 Console.WriteLine(account.CurrentSum); // 400  
5 // IAccount ning hamma ob'ektlari ham Client ob'ektlari  
  hisoblanavermaydi,  
6 Oshkor ravishda keltirish lozim  
7 Client client = (Client)account;  
8 // IAccount interfeysi Name xususiyatga ega emas, oshkor ravishda  
  keltirish lozim string clientName = ((Client)account).Name;
```

Klassdan interfeysga o'tish, hosilaviy turdan asosiy turga o'tish singari avtomatik ravishda amalga oshiriladi. CHunki ixtiyoriy Client ob'ekt IAccount interfeysini amaliyotga joriy etadi.

Teskari almashtirish – interfeyslardan uni amaliyotga joriy etuvchi sinfga o'tish asosiy sinfdan hosilaviy sinfga o'tish kabidir. Har qanday

IAccount ob'ekt Client ning ob'ekti bo'lmagani bois (chunki IAccount interfeysini boshqa klasslarni ham joriy etishi mumkin), bu kabi almashtirish uchun turlarni keltirish amali zarur bo'ladi. Va agar biz Iaccount interfeysida aniqlanmagan, ammo Client sinfining bir qismi bo'lgan Client sinf usullariga murojaat etish niyatida bo'lsak, u holda turlarni oshkor ravishda almashtirishga majbur bo'lamiz: string clientName = ((Client)account).Name;

Ro'yxat List<T>

List<T> sinfi bir turga mansub ob'ektlar ro'yxatini ifodalaydi.

Uning usullari ichidan quyidagilarni ajratib ko'rsatish mumkin:

void Add(T item): yangi elementni ro'yxatga qo'shish

- **void AddRange(ICollection collection):** ro'yxatga massiv yoki to'plamni qo'shish
- **int BinarySearch(T item):** ro'yxatda elementni binar izlash. Agar element topilgan bo'lsa, u holda usul bu elementning to'plamdagi indeksini qaytaradi. Bunda ro'yxat saralangan bo'lishi kerak.
- **int IndexOf(T item):** elementning ro'yxatga birinchi marotaba kirgan indeksini qaytaradi
- **void Insert(int index, T item):** ro'yxatdagi item elementni index o'ringa joylashtiradi
- **bool Remove(T item):** item elementni ro'yxatdan o'chiradi, agar o'chirish muvaffaqiyatli o'tgan bo'lsa, true qiymat qaytariladi.
- **void RemoveAt(int index):** elementni ko'rsatilgan index bo'yicha o'chirish
- **void Sort():** ro'yxatni saralash

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace Collections
5 {
6     class Program
7     {
8         static void Main(string[] args)
```

```

9      {
10         List<int> numbers = new List<int>() { 1, 2, 3, 45 };
11         numbers.Add(6); // elementni qo'shish
12
13         numbers.AddRange(new int[] { 7, 8, 9 });
14
15         numbers.Insert(0, 666); // ro'yxatdagi birinchi o'ringa
16 666 sonini joylashtiramiz
17
18         numbers.RemoveAt(1); // ikkinchi elementni o'chiramiz
19
20         foreach (int i in numbers)
21         {
22             Console.WriteLine(i);
23         }
24
25         List<Person> people = new List<Person>(3);
26         people.Add(new Person() { Name = "Tom" });
27         people.Add(new Person() { Name = "Bill" });
28
29         foreach (Person p in people)
30         {
31             Console.WriteLine(p.Name);
32         }
33
34         Console.ReadLine();
35     }
36 }
37
38 class Person
39 {
40     public string Name { get; set; }
41 }
    
```

Bu yerda bizda ikkita ro'yxat yaratiladi: biri int turidagi ob'ektlar uchun, boshqasi esa

Person ob'ektlari uchun. Birinchi holatda biz ro'yxatning boshlang'ich initsializatsiyasini

amalgga oshiramiz: List<int> numbers = new List<int>() { 1, 2, 3, 45 };

Ikkinchi holatda biz boshqa konstruktordan foydalanamiz, unga ro'yxatning boshlang'is sig'imini uzatamiz: List<Person> people = new List<Person>(3);. Ro'yxatning boshlang'ich sig'imini ko'rsatish (capacity) kelgusida

mahsuldorlikni oshirishga va elementni qo'shish jarayonida ajratiladigan xotira sarfini kamaytirishga imkon beradi. SHuningdek boshlang'ich sig'imni Capacity xususiyat yordamida o'rnatish mumkin, u List sinfida keltirilgan.

Ikki tomonlama bog'langan ro'yxat LinkedList<T>

LinkedList<T> sinfi ikki tomonlama bog'langan ro'yxat bo'lib, unda har bir element bir vaqtning o'zida ham keyingi ham avvalgi elementga havolani saqlaydi.

Agar oddiy List<T> ro'yxatda har bir element T turidagi ob'ekt bo'lsa, u holda LinkedList<T>da har bir tugun LinkedListNode<T> sinfining ob'ektini ifodalaydi.

Bu sinf quyidagi xususiyatlarga ega:

- **Value**: T turi orqali ko'rsatilgan tugun qiymati
- **Next**:ro'yxatdagi navbatdagi LinkedListNode<T> elementga havola. Agar keyingi element mavjud bo'lmasa, u holda null qiymat beriladi.
- **Previous**: ro'yxatdagi LinkedListNode<T> turidagi avvalgi elementga havola. Agar avvalgi element mavjud bo'lmasa, u holda null qiymat beriladi.

LinkedList<T> sinf usullaridan foydalanib, turli xil elementlariga ham boshida, ham oxirida murojaat etish mumkin:

- **AddAfter(LinkedListNode<T> node, LinkedListNode<T> newNode):** newNode tugunini ro'yxatga node tugunidan so'ng qo'shamiz.
- **AddAfter(LinkedListNode<T> node, T value):** ro'yxatga node tugunidan so'ng value qiymatli yangi tugunni qo'shadi.
- **AddBefore(LinkedListNode<T> node, LinkedListNode<T> newNode):** ro'yxatga node tugunidan avval newNode tugunini qo'shadi.
- **AddBefore(LinkedListNode<T> node, T value):** node tugunidan avval value qiymatli yangi tugunni ro'yxatga qo'shadi.
- **AddFirst(LinkedListNode<T> node):** yangi tugunni ro'yxat boshiga qo'shadi

AddFirst(T value): value qiymatli yangi tugunni ro'yxat boshiga qo'shadi

□ **AddLast(LinkedListNode<T> node):** yangi tugunni ro'yxat oxiriga qo'shadi

□ **AddLast(T value):** value qiymatli yangi tugunni ro'yxat oxiriga qo'shadi

RemoveFirst(): birinchi tugunni ro'yxatdan o'chiradi. So'ngra yangi birinchi tugun o'chirilgandan keyingi tugunga o'tadi

□ **RemoveLast():** ro'yxatdan oxirgi tugunni o'chiradi udalyaet posledniy uzel iz spiska

Ikki tomonlama ro'yxatni amalda sinab ko'raylik:

```
1 using System;
2 using System.Collections.Generic;
3
4 namespace Collections
5 {
6     class Program
7     {
8         static void Main(string[] args)
9         {
10             LinkedList<int> numbers = new LinkedList<int>();
11
12             numbers.AddLast(1); // 1 qiymatli tugunni oxirgi o'ringa qo'yamiz
13             // ro'yxatda tugun bo'lmagani uchun, oxirgisi birinchi bo'lib
14             // qoladi
15
16             numbers.AddFirst(2); // 2 qiymatli tugunni birinchi o'ringa
17             // joylashtiramiz
18             numbers.AddAfter(numbers.Last, 3); // oxirgi tugundan
19             // so'ng 3
20             // qiymatli yangi tugunni joylashtiramiz
21             // endi ro'yxatimiz quyidagi ketma-ketlikdan iborat bo'ladi: 2, 1, 3
22             foreach (int i in numbers)
23             {
24                 Console.WriteLine(i);
25             }
26
27             LinkedList<Person> persons = new LinkedList<Person>();
28
29             // persona ni ro'yxatga qo'shamiz va LinkedListNode<Person>
30             // ob'ektni hosil qilamiz, unda Tom ismi saqlanadi
```

```

        LinkedListNode<Person> tom = persons.AddLast(new Person() {
28     Name = "T
29     persons.AddLast(new Person() { Name = "John" });
30     persons.AddFirst(new Person() { Name = "Bill" });
31
32     Console.WriteLine(tom.Previous.Value.Name); //
33 Tomdan avvalgi tugunni va uning qiymatini qabul qilamiz
34     Console.WriteLine(tom.Next.Value.Name); //
35 Tomdan keyingi tugunni va uning qiymatini qabul qilamiz
36
37
38     }
39 }
40
    class Person
    {
        public string Name { get; set; }
    }
}

```

Bu erda ikkita ro'yxat yaratiladi va foydalaniladi: sonlar uchun va Person sinf ob'ektlari uchun. Sonlar borasida hammasi tushunarli bo'lsa kerak. Person sinfi bilan ishlarni o'rganaylik.

Ro'yxatga qo'shishda joylashtirish usullari (AddLast, AddFirst) qo'shilgan elementga havolani joylashtiradi LinkedListNode<T> (bizning holimizda LinkedListNode<Person>). So'ngra Previous va Next xususiyatlarni boshqarib, biz ro'yxatdagi avvalgi va keyingi tugunlarga havolalarni qabul qilishimiz mumkin.

Nazorat savollari.

1. Interfeyslarni qanday ob'ektlar belgilashi mumkin?
2. Interfeyslarni bir nechta amalga oshirish variantlarini bering.
3. Interfeysni konvertatsiya qilish qanday amalga oshiriladi?
4. Bir xil turdagi ob'ektlar ro'yxati yordamida interfeysning o'zgarishini tavsiflang.
5. Ikki marta bog'langan ro'yxat nima?

8. KOLLEKSIYA VA ITERATORLAR

Reja:

1. To'plamlar bilan tanishtirish.
2. IEnumerable va IEnumerator interfeyslari.
3. IEnumerable va IEnumeratorni amalga oshirish.
4. Nazorat savollari.

To'plamlarga kirish

C# tilida bir turdagi ob'ektlarni saqlovchi massivlar mavjud bo'lsada, ular bilan ishlash har doim ham qulay bo'lavermaydi. Masalan, massiv belgilangan sondagi ob'ektlarni saqladi, ammo, bizga qancha ob'ekt kerak bo'lishi oldindan ma'lum bo'lmaschi. Bunday holda bizga kolleksiyalardan foydalanish ancha qulayroqdir. Kolleksiyalarning yana bir afzalligi shundaki, ularning ayrimlari standart ma'lumotlar tuzilmasini, masalan, stek, navbat, lug'atni amaliyotga joriy etadi, ular turli xil maxsus masalalarni yechish uchun qo'l keladi.

Kolleksiya sinflarining katta qismi System.Collections nomlar fazosida (oddiy umumlashmagan kolleksiya sinflari), System.Collections.Generic (umumlashgan yoki bir turdagi kolleksiya sinflari) va System.Collections.Specialized (maxsus kolleksiya sinflari) da saqlanadi. SHuningdek, masalalarning parallel ravishda bajarilishi va ko'p oqimli murojatni ta'minlash uchun System.Collections.Concurrent nomlar fazosidan kolleksiya sinflari tadbiiq etiladi.

Barcha kolleksiyalarni yaratish uchun asos

bo'lib **IEnumerator** va **IEnumerable** interfeyslarni joriy etish hisoblanadi (va ularning umumlashgan egizaklarini **IEnumerator<T>** va **IEnumerable<T>**). IEnumerator interfeysi kolleksiyalarni **foreach** sikli yordamida ketma-ket saralovchi hisobchidir. IEnumerable interfeysi esa GetEnumerator usuli yordamida mazkur interfeysni amaliyotga joriy etuvchi sinflarga bu hisobchini taqdim etadi. SHuning uchun

IEnumerable (IEnumerable<T>) interfeysi barcha kolleksiyalar uchun asos hisoblanadi.

Ikkita kolleksiyaning yaratilishi va tadbiiq etilishini ko'raylik:

```

1  using System;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  namespace Collections
6  {
7      class Program
8      {
9          static void Main(string[] args)
10         {
11             // umumlashmagan
12             ArrayList kolleksiyai
13             ArrayList objectList = new ArrayList() { 1, 2, "string",
14             'c', 2.0f };
15
16             object obj = 45.8;
17
18             objectList.Add(obj);
19             objectList.Add("string2");
20             objectList.RemoveAt(0); // birinchi elementni o'chirish
21             foreach (object o in objectList)
22             {
23                 Console.WriteLine(o);
24             }
25             Console.WriteLine("Kolleksiyadagi elementlarning umumiy
26             soni:
27             {0}", objectList.Count);
28             // umumlashgan List kolleksiyasi
29             List<string> countries = new List<string>() { "Rossiya",
30             "AQSH", "Buyuk Britaniya", "Xitoy" };
31             countries.Add("Fransiya");
32             countries.RemoveAt(1); // ikkinchi elementni o'chirish
33             foreach (string s in countries)
34             {
35                 Console.WriteLine(s);
36             }
37             Console.ReadLine();
38         }
39     }
40 }

```

Bu yerda ikkita kolleksiyadan foydalaniladi: umumlashmagan - ArrayList va umumlashgan - List. Ko'pgina kolleksiyalar elementlarning qo'shilishini qo'llab-quvvatlaydi. Masalan, mazkur holatda qo'shish Add usuli yordamida amalga oshiriladi, ammo boshqa kolleksiyalar uchun usulning nomlanishi farq qilishi mumkin. SHuningdek, ko'pgina kolleksiyalar o'chirishni amalga oshiradi (mazkur misolda RemoveAt usuli yordamida amalga oshiriladi).

Count kolleksiyaning elementlar sonini ko'rish mumkin.

Kolleksiyalar IEnumerable/IEnumerable<T> interfeysini amaliyotga joriy etgani uchun, ularning barchasi foreach siklidagi saralashni qo'llab-quvvatlaydi.

Tadbiq etishning aniq usullari kolleksiyaning bir sinfida boshqasidan farq qilishi mumkin, ammo umumiy tamoyillar kolleksiyaning barcha sinflariga nisbatan bir xil bo'ladi.

IEnumerable va IEnumerator interfeyslari

Ko'rib turganimizdek, ko'pgina kolleksiyalar uchun asos bo'lib IEnumerable va Ienumerator interfeyslarning amaliyotga joriy etilishi hisoblanadi. Bunday joriy etilish hisobiga biz foreach siklidagi ob'ektlarni saralashimiz mumkin:

```
1  foreach(var item in sanaluvchi ob'ekt)
2  {
3
4  }
```

Saralanuvchi kolleksiya IEnumerable interfeysini amaliyotga joriy etishi lozim.

IEnumerable interfeysi boshqa saralovchi-interfeysga havolani qaytaruvchi usulga ega :

```
1  public interface IEnumerable
2  {
3      IEnumerator GetEnumerator();
4  }
```

Ienumerator interfeysi esa kontenerdagi ichki ob'ektlarni saralash uchun funksionalni aniqlaydi:

```
1  public interface IEnumerator
2  {
```

```

3     bool MoveNext(); // elementlar konteynerida bitta o‘ringa ko‘chish
4     object Current {get;} // konteynerdagi joriy element
5     void Reset(); // konteynerning boshiga ko‘chish
6 }

```

MoveNext() usul elementga ko‘rsatkichni ketma-ketlikdagi keyingi o‘ringa ko‘chiradi. Agar ketma-ketlik hali tugamagan bo‘lsa, u holda true qiymat qaytaradi. Agar ketma-ketlik tugagan bo‘lsa, u holda false qiymat qaytariladi.

Current xususiyat ko‘rsatkich ishora qilgan ketma-ketlikdagi ob’ektni qaytaradi.

Reset() usuli o‘rin ko‘rsatkichini boshlang‘ich holatga qaytaradi.

Ko‘rsatkich qay tarzda ko‘chirilishi va elementlarning qanday tartibda olinishi interfeysning amaliyotga joriy etilishiga bog‘liq. Har xil amaliyotlarda mantaq har xil tarzda qurilishi mumkin.

Masalan, foreach siklidan foydalanmasdan Ienumerator interfeysini kolleksiya yordamida saralaymiz:

```

1     using System;
2     using System.Collections;
3
4     namespace HelloApp
5     {
6         class Program
7         {
8             static void Main(string[] args)
9             {
10                int[] numbers = { 0, 2, 4, 6, 8, 10 };
11
12                IEnumerator ie = numbers.GetEnumerator(); // false qaytarilgunga
13                qadar
14
15                while (ie.MoveNext()) // IEnumerator ni olamiz
16                {
17                    int item = (int)ie.Current; // joriy o‘rindagi elementni olamiz
18                    Console.WriteLine(item);
19                }
20                ie.Reset(); // ko‘rsatkichni massiv boshiga o‘tkazamiz
21                Console.Read();
22            }

```

```

    }
}

```

IEnumerable va IEnumerator ni amaliyotga joriy etish
IEnumerable ning eng sodda realizatsiyasini misolda ko‘raylik:

```

1  using System;
2  using System.Collections;
3
4  namespace HelloApp
5  {
6      class Week : IEnumerable
7      {
8          string[] days = { "Monday", "Tuesday", "Wednesday", "Thursday",
9                          "Friday", "Saturday", "Sunday" };
10
11         public IEnumerator GetEnumerator()
12         {
13             return days.GetEnumerator();
14         }
15     }
16     class Program
17     {
18         static void Main(string[] args)
19         {
20             Week week = new Week();
21             foreach(var day in week)
22             {
23                 Console.WriteLine(day)
24             };
25             Console.Read();
26         }
27     }
28 }

```

Mazkur holatda Week sinfi haftani ifodalab, barcha hafta kunlarini saqlaydi va IEnumerable interfeysni amaliyotga joriy etadi. Ammo mazkur holatda biz juda ham oson yo‘l tutdik-IEnumerator ni amaliyotga joriy etish o‘rniga biz shunchaki GetEnumerator usulida massiv uchun IEnumerator ob’ektni qaytaramiz.

```

1  public IEnumerator GetEnumerator()
2  {
3      return days.GetEnumerator();
4  }

```


SHunga muvofiq biz foreach siklida barcha hafta kunlarini saralab chiqishimiz mumkin.

Ayni paytda, shuni alohida qayd etish joizki, kolleksiyani foreach orqali saralash uchun

IEnumerable interfeysini amaliyotga joriy etish shart emas. Sinfda Ienumerator ob'ektini qaytaruvchi **GetEnumerator** ochiq usulni e'lon qilish etarlidir.

Masalan:

```
1 class Week
2 {
3     string[] days = { "Monday", "Tuesday", "Wednesday", "Thursday",
4                       "Friday", "Saturday", "Sunday" };
5
6     public IEnumerator GetEnumerator()
7     {
8         return days.GetEnumerator();
9     }
10 }
```

Ammo bu juda oson edi- biz shunchaki tayyor massiv saralagichidan foydalanamiz xolos. Ammo, ob'ektlarni saralashning xususiy mantiqini berish kerakdir. Buning

uchun **IEnumerator** interfeysini amaliyotga joriy etamiz:

```
1 using System;
2 using System.Collections;
3
4 namespace HelloApp
5 {
6     class WeekEnumerator : IEnumerator
7     {
8         string[] days;
9         int position = -1;
10        public WeekEnumerator(string[] days)
11        {
12            this.days = days;
13        }
14        public object Current
```

```

15     {
16         get
17         {
18             if (position == -1 || position >= days.Length)
19                 throw new InvalidOperationException();
20             return days[position];
21         }
22     }
23
24     public bool MoveNext()
25     {
26         if(position < days.Length - 1)
27         {
28             position++;
29             return true;
30         }
31         else
32             return false;
33     }
34
35     public void Reset()
36     {
37         position = -1;
38     }
39 }
40 class Week
41 {
42     string[] days = { "Monday", "Tuesday", "Wednesday", "Thursday",
43                     "Friday", "Saturday", "Sunday" };
44
45     public IEnumerator GetEnumerator()
46     {
47         return new WeekEnumerator(days);
48     }
49 }
50 class Program
51 {
52     static void Main(string[] args)
53     {
54         Week week = new Week();
55         foreach(var day in week)
56         {
57             Console.WriteLine(day)
58             ;
59         }

```

```

59         Console.Read();
60     }
61 }
62 }

```

Bu yerda endilikda Week sinfi biriktirilgan saralagichdan emas, WeekEnumerator, dan foydalanadi va u IEnumerator ni amaliyotga joriy etadi.

Saralagichni amaliyotga joriy etishdagi asosiy jihat- ko'rsatkichni elementga ko'chirishdir. WeekEnumerator sinfida joriy o'rinni saqlash uchun maxsus position o'zgaruvchisi belgilangan. Qayd etish joizki, eng boshida (boshlang'ich holatda) ko'rsatkich birinchi elementdan oldingi o'ringa ishora qilishi kerak. Eng avvalida (boshlang'ich holatda) ko'rsatkich birinchi elementdan avvalgi o'ringa ishora qilishi kerak. foreach sikli amalga oshiriladi, bu sikl avvalambor MoveNext ni chaqiradi va umuman olganda ko'rsatkichni bitta pog'ona oldinga ko'chiradi shundan keyingina joriy o'rindagi elementni olish uchun Current xususiyatga o'tadi.

YUqoridagi misollarda interfeyslarning umumlashmagan versiyalaridan foydalanildi ammo biz shuningdek ularning umumlashgan egizaklaridan ham foydalanishimiz mumkin:

```

using System;
using System.Collections;
using System.Collections.Generic;

namespace HelloApp
{
    class Week
    {
        string[] days = { "Monday", "Tuesday", "Wednesday", "Thursday",
                          "Friday", "Saturday", "Sunday" };

        public IEnumerator<string> GetEnumerator()
        {
            return new WeekEnumerator(days);
        }
    }

    class WeekEnumerator : IEnumerator<string>

```

```

{
    string[] days;
    int position = -1;
    public WeekEnumerator(string[] days)
    {
        this.days = days;
    }

    public string Current
    {
        get
        {
            if (position == -1 || position >= days.Length)
                throw new InvalidOperationException();
            return days[position];
        }
    }

    object IEnumerator.Current => throw new NotImplementedException();

    public bool MoveNext()
    {
        if(position < days.Length - 1)
        {
            position++;
            return true;
        }
        else
            return false;
    }

    public void Reset()
    {
        position = -1;
    }
    public void Dispose() { }
}
class Program
{

```

```

static void Main(string[] args)
{
    Week week = new Week();
    foreach(var day in week)
    {
        Console.WriteLine(day);
    }
    Console.Read();
}
}
}

```

Iteratorlar va Yield operatori

□

Iterator umuman olganda qiymatlar kolleksiyaini saralash uchun **yield** operatoridan foydalanuvchi kod blokini ifodalaydi. Berilgan blok usul, operator tanasini va **get** blokini ifodalashi mumkin.

Iterator ikkita maxsus yo‘riqdan foydalanadi:

- **yield return**: qaytariluvchi elementni aniqlaydi
- **yield break**: ketma-ketlik boshqa elementlarga ega emasligiga ishora qiladi

Kichik misol ko‘raylik:

```

1  using System;
2  using System.Collections;
3
4  namespace HelloApp
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             Numbers numbers = new
11             Numbers();
12             foreach (int n in numbers)
13             {
14                 Console.WriteLine(n);
15             }
16             Console.ReadKey();
17         }
18     }
19 }

```

```

17     }
18
19     class Numbers
20     {
21         public IEnumerator GetEnumerator()
22         {
23             for(int i = 0; i < 6; i++)
24             {
25                 yield return i * i;
26             }
27         }
28     }
29 }

```

Numbers sinfida **GetEnumerator()** usul shartli ravishda iteratorni ifodalaydi. **yield operatori yordamida return** ma'lum bir qiymatni qaytaradi (bizning holimizda sonning kvadrati).

Dasturda foreach sikli yordamida Numbers ob'ektini oddiy kolleksiya sifatida saralab chiqishimiz mumkin. Har bir elementni foreach siklida qabul qilganda yield return operatori ishga tushadi u bitta elementni qaytarib, joriy pozitsiyani xotirada saqlab qoladi.

Boshqa misol: bizda Library kolleksiya mavjud bo'lib u kitoblar omborxonasini ifodalasin. Bu tuplamni saralash uchun yield operatoridan foydalanamiz:

```

1     class Book
2     {
8         public Book(string name)
2         {
9             this.Name = name;
3         }
0         public string Name { get; set; }
3     }
1
3
2     class Library
    {
        private Book[] books;

        public Library()

```

```

{ } { new Book("Otalar va farzandlar"), new Book("Urush va
tipchlik"),
o         new Book("Evgeniy Onegin") };
o     }
k
s     public int Length
=     {
n         get { return books.Length; }
e     }
w     public IEnumerator GetEnumerator()
B     {
o         for (int i = 0; i < books.Length; i++)
o         {
k             yield return books[i];
[         }
    }
}

```

GetEnumerator() usuli iteratorni ifodalaydi. Va biz foreach siklida Library operatorini saralaydigan bo'lsak, u holda yield return books[i] chaqiriladi;. yield return operatoriga murojaat qilganda joriy o'rin saqlanib qoladi. foreach usuli yangi ob'ektni olish uchun navbatdagi iteratsiyaga o'tganda iterator ijroni shu joydan boshlaydi.

Va asosiy dasturda foreach siklida iteratorning amalga oshirilishi hisobiga saralash amalga oshiriladi;

```

1  foreach (Book b in library)
2  {
3      Console.WriteLine(b.Name);
4  }

```

GetEnumerator() usuli iteratorni amaliyotga joriy etishda for siklida massiv saralangan bo‘lsada, buni amalga oshirish majburiy emas. Biz yield return operatorini bir necha marotaba chaqirishimiz mumkin xolos:

```

1  IEnumerator IEnumerable.GetEnumerator()
2  {
3      yield return books[0];
4      yield return books[1];
5      yield return books[2];
6  }

```

Bu holatda yield return operator hal gal chaqirilganda iterator shuningdek joriy o‘rinni xotiraga saqlab qo‘yadi va keyingi chaqirishlarda ulardan boshlaydi.

Nomlangan iterator

YUqorida iteratorni yaratish uchun biz GetEnumerator usulidan foydalandik. Ammo yield operatoridan ixtiyoriy usul ichida foydalanish mumkin, lekin bu usul IEnumerable interfeysining ob’ektini qaytarishi shart. Bu kabi usul nomlangan iterator deb ham ataladi.

Bunday nomlangan iteratorni Library sinfida yaratamiz va undan foydalanamiz:

```

1  class Book
2  {
3      public Book(string name)
4      {
5          this.Name=name;
6      }
7      public string Name { get; set; }
8  }
9
10 class Library
11 {
12     private Book[] books;
13 public Library()
14     {
15         books = new Book[] { new Book("Otalar va farzandlar"),
16 new Book("Urush va tinchlik"),
17         new Book("Evgeniy Onegin") };
18     }
19 }
20
21     public int Length
22     {

```



```

23         get { return books.Length; }
24     }
25
26     public IEnumerable GetBooks(int max)
27     {
28         for (int i = 0; i < max; i++)
29         {
30             if (i == books.Length)
31             {
32                 yield break;
33             }
34             else
35             {
36                 yield return books[i];
37             }
38         }
39     }

```

Bu erda aniqlangan iterator- IEnumerable GetBooks(int max) usuli parametr sifatida chiqariluvchi ob'ektlar sonini qabul qiladi. Dastur ishlashi davomida uning qiymati books massivining uzunligidan katta bo'lib qolishi mumkin. Xato vujudga kelmasligi uchun, **yield break** operatoridan foydalaniladi. Bu operator iteratorning ijro etilishini to'xtatadi.

Iteratorni tadbiq etish:

```

1  Library library = new Library();
2
3  foreach (Book b in library.GetBooks(5))
4  {
5      Console.WriteLine(b.Name);
6  }

```

library.GetBooks(5) ni chaqirish 5 tadan ko'p bo'lmagan Book ob'ektlaridan tashkil topadi. Ammo bizda bunday ob'ektlardan atigi uchta bo'lgani uchun, GetBooks usulida uchta amalidan so'ng yield break operator ishga tushib ketadi.

Nazorat savollari.

1. System.Collection nom maydonlarida joylashgan sinflar to'plamini tavsiflang.
2. Ikkita to'plamni yaratish va ishlatish tartibini keltiring va tavsiflang.
3. Klass elementlari soni qanday aniqlanadi?
4. IEnumerable interfeysini tavsiflang
5. IEnumerator interfeysini tavsiflang.

Xulosa

Dasturiy ta'minot injiniringi fani dasturiy injiniring sintaksisini o'rgatadi va unda amaliy dasturlar yaratish, ixtiyoriy murakkablikdagi dasturiy ta'minot yaratish, zamonaviy taqsimlangan dasturiy ta'minot va ma'lumotlar bazasini boshqarish tizimlarini yaratish uchun zarur platformalarda ishlash. Dasturiy ta'minot muhandisligi bo'yicha o'quv qo'llanmasining birinchi qismi sinfga kirishni boshqarish, rekursiya, inkapsulyatsiya va meros, interfeyslar va to'plamlarni yaratishni o'z ichiga oladi.

Bundan tashqari, ushbu kurs doirasida talabalar Entity Framework, MVC 5 va ulardagi loyihalar bilan ishlash uchun dasturlash texnologiyalarini o'rganadilar va ulardan amaliy ishlarda, ilmiy tadqiqotlarda hamda ta'lim tizimida samarali foydalanish imkoniyatiga ega bo'ladilar.

Foydalanilgan adabiyotlar

1. Roger Pressman, Bruce Maxim, Software Engineering: A Practitioner's Approach, John Wiley & Sons, USA 2014.
2. Ian Sommerville. Software Engineering Hardcover. Pearson 2010 USA
3. Robert W. Sebesta, Concepts of Programming Languages, John Wiley & Sons, USA 2015.
4. Fundamentals of Computer Programming With C# (The Bulgarian C# Programming Book). Svetlin Nakov & Co., 2013.
5. SHildt, Gerbert. S# 4.0: polnoe rukovodstvo. : Per. s angl. — M. : OOO "I.D. Vilyams", 2011. - 1056 s.
6. Troelsen i Djepiks. YAзык программирования C# 7 i платформы .NET i .NET Core. Vilyams 2018. - 1328s.
7. Aleksey Vasilev: Programmirovaniye na C# dlya nachinayuyshix. Osnovnyye svedeniya. -M.: Bombora., 2018. – 592s.
8. Aleks, D. Asinxronnoe programmirovaniye v C# 5.0 / D. Aleks. - M.: DMK, 2015. - 120 c.
9. Borovskiy, A.N. Qt4.7+. Prakticheskoe programmirovaniye na C++. / A.N. Borovskiy. - SPb.: BHV, 2012. - 496 c.
10. Brey, B. Primeneniye mikrokontrollerov PIC 18. Arxitektura, programmirovaniye i postroeniye interfeysov s primeneniem S i assemblera / B. Brey. - SPb.: KORONA-Vek, 2014. - 576 c.
11. Byarne, Straustrup Programmirovaniye: prinsipy i praktika s ispolzovaniem C++ / Straustrup Byarne. - M.: Vilyams, 2016. - 1328 c.
12. Vagner, B. C#. Effektivnoe programmirovaniye. 50 rekomendatsiy po usovershenstvovaniyu programmirovaniya na C# / B. Vagner; Per. s angl. M. Gorelik. - M.: Lori, 2013. - 256 c.



**O'ZBEKISTON RESPUBLIKASI
OLIIY VA O'RTA MAXSUS
TA'LIM VAZIRLIGI**

GUVOHNOMA



**O'QUV ADABIYOTINING
NASHR RUXSATNOMASI**

O'zbekiston Respublikasi Oliy va o'rtta maxsus ta'lim vazirligining 20 22 yil " 9 " sentabr dagi " 302 " -sonli buyrug'iga asosan

D.T. Muzqammardiyeva L.P. Varlamova, S.A. Baxromova
(muallifning familiyasi, ismi-sharifi)
5330100-Axborot tizimlarining matematik va dasturiy ta'minoti

(ta'lim yo'nalishi (mutaxassisligi))

ning

talabari (o'quvchilari) uchun tavsiya etilgan
Dasturiy injining C#-dasturlashi asoslari

(o'quv adabiyotining nomi va turi, darajalik, o'quv qo'llanma)

nomli o'quv qo'llanmasi

ga

O'zbekiston Respublikasi Vazirlar Mahkamasi tomonidan litsenziya berilgan nashriyotlarda nashr etishga ruxsat berildi.



Vazir  A. Toshkulov
(imzo)

Ro'yxatga olish raqami

302-0383

