C#sharp (beginer + basics)

www.albdarb.com
Ալբերտ Ս. Դարբինյան
Կիրառական մաթեմատիկայի ֆակուլտետ ,Պոլիտեխնիկ
Microsoft IT Academy
Ինֆորմատիկա և Հաշվողական Տեխնիկա, ԳՊՀ

Հարցաշար

Ալգորիթմական C#sharp (Algorithmic C#)
1. Ծրագրի հրամանների կատարման հաջորդականությունը [albdarb] (էջ 5)
2. Թարգմանիչներ: Console Application – (թվային մուտք) [albdarb] (էջ 6)
3. Ներդրված տիպեր [Шилдт-68; Нейгел-78] (էջ 6)
4. Օպերացիաներ և օպերատորներ [Шилдт-97] (էջ 8)
   Վերագրման օպերատոր [Шилдт- 106]
   Թվաբանական օպերացիաներ (+, -, *, /, %) [Шилдт- 97]
   Համեմատման օպերացիաներ (< ,>,==,!=, <=, >=) [Шилдт-101]
   Տրամաբանական օպերացիաներ (&&, ||, !) [Шилдт-101]
   Մեծացման և փոքրացման օպերացիաներ, (++, --) [Шилдт- 98]
5. Պայմանական օպերատոր if, if….else [Шилдт-121] (էջ 10)
6. Պայմանական օպերատոր (?) [Шилдт- 117] (էջ 10)
7. Ցիկլիկ օպերատորներ - while, do… while, for [Шилдт-129] (էջ 11)
8. Անցումային օպերատորներ – break, continue, goto [Шилдт-139] (էջ 13)
9. switch օպերատոր [Шилдт-125] (էջ 14)
10. Բիթային օպերացիաներ (|, &, ~, ^, <<, >>) [Шилдт-107] (էջ 16)
11. Տիպերի վերափոխարկման օպերատոր ( ) [Шилдт-89] (էջ 17)
12. C/C# պարզ տարբերություններ [Шилдт-97, 121] (էջ 17)
13. Զանգված [Шилдт -139; Нейгел- 184] (էջ 18)
   Պրակտիկ խնդիրներ մատրիցաներից [albdarb]

Ֆունկցիոնալ ծրագրավորում (Functional programming)
14. Ծրագրավորման լեզուների զարգացման էտապները [albdarb] (էջ 23)
15. Մեթոդներ [Шилдт-155] (էջ 24)
   մեթոդի պարամետրեր, արգումենտներ [Шилдт-162]
   մեթոդի վերադարձվող արժեք, return օպերատոր [Шилдт-158]
16. Գերբեռնված մեթոդներ [Шилдт-235] (էջ 27)
17. Ռեկուրսիա [Шилдт-257] (էջ 28)
18. Լոկալ փոփոխականներ, բլոկ { } [albdarb] (էջ 30)
19. Մեթոդ և ստեկ հիշողություն [albdarb] (էջ 31)
   Տնային ալգորիթմական շախմատից [albdarb]

C#sharp հիմունքներ (C # Basics)
20. Կլասներ, օբյեկտներ [Шилдт-147; Троелсен-185] (էջ 32)
21. Թույլատրության մոդիֆիկատորներ [Троелсен-207; Шилдт-210] (էջ 33)
22. Կոնստուկտոր, լռելիությամբ կոնստրուկտոր [Шилдт-166; Троелсен-188] (էջ 34)
23. Պարամետրով և գերբեռնված կոնստրուկտորներ [Шилдт-168] (էջ 34)

24. Ինիցալիզատոր, կոմպոզիցիա, ագրեգացիա [albdarb] (էջ 35)
25. Անունների տարածք, Պսեվդոանուն [Шилдт-513; Троелсен-494,497; Нейгел-92,94] (էջ 36)
26. Անունի տարածք և global:: [Шилдт-524; Уотсон-369] (էջ 37)
27. Չափային և հղիչային տիպեր [Нейгел-371; Рихтер-129; Троелсен-174] (էջ 38)
28. boxing, unboxing [Рихтер-136; Шилдт-370; Нейгел-217] (էջ 39)
29. null, ?, ?? [Шилдт-695; Нейгел-211] (էջ 40)
30. Ստրուկտուրաների կազմակերպումը [Нейгел-130; Шилдт-391; Троелсен-172] (էջ 41)
Օրինակ. Լոկալ տվյալների կուտակիչ [albdarb]
31. new չափային տիպերի համար, default [Троелсен-120; Шилдт -170] (էջ 45)
32. Կոպիայի կոնստրուկտոր [Либерти-97] (էջ 45)
33. Կլասի ստատիկ անդամներ [Троелсен-197; Шилдт-260] (էջ 46)
34. Ստատիկ կոնստրուկտոր, Ստատիկ կլաս [Троелсен-201; Нейгел-126; Шилдт-265; Рихтер-172] (էջ 47)
35. Փակ կոնստրուկտոր, Սինգլտոն փաթերն [Либерти-101] [Нэш-96] (էջ 48)
36. this հղում [Троелсен-191; Шилдт -174] (էջ 50)
37. this որպես մեթոդի արգումենտ [Либерти-98] (էջ 51)
38. Կոնստրուկտոր և this [Троелсен-193; Нейгел-128; Шилдт-245] (էջ 51)
39. const հաստատուն, readonly հաստատուն [Рихтер-191; Троелсен-226; Нейгел-129; Нэш-63] (էջ 52)
40. enum հաստատուն (թվարկում) [Троелсен-167; Нейгел-90; Шилдт -397] (էջ 53)
41. System.Enum [Троелсен-169] (էջ 53)
42. foreach ցիկլ [Троелсен-145; Нейгел-197] (էջ 54)
43. Անհամաչափ զանգված [Троелсен-163; Нейгел-188] (էջ 55)
44. System.Array [Троелсен-165; Либерти-192] (էջ 56)
45. System.String [Нейгел-266; Троелсен-127; Либерти-252] (էջ 57)
46. System.Text.StringBuilder [Троелсен-133; Petzold-546] (էջ 59)
47. String.Format [Шилдт – 812; Нейгел-215; Covaci – 150] (էջ 60)
48. Հատկանիշ [Шилдт-313; Троелсен-212; Рихтер-248] (էջ 61)
49. Ավտոմատ հատկանիշներ [Троелсен-219; Нэш -69; Шилдт-318; Нейгел-124; Скит-224] (էջ 62)
50. Օպերատորների գերբեռնում [Троелсен-426; Нейгел-219; Шилдт-269] (էջ 63)
51. Ինդեքսատոր [Троелсен-421; Шилдт-303] (էջ 66)
52. Առանց կուտակիչի ինդեքսատոր [Шилдт-310] (էջ 67)
53. Գերբեռնված ինդեքսատոր [Шилдт-307; Либерти-216] (էջ 67)
54. Main() մեթոդ [Троелсен-108; Шилдт-254; Нейгел-95] (էջ 68)
55. Մեթոդի անվանական արգումենտներ [Шилдт-252; Нейгел-121; Уотсон-411] (էջ 69)
56. Մեթոդի ոչ պարտադիր պարամետրեր [Шилдт-247; Нейгел-121; Уотсон-411] (էջ 69)
57. Մեթոդի հղող պարամետրեր - ref, out [Троелсен-150; Шилдт-222] (էջ 70)
58. ref, out հղիչային տիպերի համար [Шилдт-227] (էջ 72)
59. Մեթոդի պարամետր – params [Троелсен-154; Шилдт-229] (էջ 73)
60. partial կլասներ, partial մեթոդներ [Нэш-91; Нейгел-133; Троелсен-448] (էջ 75)
61. var - ոչ հստակ տիպայնացում [Троелсен – 140; Скит -226; Нейгел-72; Уотсон-400] (էջ 76)
62. Անանուն տիպեր [Уотсон-402; Нейгел-130; Скит Джон-239; Шилдт-663] (էջ 77)
63. Ընդլայնված մեթոդներ [Троелсен-440; Нэш-477; Уотсон-417] (էջ 78)
64. Կանոնավոր արտահայտություններ [Албахари –981; Нейгел-277] (էջ 80)

Օբյեկտ կողմնորոշվածություն (Object oriented)
65. Ժառանգում, protected [Троелсен-231,240; Шилдт-329,336] (էջ 83)
66. Ժառանգում և new, base [Шилдт-343; Нейгел-144] (էջ 84)
67. Բազային կոնստրուկտորի կանչ : base [Троелсен-238; Шилдт-339; Нейгел-149] (էջ 84)
68. Ներդրված տիպերի վերափոխարկում [Троелсен-135; Нейгел-228] (էջ 86)
69. Կոնվերտացիաներ [Covaci – 122; Kort-185] (էջ 87)
70. Ժառանգական կապվածությամբ տիպերի վերափոխարկում [Троелсен-434,403; Нейгел-233] (էջ 88)
71. Հղում բազային կլասով [Шилдт-351] (էջ 88)
72. Վիրտուալ մեթոդներ, virtual, override, Պոլիմորֆիզմ [Шилдт-355,43; Троелсен-245] (էջ 89)
73. Տիպերի վերափոխարկում implicit, explicit [Троелсен-435,403; Нейгел-230] (էջ 90)

74. Մեկուսացված մեթոդներ և կլասներ [Троелсен-248, 234; Шилдт-367] (էջ 91)
75. Աբստրակտ կլասներ [Троелсен-249; Шилдт-363] (էջ 92)
76. Ներդրված կլասներ [Троелсен-243; Либерти-141] (էջ 93)

Օբյեկտ (Object)
77. Object կլաս և նրա անդամները [Троелсен-258; Нэш-114; Шилдт-368; Нейгел-135] (էջ 94)
78. Object կլասի ToString() անդամի վերավորոշումը [Троелсен-261; Нейгел-137] (էջ 95)
79. Object կլասի Equals() անդամի վերաորոշումը [Троелсен-261; Нэш-445; Рихтер-155] (էջ 95)
80. System.Object.GetHashCode() մեթոդի վերաորոշումը [Троелсен - 262, 258; Нэш-448] (էջ 96)
81. Object կլասի static անդամները [Троелсен-264; Нейгел-218] (էջ 97)
82. Object. MemberwiseClone [albdarb; Нэш-427] (էջ 98)
83. object - որպես տիպերի ընդանրություն [Shiltd-372] (էջ 99)
84. Դինամիկ տիպեր - dynamic [Троелсен-648; Нэш-557; Шилдт-703] (էջ 100)

Ինտերֆեյս (Interface)
85. Ինտերֆեյս [Либерти-162; Шилдт-375; Троелсен-320] (էջ 102)
86. Ինտերֆեյսի հղում [Шилдт -381; Либерти-171] (էջ 103)
87. Ինտերֆեյսի is և as պատկանելիության ձևեր [Либерти-173; Троелсен-329] (էջ 104)
88. Ինտերֆեյսը մեթոդի պարամետր և վերադարձվող արժեք [Троелсен-330] (էջ 106)
89. Ինտերֆեյսների ժառանգականություն, բազմաժառանգում [Шилдт -387; Троелсен -337, 304] (էջ 107)
90. Ինտերֆեյսների բացահայտ իրականացում [Троелсен -334,301; Шилдт-388] (էջ 108)
91. Ինտերֆեյս ICloneable [Троелсен-346; Шилдт-779] (էջ 109)
92. Ինտերֆեյս IEnumerable, IEnumerator [Троелсен-340; Шилдт - 1001] (էջ 111)
93. foreach և IEnumerable [Нейгел-197; Уотсон-295] (էջ 113)
94. Իտերատորներ, օպերատոր yield [Троелсен-343; Нейгел-197; Уотсон-295; Шилдт-1003] (էջ 113)
95. Անվանական իտերատորներ [Троелсен-344; Нейгел-199] (էջ 114)
96. Ինտերֆեյս ICamparable [Троелсен-350; Шилдт-778] (էջ 116)

Դելեգատ (Delegate), **Իրադարձություն** (Event)
97. Դելեգատներ [Шилдт-473; Троелсен-386; Нейгел-241] (էջ 118)
98. Խմբային դելեգատներ [Шилдт-476; Троелсен-396; Нейгел-251; Либерти-316] (էջ 119)
99. Delegate և MulticastDelegate բազային կլասներ [Троелсен-389; Нэш - 288] (էջ 120)
100. Անանուն մեթոդներ [Нейгел-254; Нэш-314; Троелсен-411; Шилдт-484] (էջ 122)
101. Lambda արտահայտություն [Шилдт-488; Нэш-503; Нейгел-255; Троелсен-413; Скит -246] (էջ 123)
102. Covariance և Contravariance դելեգատներ [Шилдт-481; Троелсен-400] (էջ 125)
103. Իրադարձություն [Троелсен-404; Шилдт-494] (էջ 127)
104. FCL գրադարանի իրադարձություններ, EventHandler դելեգատ [Шилдт-506,508; Либерти-321] (էջ 128)

Բացառություն (Exception)
105. Բացառիկ իրավիճակներ [Троелсен-265; Нейгел-419; Шилдт-403] (էջ 130)
106. Արհեստական բացառիկ իրավիճակ [Шилдт-414] (էջ 131)
107. Կրկնակի արտակարգ իրավիճակ [Шилдт-415] (էջ 132)
108. Բազմակի catch – ի օգտագործում [Троелсен-282; Нейгел-423] (էջ 133)
109. System.Exception [Шилдт-418; Троелсен-268; Нейгел-427] (էջ 134)
110. Ժառանգված բացառիկ իրավիճակ [Шилдт-422] (էջ 135)
111. checked , unchecked [Шилдт -428 ; Троелсен-139; Нейгел-209] (էջ 136)

Ընդհանրացում (Generics)
112. Ընդհանրացումներ [Нэш-309; Шилдт-575; Нейгел-161; Рихтер-289; Троелсен-356] (էջ 137)
113. Կլասների ընդհանրացում և where սահմանափակում [Шилдт-585; Троелсен-383] (էջ 138)
114. Generics - մեթոդներ [Нэш-316; Шилдт-607; Троелсен-376] (էջ 141)
115. Generics - դելեգատներ[Нэш-318; Шилдт-610; Троелсен-402] (էջ 142)
116. Action<> և Func<> դելեգատներ [Троелсен -371; Нейгел- 247; Covaci - 216] (էջ 143)

117.Generics – ինտեֆեյսներ [Шилдт -612; Нейгел -171] (էջ 144)
118.Ինտերֆեյսի Covariance ընդհանրացումներ [Шилдт -626; Нейгел-171; Уотсон-362] (էջ 145)
119.Ինտերֆեյսի Contravariance ընդհանրացումներ [Шилдт -630; Нейгел-174; Уотсон-364] (էջ 146)
120.Կլասի հետաձգված ինիցալիզացիա - Lazy<> [Троелсен-314; Гриффитс-873] (էջ 146)

albdarb.com
user: books psw: “sharp” psw: “sharp2” psw:”sharp3” (ռուսերեն գրականություն)
user: books psw: “sharpeng” psw: “sharpeng2” (անգլերեն գրականություն)

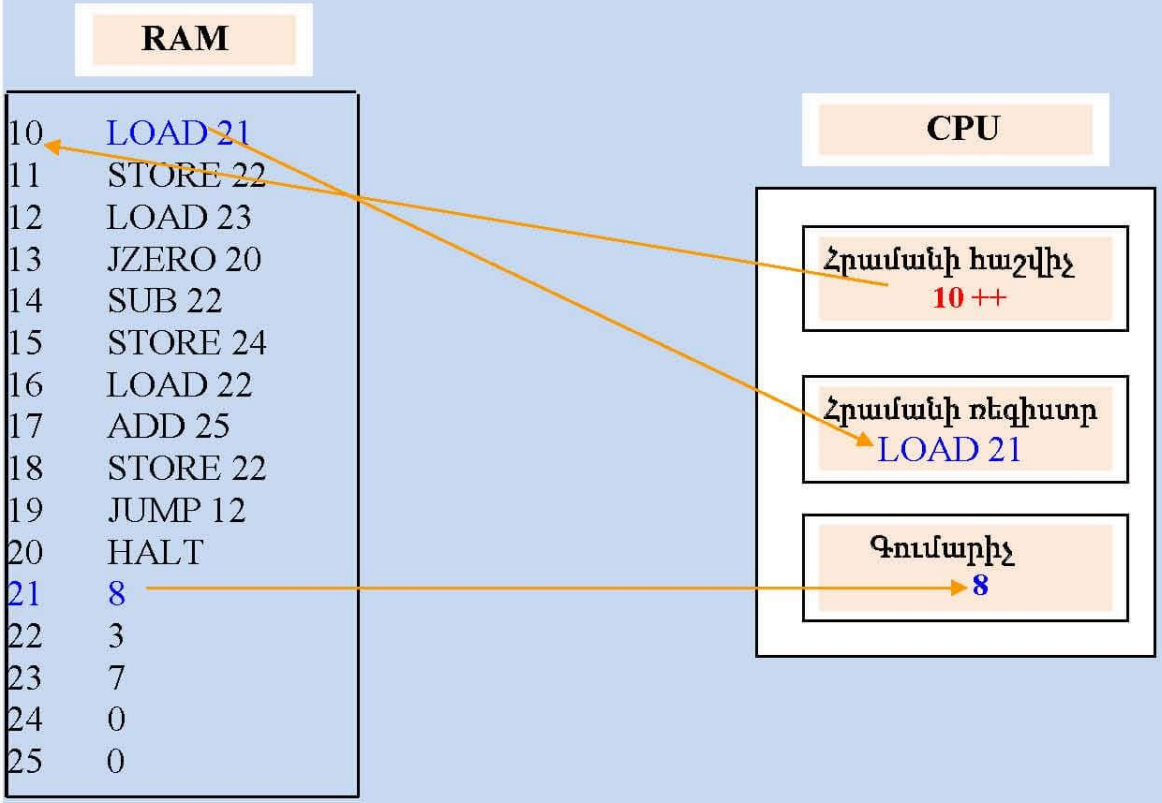
ԳՐԱԿԱՆՈՒԹՅՈՒՆ

1. Шилдт Г. -“Полное руководство C#4.0”, 2010.
2. Троелсен Э. - “C# 2010 и платформа .NET 4.0”, 5ed, 2010.
3. Нэш Трей – “C# 2010. Ускоренный курс для профессионалов”,2010.
4. Нейгел и др…. - ”C# 4 и платформа .NET 4 для профессиоралов”, 2010.
5. Рихтер Джеффри –“ CLR via C#. Программирование на платформе Microsoft .NET Framework 4.0 на языке C#” , 2010.
6. Либерти Дж. - “Программирование на C # “, 2ed, 2002eng.
7. Уотсон К., Нейгел К. – “Visual C# 2010: полный курс. ”, 2010.
8. Скит Джон– “C# программирование для профессионалов”, 2011.
9. Йен Гриффитс – “Программирование на C# 5.0”,2013.
10. Tiberiu Covaci…”MCSD Certification Toolkit (Exam 70-483): Programming in C#”, 2013.
11. Wouter de Kort - “Exam Ref 70-483: Programming in C#”, 2013.
12. Албахари Джозеф, Албахари Бен - C# 5.0. Справочник. Полное описание языка, 2012.
13. Троелсен Э. - Язык программирования C# 5 и платформа .NET 4.5, 6ed, 2012.
14. Рихтер Дж. - CLR via C#. Программирование на платформе NET Framework 4.5 на языке C#, 2012.
15. Christian Nagel … , “Professional C# 2012 and .NET 4.5”, 2012.
16. Дейтел П., Дейтел Х. – “Как программировать на Visual C# 2012”,2012.

Ալգորիթմական C#

1.Ծրագրի հրամանների կատարման հաջորդականությունը

- Համակարգիչների կառուցվածքը տվել է Ֆոն Նեյմանը դեռ 40 ական թվականներին: Նա նշել է, որ ապագա համակարգիչը պետք է ունենա ղեկավարման մաս (այժմ CPU), հիշողություն (այժմ RAM), մուտք/ելքի մաս: Բացի դրանից, այն պետք է լինի էլեկտրոնային, հաշվի երկուական համակարգով և **հրամանները կատարի հաջորդաբար**:



- Ծրագրի հրամանները կատարվում են պրոցեսորի միջոցով: Նրանք պետք է գտնվեն օպերատիվ հիշողությունում: Ծրագրի **հրամանները** և **տվյալները** ստանում են հասցեներ և սկզբից ծրագրի կատարման առաջին հրամանի հասցեն տեղադրվում է հրամանի հաշվիչում: Հաջորդ քայլում հաշվիչի պարունակության հասցեի համապատասխան հրամանը բերվում է հրամանի ռեգիստր: Սկսվում է կատարվել ռեգիստրում գտնվող հրամանը, միաժամանակ կատարվում է շատ կարևոր գործողություն, **հրամանի հաշվիչի պարունակությունը ավելանում է մեկով**, որը նշում է հաջորդ հրամանի հասցեն: Այս ձևով ապահովվում է ծրագրի ավտոմատ կատարումը: Ավելացնենք, որ համակարգիչը գլխավորապես կատարում է միայն գումարում և ցանկացած գործողության կատարումը կազմակերպվում է գումարիչի միջոցով: Օրինակ, եթե անհրաժեշտ է 22 հացեի պարունակություն 3 թիվը տանել 23 հասցե, ապա այն սկզբից բերվում է գումարիչ, հետ նոր միայն այնտեղից 23 հասցե:
- Ծրագրավորման տեսանկյունից կարելի է ասել, որ ծրագիրը բաղկացած է հրամաններից (ADD, LOAD, HALT), տվյալներից (8, 3, 7, 0, 0) և հիշողության հասցեներից (10, 11, 12, 13 …..):
- **Մինիմալ ինֆորմացիոն միավորը**, որը կարելի է գրել կամ կարդալ հիշողությունից, դա 8 բիթն է: Այդ պատճառով, եթե անհրաժեշտ է նույնիսկ գրել 1 բիթանոց տվյալ, միևնույնն է պետք է զբաղեցնել 8 բիթ տարածք:

2. Թարգմանիչներ: Console Application – (թվային մուտք)

[albdarb]

- Ցանկացած խնդիր ծրագրավորելու համար անհրաժեշտ է թարգմանիչ: Թարգմանիչը որևէ լեզվով գրած ծրագիրը թարգմանում է երկուական կոդի, որպեսզի համակարգիչը կարողանա այն կատարել: Ինչպես թարգմանիչն է հիշողության մեջ տեղ գրավում, այնպես էլ որևէ լեզվով գրած ծրագիրը և նրա թարգմանված 2- ական կոդը:
- Թարգմանիչները լինում են երկու տեսակ: Ինտերպրետատոր (նախկին Basic լեզուն, ժամանակակից "բրովզերները – Google Chrom, Microsoft intrnet Explorer ") կամ կոմպիլիատոր(C, Pascal, C++): Ինտերպրետատրի ժամանակ ծրագիրը թարգմանվում է մաս մաս ու տեղում կատարվում: Ծրագրի ամեն նոր կանչի ժամանակ կատրվում է նոր թարգմանում: Կոմպիլիատրի ժամանակ ծրագիրը թարգմանվում է ամբողջությամբ, միայն մեկ անգամ և կատարվում թարգմնված ծրագրի բազմակի օգտագործում: Ինտերպրտատորը աշխատում է դանդաղ, սակայն օգնում սխալները շուտ հայտնաբերել: Կոմպիլիատորը ավելի պրոֆեսիոնալ մոտեցում է պահանջում և նրա թարգմանած ծրագիրը աշխատում է ավելի արագ:
- C#sharp լեզուն կարելի է ասել ավելի շատ կոմպիլիտոր է քան ինտերպրետատոր: Ավելի շուտ, C# դա dotNET համկարգի լեզու է և աշխատում է նոր սկզբունքով: Այդ զկզբունքը կոչվում է ղեկավարման կոդ, որը կանցնենք ավիլի ուշ:
- Console Application -ի օրինակ. (անդուր սև սպիտակ մուտք/ելքի տեսքով)

```
using System;
class M
{
    public static void Main(string[] args)
    {
        Console.WriteLine("baerv bolorin"); // ելք
        int a = 999; // փոփոխականի հայտարարում
        int b = 222;
        int c = a + b;
        Console.WriteLine("The sum of " + a + " and " + b + " = " + c); // ելք
        Console.WriteLine("The sum of {0} and {1} = {2}", a, b, c); // ելքի այլ տարբերակ
        string aa = Console.ReadLine();// մուտք
        a = Convert.ToInt32(aa); // կոնվերտացիա սիմվոլային տողից - թիվ
        //a=Int32.Parse(aa); // կոնվերտացիայի այլ տարբերակ
        b = Convert.ToInt32(Console.ReadLine()); // մուտք
        //b=Int32.Parse( Console.ReadLine() ); // մուտք
        c = a + b;
        Console.WriteLine(c); // ելք
        Console.ReadKey();// ծրագրի ՀԱՐՄԱՐ սպասում արդյունքը դիտելու համար
    }
}
```

……

3.Ներդրված տիպեր

[Нейгел-78; Шилдт-68]

- Ծրագրավորումը, դա լուծման նպատակով խնդրի ալգորիթմային ներկայացումն է, որը կատարվում է համակարգչի միջոցով: Խնդրի լուծման համար անհրաժեշտ է տվյալների մշակում, որը իր հերթին բերում է տվյալների պահման(հիշման) անհրաժեշտության: Տվյալների պահումը և մշակումը կատարվում է տիպերի միջոցով: Ծրագրավորման լեզուները ունեն մեծ քանակով տիպեր, որոնք թույլ են տալիս տվյալները օգտագործել օպտիմալ ձևով:
- C# լեզուն խիստ տիպային լեզու է: Տիպային լեզուների առավելություններից է համարվում հիշողության օպտիմալ օգտագործումը: Մյուս առավելութուններից է կոդի սխալների շուտ

հայտնաբերումը, որը լավ կլինի կատարվի թարգմանության ժամանակ: Ժամանակակից լեզուներում երկրորդ չափանիշը ավելի առաջնային է համարվում:

- Ներկայացնենք C# լեզվի ներդրված տիպերը (built-in types):

byte	8bit	0-255
sbyte	8bit	-128 to 127
short	16bit	-32768 to 32767
ushort	16bit	0 to 65535
int	32bit	-2147483648 to 2147483647
uint	32bit	0 to 4294967295
long	64bit	.....
ulong	64bit	0 to 18446744073709551615
float	32bit	..... 10^{38}
double	64bit	..... 10^{308}
decimal	128bit	..... 10^{28} - ունի շատ մեծ ճշտություն
char	16bit	(unicode) - 2^{16} = ~65000 սիմվոլներ
bool	8bit	ընդունում է միայն (true or false) արժեքներ
string	աշխատանք տողերի հետ	
object	ընդհանուր տիպայնացում	

- // Տնային: Հարց: float: Ինչպես կարելի է 32 բիթով ստանալ 10^{38} թիվ:
- Պարզվում է, որ գոյություն ունեն նաև մեծ քանակով (տասնյակ հազարավոր) այլ տիպեր, որոնք չեն համարվում առաջին անհրաժեշտության տիպեր: Նրանք չեն ստանում կապույտ գույն և շատ ավելի դանդաղագործ են: Պարզ է, որ ներդրված տիպերը ավելի արագագործ են, ուղղակի բոլորին հնարավոր չէ ընդգրկել ներդրված մասում (աշխատում է էնեգիայի պահպանման օրենքը- մի բան պիտի զիջես, որպեսզի մի այլ բան ստանաս):
- byte, sbyte, short, ushort, int, uint, long, ulong տիպերը աշխատում են ամբողջ թվերի հետ: Կա պայմանավորվածություն: Տիպին հատկացվող բիթային տարածքի ամենաձախ բիթը համարվում է թվի նշանի բիթ և եթե այն զրո է, ապա թիվը դրական է: Եթե այն մեկ է, ապա թիվը բացասական է: sbyte, short, int, long, թույլատրում են աշխատել նաև բացասական թվերի հետ: byte, ushort, uint, ulong տիպերը աշխատում են միայն դրական թվերի հետ:
- float, double, decimal տիպերը աշխատում են կոտորակային թվերի հետ: Նրանք պարունակում են շատ մեծ թվեր: Մեծ թվերի հետ աշխատանքը կազմակերպվում է հետևյալ հնարքով, - թիվը կարելի է ներկայացնել երկու բաղկացուցիչով: Առաջինը կոչվում է "մանտիս" և պարունակում է միայն կոտորակային թիվ: Երկրորդը կոչվում է "կարգայնություն" և ներկայացվում է 10 ի աստիճաններով: Օրինակ.
  56789= 0,56789 * 10^5 :
  Թվերի այսպիսի ներկայացման դեպքում կարելի է շատ փոքր կորուստով աշխատել մեծ թվերի հետ, կատարելով մաթեմաթիկական գործողություններ:
  Վերցնենք float տիպը: Այն ունի 32 բիթ հիշողության տարածք (նույնը ունի նաև int տիպը, սակայն այն 4 միլիարդից ավել չի կարող պահել): 10^{38} աստիճան ստանալու համար 26 բիթ հատկացվում է "մանտիսին" և 6 բիթ կարգայնությանը: 6 բիթ ուղղակի, որպեսզի 38 թիվը 2 ականով հնարավոր լինի ներկայանել:
- decimal տիպը նորույթ է: Նրա բիթային տարածքը շատ մեծ է(128բիթ), սակայն բիթերի մեծ մասը հատկացվում է մանտիսին "սուպեր" ճշտություն ապահովելու համար(երևի որոշ բնագավառներում անհրաժեշտ է):
- Ըստ մեծության կոտորակային թվերը դասավորվում են decimal, float, double հերթականությամբ:

```
double doub = 1.0E308; //փորձել
float fl=1.0E38f;
decimal dec = 1.0E28m;
```

- char տիպը նախատեսվում մեկ նիշ(սիմվոլ) պահելու համար: C# ում այն ունի 16 բիթ տարածք(նախկին լեզուներում 8 բիթ էր), որը հնարավորություն է տալիս հաղթահարել լեզվային սիմվոլների սահմանափակումները: Այլ խոսքով, առանց միջնորդ ծրագրերի(KDWin - մադեվին), կարելի է աշխատել աշխարի բոլոր լեզուների հետ: Այստեղ նույնպես ունենք հիշողության կորուստ, սակայն շահում ենք ավելի "կոնֆորտ" աշխատանքում:
- bool տրամաբանական տիպ է: Այն ունի վերացականության հատկություն: Ինչպես գիտենք, համակարգիչը կառուցված է տրամաբանական էլեմենտներից: Նրանք են "և" "կամ" "ոչ": Այլ խոսքով համակարգչի ԴՆԹ –ն կազմված է 3 ելեմենտից: Նշված էլեմենտներից կազմված են սխեմաները, որոնք կատարում են տրամաբանական աշխատանքներ: Ահա այս տրամաբանական մոտեցումը կիրառվում է նաև ծրագրավորման մեջ: Ծրագրի որևէ հատված մեզ կարող է ասել ինչ որ բան ճիշտ է, թե ոչ: Ճիշտ է գաղափարը պայմանավորվենք true անվանել, ճիշտ չէ գաղափարը պայմանավորվենք false անվանել, պարզ է համապատասխան տարածք տրամադելով ծրագրի այս պահվածքին:
- string և object տիպերը կանցնենք ավելի ուշ: Նշենք, որ string տիպը աշխատում է տողերի հետ, որոնք կարելի է համարել սիմվոլների խումբ: Իսկ object տիպը ամենատարողունակ տիպն է և կարող է հանդես գալ ցանկացած տիպի փոխարեն:

……

- Տնային: Ցանկալի է տիրապետել "Ինֆորմատիկա" առարկայից 2 –ական թվերի հետ գործողությունների կատարմանը: Իմանալ 16 ական թվերի մասին: Կատարել 2 –ականից 16 ական և հակառակ գործողություններ: Տիրապետել բացասական թվերի գաղափարին:

……

4.Օպերացիաներ և օպերատորներ

[Шилдт-97]

- Պայմանավորվենք, որ համակարգչի տեսակետից **օպերացիաները** ավելի թույլ գործողություններ են կատարում, իսկ **օպերատորները** ավելի բարդ: Վեջին հաշվով նրանք նույնն են:
- Նշենք նաև, որ ցանկացած հրաման ավարտվում է կետ ստորակետով (;) :

Վերագրման օպերատոր (=) [Шилдт- 106]

c=a+b;

- Վերագրման օպերացիան դա մաթեմատիկական նույնություն չէ: Այսինքն, եթե հավասարության աջ մասը փոփոխենք, ապա ձախ մասը համապատասխանորեն կփոփոխվի: Իսկ, եթե ձախ մասն ենք փոխում աջ մասը չի փոխվում: Մաթեմատիկայում այդպես չէ: Նշված երևույթը կապված է ծրագրավորման առանձնահատկություն հետ: Այն խիստ կախված է համակարգչի ներքին կառուցվածքից, հիշողության կազմակերպումից և ռեսուրսներից:

Թվաբանական օպերացիաներ (+, –, *, /, %) [Шилдт- 97]

- Ցանկացած թվային տիպի հետ կարելի է կատարել թվաբանական գործողություններ, որոնք նման են մաթեմատիկականին:
- / - բաժանում գործողությունը ամբողջ տիպերի դեպքում (byte, short, int, long) պահում է միայն թվի ամբողջ մասը: % գործողությունը կատարում է բաժանում և պահում է մնացորդը: Նշենք որ, կոտարակային թվերի դեպքում / - բաժանում կատարվում է նորմալ և % գործողությունը նույնպես պահում է մնացորդը:

```
using System; // Գտնել եռանիշ թվի թվանշանների գումարը:
class Program
{
     static void Main(string[] args)
    {
      int a = 567;
      int a1, a2, a3;
```

```
      a1 = a / 100;
      a2 = a % 100 / 10;
      a3 = a % 10;
      Console.WriteLine(a1 + a2 + a3);
     }
}

// (տնային) ցանկացած թվի  թվանշանների  առանձնացում  և գումար:
// մի տանջվեք, չի ստացվի: Դրա համար նոր հրամաններ են պետք:
```

Համեմատման օպերացիաներ (**< ,>,==,!=, <=, >=**) [Шилдт-101]

Տրամաբանական օպերացիաներ (**&&, ||, !**) [Шилдт-101]

- Ֆոն Նեյմանի առաջարկած համակարգչի կարուցվածքի հիման վրա, հրամանները կատարվում են հաջորդաբար, իսկ թարգմանիչում դա արտահայտվում է վերևից ներքև զաղափարով (այսպես ասած ազդում է "գրավիտացիան"): Հրամանների կատարման հաջորդականությունը ուղղեկցվում է պայմաններով և ցիլկերով: Պայմանի ստեղծման ժամանակ օգտագործվում է նոր օպերացիաներ, որոնք փոփոխականների միջև կատարում են ինչպես համեմատում (**< ,>,==,!=, <=, >=**), այնպես էլ տրամաբանական միացում կամ խմբավորում (**&&, ||, !**) "և" "կամ" "ոչ" համակարգով:
- Հրամանները ունեն կատարման հետևյալ հաջորդականությունը - սկզբից ( ) կլոր փակագծեր, այնուհետև գալիս է տրամաբանական "և" **&&,** վերջապես տրամաբանական "կամ"**||**:
- ✓ Օրինակը ապացուցում է, որ **&&** ավելի առաջնային է, քան || օպերացիան.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 999;
        char c = 'A';
        bool b = (16 > 8) || (a > 6) && (c == 'B');
        Console.WriteLine(b); //True
    }
}
```

Մեծացման և փոքրացման օպերացիաներ (**++, --**), *Պրեֆիքս և պոստֆիքս* [Шилдт- 98]

- **++** գործողությունը համակարգչում մեկով ավելանալու զաղափարն է առաջ տանում, որը նաև ապահովում է պրոցեսորի(CPU) ավտոմատ իրականացումը, ծրագրի հրամանների կատարման հաջորդականության տեսանկյունից: **++** գործողությունը կոչվում է "Increment", որի հակառակ գործողությունը (փոփոխականի պարունակությունը մեկով պակասացնելը) կոչվում է "Decrement":
- (**++, --**) գործողությունները կարելի է կիրառել ինչպես փոփոխականից առաջ, այնպես էլ հետո (prefix, postfix): Եթե (**++, --**) գործողությունները ուղղեկցվում են այլ հրամանների հետ, ապա այդ ժամանակ սկսում է դեր խաղալ (**++, --**) գործողությունների տեղը և որոշվում է, սկզբից կատարել այլ հրամանը հետո (**++, --**), թե հակառակը:
- a++;
  a=a+1;
  a+=1;

Նշված երեք հարմանները կատարում են նույն գործողությունը, ավելացնում են a փոփոխականը մկով: Եթե անհրաժեշտ է ավելացնել փոփոխականի արժեքը այլ թվով, ապա օգտագործվում է երկրորդ կամ երրորդ տարբերակը: a=a+5; կամ a+=5;

```
using System;
class Program
{
```

```
    static void Main(string[] args)
    {
        int a = 4;
        a++;
        Console.WriteLine(a);
        Console.WriteLine(++a);
        Console.WriteLine(a++);
        Console.WriteLine(a);
    }
}
```

- Օրինակում ծրագիրը կարտածի 5 6 6 7 հաջորդականությունը:

……

5.Պայմանական օպերատոր if, if….else

[Шилдт-121]

- Պայմանական օպերատորը հնարավորություն է տալիս կատարել ծրագրի հրամանների կատարման հաջորդականության ճյուղավորում կախված պայմանից:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int a=999;
        char c = 'A';
        if(a==999 && c=='A')
            Console.WriteLine("barev");
        else
            Console.WriteLine("again barev");
    }
}
//---------------------------------
```

- if պայմանական օպերատրում թույլատրվում է օգտագործել վերագրման (=) օպերատոր: Անհրաժեշտ է օգտվել ( ) կլոր փակագծերից:

```
        int a;
        if ((a = 88) > 0)
               Console.WriteLine("barev");
```

……

6.Պայմանական օպերատոր (?)

[Шилдт- 117]

- (?) պայմանական օպերատորը if… else օպերատորի նման կատարում է ծրագրում ճյուղավորման աշխատանք: Ինչպես նաև ունի առանձնահատկություն, օգտագործվել ֆունկցիաների (մեթոդների) պարամետրերում և առանձնահատուկ տեղերում:

```
using System;
class Program
{
        static void Main(string[] args)
        {
            int a = 5;
            int c = (a > 0) ? 44 : 99;
            Console.WriteLine(c);
            Console.WriteLine((a < 0) ? 55 : 77);
        }
}
```

……

7.Ցիկլիկ օպերատորներ - while, do… while, for

[Шилдт-129]

- Ցիկլիկ օպերատորը հնարավորություն է տալիս ծրագրի որևէ հատված կրկնել և այդ հնարավորությունն է ստիպում համակարգչին լուծել բարդ խնդիրներ: Առաջին ծրագրավորողը համարվում է Ադա Բայրոնը, ով և առաջարկել է ցիկլի գաղափարը:
- while օպերատորը ստուգում է պայմանը և կատարում ցիկլը:
- do… while օպերատորը սկզբից կատարում է ցիկլը, հետո ստուգում պայմանը և դուրս է գալիս ցիկլից պայմանը չբավարարվելու դեպքում:
- for օպերատորը օգտագործվում է այն դեպքում, երբ առկա է որևէ փոփոխականի սկզբնական արժեք, նրա մեծության ստուգման պայման և փոփոխականի ինքնամեծացում կամ փոքրացում: Այլ խոսքով երեք էրնեկ մի տեղ:
- Ցիկլերը, ինչպես նաև if օպերատորը ունեն առանձնահատկություն: Զևավոր փակագծեր պարտադիր են, եթե ցիկլը ունի միակ հրաման տիպի հայտարարումով: Հաջորդ օրինակը առանձնահատուկ դեպք է.

```
 static void Main(string[] args)
 {
     for (int i = 0; i < 5; i++)
    // {
         // //System.Console.WriteLine();
         int a=99; //error
     //}
 }
```

- Հավերժ ցիկլեր կարելի է ստանալ while(true) կամ for(; ;) հրամաններով:

```
using System; // do… while
class Program
{
        static void Main(string[] args)
        {
            int a = -10;
            do
            {
                Console.WriteLine("barev");

            } while (a > 0);
        }
}
//-----------------------------------
using System; // while,   for
class Program
{
    static void Main(string[] args)
    {
        int n = 8;
        int sum = 0;
        int i = 0;
        while (i <= n)
        {
            sum = sum + i;
            i++;
        }
        Console.WriteLine(sum);
        sum = 0;
        for (int i2 = 0; i2 <= n; i2++)
            sum = sum + i2;
        Console.WriteLine(sum);
}   }
```

```
//-----------------------------------
using System; // n թվի  թվանշանների  գումար // կիսատնային
class M
{
    static void Main(string[] args)
    {
        for (; ;)
        {
            int sum = 0;
            Console.Write("tiv = ");
            int n = Convert.ToInt32(Console.ReadLine());
            int tiv = 0;
            while (n > 0)
            {
                tiv = n % 10;
                n = n / 10;
                Console.Write(tiv+"  ");
                sum += tiv;
            }
            Console.WriteLine("gumar = "+sum);
        }
    }
}
//-----------------------------------
// n թվի  թվանշանների  առանձնացում և ցուցադրում նույն հաջորդականությամբ – (տնային)
// Պարզ թվի որորշում – (տնային)
Ֆիբոնաչիի շարք 1, 1, 2 , 3, 5, 8, 13, 21, 34, 55 ……..
Շարքի առաջին երկու անդամները հավասար են 1: Մնացած անդամները որոշվում են նախորդ երկու անդամների գումարով:
// Ֆիբոնաչիի շարքի անդամի որոշում – (տնային)
//-----------------------------------
using System; // n թվի  թվանշանների  առանձնացում – (տնային)
class M
{
    static void Main(string[] args)
    {
        for (; ;)
        {
            Console.Write("tiv = ");
            int n = Convert.ToInt32(Console.ReadLine());
            int tvanshan = 0;
            int nn = n;
            int k = 0; // թվանշանների քանակ
            while (n > 0)
            {
                n = n / 10;
                k++;
            }
            Console.WriteLine("tvanshanneri qanak  = "+k);
            int kk = 1; // թվի կարգայնություն
            for (int i = 1; i < k;i++ )
                kk*=10;
            Console.WriteLine("kargaynutiun = "+kk);
            for (int i = 0; i < k;i++ )
            {
                tvanshan = nn / kk;
                nn = nn % kk;
                kk = kk / 10;
```

```
                Console.Write(tvanshan + "  ");
            }
            Console.WriteLine();
        }
    }
}
//-----------------------------------

using System; // Ֆիբոնաչի
class M
{
    static void Main(string[] args)
    {
        for (; ; )
        {
            Console.Write("tiv = ");
            int n = Convert.ToInt32(Console.ReadLine());
            int a1 = 1;
            int a2 = 1;
            int fib = 0;
            Console.Write(a1+" ");
            Console.Write(a2+" ");
            for (int i = 2; i < n;i++ )
            {
                fib = a1 + a2;
                Console.Write(fib+" ");
                a2 = a1;
                a1 = fib;
            }
            Console.WriteLine();
        }
    }
}
```

……

8.Անցումային օպերատորներ – break, continue, goto

[Шилдт-139]

- Ցիկլերում թույլատրվում է օգտագործել break օպերատորը, որը ցիկլից դուրս է բերում ծրագրի հատվածը:
- continue օպերատորի կատարման ժամանակ բաց է թողնվում նրան հաջորդող բոլոր հրամանները, սակայն ցիկլից դուրս չի գալիս ծրագիրը:
- goto օպերատորը կատարում է թռիչք ծրագրի ցանկացած մասից ցանկացած այլ մաս, նշված “մետկայով:”: Ժամանակակից լեզուներում խորհուրդ չի տրվում օգտագործել այս օպերատորը, որովհետև կա ծրագրավորման դառը փորձ, այսպես կոչված “սպագետտի կոդ”, որի իմաստը կայանում է հետևյալում. երբ շատ goto - ներ եք օգտագործում, ապա ծրագրային կոդը խճճվում է՝ բերելով ծրագրի խափանման:

```
using System; // break
class Program
{
        static void Main(string[] args)
        {
            int a = 5;
            while (true)
            {
                Console.WriteLine("barev");
                a--;
```

```
                if (a <= 0)
                    break;
            }
        }
}
// ------------------------------------
using System; // continue
class Program
{
        static void Main(string[] args)
        {
            int a = 8;
            while (a>0)
            {
                Console.WriteLine("barev");
                a--;
                if (a >=3)
                    continue;
                Console.WriteLine("barev2222");
            }
        }
}
// ------------------------------------
using System; // if + goto = ցիկլ
class Program
{
        static void Main(string[] args)
        {
            int a = 5;
         aa:
            if (a > 0)
            {
                Console.WriteLine("barev");
                a--;
                goto aa;
            }
        }
}
```

……

9.Բիթային օպերացիաներ (|, &, ~, ^, <<, >>)

[Шилдт-107]

- Կա հետևյալ պատմությունը: Pascal և C լեզուների պայքարի ժամանակ հաղթող դուրս եկավ C լեզուն, որը գրավեց և արտադրական ծրագրավորումը և ուսման մեջ կիրառումը: C լեզվի հաղթանակի հիմքերում է ընկած այնպիսի բարդ հնարավորություններ, ինչպիսին են բիթային օպերացիաները, կամ ասենք հասցեներով աշխատելու հնարավորությունները ցուցիչների և հղիչների միջոցով: Նշված գործողությունները համարվում էին ասեմբլեր լեզվի յուրահատկություններ և ուսուցման տեսանկյունից տանում են դժավարությունների:
- Բիթաին օպերացիաները նույնպես կիրառվում է C#sharp լեզվում: Այն հարավորություն է տալիս աշխատել բիթերի մակարդակով, որը կարելի է կիրառել ինչպես տրամաբանական խնդիրներում, այնպես էլ տարբեր արտաքին միջավայրներում (ասենք արտաքին սարքեր), որտեղ գործում են միջավայրի անհատական հրամանները: Այդ հրամանները շատ ավելի հեշտ ղեկավարվում են բիթային օպերացիաների միջոցով: Օրինակ արտաքին սարքին կարելի է ուղղարկել տարբեր կոդեր,

որոնք այդ սարքի համար դառնում են հրամաններ: Այդպիսի ծրագրային իրականացումը կոչվում է սարքի “driver” ծրագիր:

✓ | բիթային “կամ” աշխատում հետևյալ ձևով

0 կամ 0 հավասար է 0;
0 կամ 1 հավասար է 1;
1 կամ 0 հավասար է 1;
1 կամ 1 հավասար է 1;

✓ & բիթային “և” աշխատում հետևյալ ձևով

0 և 0 հավասար է 0;
0 և 1 հավասար է 0;
1 և 0 հավասար է 0;
1 և 1 հավասար է 1;

✓ ^ բիթային բացառում ըստ “կամ” ի աշխատում հետևյալ ձևով

0 ^ 0 հավասար է 0;
0 ^ 1 հավասար է 1;
1 ^ 0 հավասար է 1;
1 ^ 1 հավասար է 0;

✓ <<, >> բիթային շեղում աջ կամ ձախ

հնարավորություն է տալիս ըստ նշված տասական թվի չափի, 2 ական կոդը տեղաշարժել աջ կամ ձախ: Երկու դեպքում էլ աջից կամ ձախից կատարվում է տվյալի կորուստ, անհայտ բիթերում լրացնելով զրոներ:

✓ ~ բիթային ժխտում

Նշված երկուական կոդում զրոները փոխվում է մեկերով, իսկ մեկերը զրոներով: Պարզ է, եթե դրական թիվ է ժխտվում, ապա նրա նշանը նույնպես ժխտվում է, որի պատճառով թիվը դառնում է բացասական: Համակարգիչը բացասական թվերը ներկայացնում է իր ձևով:

```
//.... 0000 0001  = 1
//... .0000 0000  = 0
//.... 1111 1111  = -1
//.... 1111 1110  = -2
//.... 1111 1101  = -3
//.... 1111 1100  = -4
using System;
class Program
{
    static void Main(string[] args)
    {
        int a = 5;//..... 0000 0101
        int b = 3;//..... 0000 0011
        int c = a & b;// բիթային տրամաբանական "և" օպերացիա
        Console.WriteLine(c); //..... 0001 =1
        c = a | b;// բիթային տրամաբանական "կամ" օպերացիա
        Console.WriteLine(c); //..... 0111 = 7
        c = a ^ b;// բիթային տրամաբանական բացառում ըստ "կամ" օպերացիայի
        Console.WriteLine(c); //..... 0110 = 6
        c =  a>>1;// բիթային տրամաբանական շեղում աջ մեկ բիթ
        Console.WriteLine(c); //..... 0010 = 2
        c = a << 1;// բիթային տրամաբանական շեղում ձախ մեկ բիթ
        Console.WriteLine(c); //..... 1010 = 10
        c = ~b;//.... 1111 1100  = -4 // բիթային ժխտման օպերացիա
        Console.WriteLine(c);// -4  // ....1111 1100 = fc
        Console.WriteLine(string.Format("{0:x}", c)); // ffffffc
        //.... 0000 0001  = 1
```

```
        //... .0000 0000  =  0
        //.... 1111 1111  = -1
        //.... 1111 1110  = -2
        //.... 1111 1101  = -3
        //.... 1111 1100  = -4
    }
}
```

……

10.switch օպերատոր

[Шилдт-125]

- switch օպերատորը փոխարինում է if… else օպերատորին, եթե անհրաժեշտություն կա միաժամանակ ստուգել մեծ քանակով պայմաններ:

```
using System;
class Program
{
        static void Main(string[] args)
        {
            int a;
             for (; ;)
             {
                 Console.WriteLine("tiv = ");
                 a = Convert.ToInt32(Console.ReadLine());
                 switch (a)
                 {
                     case 1: Console.WriteLine("1111111");
                         goto aa;
                         break;
                     case 2:
                     aa: Console.WriteLine("2222222");
                         break;
                     case 3: Console.WriteLine("3333333");
                         break;
                     default: Console.WriteLine("barev");
                         break;
                 }
             }
        }
}
```

- Հաջորդ օրինակը ցույց է տալիս, թէ ինչպես կարելի է case ում ոչինչ չգրել: Այս դեպքը թույլատրելի է և հնարավորություն է տալիս մի քանի պայմաններ կատարեն նույն գործողությունը:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int n = 2;
        switch (n)
        {
            case 1:
            case 2:
            case 3:
                Console.WriteLine("barev");
                break;
        }
    }
}
```

11.Տիպերի վերափոխման օպերատոր ( )

[Шилдт-89]

- Տիպերի վերափոխումը, որը օգտագործում է ( ) փակագծերը, կարելի է անվանել օպերատոր, այլ ոչ թէ օպերացիա, իր կարևորության և բարդության հետ կապված: Տվյալները ստանալով տիպեր, ծրագրի կատարման ժամանակ անցնում են տարբեր թույլատրելի տարածքներով: Նշված տարածքները կարող են լինել տարբեր տիպեր: Անցումները տարածքներում կարող են լինել մեծից փոքր և փոքրից մեծ: Փոքրից մեծ անցումը բնական է կատարվում առանց որևէ միաջամտման, իսկ մեծից փոքրի դեպքում, հնարավոր է տեղի ունենա տվյալների կորուստ: Դրա համար, եթե ծրագրավորողը օգտագործում է կլոր փակագծեր ( ) տիպերի վերափոխարկաման օպերատորը, ապա թարգմանիչը շարունակում է աշխատանքը, չհամարելով այն սխալ:
- Նշենք, որ տիպերի վերափոխարկում կարելի է կատարել ոչ բոլոր տիպերի միջև (օրինակ bool ից int հնարավոր չէ), այլ կարելի է միայն միևնույն ընտանիքի տիպերի միջև (օրինակ բոլոր թվային տիպերի միջև - byte, sbyte, short, ushort, int, uint, long, ulong, float, double, decimal):

……

12.C/C# պարզ տարբերություններ

[Шилдт-97, 121]

Նշենք C# -ի որոշ նորույթ առավելություններ C/C++ լեզվի նկատմամբ, նրա պրոցեդուրային մասով:

- Verbatim string

```
WriteLine ( @"text
                    text 2");
```

կարելի է գրել մի քանի տող, որը պահում է ընդհանուր տեսքը և տողում թույլատրում է "Enter":

- C - թույլատրում է չարժեքավորված լոկալ փոփոխականի հետ աշխատել, իսկ C# - ոչ;
- C - թույլատրում է բլոկում փոփոխականի անունը համընկնի բլոկից դուրս, ուղղակի նա ծածկում է նրան և եթե գլոբալ է, կարելի է միայն :: կետի օգնությամբ տեսանելիությունը մեծացնել: C# ում սա չի թույլատրվում:
- C- ում % թվաբանական գործողությունը միայն ամբողջ թվերին է թույլատրվում: C#-մ նաև կոտորակային թվերին:
- C# -ում տրամաբանական արտահայտությունը վերադարձնում է միայն true կամ false : Այդ դեպքում (==) – հնարավոր չէ խառնել (=) – ի հետ, որովհետև պայմանը միշտ վերադարձնում է true կամ false:
- Տրամաբանական արտահայտության կրճատված (&&,||) և ոչ կրճատված (&,|) պայմաններ: (&,|) համարվում է բիթային օպերացիա և կախված արտահայտությունից միաժամանակ կարող է դառնալ ոչ կրճատված տրամաբանական օպերացիա: Ոչ կրճատված նշանակում է տրամաբանական արտահայտության բոլոր անդամները ստուգվում են, եթե նույնիսկ արտահայտության արդյունքը հայտնի է լինում ավելի շուտ:

```
static void Main()
{
    int a = 9;
    // if(4>8&&9<0&6>0&&88==88&&a++>8)
    if(4>8&9<0&6>0&88==88&a++>8)
        Console.WriteLine("barev");
    Console.WriteLine(a);
}
```

- switch – ի պարամետր C# -ում ընդունում է նաև string և enum; C# break - ը համարվում պարտադիր: Եթե break չկա, ապա goto case n:
- for – ի պայմանում տեսանելիության տիրույթը նույնպես ընդգրկվում է է բլոկի սահմանում: C- ում այդպես չէ: Օրինակում նշվածը չի բերում սխալի, իսկ C լեզվում երկրորդ int հայտարարումը պետք է հանել:

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

```
        {
            for (int i = 0; i < 8; i++)
            {              }
            for (int i = 0; i < 8; i++)
            {             }
        }
```

- int @for = 55; int @new = 19; Կարելի է @ նշանի օգնությամբ հատուկ անունը օգտագործել որպես փոփոխականի անուն:

……..……..……..……

13.Զանգված

[Троелсен-160; Нейгел- 184; Нэш-248; Рихтер-406]

- Զանգվածը տվյալների կազմակերպման այնպիսի ձև է, երբ անհրաժեշտ է միևնույն տիպի տվյալները պահել համարակալված (կոլեկցիայի ձևով): Զանգվածների հետ աշխատանք կատարվում է **ինդեքսով**: Հայտարարման ձևը՝

int [ ] myArray = new int [5]; վեկտոր

int [ ] myArray = { 2, 4, 6, 8, 10 }; վեկտորի հայտարարում և վերագրում

int [,] myArray = new int [8,8]; մատրիցա

- Զանգվածը հայտարարման ժամանակ ընդունում է ըստ տիպի լռելիության արժեքներ:
- Զանգվածի սահմանները գտնվում են հսկման տակ և չափը նշվում է **Length** հատկանիշի միջոցով:

```
using System; //պատահական թվերի վեկտոր
class M
{
    static void Main()
    {
            int[] A = new int[8];
            Random r = new Random();
            for (int i = 0; i < 8; i++)
                A[i] = r.Next(10);
            for (int i = 0; i < A.Length; i++)
                Console.Write(A[i] + "  ");
    }
}
//--------------------

using System; // մաքսիմումի փնտրում
class M
{
    public static void Main()
    {
        Random r = new Random();
        int[] A = new int[10];
        for (int i = 0; i < 10; i++)
            A[i] = 50-r.Next()%100;
        int max=A[0];
        for (int i = 0; i < 10;i++ )
            Console.Write(A[i] + " ");
        for (int i = 0; i < 10; i++)
            if (A[i] > max)
                max = A[i];
        Console.WriteLine();
        Console.WriteLine("max = " + max);
    }
}
```

```
//--------------------

using System; // վերադասավորում
class M
{
    static void Main()
    {
        int n=16;
        int[] A = new int[n];
        Random r = new Random();
        for (int i = 0; i < n; i++)
            A[i] = r.Next(100);
        for (int i = 0; i < n; i++)
            Console.Write(A[i] + "  ");
        Console.WriteLine();
        int a;
        for (int v = 0; v < n;v++ )
            for (int i = 0; i < n - 1; i++)
            {
                if (A[i] > A[i + 1])
                {
                    a = A[i];
                    A[i] = A[i + 1];
                    A[i + 1] = a;
                }
            }
        for (int i = 0; i < n; i++)
            Console.Write(A[i] + "  ");
        Console.WriteLine();
    }
}
//Տնային հատվածը
                    a = A[i];
                    A[i] = A[i + 1];
                    A[i + 1] = a;
իրականացնել առանց a փոփոխականի
//--------------------

using System; //մատրիցայի  ցուցադրում
class M
{
    static void Main()
    {
        int[,] A = new int[8,8];

        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
                Console.Write(A[i, j]+" ");
            Console.WriteLine();
        }
    }
}
//--------------------

using System; //մատրիցայի  ցուցադրում անհրաժեշտ “պրոբելով”
class Program
{
    static void Main(string[] args)
```

```
    {
        int n = 8;
        Random r = new Random();
        int[,] A = new int[n, n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                A[i, j] = r.Next(1, 100);
        Console.WriteLine();
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < n; j++)
                if (A[i, j] > 9)
                    Console.Write(A[i, j] + " ");
                else
                    Console.Write(A[i, j] + "  ");
            Console.WriteLine();
            Console.WriteLine();
        }
    }
}

//---------------------

using System; // նավակի հարվածներ
class M
{
    static void Main()
    {
        int[,] A = new int[8,8];
        Console.Write("navi y= ");
        int a = Convert.ToInt32(Console.ReadLine()); // կոորդինատ i
        Console.Write("navi x= ");
        int b = Convert.ToInt32(Console.ReadLine()); // կոորդինատ j
        for (int k = 0; k < 8; k++)
        {
            A[a-1, k] = 1;
            A[k, b-1] = 1;
        }
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
                Console.Write(A[i, j]+" ");
            Console.WriteLine();
        }
    }
}

// (տնային) ցուցադրել փղի հարվածները մատրիցայում
// (տնային) ցուցադրել ձիու հարվածները մատրիցայում
// (տնային) ցուցադրել թագուհու  հարվածները մատրիցայում
// (տնային) Լուտո 36: 9x6 մատրիցայում պատահական ձևով լրացնել 36 չկրկնվող թվեր:

//-----------------------

using System; // շախմատ, փղի քայլեր
class M
{
    static void Main()
    {
```

```
        int[,] a = new int[8, 8];
        Console.Write("x=");
        int x = Convert.ToInt16(Console.ReadLine());
        Console.Write("y=");
        int y = Convert.ToInt16(Console.ReadLine());
        int ii = 0;
        int jj = 0;
        while (x + ii < 8 && y + jj < 8)
        {
            a[x + ii, y + jj] = 1;
            ii++;
            jj++;
        }
        ii = 0;
        jj = 0;
        while (x - ii >= 0 && y - jj >= 0)
        {
            a[x - ii, y - jj] = 1;
            ii++;
            jj++;
        }
        ii = 0;
        jj = 0;
        while (x + ii < 8 && y - jj >= 0)
        {
            a[x + ii, y - jj] = 1;
            ii++;
            jj++;
        }
        ii = 0;
        jj = 0;
        while (x - ii >= 0 && y + jj < 8)
        {
            a[x - ii, y + jj] = 1;
            ii++;
            jj++;
        }
        for (int i = 0; i < 8; i++)
        {
            for (int j = 0; j < 8; j++)
                Console.Write(a[i, j] + " ");
            Console.WriteLine();
        }
        Console.ReadKey();
    }
}
//-----------------------

using System; // Loto 36
class Program
{
    static void Main(string[] args)
    {
        Random r = new Random();
        int[,] A = new int[4, 9];
        int[] V = new int[36];
        int k = 1;
        int m = 0;
        for (int i = 0; i < 4; i++)
            for (int j = 0; j < 9; j++)
```

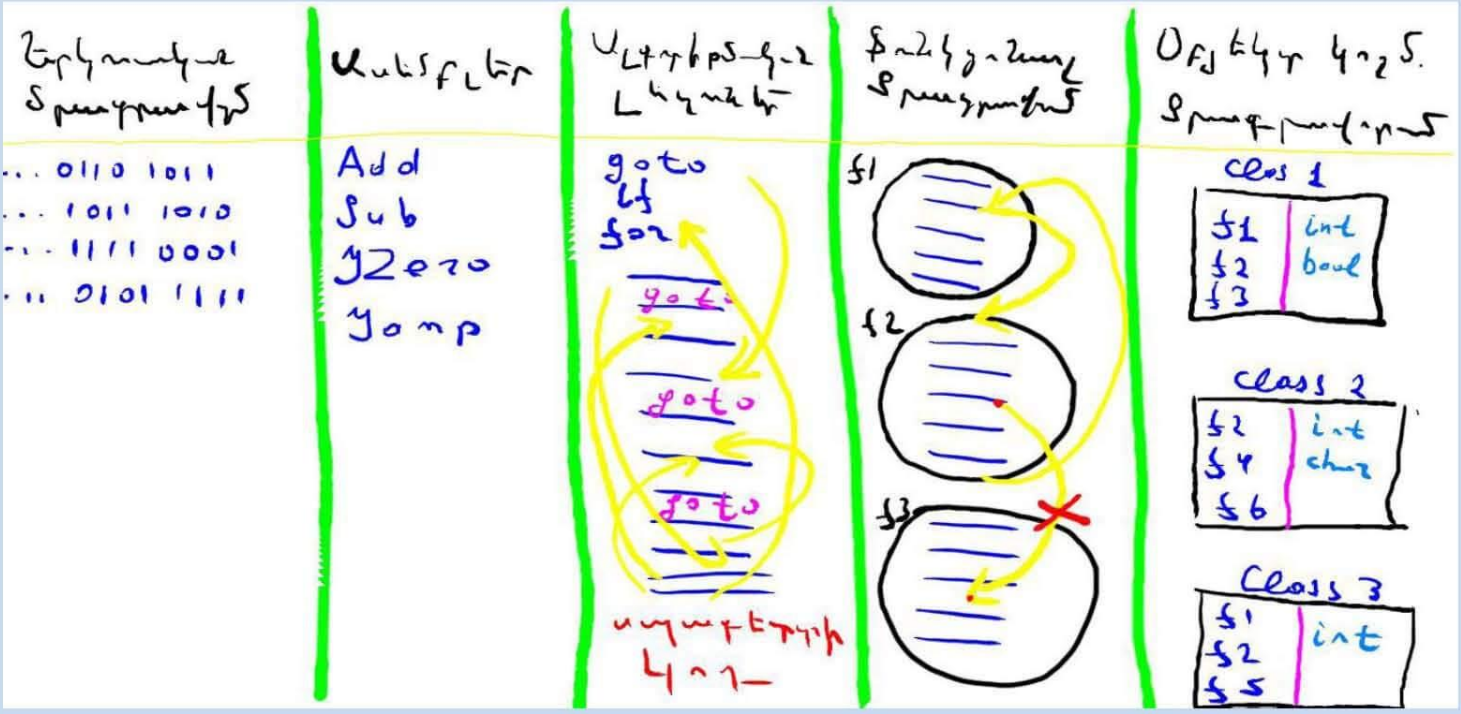
```
            {
            gg:
                while (true)
                {
                    k = 1 + r.Next() % 36;
                    for (int ii = 0; ii < m; ii++)
                        if (k == V[ii])
                            goto gg;
                    A[i, j] = k;
                    V[m] = k;
                    m++;
                    break;
                }
            }
        for (int i = 0; i < 4; i++)
        {
            for (int j = 0; j < 9; j++)
                if (A[i, j] > 9)
                    Console.Write(A[i, j] + " ");
                else
                    Console.Write(A[i, j] + "  ");
            Console.WriteLine();
            Console.WriteLine();
        }
    }
}
```

Տնային մատրիցաներից

- Թամբային կետ: Տրված է ամբողջթվային MxN չափի մատրիցը: Որոշել և արտածել մատրիցի այն տարրը, որը իր տողում մեծագույնն է, իսկ սյունում փոքրագույնը: Եթե այդպիսի տարր չկա, ապա արտածել “NO” հաղորդագրությունը:
- Տրված է NxN չափի քառակուսային մատրից: Առանց լրացուցիչ մատրից կիրառելու տեղերով փոխել $A_{1,1}$ և $A_{N,N}$, $A_{1,2}$ և $A_{N,N-1}$ և այլն տարրերը, այսինքն օժանդակ անկյունագծի նկատմամբ իրականացնել հայելային արտապատկերում: Արտածել ստացված մատրիցը:
- Տրված է MxN չափի մատրից: Արտածել մատրիցի կենտ համարով տողերի տարրերը ձախից աջ հաջորդականությամբ, իսկ զույգ համարովները՝ աջից ձախ հաջորդականությամբ:
- Տրված է MxN չափի մատրիցը: Հաշվել և արտածել մատրիցի ամեն տողերի փոքրագույն տարրերից մեծագույնի արժեքը:
- Տրված է A մատրիցա: Գտնել B վեկտորի տարրերը, որտեղ b_i - ն ստացվում է A մատրիցայի i - րդ տողի առաջին բացասական տարրից հետո եկող անդամների գումարից:

Ֆունկցիոնալ ծրագրավորում

14. Ծրագրավորման լեզուների զարգացման էտապները



- Ծրագրավորումը սկզբից եղել է **երկուական** տեսքի հրամաններով: Ծանրագույն ժամանակներ: Աշխատանք մեքենայական ձևով: Արագ անհրաժեշտություն առաջացավ մարդկայնացնել ծրագրավորումը և այն բերեց լեզվի հնարման անհրաժեշտության: Այն կոչվեց **ասեմբլեր** լեզու: Ասեմբլերի ամեն հրաման համապատասխանում է երկուական լեզվի մեկ հրամանի: Պարզ է, լեզվի առկայությունը պահանջեց թարգմանիչի անհրաժեշտության և այսպես առաջացավ ասեմբլեր թարգմանիչը:
- Հաջորդ էտապը առաջացավ արտադրողականության բարձրացման բնական պահանջի վրա: Գյուղատնտեսությունում այն կոչվում է ռենտայի օրենք - մեկ կարտոֆիլ զցիր ութը ստացիր: Ծրագրավորումը այն օգտագործեց հետևյալ ձևով - մեկ հարմանը կկատարի մկից ավել մեքենայական հրամաններ: Հաջորդ լեզուները անվանվեցին բարձր կարգի լեզուներ կամ **ալգորիթմական լեզուներ,** իրենց հետ բերելով սեփական թարգմանիչները:
- Ինչ է **ֆունկցիոնալ** ծրագրավորում: Նա անհրաժեշտություն էր ալգորիթմականից հետո: Քանի, որ ծրագրավորման խնդիրները անընդհատ պահանջում են ավելի շատ հրամանների առկայություն, ապա ալգորիթմական լեզուները ծրագրի հրամանների խճճման պատճառով մի պահից սկսած չէին կարող բավարարել խնդրի լուծման աշխատանքային պահանջները: Նույնիսկ ընդունելի անուն ստացավ պրոբլեմը, որը անվանվեց "**սպագետի**" կոդ: Խնդրի լուծման համար առաջարկվեց ծրագրի հրամանները խմբավորել, ասենք ֆունկցիաների ձևով և ստանալ ոչ միայն որոշ թրիչքների խզում, այլ նաև կոդի բազմակի օգտագծման հնարավորություն ֆունկցիաների ձևով: Ֆունկցիոնալ ծրագրավորումը ստացավ տարբեր անվանումներ և զարգացումներ, որոնցից են կառուցվածքային կամ մոդուլային անվանումները: Պետք է հիշել մի շատ կարևոր աքսիոմա, - լեզուները առաջանաում են, որովհետև հին ձևով անհնար է "ապրել", ծրագրի հրամանների անվերջ աճման պահանջի պատճառով: Ցանկացած նոր ետապ **"քաոսից"** դուրս գալու հնարք է: Այլ խոսքով ամեն էտապ պայքարում է **էնտրոպիայի** դեմ: (էնտրոպիա – քաոսի չափման միավոր):
- Հաջորդ էտապը կոչվում է **օբյեկտ-կողմնորոշված** ծրագրավորում: Այն բարձրացնում է ծրագրավորման **աբստրակցիան**: Քանի, որ ֆունկցիոնալ ծրագրավորումը հրամանների մեծ քանակի դեպքում բերում է խճճում ֆունկցիաների մակարդակով: Աներաժեշտություն առաջացավ

այդ ֆունկցիաները **պարփակել** ծրագրավորման մի այլ միավորի մեջ, որը անվանվեց **կլաս** (դաս): Միավորումը իրականացվեց նորամուծությամբ, այն է, ֆունկցիաների հետ ընդգրկվեցին (իրենց կանայք) նաև տվյալներ:

- Չպետք է մոռանալ, որ այժմ ծրագրավորումը գտնվում է օբյեկտ կողմնորոշված տիրույթում, սակայն կարելի է ավելացնել, որ Microsoft -ի կողմից առաջարկված **ղեկավարման կոդի** միջավայրում ծրագրավորումը իր ուղղությունն է հարթում նոր ժամանակներում: Microsoft –ի նոր "պրոյեկտի" անունը կոչվում է .NET կամ **dotNET**:

……..…….

15.Մեթոդներ

[Шилдт-155]

Մեթոդի պարամետրեր, արգումենտներ *[Шилдт-162]*

Մեթոդի վերադարձվող արժեք *[Шилдт-159]*

return օպերատոր *[Шилдт-158]*

- Ծրագրի ֆունկցիոնալ գործողությունների շարքին է պատկանում մեթոդը: Պարզագույն ներկայացման ձևը՝

```
void ff()
{
        // հրամաններ
}
```

Որտեղ մեթոդը ունի անուն ff, վերադարձվող արժեք `void` և կլոր փակագծեր ( ): `void` գրվում է այն դեպքում, երբ մեթոդը ոչինչ չի վերադարձնում:

- Ընդանրապես ցանկալի է և գործողության հնարավորությունները մեծ են, եթե մեթոդը կարող է ստանալ տվյալներ, որոնք կոչվում են մեթոդի պարամետրեր և եթե մեթոդը վերադարձնում է արժեք: Մեթոդը պարամետրեր ստանում է կլոր փակագծերում և արժեք վերադարձնում է `return` հրամանով: Այդ դեպքում անպայման նշվում է վերադարձվող արժեքի տիպը (int, bool double), որը գրվում է մեթոդի անունից ձախ և պետք է գտնվի համաձայնության մեջ `return` հրամանի հետ:

```
int ff(int a, char c)
{
       // հրամաններ
       return   հրաման; // int տիպ պետք է լինի
}
```

- Ծրագիրը իր ստարտը սկսում է `Main()` մեթոդի կատարումով, որը ենթարկվում է օպերացիոն համակարգի կանչին:

```
class Program
{
    static void Main()
    {
    }
}
//-------------------------------------------------

using System; // պարամետր և վերադարձվող արժեք
class A
{
        int a;
        int b;
        char c;
        public  int ff(int aa,char cc, int bb)
        {
            System.Console.WriteLine("barev !");
            a = aa;
            b = bb;
```

```
            c = cc;
            Console.WriteLine(c);
            return a + b;
        }
    }
    class Program
    {
       static void Main()
        {
            A ob = new A();
            ob.a = 333; // error !
            int sum=ob.ff(333,'W',555);
            Console.WriteLine("gumar = " +sum);
        }
     }
}
//-----------------------------------------------
```

- **return** հրաման կարելի է օգտագործել նաև void ոչինչ չվերադարձվող արժեքի հետ: Այն չունի հաջորդող տվյալ կամ հրաման և ուղղակի դուրս է բերում ծրագիրը մեթոդից:

```
using System;
class A
{
        public void f()
        {
            return;
            Console.WriteLine("barev");
        }
}
class Program
{
        static void Main(string[] args)
        {
            A ob = new A();
            ob.f();
        }
}
//-----------------------------------------------
```

- Պայմանավորվենք: Եթե մեթոդը նկարագրվում է, ապա այն ունի պարամետրեր տվյալներ ստանալու համար: Եթե մեթոդը կանչվում է, ապա պարամետրերը կկոչվեն արգումենտներ:
- Մեթոդը արգումենտում կարող է ստանալ այլ մեթոդ, եթե արգումենտի մեթոդի վերադարձվող արժեքի տիպը համընկնում է մեթոդի պարամետրի տիպին:

```
using System;
class A
{
    public void f(int a) //մեթոդի նկարագրություն
    {
        Console.WriteLine(a);
    }
    public int f2()
    {
        return 2015;
    }
}
class Program
{
    static string f()
```

```
    {
        return "albdarb.com";
    }
    static void Main(string[] args)
    {
        A ob = new A();
        ob.f(ob.f2()); //մեթոդի կանչ
        Console.WriteLine(f());
    }
}

// մաքսիմումի ծրագիրը մեթոդներով
using System;
class Program
{
    static int[] A = new int[10];
    static void Rand()
    {
        Random r = new Random();
        for (int i = 0; i < 10; i++)
            A[i] = r.Next(100);
    }
    static void List()
    {
        for (int i = 0; i < 10; i++)
            Console.Write(A[i] + " ");
    }
    static int Max()
    {
        int m = A[0];
        for (int i = 0; i < 10; i++)
            if (A[i] > m)
                m = A[i];
        return m;
    }
    static void Main(string[] args)
    {
        int m;
        Rand();
        List();
        m = Max();
        Console.WriteLine();
        Console.Write("maximum = " + m);
        Console.WriteLine();
    }
}

// (տնային) վերադասավորման ծրագիր մեթոդներով
//-----------------------------------------------
```

// Սորտավորման ծրագիրը մեթոդների միջոցով

- Օրինակում ցուցադրվում է, թէ ինչպես մեթոդը զանգվածի արժեքները վերադարձնում է պարամետրի միջոցով:

```
using System;
class AA
{
    //int[] A = new int[10];
    void frand(int[] A)
    {
```

```
        Random r = new Random();
        for (int i = 0; i < 10; i++)
            A[i] = r.Next(100);
    }
    void sort(int[] A)
    {
        for (int k = 0; k < 10; k++)
            for (int i = 0; i < 9; i++)
                if (A[i] > A[i + 1])
                {
                    A[i + 1] = A[i] + A[i + 1];  // տվյալների փոխատեղում
                    A[i] = A[i + 1] - A[i];      // առանց միջանկյալ
                    A[i + 1] = A[i + 1] - A[i]; // փոփոխականի
                }
    }
    void output(int[] A)
    {
        for (int i = 0; i < 10; i++)
            Console.Write(A[i] + " ");
        Console.WriteLine();
    }
    public void f()
    {
        int[] A = new int[10];
        frand(A);
        output(A);
        sort(A);
        output(A);
    }
}
class Program
{
    static void Main(string[] args)
    {
        AA ob = new AA();
        ob.f();
    }
}
```

16.Գերբեռնված մեթոդներ

[Шилдт-235]

- Ինչպես գիտենք տվյալ տեսանելիության տիրույթում նույն անունով փոփոխական չի թույլատրվում հայտարարել: Նույնը չի վերաբերվում մեթոդներին: Կարելի է նույն անունով մեթոդ ստեղծել, եթե կա հետևյալ սահմանափակումները: Մեթոդների պարամետրերը պեք է լինեն տարբեր: Կամ ավելի ճիշտ մեթոդները ունենան տարբեր քանակի պարամետրեր, կամ եթե նույն քանակի են , ապա տարբերվեն տիպով: Վերադարձվող արժեքի տիպը որոշիչ դեր չի խաղում:
- Գերբեռնված մեթոդները հնարավորություն են տալիս ինչպես ունենալ ավելի հասկանալի ծրագրային տեսք, այնպես էլ էֆեկտիվ օգտագործել համակարգչային ռեսուրսները:
- Օրինակ, `Console.WriteLine(…)` մեթոդը ունի 19 գերբեռնված տարբերակներ:

```
using System;
class A
{
        public int f(int a,int b)
        {
            return a + b;
```

```
        }
        public double f(double a, double b)
        {
            return a + b;
        }
}
class Program
{
        static void Main(string[] args)
        {
            A ob = new A();
            Console.WriteLine(ob.f(4.4, 6.7));
            Console.WriteLine(ob.f(4, 6));
        }
}
```

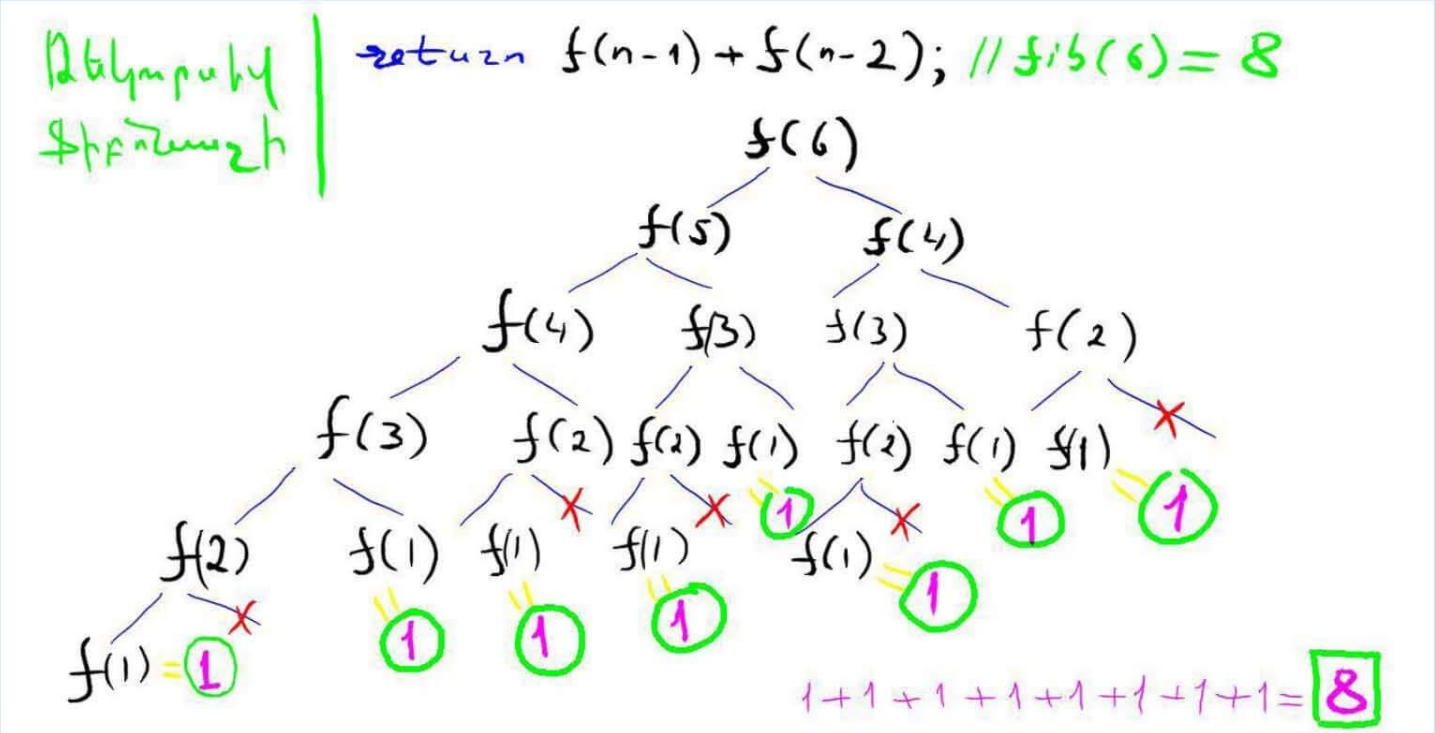
17.Ռեկուրսիա

[Шилдт-257]

- Ռեկուրսիան, դա ծրագրավորման հնարավորություն է, երբ մեթոդը կանչում է ինքն իրեն: Այս ձևով կարելի է ապահովել ցիկլի աշխատանքի մի այլ ձև: Եթե միջոցներ չձեռնարկվի, ապա մեթոդը իրեն կկանչի ավներջ, կամ ավելի ճիշտ մինչև հատկացված հիշողության (ստեկ հիշողության) լցվելը: Հիմանականում կիրառվում է մեթոդի ռեկուրսիվ կանչը մեթոդի կանչը հսկող փոփոխականի փոքրացման միջոցով:

```
using System; // ռեկուրսիվ անվերջ կանչ
class A
{
        public void f()
        {
            Console.Write("barev");
            f();
        }
}
class Program
{
        static void Main(string[] args)
        {
            A ob = new A();
            ob.f();
        }
}

using System; // ռեկուրսիվ 5 բարև
class Program
{
    static void Main(string[] args)
    {
        int n = 5;
        f(n);
    }
    static void f(int n)
    {

        if (n > 0)
        {
            Console.WriteLine("barev");
            n--;
```

```
            f(n); // կամ
            // f(--n);
            // f(n--); // Ինչու ճիշտ չի աշխատում
        }
        else
            return;
    }
}
//-------------------------------------

using System; // թվի ֆակտորիալ ռեկուրսիայով
class Program
{
    static void Main(string[] args)
    {
        for (; ;)
        {
            Console.Write("neretseq mutq em uzum, tiv = ");
            int a = Convert.ToInt32(Console.ReadLine());
            int rezult = fact(a);
            if (a == 0)
                return;
            Console.WriteLine(rezult);
        }
    }
    static int fact(int n)
    {
        if (n == 0)
            return 1;
        return n * fact(n - 1);
    }
}

// Տնային - ֆիբոնաչի շարքի (1 1 2 3 5 8 13 21 34 …….) անդամի որոշումը ռեկուրսիվ մեթոդով

using System; // ֆիբոնաչի ռեկուրսիայով
class Program
{
    static public int f(int n)
    {
        if (n > 2)
            return f(n-1)+f(n-2);
        else
            return 1;
    }
    static void Main(string[] args)
    {
        for (; ;)
        {
            int a = Convert.ToInt32(Console.ReadLine());
            Console.Write("fib"+a + " =");
            Console.WriteLine(f(a));
        }
    }
}
```



✓ Ռեկուրսիվ f(2) կամ f(1) վերադարձնում են մեկ արժեք: Նրանցից հետո չի շարունակվում հաշվարկը:

18.Լոկալ փոփոխականներ, բլոկ { }, տեսանելության տիրույթ

[albdarb]

- Ցանկացած մեթոդի փոփոխականներ կոչվում են լոկալ փոփոխականներ: Նրանք անպայման պետք է արժեքավորվեն մինչև օգտագործվելը: Լոկալ փոփոխականները վերանում են մեթոդի ավարտից հետո (թիթեռի կյանք): Այդ փոփոխականները պահվում են հիշողության այն մասում, որը կոչվում է **ստեկ** հիշողություն: Ստեկը հիշողության տրամաբանական կազմակերպման ձև է, որտեղ տվյալները պահվում և կարդացվում են հատուկ ձևով – հաջորդաբար ներառված տվյալներից "առաջինը կարդում ենք վերջում, իսկ վեջինը կարդում ենք առաջինը" սկզբունքով (ավտոմատի "մագազինի" նման): Ստեկ հիշողությունը նույնպես գտնվում է RAM ում: Այն ձևավորվում է ծրագրի թարգմանման ժամանակ: Նրա տարածքը սահմանափակ է: Աշխատում է LIFO սկզբունքով: Last in First Out:
- Մեթոդի ցանկացած մասում կարելի է դնել ձևավոր փակագծեր { }: Այն կոչվում է բլոկ: Նույնիսկ պայմանական օպերատորների և ցիկլերի ձևավոր փակագծերը բլոկներ են: Նրանք ստեղծում են տեսանելիության տիրույթ: Ներքին բլոկից դեպի արտաքին բլոկը տեսանելի է, իսկ հակառակը, ոչ:

```
using System;
class Program
{
        static void Main(string[] args)
        {
            int x;
            // Console.WriteLine(x);// error // արժեքվորված չէ
            int a = 99;
            {   // բլոկ
                // int a = 11; // error // int a կրկնվում է
                // Console.WriteLine(b); // error // տեսանելության տիրույթում չէ
                int b = 88;
                {
```

```
                    int c = 77;
                    Console.WriteLine(a);
                    Console.WriteLine(b);
                    Console.WriteLine(c);
                }
            }
        }
}
//-------------------------------------

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

            for (int i = 0; i < 3; i++)
            //{
                int a = 999;  // error
            //}
        }
}
```

19.Մեթոդ և ստեկ հիշողություն

[albdarb]

- Երբ կանչվում է մեթոդը, ապա կատարվում է հետևյալ գործողությունների հաջորդականությունը.

1. Ստեկում հիշվում է մեթոդի կանչի հաջորդ հասցեն ծրագրի հետագա աշխատանքի համար:
2. Ստեկում հատկացվում է վրադարձվող արժեքի տիպի չափով տեղ, `return` օպերատրի տվյալի համար:
3. Ստեկ է ուղղարկվում մեթոդի պարամետրերը:
4. Ստեկում տեղադրվում է մեթոդի բոլոր լոկալ փոփոխականները:

- Մեթոդի տվյալները ինչ հաջորդականությամբ գրանցվում են ստեկում, հակառակ հաջորդականությամբ ջնջվում են:

// Տնային, կիրառել մեթոդներ

1. ♞ Նշված դաշտից ձին քայլի պատահական ազատ դաշտերով լինելով մեկ անգամ:
2. ♛ Պատահական տեղադրել շախմատի տախտակի վրա թագուհիներ այնպես, որ միմյանց չհարվածեն:

……

C#sharp հիմունքներ

20.Կլասներ, օբյեկտներ

[Шилдт-147; Троелсен-185]

- Կլասները համարվում են օբյեկտ-կողմնորոշված ծրագրավորման հիմնային կառուցվածքային բլոկներ: Մինչև օբյեկտայինը, ծրագրավորումը անցել է հետևյալ էտապները: Երկուական ծրագրավորում (011101), ասեմբլեր, բարձր կարգի լեզուներ (Algol, Fortran, Cobol), պրոցեդուրային կամ ֆունկցիոնալ լեզուներ (C, Pascal), օբյեկտ կողմնորոշված լեզուներ (C++, Java, C#): Ցանկացած էտապից մյուսին անցնելու համար անհրաժեշտ է օբյեկտիվ պայմաններ: Այդպիսի պայմաններից է ծրագրային հրամանների անվերջ մեծացումը: Օբյեկտ-կողմնորոշված ծրագրավորումը դա հիմնային անցում է: Այն քառսից դուրս գալու հնարք է, որի հիմքում ընկած է մարդկային հասարակության մեծ փորձը բարդ խնդիրների լուծման համար: Այն մոդելավորվում էր որպես օբյեկտներ, շրջապատը և խնդիրները պարզեցնելու համար: Ստեղծվեց մարդկային լեզուն և խոսքը, որը պարունակում է ինչպես գործողություն(բայ) այնպես էլ առարկա (գոյական): Ահա այս մոդելը արտապատկերվել է ծրագրավորման զարգացման նոր էտապի մեջ և հետաքրքիրը կայանաում է նրանում, որ այն տվել է արդյունք: Ծրագրավորումը բերվեց մի վիճակի, երբ խնդիրը կատարվում է նկարագրությունների միջոցով: Որպես ծրագրավորման միավոր ստեղծվեց կլասներ, որոնց թույլատրվեց պահել և տվյալ և գործողություն միաժամանակ: Կլասը դարձավ մարդկային վարվեցողության արտապատկերողը բարդ խնդիրներ լուծելու համար:
- Ենթադրենք մոդելավորում ենք աթոռ կլասը: Որպես տվյալներ կարելի է ներկայացնել նյութերը, ասենք փայտ, “պլաստմասա”, “ռեզին”, երկաթ, կտորեղեն, “գուպկա” և այլն: Որպես գործողություն կարելի է ներկայացնել ամրացումներ, կոնֆորտացնող միջոցներ, նստելու հնարավորություն, պաշտպանական գործողության հնարավորություն (մի խփեք ուրիշի գլխին):
- Կլասը կատարեց նոր ծառայություն, այն թույլ է տալիս սահմանափակ քանակով ներդրված տիպերի հետ զուգահեր ստեղծել անսահմանափակ քանակությամբ նոր տիպեր: (user-defined types):
- Կլասի տեխնիկական կիրառման համար օգտագործվում է օբյեկտ գաղափարը: Կարելի է համարել, որ կլասը դա շաբլոն է, իսկ օբյեկտը նրա իրականացումը: Օրինակ, առանձնատան նախագիծը դա կլաս է, այպես ասած շաբլոն, իսկ շինարարությունից հետո առաջացած տունը, դա օբյեկտն է: Օբյեկտը մեծ նյութականություն է և ծրագրի տեսանկյունից, զբաղեցրած հիշողություն:
- Կլասը կարող է պարունակել տվյալներ (`int a`, char c, bool b այլն), որոնց անվանում են դաշտեր և գործողություններ (ֆունկցիաներ), որոնցից են մեթոդները, հատկանիշները, ինդեքսատրները, իրադարձությունները և այլն:
- Թարգմանման ժամանակ կատարվում է հիշողության հատկացում միայն ֆունկցիաների հրամանների համար, իսկ տվյալների համար, ոչ: Կլասի տվյալների համար հիշողություն հատկացվում է ծրագրի կատարման ժամանակ և միայն օբյեկտի ստեղծման դեպքում:
- Օբյեկտ ստեղծվում է հետևյալ ձևով՝ A ob = new A(); Կլասի անդամներին դիմում ենք օբյեկտ և կետի միջոցով՝ `ob.ff()` կամ `ob.a`: Կարող են լինել բազում օբյեկտներ տվյալ կլասի համար: Ցանկացած օբյեկտի ստեղծվելու ժամանակ կատրավում է տվյալ օբյեկտի տվյալների համար սեփական հիշողության առանձնացում: Սակայն օբյեկտները չունեն սեփական ֆունկցիաները (մեթոդները): Այդ ֆունկցիաները բոլորի օբյեկտների համար նույնն են:

```
class A
{
        int a; // տվյալ
        public void ff() //գործողություն
        {
           a = 44;
           System.Console.WriteLine(a);
        }
}
```

```
class M
{
        static void Main()
        {
            A ob = new A(); // օբյեկտի ստեղծում
            ob.ff();  // կլասի անդամի օգտագործում
        }
}
```

……

21. Թույլատրության մոդիֆիկատորներ

[Троелсен-207; Шилдт-210]

- Օբյեկտ-կողմնորոշված ծրագրավորումը առանձնանում է տվյալների պաշտպանվածության (encapsulation) մեծացման հնարավորությունով: Այն պահում է բնության էներգիայի պահպանման օրենքը: Տալիս է հնարավորություն ավելորդ միջամտումներից խուսափելու համար: (Հիշենք կրիայի պարփակվածությունը, որով նա գոյատևում է արդեն հարյուր միլիոնավոր տարիներ): Ինչպես ասում է Ստրաուստրուպը (C++ հեղինակ), պաշտպանվածությունը ուժեղացվում է ոչ թէ վնասարարի դեմ, այլ պատահական վնասի դեմ, որը ավելի շատ է տարածված ծրագրավորման ասպարեզում: Այն ապահովվում է թույլատրության մոդիֆիկատորների օգնությամբ:
- Թույլատրության մոդիֆիկատորներ (Access Modifiers)
  public
  private
  protected // կանցնենք ավելի ուշ
  internal // կանցնենք ավելի ուշ
  protected internal // կանցնենք ավելի ուշ
- class -ը և interface, struct, enum, delegate, որոնք կանցնենք ավելի ուշ, նույնպես կարող են ունենալ թույլատրություն, միայն public և internal:
  internal - նշանակում է ընթացիք հավաքագրման բլոկ:
  public - նշանակում է արտաքին հավաքագրման բլոկ:
  Լռելիությամբ նրանք նշանակվում են internal:
- **Կարևոր գաղափար:** Քանի, որ տվյալներ շատ կանոր են և նրանց վնասվելը (միտումնավոր կամ պատահական) թանկ է նստում, ապա ծրագրավորման մեջ ընդունված կարգ է, տվյալները փակել (private) և նրանց հետ աշխատելու համար ավելացնել բաց public մեթոդներ: Ծրագրավորման այս ձևը կոչվում է պարփակվածություն: (**encapsulation**):

Այսպիսով առաջանում է պաշտպանվածության մի ձև, որ եթե պետք է վնասել տվյալը, ապա պետք է ծրագրային փոփոխություն կատարել մեթոդում, որը այդքան էլ հեշտ չէ:

```
class A // encapsulation
{
    private int a; // պարփակված տվյալ
    public void f_read() //գործողություն գրել
    {
        a = 999;
    }
    public void f_write() //գործողություն կարդալ
    {
        System.Console.WriteLine(a);
    }
}
class M
{
    static void Main()
```

```
    {
        A ob = new A(); // օբյեկտի ստեղծում
        ob.f_read();    // կլասի փակ անդամի արժեքավորում
        ob.f_write();   // կլասի փակ անդամի օգտագործում
    }
}
```

……

22.Կոնստրուկտոր, լռելիությամբ կոնստրուկտոր

[Шилдт-147; Троелсен-188]

- Կլասներում շատ հաճախ անհրաժեշտ է լինում տվյալներին տալ նախնական արժեքներ: Այն հնարավոր է մեթոդի միջոցով և կանչել այդ մեթոդը յուրաքանչյուր օբյեկտի ստեղծման ժամանակ: Սակայն այդ գործողությունը հնարավոր նաև է կատարել հատուկ միջոցով, որ կոչվում է կոնստրուկտոր: Այն հնարավորություն է տալիս տվյալների նախնական արժեքավորումը **ավտոմատացնել** (չկանչելով մեթոդ):
- Այսպիսով, ցանկացած կլաս ունի մեթոդ, որի անունը **համընկնում է** կլասի անվան հետ: Այն կոչվում է կոնստրուկտոր և նախատեսված է կլասի դաշտերին (տվյալներին) **նախնական արժեքներ** վերագրելու համար: Կոնստրուկտորը աշխատում է, երբ **ստեղծվում է օբյեկտ**: Ավելին, կոնստրուկտորը ոչինչ չի վերադարձնում, նույիսկ void:
- Կոնստրուկտորը կարող է լինել փակ կամ բաց: Եթե կոնստրուկտորը փակ է, ապա հնարավոր չէ ստեղծել տվյալ կլասի օբյեկտը արտաքին միջավայրից: Պարզ է, որ կլասի անդամներին հնարավոր է դիմել այլ մեթոդներով, որոնցից են կլասի սկատիկ անդամները (կանցնենք հետո):
- Եթե կոնստրուկտորը չունի պարամետր, ապա կոչվում է լռելիությամբ կոնստրուտոր: Օրենք. Եթե ծրագրում ոչ մի կոնստրուկտոր չի նշված, ապա կոմպիլիտորը ինքն է ստեղծում իր լռելիությամբ կոնստրուկտորը:
- Օրինակում երևում է, թէ ինչպես չդիմելով որևէ մեթոդի ծրագիրը բարևում է և **a** դաշտին տալիս է 567 արժեք:

```
class A
{
    int a;
    public A()
    {
        a = 567;
        System.Console.WriteLine("barev");
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
    }
}
```

……

23.Պարամետրով և գերբեռնված կոնստրուկտորներ

[Шилдт-168; Троелсен-189]

- Քանի, որ կոնստրուկտորը մեթոդ է, ապա այն կարող է ունենալ պարամետրեր և գերբեռնվել:
- Օրենք. եթե կա պարամետրով կոնստրուկտոր, ապա լռելիությամբ կոնստրուկտոր ստեղծելը մնում է ծրագրավորողի վրա:
- Օրինակում երևում է լռելիությամբ կոնստուկտրի անհրաժեշտությունը:

```
using System;
```

```
class A
{
    int aa;
    char cc;
    public A(int a, char c)
    {
        aa = a;
        cc = c;
        Console.WriteLine(cc);
        Console.WriteLine(aa);
    }
    public A(int a)
    {
        aa = a;
        Console.WriteLine(aa);
    }
    public A() // Բացակայության դեպքում կտա  error !
    {
    }
}
class Program
{
    static void Main(string[] args)
    {
        A ob = new A(); //Ստիպում է լռելիությամբ կոնստրուկտոր գրել
        A ob2 = new A(2);
        A ob3 = new A(999, 'y');
    }
}
```

.....

24.Ինիցիալիզատորներ, կոմպոզիցիա, ագրեգացիա

[albdarb]

- Տվյալներին նախնական արժեքներ տալը կոչվում է ինիցալիզացիա: Ինիցալիզացիան կարելի է կատարել ինչպես կոնստրուկտորի միջոցով, այնպես էլ վերագրման օպերատորով, կլասի դաշտին արժեքավորելով հայտարաման ժամանակ: Սկզբից կատարվում է ինիցալիզատորը, նոր հետո կոնստրուկտորը: Ինիցիալիզատորը նույնպես աշխատում է օբյեկտի ստեղծման ժամանակ:
- Եթե որպես կլասի դաշտ (անդամ տվյալ) հանդես է գալիս այլ կլասի օբյեկտի ստեղծում, ապա դա կոչվում է կոմպոզիցիա: Եվ եթե օբյեկտի ստեղծման փոխարեն միայն հղում է վերագրվում, ապա կոմպոզիցան դառնում է ագրեգացիա:
- Կոմպոզիցիայի դեպքում կոնստրուկտորների կատարման հերթականությունը կատարվում հետևյալ ձևով - կզբից կոմպոզիցիան, հետո սեփական կոնստրուկտորը:

```
class A
{
    int a = 44;        // initializer
    B ob = new B();   // composition
    B ob2;            // agregatia
    public A()       // default constructor
    {
          System.Console.WriteLine("AAAAAAA");
    }
}
class B
{
     public B()
```

```
        {
            System.Console.WriteLine("BBBBBBBB");
        }
    }
    class M
    {
        static void Main()
        {
            A ob = new A();
        }
    }
```

- Ամենաքիչ հրամաններով ինիցիալիզացիա, կամ օբյեկտի հայտարարումը և ինիզալիզացիան մեկ հրամանով

```
class A
{
    public int a;
    public char c;
}
class Program
{
    static void Main(string[] args)
    {
        A o = new A { a = 1, c = 'C' };
        System.Console.WriteLine(o.a +" "+ o.c);
    }
}
```

……

25.Անունների տարածք, պսեվդոանուն

[Шилдт-513; Троелсен-494; Нейгел-92]

- Ծրագրային կոդի բարդացման հետ կապված կատարվում է խմբավորում անունի տարածքում, որը հնարավորություն է տալիս կոնֆլիկտներից խուսափել: Անունի տարածքը համարվում է պարփակվածության (encapsulation) ձև: Օրինակ, եթե մարդուն օբյեկտ համարենք և տվյալ անուն ազգանունով կարող են շատ մարդիկ լինել, ապա հասցեն կդառնա անունի տարածք, որը կտարբերակի այդ մարդկանց:
- Նայեք “Տերմինատոր 1” ֆիլմը և կհասկանաք անունի տարածքը:
- Անունի տարածքը հայտարարվում է **namespace** հրամանով: Անունի տարածքից օգտվում են **using** –ով, կամ անմիջական կցում են կլասի անվան հետ:

```
using System;
using System.Threading;
namespace barev
{
    class M
    {
        string ss = "barev";
        public void ff()
        {
            Console.WriteLine(ss);
        }
    }
}
namespace myprog
{
    class M
    {
        static void Main()
        {
```

```
            System.Console.WriteLine("Hello World");
            Thread.Sleep(1500);
            barev.M ob = new barev.M();
            ob.ff();
            Console.ReadKey();
        }
    }
}
```

.....

Անունների տարածքի պսեվդոանուն [Троелсен-497; Нейгел-94]

- Երբեմն անունի տարածքը ստացվում է երկար: Այն կարճացնելու համար օգտագործվում է պսեվդոանուն, որից կարելի է օգտվել ծրագրում:

```
using System;
using NN = Armenia.Yerevan.Darbinyan.program;
class M
{
    public static int Main()
    {
        NN.M ob = new NN.M();
        Console.WriteLine(ob.ff());
        return 0;
    }
}
namespace Armenia.Yerevan.Darbinyan.program
{
    class M
    {
        public string ff()
        {
            return this.GetType().Namespace;
        }
    }
}
```

.....

26.Անունի տարածք և global

[Шилдт-524; Уотсон-369]

- Տվյալ ծրագրային պրոյեկտի սահմաններում կարելի է ունենալ և անունի տարածքի առկայություն և առանց անունի տարածքի ծրագրի հատված: Հնարավոր է նշված ծրագրի տարբեր հատվածները ներառեն նաև միևնույն անունով կլաս: Այդ կլասները տարբերակելու համար օգտվում են global հրամանից:

```
using System;
class A
{
    public A()
    {
        Console.WriteLine("global A");
    }
}
namespace ConsoleApplication1
{
    class A
    {
        public A()
        {
```

```
            Console.WriteLine("my A");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            A a1 = new A();
            global::A a2 = new global::A();
        }
    }
}
```

.....

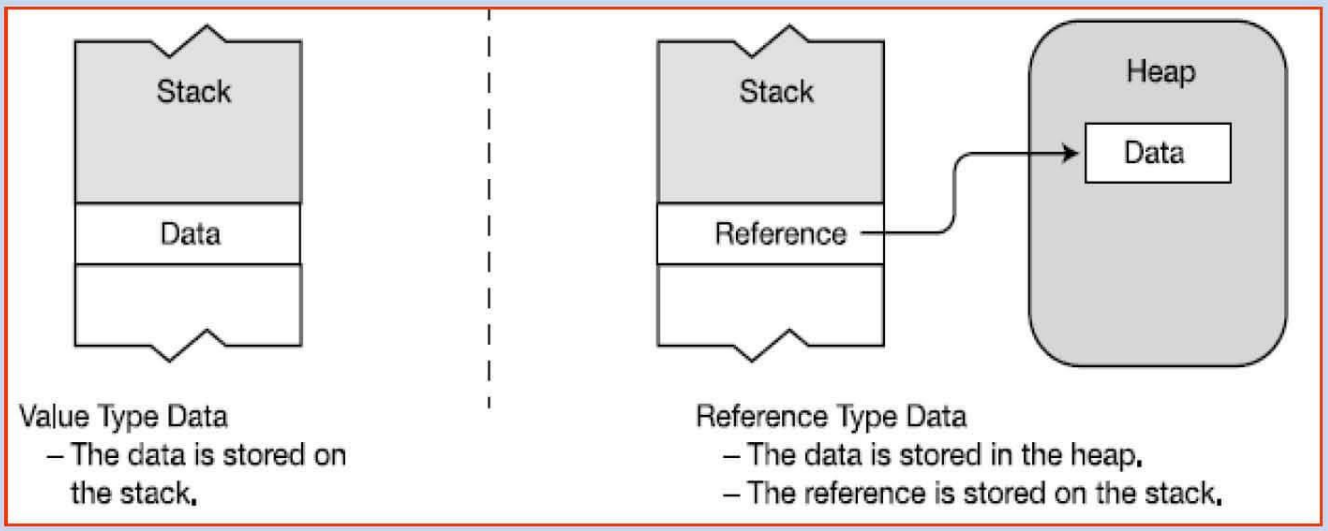
27.Չափային և հղիչային տիպեր

[Нейгел -371; Рихтер -129; Троелсен -174]

- C# լեզվում տիպերի բաժանման նոր մոտեցում է առաջարկվում: Տիպերը խմբավորվում են ըստ իրենց ֆիզիկական իրականացման չափանիշի – ստեկի և կույտի (դինամիկ հիշողություն):

Կրկնենք: Ստեկ (stack), դա հիշողության տրամաբանական կազմակերպման ձև է, որը աշխատում է հետևյալ մեթոդով – վերջի պահվածը կարդացվում է առաջինը: Ստեկը հատկացվում է թարգմանության ժամանակ և ազատվում է, երբ ծրագիրը հատում է բլոկի տեսանելիության սահմանը:
Դինամիկ հիշողություն կամ կույտ (managed heap), դա ազատ հիշողությունն է, որը ծրագրին հատկացվում է կատարման ժամանակ: Կույտը ազատվում է նույնպես ավտոմատ ձևով հատուկ միջոցներով:

- C# -ում տիպերը բաժանվում են երկու խմբի, չափային տիպերի (value type), որոնց համար տեղ հատկացվում է ստեկում և հղիչային տիպերի (reference type), որոնք տեղավորվում են դինամիկ հիշողությունում, կամ այսպես ասած կույտում:
  Չափային տիպեր են int, long, bool, float, char,... enum, struct:
  Հղիչային տիպեր են class, Array, interface, string, delegate, object:
- Այսպիսով, եթե ևս մեկ անգամ նշենք, չափայինի դեպքում օբյեկտը կամ փոփոխականը ստեղծվում է ծրագրի թարգմանման ժամանակ, ստեկում: Հղիչայինի դեպքում օբյեկտ ստեղծվում է ծրագրի կատարման ժամանակ, դինամիկ հիշողությունում, հղիչը թողնելով ստեկում:
- Հղիչայինի դեպքում ռեսուրսների օգտագործումը մեծանում է, որը բերում է արագության անկման: Չափայինի դեպքում ծրագիրը ավելի արագագործ է, իսկ ծրագրավորման հնարավորությունները սահմանափակ են:



```
using System;
struct S //կանցնենք հեռո
{
    int a; //stack
    string s; //heap
    public void f(int aa, string ss)
    {
        int b = 886; //stack
        string s2 = "albdarb"; //heap
        M ob = new M(); //heap
        S ob2 = new S(); //stack
    }
}
class A
{
    int a; //heap
    string s; //heap
    public void f(int aa, string ss)
    {
        int b = 886; //stack
        string s2 = "albdarb";//heap
        int[,] Z = new int[8,8]; //heap
        M ob = new M(); //heap
        S ob2 = new S(); //stack
    }
}
class M
{
    static void Main(string[] args)
    {
        S ob = new S();
        ob.f(555, "alb");
        A ob2 = new A();
        ob2.f(555, "alb");
    }
}
```

.....

28.boxing / unboxing

[Рихтер -123; Шилдт -370; Нейгел -212]

- Եթե տիպը դիտարկենք ծրագրի կատարման ժամանակային ընթացքում, ապա այն անցնում է տարբեր ձևափոխման սահմաններով: Տիպերը ավելի մեծ հնարավորությունից կարող է անցնել ավելի փոքրին և հակառակը: Անցումը համարվում է ավելի բարդ, եթե անցում կա չափայինից հղիչային և հակառակը: Այդպիսի անցումների մեխանիզմը կոչվում է boxing/unboxing: Առաջինի դեպքում չափային տիպը արհեստական ձևով տեղափոխվում է կույտ: Երկրորդի դեպքում կույտ տեղափոխված տիպը վերադարձվում է ստեկ: Օրինակ, boxing – ի անհրաժեշտություն կա, եթե մեթոդը չափային տիպ է ուղարկում մի այլ մեթոդի, որը ընդունում է միայն հղիչային տիպ:
- Նշված գործողությունները ապահովվում են արդեն գոյություն ունեցող միջոցներով: Դրանք են object տիպի օգտագործումը և տիպերի վերափոխման ( ) օպերացիան:

```
int aaa = 40;
object bbb = aaa; // boxing
int aaa = (int)bbb; // unboxing
```

.....

```
using System;
class M
{
```

```
    static void Main()
    {
        int a = 40;
        object b = a;
        b = 88;
        int c = (int)b;
        Console.WriteLine(a + "\n" + b + "\n" + c);
        //.....................................................................................
        // եթե կույտից վերադառնում ենք ոչ ճիշտ տիպով, ապա առաջանում է արտակարգ իրավիճակ,
        // որը կանցնենք ավելի ուշ:
        int s = 55;
        object ob = s;
        try  // կանցնենք հետո
        {
            string ss = (string)ob;
        }
        catch (InvalidCastException e)
        {
            Console.Write(e.ToString());
        }
    }
}
.................................................................

//example 2  // Рихтер,123(144)
using System.Collections;
struct Point   // Declare a value type.
{
    public int x, y;
}
class Program
{
    public static void Main()
    {
        ArrayList a = new ArrayList(); // կանցնենք հետո
        Point p; // Allocate a Point (not in the heap).
        for (int i = 0; i < 10; i++)
        {
            p.x = p.y = i; // Initialize the members in the value type.
            a.Add(p); // Box the value type and add the
                      // reference to the ArrayList.
        }
    }
}
.............................................................................................................
```

29.null, ?, ??

[Шилдт-695 ; Нейгел-211]

- Հղիչային տիպերին եթե ոչինիչ չենք վերագրում, ապա կարող ենք վրագրել `null` արժեք: Եթե չափային տիպը լոկալ տիպ է, ապա նրան մինչև օգտագործելը պետք է անպայման արժեք վերագրել:
- Կարելի է չափային տիպը օգտագործել առանց նախնական վերագրման, եթե նրան տրվել է `null` արժեք: Այդպիսի վերագրում կարելի է կատարել երկու ձևով:

```
Nullable<int> b=null; // կանցնենք հետո
int ? a=null;
```

- Այսպիսի անհրաժեշտություն կարող է առաջանալ, օրինակ բազայի ձևավորման ժամանակ, երբ որոշ դաշտեր արժեքներ չեն ձևավորում:
- `null` փոփոխականներում `HasValue` հատկանիշը հնարավորություն է տալիս ստուգել արժեքի գոյությունը կամ բացակայությունը:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Nullable<int> b = null;
        Console.WriteLine(b);
        int? a = null;
        Console.WriteLine(a);
        if (a.HasValue)
            Console.WriteLine("a has value");
        else
            Console.WriteLine("a =   no has value");
    }
}
```

- Եթե ունար օպերացիաներում ցանկացած փոփոխական `null` է, ապա արդյունքը ստացվում է `null`:

```
int? a = null;
int? b = a + 9;
```

b – ն ստանում է `null` արժեք:

- Նշվածից խուսափելու համար օգտագործվում է ?? օպերատորը, որը `null` արժեքները վերածում է լռելիությամբ արժեքների:
- ?? օպերատրի օգտագործման ժամանակ, եթե արտահայտությանը ստանում արժեք, ապա մնացած անդամները չեն դիտարկվում:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        int? a = null;
        int? b = a + 9;
        Console.WriteLine(b);
        b= a ?? 886;
        Console.WriteLine(b);
        a = 5;
        b = a ?? 886;
        Console.WriteLine(b);
    }
}
```

.....

30.Ստրուկտուրաներ

[Нейгел -130; Шилдт -391; Троелсен -172]

- Ստրուկտուրան իր օգտագործումը սկսել է C լեզվում: Այս կառուցվածքը սկզբից միայն կատարում էր տարբեր տիպերի հավաքագրում մեկ անվան տակ: Հաջորդ զարգացումը կիրառվեց C++ ում, որտեղ ստրուկտուրան բացի տվյալներից ներառեց նաև ֆունկցիաներ և ուներ հավասար համարժեքություն կլասին: C# ում ստրուկտուրան նոր սահմանում է ստանում:
- Ինչպես գիտենք կլասի օբյեկտները ստեղծվում են կույտում: Եթե կա անհրաժեշտություն այն ստեղծել ստեկում, ապա առաջարկվում է նոր չափային տիպ, որը կոչվում է ստրուկտուրա:

- Ստրուկտուրան չի կարող **ժառանգել** կլաս կամ այլ ստրուկտուրա, բայց կարող է օգտագործել interface: Ելնելով ժարանգականության սահմանափակումից, ստրուկտուրան չի օգտագործում մոդիֆիկատորներ abstract, virtual, protected: // կանցնենք հետո
- Ստրուկտուրան չունի դեստրուկտոր (կանցնենք հետո): Չի կարելի որոշել կոնստրուկտոր ըստ **լռելիությամբ** (առանց պարամետրերի):
- Կարելի է օգտագործել new օբյեկտ ստեղծելու համար, (բայց պարտադիր չէ): Եթե new չկա, ապա նախնական ինիցիալիզացիան կատարվում է ծրագրավորողի կողմից:
  Օբյեկտի ստեղծում երկու ձևով, SSS ob = new SSS(); կամ SSS ob;
- Չի թույլատրվում փոփոխականներին **ինիցիալիզացիա** տեղում: Չի թույլատրվում **կոմպոզիցիա**:
- Չկա բազային կլաս System.Structure, սակայն այն առաջացել է System.ValueType աբստրակտ կլասից:

```
using System;
struct S
{
    public int a;
    public void f()
    {
        Console.WriteLine(a);
        Console.WriteLine("barev");
    }
    public S(int aa)
    {
        a = aa;
    }
    //public S() // error!
    //{
    //}
}
class Program
{
    static void Main(string[] args)
    {
        S ob;
        ob.a = 99;
        ob.f();
        S ob2 = new S(44);
        ob2.f();
    }
}
```

- Եթե ստրուկտուրայի երկու օբյեկտ վերագրվում են մեկը մյուսին, ունենում ենք երկու կոպիա: Կլասի դեպքում կոպիան մեկն է, հղումները երկուսը: Օրինակ.

```
using System;
//class S
struct S
{
    public int x;
}
class M
{
    public static int Main()
    {
        S s1 = new S();
        s1.x = 100;
        S s2 = new S();
        s2 = s1;
        Console.WriteLine(s1.x);
```

```
        Console.WriteLine(s2.x);
        Console.WriteLine("Changing s2.x");
        s2.x = 900;
        Console.WriteLine(s1.x);
        Console.WriteLine(s2.x);
        return 0;
    }
}
//.........................
```

Օրինակ. Լոկալ տվյալների կուտակիչ

[albdarb]

- Տվյալների կուտակիչ ստանալու համար անհրաժեշտ է, առաջին, ստեղծել կլաս կամ ստրուկտուրա կուտակիչի անհրաժեշտ դաշտերով: Երկրորդ, օգտվել զանգվածից, կիրառվող կլասը կամ ստրուկտուրան կուտակիչ դարձնելու համար:
- Այլ խոսքով: Կուտակիչ = կլաս + զանգված:

```
using System;
struct Anketa
//class Anketa
{
    public int age;
    public string name;
}
class Program
{
    static void Main(string[] args)
    {
        int N = 25;
        string t = "";
        int x = 0; // Մուտքագրման քանակ
        int k = 0; // Ընդհանուր մուտքեր
        Anketa[] ob = new Anketa[N];
        //for(int i=0;i<N;i++)     // class AA
        //    ob[i] = new Anketa();
        for (; ; )
        {
            Console.WriteLine(@"0-Exit; 1-input; 2-List; 3-search by age; 4-search by name
            5=Sort by Age; 6=sort by name; 7-delete by name ");
            t = Console.ReadLine();
            switch (t)
            {
                case "0":
                    return;
                case "1":
                    Console.Write("mutqagrvogh tvyalneri qanak =  ");
                    x = Convert.ToInt32(Console.ReadLine());
                    for (int i = k; i < k + x; i++)
                    {
                        Console.Write("Enter name:");
                        ob[i].name = Console.ReadLine();
                        Console.Write("Enter age:");
                        ob[i].age = Convert.ToInt32(Console.ReadLine());
                    }
                    k = k + x;
                    break;
                case "2":
                    for (int i = 0; i < k; i++)
```

```
                {
                    Console.Write(ob[i].name + "\t");
                    Console.WriteLine(ob[i].age);
                }
                break;
            case "3":
                Console.WriteLine("search age = ");
                int tariq = Convert.ToInt32(Console.ReadLine());
                for (int i = 0; i < k; i++)
                {
                    if (ob[i].age == tariq)
                    {
                        Console.Write(ob[i].name + "\t");
                        Console.WriteLine(ob[i].age);
                    }
                }
                break;
            case "4":
                Console.WriteLine("Enter name:");
                string anun = Console.ReadLine();
                for (int i = 0; i < k; i++)
                    if (anun == ob[i].name)
                        Console.WriteLine(ob[i].name + "\t" + ob[i].age);
                break;
            case "5":
                    Anketa ob2 = new Anketa();
                    for (int z = 0; z < k; z++)
                        for (int i = 0; i < k - 1; i++)
                        {
                            if (ob[i].age > ob[i + 1].age)
                            {
                                ob2 = ob[i];
                                ob[i] = ob[i + 1];
                                ob[i + 1] = ob2;
                            }
                        }
                    break;
            case"6":
                for (int z = 0; z < k; z++)
                    for (int i = 0; i < k - 1; i++)
                    {
                        if (ob[i].name.CompareTo(ob[i + 1].name)>0)
                        {
                            ob2 = ob[i];
                            ob[i] = ob[i + 1];
                            ob[i + 1] = ob2;
                        }
                    }
                break;
            case"7":
                Console.WriteLine("anun delete = ");
                string s= Console.ReadLine();
                for (int i = 0; i < k; i++)
                    if (ob[i].name == s)
                    {
                        k--;
                        for (int z = i; z < k; z++)
                            ob[z] = ob[z + 1];
                        // i-- ;  // “բազի" դեմ
                    }
```

```
                        break;
                default:
                    Console.WriteLine("ara sxal a, miayn tiv");
                    break;
            }
        }
    }
}
// տնային -7-delete by name,  գտնել "բազ" իրար հաջորդող նույն անվան համար
```

.....

31.new չափային տիպերի համար, default

[Троелсен -120; Шилдт -170]

- new նշանակում է կոնստրուկտորի աշխատանք և նախատեսվում է լռելիությամբ նախնական արժեքները վերագրելու համար: Օրինակ. `int a = new int()` – նույնպես թույլատրելի է և a -ին տրվում է 0 արժեք :
- new չափային տիպերում օգտագործելը նշանակում է C# լեզվի լրիվ օբյեկտային կողմնորոշվածության դրսևորում:
- default հրամանը նշանակում է տիպի լռելիությամբ արժեք:
  `System.Console.WriteLine(default(bool));`
  Վերադարձնում է `False` արժեք:

.....

32.Կոպիայի կոնստրուկտոր

[Либерти -97]

- Ինչպես գիտենք կլասի օբյեկտների վերագրման ժամանակ միայն հղումն է վերագրվում և երկու հղումները դիմում են հիշողության նույն օբյեկտին: Որպեսզի հնարավոր լինի ստանալ երկու առանձին կոպիաներ, պետք է օգտվել կոպիայի կոնստրուկտորից:
- C# լեզուն չի ապահովում կոպիայի կոնստրուկտոր, և այն դառնում է ծրագրավորողի պարտականությունը: Անհրաժեշտ է պարամետրի միջոցով ստանալ օբյեկտ և էլեմենտ առ էլեմենտ կատարել վերագրումներ:

```
A ob =new A();
A ob1 = new A(ob);

class A
{
    public int a = 20;
    char c = 'w';
    string s = "hello";
    public A()
    {
    }
    public A(A ob)
    {
        a = ob.a;
        c = ob.c;
        s = ob.s;
    }
    public void ff()
    {
        System.Console.WriteLine(a);
    }
}
class M
```

```
{
    static void Main()
    {
        A ob = new A();
        A ob1 = new A(ob);
        //A ob1 = ob; // result …  20   99
        ob.ff();
        ob.a = 99;
        ob1.ff();
    }
}
```

……

33.Կլասի static անդամներ

[Троелсен -197; Шилдт -260]

- Կլասի ստատիկ անդամները օգտագործվում են կլասի անունով - A.ff(), այլ ոչ թե օբյեկտի միջոցով - ob.ff():
- Ստատիկ մեթոդները կարող են աշխատել միայն ստատիկ անդամների հետ: Իսկ սովորական մեթոդները - ցանկացած անդամների հետ:
- Որպեսզի ստատիկ մեթոդը աշխատի ոչ ստատիկ անդամի հետ, մեթոդը պետք է ստեղծի օբյեկտ:
- Հիմնականում լինում են ստատիկ անդամ տվյալներ և ստատիկ անդամ մեթոդ: Ստատիկ տվյալները տարբերվում են ոչ ստատիկից նրանով, որ անկախ օբյեկտների քանակից տվյալ դաշտի մեկ օրինակ գոյություն ունի: Սա հնարավորություն է տալիս տարբեր օբյեկտներ աշխատեն միևնույն տվյալի հետ:
- Ստատիկ անդամ տվյալը գործում է դոմենի սահմանում:
- Օրինակը ցույց է տալիս ստատիկ մեթոդի աշխատանքը և ինչպես կարելի է տատիկ մեթոդով դիմել ոչ ստատիկ անդամին: Ուղղակի դա կատարվում է օբյեկտ ստեղծելու ձևով:

```
using System; //Static Methods
class A
{     public static void ff()
      {
            Console.WriteLine("I'm static ff() class A");
      }
}
class M
{
    static void Main()
    {
        A.ff();
        ff();
        // ff2() !error
        M ob = new M();
        ob.ff2();
    }
    static void ff()
    {
         Console.WriteLine("I'm static ff() class M");
    }
    void ff2()
    {
        Console.WriteLine("I'm ff2() class M");
        ff();// Թույլատրելի է


    }
}
```

```
//-------------------------------------------------------
using System;  //Static Data
class A
{
    private static int ii = 0;
    public A()
    {
        ii++;
    }
    public static void ff()
    {
        Console.WriteLine("{0} class AAA ", ii);
    }
}
class M
{
    static void Main()
    {
        A.ff();
        A aa = new A();
        A.ff();
        A aa2 = new A();
        A.ff();
    }
}
```

.....

34.Ստատիկ կոնստրուկտոր, Ստատիկ կլասներ

[Шилдт-265; Троелсен -201; Нейгел-126] [Шилдт-266; Троелсен -202; Рихтер-172]

- Կարելի է օգտագործել ստատիկ կոնստուկտոր (միայն մեկ հատ): Դա երաշխավորում է, որ նա կաշխատի մինչև օբյեկտ ստեղծելը և միայն մեկ անգամ, անկախ օբյեկտների քանակից: Նա աշխատում է օբյեկտ ստեղծելուց առաջ, կամ առաջին ստատիկ անդամի դիմման ժամանակ: Նա չի կարող ունենալ ցանկացած մոդիֆիկատոր (public, private): Հիմնականում օգտագործվում է, եթե անհրաժեշտ է ստատիկ անդամներին տալ նախնական արժեք: Չի կարող ունենալ պարամետր:

```
using System;
public class A
{
    public static void ff()
    {
        Console.WriteLine("barev");
    }
    static A()
    {
        Console.WriteLine("static constructor");
    }
}
public class M
{
    static void Main()
    {
        //A ob = new A();
        A.ff();
    }
}
//.............................................................
```

Ստատիկ կլասներ

[Шилдт-266; Троелсен -202; Рихтер-172]

- C# 2005 –ում կլասը նույնպես կարող է լինել ստատիկ: Այդ դեպքում չի թույլատրվում new–ով ստեղծել կլասի օբյեկտ և պետք է ընդգրկի միայն ստատիկ անդամներ:
- Այսպիսով – կլասը փակելու համար կա երեք ձև - abstract կլաս, static կլաս և private constructor:

```
static class A
{
}
class A
{
     private A(){     }
}
abstract class A
{
}
```

- Նշված երկու տարբերակները տիպերի անվտանգության տեսակետից ավելի թույլ են, եթե անհրաժեշտ է կլասում ունենալ միայն ստատիկ անդամներ:

……

35.Փակ կոնստրուկտոր, Սինգլտոն փաթերն

[Либерти-101] [Нэш-96]

- Կլասի կոնստրոկտորը կարող է լինել փակ: Այս դեպքում արգելված է ստեղծել կլասի օբյեկտ արտաքին միջավայրից: Հնարավոր չէ ժառանգել, որովհետև ժառանգման ժամանակ աշխատում է կոնստրուկտորը: Փակ կոնստրուկտրով կլասները կարելի է օգտագործել ստատիկ անդամներով և հիմնականում նախատեսվում է ստատիկ տվյալների պահպանման համար:

```
using System;
public class A
{
    static string s = "private constructor";
    public static void ff()
    {
        Console.WriteLine(s);
    }
    private A()
    {      }
}
public class M
{
    static void Main()
    {
        //A a=new A(); !error
        A.ff();
    }
}
//……
```

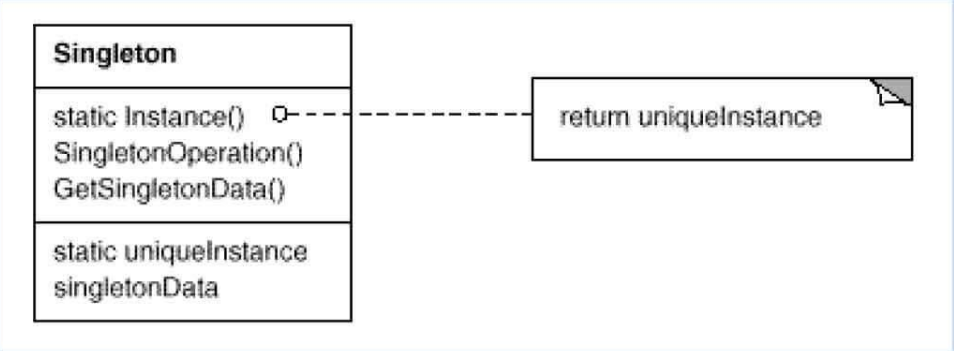
Սինգլտոն փաթերն

[Нэш-96]

- Гамма Э.... “Приемы объектно-ориентированного проектирования. Паттерны проектирования”. 1995eng.2010rus. ebook(rus).
- Նշված գրքով, որը գրվել է չորս հեղինակների կողմից (երբրմն անվանում են Gang of Four) 1995 թ. ին սկիզբ դրվեց ծրագրավորման ուղղության, որը կոչվում է Desigen Patterns: Փաթերները, դրանք

օբյեկտ կողմնորոշված ծրագրավորման պատրաստի բլոկներ են, որոնք արդեն մշակված են, ունեն մեծ անհրաժեշտություն և կիրառելի են տարբեր խնդիրներում:

- Սինգլտոն փաթերնը ամենապարզն է: Այն հնարավորություն է տալիս ստեղծել այսպես կոչված շաբլոն, որը ապահովում է տվյալ կլասի միայն մեկ օբյեկտի առկայություն:



- Սինգլտոն փաթերնը ստեղծվում է հետևյալ ձևով: Փակվում է կոնստրուկտորը, որը արգելում է կլասի օբյեկտի ստեղծումը: Կատարվում է public static մեթոդով կլասի օբյեկտի ստեղծում և պայմանական օպերատրի միջոցով կազմակերպվում է այնպես, որպեսզի ստեղծվի կլասի միայն մեկ օբյեկտ:
- Նշենք պարզեցված օրինակ, որը ապահովում է մեկ օբյեկտի առկայությունը, սակայն լինելով պարզ ունի թերություն, միշտ աշխատում է կոնստրուկտոր և ստեղծվում է օբյեկտ նախկին անունով:

```
using System;
public class A
{

    public static void ff()
    {
         A ob = new A();
    }
    A()
    {
        Console.WriteLine("constructor");
    }
}
public class M
{
    static void Main()
    {
        //A ob = new A(); error !
        A.ff();
        A.ff();
        A.ff();

    }
}
```

- Նշված թերությունից խուսափելու համար բերենք “օրիգինալ” Սինգլտոն փաթերնը, որտեղ օգտագործվում է `if` պայմանական օպերատորը և `null` թույլատրելի հղումը:

```
using System;
public class A
{
    static A instance = null;
    public static A ff()
    {
        if (instance == null)
```

```
        {
            instance = new A();
        }
        return instance;
    }
    A()
    {
        Console.WriteLine("constructor");
    }
}
public class M
{
    static void Main()
    {
        A a = A.ff();
        A b = A.ff();
        A c = A.ff();
    }
}
```

……..….

// Տնային – N քանակով օբյեկտների փաթերն

……..….

36.this հղում

[Троелсен -191; Шилдт-174]

- this կարելի է անվանել միջոց, որ հղում է օբյեկտին: Կամ այլ խոսքով, պահում է օբյեկտի հասցեն:
- Կլասի ցանկացած օրինակ (օբյեկտ) դինամիկ ղեկավարվող հիշողությունում իր առանձին տեղն է զբաղեցնում: Մեթոդների համար կա հիշողության առանձին տեղ և գոյություն ունի միայն մեկ օրինակ: Այսպիսով ստացվում է այնպես, որ մեթոդը կարող է աշխատել տարբեր օբյեկտների տվյալների հետ: Երբ անհրաժեշտ է իմանալ, թե մեթոդը ստույգ որ օբյեկտի հետ գործ ունի, օգտագործվում է this: Մեթոդի պարամետրերի ցուցակում այն կա, բայց չի նշվում:
- this – ով կարելի է մեթոդի լոկալ փոփոխականի փոխարեն դիմել կլասի նույն անունով անդամ փոփոխականին:
- this – ով կարելի է փոխանցել ընթացիկ օբյեկտը այլ մեթոդի պարամետրին:
- this – ը օգտագործվում է ինդեքսատորների կազմակերպման ժամանակ:
- this – ը կարող է կատարել լրացուցիչ բազմաթիվ գործողություններ: Ուղղակի պետք է հիշել, կամ այն ցույց է տալիս հղում (հիշողության հասցե), կամ հանդես է գալիս որպես հատուկ միջոց (բանալի բառ) տարբեր ծրգրային դետալներ կազմակերպելու համար:

```
using System;
class A
{
    int a;
    public void ff()
    {
        int a;
        a = 55;
        this.a = 44;
        Console.WriteLine(a);
        Console.WriteLine(this.a);
    }
}
class M
{
    static void Main()
```

```
    {
        A ob = new A();
        ob.ff();
    }
}
```

……..……

37.this որպես մեթոդի արգումենտ

[Либерти-98]

- this – ը թույլատրվում է գրել նաև մեթոդի արգումենտում:
- Օրինակում երևում է, որ A կլասի ff() մեթոդը ստանում է B կլասի օբյեկտ, որին ուղղեկցող A կլասի հղումը տեղադրվում է this – ի մեջ: Քանի որ A կլասի ff() մեթոդը օգտագործում է B կլասի ff2() մեթոդը և նա պարտադրում է A կլասի օբյեկտ, ապա կարելի արդեն this – ի մեջ տեղակայված A կլասի օբյեկտը օգտագործել և այն փոխանցել պարամետրի միջոցով:

```
using System;
class A
{
    public int a = 77;
    public void ff(B bb)
    {
        bb.ff2(this);
    }
}
class B
{
    public void ff2(A ob)
    {
        Console.WriteLine("a=" + ob.a);
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        B ob2 = new B();
        ob.ff(ob2);
    }
}
```

……

38.Կոնստրուկտոր և this

[Троелсен-193; Нейгел-128; Шилдт-245]

- Կարելի է մի կոնստրուկտորի միջոցով կանչել այլ կոնստրուկտոր, կոդի ավելցուկից խուսափելու համար: Սա this-ի օգտագործման ևս մի հնարավորություն է, սակայն այն օգտագործվում է որպես “հանգույցային” բառ:

```
class A
{
    public int n;
    public string anun;
    public float pi;
    public A(string s, float f)
    {
```

```
        anun = s;
        pi = f;
    }
    public A(int nn) : this("Ani", 3.14f)
    {
       n = nn;
    }
}
class M
{
    static void Main()
    {
        A ob2 = new A(777);
        System.Console.WriteLine(ob2.pi + " " + ob2.anun + " " + ob2.n);
        A ob = new A("Nune", 3.14f);
        System.Console.WriteLine(ob.pi + " " + ob.anun + " " + ob.n);
    }
}
```

……

39.const հաստատուն, readonly հաստատուն

[Рихтер-191; Троелсен-226] [Троелсен-228; Нейгел-129; Нэш-63]

- const հաստատունի արժեքը հայտնի է լինում թարգմանման ժամանակ:
- Չի թույլատրվում const հաստատունը նշել որպես ֆունկցիայի պարամետր և վերադարձվող արժեք:
- const հաստատունի օգտագործվում է ստատիկ անդամի նման, այսինքն կլասի անունով:

```
using System;
class A
{
    public const int a = 555;
}
class M
{
    static void Main()
    {
        Console.WriteLine(A.a);
        // A ob = new A();
        //Console.WriteLine(ob.a);// error !!
        const int b = 777;
        Console.WriteLine(b);
    }
}
//……
```

readonly հաստատուն [Троелсен-228; Нейгел-129; Нэш-63]

- Շատ դեպքերում հաստատունի արժեքը անհրաժեշտ է լինում վերագրել օգտագործման ժամանակ, այլ ոչ թե թարգմանման: Այդ դեպքում ինիցալիզացիան կատարվում է կանստրուկտորում և կարող է կատարվել մեկից ավել:
- Ի տարբերություն const անդամի, readonly անդամին դիմում ենք նաև օբյկտով: readonly հաստատունը կարող է լինել ստատիկ:

```
using System;
class A
{
    public readonly int ScreenWidth = 99; //ok!!
    public readonly int ScreenHeight;
```

```
    public A()
    {
        this.ScreenWidth = 1024;
        this.ScreenHeight = 100;
        this.ScreenHeight = 768;
    }
}
class M
{
    public static void Main()
    {
        A ob = new A();
        Console.WriteLine(ob.ScreenWidth);
        Console.WriteLine(ob.ScreenHeight);
        // ob. ScreenHeight=8888; //error!!
    }
}
```

.....

40.Հաստատուն enum (թվարկում)

[Шилдт-397; Троелсен-167; Нейгел-90]

- Թվարկումը չափային տիպ է, որը կարող է պահպանել թվային հաստատուններ սիմվոլային անվանումների միջոցով: Թվարկման էլեմենտների տիպեր կարող են լինել byte, short, int, long: Լռելիությամբ ընդունում է int:
- Թվարկումը կարող է ներդրվել կլասում, կամ նրանից դուրս: Ինչպես հաստատունը, թվարկմանը նույպես դիմում են կլասի անունով, կետով ավելացնելով անդամները:

```
using System;
enum ee2 : short { a, b, cc55 };
class A
{
    public enum ee { red, blue = 7, black } ;
}
class M
{
    public static void Main()
    {
        Console.WriteLine((int)A.ee.red);
        Console.WriteLine((int)A.ee.blue);
        Console.WriteLine((int)A.ee.black);
        Console.WriteLine(ee2.a);
        Console.WriteLine((int)ee2.b);
    }
}
```

.....

41.System.Enum

[Троелсен-169]

- Շարփ լեզուն շարունակում է իր սովորույթը: Եթե կա զաղափար, ապա առաջարվում է նաև նրան ծառայող կլաս:
- Թվարկումը կարող է օգտվել Enum կլասից, որի անդամներից են.

✓ GetUnderlyingType() – Վերադարձնում է օգտագործվող տիպը:
✓ Format() – թվարկման արժեքը նշված ֆորմատով փոխարկում է սիմվոլային համարժեքության:
✓ GetValues() – վերադարձնում է զանգված, որը պարունակում է թվարկման անդամները:

✓ IsDefined – հատկանիշ, որը հնարավորություն է տալիս իմանալ սիմվոլային տողը հանդիսանում է արդյոք թվարկման էլեմենտ:
✓ GetName() – վերադարձնում է անունը տված թվարկման արժեքի համար:

```
using System;
enum ee : byte
{
    red, blue = 57, black
}
class M
{
    public static void Main()
    {
        Console.WriteLine(Enum.GetUnderlyingType(typeof(ee)));
        Console.WriteLine(Enum.GetName(typeof(ee), 0));
        Console.WriteLine(Enum.IsDefined(typeof(ee), "red"));
        Console.WriteLine(Enum.Format(typeof(ee), ee.blue, "x"));
        //"d"-decimal //"G" - string
        Array ob = Enum.GetValues(typeof(ee));
        Console.WriteLine(ob.Length);
        foreach (ee k in ob)
            Console.WriteLine((int)k);
    }
}
```

.....

42.foreach – ցիկլ

[Троелсен-145; Нейгел-197]

- foreach - ը ցիկլի կազմակերպման ձև է, որը հնարավորություն է տալիս կուտակիչից – զանգվածից կարդալ ինֆորմացիան: foreach - ի աշխատանքի դեպքում ցիկլը աշխատում է բոլոր էլեմենտների համար, սկզբից մինչև վերջ:
- for –ի հետ համեմատած պետք չէ ինիցիալիզացնել ցիկլի փոփոխական, պետք չէ ստուգել բուլիան արտահայտություն և ցիկլի փոփոխականը մեծացնել:
- foreach - ը համարվում է ավելի քիչ ռեսուրս պահանջող միջոց for –ի հետ համեմատած:
- Զանգվածի պարունակությունը չի ձևավորվում foreach -ով :

```
using System;
class M
{
    public static void Main()
    {
        int ss = 0;
        int[] A = new int[10];
        for (int i = 0; i < 10; i++)
            A[i] = i;
        foreach (int x in A)
        {
            Console.WriteLine("value= " + x);
            ss += x;
        }
        Console.WriteLine("Sum = " + ss);
    }
}
```

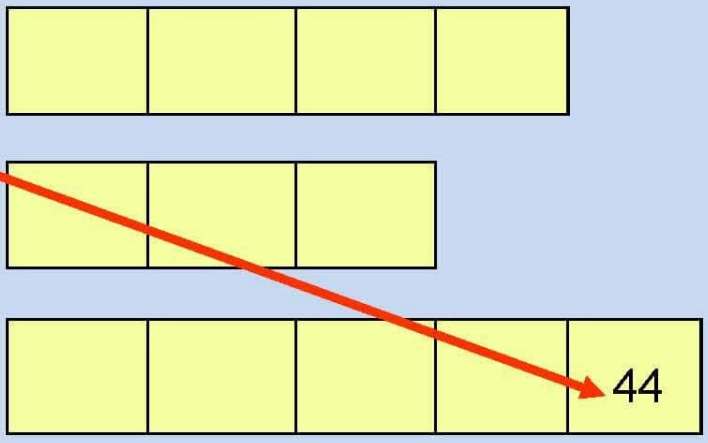
.....

43.Անհամաչափ զանգված

[Троелсен-163; Нейгел-188]

- Թույլատրվում է երկչափ զանգվածի (մատրիցայի) տողերի վեկտորների չափերը լինեն տարբեր: Այն կոչվում անհամաչափ զանգված: Օգտագործվում է հատուկ խնդիրների դեպքում հիշողության օպտիմալ կազմակերպման համար: Այդ դեպքում պետք է հայտարարման ժամանակ ֆիքսել տողերի չափերը և ամեն տողում հայտարարել կամայական չափի նոր վեկտոր:

```
int[ ][ ] AA = new int[3][ ];
AA[0] = new int[4];
AA[1] = new int[3];
AA[2] = new int[5];
AA [2][4] = 44;
```




```
using System; // ուղղանկյուն սեղան
class M
{
    public static void Main()
    {
        int[][] AA = new int[5][];
        for (int i = 0; i < AA.Length; i++)
            AA[i] = new int[i + 8];
        for (int i = 0; i < 5; i++)
        {
            for (int j = 0; j < AA[i].Length; j++)
                Console.Write(AA[i][j]);
            Console.WriteLine();
        }
    }
}
//-------------------------------------
class M // եռաչափ անհամաչափ զանգված
{
    static void Main(string[] args)
    {
        int[][][] AAA = new int[2][][];
        AAA[0] = new int[2][];
        AAA[1] = new int[4][];
        AAA[0][0] = new int[4];
        AAA[0][1] = new int[5];
        AAA[1][0] = new int[6];
        AAA[1][1] = new int[7];
        AAA[1][2] = new int[8];
        AAA[1][3] = new int[9];
    }
}
```

.....

44.System.Array

[Троелсен-165; Либерти-192]

- Եթե հայտարավում է զանգված, ապա այն կարող է օգտվել Array կլասի բոլոր անդամներից:
- Array կլասը ունի հետևյալ անդամները:

✓ **BinarySearch()** - Overloaded static method that searches a one-dimensional sorted array.
✓ **Clear()** - static method that sets a range of elements in the array either to zero or to a null reference.
✓ **Copy()** - Overloaded static method that copies a section of one array to another array.
✓ **IndexOf()** - Overloaded static method that returns the index (offset) of the first instance of a value in a one-dimensional array.
✓ **LastIndexOf()** - Overloaded static method that returns the index of the last instance of a value in a one-dimensional array.
✓ **Reverse()** -Overloaded static method that reverses the order of the elements in a onedimensional array
✓ **Sort()** - Overloaded static method that sorts the values in a one-dimensional array.
✓ **IsFixedSize** - property that returns a value indicating whether the array has a fixed size.
✓ **Length** - property that returns the length of the array.
✓ **Rank** - property that returns the number of dimensions of the array.
✓ **GetLength()** - method that returns the length of the specified dimension in the array.
✓ **SetValue()** - Overloaded method that sets the specified array elements to a value.

```
using System;
class M
{
    public static void Main()
    {
        int[] aa = new int[10];
        int[] aa2 = new int[10];
        for (int i = 0; i < 10; i++)
            aa[i] = i + 5;
        Array.Clear(aa, 2, 4);
        aa.SetValue(99, 2);
        foreach (int x in aa)
            Console.Write(" " + x);
        Array.Sort(aa);
        Console.WriteLine();
        foreach (int x in aa)
            Console.Write(" " + x);
        Console.WriteLine();
        Console.WriteLine("search = " + Array.BinarySearch(aa, 12));
        aa[5] = 14;
        Console.WriteLine(Array.LastIndexOf(aa, 14));
        Console.WriteLine(Array.IndexOf(aa, 14));
        Console.WriteLine("rank = " + aa.Rank);
        Array.Reverse(aa);
        foreach (int x in aa)
            Console.Write(" " + x);
        Console.WriteLine();
        Array.Copy(aa, aa2, 5);
        foreach (int x in aa2)
            Console.Write(" " + x);
        Console.WriteLine();
        Console.WriteLine(aa.IsFixedSize);
        ArrayList ary = new ArrayList();
        Console.WriteLine(ary.IsFixedSize);
    }
}
```

45.Տողի մշակում (string)

[Нейгел-266; Троелсен-127]

- **System.String**

string տիպը հնարավորություն է տալիս աշխատել տողերի հետ, որոնք բաղկացած են unicode–ի սիմվոլներից: string – ը ունի առանձնահատկություն, աշխատանքը կատարվում է միայն նրա կոպիայի հետ: string – ը և String կլասը ամբողջությամբ նույնն են:

```
string ss ="barev !";
int n=55;
ss=n.ToString();
```

- String կլասը ունի հետևյալ անդամները:

✓ **Empty** - static field that represents the empty string.
✓ **Compare()** - Overloaded static method that compares two strings.
✓ **Concat()** - Overloaded static method that creates a new string from one or more strings.
✓ **Copy()** - static method that creates a new string by copying another.
✓ **Equals()** - Overloaded static and instance methods that determines if two strings have
✓ **Format( )** - Overloaded static method that formats a string using a format specification..
✓ **Join()** - Overloaded static method that concatenates a specified string between each element of a string array.
✓ **Length** - The number of characters in the instance.
✓ **EndsWith()** - Indicates whether the specified string matches the end of this string.
✓ **LastIndexOf()** - Reports the index of the last occurrence of a specified character or string within the string.
✓ **Insert()** - Returns a new string with the specified string inserted.
✓ **Remove()** - Deletes the specified number of characters.
✓ **Substring()** - Retrieves a substring.
✓ **ToLower()** - Returns a copy of the string in lowercase.
✓ **ToUpper()** - Returns a copy of the string in uppercase

- Ուշադրություն դարձրեք օրինակի վեջին չորս հրամաններին:

```
s1 = "a";
s2 = "a";
Console.WriteLine(ReferenceEquals(s1, s2)); // true
Console.WriteLine(Equals(s1, s2)); //true
```

- Եթե երկու string փոփոխական ունեն նույն պարունակությունը, ապա նրանց հղումները թարգմանիչը նույնացնում է: Դա հնարավոր է (կամ անհրաժեշտ է), քանի որ string – ը աշխատում է կոպիայի հետ: Այլ խոսքով `Console.WriteLine(ReferenceEquals(s1, s2))` հրամանը պետք է տար false, սակայն դա այդպես չէ: Եթե վերագրենք տարբեր արժեքներ

```
s1 = "a";
s2 = "aaaa";
```

ապա `Console.WriteLine(ReferenceEquals(s1, s2))` հրամանը կարտածի false:

```
using System;
class M
{
    static void Main()
    {
        string s1 = "abcd";
        String s2 = "ABCD";
        Console.WriteLine(string.Compare(s1, s2));
        Console.WriteLine(string.Compare(s1, s2, true));
        Console.WriteLine(string.Concat(s1, s2));
        Console.WriteLine(s1 + s2);
        string s = string.Copy(s2);
        Console.WriteLine(s);
```

```
        s = s2;
        Console.WriteLine(s);
        Console.WriteLine(s.Equals(s2));
        Console.WriteLine(string.Equals(s1, s2));
        Console.WriteLine(s1 == s2);
        s = " ";
        string[] a = { "alb", "ert", "\t", "dar",
                                    "bin", "yan" };
        Console.WriteLine(string.Join(s, a));
        Console.WriteLine(s1.Length);
        Console.WriteLine(s1.EndsWith("cd"));
        Console.WriteLine(s2.Insert(2, "barev"));
        s = s1 + s2;
        Console.WriteLine(s.LastIndexOf("BC"));
        Console.WriteLine(s.Remove(4, 2));
        Console.WriteLine(s.Substring(2, 3));
        Console.WriteLine(s.ToLower());
        Console.WriteLine(s.ToUpper());
        s1 = "a";
        s2 = "a";
        Console.WriteLine(ReferenceEquals(s1, s2));
        Console.WriteLine(Equals(s1, s2));
    }
}
//.....................................................
```

- Հաջորդ օրինակը ցույց է տալիս, որ հիշողություն չի ավելանում s = "aa" երկրորդ հրամանի ավելացման դեպքում, քանի որ string փոփոխականը աշխատում է կոպիայի հետ և այն արդեն գոյություն ունի: Ծրագիրը կարելի է աշխատացնել “կոմենտարյաներ” դնելով և հիշողության զբաղեցրած ծավալը ստուգելով:
- `GC.GetTotalMemory(false)` հիշողություն հաշվող հրամանը, որը որոշում է օբյեկտի զբաղեցրած ծավալը, կանցնենք ավելի ուշ (համեցեք C# dotNET):

```
using System;
class A
{
    string s = "12";
    public void f()
    {
        s = "aa";
        s = "bb";
        s = "aa";
    }
}
class Program
{
    static void Main()
    {
        A o = new A();
        o.f();
        GC.Collect();
        Console.WriteLine("memory=" + GC.GetTotalMemory(false));
    }
}
.....................................................................................................
```

46.System.Text.StringBuilder

[Троелсен -133]

- Ի տարբերություն string տիպի, որը աշխատում է կոպիայի հետ, StringBulder կլասը հնարավորություն է տալիս աշխատել բնօրինակի հետ, որը իր հերթին բերում է էֆեկտիվության տողի բազմակի փոփոխության ժամանակ:
- Օրինակում երևում է, որ նույնիսկ տողի փոփոխության իտերացիան 100 անգամ մեծացնելուց հետո StringBulder տողը ավելի արագ է մշակվում:
- Եթե անհրաժեշտ է վերածել StringBulder – ը string օբյեկտի, ապա օգտագործվում է ToString() մեթոդը:

```
using System;
class Program
{
    const int iter = 20000;
    static void Main(string[] args)
    {
        DateTime dt = DateTime.Now;
        string s = "";
        for (int i = 0; i < iter; i++)
            s+= "aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n";
        Console.WriteLine(DateTime.Now - dt);
    }
}
//.............................................

using System;
using System.Text;
class Program
{
    const int iter = 2000000;
    static void Main(string[] args)
    {
        DateTime dt = DateTime.Now;
        StringBuilder sb = new StringBuilder();
        for (int i = 0; i < iter; i++)
            sb.Append("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa\n");
        Console.WriteLine(DateTime.Now - dt);
    }
}
//.............................................
```

- Հաջորդ օրինակը ցույց է տալիս, թե ինչպես կարելի է ստուգել զբաղեցրած հիշողությունը string – ի և StringBulder – ի համար: Ստացվում է, որ string – ը գործողությունների արդյունքում ավելի մեծ տարածք է վերցնում, եթե չի օգտագործվում GC.Collect() հիշողություն մաքրող հրամանը:

```
using System;
using System.Text;
class Program
{
    static void Main()
    {
        string s = "1234567890";
        for (int i = 0; i < 1000; i++)
            s = s + "abcdaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"; //GC.Collect();
        Console.WriteLine("memory=" + GC.GetTotalMemory(false));
        //-------------------------------
```

```
        //StringBuilder sb = new StringBuilder("1234567890");
        //for (int i = 0; i < 1000; i++)
        //    sb.Append("abcdaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"); //GC.Collect();
        //Console.WriteLine("memory=" + GC.GetTotalMemory(false));
    }
}
```

.....

47. String.Format

[Шилдт – 812; Нейгел -271; Covaci – 150]

```
using System;
class M
{
    static void Main()
    {
        double temp = 15624.567;
        double temp2 = 0.12;
        int temp3 = 47;
        Console.WriteLine(String.Format("{0:N2}", temp));
        Console.WriteLine(String.Format("{0:F2}", temp));
        Console.WriteLine(String.Format("{0:e}", temp));
        Console.WriteLine(String.Format("{0:c}", temp));
        Console.WriteLine(String.Format("{0:x}", temp3));
        Console.WriteLine(String.Format("{0:p}", temp2));
        Console.WriteLine(String.Format("{0:N2} *** {0:e} *** {2:p} ", temp, temp2, temp3));
        Console.WriteLine(String.Format("{0:10,0##.#00}", 23.6));
        Console.WriteLine(String.Format("{0:###0}", 23));
        Console.WriteLine(String.Format("{0:000#}", 23));
        //---------------------------------------------
        DateTime dt = DateTime.Now;
        Console.WriteLine(String.Format("{0:d}", dt));
        Console.WriteLine(String.Format("{0:D}", dt));
        Console.WriteLine(String.Format("{0:t}", dt));
        Console.WriteLine(String.Format("{0:T}", dt));
        Console.WriteLine(String.Format("{0:g}", dt));
        Console.WriteLine(String.Format("{0:G}", dt));
        Console.WriteLine(String.Format("{0:M}", dt));
        Console.WriteLine(String.Format("{0:R}", dt));
        Console.WriteLine(String.Format("{0:s}", dt));
        Console.WriteLine(String.Format("{0:U}", dt));
        Console.WriteLine(String.Format("{0:Y}", dt));
        Console.WriteLine(String.Format("{0:hh mm ss}", dt));
        Console.WriteLine(String.Format("{0:hh:mm:ss - MM yyyy}", dt));
        Console.WriteLine(String.Format("{0:hh:mm:ss - MMM yyyy}", dt));
        Console.WriteLine(String.Format("{0:hh:mm:ss - MMMM yy}", dt));
    }
}
```

.....

48.Հատկանիշ (Properties)

[Шилдт-313; Троелсен -212; Рихтер -248]

- Օբյեկտային կողմնորոշված ծրագրավորումը պարտադրում է, որպեսզի կլասի անդամ տվյալները լինեն փակ, և նրանց օգտագործումը կատարել բաց ֆունկցիաների միջոցով: C# - ում այսպիսի աշխատանքը մտցված է բազային համակարգ և իրագործվում է հատկանիշի (properties) միջոցով: Երբեմն նրան անվանում են ինտելեկտուալ դաշտեր, իսկ էությունը ավելի մոտ է պարզեցված ֆունկցիա գաղափարին: Հատկանիշները իրագործվում են set և get աքսեսուարների միջոցով:
- set – ը հնարավորություն է տալիս տվյալներին վերագրել արժեք: Այն օգտագործում է value հիմնային հրամանը, որը ներքին փոփոխական է և ծառայում է արժեք ստանալու համար: Հատկանիշի օգտագործման համար գրում ենք հատկանիշի անունը, վերագրման օպերատորը և անհրաժեշտ արժեքը:
- get – ը հնարավորություն է տալիս ստանալ հատկանիշի պարունակությունը: Այստեղ օգտագործվում է return հրամանը: Այս դեպքում հատկանիշի անունը պետք է գտնվի վերագրման օպերատրի աջ մասում: Պարզ է, որ ձախ մասը ստանում է հատկանիշի արժեքը:
- Նշենք, որ հատկանիշը կարող է աշխատել նաև կամ միայն set կամ միայն get տարբերակով:
- set և get աքսեսուարները կարող են ստանալ թույլատրության մոդիֆիկատորներ (private, protected, և այլն), սակայն միայն որևէ մեկով: Այլ խոսքով, եթե մոդիֆիկատորը դրված է մեկի վրա, ապա մյուս աքսեսուարին չի թույլատրվում մոդիֆիկատոր դնել: Դա պարզ է, քանի որ հատկանիշի գլխավոր մոդիֆիկատորը կկորցնի իր իմասը: Փորձել:
- Նշենք, որ հատկանիշը կարող է աշխատել նաև կամ միայն set կամ միայն get տարբերակով:
- Հատկանիշները նույնպես "ինկապսուլացիայի" մի ձև են, երբ անհրաժեշտ է տվյալները ծածկել (փակել, private) և նրանց դիմել բաց (public) գործողություններով: (Գնացինք Արաբական աշխարհ աղջիկ ուզելու):

```
class A
{
    string s;
    public string PP
    {
        get
        {
            return s;
        }
        set
        {
            s = value;
        }
    }
}

class M
{
    static void Main()
    {
        A ob = new A();
        ob.PP = "55555";
        string z = ob.PP;
        System.Console.WriteLine(z);
    }
}
```

.....

```
//  example with function
class A
{
    string s;
    public string ff()
    {
        return this.s;
    }
    public void ff2(string ss)
    {
        this.s = ss;
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        ob.ff2("5555");
        string z = ob.ff();
        System.Console.WriteLine(z);
    }
}
```

.....

49.Ավտոմատ հատկանիշներ

[Троелсен -219; Шилдт-318; Нэш -69; Нейгел-124]

- Եթե հատկանիշը օգտագործվում է միայն դաշտերին արժեք տալու և կարդալու համար, ապա Sharp 2008 –ում նույն աշխատանքը կատարելու համար նախատեսված է ավելի պարզ ձև , որը կոչվում է ավտոմատ հատկանիշներ: **public int pp { get; set; }**

```
using System;
class A
{
    public int pp { get; set; }
    public string pp2 { get;set; }
}
class M
{
    static void Main()
    {
        A o = new A();
        Console.WriteLine(o.pp);
        o.pp = 999;
        Console.WriteLine(o.pp);
        Console.WriteLine(o.pp2);
        Console.ReadKey();
    }
}
```

- Չի թույլատրվում միայն get կամ միայն set :
- Կարող է ընդունել լռելիությամբ արժեքներ:
- Ավտոհատկանիշը աբողջությամբ փոխարինում է դաշտին(կլասի int a), ունենալով որոշ առավելություններ: Օրինակ, կարելի է տալ թույլատրության մոդիֆիկատորներ կամ միայն get -ին կամ միայն set –ին: Կամ ստանալ օրինակ, որոշ ատրիբուտներ, որ չի կարող ստանալ դաշտը:

.....

50.operator - Օպերատորների գերբեռնում

[Троелсен-426; Нейгел-219; Шилдт-269]

- Կլասների միջոցով ստեղծված օբյեկտների միջև առանց օպերատորների գերբեռնման հնարավոր չէ կատարել բազային տիպերի օպերացիաներ, բացի վերագրման օպերատրից և == համեմատումից: Նշվածը հնարավոր է ֆունկցիայի միջոցով ծրագրային իրականացումով. A ob =ob1.ff(ob2);
- C# - ը ունի ներդրված հնարավորություն operator հրամանի միջոցով:
  A ob = ob1+ob2; և այլ հրամաններ դառնում է հնարավոր:

```
public static A operator + (A ob1, A ob2)
{       ……
        return ob;
}
```

- Օպերատրի պարամետրում արգելվում է օգտագործել ref և out: (կանցնենք)
- public static լինելը պարտադիր է: Օպերատորը, կախված թե ինչ օպերացիա է անում, "ունար" թէ "բինար", կարող է ընդունել մեկ կամ երկու պարամետր:
- "Ունար" օպերացիայի դեպքում օպերատրի պարամետրը պետք է ունենա այն տիպը, որի համար որոշվում է: "Բինար" օպերացիայի համար, գոնե մեկ պարամետր պետք է լինի որոշվող կլասի տիպ:
- +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=

Նշված կրճատված օպերացիաները գերբեռնվում են ավտոմատ, եթե համապատասխան "բինար" օպերացիաները գերբեռնվել են:

- ==, != թույլատրվում է գերբեռնել, սակայն առանց գերբերռնման նույնպես աշխատում են, որովհետև == փոխարինում է object կլասի Equals անդամին: (կանցնենք)
- Չի թույլատրվում գերբեռնել հետևյալ օպերատորները.
  ( ), [], =, &&, ||, new, default, checked, unchecked, typeof, ->, is, as, =>, ?? , sizeof
- =, չի գերբեռնվում, որովհետև արդեն թույլատրվում նույն կլասի օբյեկտները վերագրել:
- ( ) տիպերի վերափոխարկում, չի գերբեռնվում, սակայն կա նրան համապատասխան օպերացիաներ implicit և explicit: (կանցնենք)
- [], զանգվածի ինդեքսացինա չի գերբեռնվում, որովհետև կա հատուկ միջոց, որը կոչվում է **ինդեքսատոր**:

```
using System; // overload opererator ++  +
class A
{
    int a = 50;
    double d = 2.22;
    string s = "AniNune ";
    public static A operator +(A op1, A op2)
    {
        A result = new A();
        result.a = op1.a + op2.a;
        result.d = op1.d + op2.d;
        result.s = op1.s + op2.s;
        return result;
    }
    public static A operator ++(A op)
    {
        A result = new A();
        result.a = ++op.a;
        return result;
    }
    public void ff()
    {
        Console.WriteLine(a + " " + s + " " + d);
    }
}
```

```
class M
{
    public static void Main()
    {
        A ob1 = new A();
        A ob2 = new A();
        A ob = ob1 + ob2;
        ob.ff();
        ob++;
        ob.ff();
    }
}
// Տնայիններ
//-----------------------------------------------------
// overload operator == !=
// overload operator true false
// overload opererator &
// overload operator * / %  > <
// overload operator | ! ~ ^  >> <<
//-----------------------------------------------------
using System; // overload operator == !=
class A
{
    public int a = 50;
    public static bool operator ==(A op1, A op2)
    {
        if(op1.a==op2.a)
            return true;
        else
            return false;
    }
    public static bool operator !=(A op1, A op2)
    {
        if (op1.a != op2.a)

            return true;
        else
            return false;
    }
}
class M
{
    public static void Main()
    {
        A ob1 = new A();
        A ob2 = new A();
        Console.WriteLine(ob1==ob2);
        ob1.a = 99;
        Console.WriteLine(ob1 == ob2);
    }
}
// ----------------------------------------
using System;// overload operator true false
class A
{
    public int a = 50;
    public static bool operator true (A op)
    {
        if(op.a==0)
```

```
            return true;
        else
            return false;
    }
    public static bool operator false(A op)
    {
        if (op.a!=0)

            return true;
        else
            return false;
    }
}
class M
{
    public static void Main()
    {
        A ob = new A();
        if(ob)
        Console.WriteLine("1");
        else
            Console.WriteLine("0");
        ob.a = 0;
        if (ob)
            Console.WriteLine("1");
        else
            Console.WriteLine("0");
    }
}
//----------------------------
using System;// overload opererator &
class A
{
    public int a = 50;
    public static bool operator & (A op1,A op2)
    {
        if(op1.a==0&op2.a==0)
            return true;
        else
            return false;
    }
}
class M
{
    public static void Main()
    {
        A ob1 = new A();
        A ob2 = new A();
        if(ob1&ob2)
            Console.WriteLine("1");
        else
            Console.WriteLine("0");
        ob1.a = 0;
        ob2.a = 0;
        if (ob1 & ob2)
            Console.WriteLine("1");
        else
            Console.WriteLine("0");
    }
}
```

51.Ինդեքսատոր

[Троелсен-421; Шилдт-303]

- Ապահովում է մեխանիզմ, որի միջոցով կլասի անդամներին կարելի է դիմել զանգվածի ձևով:
- Ինչպես գիտենք C++ - ում զանգվածի [ ] այս էլեմենտը կարելի է գերբեռնել , որպես օպերացիաների վերաբեռնում operator հրամանով: C# - ում այս նույն էֆեկտին կարելի է հասնել ինդեքսատրի միջոցով: Նաև, որ C# - ում [ ] այս էլեմենտը չի կարելի գերբեռնել և ստիպված պետք է դիմել ինդեքսատրի:

```
class A
{
    public object this[int ii]
    {
        get { }
        set { }
    }
}
```

- Եթե հատկանիշներին անվանում են խելացի դաշտեր, ապա ինդեքսատրներին – խելացի զանգվածներ: Ինչպես երևում է նկարագրությունից, օգտագործվում է get, set աքսեսուարները և օգտագործման ժամանակ: Եթե ինդեքսատրը գտնվում է ձախ վերագրման օպերատրից, կանչվում է set աքսեսորը: Աջի դեպքում – get:
- Ինդեքսատորը չունի անուն, որովհետև նա միայն իր կլասի անդամ է և դրա համար էլ օգտագործվում է this - ը: . Չի կարող լինել ստատիկ:
- Ինդեքսատրը կարելի է օգտագործել հետևյալ կերպ՝

```
A ob = new A();
ob[0] = someObject;
Console.WriteLine("{0}", ob[0]);
```

```
using System;
class A
{
    private int[] a = new int[5];
    public int this[int ii]
    {
        set
        {
            a[ii] = value;
        }
        get
        {
            return a[ii];
        }
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        for (int i = 0; i < 5; i++)
            ob[i] = i;
        for (int i = 0; i < 5; i++)
            Console.WriteLine(ob[i]);
    }
}
```

.....

52.Առանց կուտակիչի ինդեքսատոր

[Шилдт-310]

- Պարտադիր չէ ունենալ կուտակիչ ինդեքսատոր օգտագործելու համար: Այդ դեպքում get ում գրվում է ծրագիրի ալգորիթմական իրականացում, որը մշակում է ինդեքսը և վերադարձնում արժեք:

```
using System;
class A
{
    public int this[int ii]
    {
        get
        {
            if ((ii >= 0) && (ii < 16))
                return ff(ii);
            else return -1;
        }
    }
    int ff(int p)
    {
        int result = 1;
        for (int i = 0; i < p; i++)
            result *= 2;
        return result;
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        Console.Write("8 pow 2: ");
        for (int i = 0; i < 8; i++)
            Console.Write(ob[i] + " ");
        Console.WriteLine();
        Console.Write("errors ");
        Console.WriteLine(ob[-1] + " " + ob[17]);
    }
}
```

.....

53.Գերբեռնված ինդեքսատոր

[Шилдт-307; Либерти-216]

- Թույլատրվում է գերբեռնել ինդեքսատորը: Ներդրված տիպերը կարող են գերբեռնվել ինդեքսատորներում: char, int, string, bool, decimal, object և այլն:

```
using System;
public class A
{
    public string[] ss={"ani","nune","darbinyan"};
    public string this[int index]
    {
        get
        {
            return ss[index];
        }
```

```
        set
        {
            ss[index] = value;
        }
    }
    public string this[string index]
    {
        get
        {
             string s = "";
             for (int i = 0; i < 3; i++)
                 if (ss[i] == index)
                     s = ss[i];
             return s;
        }
        set
        {
              for (int i = 0; i < 3; i++)
                if (ss[i] == index)
                    ss[i] = value;
        }
    }
}
public class M
{
    public static void Main()
    {
        A ob = new A();
        for (int i = 0; i < 3; i++)
            Console.WriteLine(ob[i]);
        ob[0] = "aaa";
        ob["aaa"] = "alb";
        for (int i = 0; i < 3; i++)
            Console.WriteLine(ob[i]);
        if(ob["nune"]=="nune")
        Console.WriteLine("= name nune exists =");
        else
            Console.WriteLine("false");
    }
}
```

.....

54.Main() մեթոդի պարամետր

[Троелсен-108; Шилдт-255; Нейгел-95]

- Main ֆունկցիան կարող է ընդունել (`string[] args`) պարամետր: Նշված պարամետրը ծառայում է, երբ անհրաժեշտ է ծրագիրը աշխատացնել հրամանի տողով (Windows - ում Start -> Run) և ցանկություն կա հրամանի տողում ներառել տվյալներ: Տվյալները այդ դեպքում իրենցից ներկայացնում են տողեր, որոնք տարանջատվում են դատարկ սիմվոլով (Space): Նշված պարամետրերը ավտոմատ գրանցվում են (`string[] args`) զանգվածում, որը իր հերթին կարելի է օգտագործել ծրագրում:

```
using System;
class M
{
    static void Main(string[] args)
    {
```

```
        for (int i = 0; i < args.Length; i++) Console.WriteLine(args[i]);
        // string[] s = Environment.GetCommandLineArgs();
        // Console.WriteLine(s[0]);
        Console.ReadKey();
    }
}
```

.....

55.Մեթոդի անվանական արգումենտներ (C# 4)

[Шилдт-252; Нейгел-121; Уотсон-411; Covaci – 94]

- (Named Arguments.)
- Եթե մեթոդը ունի մեկից ավելի պարամետրեր, ապա անվանական արգումենտները հնարավորություն են տալիս ուղարկվող արգումենտների հաջորդականությունը խախտել:
- Կազմակերպման համար օգտագործվում է (:) վերջակետ:

```
using System;
namespace albdarb
{
    class M
    {
        static void f(string name, byte age)
        {
            Console.WriteLine(name+age);
        }
        static void Main(string[] args)
        {
            f("alb ", 99);
            f(age: 100,name:"darb ");
        }
    }
}
```

.....

56.Մեթոդի ոչ պարտադիր պարամետրեր (C# 4)

[Шилдт-247; Нейгел-121; Уотсон-411]

- (Optional Paramters).
- Պարամետրերը կարող են լինել նաև ոչ պարտադիր: Այդ դեպքում չնշված արգումենտները պարամետրում արժեքավորվում են և պարամետրերի ցուցակում գտնվում են վերջում:
- Անվանակն պարամետրերը թույլատրված են մեթոդներում, կոնստրուկտորներում, ինդեքսատրներում, դելեգատներում:
- Օգտագործվում է վերագրման (=) օպերատոր:

```
using System;
namespace albdarb
{
    class M
    {
        static void f(string name, byte age=99)
        {
            Console.WriteLine(name+age);
        }
        static void Main(string[] args)
        {
            f("alb ");
        }
    }
}
```

//.....

- Պարամետրերը կարող են լինել լիտերալ, հաստատուն, նոր օբյեկտ, կամ չափային տիպի լռելիությամբ արժեք: Կան նաև որոշ սահմանափակումներ: (ստուգել)

```
using System;
namespace albdarb
{
    class M
    {
        const byte tariq=101;
        //byte tariq = 101; error!!!!!
        static void f(string name, byte age=tariq)
        {
            Console.WriteLine(name+age);
        }
        static void Main(string[] args)
        {
            f("alb ");
            f("darb ",100);
        }
    }
}
```

- Անվանական և ոչ պարտադիր արգումենտներ խորհուրդ է տրվում օգտագործել միայն անհրաժեշտության դեպքում: Օրինակ, շատ քանակությամբ պարամետրերի առկայության դեպքում, երբ կա նաև քիչ կազմով արգումենտներով կանչ:

.....

57.Մեթոդի հղող պարամետրեր - ref, out

[Троелсен-150; Шилдт-222; Нэш-63]

- Ինչպես գիտենք մեթոդը կարող է վերադարձնել չափային տիպի միայն մեկ արժեք return – ի միջոցով: Եթե անհրաժեշտ է վերադարձնել մեկից ավել արժեքներ, ապա դա հնարավոր է պարամետրերի չափային տիպերի հղումների միջոցով, օգտագործելով ref և out:
- ref – ի դեպքում փոփոխականները մեթոդից դուրս պետք է ստանան նախնական արժեքներ:
- out – ի դեպքում նախնական արժեքների վերագրման անհրաժեշտություն չկա, սակայն պարտադիր է նախնական արժեքները մեթոդում :
- ref և out հղող պարամետրերի օգտագործման ժամանակ ավտոմատ ներքին ձևով կատարվում է boxing/unboxing:

```
using System;
class M
{
    static void ff(out int a)
    {
        a = 4;
        a = a * a * a;
    }
    static void Main()
    {
        int b;
        ff(out b);
        Console.WriteLine(b);
    }
}
//.....
```

```
using System;
class M
{
    static void ff(ref int a)
    {
        a = a * a * a;
    }
    static void Main()
    {
        int b = 3;
        ff(ref b);
        Console.WriteLine(b);
    }
}
//.....................................................................

using System; //example 2
class A
{
    int H;
    int M;
    int S;
    public void ff()
    {
        Console.WriteLine("{0}:{1}:{2}", H, M, S);
    }
    //public void ff2(int h, int m, int s)
    public void ff2(ref int h, ref int m, out int s)
    {
        h = H;
        m = M;
        s = S;
    }
    public A(DateTime dt)
    {
        H = dt.Hour;
        M = dt.Minute;
        S = dt.Second;
    }
}
class M
{
    static void Main()
    {
        DateTime ct = DateTime.Now;
        A ob = new A(ct);
        ob.ff();
        int theH = 0; // int theH;  //  error ?
        int theM = 0;
        int theS;     // int theS = 0; //  OK !
        //ob.ff2(theH, theM , theS);
        ob.ff2(ref theH, ref theM, out theS);
        Console.WriteLine("best time: {0}:{1}:{2}", theH, theM, theS);
    }
}
```

- Միայն կոնստրուկտորի ներսում օգտագործվող մեթոդը կարող ունենալ պարամետրում ref, որը օգտվում է readonly դաշտից: [Հայ,63]

```
using System;
```

```
class A
{
    public readonly int x=886;
    public A()
    {
        x = 111;
        x = 999;
        ff(ref x);
    }
    void ff(ref int r)
    {
        r=555;
    }
}
class Program
{
    static void Main(string[] args)
    {
        A o = new A();
        Console.WriteLine(o.x);
    }
}
```

.....

58.ref, out հղիչային տիպերի համար

[Шилдт-227]

- Թույլատրվում է օգտագործել ref, out հղիչային տիպերի համար: Այդ դեպքում ինքը հղումն ուղղարկվում է ըստ հղման: Կամ այլ խոսքով ասած, հնարավորություն է տրվում մեթոդում փոփոխել օբյեկտը, որին կար հղում:

```
using System;
class A
{
    public int a;
    public A(int a)
    {
        this.a = a;
    }
}
class M
{
    static void Main(string[] args)
    {
        A ob = new A(1);
        f(ref ob);
        Console.WriteLine(ob.a);
    }
    static void f(ref A ob2)
    {
        ob2 = new A(2);
    }
}
//,,,
```

```
    static void Main(string[] args)
    {
        int a = 8;
        crate(a, ref Ar);
   //crate(a, Ar);
        foreach (int item in Ar)
            Console.WriteLine(item.ToString());
    }
    //static void crate(int n,  int[] Ar)// Rantime error!
    static void crate(int n, ref int[] Ar)
    {
        Ar = new int[n];
    }
}
```

.....

59.Մեթոդի պարամետր - params

[Троелсен-154; Шилдт-229]

- params պարամետրը հնարավորություն է տալիս մեթոդին աշխատել փոփոխվող քանակ ունեցող պարամետրերով: Այդ դեպքում պետք է նաև զանգվածի օգտագործում:
- Մեթոդը կարող է ունենալ նաև այլ պարամետրեր, միայն params –ը պետք է գտնվի պարամետրերի վերջում:
- Նույն հնարավորությունը կարելի է կատարել առանց params –ի օգտագործման, նույնպես զանգվածի միջոցով, ուղղակի նախնական նրան սկզբնարժեքներ տալով:

```
using System; // params
class AAA
{
    public int ff(params int[] mm)
    {
        int k;
        k = mm[0];
        for (int i = 1; i < mm.Length; i++)
            if (mm[i] < k) k = mm[i];
        return k;
    }
}
class M
{
    public static void Main()
    {
        AAA ob = new AAA();
        int min, a = 10, b = 20;
        min = ob.ff(a, b);
        Console.WriteLine("min = " + min);
        min = ob.ff(a, b, -1);
        Console.WriteLine("min = " + min);
        min = ob.ff(18, 28, 38, 8, 58);
        Console.WriteLine("min = " + min);
        int[] mm2 = { 55, 88, -3, 99, 44, 90, 234 };
        min = ob.ff(mm2);
        Console.WriteLine("min = " + min);
    }
}
```

```
Output
min = 10
min = -1
min = 8
min = -3
```

// Մեթոդը կարող է ունենալ նաև այլ պարամետրեր

```
using System;
class AAA
{
    public int ff(string s,params int[] mm)
    {
        int k = mm[0];
        for (int i = 1; i < mm.Length; i++)
            if (mm[i] < k) k = mm[i];
        Console.Write(s);
        return k;
    }
}
class M
{
    public static void Main()
    {
        AAA ob = new AAA();
        int min, a = 10, b = 20;
        min = ob.ff("min= ",a, b);
        Console.WriteLine(min);
        min = ob.ff("min= ",a, b, -1);
        Console.WriteLine(min);
        min = ob.ff("min= ",18, 28, 38, 8, 58);
        Console.WriteLine("min = " + min);
        int[] mm2 = { 55, 88, -3, 99, 44, 90, 234 };
        min = ob.ff("min= ",mm2);
        Console.WriteLine(min);
    }
}

// Առանց params –ի օրինակ.
using System;
class Program
{
      static void Main(string[] args)
      {
         int L=7;
         int[] a =new int [L];
         for (int i = 0; i < a.Length; i++)
         a[i] = i;
         f(a);
         Console.WriteLine();
         L = 10;
         a = new int[L];
         for (int i = 0; i < a.Length; i++)
         a[i] = i*i;
         f(a);
}
      static void f(int []aa)
      {
          for(int i=0;i<aa.Length;i++)
          Console.Write(aa[i]+" ");
      }
}
```

60.partial կլասներ, partial մեթոդներ

[Нэш-91; Нейгел-133] [Нэш-92; Троелсен-448]

- Մինչեվ C#2005 – ը կլասը գտնվում էր մեկ ֆայլում: Նոր թարգմանիչներում ներդրվեց հնարավորություն, երբ կլասը կարող է տեղակայվել տարբեր ֆայլերում:
- Հնարավորություն էր տրվում խուսափել մեծ չափերի ֆայլերից և հարմար խմբավորել անդամները:

```
// File1.cs
partial class A
{
    // Constructors.
    public A() { }
    // All public members.
    public void f1()
    {
        a = 555;
    }
    public void f2()
    {
        s = "barev";
    }
}
// File2.cs
partial class A
{
    int a;
    string s;
}
class M
{
    static void Main()
    {
        A aa = new A();
        aa.f1();
        aa.f2();
    }
}
//.....
```

partial Մեթոդներ

[Нэш-92; Троелсен-448]

- Մասնատված մեթոդների օգնությամբ կարելի է մեթոդը բաժանել նկարագրության և կիրառական մասերի: Այն կարող է հայտարավել մասնատված կլասի մի մասում, իսկ կողը գրել նույն կլասի այլ հատվածում: Եթե մեթոդը հայտարավել է, սակայն հետո չի իրականացվել, ապա վերջնական կոդում չի ընդգրկվում – գովելի առանձնահատկություն: (Նման է C++ ի .h և .cpp բաժանմանը):
- Մասնատված մեթոդները ունեն սահմանափակումներ

✓ Կարող են հայտարարվել միայն մասնակի կլասներում:
✓ Վերադարձնում են void:
✓ Չեն կարող ընդունել out պարամետր, սակայն թույլատրելի է ref – ը, ինչպես նաև params, this:
✓ Չեն կարող լինել external:
✓ Չեն կարող լինել virtual և չունեն թույլատրության մոդիֆիկատոր, քանի որ ոչ ակնհայտ նրանք փակ են:
✓ Չեն կարող լինել unsafe: Կարող են լինել ստատիկ ի հակառակ Հայ – ի:
✓ Կարող են ընդհանրացվել (generics) մասնակիորեն:

✓ Չեն կարող կանչվել դելեգատի միջոցով, որովհետև վերջնական թարգմանված կոդում նրանց լինելը երաշխավորված չէ:

- Մասնատված մեթոդները պայմանական կոմպիլացիայի (պրեպրոցեսորներ) #if, #elif, #endif օգտագործման խիստ տիպիզացված ձևն է:

```
using System;
class M
{
      static void Main()
      {
            A.f2();
      }
}
partial class A
{
      public static void f2()
      {
            f();
            Console.WriteLine("barev");
      }
      static partial void f();
}
//partial class A
//{
//     static partial void f()
//     {
//         Console.WriteLine("eli barev");
//     }
//}
```

.....

61.var - ոչ հստակ տիպայնացում

[Нейгел-72; Уотсон-400; Скит-226]

- Եթե անհրաժեշտ է տիպը որոշել արտահայտության կոնտեքստից, ապա օգտագործվում է var տիպը:
- int a=886; արտահայտությունը կարելի է գրել նաև հետևյալ ձևով.
  var a=886;
- var - ը դա ոչ հստակ տիպայնացում է, որը կոդը դարձնում է ավելի կարդացվող և բարձրացում ծրագրի կարգավորվածությունը:
- var - տիպը որոշվում է թարգմանության ժամանակ: Կոմպիլիարորի կողմից ձևավորված var տիպը հնարավոր չէ վերափոխել:
- var - տիպը կարելի է կիրառել միայն լոկալ փոփոխականների համար:
- var - տիպը անպայման պետք է արժեք ունենա:
- var - տիպը չի կարող ստանալ null արժեք:
- var - տիպի ինիցիալիզացիան պետք է լինի արտահայտություն:

.....

62.Անանուն տիպեր

[Шилдт-663; Уотсон-402; Нейгел-130; Skeet-239]

- Երբ անհրաժեշտ է ունենալ կլաս, որը միայն տվյալներ է պահում, ապա ցանկալի է օգտագործել անանուն տիպեր:
- var և new միջոցով կարելի է ստանալ անանուն տիպ:

Ենթադրենք ունենք կլաս.

```
class A
{
    public string ps { get; set; }
    public int n;
}
```

Նրան կարելի է արժեքավորել և օգտագործել հետևյալ ձևով.

```
        A o = new A
        {
            ps = "barev",
            n = 555
        };
```

Կամ անանուն տիպերով.

```
        var A = new
        {
            ps = "barev",
            n = 555
        };
```

- Անաուն տիպերի դեպքում չկա կլասի օբյեկտի իդենտիֆիկատոր, սակայն կոմպիլիատորը ստեղծում է իր կողմից ներքին ձևով:
- Լայն կիրառություն ունի բազաների հետ կատարվող աշխատանքի ժամանակ:

```
using System;
class A
{
    public string ps { get; set; }
    public int n;
}
class Program
{
    static void Main(string[] args)
    {
        A o = new A();
        o.n = 999;
        o.ps = "darb";
        Console.WriteLine(o.ps + o.n);
        A o1 = new A
        {
            ps = "barev",
            n = 555
        };
        Console.WriteLine(o1.ps+o1.n);
        var o2 = new
        {
            ps = "alb",
            n = 777
        };
        Console.WriteLine(o2.ps + o2.n);
        //o2.n = 111;//error
        //o2.ps = "errror"; // error
    }
```

```
}
//.............................................

using System;// Уотсон- 404
class M
{
    static void Main()
    {
        var curries = new[]
        {
            new
            {
            MainIngredient = "Lamb",
            Style = "Dhansak",
            Spiciness = 5
            },
            new
            {
            MainIngredient = "Lamb",
            Style = "Dhansak",
            Spiciness = 5
            },
            new
            {
            MainIngredient = "Chicken",
            Style = "Dhansak",
            Spiciness = 5
            }
        };
        Console.WriteLine(curries[0].GetHashCode());
        Console.WriteLine(curries[1].GetHashCode());
        Console.WriteLine(curries[2].GetHashCode());
        Console.WriteLine(curries[0].Equals(curries[1]));
        Console.WriteLine(curries[0].Equals(curries[2]));
        Console.WriteLine(curries[0] == curries[1]);
        Console.WriteLine(curries[0] == curries[2]);
    }
}
```

.....

63.Ընդլայնված մեթոդներ

[Троелсен-440; Шилдт -678; Нэш-477; Уотсон-417]

- Ինչպես գիտենք երբ տիպը որոշված և թարգմանված է հավաքագրման բլոկ, այն հիմնականում **փոփոխման ենթակա չէ**: Միակ ձևը փոփոխման դա ավելացնել նոր անդամներ և վերաթարգմանել: Կամ կարելի է օգտվել `System.Reflection.Emit` անունի տարածքից օպերատիվ հիշողությունում դինամիկ փոփոխություններ կատարելու համար:
- Ընդլայնված մեթոդները տալիս են նոր հնարավորություն– առանց վերաթարգմանման տիպում ստանալ նոր գործողություններ (մեթոդներ) տվյալ տիպի համար:
- Ընդլայնված մեթոդները պետք է հայտարավեն **ստատիկ կլասում**: Պետք է առաջին պարամետրը լինի **նշված `this`** – ով:
- Առանձնահատկություն: Եթե ստատիկ կլասը `public` է, ապա օգտագործողի կլասը նույնպես պետք է լինի ստատիկ: Կամ այլ խոսքով: Ընդլայնված մեթոդ օգտագործող կլասը նույնպես պետք է լինի `static`, հակառակ դեպքում ստացվում է թարգմանման սխալ: Օրինակում ներկայացված `class` A-ի և `class` M-ի `public` լինելը պատադիր չէ:

✓ Գրադարանի string տիպի օրինակ.

```
using System; // Հաշ-475
public static class A
{
    public static void ff(this String s)
    {
        Console.WriteLine("barev");
    }
}
public class M
{
    static void Main()
    {
        String str = "";
        // Call the extension method
        str.ff();
        // Call the same method the old way
        A.ff(str);
    }
}
// ---------------------------
```

✓ Օգտագործողի B տիպի օրինակ.

```
using System;
// public  static class Albert //error!!
static class Albert
{
    public static void  albdarb(this B ob,int n)
    {
        Console.WriteLine(++n);
    }
}
class B
{
    //public void albdarb(B ob, int n)
    //{
    //    Console.WriteLine(++n);
    //}
}
 class M
{
    static void Main()
    {
        B ob = new B();
        ob.albdarb(5);
    }
}
```

.....

64. Կանոնավոր արտահայտություններ

[Албахари –981; Нейгел -277,277]

- Կանոնավոր արտահայտությունները նախատեսված են տողերը հատուկ ձևով մշակելու համար:
- ✓ Կարելի է նշել կրկնվող սիմվոլներ հետագա մաքրման համար:
- ✓ Դարձնել բոլոր բառերի առաջին տառը մեծատառ:
- ✓ Երեք տառից ավել բառերի սկզբնատառը դարձնել մեծատառ:
- ✓ Առանձնացնել ինտերնտի URI հասցեի տարբեր էլեմենտներ:
- Նույն արդյունքին կարելի է հասնել String և StringBuilder կլասներով, սակայն ավելի աշխատատար կլինի:
- Սկզբունքորեն կանոնավոր արտահայտություններ ստանալու համար օգտվում ենք System. Text .RegularExpressions .RegEx կլասից: Այն պարունակում է բազում հատուկ սիմվոլներ:
- RegEx կլասի Match() մեթոդը ընդունում է պարամետրեր – տեքստը, շաբլոնը և անհրաժեշտ հատկություններ `RegexOptions` թվարկումով: (Օրինակ. `RegexOptions.IgnoreCase` նշում է անտեսել մեծատառ փոքրատառ):
- Match() մեթոդը հիշեցնում է `string` տիպի IndexOf() մեթոդին, այն տարբերությամբ, որ IndexOf() աշխատում է լիտերալի հետ, իսկ RegEx.Match() աշխատում է շաբլոնի հետ:
- Match() մեթոդը վերադարձնում է առաջին պատահած արդյունքը: Եթե անհրաժեշտ է հաջորը, ապա կարելի է կիրառել NextMatch() մեթոդը:

```
Match m1 = Regex.Match("One color? There are two colours in my head!",@"colo");
Match m2 = m1.NextMatch();
Console.WriteLine(m1.Index); // 4
Console.WriteLine(m2.Index); // 25
```

- Կարելի է ստանալ արդյունքների կոլեկտիա օգտվելով RegEx.Matches() մեթոդից:
- Matches() մեթոդը վերադարձնում է MatchCollection տիպի զանգված, որտեղ պահվում են արդյունքը և կարող ենք արտածել foreach ով:

Օրինակ:

- ✓ \b նշանակում բառի վերջ և սկիզբը: \balb գտնում է alb տառերով սկսվող բառերը: alb\b դեպքում ստուգում է վերջը:
- ✓ \S նշանակում է ցանկացած սիմվոլ բացի "պրոբելից":
- ✓ Ինչպես գիտենք * հին ծրագրերում նշանակում է ցանկացած քանակով սիմվոլ:
- ✓ \S* նշանակում է ցանկացած քանակությամբ ցանկացած սիմվոլ բացի "պրոբելից":
- ✓ @ նշանի կիրառում նշանակում է հատուկ սիմվոլների օգտագործում:
- ✓ Եթե անհրաժեշտ է հտուկ սիմվոլի կիրառություն օգտագործվում է թեք գիծ \: օրինակ աստղանիշ կարելի է ստանալ * հրամանով:

```
using System;
using System.Text.RegularExpressions;
class Program
{
    static void Main(string[] args)
    {
        string t = @"This (aionalbion) comprehensive (ionalb) compendium provides a broad and
thorough investigation of all aspects of programming with ASP.NET. Entirely revised and
updated for the fourth release of .NET, this book will give you the information you need to
master ASP.NET and build a dynamic, successful, enterprise Web application.";
        //const string pattern = "ion";
        //string pattern = @"ion\b";
        const string pattern = @"\ba\S*ion\b";
        MatchCollection mc = Regex.Matches(t,pattern, RegexOptions.IgnoreCase);
```

```
        foreach( Match m in mc)
        {
            Console.WriteLine(m.Index);
        }
    }
}
```

- **RegexOptions** թվրակման անդամները.

CultureInvariant : Specifi es that the culture of the string is ignored. **ExplicitCapture:** Modifi es the way the match is collected by making sure that valid captures are the ones that are explicitly named.
IgnoreCase: Ignores the case of the string that is input. Անտեսում է մեծատառը կամ փոքրատառը:
IgnorePatternWhitespace Removes unescaped whitespace from the string and enables comments that are specifi ed with the pound or hash sign.
Multiline: Changes the characters ^ and $ so that they are applied to the beginning and end of each line and not just to the beginning and end of the entire string.
RightToLeft :Causes the inputted string to be read from right to left instead of the default left to right (ideal for some Asian and other languages that are read in this direction).
Singleline: Specifi es a single-line mode where the meaning of the dot (.) is changed to match every character.

- **Հատուկ սիմվոլների ցուցակը:**

? = Նախորդ սիմվոլը կարող է կրկնվել 0 կամ 1 անգամ (Preceding character may be repeated zero or one time), այլ խոսքով ? ով նշում են, որ էլեմանտը պարտադիր չէ:

```
Console.WriteLine(Regex.IsMatch("color", @"colou?r")); // True
Console.WriteLine(Regex.Match("color", @"colou?r").Success); // True
Console.WriteLine(Regex.Match("colour", @"colou?r").Success); // True
Console.WriteLine(Regex.Match("colouur", @"colou?r").Success); // False
//------------------
Match m1 = Regex.Match("One color? There are two colours in my head!",@"colou?rs?");
Match m2 = m1.NextMatch();
Console.WriteLine(m1); // color
Console.WriteLine(m2); // colours
//------------------
Match m = Regex.Match("any colour you like", @"colou?r");
Console.WriteLine(m.Value); // colour
Console.WriteLine(m.Success); // True
Console.WriteLine(m.Index); // 4
Console.WriteLine(m.Length); // 6
```

[ ] = քառակուսի փակագծերը հնարավորություն են տալիս կատարել սիմվոլների ընտրություն: ma[n|p] արտահայտությունը նշում է man և map բառերը: Եթե նշվում է գծիկ, ապա դա նշանակում է միջակայք: [1-9] կամ [a-z]:

```
Console.WriteLine(Regex.Matches("That is that.", "[Tt]hat").Count); // 2
```

^ (բացի) = Ցանկացած սիմվոլ բացի [] փակագծերի նշվածները:

```
Console.Write(Regex.Match("quiz qwerty", "q[^aeiou]").Index); // 5
Console.Write(Regex.Match("b1-c4", @"[a-h]\d-[a-h]\d").Success); // True
```

\d նշանակում է ցանկացած թիվ
\D նշանակում է ոչ թվային սիմվոլ
\w նշանակում է տառեր, թվեր և տակի գիծ

^ = Տեքստի սկիզբ

```
Console.WriteLine(Regex.Match("Not now", "^[Nn]o")); // No
```

$ = Տեքստի վեջը

```
Console.WriteLine(Regex.Match("f = 0.2F", "[Ff]$")); // F
```

. = Ցանկացած միայնակ սիմվոլ բացի (\ n) տողադարձի

***** = Նախորդ սիմվոլը կարող է կրկնվել 0 կամ n անգամ

```
Console.WriteLine(Regex.Match("cv15.doc", @"cv\d*\.doc").Success); // True
Console.WriteLine(Regex.Match("cvjoint.doc", @"cv.*\.doc").Success); // True
```

+ = Նախորդ սիմվոլը կարող է կրկնվել 1 կամ n անգամ

```
Console.WriteLine(Regex.Matches("slow! yeah slooow!", "slo+w").Count); // 2
```

{n} = ստույգ n համընկնում

{n,} = ամենաքիչը n համընկնում

{n,m} = համընկնում {n,m} միջակայքում

```
Regex bp = new Regex(@"\d{2,3}/\d{2,3}");
Console.WriteLine(bp.Match("It used to be 160/110")); // 160/110
Console.WriteLine(bp.Match("Now it's only 115/75")); // 115/75
```

\s = Ցանկացած "պրոբել" սիմվոլ

\S = Ցանկացած սիմվոլ, որ "պրոբել" չի

\b = Բառի սահման

\B = Ցանկացած դիրք բացի բառի սահմանից

- Կանոնավոր արտահայտությունների կարևոր հատկություններից է նիշերի խմբավորումը:
- ( ) կլոր փակագծերով կարելի է խմբավորել ցանկացած հատված: Օրինակ. (an)+ հարամանը հայտնաբերում է ցանկացած an կրկնվող տեքստ: bananas came to Europe late in the annals of history տեքստում կհայտնաբերի an, anan, ananan ենթատեքստերը: an+ ի դեպքում կհայտնաբերի ann, որը հանդիպում է annals բառում:

```
Console.WriteLine(Regex.IsMatch("Jenny", "Jen(ny|nifer)")); // True
```

Օրինակ 2

Ինտերնետում գործում է հետևյալ կառույցը. <protokol>://<adress>:<port>

http://www.albdarb.com: 4355

✓ \ b (\ S+) : // ([^ :]+) (? :: (\S+)) ? \b

\ b նշում է սկիզբը և վեջը տեքստի որոշակի մասի:

✓ առաջին խունբ (\ S+) : // = http://

նշանակում է մեկ կամ ավել ոչ պրոբելային սիմվոլներ, որից հետո գալիս է : // նշանները

✓ երկրորդ խումբ ([^ :]+) = www.albdarb.com

նշում է կայքը, որը կարող է վեջանալ : ով, որ նշանակում է հաջրդ խմբի արկայություն:

✓ երրորդ խումբ (? :: (\S+)) ? = : 4355

վերջին ? հարցականը նշում է, որ : 4355 կարող է և չլինել:

Սակայն կարող է լինել միայն : սիմվոլը, որը կազմակերպվում է (\S+) ներդրված խմբով:

? : : հրամանում ? : նշանակում է կարելի է չպահել այս խումբը, իսկ երկրորդ : վերջակետը նշում է փնտրվող տեքստը:

……

Օբյեկտ կողմնորոշվածություն

65.Ժառանգականություն (Inheritance) , protected

[Троелсен-231,240; Шилдт-329]

- Ժառանգականությունը օբյեկտ կողմնորոշված ծրագրավորման գլխավոր հենքերից է: Այն հնարավորություն է տալիս կոդի բազմակի օգտագործում (code reuse):
- Ժառանգականությունը ապահովում է “պատկանում է” ենթակայության հարաբերություն (”is-a”): Նշենք, որ մյուս ենթակայության սկզբունքը դա “ընդգրկում է”, որը կատարվում է կոմպոզիցիայի միջոցով (has-a):
- Ժառանգականության ժամանակ կարելի է օգտագործել մասնակի պաշտպանվածության ձև protected իդենտիֆիկատորի միջոցով, որը գործում է միայն ժառանգության սահմաններում:
- Ժառանգականության ժամանակ ավտոմատ կատարվում է լռելիությամբ կոնստրուկտորները և կոնստրուկտորների կատարման հաջորդականությունը կատարվում է բազայինից դեպի ժառանգ կլաս:
- Կլասը ժառանգում է օգտագործելով վերջակետ սիմվոլը: Իտարբերություն C++ ի, շարփում բազմաժառանգում չի թույլատրվում:

```
using System;
class A
{
    //public void fA()
    protected void fA()
    {
        Console.WriteLine("f A");
    }
}
class B : A
{
    public void fB()
    {
        fA();
        Console.WriteLine("f B");
    }
}
class M
{
    public static void Main()
    {
        A ob = new A();
        //ob.fA();//error!
        B ob2 = new B();
        ob2.fB();
        //ob2.fA();// error, if -  protected void fA()
        //ob2.fA();// ok,  if  - public void fA()
    }
}
```

.....

66.Ժառանգում base, new

[Шилдт-343; Нейгел-144]

- Կլասի անդամը կարող է համընկնել բազայինի հետ և ծածկում է բազայինին: Բազայինին դիմելու համար օգտվում ենք base - ից:
- new –ն միջոց է զգուշացման դեմ, այսինքն տեղյակ ենք համընկման մասին:
- Համընկնող բազային անդամին կարելի է դիմել նաև տիպերի վերափոխման մեթոդով (explicit): կանցնենք ավելի ուշ:

```
using System;
class A
{
    protected int a = 9;
    public void ff()
    {
        Console.WriteLine("ffA  " + a);
    }
}
class B : A
{
    new int a = 88;          // int a=88; //no error// warning
    public new void ff()     // public void ff() // no error
    {
        Console.WriteLine("ffB  " + a);
        Console.WriteLine("ffB  " + base.a);
    }
}
class M
{
    public static void Main()
    {
        A ob = new A();
        ob.ff();
        B ob2 = new B();
        ob2.ff();
        ((A)ob2).ff();   // explicit
    }
}
```

```
Output
ffA  9
ffB  88
ffB  9
ffA  9
```

.....

67.Բազային կոնստրուկտոր և :base

[Троелсен-238; Шилдт-339; Нейгел-150]

- Այսպիսի անհրաժեշտություն առաջանում է, երբ ժառանգ և բազային կլասի կոնստրուկտորները ունեն պարամետրեր և կատարվում է պարամետրի փոխանցում:
- Սկզբից նշենք կոնստրուկտորների կանչման հաջորդականությունը.

```
// Կոնստրուկտորների կանչման հաջորդականությունը սկսում է բազայինից
class A
{
    public A()
    {
        Console.WriteLine("aaaaaa");
    }
    protected A(int n)
    {
        Console.WriteLine(n);
    }
    protected void ff()
    {
```

```
        Console.WriteLine("es ff - n em");
    }
}
class B : A
{
    public B()
    {
        Console.WriteLine("bbbbbb");
    }
}
class C : B
{
    public C()
    {
        Console.WriteLine("cccccc");
    }
    public void fC()
    {
        ff();
    }
}
class Program
{
    static void Main(string[] args)
    {
        C ob = new C();
        ob.fC();
        // A ob2 = new A(4); //error
    }
}
//---------------------------------
```

- Այժմ փորձենք հասկանալ պարամետրով կոնստրուկտորը:

```
class A
{
    public A(int a)
    {     }
}
class B : A
{   }
class Program
{
    static void Main(string[] args)
    {        // error !!    }
}
// ----------------------------------------------------
class A
{
    //public A(){  }
    public A(int a)    {   }
}
class B : A
{
    public B()
    {
    }
    public B(char c, int a): base(a)
    {
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        B ob = new B(); // error !
    }
}
//-----------------------------------------------------------
```

- Եվ այպես ցուցադրենք, թէ ինչպես է պարամետր փոխանցվում բազային կոնստրուկտորին:

```
using System;
class A
{
    public A(int m)
    {
        Console.WriteLine("aaaaa " + m);
    }
}
class B : A
{
    public B(char c, int m) : base(m)
    {
        Console.WriteLine("bbbbb " + c);
    }
}
class M
{
    static void Main()
    {
        B ob = new B('B', 9);
    }
}
```

```
Output
aaaaa 9
bbbbb B
```

.....

68.Ներդրված տիպերի վերափոխարկում

[Нейгел-228; Троелсен-135]

- Ներդրված տիպրը թույլ են տալիս կատարել վերափոխարկում, եթե նրանք նույն ընտանիքից են: Եթե տիպերը տարբեր են, ապա վերափոխրկում կարելի է կատարել հատուկ միջոցներով, օրինակ օգտվելով `Convert` կլասից:

```
int a = 123;
long b = a;  // Implicit conversion from int to long
int c = ( int ) b; // Explicit conversion from long to int
static void Main(string[] args)
{
        int a = 99;
        uint b = 886;
        a = b;//error
        b = a;//error
        a = (int)b;
        b =(uint) a;
        bool b = true;
        int a = 99;
        a = Convert.ToInt32(b);
        b = Convert.ToBoolean(a);
}
```

69.Կոնվերտացիաներ

[Covaci – 122; Kort-185]

- `Convert, Parse, TryParse, BitConverter`
- `Convert` class to convert between base types. The supported base types are Boolean, Char, SByte, Byte, Int16, Int32, Int64, UInt16, Uint32, Uint64, Single, Double, Decimal, DateTime, and String. The difference between Parse/TryParse and Convert is that Convert enables `null` values. It doesn't throw an ArgumentNullException.

```
int i = Convert.ToInt32(null);
Console.WriteLine(i); // Displays 0
```

- A difference between Convert and the Parse methods is that Parse takes a string only as input, while Convert can also take other base types as input. The Parse and TryParse methods can be used when you have a `string` that you want to convert to a specific data type.
- The System. `BitConverter` class provides methods to convert values to and from arrays of bytes. The GetBytes method returns an array of bytes representing the value that you pass to it. The System.BitConverter class also provides methods such as ToInt32 and ToSingle to convert byte values stored in arrays back into specific data types.

```
using System;
class M
{
    static void Main()
    {
        string v = "true";
        bool b = bool.Parse(v);
        Console.WriteLine(b);
        b = Convert.ToBoolean(v);
        Console.WriteLine(b);
        string v2 = "1";
        int result;
        bool success = int.TryParse(v2, out result);
        if (success)
        {
            Console.WriteLine("valid integer");
            Console.WriteLine(v2);
            Console.WriteLine(result);
        }
        else
        {
            Console.WriteLine(" not valid integer");
        }
        Console.WriteLine("----------------");
        int packedValue=12345;
        byte[] valueBytes = BitConverter.GetBytes(packedValue);
        foreach(byte x in valueBytes)
            Console.WriteLine(x);
        Console.WriteLine("----------------");
        short value1, value2;
        value1 = BitConverter.ToInt16(valueBytes, 0);
        value2 = BitConverter.ToInt16(valueBytes, 2);
        Console.WriteLine(value1);
        Console.WriteLine(value2);
    }
}
```

.....

70.Ժառանգական կապվածությամբ տիպերի վերափոխարկում

[Троелсен-434,403;Нейгел-233]

- Տարբեր տիպեր, որոնք ստեղծվում են օգտագործողի կողմից (կլասներ, ստրուկտուրաներ), չի թույլատրվում կատարել վերագրում: Սակայն կարելի կատարել վերագրում, եթե այդ տիպերը կապված են ժառանգականությամբ: Վերագրել այդ դեպքում կարելի է սովորական ձևով կամ ակնհայտ ձևով օգտագործելով կլոր փակագծեր:

```
class A
{
}
class B : A
{
}
class M
{
    static void Main()
    {

        A a = new A();
        B b = new B();
        a = b; //ok
        //b = a; // error
        b = (B)a; // ok
        //-----------------------------------
        // Implicit cast between derived to base.
        A aa = new B(); // սովորական ձև
        // Must explicitly cast to store base reference in derived type.
        B bb = (B)aa; // ակնհայտ ձև
    }
}
```

.....

71.Հղում բազային կլասով

[Шилдт-251]

- Նույն կլասի տարբեր օբյեկտների ցուցիչներ կարելի է վերագրել, իսկ տարբեր կլասների ոչ: Եթե նրանք ժառանգականությամբ են կապված, ապա կա բացառություն: Բազային ցուցիչին կարելի է վերագրել ժառանգի ցուցիչ: Այդ դեպքում նա հղում է ժառանգ կլասին: Այդ ժառանգ ցուցիչով կարող է դիմել բազային կլասի անդամներին, իսկ ժառանգի անդամներին միևնույնն է ոչ:

```
using System;
class A
{
    public int a = 9;
}
class B : A
{
    public int b = 88;
}
class M
{
    public static void Main()
    {
        A obA = new A();
        B obB = new B();
        Console.WriteLine(obA.a);
        obA = obB;
        //obB=obA; //error
        obA.a = 44;
```

output
9
44

```
        Console.WriteLine(obA.a);
        //obA.b=55;//error
    }
}
```

72.Վիրտուալ մեթոդներ, Պոլիմորֆիզմ (Polymorphism)

[Шилдт-335, 43; Троелсен-245]

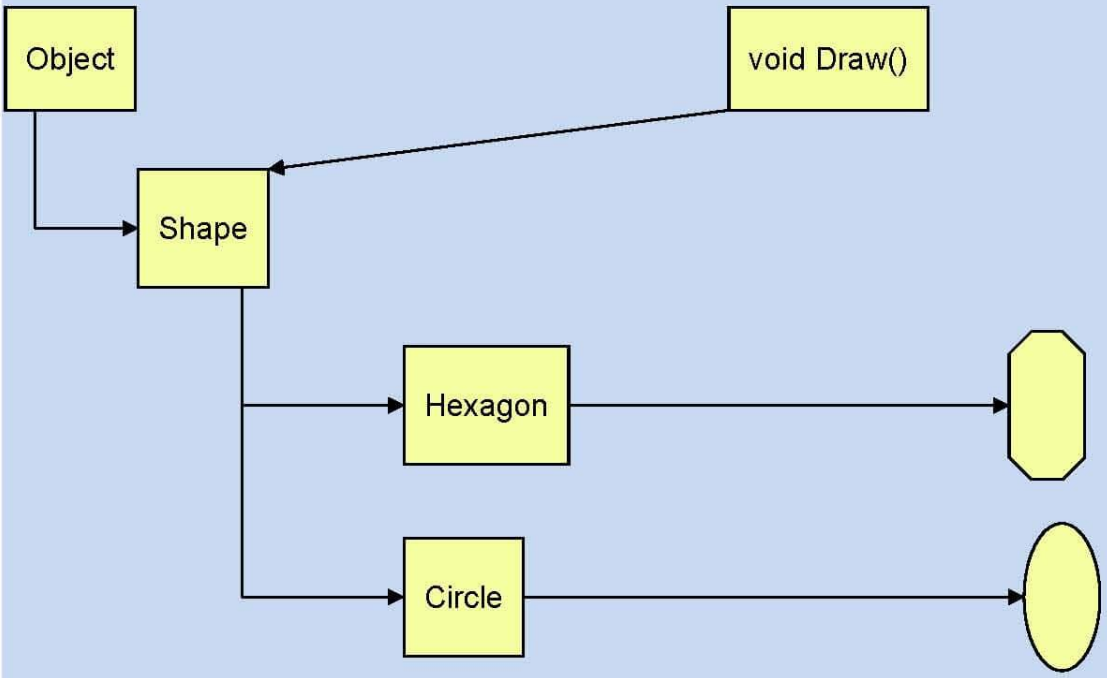
- Եթե անհրաժեշտ է բազային ցուցիչով դիմել ժառանգ կլասի մեթոդներրին, ապա մեթոդները պետք է լինեն և բազայինում և ժառանգում, ունենան նույն անունը, պարամետրերը և բազայինը լինի վիրտուալ:
- Առանց վիրտուալ մեթոդի բազային ցուցիչին կարելի է վերագրել ածանցյալ կլասի ցուցիչ և բազային ցուցիչով կարելի է դիմել միայն բազային անդամներին: Եթե նույն անունով մեթոդ կա, ապա այդ մեթոդը վիրտուալացնելով կարելի է հնարավորությունը մեծացնել - դիմել ածանցյալ կլասին:
- Այսպիսով - բազային ցուցիչով կարելի է դիմել ածանցյալ կլասի նույն անունով մեթոդներին:
- virtual – բազային մեթոդի սպեցիֆիկատր
- override – ժառանգ մեթոդի սպեցիֆիկատր

```
using System;
class A
{
    public virtual void ff()
    {
        Console.WriteLine("ffA  ");
    }
}
class B : A
{
    public override void ff()
    {
        Console.WriteLine("ffB  ");
    }
}
class D : A
{
    public override void ff()
    {
        Console.WriteLine("ffD  ");
    }
}
class M
{
    public static void Main()
    {
        A ob = new A();
        ob.ff();
        B ob2 = new B();
        ob = ob2;
        ob.ff();
        D ob3 = new D();
        ob = ob3;
        ob.ff();
    }
}
```

```
output
ffA
ffB
ffD
```

Պոլիմորֆիզմ (Polymorphism) [Троелсен-245; Шилдт-43]

- Բազմա - ձևություն: Գաղափարը վերցրված է “բիոլոգիայից”:
- Թարգմանման ժամանակ պոլիմորֆիզմ (ստատիկ պոլիմորֆիզմ) – իրագործվում է գերբեռնված մեթոդների միջոցով:
- Կատարման ժամանակ պոլիմորֆիզմ (դինամիկ պոլիմորֆիզմ) - իրագործվում է վիրտուալ մեթոդների օգնությամբ:



……

73.Տիպերի վերափոխարկում implicit, explicit

[Троелсен-435,403; Нейгел-230]

- Եթե անհրաժեշտ է տարբեր տիպերի օբյեկտներ վերագրել, որոնք կապված չեն ժառանգականությամբ, ապա պետք է օգտագործել օպերատորների գերբեռնում: implicit –ի միջոցով կատարվում է տիպերի թաքնված (ենթադրյալ) վերափոխարկում: explicit –ի միջոցով կատարվում է տիպերի բացահայտ վերափոխարկում ( ) փակագծի օգտագործումով:

```
using System; // implicit
namespace ConsoleAppImplicit
{
    class A
    {
        public uint dollar;
        public ushort cent;
        public A(uint d, ushort c)
        {
            dollar = d;
            cent = c;
        }
        public static implicit operator float(A ob) // **
        {                                           //
            return ob.dollar + ob.cent/100f;        //
        }                                           //
    }
    class Program
    {
        static void Main(string[] args)
        {
```

```
            A ob = new A(55, 99);
            float f = ob;// ** error!!
            Console.WriteLine(f);
        }
    }
}
//-----------------------------------------
using System; // explicit
namespace ConsoleAppExplicit
{
    class A
    {
        public uint dollar;
        public ushort cent;
        public A(uint d, ushort c)
        {
            dollar = d;
            cent = c;
        }
        public static explicit operator A(float f)   // **
        {                                            //
             uint  dd= (uint)f;                      //
             ushort cc = (ushort)((f-dd)*100);       //
             return   new A(dd,cc);                  //
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            float f=33.77f;
            A ob = (A)f;// ** error!!
            Console.WriteLine(ob.dollar+"   " + ob.cent);
        }
    }
}
```

.....

74.Մեկուսացված մեթոդներ և կլասներ, sealed

[Троелсен-248, 234; Шилдт-367; Нейгел-146]

- Երբեմն անհրաժեշտ է լինում կլասի ժառանգականությունը արգելել ոչ թէ ամբողջ կլասի համար, այլ որոշ անդամների համար: Դա հանարավոր է, եթե անդամները վիրտուալ են և օգտագործում են sealed սահմանափակումը:

```
class A
{
    protected virtual void f()
    {
    }
}
class B : A
{
    protected override sealed void f()
    {
    }
}
class D : B
{
    protected override void f()   //error !!
```

```
    {      }
}
class M
{
    static void Main()
    {
    }
}
```

- **Մեկուսացված կլասներ**

✓ Մեկուսացված կլասները դրանք անժառանգ կլասներ են, որոնց արգելված է դարձնել բազային կլաս և կատարել նրանցից ժառանգում:

✓ Իրականցվում է sealed – ի միջոցով:

```
sealed class B:A
{
        int a;
}
```

……

75.Աբստրակտ կլասներ

[Троелсен-249; Либерти-136; Шилдт-363]

- Կլասը կարող է լինել աբստրակտ: Այս ձևով արգելվում է նրա օբյեկտի ստեղծումը:
- Եթե կա անհրաժեշտություն ժառանգ կլասում պարտադիր իրականացնել բազային կլասի մեթոդները, ապա բազային կլասում մեթոդը հայտարարվում է աբստրակտ: Այդ դեպքում կլասը ավտոմատ դառնում է աբստրակտ:
- Աբստրակտ մեթոդը բազային կլասում չի կարող ունենալ իրականացում:
- Չի թույլատրվում ստեղծել աբստրակտ կլասի օբյեկտ և այն նույնանման է փակ կոնստրուկտորի կիրառմանը:

```
abstract class A
{
   abstract public void f();
}
```

- Ժառանգ կլասում մեթոդի համար օգտագործվում է override սպեցիֆիկատորը:

```
using System;
abstract class A
{
    abstract public void f();
}
sealed class B : A
{
    public override void f()
    {
        Console.WriteLine("ok!");
    }
}
class M
{
    static void Main()
    {
        B ob = new B();
        ob.f();
        //A ob = new A();// error !
    }
}
```

76.Ներդրված կլասներ

[Троелсен-243; Либерти-141]

- Թույլատրվում է կլասի մեջ կլաս հայտարարել: Դա կոչվում է ներդրված կլաս:
- Եթե ներդրված կլասի անդամը փակ է, ապա չի կարող օգտվել ոչ արտաքին կլասը, ոչ այլ կլասներ:
- Ներդրված կլասը կարող է օգտագործել արտաքին կլասի փակ անդամները:
- Եթե ներդրված կլասն է փակ, ապա արտաքին կլասը կարող օգտվել ներդրված կլասից, սակայն չեն կարող օգտվել այլ կլասներ:
- Հարց: Կլասը ինչ մոդոֆիկատոր կարող է ընդունել: Պատասխան: Բոլոր թույլատրության մոդիֆիկատորները, քանի որ, կլասը հնարավոր է լինի ներդրված: Իսկ ինչպես գիտենք, եթե կլասը ներդրված չէ, ապա միայն public և internal: (լռելիությամբ internal):

```
using System;
class A
{
    public void f()
    {
        Console.WriteLine("fA");
        B ob3 = new B();
        ob3.f2();
    }
    public class B
    {
        public void f2()
        {
            Console.WriteLine("fB");
            M ob5 = new M();
            Console.WriteLine(ob5.a);
        }
    }
}

class M
{
    public int a = new int();
    public static void Main()
    {
        A ob = new A();
        ob.f();
        A.B ob2 = new A.B();
        ob2.f2();
        Console.WriteLine(ob);
        Console.WriteLine(ob2);//A + B
    }
}
```

```
output
fA
fB
0
fB
0
A
A+B
```

.....

Օբյեկտ

77.System.Object

[Троелсен-258; Нэш-114; Шилдт-368; Нейгел-135]

- Բոլոր տիպերը առաջացել են `System.Object` կլասից, նույնիսկ չափային տիպերը:
- Նկարագրենք `Object` կլասի անդամները.

```
namespace System
{
    public class Object
    {
        public virtual Boolean Equals(Object obj);
        public virtual Int32 GetHashCode();
        public Type GetType();
        public virtual String ToString();
        protected virtual void Finalize();
        protected Object MemberwiseClone();
        public static bool Equals(object objA, object objB);
        public static bool ReferenceEquals(object objA, object objB);
    }
}
```

- virtual Equals( ) - Վերադարձնում է true, եթե երկու հղումներ նույն օբյեկտն են նշում: Օբյեկտները համեմատելու համար անհրաժեշտ է վերաորոշել:
- virtual GetHashCode( ) – Վերադարձնում է ամբողջ թիվ, որը ցույց է տալիս օբյեկտի իդենտիֆիկացիոն համարը: Այդ համարը վերագրվում է օբյեկտին նրա ստեղծման ժամանակ և ուղեկցվում է մինչև նրա վերանալը: Համարակալումը սկսում է 2 – ից:
- GetType( ) – Վերադարձնում է օբյեկտի տիպը իր անունի տարածքով:
- virtual ToString( ) - Լռելիությամբ վերադարձնում է օբյեկտի անունը իր անունի տարածքով:
- protected virtual Finalize( ) - Ազատում է ռեսուրսները: Կանչվում է դեստրուկտրի կողմից:
- protected MemberwiseClone() - Ստեղծում է օբյկտի մակերեսային կոպիան:
- static Equals( ) - Ստուգում է երկու օբյեկտներ:
- static ReferenceEquals() - Ստուգում է երկու հղումները:

```
using System;
class A
{
    public int a;
    public string name;
    public A(string s1, int a1)
    {
        a = a1;
        name = s1;
    }
}
class M
{
    static void Main()
    {
        A ob1 = new A("ani", 10);
        A ob2 = new A("nune", 10);
        Console.WriteLine("ToString:{0}", ob1.ToString());
        Console.WriteLine("Type:{0}", ob2.GetType());
        Console.WriteLine("hashcode ob1: " + ob1.GetHashCode());
        Console.WriteLine("hashcode ob2: " + ob2.GetHashCode());
    }
}
```

78.Object կլասի ToString() անդամի վերաորոշումը

[Троелсен-261; Нейгел-137]

```
public virtual String ToString();
```

- Կարելի է վերավորոշել ToString( ) - ը, քանի որ այն virtual է: Դա կատարվում է այն դեպքում, երբ օգտագործողը ցանկանում է տեսնել օբյեկտի տեքստային ներկայացումը իր ձևով: Ուղղակի պետք չէ քմահաճորեն այն ներկայացնել: ToString( ) - ը պետք է արտապատկերի ավելի բնորոշիչ ձևով:

```
using System;
class A
{
    private int a;
    public A(int aa)
    {
        a = aa;
    }
    public override string ToString() //
    {
        return a.ToString(); //
    }               //
}
class M
{
    static void Main()
    {
        int b = 5;
        Console.WriteLine(b.ToString());
        A ob = new A(7);
        Console.WriteLine(ob.ToString());
    }
}
```

Output:
5
7

Output: //
5
AAA

.....

79.Object կլասի Equals() անդամի վերաորոշումը

[Троелсен-261; Нэш-444,447; Рихтер-155]

```
public virtual Boolean Equals(Object obj);
```

- Equals() - ը համեմատում է երկու օբյեկտների հղումները: Եթե անհրաժեշտ է համեմատել օբյեկտի պարունակությունը, ապա այն պետք է վերավորոշել, քանի որ virtual է:

```
using System;
class A
{
    private int a;
    public A(int aa)
    {
        a = aa;
    }
    public override bool Equals(object obj)
    {
            A temp;
            temp = (A)obj;
            //if (((A)obj).a == this.a)
            if (temp.a == this.a)
            {
                return true;
            }
```

```
            else
            {
                return false;
            }

        }
}
class M
{
    static void Main()
    {
        A ob = new A(99);
        A ob2 = new A(99);
        if (ob2.Equals(ob))
            Console.WriteLine("Equals");
        else
            Console.WriteLine("not equals");
    }
}
```

- Հարց: ինչու է թարգմանիչը տալիս “warning” Rebuild Solution թարգմանությունով: Որովհետև, եթե վերավորոշվել է Equals() վիրտուալ ֆունկցիան, ապա ցանկալի է վերաորոշել նաև GetHashCode( ) վիրտուլ ֆունկցիան: Հակարակ դեպքում կոմպիլիատորը տալիս է զգուշացում:

……

80.GetHashCode() մեթոդի վերաորոշումը

[Троелсен - 262, 258; Нэш-448]

- Օբյեկտը իր ստեղծման ժամանակ ստանում է նաև Hash կոդ, որպեսզի տարբերվի մյուս օբյեկտներից: Կոդի գեներացման ժամանկ համակարգը օգտվում է օբյեկտի ընթացիք զբաղեցրած հասցեից և ստանում թիվ:
- Գոյություն ունեն տարբեր ալգորիթմներ, որոնք կարող են վերատադրել Hash կոդ: Պարզագույն ձևերից է օրինակ `ToString().GetHashCode()` հրամանը:
- Օրինակը ցուցադրում է, թէ ինչպես միևնույն int a արժեքի դեպքում Hash կոդը նույնն է ստացվում:

```
using System;
class A
{
    public   int a;
    public A(int aa)
    {
        a = aa;
    }
    //public override int GetHashCode()
    //{
    //    return a.ToString().GetHashCode();
    //}
    public override bool Equals(object obj)
    {
        A temp = (A)obj;
        if (temp.a == this.a)
        {
            return true;
        }
        else
        {
            return false;
```

```
        }
    }
}
class M
{
    static void Main()
    {
        A ob = new A(99);
        A ob2 = new A(99);
        Console.WriteLine(ob.GetHashCode());
        Console.WriteLine(ob2.GetHashCode());
        if (ob2.Equals(ob))
            Console.WriteLine("Equals");
        else
            Console.WriteLine("not equals");
    }
}
```

- **GetHashCode()** ի վերաորոշումը միայն **Warning** ի դեմ չէ: Այն ավելի կարևոր և օգտակար կիրառում ունի: Օրինակ, Hashtable կոլեկցիայում (System.Collections անունի տարածք), տվյալներ գրելու համար անհրաժեշտ է երկուսը, և Equals և GetHashCode վերորոշել, որովհետև Hashtable ը ճիշտ օբյեկտ ընտրելու համար կանչում է Equals() և GetHashCode() մեթոդները: Այլ խոսքով, եթե անհրաժեշտ է A կլասի օբյեկտները տեղադրել Hashtable կոլեկցիայում, ապա այդ դեպքում պարտադիր է երկու վերաորոշումները:

……

81.Object կլասի static անդամները

[Троелсен-264; Нейгел-218]

```
public static bool Equals(object objA, object objB);
public static bool ReferenceEquals(object objA, object objB);
```

- Ստատիկ **Equals()** մեթոդը նման է օբյեկտի **Equals()** մեթոդին, ուղղակի այն հնարավոր չէ վերաորոշել:
- Ստատիկ **ReferenceEquals()** մեթոդը համեմատում է միայն հղումները:
- Եթե համեմատվող օբյեկտների տիպը հղիչային է , ապա **Equals()** և **ReferenceEquals()** մեթոդները տալիս են նույն արդյունքը, այսինքն համեմատում են հղումները:
- Եթե համեմատվող օբյեկտների տիպը չափային է (ասենք ստրուկտուրա է), ապա **Equals()** և **ReferenceEquals()** մեթոդները տալիս են տարբեր արդյունքներ, այսինքն **Equals()** համեմատում է օբյեկտները իսկ **ReferenceEquals()** -ը համեմատում է հղումները:

```
using System;
//struct A //struct – value type
class A
{
    public int a;
    public string name;
    public A(string s1, int a1)
    {
        a = a1;
        name = s1;
    }
}
class M
{
    static void Main()
```

```
    {
        A ob1 = new A("Ani", 10);
        A ob2 = new A("Ani", 10);
        //Console.WriteLine("Working with value types"); // struct
        Console.WriteLine("Working with reference types"); //class
        Console.WriteLine(Equals(ob1, ob2)); //false
        Console.WriteLine(ob1.Equals(ob2)); //false
        Console.WriteLine(ReferenceEquals(ob1, ob2)); //false
    }
}
```

.....

82.Object.MemberwiseClone()

[Հայ-115, 427]

- **MemberwiseClone()** մեթոդը հնարավորություն է տալիս կատարել օբյեկտների **մակերեսային** կոպիա: Դա նշանակում է, որ չափային տիպերը կրկնօրինակվում են կատարելով բիթ առ բիթ կոպիա, իսկ օբյեկտի կոմպոզիցաները (հղիչային օբյեկտները) հղում են նույն օբյեկտին, այսինքն չկա օբյեկտի կոպիա:
- Եթե այսպիսի կոպիան չի բավարարում, ապա կա հնարավորություն կատարել ավելի լիարժեք կոպիա, օգտվելով **Icloneable** ինտերֆեյսից կատարելով նրա բոլոր պահանջները, մասնավորապես **Clone()** մեթոդի իրականացումը, որը կանցնենք ավելի ուշ:
- protected -ը ստիպում է սահմանափակվել այն հնարավորությունով, որ միայն կոպիա կատարող կլասը կարող է կատարել **MemberwiseClone()** հրամանը չօգտվելով new – ից և կոնստրուկտորի կանչից: public – ի դեպքում իրավիճակը ավելի կբարդանար այլ կլասների միջամտման պատճառով:

```
//using System;
class A
{
    public void f()
    {
        A ob = new A();
        A ob2 = new A();
        ob2 = (A)ob.MemberwiseClone(); //ok!
    }
}
class Program
{
    static void Main(string[] args)
    {
        A ob = new A();
        A ob2 = new A();
        ob2 = (A)ob.MemberwiseClone(); // error ? // why?
    }
}
//----------------------------
```

- Հաջորդ օրինակը ցույց է տալիս, որ **B** կլասի **N** –ին, եթե վերագրում ենք **1990**, ապա երկու օբյեկտների համար այն նույնն է կամ այլ խոսքով հղում են նույն արժեքին, քանի որ կա մեկ կոպիա, դրա համար էլ կոչվում է մակերեսային կոպիա: **A** կլասի **name** դաշտը օգտագործվում է ուղղակի առանց կոմպոզիցիայի, դրա համար նրանից կա երկու օրինակ, **"Ani"** և **"Nune"** տարբեր արժեքներ տարբեր օբյեկտներին համար:

```
using System;
class B
{
    public int N;
    public B(int n)
    {
        N = n;
    }
}
class A
{
    public string name;
    public B b;
}
class M : A
{
    public static void Main()
    {
        M ob1 = new M();
        ob1.name = "Ani";
        ob1.b = new B(1987);
        M ob2 = (M)ob1.MemberwiseClone();
        Console.WriteLine("ob1  values...");
        Console.WriteLine("{0} : {1}", ob1.name, ob1.b.N);
        Console.WriteLine("ob2  values...");
        Console.WriteLine("{0} : {1}", ob2.name, ob2.b.N);
        Console.WriteLine("------------------");
        ob1.name = "Nune";
        ob1.b.N = 1990;
        Console.WriteLine("ob1  values...");
        Console.WriteLine("{0} : {1}", ob1.name, ob1.b.N);
        Console.WriteLine("ob2  values...");
        Console.WriteLine("{0} : {1}", ob2.name, ob2.b.N);
    }
}
```

.....

83.object - որպես տիպերի ընդանրություն

[Шилдт-372]

- `object` տիպը և `Object` կլասը ընդգրկում են նույն գործողությունները ունենալով լայն հնարավորություններ: `object` ը հնարավորություն է տալիս ստեղծել տիպերի ընդանրություններ: Օրինակ, եթե զանգված է հայտարարված, ապա այն կարող է ընդունել միայն նախորոք նշված տիպը: Իսկ եթե կիրառվի `object` տիպը, ապա նրա սահմանման համաձայն այն կարող է ընդգրկել կամ փոխարինել մնացած բոլոր տիպերին: Օրինակում երևում է, որ զանգվածի էլեմենտները արժեքավորվում են տարբեր տիպերով:

```
using System;
class M
{
    static void Main()
    {
        object[] ob = new object[5];
        ob[0] = 88;
        ob[1] = 44.44;
        ob[2] = 'a';
        ob[3] = "Ani";
```

Output:
88
44,44
a
Ani
False

```
        ob[4] = false;
        for (int i = 0; i < 5; i++)
            Console.WriteLine(ob[i]);
    }
}
//.....................................................
class M  //Троелсен-360
{
    static void Main()
    {
        int a = 25;
        object o = 50;
        long a2 = (long)a;
        //long a2 = (long)o; //exeption   ?
       long a3 = (long)(int)o; // ok!!
    }
}
```

.....

84.Դինամիկ տիպեր – dynamic

[Троелсен-648; Нэш-557; Шилдт-703; Нейгел-359; Уотсон-406]

- object, var, dynamic – ոչ ակնհայտ տիպային հայտարարում:
- object և var տիպիզացումը կատարվում է ծրագրի թարգմանման ժամանակ:
- var – փոփոխականի տիպը որշվում է առաջին վերագրման ժամանակ և փոփոխման ենթակա չէ:
- object – փոփոխականի տիպը կարող է փոխվել:
- dynamic - Համարժեք է object տիպին, սակայն տիպայնացումը կատրվում է կատարման ժամանակ և չկա թարգմանման հսկում:
- Ի տարբերություն var – ի դինամիկ տիպը կարող է լինել կլասի դաշտ, ֆունկցիայի պարամետր, վերադարձվող արժեքի տիպ:
- dynamic կարող են լինել նաև լոկալ փոփոխականները:

```
class A
{
        dynamic a = 999;
        dynamic p { get; set; }
        public dynamic f(dynamic k)
        {
        return k;
        }
}
```

- **Սահմանափակումներ**

Դինամիկ տիպը չի կարող ընդունել լամդա արտահայտություն կամ անանիմ մեթոդ: (Կանցնենք ավելի ուշ)

```
dynamic a = GetDynamicObject();
// Error! Methods on dynamic data can't use lambdas!
a.Method(arg => Console.WriteLine(arg));
```

Չի կարող հասկանալ ընդլայնված մեթոդներ և այդ պատճառով LINQ -ում չի կիրառվում: (Կանցնենք ավելի ուշ)

```
dynamic a = GetDynamicObject();
// Error! Dynamic data can't find the Select() extension method!
```

```
var data = from d in a select d;
```

- Դինամիկ տիպի օգտագործումը ստիպում է կատարել ընտրություն տիպի անապահովության և օպտիմալ կոդի միջև: Այն **կիրառելի** է ռեֆլեքսիայում ուշացած կապվածության համար: Կամ ասենք COM գրադարաններից օգտվելու համար (office փաթեթի տվյալներ):
- dotNET – ում օգտագործվում է Dynamic Language Runtime (DLR). DLR -ը նորույթ չէ և օրինակ Smalltalk, LISP, Ruby, Python լեզուները աշխատում են դինամկ միջավայրում:
- Կոդը ստանում է մեծ ճկունություն: Կարելի է հանգիստ կատարել կոդում փոփոխություն չդիմելով տիպերի բարդ վերափոխման մեխանիզմին:
- Փոխհարաբերության պարզեցում տարբեր պլատֆորմների և լեզուների միջև:
- Կատարման ժամանակ հիշողությունում տիպերի անդամների ավելացում կամ փոփոխում:
- Համագործակցում **IronPython** և **IronRuby** լեզուների հետ, որոնք աշխատում են դինամիկ միջավայրում և ունեն NET գրադարաններից օգտվելու հնարավորություն:

```
using System;
class M  //Троелсен-360
{
    static void Main()
    {
        // var v = 3;
        // v = 'a'; //error!!!
        dynamic a = 555; // Run Time error!!
        // object a = 555; // Compiler  error!!
        bool b = true;
        b = a;
        Console.ReadKey();
    }
}
```

.....

Ինտերֆեյս

85.Ինտերֆեյս

[Шилдт-375; Троелсен-320; Либерти-162]

- Ինտերֆեյսը տրամաբանական կառույց է: Նրան օգտագործելու համար պետք է ժառանգել:
- Ինտերֆեյսը որոշում է անդամներ, որոնք պետք է պարտադիր իրականացնեն կլասները կամ ստրուկտուրաները: Ինքը ինտերֆեյսը չի իրականացնում այդ անդամները:
- Ինչպես գիտենք աբստրակտ կլասները ունենալով աբստրակտ մեթոդներ ստիպում են ժառանգման ժամանակ օգտագործող կլասներին անպայման իրականացնել այդ մեթոդները: (աբստրակտ կլասներից չի կարելի օբյեկտ ստանալ): Այս ձևով աբստրակտ կլասը դառնում է ինտերֆեյս մյուս կլասների համար: C# -ում այս միտքը զարգացվել է առանձին ձևով – ինտերֆեյսներով:
- Աբստրակտ կլասները կարող են ունենալ ոչ աբստրակտ անդամներ և նրանց կարելի է իրականացնել ժառանգ կլասների միջոցով:
- Ինտեֆեյսում ոչ մի մեթոդ չի կարող ունենալ որոշում (մարմին) (աբստրակտում – ոչ աբստրակտ մեթոդները կարող են): Այսինքն ինտերֆեյսում նշվում է ինչ ձևով պետք է իրականացվի մեթոդը, բայց կոնկրետ չի նշվում ինչպես: Եթե կլասը օգտագործում է ինտերֆեյս, ապա նա պարտավոր է ինտերֆեյսի բոլոր անդամները որոշել - այսինքն կիրառել:

Գրման ձևը՝

```
public interface  I2
{     bool f();
      int p {get;set;}
      string this [int xxx]{get; set;}
      event ddd eed;
}
```

Օգտագործումը՝

```
class A : I2
{  //.....   }
```

- Ինտերֆեյսը կարող է հայարարել մեթոդ, հատկանիշ, ինդեքսատոր, իրադարձություն:
- Ինտերֆեյսը չի կարող ունենալ տվյալների դաշտ: Չեն կարող որոշել կոնստրուկտոր, դեստրուկտոր: Նրա ոչ մի անդամ չի կարող լինել static: Իմաստ չունի գրել անդամներին թույլատրելիության մոդիֆիկատոր public, որովհետև նա միշտ բաց է:
- Ժառանգման ժամանակ նույնպես անհրաժեշտ է թույլատրել public –ով:

```
interface I2
{
    bool f();
}
class A : I2
{
    public bool f()
    {
        return true;
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        System.Console.WriteLine(ob.f());
    }
}
```

86.Ինտերֆեյսի հղում

[Шилдт-381; Либерти-171]

```
interface  I2
{
   void f();
}
class A: I2
{
   public void f()
        {
             Console.WriteLine("barev !!");
        }
}
```

- Կարելի է ինտերֆեյսի անդամին դիմել հետևյալ ձևով
  A ob = new A(); ob.f();
  Այնպես էլ այս ձևին ալտերնատիվ - կարելի է ստեղծել ինտերֆեյսի հղում, որը ստանում է կլասի օբյեկտ տիպերի վերափոխման օպերացիայով (գիտենք, ինտերֆեյսի օբյեկտ չի կարելի ստեղծել):
  A ob = new A();
  I2 iob = (I2)ob; iob.f();
  կամ մեկ տողով - I2 iob = (I2) new A();
  Այս ձևը դառնում է կլասի անդամներին դիմելու ինտերֆեյսի ցուցիչով ձև:
  iob.f(); և ոչ թե ob.f();
- Այսպիսով կարելի է հայտարարել հղիչային փոփոխական ինտերֆեյսային տիպի: Այդպիսի փոփոխականը կարող է հղել ցանկացած օբյեկտ, որը իրականացնում է ինտերֆեյսը: Երբ ինտերֆեյսը հղիչի միջոցով է կանչում մեթոդը, կատարվում է մեթոդի այն տարբերակը, որը իրականացնում է այդ օբյեկտը: Դա նման է բազային հղիչով ածանցյալ կլասի օբյեկտներին դիմելուն, կամ այլ խոսքով պոլիմորֆ օգտագործում:

```
public interface I2
{
    int f();
}
class A : I2
{
    int a;
    public A()
    {
        a = 0;
    }
    public int f()
    {
        a += 2;
        return a;
    }
}
class B : I2
{
    int b;
    public B()
    {
        b = 5;
    }
    public int f()
```

```
    {
        b += 19;
        return b;
    }
}
class M
{
    public static void Main()
    {
        A ob1 = new A();
        B ob2 = new B();
        I2 iob;
        iob = ob1;
        System.Console.WriteLine("a = " + iob.f());
        iob = ob2;
        System.Console.WriteLine("b = " + iob.f());
    }
}
```

.....

87.Ինտերֆեյսի is և as պատկանելիության ձևեր

[Троелсен-329; Либерти-173]

- Շատ դեպքերում պարզ չի լինում թե արդյոք կլասը անհրաժեշտ ինտերֆեյս ապահովում է, թե ոչ: Պատկանելիությունը ստուգելու համար կա երեք ձև.

✓ առաջինը - արտակարգ իրավիճակի մշակումով try...catch , որը տալիս է InvalidCastException իրավիճակ և որը պետք է մշակել(ավելի ուշ):

✓ երկրորդը – as, այս դեպքում եթե հղումը ինտերֆեյս չի հայտնաբերում, ապա ստանում ենք null հղում:

✓ երրորդը – is - այս դեպքում համեմատվում է օբյեկտի հետ և անհամապատասխանության դեպքում վերադարձնում է false :

```
using System;//interface try..catch
interface I2
{
    byte PP { get; }
}
class A : I2
{
    public byte PP
    {
        get { return 4; }
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        I2 iob;
        try
        {
            iob = (I2)ob;
            Console.WriteLine(iob.PP);
        }
        catch (InvalidCastException e)
        {
            Console.WriteLine(e.Message);
```

```
        }
    }
}
//...............................

interface I2 //interface as
{
    byte PP
    {
        get;
    }
}
class A : I2
{
    public byte PP
    {
        get
        {
            return 44;
        }
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        I2 iob = ob as I2;
        if (iob != null)
            System.Console.WriteLine(iob.PP);
        else
            System.Console.WriteLine("don't");
    }
}
//...............................

interface I2 //interface is
{
    byte PP
    {
        get;
    }
}
class A : I2
{
    public byte PP
    {
        get
        {
            return 44;
        }
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        if (ob is I2)
```

```
            System.Console.WriteLine(ob.PP);
        else
            System.Console.WriteLine("don't");
    }
}
```

.....

88.Ինտերֆեյսը որպես պարամետր և վերադարձվող արժեք

[Троелсен-330]

- Ինտերֆեյսը կարող է օգտագործվել ինչպես մեթոդի պարամետր, այնպես էլ մեթոդի վերադարձվող արժեք:

```
interface I2
{
    void ff();
}
class A : I2
{
    public void ff()
    {
        System.Console.WriteLine("interface - parameter");
    }
}
class M
{
    static void ffM(I2 iob)
    {

        iob.ff();
    }
    static void Main()
    {
        A ob = new A();
        I2 iob2 = ob;
        ffM(iob2);  //  ffc((I2)ob);
    }
}

// Ինտերֆեյսը որպես վերադարձվող արժեք  [Троелсен-332]
interface I2
{
    void ff();
}
class A : I2
{
    public void ff()
    {
        System.Console.WriteLine("interface - return");
    }
}
class M
{
    static I2 ffc(object ob)
    {
        if (ob is I2)
```

```
            return (I2)ob;
        else
            return null;
    }
    static void Main()
    {
        A ob = new A();
        I2 iob = ffc(ob);
        if (iob != null)
            System.Console.WriteLine("return I2");
        else
            System.Console.WriteLine("have not I2");
    }
}
```

.....

89. Ինտերֆեյսների ժառանգականություն, բազմաժառանգում

[Троелсен-337]

- Ինչպես կլասում ստեղծվում է ժառանգականություն, այնպես էլ ինտեֆեյսը կարող են կազմվել ենթակայության սկզբունքով, կազմելով հիերարխիկ ծառ:

```
interface I1
{
    void ff();
}
interface I2 : I1
{
    void ff2();
}
class A : I2
{
    public void ff()
    {
        System.Console.WriteLine("interface 1");
    }
    public void ff2()
    {
        System.Console.WriteLine("interface 2");
    }
}
class M
{
    public static void Main()
    {
        A ob = new A();
        I2 iob2 = (I2)ob;
        iob2.ff();
        iob2.ff2();
    }
}
```

.....

90.Ինտերֆեյսների բացահայտ իրականացում

[Троелсен-338]

- Քանի որ C# - ում կլասին չի թույլատրվում ժառանգել մեկից ավելի կլասներ, օգտագործվում է մեկից ավելի ինտերֆեյսների ժառանգության կիրառությունը:
```
class C : A, I1,I2
{
}
```
- Ինտերֆեյսում նույնպես թույլատրելի է բազմաժառանգականությունը:
```
interface I1 : I2,I3
{
    void ff();
}
```
- Հնարավոր է դեպք, երբ տարբեր ինտերֆեյսների մեթոդներ ունեն նույն անունը և տիպը և բազմաժառանգման դեպքում լինել կոնֆլիկտներ: Կարելի է անդամին դիմել գրելով նաև ինտերֆեյսի անունը: Այս ձևը հնարավորություն է տալիս խուսափել կոնֆլիկտներից: Համնկնող անդամները պետք է լինեն բացահայտ իրականացումով:
```
class A : I2,I1
{
   void I1.ff ()
   {      }
          void I2.ff ()
   {      }
}
```
- Բացահայտ իրականացման դեպքում անդամը միշտ ոչ ակնհայտ փակ է և չի կարող ընդունել մոդիֆիկտորներ abstract, virtual, override, new:
- Բացահայտ իրականացումը օբյեկտով չի կարող աշխատել: Օգտագործման միակ ձևը, դա ինտերֆեյսի ցուցիչով է:
- Եթե անհրաժեշտ է օբյեկտով կանչ, ապա պետք է գրված լինի ոչ բացահայտ իրականացում, որը պարզ է այս դեպքում չի կարող աշխատել ինտերֆեյսի ցուցիչով:

```
interface I1
{
    void ff();
}
interface I2
{
    void ff();
}
class A : I1, I2
{
    void I1.ff()
    {
       System.Console.WriteLine("A.I1.ff called");
    }
    void I2.ff()
    {
      System.Console.WriteLine("A.I2.ff called");
    }
    public void ff()
    {
        System.Console.WriteLine("A ff() called");
    }
}
class M
{
```

```
    static void Main()
    {
        A ob = new A();
        ob.ff();
        I1 iob1 = (I1)ob;
        iob1.ff();
        //((I1)ob).ff(); one line
        ((I2)ob).ff();
    }
}
//-----------------------------------
```

- Կլասի փակ անդամը կարելի է նաև օգտագործել ինտերֆեյսի բացահայտ իրականացումով:

```
interface I
{
    void ff();
}

class A:I
{
    void I.ff()
    {
         System.Console.WriteLine("Welcome");
    }
}
class M
{
    static void Main()
    {
        I i = new A();
        i.ff();
    }
}
```

.....

91.Ինտերֆեյս ICloneable

[Троелсен-346; Шилдт-779]

- Ինչպես նշել ենք Object կլասը ունի մակերեսային կոպիայի (կլոնավորման) հնարավորություն `MemberwiseClone()` մեթոդով: Եթե անհրաժեշտ է խորը կլոնավորում, այսինքն կոմպոզիցան նույնպես կլոնավորել, այլ ոչ թէ հղել, ապա օգտագործվում է ICloneable ինտերֆեյսը: Ժառանգելով այն պարտավորվում ենք իրականացնել `public object Clone()` մեթոդը:
- Առաջին օրինակում մակերեսային է կլոնավորումը, որովհետև կոմպոզիցիայի հղում չկա: Եվ `MemberwiseClone()` մեթոդով և առանց նրա նույն արդյունքն է ստացվում:
- Երկրորդ օրինակում կատարվում է խորը կլոնավորում, `Clone()` մեթոդում գրելով կոմպոզիցիայի անդամների վերագրումը: Այս հնարավորությունը նման է կոպիայի կոնստրուկտորին:

```
// shallow copy // մակերեսային կլոնավորում
using System;
class A : ICloneable
{
    public int x;
    public A(int x)
    {
        this.x = x;
    }
    public override string ToString()
```

```
    {
        return string.Format("X = {0}", x);
    }
    public object Clone()
    {
       // return new A(this.x);
        return this.MemberwiseClone();
    }
}
class M
{
    static void Main()
    {
        A ob1 = new A(50);
        A ob2 = ob1; // Վերագրում
        Console.WriteLine(ob1);
        Console.WriteLine(ob2);
        Console.WriteLine();
        ob2.x = 0;
        Console.WriteLine(ob1);
        Console.WriteLine(ob2);
        A ob3 = new A(100);
        A ob4 = (A)ob3.Clone(); // Կլոնավորում
        Console.WriteLine();
        Console.WriteLine("clone......");
        ob4.x = 999;
        Console.WriteLine(ob3);
        Console.WriteLine(ob4);
    }
}
//.....................................................................................
// Deep Copy // խորը կլոնավորում
using System;
class B
{
    public int x;
}
class A : ICloneable
{
    public string name;
    public B obB = new B();
    public A(int x, string s)
    {
        this.name = s;
        obB.x = x;
    }
    public override string ToString()
    {
        return string.Format("{0}; {1}", name, obB.x);
    }
    public object Clone()
    {
        A temp = (A)this.MemberwiseClone();
        B ob = new B();
        ob.x = this.obB.x;
        temp.obB = ob;
        return temp;
        //return this.MemberwiseClone(); // shallow copy
    }
```

```
}
class M
{
    static void Main()
    {
        A ob1 = new A(100, "ani");
        A ob2 = (A)ob1.Clone();
        Console.WriteLine(ob1);
        Console.WriteLine("clone.....");
        Console.WriteLine(ob2);
        Console.WriteLine();
        Console.WriteLine("chenged....");
        ob2.name = "nune";
        ob2.obB.x = 999;
        Console.WriteLine(ob1);
        Console.WriteLine("clone.....");
        Console.WriteLine(ob2);
    }
}
```

.....

92.Ինտերֆեյս IEnumerable

[Троелсен-340; Шилдт - 1001]

- foreach հրամանը կարողանում է ցանկացած զանգվածի անդամներ կարդալ ըստ հերթականության, որովհետև Array կլասը իրականացնում է IEnumerable ինտերֆեյսը:
- Ցանկացած տիպ կարող է ունենալ նույն պահվածքը, եթե այն իրականացնում է GetEnumerator() մեթոդը: Այն ֆորմալ ձևով հայտարարված է IEnumerable ինտերֆեյսում, որը գտնվում է System.Collections անունի տարածքում և վերադարձնում է IEnumerator ինտերֆեյս:

```
    public interface IEnumerable
    {
        IEnumerator GetEnumerator();
    }
    public interface IEnumerator
    {
        bool MoveNext(); // Կուրսորի տեղաշարժ
        object Current { get; } // Ընթացիք էլեմենտի ստացում
        void Reset(); // Տեղաշաժվող կուրսորի տեղաշարժի “reset”
    }
//.........................................
using System;
using System.Collections;
class Car
{
    public string name;
    public Car(string n)
    {
        name = n;
    }
}
class Garage
{
    Car[] c = new Car[4];
    public Garage()
    {
        c[0] = new Car("toyota");
        c[1] = new Car("mercedes");
        c[2] = new Car("nissan");
```

```
        c[3] = new Car("lexus");
    }
}
class Program
{

    static void Main(string[] args)
    {
        Garage g = new Garage();
        foreach(Car c in g) //error!!
        {
            Console.WriteLine(c.name);
        }

    }
}
//.........................................
using System;
using System.Collections;
class Car
{
    public string name;
    public Car(string n)
    {
        name = n;
    }
}
class Garage:IEnumerable
{
    Car[] c = new Car[4];
    public Garage()
    {
        c[0] = new Car("toyota");
        c[1] = new Car("mercedes");
        c[2] = new Car("nissan");
        c[3] = new Car("lexus");
    }
    public IEnumerator GetEnumerator()
    {
        return c.GetEnumerator();
    }
    // Եթե անհրաժեշտ է թաքցնել Ienumerable ինտերֆեյի հնարավորությունները օբյեկտի
    // մակարդակով, կարելի է օգտվել ինտերֆեյսի բացահայտ իրականացումից
    //IEnumerator IEnumerable.GetEnumerator()
    //{
    //    return c.GetEnumerator();
    //}
}
class Program
{
    static void Main(string[] args)
    {
        Garage g = new Garage();
        foreach(Car c in g)
        {
            Console.WriteLine(c.name);
        }
    }
}
```

93. foreach և IEnumerable

[Нейгел-197; Уотсон-295]

- foreach - ի մասին: Թարգմանության ժամանակ այն չի վերածվում միջանկյալ լեզվի օպերատորի: Այն վերածվում է IEnumerable ինտերֆեյսի մեթոդների և հատկանիշների:

```
foreach (var p in persons)
{
    Console.WriteLine(p);
}
```

Վերածվում է

```
IEnumerator<Person> enumerator = persons.GetEnumerator();
while (enumerator.MoveNext())
{
    Person p = (Person)enumerator.Current;
    Console.WriteLine(p);
}
```

- Սկզբից կանչվում է GetEnumerator() մեթոդը զանգվածի հաշվիչ ստանալու համար: while ցիկլը կատարվաում է այնքան ժամանակ, քանի դեռ MoveNext() մեթոդը վերադարձնում է true: Զանգվածի էլեմենտները հասանելի են դառնում Current հատկանիշի միջոցով:

……

94.Իտերատորներ, Օպերատոր yield

[Шилдт-1003; Троелсен-343; Нейгел-197; Уотсон-295]

- Իտերատորը, դա կոդերի բլոկ է, որը հնարավորություն է տալիս ամբողջ տվյալները կարդալ ըստ հերթականության:
- Երբեմն անհրաժեշտ է լինում foreach - ի միջոցով աշխատացնել սեփական կլասները որպես կոլեկցիաներ (համարակալվող համակարգեր) չօգտվելով IEnumerable ինտերֆեյսի ժառանգումից: Դրա համա կարելի է օգտվել իտերատորներից, որը կազմակերպվում է yield return հրամանի միջոցով:
- Իտերատորը հնարավորություն է տալիս ընդհատել իտերատրի ցիկլը yield break հրամանով:

```
using System;
using System.Collections;
class Car
{
    public string name;
    public Car(string n)
    {
        name = n;
    }
}
class Garage
{
    Car[] c = new Car[4];
    public Garage()
    {
        c[0] = new Car("toyota");
        c[1] = new Car("mercedes");
        c[2] = new Car("nissan");
        c[3] = new Car("lexus");
    }
    public IEnumerator GetEnumerator()
    {
        yield return c[0];//
        yield return c[1];//
```

```
        yield return c[2];//
        yield return c[3];//
        //foreach (Car x in c) //կամ
        //    yield return x;
    }
}
class Program
{
    static void Main(string[] args)
    {
        Garage g = new Garage();
        foreach (Car c in g)
        {
            Console.WriteLine(c.name);
        }
    }
}
//-----------------
```

- Իտերատորում պարտադիր չէ կուտակիչի առկայություն և կարելի է ընդհատել ցիկլը `yield break` հրամանով;

```
using System;//Шилдт-1004;
using System.Collections;
class A
{
    char ch = 'A';
    public IEnumerator GetEnumerator()
    {
        for (int i = 0; i < 26; i++)
        {
            if (i == 10)
            yield break;
            yield return (char)(ch + i);
        }
    }
}
class M
{
    static void Main()
    {
        A ob = new A();
        foreach (char ch in ob)
            Console.Write(ch + " ");
        Console.WriteLine();
    }
}
```

.....

95.Անվանական իտերատորներ

[Троелсен-344; Нейгел-199]

- Իտերատորը օգտագործվում է ցանկացած մեթոդում: Այդ դեպքում մեթոդը կոչվում է **անվանական իտերատոր** և թույլատրվում է օգտագործել for() ցիկլ:

```
using System;
using System.Collections;
class A
{
    string[] names = { "aaa", "bbb", "ddd", "ccc" };
    public IEnumerator GetEnumerator()
    {
```

```
        for (int i = 0; i < 4; i++)
        {
            yield return names[i];
        }
    }
    public IEnumerable Reverse()
    {
        for (int i = 3; i >= 0; i--)
        {
            yield return names[i];
        }
    }
}
class Program
{
    static void Main(string[] args)
    {
        var titles = new A();
        foreach (var t in titles)
        {
            Console.WriteLine(t);
        }
        Console.WriteLine();
        Console.WriteLine("revers........");
        foreach (var t in titles.Reverse())
        {
            Console.WriteLine(t);
        }
    }
}
//----------------------------
```

- Անվանական իտերատրի առավելություններից է պարամետրի փոխանցումն հնարավորությունը:

```
using System; //Шилдт-1007;
using System.Collections;
class A
{
    char ch = 'A';
    public IEnumerable ff(int end)
    {
        for (int i = 0; i < end; i++)
            yield return (char)(ch + i);
    }
    public IEnumerable ff(int begin, int end)
    {
        for (int i = begin; i < end; i++)
            yield return (char)(ch + i);
    }
}
class M
{    static void Main()
    {
        A ob = new A();
        foreach (char ch in ob.ff(7))
            Console.Write(ch + " ");
        Console.WriteLine();
        foreach (char ch in ob.ff(5, 12))
            Console.Write(ch + " ");
    }
}
```

96.Ինտերֆեյս ICamparable

[Троелсен-350; Шилдт-778]

- Հնարավորություն է տալիս համեմատել երկու օբյեկտներ:

```
interface IComparable
{
    int CompareTo(object o);
}
```

- Ներկայացված կառուցվածքի մեջ `int CompareTo(object o)` մեթոդ կանչող օբյեկտը համեմատվում է այն օբյեկտի հետ, որը որոշվում է պարամետրով: Եթե կանչող օբյեկտի պարունակությունը մեծ է պարամետում նշվածից, ապա վերադարձվում է “1” արժեք: Եթե կանչող օբյեկտի պարունակությունը փոքր է պարամետում նշվածից, ապա վերադարձվում է “-1” արժեք: Եթե երկու օբյեկտների պարունակությունը նույնն է, ապա վերադարձվում է “0” արժեք:
- `IComparable` ինտերֆեյսի օգտագործման տեղերից մեկը `System.Array` կլասն է: `Array` կլասը ունի Sort() մեթոդը, որը կարող է վերադասավորել ներդրված տիպերով անդամները(int, string և այլն): Այդ անդամները կարելի է վերադասավորել թվային կամ սիմվոլային հաջորդականությամբ, քանի որ նշված տիպրի համար իրականացված է `IComparable` ինտերֆեյսը:
- Վերցնենք հետևյալ կլասը.

```
class Car
{
    public string CarName;
    public int CarID { get; set; }
}
```

- Ինչ կարող է լինել, եթե օրինակ օգտագործենք ասենք `Car` տիպի զանգված և կիրառենք `Array.Sort(obCar)` հրամանը:
- Կառաջանա արտակարգ իրավիճակ, քանի որ `Car` տիպի համար `IComparable` ինտերֆեյս իրականացված չէ: Որպեսզի հնարավոր լինի կիրառել `Array.Sort(obCar)` հրամանը, ապա անհրաժեշտ է իրականացնել `IComparable` ինտերֆեյսի `int CompareTo(object o)` մեթոդը: Անհրաժեշտ է ընտրել նաև `Car` կլասի ըստ որ անդամի է կատարվում վերադասավորման կարգավորումը: Օրինակ կարելի է վերցնել ըստ `CarID` դաշտի:

```
using System;
class Car : IComparable
{
    public string CarName;
    public int CarID { get; set; }
    public Car(string name,int id)
    {
        CarName = name;
        CarID = id;
    }
    public int CompareTo(object o)
    {
        Car temp = o as Car;
        if (temp != null)
        {
            if (this.CarID > temp.CarID)
                return 1;
            if (this.CarID < temp.CarID)
                return -1;
            else
                return 0;
        }
        else
```

```
            throw new ArgumentException("Parameter is not a Car!");
        // ............. 2nd way
        //Car temp = o as Car;
        //if (temp != null)
        //{     return CarID.CompareTo(temp.CarID); }
        //else    throw new ArgumentException("Parameter is not a Car!");
    }
}

class Program
{
    static void Main(string[] args)
    {
        Car[] ob=new Car[5];
        ob[0] = new Car("Toyota",55);
        ob[1] = new Car("Mersedes", 33);
        ob[2] = new Car("Lexuz", 9);
        ob[3] = new Car("BMW",  44);
        ob[4] = new Car("Nissan", 5);
        foreach(Car c in ob)
            Console.WriteLine(c.CarID+ "\t"+ c.CarName);
        Array.Sort(ob);
        Console.WriteLine();
        foreach (Car c in ob)
            Console.WriteLine(c.CarID + "\t" + c.CarName);
    }
}
```

.....

Դելեգատներ, Իրադարձություն

97.Դելեգատներ

[Шилдт-473; Троелсен-386; Нейгел-241]

- Դելեգատը օբյեկտ է, որը կարող է հղել մեթոդին: Ստեղծելով դելեգատի օբյեկտ կարելի է նրա միջոցով կանչել մեթոդ: Նա նման է C++ -ի ցուցիչի, որը դիմում է ֆունկցիային: Սակայն դելեգատը համարվում է ղեկավարվող օբյեկտ, և դա նշանակում է կատարման ժամանակ օգտվել որոշ առավելություններից (չթույլատվող հասցեներից խուսափում, հիշողության խախտման խուսափում…):
- Ինչպես գիտենք կարելի է հղել միայն օբյեկտին: Ֆունկցիան օբյեկտ չէ, սակայն նա նույնպես գտնվում է ֆիզիկական հասցեում և այդ հասցեով է կանչվում ֆունկցիան: Ահա այդ հասցեն վերագրվում է դելեգատին:
- Որտեղ են օգտագործվում դելեգատները:
- ✓ առաջինը - դելեգատները ապահովում են իրադարձությունների աշխատանքը:
- ✓ երկրորդ - նրանք թույլ են տալիս ծրագրի օգտագործման ժամանակ կատարել մեթոդ, որը հայտնի չէ թարգմանման ժամանակ:
- ✓ երրորդ - (callback) - հետադարձ կանչի ֆունկցիա, որի աշխատանքը հիմնված է ֆունկցիայի ցուցիչի վրա: Երբ ֆունկցիան վերջացնում է իր աշխատանքը, նա տեղեկացնում է դրա մասին: Նախկինում նրանք օգտագործվում էին Windows API ֆունկցիաներից օգտվելու համար:
- Օրինակ `public delegate int ddd(int x, int y);` Այս հայտարարումը թարգմանիչը վերածում է հետևյալ կլասի:

```
sealed class ddd: System.MulticastDelegate
{
      public ddd(uint functionAddress);
      public int Invoke(int x, int y);
      public IAsyncResult BeginInvoke(int x, int y,AsyncCallback cb, object state);
      public int EndInvoke(IAsyncResult result);
}
```

- Invoke() մեթոդը աշխատում է սինխրոն և դա նշանակում է քանի դեռ չի ավարտվել կանչված մեթոդը ծրագրի տվյալ հատվածը չի կարող շարունակվել և զարմանալի է, սակայն անմիջական հնարավոր չէ կանչել Invoke() մեթոդը:
- BeginInvoke() and EndInvoke() մեթոդները ապահովում է ասինխրոն աշխատանք, որը նշանակում է ծրագիրը կարող է շարունակվել երկրորդական հոսքում զուգահեռ ձևով: Նույն արդյունքին կարելի է հասնել հոսքերի միջոցով, սակայն նշված ձևը օգտագործման տեսակետից ավելի պարզ է:

Դելեգատների օգտագործումը [Шилдт-474; Либерти-301; Нейгел-243]

- Դելեգատի հայտարարման ժամանակ որոշվում է ֆունկցիայի վերադարձնող արժեքը, պարամետրերի տեսակները և քանակը:
  `delegate int  ddd(int a1, int a2);`
- Այնուհետև պետք է հայտարարել դելեգատի օբյեկտ: Որից հետո ստեղծում ենք մեթոդ, որը մշակում է դելեգատը: Նա պետք է ունենա նույն սիգնատուրան (վերադարձվող արժեք, պարամետրերի տեսակ և քանակի նույնություն), որը ունի դելեգատը:

```
class A
{
    ddd ddobject; // հայտարարել դելեգատի օբյեկտ
    public int ff(int a1, int a2) // մշակում է դելեգատը
    {
        return a1 + a2;
    }
}
```

- Պետք է միացնել մշակողը դելեգատին: Ասենք Main() ծրագրում:

```
   A ob = new A();
   ob. ddobject =  new ddd (ob.ff);
```

- մեթոդի կանչելը դելեգատի միջոցով:

```
   int result = ddobject (5,9);  // result = 14
using System;
delegate string ddd(string ss);
class M
{
    static string replaceSpaces(string a)
    {
        Console.WriteLine("Replacement");
        return a.Replace(' ', '-');
    }
    static string removeSpaces(string a)
    {
        string temp = "";
        Console.WriteLine("removal");
        for (int i = 0; i < a.Length; i++)
            if (a[i] != ' ')
                temp += a[i];
        return temp;
    }
    static string reverse(string a)
    {
        string temp = "";
        Console.WriteLine("revers");
        for (int i = a.Length - 1; i >= 0; i--)
            temp += a[i];
        return temp;
    }
    public static void Main()
    {
        ddd ddob = new ddd(replaceSpaces);
        Console.WriteLine(ddob("It is the simple test"));
        ddob = new ddd(removeSpaces);
        Console.WriteLine(ddob("It is the simple test "));
        ddob = new ddd(reverse);
        Console.WriteLine(ddob("It is the simple test "));
    }
}
```

```
Output
replacement
it-is-the-simple-test
removal
itisthesimpletest
revers
tset elpmis eht si ti
```

.....

98.Խմբային դելեգատներ

[Шилдт-476; Троелсен-396; Нейгел-251; Либерти-316]

- Խմբային դելեգատներ - դա միայնակ դելեգատ է , որը փոխարինում է երկու և ավելի դելեգատների: Նա ձևավորվում է + = նշանով մեկ դելեգատը միացնելով մյուսին: (- =) – ով կարելի է հանել խմբից: Կանչելով դելեգատը կատարվում է դելեգատների խումբը:
- Նրանք պետք է ունենան հիմնականում void վերադարձվող արժեք: (Либерти,317) – կարելի է առանց void – ի, այդ դեպքում վերադարձնում է վերջին դելեգատի վերադարձած արժեքը:
- Կանչելու հերթականությունը պաշտպանվում են նույն ձևով, ինչ ձևով ձևավորվել է դելեգատների խումբը:

```
using System;
public delegate void ddd(int a1, int a2);
class M
{
    ddd ddd1; ddd ddd2; ddd ddd3;
```

```
    public void ffAdd(int a1, int a2)
    {
        Console.WriteLine("add " + (a1 + a2));
    }
    public void ffSub(int a1, int a2)
    {
        Console.WriteLine("sub " + (a1 - a2));
    }
    public void ffMul(int a1, int a2)
    {
        Console.WriteLine("mul " + (a1 * a2));
    }
    public static void Main()
    {
        M ob = new M();
        ob.ddd1 = new ddd(ob.ffAdd);
        ob.ddd2 = new ddd(ob.ffSub);
        ob.ddd3 = new ddd(ob.ffMul);
        ddd Multiddd = ob.ddd1;
        Multiddd += ob.ddd2;
        Multiddd += ob.ddd3;
        Multiddd(7, 5);
        Multiddd -= ob.ddd3;
        Multiddd(4, 4);
    }
}
```

……

99.Delegate և MulticastDelegate բազային կլասներ

[Троелсен-389; Нэш - 288]

- System.MulticastDelegate կլասի որոշ անդամները

```
public abstract class MulticastDelegate : Delegate
{
      // Returns the list of methods "pointed to."
      public sealed override Delegate[] GetInvocationList();
      // Overloaded operators.
      public static bool operator ==(MulticastDelegate d1, MulticastDelegate d2);
      public static bool operator !=(MulticastDelegate d1, MulticastDelegate d2);
}
```

- System.Delegate կլասի որոշ անդամները

```
public abstract class Delegate : ICloneable, ISerializable
{
    // Methods to interact with the list of functions.
    public static Delegate Combine(params Delegate[] delegates);
    public static Delegate Combine(Delegate a, Delegate b);
    public static Delegate Remove(Delegate source, Delegate value);
    // Properties that expose the delegate target.
    public MethodInfo Method { get; }
    public object Target { get; }
}
```

✓ Method – հատկանիշ, վերադարձնում է մեթոդի անունը, որին նշում է դելեգատը:

✓ Target - հատկանիշ, եթե վերադարձնում է null, ապա դելեգատը նշում է ստատիկ մեթոդ: Եթե դելեգատը նշում է սովորական մեթոդ, ապա հատկանիշը վերադարձնում է կլասի անունը:

- ✓ Combine() – static մեթոդը օգտագործվում է այնպիսի դելեգատի ստեղծման համար, որը նշում է մի քանի ֆունկցիաներ: Օգտագործվում է ավելացման համար:
- ✓ Remove() - static մեթոդը ջնջում է դելեգատը:
- ✓ GetInvocationList() – վերադարձնում է ֆունկցիաների ցուցակի զանգված: Հնարավորություն է տրվում խմբային դելեգատի հերթի փոփոխություն:

```
using System;
public delegate void ddd(int a1, int a2);
class M
{
    static ddd dd1;
    ddd dd2;
    static ddd dd3;
    public static  void ffAdd(int a1, int a2)
    {
        Console.WriteLine("add " + (a1 + a2));
    }
    public   void ffSub(int a1, int a2)
    {
        Console.WriteLine("sub " + (a1 - a2));
    }
    public static  void ffMul(int a1, int a2)
    {
        Console.WriteLine("mul " + (a1 * a2));
    }
    public static void Main()
    {
        M ob = new M();
        dd1 = new ddd(ffAdd);
        ob.dd2 = new ddd(ob.ffSub);
        dd3 = new ddd(ffMul);
        // Method
        Console.WriteLine(dd1.Method);
        Console.WriteLine(ob.dd2.Method);
        Console.WriteLine(dd3.Method);
        // Target
        if(dd1.Target==null)
            Console.WriteLine("static");
        if (ob.dd2.Target != null)
        {
            Console.WriteLine("no static");
            Console.WriteLine(ob.dd2.Target);
        }
        // Multidelegate
        ddd Multidd = dd1;
        Multidd += ob.dd2;
        Multidd += dd3;
        Console.WriteLine("Multidelegate");
        Multidd(7, 5);
        // Combine
        Console.WriteLine("Combining dd2");
        ddd combinedDelegate = (ddd)Delegate.Combine(Multidd, ob.dd2);
        combinedDelegate(7, 5);
        // Remove
        Console.WriteLine("Removing dd1");
        ddd remDelegate = (ddd)Delegate.Remove(Multidd,dd1);
        remDelegate(7, 5);
        // GetInvocationList()
        Console.WriteLine("Calling diferend delegate from Multidd");
```

```
        Delegate[] delegateList = Multidd.GetInvocationList();
        ((ddd)delegateList[2])(7, 5);
        ((ddd)delegateList[0])(7, 5);
        ((ddd)delegateList[1])(7, 5);
        Console.WriteLine("Calling delegateList with foreach");
        foreach (var del in delegateList)
        {
            ((ddd)del)(7, 5);
        }
    }
}
```

.....

100.Անանուն մեթոդներ

[Нейгел-254; Нэш-295; Троелсен-411; Шилдт-484]

- Ինչպես գիտենք դելեգատ օգտագործելու համար անհրաժեշտ է համապատասխանող ֆունկցիա: Սակայն կարելի է օգտագործել դելեգատ, որի կատարման ֆունկցիան գոյություն չունի առանձին, այլ անմիջապես և առանց անունի հաջորդում է դելեգատին: Դրանց անվանում են անանուն մեթոդներ:
- Անանուն մեթոդների օգտակարությունը կայանում է նրանում, որ կոդը լինում է ավելի պարզ և քիչ հրամաններ է գրվում: Բայց միևնույնն է , թարգմանության ժամանակ տեղադրվում է մեթոդի անուն և արագությանը չի ավելանում: Առավելությունը զգացվում է, երբ օգտագործվում է իրադարձություններում, առավելապես երբ երկու իրադարձություն օգտվում են միևնույն դելեգատից:
- Нейгел-255 **սխալ** –(Չի թույլատրվում օգտագործել անանուն մեթոդներում **goto**, **break, continue)** : Ոչ, թույլատրվում է:
- **unsafe** կոդ նույնպես չի թույլատրվում օգտագործել անանուն մեթոդներում:
- Չի թույլատրվում նաև **ref , out** պարամետրերը:
- Ցանկալի չէ, եթե անհրաժեշտ է միևնույն աննուն մեթոդը գրել և օգտագործել մեկից ավելի:

```
using System;
delegate string ddd(string ss);
class M
{
    public static void Main()
    {
        string s = "It is the simple test";
        Console.WriteLine(s);
        ddd ddob = delegate(string ss)
        {
            Console.WriteLine("Replacement of blanks with hyphens");
            return ss.Replace(' ', '-');
        };
        Console.WriteLine(ddob(s));
        ddob = delegate(string ss)
        {
            string temp = "";
            Console.WriteLine("revers string");
            for (int i = ss.Length - 1; i >= 0; i--)
                temp += ss[i];
            return temp;
        };
        Console.WriteLine(ddob(s));
    }
}
```

101.Lambda արտահայտություն

[Шилдт-488; Нэш-503; Нейгел-255; Троелсен-413; Скит Jon-246]

- Լյամդա արտահայտությունները ֆունկցիոնալ ծրագրավորման համակարգ է: Այն փոխարինում են անանուն ֆունկցիաներին լինելով ավելի նոր արտահայտման ձև: Օգտագործվում է դելեգատների, իրադարձությունների և LINQ երի հետ:
- Օգտագործվում է => լյամբդա օպերատորը, որը արտահայտությունը բաժանում է երկու մասի: Ձախ մասում պարամետրերն են, իսկ աջ մասում արտահայտության մարմինը:

✓ c=>c+2 ավելացնում ենք 2 ով

✓ n=>n%2==0 վերադարձնում է տրամաբանական արժեք

- Սկզբից հայտարարվում է դելեգատ, որը համատեղելի է լյամդա արտահայտությանը (դելեգատը պետք է համապատասխանի լյամդիայի պարամետրերի և վերադարձվող արժեքի տիպերին), հետո դելեգատի օբյեկտ, որին վերագրվում է լյամդա արտահայտությունը:
- Եթե պարամետրերի քանակը մեկից ավել է, ապա նրանք ընդգրկվում են կլոր փակագծերի մեջ:
- Լյամդայի մարմինը ընդգրկում են ձևավոր փակագծի մեջ, եթե մեկից ավելի օպերատորներ են մասնակցում:

```
using System; //delegate
delegate int ddd(string s);
class M
{
    static int f(string s)
    {
        return s.Length;
    }
    static void Main(string[] args)
    {
        ddd ob = new ddd(f);
        Console.WriteLine(ob("albert"));
    }
}
//.................................................................

using System; // anonimus
delegate int ddd(string s);
class M
{
    static void Main(string[] args)
    {
        ddd ob = delegate(string c)
        {
            return c.Length;
        };
        Console.WriteLine(ob("albert"));
    }
}
//.................................................................

using System; //lambda
delegate int ddd(string s);
class M
{
    static void Main(string[] args)
    {
        ddd ob = t => t.Length;
        Console.WriteLine(ob("albert"));
    }
}
```

```
//.............................................................
using System; //Шилдт
delegate int Incr(int v);
delegate bool IsEven(int v);
class M
{
    static void Main()
    {
        Incr incr = count => count + 2;
        Console.WriteLine("Use incr lambda expression: ");
        int x = -10;
        while (x <= 0)
        {
            Console.Write(x + " ");
            x = incr(x);
        }
        Console.WriteLine("\n");
        IsEven isEven = n => n % 2 == 0;
        Console.WriteLine("Use isEven lambda expression: ");
        for (int i = 1; i <= 10; i++)
            if (isEven(i)) Console.WriteLine(i + " is even.");
    }
}
//.............................................................

using System; //Шилдт
delegate int IntOp(int end);
class M
{
    static void Main()
    {
        IntOp fact = n =>
        {
            int r = 1;
            for (int i = 1; i <= n; i++)
                r = i * r;
            return r;
        };
        Console.WriteLine("The factorial of 5 is " + fact(5));
    }
}
//.............................................................

using System; //replaceSpaces with Lambda //Шилдт 493
delegate string ddd(string ss);
class M
{
    public static void Main()
    {
        ddd replaceSpaces = ss =>
        {
            return ss.Replace(' ', '-');
        };
        string str = replaceSpaces("It is the simple test");
        Console.WriteLine(str);
    }
}
.............................................................................................................
```

102.Covariance և Contravariance դելեգատներ

[Шилдт-481rus418eng]

- Մեթոդը, որը տրվում է դելեգատին պետք է համնկնի և վերադարձնող արժեքով և սիգնատուրայով: Սակայն այս օրենքը այդքան էլ խիստ չէ, եթե կա ածանցիալ կլասների (այսինքն ժառանգականությամբ կապված) հարաբերություն և աշխատում է "կովարյանտություն" և "կոնտրավարյանտություն" հարաբերությունը:
- **Covariance** – հնարավորություն է տալիս դելեգատը աշխատի մեթոդի հետ, որի վերադարձնող արժեքի տիպը լինի դելեգատի տիպին ածանցիալ կլասի տիպ:
- **Contravariance** - հնարավորություն է տալիս դելեգատը աշխատի մեթոդի հետ, որի պարամետրը լինի բազային տիպ դելեգատի պարամետրի տիպի նկատմամբ:

```
using System;
class X
{
    public int n;
}

class Y : X
{
}

delegate X ddd(Y ob);

class CoCont
{
    public static Y ff(X ob) // Y - Covariance // X - Contravariance
    {
        Y temp = new Y();
        temp.n = ob.n + 1;
        return temp;
    }

class Program
{
    static void Main(string[] args)
    {
        X Xob = new X();
        Y Yob = new Y();
        ddd obddd = CoCont.ff;
        Xob = obddd(Yob);
        Console.WriteLine(Xob.n);
    }
}
//------------------------------

using System;// Covariance
class X
{
    public int n;
}
class Y : X
{
}
delegate X ddd(X ob);
class CoCont
{
    public static Y ff(X ob)
```

```
    {
        Y temp = new Y();
        temp.n = ob.n + 1;
        return temp;
    }
}
class Program
{
    static void Main(string[] args)
    {
        X Xob = new X();
        Y Yob = new Y();
        ddd obddd = CoCont.ff;
        Xob = obddd(Yob);
        Console.WriteLine(Xob.n);
    }
}
//--------------------------------

using System;// Contravariance
class X
{
    public int n;
}
class Y : X
{
}
delegate Y ddd(Y ob);
class CoCont
{
    public static Y ff(X ob)
    {
        Y temp = new Y();
        temp.n = ob.n + 1;
        return temp;
    }
}
class Program
{
    static void Main(string[] args)
    {
        X Xob = new X();
        Y Yob = new Y();
        ddd obddd = CoCont.ff;
        Xob = obddd(Yob);
        Console.WriteLine(Xob.n);
    }
}
```

.....

103.Իրադարձություն - event

[Троелсен-404; Шилдт-494]

- DOS և Windows օպերացիոն համակարգերը ունեն աշխատանքային հիմնարար տարբերություն, երբ տեղի է ունենում հայցեր արտաքին ազդակներից: DOS–ը ժամանակ առ ժամանակ հետևում է սարքերին, որոնց հետ ինֆորմացիա պետք է փոխանակի - այն կոչվում է հաջորդական հայցով (**sreial polling**): Windows–ում կազմակերպվում է իրադարձությունների ձևով – (**event driven**): Դա միջոց է , երբ Windows–ը տալիս է նշան, որ ինչ որ բան է տեղի ունեցել (օրինակ մկնիկից քլիք է կատարվել) և միայն այդ դեպքում է կատարվում համապատասխան աշխատանքներ: (Petzold,61rus):
- event, դա ավտոմատ տեղեկացում է որոշ գործողության կատարման մասին: Ում որ պետք է դա նա իր մոտ գրանցում է իրադարձության մշակող – event handler: Երբ իրադարձություն է տեղի ունենում, կանչվում են բոլոր գրանցված իրադարցության մշակողները :
- C# -ում ցանկացած օբյեկտ կարող է հրատարակել իրադարձություններ, իսկ այլ կլասներ - գրանցվեն նրան: Երբ հրատարակող կլասը իրադարձություն է իրականացնում, բոլոր գրանցված կլասները իմանում են դրա մասին: Հրատարակիչը որոշում է դելեգատ, որի միջոցով է իրագործվում գրանցված կլասների մշակումը:
- Օրինակ – ստեղը իրենից ներկայացնում է հրատարակիչ, որը հրատարակում է քլիք իրադարձություն: Մյուս կլասները կարող են դառնալ գրանցվողներ մաուս քլիքը օգտագործելու համար:
- Իրադարձություն հայտարարվում է որևէ կլասում.
  `public event ddd eee;` // որտեղ ddd - դելեգատի անունը է:
- Պետք է հայտարարված լինի նաև դելեգատ, ասենք կլասից դուրս
  `delegate void ddd();`
  Նրանք void են վերադարձնում, որովհետև հիմնականում օգտագործվում են խմբակային ձևով (+=):
- Իրադարձության մշակող (այլ կլասում), որը կատարում է իրադարձության իրականացումը:
  `static void Onff() { }`
- Ռեգիստրացիա - իրադարձության մշակողը ռեգիստրացիա է լինում(օրինակ Main() – ում):
  `ob.eee += new ddd(Onff);`
- Իրադարձության գեներացիա օրինակ մկնիկի քլիք, ժամանակի ավարտ և այլն (օրինակում արհեստական է): Որպես հետևանք կանչվում է բոլոր գրանցված իրադարձությունները: (Մեր օրինակում մեկ հատ է):

```
using System;
delegate void ddd();

class A
{
    public event ddd eee;
    public void ff()
    {
        if (eee != null)
            eee();
    }
}

class M
{
    static void Onff()
    {
        Console.WriteLine("iradartsutiun");
    }
    public static void Main()
```

```
    {
        A ob = new A();
        ob.eee += new ddd(Onff);
        ob.ff();
    }
}
```

.....

104.FCL գրադարանի իրադարձություններ, EventHandler - դելեգատ

[Либерти-321; Шилдт-506, 508]

- Ընդունված կարգ է .NET FRAMEWORK գրադարանում իրադարձության մշակողները վերադաձնում են void և ընդունում են երկու պարամետրեր: Առաջին պարամետրը հղում է իրադարձության աղբյուրի օբյեկտին - այսինքն հրատարակչին: Երկրորդը – պարամետր է , որը հանդիսանում է EventArgs կլասի անդամ կամ նրան ժառանգած անդամ, պարունակում է մնացած տվյալները , որը նախատեսված է մշակողի համար:
- Խորհուրդ է տրվում բոլոր ծրագրավորողներին այս կանոնը ընդունել:

```
using System;
class A : EventArgs
{
    public int c;
}
delegate void ddd(object sender, A e);

class B
{
    public event ddd eee;
    public void ff()
    {
        A a = new A();
        if (eee != null)
        {
            a.c++;
            eee(this, a);
        }
    }
}

class X
{
    public void Onff(object sender, A e)
    {
        Console.WriteLine(e.c + " object X");
        Console.WriteLine(" class " + sender);
    }
}

class Y
{
    public void Onff(object sender, A e)
    {
        Console.WriteLine(e.c + " object Y");
        Console.WriteLine(" class " + sender);
        Console.WriteLine();
    }
}
```

```
class M
{
    public static void Main()
    {
        X ob1 = new X();
        Y ob2 = new Y();
        B ob = new B();
        ob.eee += new ddd(ob1.Onff);
        ob.eee += new ddd(ob2.Onff);
        ob.ff();
    }
}
//.....................................
```

EventHandler – դելեգատ [Шилдт-508]

- FCL գրադարանը ունի ներդրված `EventHandler` `delegate`, որը պարզեցնում է աշխատանքը: Նրան կարելի է օգտագործել իրադարձության մշակման կանչի համար, որը չի պահանջում ժառանգում EventArgs կլասից:

```
using System;
class A
{
    public event EventHandler eee;
    public void ff()
    {
        if (eee != null)
            eee(this, EventArgs.Empty);
    }
}

class M
{
    static void Onff(object sender, EventArgs e)
    {
        Console.WriteLine("iradartsutin " + sender);
    }
    public static void Main()
    {
        A ob = new A();
        ob.eee += Onff;
        ob.ff();
    }
}
```

output
iradartsutiun A

.....

Բացառություն (Exception)

105.Բացառիկ իրավիճակներ

[Шилдт-403; Троелсен-265; Нейгел-419]

- Եթե ծրագրում կարող է տեղի ունենալ չնախատեսված անցումներ, ապա նախատեսվում է մի միջոց, որը կրող է ծրագրի հատվածը դնել հսկման տակ: Այդ միջոցը կոչվում է բացառիկ իրավիճակներ:
- Բացառիկ իրավիճակները առաջանում են սխալների տեսակներից: Սխալի տեսակներ են համարվում.

✓ Ծրագրային սխալներ – օրինակ ստեկի տարածքի լրացում, կամ զանգվածի սահմանի խախտում, զրոյի բաժանում:

✓ Օգտագործողի սխալներ - թվի փոխարեն տեքստի մուտք…

✓ Արտակարգ իրավիճակներ – չնախատեսված տարբեր վիճակներ,

✓ կապի խզում, չթույլատրված ֆայլից օգտվել …

- Օգտագործվող հանգույցային բառերն են.
  `try{...}` `catch{…}` `finally{…}` `throw`
- Թվային ինֆորմացիայի փոխարեն սխալի զդուշացումը կատարվում է տեքստային հայտարարությունների ձևով:
- Բոլոր բացառիկ իրավիրճակները որոշված են կամ ժառանգված են `System.Exception` կլասից:
- Ծրագրի կասկածելի մասը դրվում է հսկման տակ try - ի միջոցով: Եթե անցանկալի երևույթ է տեղի ունենում, ապա ծրագիրը չի կախվում և կատարվում է catch -ի պարունակությունը: finally – ն պարտադիր չէ, սակայն եթե այն կա, ապա վերջում կատարվում է նրա պարունակությունը:
- Եթե կա try, ապա անպայման պետք է լինի կամ catch{…} կամ finally{…} կամ երկուսը միասին:

```
// try..catch
using System;
class M
{
    static void Main()
    {
        int a = 8;
        int b = 0;
        try
        {
            Console.WriteLine(a / b);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Console.WriteLine(e.TargetSite);
            Console.WriteLine(e.Source);
        }
        finally
        {
            Console.WriteLine("!!!!!!");
        }

    }
}
```

……

106.Արհեստական բացառիկ իրավիճակ

[Шилдт -414]

- Կարելի է նաև ստեղծել արհեստական արտակարգ իրավիճակ throw հրամանի միջոցով:
  throw exceptOb;
  exceptOb կարող է լինել System.Exception կլասի օբյեկտ կամ նրան ժառանգող:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        try
        {
            throw new DivideByZeroException();
        }
        catch (DivideByZeroException)
        {
            Console.WriteLine("mshakum");
        }
    }
}
/-----------------------------------

//throw - finally
using System;
class M
{
    static void Main()
    {
        int a = 8;
        int b = 0;
        try
        {
            throw new Exception();
            Console.WriteLine(a / b);
        }
        catch (Exception e)
        {
            Console.WriteLine(e.Message);
            Console.WriteLine(e.StackTrace);
            Console.WriteLine(e.HelpLink);
            // Exception ex =new Exception();
            // ex.HelpLink="http:\\www.albdarb.com";
            // throw ex;
        }
        finally
        {
            // return; error !
            Console.WriteLine("!!!!!!");
        }
    }
}
```

.....

107. Բազմակի catch – ի օգտագործում

[Троелсен-282; Нейгел-423]

- Բազմակի catch թույլարտվում է, սակայն ներկայցման հերթականությունը ունի նշանակություն և հիերարխիայի ավելի բարձրը ծածակում է ենթականերին, որը իր հերթին կարող է բերել կոնֆլիկտի:
- Բազմակի catch –ի դեպքում մշակվում է միայն մեկը, մյուսները անտեսվում են:
- Եթե catch –ում չկա պարամետր, ապա այն կոչվում է "ունիվերսալ" ցանկացած խափանում անցնում է այտեղով:
- Եթե կա բազմակի catch, ցանկալի է ավելի մասնակի դեպքերը գրել ավելի վերև, իսկ "ունիվերսալ" catch – ը վերջում: Հակառակ դեպքում մյուս catch –երը իմաստ չունեն և տալիս է թարգմանման սխալ:

```
using System;
class M
{
    static void Main()
    {
        string s;
        while (true)
        {
            try
            {
                Console.WriteLine("Enter number >0 end <5");
                s = Console.ReadLine();
                if (s == "")
                    break;
                int c = Convert.ToInt16(s);
                if (c < 0 || c > 5)
                    throw new IndexOutOfRangeException("; your input = " + c);
                throw new Exception ("; your input = "+c);
                Console.WriteLine("nuber = " + c);
            }
            catch (IndexOutOfRangeException ex)
            {
                Console.WriteLine("only >0 and <5 " + ex.Message);
            }

            catch (Exception ex)
            {
                Console.WriteLine("iaaaa heto: " + ex.Message);
            }
            catch
            {
                Console.WriteLine("other situation ");
            }
            finally
            {
                Console.WriteLine("!!!!!!");
            }
        }
    }
}
```

.....

108.Կրկնակի արտակարգ իրավիճակ (Rethrowing an Exception)

[Шилдт -415]

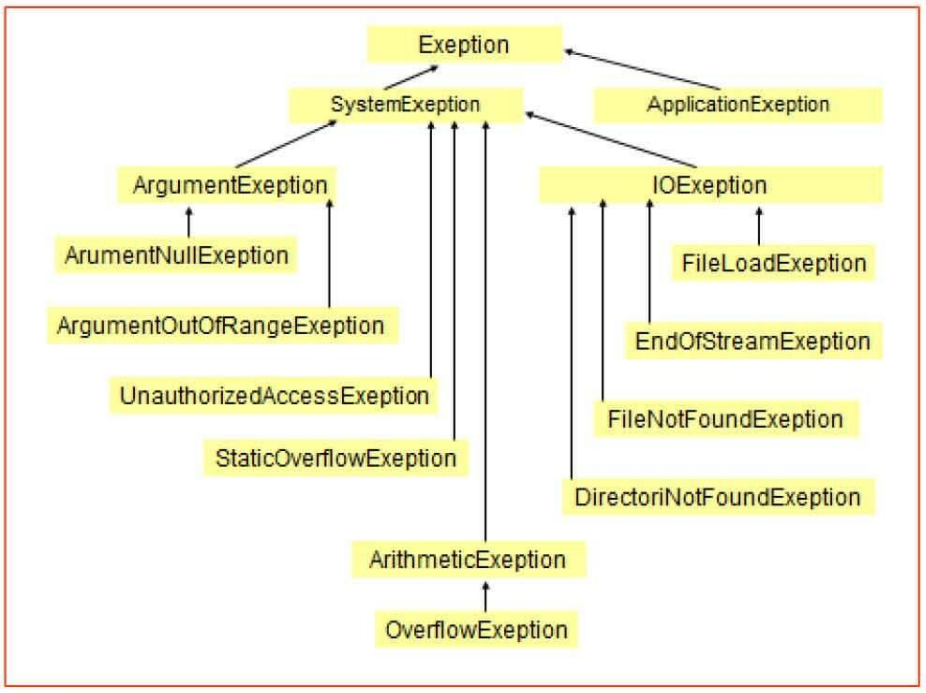
- Որևէ catch -ի բլոկում մշակվող արտակարգ իրավիճակը կարող է ստեղծել կրկնակի արտակարգ իրավիճակ, որը պետք է մշակվի արտաքին բլոկում: Դա տեղի ունի այն դեպքում, երբ անհաժեշտ է ունենալ մի քանի մշակողներ միևնույն արտակարգ իրավիճակի տարբեր "ասպեկտների" համար: Այսպիսի հնարավորության կարելի է հասնել օգտագործելով throw հրամանը առանց օբյեկտի.
  throw;
- Պետք է մոռանալ, որ throw հանդիպման դեպքում մշակումը չի կատարվում տվյալ catch -ի բլոկում, այլ հետ ե ուղղարկվում արտաքին catch -ի բլոկ: (Microsoft -ը շատ է սիրում այս գաղափարը):
- Օրինակում զրոյի բաժանումը կատարվում է `GenException()` մեթոդում, սակայն, երբ զանգվածի սահմանի անցում է կատարվում, ապա այն մշակվում է կրկնակի ձևով: Այլ խոսքով `IndexOutOfRangeException` -ը մշակվում է Main ում:

```
using System;// Шилдт 415
class R
{
    public static void GenException()
    {
        int[] numer = { 4, 8, 16, 32, 64, 128, 256, 512 };
        int[] denom = { 2, 0, 4, 4, 0, 8 };
        for (int i = 0; i < numer.Length; i++)
        {
            try
            {
                Console.WriteLine(numer[i] + " / " +
                denom[i] + " is " +
                numer[i] / denom[i]);
            }
            catch (DivideByZeroException)
            {
                Console.WriteLine("Can't divide by Zero!");
            }
            catch (IndexOutOfRangeException)
            {
                Console.WriteLine("No matching element found.");
                throw; // rethrow the exception
            }
        }
    }
}
class M
{
    static void Main()
    {
        try
        {
            R.GenException();
        }
        catch (IndexOutOfRangeException)
        {
            // recatch exception
            Console.WriteLine("Fatal error");
        }
    }
}
```

.....

109.System.Exception

[Шилдт-418; Троелсен-268; Нейгел-427]



- **`Exception`**
- `Exception` կլասը ունի հատկություններ `Message, StackTrace` և `TargetSite`.
- ✓ Message - սիմվոլային տող է, որը նշում է սխալի բնույթը:
- ✓ StackTrace կամ `ToString()` – տող է, որը նշում է սխալ առաջացնող տողի համարը և ամբողջ ճանապարհը:
- ✓ TargetSite – նշում է մեթոդը, որտեղ առաջանում է արտակարգ իրավիճակը:
- ✓ `Source` – նշում է ֆայլը:
- `Exception` կլասի կոնստրուկտորները:
- ✓ `public` Exception( ) – լռելիությամբ կոնստրուկտոր:
- ✓ `public` Exception(string *message*) – ստանում է Message :
- ✓ `public` Exception(string *message*, Exception *innerException*) – ներքին արտակարգ իրավիճակ է, երբ մի իրավիճակ առաջացնում է մի այլ արտակարգ վիճակ:
- ✓ `protected` Exception(SerializationInfo *info*, StreamingContext *context*) – նշված պարամետրերը գտնվում են System.Runtime.Serialization անունի տարածքում: Այն գործում է, երբ սխալի մշակումը կատարվում է հեռահար և դրա համար անհրաժեշտ է օբյեկտի սերիալիզացիա և դեսերիալիզացիա: (կանցնենք):
- **`SystemException`** -ից ժառանգվող և հաճախ օգտագործվող արտակագ իրավիճակներ.
- ✓ ArrayTypeMismatchException – տիպը չի համնկնում զանգվածի տիպի հետ:
- ✓ DivideByZeroException – զրոյի բաժանման փորձ:
- ✓ IndexOutOfRangeException – ինդեքսը անցել է զանգվածի չափից:
- ✓ InvalidCastException – սխալ տիպերի վերափոխարկում:
- ✓ OutOfMemoryException – անբավարար հիշողության պահանջ, որը կարող է առաջանալ `new` -ի կիրառումից:
- ✓ OverflowException – կարող է առաջանալ մաթեմատիկական հազեցումից,(переполнение):
- ✓ NullReferenceException – փորձարկում օգտագործել դատարկ հղում:

……

110.Ժառանգված բացառիկ իրավիճակ

[Шилдт-422]

- Չնայած նշված բացառությունները ծածկում են մեծ քանակով խնդիրների լուծում, C# -ի առավելություններից է նաև այն, որ հնարավոր է ստեղծել օգտագործողի կողմից բացառիկ իրավիճակներ: Դրա համար ուղղակի անհրաժեշտ է ժառանգել `Exception` կլասը, օգտվել նրա անդամներից և նույնիսկ վերառորոշել: Կարելի է օգտվել `Exception` կլասի կոնստրուկտորներից կիրառելով `base` հրամանը:
- Բացառիկ իրավիճակ մշակող օգտագործողի կլասները կարող են կապված լինել նաև ժառանգականությամբ: Ինչպես գիտենք գրադարանում գտնվող ժառանգականությամբ կապված բացառություններց բազայինը ծածկում է ժառանգին: Դրա համար ավելի թույլ բացառությունը պետք է գրված լինի ավելի առաջ `catch` մշակողների ցուցակում: Նույն մոտեցումը կիրառվում է նաև օգտագործողի կողմից ստեղծվող բացառությունների համար, որոնք կապված են ժառանգականությամբ:
- Օրինակում, եթե `catch (ExceptB exc)` և `catch (ExceptA exc)` մշակողների տեղերը փոխենք, ապա կառաջանա թարգմանության սխալ:

```
using System; //Шилдт-426
class ExceptA : Exception
{
    public ExceptA(string message) : base(message) { }
    public override string ToString()
    {
        return Message;
    }
}

class ExceptB : ExceptA
{
    public ExceptB(string message) : base(message) { }
    public override string ToString()
    {
        return Message;
    }
}

class M
{
    static void Main()
    {
        for (int x = 0; x < 3; x++)
        {
            try
            {
                if (x == 0) throw new ExceptA("Caught an ExceptA exception");
                else if (x == 1) throw new ExceptB("Caught an ExceptB exception");
                else throw new Exception();
            }
            catch (ExceptB exc)
            {
                Console.WriteLine(exc);
            }
            catch (ExceptA exc)
            {
                Console.WriteLine(exc);
            }
            catch (Exception exc)
```

```
            {
                Console.WriteLine(exc);
            }
        }
    }
}
```

.....

111.checked, unchecked

[Шилдт -428 ; Троелсен-139; Нейгел-209]

- Ներդրված տիպը կարող է ծրագրի աշխատանքի ժամանակ ստանալ արժեք, որը իր հնարավոր սահմաններից դուրս են: Այդ դեպքում տեղի է ունենում տվյալի կորուստ: Որպեսզի կարողանանք խուսափել դրանից օգտվում ենք checked հրամանից: Հրամանից հետո բլոկի մեջ գրվում է կասկածելի ծրագրի հատվածը և եթե տեղի է ունենում ինֆորմացիայի կորուստ, ապա կատարվում է ծրագրի ընդհատում և OverflowException արտակարգ իրավիճակ:
- unchecked հնարավորություն է տալիս բլոկի ներսում հսկամն հնարավորությունից հանել որոշ հրամաններ:
- Ծրագրի հրամանները լռելիությամբ unchecked հրամանի տակ են:

```
using System;
class Program
{
    static void Main(string[] args)
    {
        byte b = 254;
        //b++;
        //Console.WriteLine(b);
        byte c = 255;
        try
        {
           checked
           {
                unchecked
                {
                    c++;
                }
                b++;
            }
         }
         catch (OverflowException e)
         {
                Console.WriteLine(e.Message);
         }
        Console.WriteLine(c);
        Console.WriteLine(b);
    }
}
```

.....

Ընդհանրացումներ (Generics)

112.Ընդհանրացումներ

[Нэш-309; Шилдт-575; Нейгел-161; Рихтер-289; Троелсен-356]

- Օբյեկտ-կողմնորոշված ծրագրավորման հիմքերից է հանդիսանում **կոդի կրկնակի օգտագործումը**, որին կարելի է հասնել ժառանգականության, կոմպոզիցիայի կամ ներդրված կլասների շնորհիվ: Ջեներիքները նոր հնարավորություն են կոդի կրկնակի օգտագործման և ավելի նպատակային են դարձնում ալգորիթմների կրկնակի օգտագործումը: Օրինակ. ստեղծվում է սորտավորման, փնտրման, համեմատման և այլ ալգորիթմներ, և չի նշվում թե ինչ տիպերի հետ պետք է աշխատել: Տիպը որոշվում է դիմման ժամանակ (Рихтер) :
- Ընդհանրացումները նման են C++ -ի **շաբլոններին**, սակայն շաբլոնները ստատիկ են, այսինքն թարգմանման ժամանակ է կոդը որոշվում: Այստեղ գլխավոր թերությունը, որ այդ շաբլոնները հնարավոր չէ պահել DLL գրադարաններում (նրանք պահվում է ֆայլերում) և նրանց սիմվոլային կոդը տեսանելի է և խախտվում է հեղինակային իրավունքները:
- Ջեներիքները դինամիկ են, դա նշանակում է կոդը ստացվում է **կատարման ժամանակ**, և պարզ է այլ օգտագործողների կոդը կարելի է տրամադրել գրադարանների միջոցով, սակայն ավելի պաշտպանված է (Нэш) :
- Եթե նկարագրենք ջեներիքները, ապա նա պարամետրերով տիպ է: Ջեներիքներ կարող են լինել **կլասը, ստրուկտուրան, ինտերֆեյսը, դելեգատը, մեթոդը**: Կարելի է նշել, որ ընդհանրացում կարելի է կատարել նաև առանց ջեներիքի, օգտագործելով `Object` կլասը, քանի որ նա բազային է բոլորի համար և կարելի է նրան հղել, որպես ընդհանուր տիպ: Խնդիրը կայանում է նրանում, որ այս դեպքում չկա **տիպերի անվտանգություն** և կատարվում տիպերի աշխատատար **վերափոխարկում** (Шилдт):
- **Կրկնենք առավելությունները:**

✓ Նախնական կոդը պաշտպանված է, և տեսանելի չէ օգտվողներին: Նույնը չի կարելի ասել C+ և Java լեզուների մասին:

✓ Տիպերը գտնվում են անվտանգության մեջ: Դա հսկվում է հիմնականում թարգմանման ժամանակ և սխալը հայտնաբերվում է շուտ: Հակառակ դեպքում, երբ գործը հասնում է կատարման, ապա սկսում է աշխատել արտակարգ իրավիճակների մեխանիզմը:

✓ Արագ կոդ: Անհրաժեշտ չէ կիրառել տարբեր տիպերի **վերափոխարկումներ:**

✓ Արտադրողականության բարձրացում: Object կլասի հետ աշխատանքի կարիք չկա և **boxing/ unboxing** մեխանիզմի օգտագործում:

✓ Կոդի կրկնակի օգտագործում:

- Ընդհանրացումները թույլատրում են ընդունել ինչպես ցանկացած տիպ,այնպես էլ կարող կատարել տիպերի օգտագործման որոշակի սահմանափակումներ, օգտագործելով where օպերատորը:
- **Կլաս** ընդհանրացումը ունի հետևյալ տեսքը և նա <T> ում կարող է օգտագործել ցանկացած տիպ:

```
class A<T>
{
    public T a;
}
class M
{
    static void Main()
    {
        A<int> ob = new A<int>();
        ob.a = 5;
        // ob.a = 5.55; //error !
        A<double> ob2 = new A<double>();
        ob2.a = 5;
        ob2.a = 5.55;
```

```
    }
}
//-----------------------------------------------
```

- Տիպերը կարող են լինել նաև մեկից ավելին բաժանվելով ստորակետով <T,K,L>
- ✓ Եկու և ավելի պարամետրերով.

```
class A<T,K>
{
    public T a;
    public K b;
}
class M
{
    static void Main()
    {
       A<int,char> ob = new A<int,char>();
        ob.a = 5;
        // ob.a = "5"; //error !
        // ob.b = "5"; //error !
    }
}
```

……

113.Կլասների ընդհանրացում և where սահմանափակում

[Шилдт-585; Троелсен-383]

- Կլաս ընդհանրացումը կարող է ձևավորել որոշ սահմանափակումներ, որը կատարվում է where հրամանով: Գրման ձևը.

```
class class-name<T> where T: constraints
{
 // ...
}
```

- ✓ constraints – նշանալում է սահմանափակում:

որտեղ սահմանափակումները տարանջատվում են ստորակետով:

- Թույլատրվում են հետևյալ տիպի սահմանափակումներ.
- ✓ Բազային կլասի սահմանափակում:
- ✓ Ինտերֆեյսի սահմանափակում:
- ✓ Կոնստրուկտորի սահմանափակում, որը պահանջում է լինել լռելիությամբ: Օգտագործվում է new հրամանը, որը թույլատրում է ջեներիք կլասի ներսում ջեներիք տիպի օբյեկտ ստեղծել:
- ✓ Հղիչային տիպի սահմանափակում, որը օգտագործում է class հրամանը
- ✓ Չափային տիպի սահմանափակում, որը օգտագործում է struct հրամանը

- Թույլատրվում է մի քանի սահմանափակումներ միասին, սակայն որոշ կարգով: Օրինակ.

```
class Gen<T> where T : MyClass, IMyInterface, new()
{
}
```

- ✓ Միաժամանակ չի թույլատրվում class և struct
- Օրինակ. Եթե <T> ում նշված է տիպ, որ չի օգտագործում նշված բազային կլասը, ապա կատարվում է արգելափակում: Օրինակ.

```
using System;
class A
{
    public void Hello()
    {
        Console.WriteLine("Hello");
```

```
    }
}
// Class B inherits A.
class B : A { }
// Class C does not inherit A.
class C { }
class Test<T> where T : A
{
    T ob;
    public Test(T o)
    {
        ob = o;
    }
    public void SayHello()
    {
        ob.Hello();
    }
}
class M
{
    static void Main()
    {
        A a = new A();
        B b = new B();
        C c = new C();
        Test<A> t1 = new Test<A>(a);
        t1.SayHello();
        Test<B> t2 = new Test<B>(b);
        t2.SayHello();
       // Test<C> t3 = new Test<C>(c);
       // t3.SayHello(); // Error!
    }
}
```

- C կլասի համար չի թույլատրվում ստեղծել օբյեկտ, որովհետև այն չի ժառանգված A կլասից:

Ընդհանրացում և կոնստրուկտորի կիրառման սահմանափակում (new) [Шилдт-598]

- (new) սահամանափակումը թույլատրում է ստեղծել ընդհանրացման օբյեկտ, որը հնարավոր չէ ընդհանուր դեպքում: new -ն ստիպում է լռելիության կոնստրուկտորի առկայություն:
- (new) -ի կիրառման սահմանափակումներ.

✓ (new) -ն կարելի է օգտագործել մյուս սահմանափակումների հետ, բայց միայն վերջում:

✓ (new) -ն ցույց է տալիս, որ հնարավոր չէ պարամետր փոխանցել:

✓ new) -ն հնարավոր չէ օգտագործել struct սահմանափակման հետ, որը նշում է չափային տիպ:

```
using System; //Шилдт -598
class MyClass
{
    public MyClass()
    {
        // ...
    }
}
class Test<T> where T : new()
{
    T obj;
    public Test()
    {
        // This works because of the new() constraint.
        obj = new T(); // create a T object
```

```
    }
}
class M
{
    static void Main()
    {
        Test<MyClass> x = new Test<MyClass>();
    }
}
//-----------------------------------
```

Հղիչային և չափային տիպերի սահմանափակում: [Шилдт-599]

- Այն կատարվում է `class` և `struct` հրամանների միջոցով:

```
using System;//Шилдт -600
class MyClass
{
}
// Use a reference constraint.
class Test<T> where T : class
{
    T obj;
    public Test()
    {
        // The following statement is legal only
        // because T is guaranteed to be a reference
        // type, which can be assigned the value null.
        obj = null;
    }
}
class M
{
    static void Main()
    {
        // The following is OK because MyClass is a class.
        Test<MyClass> x = new Test<MyClass>();
        // The next line is in error because int is a value type.
        // Test<int> y = new Test<int>();
    }
}
//-----------------

using System; //Шилдт -601
struct MyStruct
{
}
class MyClass
{
}
class Test<T> where T : struct
{
    T obj;
    public Test(T x)
    {
        obj = x;
    }
}
class M
{
    static void Main()
```

```
    {
        // Both of these declarations are legal.
        Test<MyStruct> x = new Test<MyStruct>(new MyStruct());
        Test<int> y = new Test<int>(10);
        // But, the following declaration is illegal!
        // Test<MyClass> z = new Test<MyClass>(new MyClass());
    }
}
```

.....

114.Generics – մեթոդներ

[Шилдт-607; Троелсен-376, 347; Нэш-316]

- Եթե մեթոդը գտնվում է ընդհանրացում կլասում, ապա ընդհանրացված տիպը կարող է լինել նաև մեթոդում և ավտոմատ դարնում է ընդհանրացված: Սակայն կարելի է նաև ստանալ ընդհանրացված մեթոդ, որը գտնվում է ոչ ընդհանրացված կլասում:
- Ընդհանրացված մեթոդները ավելի ժամանակակից են համարվում գերբեռնված մեթոդների հետ համեմատած: Չնայած մեթոդների գերբեռնումը ունի մեծ կիրառություն, շատ դեպքերում նրանք կատարում են նույն ծրագիրը տարբեր տիպերի համար:

❖ Օրինակը ցուցադրում է պարամետրի միջոցով ուղղարկված տվյալների փոխանակում: Փոխանակումը կատրավում է և `int` և `string` տիպերի համար: `ref` օգրագործվում է նրա համար, որպեսզի մեթոդը կարողանա մեկից ավելի քանակով արժեք վերադարձնել պարամետրի միջոցով:

```
using System; //Troelsen 349
class M
{
    static void Swap<T>(ref T a, ref T b)
    {
        T temp;
        temp = a;
        a = b;
        b = temp;
    }
    static void Main()
    {
        // Swap 2 ints.
        int a = 10, b = 90;
        Console.WriteLine("Before swap: {0}, {1}", a, b);
        Swap<int>(ref a, ref b);
        Console.WriteLine("After swap: {0}, {1}", a, b);
        // Swap 2 strings.
        string s1 = "Hello", s2 = "There";
        Console.WriteLine("Before swap: {0} {1}!", s1, s2);
        Swap<string>(ref s1, ref s2);
        Console.WriteLine("After swap: {0} {1}!", s1, s2);
    }
}
```

- Ընդլայնված մեթոդները նույնպես (ինչպես կլասները) կարող են ընդունել սահմանափակումներ: [Шилдт-610];

❖ Նշված օրինակը չի աշախատի, որովհետև `where T:class` հրամանը չի թույլատրում չափային տիպի օգտագործում:

```
using System; //albdarb
class M
{
    static void Swap<T>(ref T a, ref T b) where T :class
    {
        T temp;
        temp = a;
```

```
        a = b;
        b = temp;
    }
    static void Main()
    {
        // Swap 2 ints.
        int a = 10, b = 90;
        Console.WriteLine("Before swap: {0}, {1}", a, b);
        Swap<int>(ref a, ref b); // error !!!
        Console.WriteLine("After swap: {0}, {1}", a, b);
    }
}
```

.....

115.Generics – դելեգատներ

[Троелсен-402,370; Шилдт-610; Нэш-318]

- Ինչպես մեթոդները, դելեգատները նույնպես կարող են օգտվել ընդհանրացումներից:

❖ Օրինակ.

```
using System; //Troelsen 370
public delegate void ddd<T>(T arg);
class M
{
    static void Main()
    {
        //  traditional delegate syntax.
        ddd<string> obddd = new ddd<string>(f1);
        obddd("albdarb.com");
        ddd<int> obddd2 = new ddd<int>(f2);
        obddd2(9);
        //  using method group conversion.
        ddd<int> obddd3 = f2;
        obddd3(9);
    }
    static void f1(string arg)
    {
        Console.WriteLine(arg.ToUpper());
    }
    static void f2(int arg)
    {
        Console.WriteLine(++arg);
    }
}
```

❖ Օրինակ, որը իրականացնում է լամդա արտահայտություն.

```
using System; //albdarb
public delegate void ddd<T>(T arg);
class M
{
    static void Main()
    {
        ddd<string> ob;
        ob = (a) =>
            {
                Console.WriteLine(a.ToUpper());
            };
        ob("Albert Darbinyan");
    }
}
```

- Ընդհանրացումներ դելեգատները օժտված են կովարյանտությամբ և կոնտրավարյանտությամբ:

116.Action<> և Func<> դելեգատներ

[Троелсен -371; Нейгел- 247; Covaci - 216]

- Դելեգատի օգտագործման ժամանակ հաճախ անհրաժեշտ չի լինում ունենալ դելեգատի հստակ անվանումը, համարվելով այն ոչ կարևոր: Ուղղակի պետք է լինում ինչ որ դելեգատ, որի համար կարևոր են միայն պարամետրերը: Նշված դեպքի համար նախատեսված է ներդրված դելեգատներ Action<> и Func<> :
- Action<> դելեգատը ներդրված է System անունի տարածքում: Այն կարող է նշել(հղել) մինչև 16 արգումենտներով մեթոդների և վերադարձնում է void: Քանի որ, այն ընդհանրացված է, ապա պետք է նշել նաև յուրաքանչյուր պարամետրի տիպը:

❖ Action<> դելեգատի օրինակ.

```
using System; // Troelsen 372
class Program
{
    static void DisplayMessage(string msg, ConsoleColor txtColor, int printCount)
    {
        // Set color of console text.
        ConsoleColor previous = Console.ForegroundColor;
        Console.ForegroundColor = txtColor;
        for (int i = 0; i < printCount; i++)
        {
            Console.WriteLine(msg);
        }
        // Restore color.
        Console.ForegroundColor = previous;
    }
    static void Main(string[] args)
    {
        // Use the Action<> delegate to point to DisplayMessage.
        Action<string, ConsoleColor, int> actionTarget = new Action<string,
                                                ConsoleColor, int>(DisplayMessage);
        actionTarget("Action Message!", ConsoleColor.Yellow, 5);
        Console.WriteLine("albdarb copyright");
        Console.ReadKey();
    }
}
```

- Func<> դելեգատը նույնպես կարող է ստանալ 16 պարամետր, սակայն նա տարբերվում է Action<> դելեգատից նրանով, որ կարող է նաև արժեք վերադարձնել: Վերադարձնող պարամետրը գտնվում է պարամետրերի վերջում:

❖ Func<> դելեգատի օրինակ.

```
using System; // Troelsen 376
class Program
{
    // Target for the Func<> delegate.
    static int Add(int x, int y)
    {
        return x + y;
    }
    static string SumToString(int x, int y)
    {
        return (x + y).ToString();
    }
    static void Main(string[] args)
    {
        Func<int, int, int> funcTarget = new Func<int, int, int>(Add);
```

```
        int result = funcTarget.Invoke(50, 50);
        Console.WriteLine(result);
        Func<int, int, string> funcTarget2 = new Func<int, int, string>(SumToString);
        string sum = funcTarget2(10, 10);
        Console.WriteLine(sum);
        Console.WriteLine("albdarb copyright");
        Console.ReadKey();
    }
}
```

❖ Func<> դելեգատի օրինակ լամբդա արտահայտության միջոցով.

```
using System; //alb
class Program
{
    static void Main(string[] args)
    {
        Func<string,int,int> ob;
        ob = (s,n) =>
        {
            Console.Write(s);
            n++;
            return n;
        };
        Console.WriteLine(ob("alb= ",50));
    }
}
```

❖ Action<> դելեգատը ունի նաև ոչ ընդհանրացված տարբերակ, իսկ Func<>-ը, ոչ: Օրինակ առանց ընդհանրացման.

```
using System;
class Program
{
    static void Main(string[] args)
    {
        Action ob;
        ob = () => Console.WriteLine("barev");
        ob();
    }
}
```

.....

117.Generics – ինտերֆեյսներ

[Шилдт-612]

- Ինտերֆեյսների ընդհանրացումը նման է կլասներին:

❖ Օրինակում ինտերֆեյսը որոշում է տիպ, որը պարտադրվում է օգտագործել մեթոդի պարամետրում:

```
using System;
interface IS<T>
{
    void ff(T a);
}
class A<T> : IS<T>
{
    public void ff(T a)
    {
        T aa = a;
        Console.WriteLine(aa);
```

```
    }
}
class Program
{
    static void Main(string[] args)
    {
        A<int> o1 = new A<int>();
        o1.ff(886);
        A<char> o2 = new A<char>();
        o2.ff('A');
    }
}
```

.....

118.Ինտերֆեյսի Covariance ընդհանրացումներ (NET 4)

[Шилдт - 626; Нейгел-171; Уотсон-362]

- Եթե մեթոդներում կովարյանտությունը ներդրված է, իսկ կոնտրավայրանտությունը արգելված, ապա ընդհնրացումներում արգելված են երկուսը միասին:
- Բոլոր ընդհանրացումները ինվարիանտ են: Դա նշանակում է խիստ տիպային հսկում և անվտանգություն: Նույնիսկ չի թույլատրվում տիպերի վերափոխարկում: Սակայն NET4 միջավայրում կատարվում է հնարավորությունների ընդլայնում, որը կիրառվում է դելեգատ ընդհանրացումների և ինտերֆեյս ընդհանրացումների համար: Այդ դեպքում կատարվում է ինվարիանտությունից անցում կովարիանտության և կոնտրավարյանտության:
- Եթե անհրաժեշտ է ապահովել Covariance և Contravariance հնարավոտրությունները, անհրաժեշտ է գրել out կովարիանտության համար, որպես վերադարձնող արժեք և ավելացնել in կոնտրավարյանտության համար, որպես պարամետր:
- Covariance և Contravariance հնարավոտրությունները դա պոլիմորֆիզմի կիրառման ձև է:

```
using System;
interface ICov<out T>
{
    T ff();
}
class Shape
{
}
class Rectangle:Shape
{
}
class ACov<T> : ICov<T>
{
    T t;
    public T ff()
    {
        return t;
    }

}
class Program
{
    static void Main(string[] args)
    {
        ICov<Shape> i1 = new ACov<Shape>();// invariant
        ICov<Shape> i2 = new ACov<Rectangle>();  //covariant
        //ICov<Rectangle> i3 = new ACov<Shape>();  //contravariant   //error!!!!
    }
}
```

119.Ինտերֆեյսի Contravariance ընդհանրացումներ

[Шилдт -630; Нейгел-174; Уотсон-364]

```
using System;
interface IContra<in T>
{
    void ff2(T o);
}
class Shape
{

}
class Rectangle:Shape
{

}
class AContr<T> : IContra<T>
{
    public void ff2(T o)
    {
        T oo = o;
    }
}
class Program
{
    static void Main(string[] args)
    {
        IContra<Rectangle> i1 = new AContr<Rectangle>(); // invariant
        //IContra<Shape> i2 = new AContr<Rectangle>();  //covariant   //error!!!!
        IContra<Rectangle> i3 = new AContr<Shape>();  //contravariant
    }
}
```

.....

120.Կլասի հետաձգված ինիցալիզացիա - Lazy<>

[Троелсен-314; Гриффитс-873]

- Կլասների նախագծման ժամանակ լինում են դաշտեր, որոնք հնարավոր է չօգտագործվեն ֆունկցիաների կողմից: Դա առանձնապես վնաս չէ, եթե դաշտի մեծությունը համեմատաբար փոքր է: Սակայն դաշտի մեծ չափերի դեպքում ցանկալի է կիրառել մոտեցումներ, որոնք թույլ կտան դաշտերը ստեղծել միայն օգտագործման ժամանակ: Այդպիսի ձևերից է օրինակ է Factory Method փաթերնը:
- C# 2010 –ում գոյություն ունի Lazy <> կլաս, որը նույնպես լուծում է վերը նշված խնդիրը: Այն գտնվում է System անունի տարածքում:
- Lazy <> կլասը բացի լռելիությամբ կոնստրուկտորից ունի նաև պարամետրով կոնստրուկտոր, որի պարամետրը կարող է ընդունել ընդհանրացված դելեգատ իր System.Func<> տիպով: Այն կարող է ունենալ մինչև 16 արգումենտ:
- Func <> -ի աշխատանքը պարզեցնելու համար ցանկալի է օգտագործել լյամբդա արտահայտություն:
- Օրինակում `GC.GetTotalMemory(false)` մեծ հիշողության օգտագործում է ցույց տալիս: Օրինակում ավելի քիչ է հիշողությունը է զբաղեցնում Lazy<> կլասի օգտագործումը :
- `A oA= oC.Get()` հրամանն է միայն ավելացնում հիշողություն:

```
using System; // Lazy <> // albdarb
using System.Collections.Generic;
class A
```

```
{
    int[] a = new int[100000];
}
class C
{
    Lazy<A> oL= new Lazy<A>();
   // A oA = new A();
    public A Get()
    {
        return oL.Value;
        //return oA;
    }
}
class M
{
    static void Main(string[] args)
    {
        C oC = new C();
        //A oA= oC.Get();
        Console.WriteLine(GC.GetTotalMemory(false));
        Console.ReadKey();
    }
}
//.....
```

- Եթե անհրաժեշտ է օգտագործել լամբդա արտահայտությանը, ապա այն կարելի է կիրառել հետևյալ ձևով.

```
Lazy<A> oL =
new Lazy<A>
(     () =>
      {
          return new A();
      }
);
.....
```

END C#sharp 1

Թեսթի Օրինակներ. (25 հարց 1 ժամ 20 րոպե)(25x2=50 միավոր)

❖ Ծրագիրը կարտածի (1 բալ)

```
using System;
class Program
{
      static void Main()
      {
            short a = 1;
            switch (a)
            {
                  case 1: Console.Write("1");
                  case 2: Console.Write("2");
                        break;
                  default: Console.WriteLine("d");
                        break;
            }
      }
}
```

a) 1

b) 12

c) 12d

d) d

e) ոչ, կտա թարգմանման սխալ

.....

❖ Ինչ արդյունք կարտածի ծրագիրը (2 բալ)

```
using System;
class M
{
    static void Main()
    {
        string i = "1";
        string j = "1";
        Console.WriteLine(i == j);
        Console.WriteLine(Equals(i, j));
        Console.WriteLine(i.Equals(j));
        Console.WriteLine(ReferenceEquals(i, j));
    }
}
```

a) TrueTrueTrueFalse

b) TrueFalseTrueFalse

c) TrueTrueFalseTrue

d) TrueTrueTrueTrue

.....

❖ `int @int = new int();` (1 բալ)

a) ճիշտ հրաման չէ

b) `int` փոփոխականին վերագրում է 0 արժեք

c) կատարվում է `int` հասցեի վերագրում

d) հայտարավում է նոր տիպ

e) նշվածներից ոչ մեկը ճիշտ չէ

❖ Main – ում ինտերֆեյսը օգտագործվում է (3 բալ)

```
interface I<in T>
{
    void ff(T o);
}
class Shape
{           }
class Rectangle : Shape
{           }
class A<T> : I<T>
{
    public void ff(T o)
    {
        T oo = o;
    }
}
class Program
{
    static void Main(string[] args)
    {
            I< Shape > i3 = new A< Rectangle >();
    }
}
```

a) կովարյանտ ձևով

b) կոնտրավարյանտ ձևով

c) տալիս է թարգմանման սխալ

d) նշվածներից կա երկու ճիշտ պատասխան

……

❖ Ծրագիրը (2 բալ)

```
using System;
class AAA
{
    public int a = 9;
}
class BBB : AAA
{
       public int b;
}
class M
{
        public static void Main()
        {
            AAA obAAA = new AAA();
            BBB obBBB = new BBB();
            obAAA = obBBB;
            obAAA.a = 44;
            Console.WriteLine(obAAA.a);
            obAAA.b=55;
             Console.WriteLine(obAAA.b);
         }
}
```

a) կարտածի 44,55 հաջորդականությունը

b) կտա թարգմանության սխալ

c) կտա կատարման սխալ

d) կտա զգուշացում

……