МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРКАСЬКИЙ ДЕРЖАВНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ І СИСТЕМ
Кафедра програмного забезпечення автоматизованих систем

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ

до виконання лабораторних робіт з
дисципліни **«ПРОГРАМУВАННЯ НА ОСНОВІ .NET»**
для здобувачів освітнього ступеня «бакалавр»
за спеціальністю 121 – Інженерія програмного забезпечення,
освітня програма «Інженерія програмного забезпечення»
галузі знань 12 - Інформаційні технології усіх форм навчання
усіх форм навчання

Черкаси 2024

Затверджено вченою радою ФІТІС,
протокол №___ від _____2024 р.,
згідно з рішенням кафедри
програмного забезпечення
автоматизованих систем,
протокол №____ від _____2024 р.

Упорядники: **Куницька С.Ю.**, к.т.н., доцент
Метелап В.В., к.т.н., доцент

Рецензент: Захарова В.М., к.т.н., доцент кафедри комп'ютерної інженерії та інформаційних технологій ЧДБК

Методичні рекомендації до виконання лабораторних робіт з дисципліни **«ПРОГРАМУВАННЯ НА ОСНОВІ .NET»** для здобувачів освітнього ступеня «бакалавр» за спеціальністю 121 – Інженерія програмного забезпечення, освітня програма «Інженерія програмного забезпечення» галузі знань 12 - Інформаційні технології усіх форм навчання [Електронний ресурс] / [упорядники: С.Ю. Куницька, В.В. Метелап, В.В. Немченко] ; М-во освіти і науки України, Черкас. держ. технол. ун-т. Черкаси : ЧДТУ, 2024. - 102 с. – Назва з титульного екрана.

Метою навчальної дисципліни є забезпечення здатності студентів розуміти та засвоїти основні принципи ООП для побудови ПЗ з використанням .NET, набуття практичних навичок по створенню ПЗ з використанням сучасної об'єктно-орієнтованої мови програмування C#.

Навчальне електронне видання
мережного використовування

МЕТОДИЧНІ РЕКОМЕНДАЦІЇ
до виконання лабораторних робіт з
дисципліни «ПРОГРАМУВАННЯ НА ОСНОВІ .NET»
для здобувачів освітнього ступеня «бакалавр»
за спеціальністю 121 – Інженерія програмного забезпечення,
освітня програма «Інженерія програмного забезпечення»
галузі знань 12 - Інформаційні технології усіх форм навчання
усіх форм навчання

Упорядники: Куницька Світлана Юріївна
Метелап Володимир Володимирович

В авторській редакції

ЗМІСТ

ВСТУП……4
ВИМОГИ ДО ВИКОНАННЯ, ОФОРМЛЕННЯ ТА ЗАХИСТУ ЛАБОРАТОРНИХ РОБІТ……...6
ЗАВДАННЯ ДО ЛАБОРАТОРНИХ РОБІТ ……..……7
Лабораторна робота №1……7
Лабораторна робота №2……..……21
Лабораторна робота №3…….……32
Лабораторна робота №4…….46
Лабораторна робота №5……..53
Лабораторна робота №6……..57
Лабораторна робота №7……..…….....66
Лабораторна робота №8……..…….....71
Лабораторна робота №9……..77
Лабораторна робота №10…….……...81
Лабораторна робота №11……91
Лабораторна робота №12……99
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ…….102

ВСТУП

Програма навчальної дисципліни «**Програмування на основі .NET**» з циклу професійної підготовки обов'язкових дисциплін складена з урахуванням вимог до підготовки здобувачів вищої освіти бакалаврського рівня за освітньо-професійною програмою «Інженерія програмного забезпечення».

Предметом вивчення навчальної дисципліни є знання з того, що таке основні принципи ООП, застосування та використання ООП для розробки та проектування ПЗ для сучасних операційних систем, баз даних, десктоп додатків, хмарних середовищ, web-додатків з використанням технології .NET, зокрема мови C#.

Мета та завдання навчальної дисципліни - забезпечити здатність студентів розуміти та засвоїти основні принципи ООП для побудови ПЗ з використанням .NET, набуття практичних навичок по створенню ПЗ з використанням сучасної об'єктно-орієнтованої мови програмування C#. А також детально розглядаються методи і механізми:

– ознайомлення з основними поняттями та принципами ООП, засобами і проблемами розробки ПЗ;

– опанування об'єктно–орієнтованим підходом для проектування та розробки складного програмного забезпечення.

У відповідності до освітньо–професійної програми та досвіду наукової праці розглядається розробка ПЗ з використанням об'єктно–орієнтованого підходу та об'єктно– орієнтованого проектування, викладаються поняття про основні принципи ООП та їх використання при розробці ПЗ, закріплюються знання, вміння та навички отримані при вивчених попередніх предметах з ООП використовуючи сучасну, на поточний момент, об'єктно – орієнтовану мову програмування.

На вивчення навчальної дисципліни відводиться 120 годин 4 кредити ЄКТС. Форма підсумкового контролю успішності навчання – іспит.

Об'єктом підсумкового контролю знань студентів у формі іспиту є

розв’язання контрольних завдань та відповідь на теоретичні та практичні питання.

На іспит виносяться базові питання та завдання, що потребують творчого підходу та вміння опрацьовувати лекційний матеріал та технічну інформацію самостійно.

ВИМОГИ ДО ВИКОНАННЯ, ОФОРМЛЕННЯ ТА ЗАХИСТУ ЛАБОРАТОРНИХ РОБІТ

Матеріал розроблено з урахуванням міждсциплінарних звязків та додає новий інструментарій та розуміння методів та технологій для розробки програмного забезпечення.

Лабораторні роботи розраховані на підготовлених студентів, які вивчили та засвоїли матеріал з попередніх дисциплін та дисциплін на яких базується дисципліна. Студент має прочитати, набрати, за необхідності відлагодити їх та запустити на виконання коди, що представлені в методичці.

Після виконання прикладів необхідно показати викладачу про виконання і після цього взяти індивідуальне завдання, для поглибленого засвоєння матеріалу, або придумати самостійно і виконати процес розробки. Після здійснення розробки необхідно завантажити код на репозиторій git і посилання скинути викладачу на перевірку. Оформлення звіту необхідне після підтвердження роботи програмного забезпечення викладачем та бесіди з ним по темі роботи. За умови вчасної (до наступного або під час наступного заняття) здачі викладачу програмної частини немає необхідності друкувати звіт з лабораторної роботи. Звіт повинен включати теоретичні відомості в необхідному для відповіді студента об’ємі і на його розсуд. Наприкінці звіту обов’язково повинно бути не менше 10 питань по темі лабораторної роботи складені особисто. Висновки по роботі є невід’ємною частиною звіту з вказанням переваг та недоліків виченого в порівнянні з пройденим матеріалом на інших предметах або вивченого самостійно.

За погодженням з викладачем можна придумати власне завдання по темі. Звіт складається по стандартній формі.

Виконання та захист всіх лабораторних робіт згідно навчального індивідуального плану студента є обов’язковою умовою для допуску до іспиту.

ЛАБОРАТОРНА РОБОТА №1

Тема: «Введення. Основи .NET/C#».

Введення в написання програм використовуючи C# 9.0 и платформи .NET 5. Змінні і типи даних. Літерали. Операції з числами умовні вирази. Операції з рядками. Умовна конструкція if. Цикли. Масиви. Масиви параметрів (params). Методи. Область видимості змінних. Кортежі. Enum. Модулі.

Мета роботи: Дослідити та засвоїти основні принципи роботи .NET/C# на базі основних елементів та конструкцій.

Завдання

1. Дано два дійсні числа *a* і *b*. Одержати їхню суму, різницю та добуток.
2. Дано довжину ребра куба. Знайти об'єм куба та площу його бічної поверхні.
3. Дано два дійсні додатні числа. Знайти середнє арифметичне та середнє геометричне цих чисел.
4. Дано два дійсні числа. Знайти середнє арифметичне цих чисел і середнє геометричне їхніх модулів.
5. Дано катети прямокутного трикутника. Знайти його гіпотенузу та площу.
6. Змішано v_1 літрів води температури t_1 з v_2 літрами води температури t_2. Знайти об'єм і температуру суміші, що утворилася.
7. Визначити периметр правильного n-кутника, описаного біля кола радіуса r.
8. Три опору опір опір з'єднання. R_1, R_2 , R_3 з'єднані паралельно. Знайти опір з'єднання.
9. Визначити час падіння каменя на поверхню землі з висоти h .
10. Дано x, y, z. Обчислити a, b, якщо

а) $a = \dfrac{\sqrt{|x-1|} - \sqrt[3]{|y|}}{1 + \dfrac{x^2}{2} + \dfrac{y^2}{4}}$, $b = x(\text{arctg}(z) + e^{-(x+3)})$;

б) $a=\dfrac{3+e^{y-1}}{1+x^2\left|y-\operatorname{tg} z\right|}$, $b=1+\left|y-x\right|+\dfrac{(y-x)^2}{2}+\dfrac{\left|y-x\right|^3}{3}$;

в) $a=(1+y)\dfrac{x+y/(x^2+4)}{e^{-x-2}+1/(x^2+4)}$, $b=\dfrac{1+\cos(y-2)}{x^4/2+\sin^2 z}$;

г) $a=y+\dfrac{x}{y^2+\left|\dfrac{x^2}{y+x^3/3}\right|}$, $b=(1+\operatorname{tg}^2\dfrac{z}{2})$;

д) $a=\dfrac{2\cos(x-\pi/6)}{1/2+\sin^2 y}$, $b=1+\dfrac{z^2}{3+z^2/5}$;

е) $a=\dfrac{1+\sin^2(x+y)}{2+\left|x-2x/(1+x^2y^2)\right|}+x$, $b=\cos^2(\operatorname{arctg}\dfrac{1}{z})$;

ж) $a=\ln\left|(y-\sqrt{|x|})(x-\dfrac{y}{z+x^2/4}\right|$, $b=x-\dfrac{x^2}{3!}+\dfrac{x^5}{5!}$.

11. Дано сторону рівностороннього трикутника. Знайти площу цього трикутника.

12. Обчислити період коливання маятника довжини l.

13. Визначити силу тяжіння F між тілами маси m1 та m2 , що знаходяться на відстані r один від одного.

14. Дано гіпотенузу та катет прямокутного трикутника. Знайти другий катет і радіус вписаного кола.

15. Відома довжина кола. Знайти площу кола, обмеженого цим колом.

16. Знайти площу кільця, внутрішній радіус якого дорівнює 20, а зовнішній – заданому числу r (r>20).

17. Трикутник задано величинами своїх кутів і радіусом описаного кола. Знайти сторони трикутника.

18. Знайти площу рівнобокої трапеції з основами a і b Та кутом α при більшій основі a.

19. Трикутник задано довжинами сторін. Знайти:

а) довжини висот;

б) довжини медіан;

в) довжини бісектрис;

г) радіуси вписаного й описаного кіл.

20. Трикутник задано координатами своїх вершин. Знайти:

а) периметр трикутника;

б) площа трикутника.

21. Знайти площу сектора, радіус якого дорівнює 13.7, а дуга містить задане число радіан .

22. Дано дійсні додатні числа a, b, c. За трьома сторонами з довжинами a, b, c можна побудувати трикутник. Знайти кути трикутника.

23. Дано дійсне число a. Не користуючись жодними іншими арифметичними операціями, крім множення, отримати:

а) a^4 за дві операції;

б) a^6 за три операції;

в) a^7 за чотири операції;

г) a^8 за три операції;

д) a^9 за чотири операції.

Розгалуження

24. Дано дійсні числах, y. Отримати: а) max (x, y); б) min (x, y); в) max (x, y) , min (x, y).

25. Дано дійсні числа x, y, z. Отримати: а) max (x, y, z); б) min (x, y, z), max(x, y, z). 35. Дано дійсні числа x, y, z. Обчислити: а) max (x + y + z, xyz); б) min2 (x + y + z/2, xyz) +1.

26. Дано дійсні числа a, b, c. Перевірити, чи виконуються нерівності a < b < c.

27. Дано дійсні числа a, b, c. Подвоїти ці числа, якщо a ≥ b ≥ c , і замінити їх абсолютними значеннями, якщо це не так.

28. Дано дійсні числах, y. Обчислити z: якщо x > y, то x - y, а в іншому випадку y – x +1 39. Дано два дійсні числа. Вивести перше число, якщо воно більше за друге, і обидва числа, якщо це не так.

29. Дано два дійсні числа. Замінити перше число нулем, якщо воно менше або дорівнює другому, і залишити числа без зміни в іншому випадку.

30. Дано три дійсні числа. Вибрати з них ті, що належать інтервалу (1, 3).

31. Дано дійсні числа x, y (ξ≠ y). Менше з цих двох чисел замінити їхньою півсумою, а більше - їхнім подвоєним добутком.

32. Дано три дійсні числа. Звести до квадрата ті з них, значення яких невід'ємні.

33. Якщо сума трьох попарно різних дійсних чисел x, y, z меншаза одиницю, то найменше з цих чисел замінити напівсумою двох інших; у протилежному випадку замінити найменше з x і y напівсумою двох значень, що залишилися.

34. Дано дійсні числа a, b, c, d. Якщо α≤ b ≤ c ≤ d, то кожне число замінити найбільшим із них; якщо a>b>c>d, то залишити без зміни; інакше всі числа замінюються їхніми квадратами.

35. Дано дійсні числа x, y. Якщо x і y від'ємні, то кожне значення замінити його модулем; якщо від'ємним є лише одне з них, то обидва значення збільшити на 0.5; якщо обидва значення невід'ємні й жодне з них не належить до відрізка [0.5, 2.0], то обидва значення зменшити в 10 разів; у решті випадків x і y залишити без зміни.

36. Дано дійсні додатні числа x, y, z. а) З'ясувати, чи існує трикутник із довжинами сторін x, y, z. б) Якщо трикутник існує, то відповісти - чи є він гострокутним.

37. Дано дійсні числа a, b, c (a ≠ 0). З'ясувати, чи має чи рівняння ax^2 + bx+ c = 0 дійсні корені. Якщо дійсні корені є, то знайти їх. В іншому випадку відповіддю має слугувати повідомлення, що дійсних коренів немає.

38. 51. Дано дійсні числа a, b, c (a ≠ 0). Повністю дослідити біквадратне рівняння ax 4 + bx2 + c = 0, тобто якщо дійсних коренів немає, то повинно бути видано повідомлення про це, інакше має бути видано два або чотири корені.

39. Дано дійсні числа a, b, c, d, s, t, u (s і t одночасно не дорівнюють нулю). Відомо, що точки (a, b) і (c, d) не лежать на прямій l, заданій рівнянням sx + ty + u = 0 . Пряма l розбиває координатну площину на дві півплощини.

З'ясувати, чи вірно,що точки (a, b) і (c, d) належать різним півплощинам*). *) У цій задачі, як і в низці наступних задач, треба скористатися тим, що дві точки (a, b) і (c, d), які не лежать на прямій, що визначається рівнянням sx + ty + u = 0 , належать одній півплощині, якщо sa + tb + u і sc + td + u - числа одного знака. Справедливий і більш загальний факт: якщо рівняння F(x, y) = 0 визначає пряму або криву, що розбиває координатну площину на дві частини, то точки (a, b) і (c, d), що не лежать на цій лінії, належать одній і тій самій частини площини, якщо F(a, b) і F(c, d) - числа одного знака.

40. Дано дійсні числа a, b, c, d, e, f, g, h. Відомо, що точки (e, f) і (g, h) різні. Відомо також, що точки (a, b) і (c, d) не лежать на прямій l, що проходить через точки (e, f) і (g, h). Пряма l розбиває координатну площину на дві напівплощини. З'ясувати, чи вірно, що точки (a, b) і (c, d) належать одній і тій самій півплощині*). *) У цій задачі, як і в низці наступних задач, треба скористатися тим, що рівнянням прямої, що проходить через дві різні точки (e, f) і (g, h), є рівняння (x − e)(h − f) − (y − f)(g − e) = 0. 54. Дано дійсні числа x1 , x2 , x3 , y1 , y2 , y3 . Чи належить початок координат трикутнику з вершинами (x1 , y1), (x2 , y2), (x3 , y3)? 55. Дано дійсні додатні числа a, b, c, d. З'ясувати, чи можна прямокутник зі сторонами a, b умістити всередині прямокутника зі сторонами c, d так, щоб кожна зі сторін одного прямокутника була паралельною або перпендикулярною кожній стороні другого прямокутника.

41. Дано дійсні додатні числа a, b, c, x, y. З'ясувати, чи пройде цегла з ребрами a, b, c у прямокутний отвір зі сторонами x і y. Просовувати цеглину в отвір дозволяється тільки так, щоб кожне з її ребер було паралельно або перпендикулярно кожній зі сторін отвору.

Найпростіші цикли

42. Дано дійсне число a. Знайти: а) серед чисел 1, 1 + 1/2, 1+1/2+1/3, … перше, більше a; б) таке найменше n, що 1+ 1/2 + …+ 1/n > a.

43. Дано натуральне n, дійсне x. Обчислити: а) sin x + sin2 x + ... + sinn x ; б)sin x + sin x 2 + ... + sin x n ;

44. Дано дійсні числа a, h, натуральне число n. Обчислити $f(a)+2f(a+h)+2f(a+2h)+\ldots+2f(a+(n-1)h)+f(a+nh)$, де $f(x)=(x^2+1)\cos^2 x$.

45. Дано натуральне число n. а) Скільки цифр у числі n. б) Чому дорівнює сума його цифр? в) Знайти першу цифру числа n

46. Дано натуральне n, m. Отримати суму m останніх цифр числа n.

47. Дано натуральне число n. а) З'ясувати, чи входить цифра 3 в запис числа n 2 . б) Поміняти порядок цифр числа n на зворотний. в) Переставити першу й останню цифри числа n. г) Приписати по одиниці на початок і в кінець запису числа n.

48. Алгоритм Евкліда знаходження найбільшого спільного дільника (НСД) невід'ємних цілих чисел ґрунтується на таких властивостях цієї величини. Нехай m і n - одночасно не рівні нулю цілі невід'ємні числа і нехай $m \geq n$. Тоді, якщо $n = 0$, то НОД (n, m) = m, а якщо $n \neq 0$, то для чисел m, n і r, де r - залишок від ділення m на n, виконується рівність НОД (m, n) = НОД (n, r). Наприклад, НОД(15, 6) = НОД(6, 3) = НОД(3, 0) = 3. Дано натуральні числа n, m. а) Використовуючи алгоритм Евкліда, знайти найбільший спільний дільник n і m. б) Знайти найменше спільне кратне n і m.(Як тут може допомогти алгоритм Евкліда?)

49. Дано натуральні числа m і n. Знайти такі натуральні p і q, які не мають спільних дільників, що p/q = m/n.

50. Нехай n – натуральне число та нехай n!!! Означає а) n!!; б) $(1)^{n+1}$ n!!!

Теоретичні відомості

Технологія .NET та, зокрема, мова C#, є одними з основними в IT-галузі. Вони використовуються при розробці самих різних додатків: від невеликих дестопних програм до великих web-порталів і web-сервісів, що обслуговують щодня мільйони користувачів.

C# вже не молода мова і як і вся платформа .NET вже пройшла великий шлях. Перша версія мови вийшла разом із релізом Microsoft Visual Studio .NET у лютому 2002 року. Поточною версією мови є версія C# 11, яка вийшла 8 листопада 2022 року разом із версією .NET 7.

C# є мовою з Сі-подібним синтаксисом і близьким до цього відношення до C++ і Java.

C# є об'єктно-орієнтованим. C# підтримує поліморфізм, наслідування, перевантаження операторів, статичну типізацію. Об'єктно-орієнтований підхід дозволяє вирішувати завдання по побудові крупних, але в теж час гнучких, з можливістю маштабування і розширення додатків. C# продовжує активно розвиватися.

Рольова платформа .NET. Він заснований на C#, але неможливий для технологічних платформ .NET (Windows Forms, WPF, ASP.NET, Xamarin). І, навпаки, коли говорять .NET, часто є у вигляді C#. Однак, хоча ці поняття пов'язані, ототожнювати їх некоректно. Мова C# була створена спеціально для роботи з фреймворком .NET, однак само поняття .NET кілька ширше.

Підтримка кількох мов. Основна платформа — це загальномовне середовище виконання Common Language Runtime (CLR), завдяки якому .NET підтримує кілька мов: наряду з C# це також VB.NET, C++, F#, а також різні діалекти інших мов, пов’язані з .NET, наприклад, Delphi.net. При компіляції код будь-якої з цих мов компілюється в збірку на загальній мові CIL (Common Intermediate Language) - своєму роді асемблера платформи .NET. Тому за певних умов ми можемо створити окремі модулі одного додатка на окремих мовах.

Кросплатформність. .NET є переносимою платформою (з деякими обмеженнями). Наприклад, остання версія платформи на даний момент - .NET 7 підтримується на більшості сучасних ОС Windows, Mac OS, Linux. Використовуючи різні технології на платформі .NET, можна розробляти додатки на мові C# для самих різних платформ - Windows, MacOS, Linux, Android, iOS, Tizen.

Потужна бібліотека класів. .NET представляє єдину бібліотеку класів для підтримуваних мов. І який би додаток ми не збиралися писати на C# - текстовий редактор, чат або складний web-сайт - так або інакше ми запустимо бібліотеку класів .NET.

Різноманіття технологій. Загальномовне середовище виконання CLR і базова бібліотека класів є основою для цілого стека технічної логії, яку розробники можуть використовувати при створенні тех або інших програм. Перш за все, інструменти засновані на цій технології з використанням ADO.NET і Entity Framework Core. Для побудови графічних додатків з багатим насиченим інтерфейсом - ТЕХНОЛОГІЯ WPF і WinUI, для створення більш простих графічних додатків - Windows Forms. Мобільні та настільні кросплатформні платформи – Xamarin/MAUI. Для створення web-сайтів і web-додатків - ASP.NET та інш.

До цього варто додати активно розвивається і набирає популярність Blazor - фреймворк, який працює над .NET і дозволяє створювати web-додатки як на стороні сервера, так і на стороні клієнта. А в майбутньому буде підтримуватися створення мобільних додатків і, можливо, десктоп-додатків.

Продуктивність. Так гласно ряд тестів web-додатків в .NET 7 в ряді категорій силіно випереджають web-додатки, побудовані з допомогою інших технологій. Додатки на .NET 7 в принципі відрізняються високою продуктивністю.

Також слід відзначити таку особливість мови C# і фреймворка .NET, як автоматична збірка мусора. І це каже про те, що нам не прийдеться турбуватись про прибирання мусору і звільнення памяті, на відміну від C++.

Вишеупомянутая общеязыковая среда CLR сама викликає збірник мусора і очищає пам'ять.

З 2014 року Microsoft продовжила розвивати альтернативну платформу - .NET Core, яка вже призначена для різних платформ, і вона повинна була вибрати собі всі можливості старого .NET Framework і додати нову функціональність. Потім Microsoft після цього випустив версію цієї патформи: .NET core 1, .NET core 2, .NET core 3, .NET 5. і т. д.

Керований і не керований код. Часто додаток, створений на C#, називають керованим кодом (керований код). Що це означає? І це не той випадок, який в першу чергу потрібно зробити на новій платформі .NET і який її модернізує. Але є також додатки, наприклад, створені на мові S++, які компілюються не в загальній мові CIL, як C#, VB.NET або F#, а в звичайному машинному коді. В цьому випадку .NET неможливо оновити.

В той же час платформа .NET надає можливості для взаємодії з некерованим кодом..

JIT-компіляція. Як вище писалося, код на C# скомпільований в додатки або збірки з розширеннями exe або dll на мові CIL. Далі при запуску на виконання подібного додатка відбувається JIT-компіляція (Just-In-Time) у машинному коді, який потім виконується. При цьому, оскільки наше додаток може бути більшим і містити кучу інструкцій, у поточний момент у ремені буде компілюватися лише та частина додатків, до якої безпосередньо йде звернення. Якщо ми звернемося до іншої частини коду, то вона буде скомпільована з CIL у машинний код. При цьому вже скомпільована частина додатків зберігається до завершення роботи програми. У підсумку це підвищує продуктивність.

Хід роботи

Для створення програм на c# за допомогою безкоштовного та повнофункціонального середовища visual studio community 2022.

Щоб додати у Visual Studio підтримку проектів для C# і .NET 7, у програмі встановлення серед робочих навантажень можна вибрати лише пункт ASP.NET і розробку web-додатків. Можна вибрати і більше опцій або взагалі всі опції, однак варто враховувати вільний розмір на жорсткому диску - чим більше опцій буде вибрано, відповідно тим більше місця на диску буде зайнято. При встановленні Visual Studio на ваш комп'ютері необхідні всі інструменти для розробки програм, у тому числі фреймворк .NET 7.

Після виконання програми. Она будет простенкой. Спочатку відкриваємо Visual Studio. На стартовому екрані виберем Створити новий проект (Создать новый проект).

На наступному вікні в якості типу проекту виберемо Console App , щоб ми будемо створювати консольну програму на язиці C#.

Щоб ще було знайдено потрібний тип проекту, в полі мов можна вибрати C# , а в полі типу проекту - Console .

Далі на наступному етапі нам буде запропоновано вказати ім'я проекту та каталог, де буде розміщено проект.

В поле Project Name дамо проекту будь-яку назву. Мене звуть HelloApp .

У наступному вікні Visual Studio пропонує нам вибрати версію .NET, яка буде використовуватися для проекту. Вибираємо останню на даний момент версії. -.NET 7.0:

Натисніть кнопку Create (Создать) для створення проекту, а після цього Visual Studio створить і відкриє нам проект.

Справа знаходиться у вікні Solution Explorer, у якому можна побачити структуру нашого проекту. У цьому випадку у нас сгенерована за замовчуванням STRUCTURE: вузол Dependencies - це узел містить збірки dll, які додані в проект за замовчуванням. Ці збірки містять класи бібліотеки .NET,

які використовуватимуть C#. Однак не завжди всі збірки потрібні. Непотрібні потім можна видалити, в той же час, якщо буде потрібно додати яку-небудь потрібну бібліотеку, т. о. саме в цей узел вона буде додана.

Далі йде безпосередньо сам файл коду програми Program.cs , який за замовчуванням відкритий у центральному вікні, і який має всього два рядки:

```
// See https://aka.ms/new-console-template for more information<font></font>
Console.WriteLine("Hello, World!");<font></font>
```

Перший рядок обрамлено символами // і містить коментарі - пояснення до коду.

Другий рядок являє собою код програми: Console.WriteLine("Hello World!");. Цей рядок виводить на консоль рядка "Hello World!".

Незважаючи на те, що програма містить тільки одну рядоку коду, це вже якась програма, яку ми можемо запустити. Запустити проект можна за допомогою клавіші F5 або панелі інструментів, натиснувши на зелену стрілку. І тоді я вперше підійшов до консолі, щоб написати «Hello World!».

Тепер зміню весь цей код на наступний:

```
Console.Write("Введите свое имя: ");<font></font>
var name = Console.ReadLine();       // вводим имя<font></font>
Console.WriteLine($"Привет {name}");    // выводим имя на консоль<font></font>
```

У порівнянні з автоматично згенерованим кодом я вніс кілька змін. Тепер першим рядком виводиться запрошення до введення.

Console.Write("Введите свое имя: ");

Метод Console.Write() виводить на консоль іншу рядоку. У цьому випадку це рядок "Введіть своє ім'я: ".

У другому рядку визначається ім'я рядокової змінної, в який користувач вводить інформацію з консолі:

var name = Console.ReadLine();

Ключове слово var вказує на визначення змінної. В даному випадку змінна називається name. І він присваює результат методу Console.ReadLine() ,

який дозволяє читати з консолі введений рядок. Тобто ми введемо в консоль рядок (точне ім'я) і він виявиться в змінній name.

Потім введене ім'я виводиться на консоль:

Console.WriteLine($"Привет {name}");

Щоб ввести значення змінної назви всередині виведеного на консоль рядку, застосовуються фігурні скобки {}. Тобто при виведенні рядку на консоль виображення {name}буде змінено на значення змінного імені - введене ім'я.

Однак щоб можна було ввести таким чином значення змінних внутрь рядка, перед ним вказують $.

Тепер протестуємо проект, запустивши його на виконання, також нажавши на F5 або зелену стрілочку.

Скомпільований додаток можна знайти в папці проекту в каталозі bin\Debug\net7.0. Він буде мати ім'я проекту та розширення exe. І цей файл можна запускати без Visual Studio, а також переносити його на інші комп'ютери, де встановлено .NET 7.

СТРУКТУРА ПРОГРАМ

Виконання програми. Весь код програми на мові C# вміщається у файли з розширенням .cs. Щоб запустити проект, почніть використовувати Visual Studio (також для .NET CLI) і це легко з кодом C# - файл Program.cs супроводжується:

// См. https://aka.ms/new-console-template для получения дополнительной информации

Console.WriteLine("Привет, мир!");

Саме код файлу Program.cs виконується за умовчанням, якщо ми запустимо проект на виконання. Але за необхідності ми також можемо додати інші файли з кодом C#.

Інструкції. Базовим будівельним блоком програми є інструкції (становлення). Інструкція представляє іншу дію, наприклад, арифметичну операцію, виклик методу, оголошення перемінної та присвоєння її значень. В

кінці кожної інструкції в C# ставиться точка із запятою (;). Цей знак вказує компілятору на кінці інструкції. По-перше, в проекті консолі така історія:

Console.WriteLine("Привет, мир!");

Рядок показує метод виклику Console.WriteLine, який виводить на консольний рядок. У цьому випадку виклик методу є інструкцією і тому завершується точкою із зап'ятою.

Набір інструкцій може об'єднуватися в блок коду. Блок коду полягає в фігурних скобках, а інструкції поміщаються між відкриваючою та закриваючою фігурними скобками. По-перше, код не виконав програму.с у наступному:

{
Console.WriteLine("Привет");
Console.WriteLine("Добро пожаловать в C#");
}

Тут блок коду містить дві інструкції. І при виконанні цього коду консоль виведе два рядки:

Привіт

Добро пожаловать в C#

У цьому блоці коду дві інструкції, які виводять на консоль певний рядок.

Один із блоків є одним із найпоширеніших блоків:

{
Console.WriteLine("Первый блок");
{

Console.WriteLine("Второй блок");

}

}

РЕЄСТРАЦІЯ

C# є реєстрозалежною мовою. Це означає, що в залежності від реєстру символів які-то визначені назви можуть представляти різні класи, методи, змінні і т.д. По-перше, коли ви відкриваєте консоль, це метод WriteLine - це остання кнопка: "WriteL ine". Якщо ми замість "Console.WriteLine" як цей метод обов'язково повинні називатися "WriteLine", а не "writeline" або "WRITELINE" або як-то інакше.

ЛАБОРАТОРНА РОБОТА №2

Тема 2: «Класи. ООП»

Класи і об’єкти. Структури. Типи значень та типии посилань. Простори імен, псевдоніми. Створення бібліотек класів. Модифікатори доступа. Властивості. Перевантаження методів. Статичні елементи (static). Константи, поля і структури для читання. Перевантаження операторів. Спадкування. Перевантаження методів. Перетворення типів. Віртуальні методи і властивості приховування методів. Різниця перевизначення і приховування методів абстрактні класи. Клас System.Object і його методи узагальнені типи обмеження узагальнень. Спадкування узагальнених типів.

Мета роботи: Дослідити та засвоїти основні принципи роботи ООП використовуючи C#.

Завдання

Завдання береться із попередньої лабораторної роботи і виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної, тобто через класи та об’єкти.

Теоретичні відомості

C# є повноцінною об'єктно-орієнтованою мовою. Опис об'єкта є класом, а об'єкт представляє екземпляр цього класу. Найголовніше, це випадок аналогії. У нас у всіх є інше представлення про людину, у якого є ім'я, вік, які-то інші характеристики. До цього некоторого шаблону - цей цей шаблон можна назвати класом. Бетонна конструкція така, що її можна розділити, по-перше, або и люди имеют одно имя, другие - другое имя. І реально наявна людина (фактично примірник даного класу) буде цього класу.

У княжому класі вони були ізольовані. По-перше, тип string, який вказує на символ, фактично використовуваний ссом. Тут, по-перше, клас Console, і цей метод WriteLine()можна використовувати в консолі. Будь ласка, зачекайте пізніше, тому що найчастіше вам буде запропоновано ваш поточний клас.

Вашому класу пропонуються нові типи, які пропонуються пізніше. Клас пропонується ключовим класом:

```
class назва_класа
{
        // вміст класу
}
```

Після уроку іде в мій клас і далі в фігурних дужках іде власне вміст класу. Спочатку виберіть у класі Program.cs Person, тому вам потрібно встановити це:

```
class Person
{

}
```

Крім того, клас може визначати іншу поведінку або інші дії. Для відбору класу рекомендовані методи.

Це описано в класі та методах Person:

```
class Person
{
    public string name = "Undefined"; // ім'я
    public in age; // вік

    public void Print()
    {
        Console.WriteLine($"Імя: {name} Возраст: {age}");
    }
}
```

У класі Person особа має право на name, age, яке зберігає вік людини. У відмінності від змінних, визначених у методах, поля класу можуть мати модифікатори, які вказуються перед полем. Крім того, у цьому випадку вони такі ж, як і в класі Person, позначені з модифікатором public.

Ці класи не ініціалізуються, але їх потрібно лише активувати одним значенням. Для постійних типів це 0.

Також у класі Person пропонується метод Print(). Методи класу мають доступ до його поля, і в цьому випадку звертаємося до поляів класу ім'я та вік для виведення їх значень на консоль. Цей метод включено в клас, також пропонується як модатором public.

Створення об’єкта класу. Можливо, класова схильність більшості людей включати свої об’єкти. У створених об'єктах застосовуються конструктори. Вони Простий шлях створення нового об'єкта класу це виконати ініціалізацію об'єкта. Основний синтаксис наступний:

```
новий конструктор_класса(праметры_конструктора);
```

Спочатку іде оператор new , який видає пам'ять для об'єкта, а після нього іде виклик конструктора .

Конструктор за умовчанням. Якщо вас не визначено ні одного конструктора, простий конструктор починає і не приймає ніяких параметрів.

Тепер створюємо об'єкт класу Person:

```
Особа tom = нова особа(); // створення об'єкта класу Person

// визначення класу Person
class Person
{
    public string name = "Undefined";
    публічний інт вік;

    public void Print()
    {
        Console.WriteLine($"Імя: {name} Зростання: {age}");
    }
}
```

Для створення об'єкта Person is used вираз new Person(). У цьому випадку після виконання даного виявлення в п'яти буде виділено в часток, де будуть зберігатися всі дані об'єкта Person. Змінна tomотримає посилання на створений

об'єкт, і через цю ми можемо використовувати цей об'єкт і звертатися до його функціональності.

Побудова функціональності класу - політик, методів (а також препаратів елементам класу) ласса ставиться точка, а потім елемент класу:

```
object_class
(parameter_method)
```

По-перше, ми звертаємось до політики та методів, які охоплює Особа:

```
Особа tom = нова особа(); // створення об'єкта класу Person

// Получаємо значення поля в змінному
рядку personName = tom.name;
int personAge = tom.age;
Console.WriteLine($"Ім'я: {personName} Возраст {personAge}"); // Ім'я: Undefined
Возраст: 0

// Встановлюємо нові значення поля
tom.name = "Tom";
tom.age = 37;

// працюємо над методом Print
tom.Print(); // Ім'я: Tom Возраст: 37

class Person
{
    public string name = "Undefined";
    публічний інт вік;

    public void Print()
    {
        Console.WriteLine($"Імя: {name} Зростання: {age}");
    }
}
```

Консольний висновок днной програми:

```
Ім'я: Undefined Рік: 0
Ім'я: Tom Рік: 37
```

Загальні класи змішані з іншими частинами. Не обов'язково, щоб один клас використовувався однаково. Ми працюємо над проектом на платформах Visual Studio, які підтримуються .NET CLI, і тільки на Не додавати новий клас у цей проект. Перш за все, ми зробили нову річ, ми запустили Person.cs і в тому, що вони запропонували м наступний код:

```
class Person
{
    public string name = "Undefined";
    public void Print()
    {
```

```
        Console.WriteLine($"Person {name}");
    }
}
```

Клас Person вибирається відповідно до імені та методу Print.

У файлі program.cs він надає програмний клас за замовчуванням: Person.

```
Особа tom = нова особа();
tom.name = "Том";
tom.Print(); // Особа Том
```

Visual Studio передбачає використання існуючих блоків класу. Представлення класу в Visual Studio також включено в проект а:

Оберем пункт Додати -> Новий елемент... (або Додати -> Клас...)

На додаток до нового елемента, це була альна частина з шаблонами елементів у нас вибраний пункт Class. А внизу вікна в поле Name введем назву добавляемого класса - пусть он будет называться Person:

У класі кешу також можна побачити Person, включаючи Person.cs. Зверніть увагу, що проект є новим, у цьому випадку ви повинні використовувати весь код, а не використовувати його в Program.cs.

Конструктори, ініціалізатори та деконструктори. Крім статусу створення, забудовника виділили за умовчанням. Мені теж подобається віддавати перевагу їх будівельнику. По суті, будівельник ініціює об'єкт. По-перше, це в класі, запропонованому будівельником, як написано по умолчанию.

У цьому коді конструктор надає цей метод, який використовується. Цей клас, однак, не включає будь-який елять возвращаемый тип. По-перше, ми пропонуємо клас Person як конструктор:

```
Особа tom = нова особа(); // Створення об'єкта класу Person

tom.Print(); // Ім'я: Tom Возраст: 37

class Person
{
    public string name;
    публічний інт вік;
    public Person()
    {
        Console.WriteLine("Создание объекта Person");
        ім'я = "Том";
        вік = 37;
    }
    public void Print()
```

```
    {
        Console.WriteLine($"Імя: {name} Возраст: {age}");
    }
}
```

Справа в тому, що він пропонується конструктором, який не входить в консоль і ініціалізується класом.

```
public Person()
{
        Console.WriteLine("Создание объекта Person");
        ім'я = "Том";
        вік = 37;
}
```

Конструктори люблять використовувати модифікатори, їх використовують для зміни і менем конструктора. Також у цьому випадку конструктор доступний у класі Person, ределен з модифікатором public.

Рекомендуємо забудовника, не забудьте використати об'єкт.

```
Особа tom = нова особа(); // Создание объекта Person
```

У цьому випадку Person()ми маємо версію, яка призначена для використання класу конструктора, якого у класі тепер немає. Відповідно при його виконанні на консолі буде виведено стр ока "Створення об'єкта Person".

Значна робота, яка, швидше за все, буде запропонована вантажівкам класу е. Перш за все, клас Особа, що відноситься до роботи:

```
Особа tom = нова особа(); // виклик 1-ого конструктора без параметрів
Person bob = new Person("Bob"); //визов 2-ого конструктора з одним параметром
параметр
Person sam = new Person("Sam", 25); // виклик 3-го конструктора з двома
праметрами

tom.Print(); // Ім'я: Невідомо Зростання: 18
bob.Print(); // Ім'я: Bob Возраст: 18
sam.Print(); // Ім'я: Sam Возраст: 25

class Person
{
    public string name;
    публічний інт вік;
    public Person() { name = "Невідомо"; вік = 18; } // 1 конструктор
    public Person(string n) { name = n; вік = 18; } // 2 конструктор
    public Person(string n, int a) { name = n; вік = a; } // 3 конструктор

    public void Print()
    {
        Console.WriteLine($"Імя: {name} Возраст: {age}");
    }
}
```

Тепер у класі визначено три конструктора, кожен з яких має різну кількість проміжків і встановлює значення поля ласса. Тепер ми можемо використовувати ці конструктори для цього проекту.

Консольний висновок днной програми:

```
Ім'я: Невідомо Ріст: 18
Ім'я: Боб Ріст: 18
Ім'я: Сем Ріст: 25
```

Це отмітить, це версія C# 9, і як написати виклик трубора, убрав із нього назву типу:

```
Особа tom = new (); // аналогічний новий Person();
Особа bob = new("Боб"); // аналогічний новий Person("Bob");
Особа sam = new("Сем", 25); // аналогічна нова особа("Сем", 25);
```

Ключове слово this містить посилання на поточний кземпляр/об’єкт сса.

```
Особа sam = new("Сем", 25);
sam.Print(); // Ім'я: Sam Возраст: 25

class Person
{
    public string name;
    публічний інт вік;
    public Person() { name = "Невідомо"; вік = 18; }
    public Person(string name) { this.name = name; вік = 18; }
    public Person(string name, int age)
        {
                this.name = name;
                this.age = вік;
        }
    public void Print() => Console.WriteLine($"Ім'я: {ім'я} Зростання: {вік}");
}
```

Спочатку ви бачите параметри, як і поля класу, встановлено. Необхідно розділити параметри і класи, а класи роботи идет через ключове слово this. Так, у вираженні this.name = ім'я; перша частина - this.nameозначає, що name- це поле поточного класу, а не назва парам етра імені. Якщо б у нас праметри і поля називалися по-разному, використовувати з словом thisбуло б не обов'язково. Не забувайте використовувати цей метод у.

Цепочка виклику конструкторів. Їм спочатку нагородили трьох конструкторів. Три конструктори використовують один тип дизайну - вони використовують значення поля імені та віку. Це не найважливіші. І ми можемо не дирова встановлює кевоє сово Це , передаючи нні значення параметрів:

```
class Person
{
    public string name;
    публічний інт вік;
    public Person() : this("Невідомо") // перший конструктор
    { }
    public Person(string name) : this(name, 18) // другий конструктор
    { }
    public Person(string name, int age) // три конструктори
    {
        this.name = ім'я;
        this.age = вік;
    }
    public void Print() => Console.WriteLine($"Ім'я: {ім'я} Зростання: {вік}");
}
```

У цьому випадку забудовник використовує ту саму конструкцію, хто викликає третій. Відповідно до типу параметрів, які використовує компілятор, ось цей труктор викликається. По-перше, від забудовника:

```
public Person(ім’я рядка) : this(ім’я, 18)
{ }
```

іде звертання до третього конструктору, якому передаються два знання. Перш за все, вам потрібно використовувати три конструктори, які потом код другого конструктора.

Це відмітити, це перший факт, а не конструктор, який вони визначають яких-то інших дій, крім того, як передають третьому конс труктору деякі значення. Поезія в реальності в майбутньому конструкції р, визначив для його парників значення за умовчанням:

```
Особа tom = new();
Особа bob = new("Боб");
Особа sam = new("Сем", 25);

tom.Print(); // Ім'я: Невідомо Зростання: 18
bob.Print(); // Ім'я: Bob Возраст: 18
sam.Print(); // Ім'я: Sam Возраст: 25

class Person
{
    public string name;
    публічний інт вік;
    public Person(string name = "Невідомо", int age = 18)
    {
        this.name = name;
        this.age = вік;
    }
    public void Print() => Console.WriteLine($"Ім'я: {ім'я} Зростання: {вік}");
}
```

Ми шукаємо забудовника і нам не потрібно міняти час доби, то застосовується значення по умолчанию.

Спочатку з версією C# 12 на мові C# була додана підтримка первинних конструкторів (Primary constructors). Pervicnye будівельники готові змінити параметри попередньо визначеного вашого класу і використовувати ці параметри в класі:

```
var tom = нова особа ("Том", 38);
Console.WriteLine(tom);

public class Person(string name, int age)
{
    public Person(string name) : this(name, 18) { }
    public void Print() => Console.WriteLine($"name: {name}, age: {age} ");
}
```

Клас Person визначається конструктором і двома параметрами і - ім'ям і віком. Ці параметри використовуються в методі Print.

Кадр останнього параметра в класі Це приватний стовп, який обмежує параметр. Цей блог можна виділити в класі.

Конструктори Chrome низького класу, швидше за все, запропонують найвищу якість. Ні конструктори, вони з’явилися першими. Це не той випадок, коли будівельникам не потрібно використовувати пошкоджені деталі:

```
public Person(ім’я рядка) : this(ім’я, 18) { }
```

Ініціалізатори об'єктів. Для ініціалізації об’єктів класів спочатку запустіть ініціалізатор . Ініціалізатори представляють передачу у фігурних скобках значень до ступних полів і властивостей об'єкта:

```
Person tom = нова особа { name = "Tom", age = 31 };
// або так
// Особа tom = new() { name = "Tom", age = 31 };
tom.Print(); // Ім'я: Tom Возраст: 31
```

Як ініціалізувати об'єкти можна використовувати ступним полям і властивостями об'єкта в момент створення. Перед використанням ініціалізаторів слід використовувати:

- будемо використовувати ініціалізатор, щоб ви могли використовувати його повністю. Це один із класів полюсів і

об'єктів. По-перше, перше ім'я та вік - це модифікатор загальнодоступних магазинів, поезія одна з найпопулярніших програм.

- Ініціалізатор використовується конструктором, який входить до складу коду, а також в іншому випадку одних і інших властивостей, що встановлюються в конструкторі, замінюються і використовуються з ініціализатора.

Спочатку повинні запускатися ініціалізатори, після поточного класу ставляет другой класс:

```
Person tom = нова особа{ name = "Tom", company = { title = "Microsoft"} };
tom.Print(); // Ім'я: Tom Компанія: Microsoft

class Person
{
    public string name;
    компанія публічного товариства;
    public Person()
    {
        name = "Undefined";
        компанія = нова компанія();
    }
    public void Print() => Console.WriteLine($"Імя: {name} Компанія:
{company.title}");
}

class Company
{
    public string title = "Невідомо";
}
```

Ось опис, який використовує company:

```
компанія = { title = "Microsoft"}
```

Деконструктори позиції об'єкта на окремі частини.

Спочатку помістіть у цей клас Person:

```
class Person
{
    ім'я рядка;
    int вік;
    public Person(string name, int age)
    {
        this.name = name;
        this.age = вік;
    }

    public void Deconstruct(out string personName, out int personAge)
    {
        personName = name;
```

```
        personAge = вік;
    }
}
```

У цьому випадку мені потрібно видалити розкладання з такої особи:

```
Person person = нова особа ("Том", 33);

(ім'я рядка, int вік) = особа;

Console.WriteLine(ім'я); // Том
Console.WriteLine(age); // 33
```

Значення змінним із деконструктора передається по позиції. Ось як ми визначаємо параметр personName як для вашей переменной - name, второе воззываемое значение - переменной age.

Для ваших дизайнерів це не так, це для вас. Ось чому, що було сказано:

```
Person person = нова особа ("Том", 33);

ім'я рядка; int вік;
person.Deconstruct(out name, out age);
```

До 100% використання дизайнером без попереднього проектування Це постійно, лише дизайнер відкриває його. Однак для цієї мети це не так. І замість повертаних значень ми можемо використати прочерк _ . Перш за все, не забудьте його використовувати:

```
Person person = нова особа ("Том", 33);

(_, int age) = особа;

Console.WriteLine(вік); // 33
```

Наскільки перше повертаемое значення - це ім'я користувача, яке не потрібно, у випадку замість змінної перевірки.

ЛАБОРАТОРНА РОБОТА №3

Тема 3: «Обробка винятків»

Конструкція try..catch..finally. Блок catch і фільтри винятків Типи виключень. клас Exception Створення класів винятків Пошук блоку catch при обробці виключень. Генерація виключення і оператор throw.

Мета роботи: Дослідити та засвоїти основні принципи роботи .NET/C# на базі основних елементів та конструкцій обробки виключних ситуацій.

Завдання

Завдання береться із попередньої лабораторної роботи і виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної, тобто через обробку виключних ситуацій, їх класи та об’єкти.

Теоретичні відомості

Конструкция try..catch..finally

Інформація про випуск програм необхідна. Використання C# забезпечує функції цих робіт. Це в C# забезпечує конструкцію try...catch...finally.

```
try
{

}
catch
{

}
finally
{

}
```

Перед ізоляцією блоку try...catch..finally скористайтеся інструкціями в bl оке try . Цей блок не включений, тому після його випуску ачинает виконуватися блок finally. І потім конструкція try..catch..finally завершує свою роботу.

Якщо ж у блоці try вдруге виникає виключення, то звичайний порядок виконання зупиняється і середовище CLR починає шукати блок catch, який може працювати на цій підписці. Це блок, який ловить сам себе, ним користуються, а після його закриття блок виповнюється остаточно.

Якщо потрібний блок catch не знайдено, то при виникненні виключення програми мма аварійно завершує своє виконання.

Розглянемо наступний приклад:

```
int x = 5;
int y = x / 0;
Console.WriteLine($"Результат: {y}");
Console.WriteLine("Конец програми");
```

У цьому випадку дані доступні на 0, що забезпечується генерацією. Перш ніж почати використовувати Visual Studio нижче, ое інформує про виключення.

У цьому випадку мого відео було включено це, яке було представлено за типом System.DivideByZeroException, то є спроба делення на ноль. Нижче наведена детальна інформація.

А в даному випадку це справа без програм.

Не використовуйте вищезгадану програму, следует і використовуйте роботу, щоб включити конструкцію спробувати...зловити...нарешті. Отже, спочатку опишіть роботу:

```
try
{
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
}
catch
{
        Console.WriteLine("Возникло виключення!");
}
finally
{
        Console.WriteLine("Заблокувати остаточно");
}
Console.WriteLine("Конец програми");
```

У цьому випадку ви можете включити блок try, наприклад і намагаємось ділити на нуль, виконання програми зупиниться.

```
int y = x / 0;
```

CLR має перехопити блок і передати цей блок.

Після блоку catch вам потрібно буде остаточно відкрити блок.

Возникло исключение!

Блок остаточно

Підключає програми

Крім того, програму не потрібно включати в програму ль і відповідно не буде виводити результат цього деления, але те перь вона не буде аварійно завершуватися, виключення буде в блоці catch.

Важливо відзначити, що в цій конструкції вам потрібно спробувати блок . Перш ніж завершити блок, ви можете спробувати остаточно натиснути блок:

```
спробуйте
{
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
}
catch
{
        Console.WriteLine("Возникло виключення!");
}
```

І, навпаки, при наявності блоку finally ми можемо опустити опустити блок catch і і не обробляти виключення:

```
спробуйте
{
        int x = 5;
        int y = x / 0;
        Console.WriteLine($"Результат: {y}");
}
finally
{
        Console.WriteLine("Заблокувати остаточно");
}
```

Ряд виняткових ситуацій може бути передбачений розробником. По-перше, поставте в програму цей метод, він запускає рядок, тому інвертує її в число і і вичисляє квадрат цього чесла:

```
Квадрат ("12"); // Квадрат 12: 144
Square("ab"); // !Исключение

void Square(string data)
{
    int x = int.Parse(data);
    Console.WriteLine($"Квадрат числа {x}: {x * x}");
}
```

Цей метод корисно використовувати будь-яким способом, таким як побудова, зберігання нецифрових символів, то програма попаде в помилку. У цьому випадку немає необхідності відкривати блок try..catch, щоб опрацювати можливу помилку. Однак оптимально довести, що ми маємо найкращі результати.

```
Квадрат ("12"); // Квадрат 12: 144
Square("ab"); // Некоректний ввід

void Square(string data)
{
    if (int.TryParse(data, out var x))
    {
        Console.WriteLine($"Квадрат числа {x}: {x * x}");
    }
    else
    {
        Console.WriteLine("Некоректний ввід");
    }
}
```

Спосіб int.TryParse()використовується true, необхідно провести false. Зверніть увагу, що ви зможете почекати до кінця кроку. Крім того, не можна try...catch працювати над цією ексклюзивною ситуацією.

З точки зору блоків try..catch більше дно, ніж застосування умовних

конструкцій. Тому по можливості замість try..catch краще використовувати умовні конструкції на перевірку виняткових ситуацій.

Блок уловлювання та включення фільтра.

Вибір блоку Catch. У роботу входить блок catch, який не входить в комплект.

- `catch`
- `{`
- `  // виконувані інструкції`
- `}`

Обробляє будь-які виключення, які виникли в блоці try. Було продемонстровано перший блок.

- `catch (тип_исключения)`
- `{`
- `  // виконувані інструкції`
- `}`

Обробляє тільки ті виключення, які відповідають типу, якому в скобках після оператора catch.

По-перше, ми повністю працюємо над тим, щоб включити тип DivideByZeroException:

```
спробуйте
{
  int x = 5;
  int y = x / 0;
  Console.WriteLine($"Результат: {y}");
}
catch(DivideByZeroException)
{
  Console.WriteLine("Возникло виключення DivideByZeroException");
}
```

Однак цей блок намагається включити ці типи від DivideByZeroException, то вони не будуть оброблятися.

-
- `пастка`
-

Обробляє тільки ті виключення, які відповідають типу, якому в скобках після оператора catch. Наприклад:

```
спробуйте
{
     int x = 5;
     int y = x / 0;
     Console.WriteLine($"Результат: {y}");
}
catch(DivideByZeroException ex)
{
     Console.WriteLine($"Возникло виключення {ex.Message}");
```

```
}
```

Насправді це аналогічно темі включення і тут використовується змінна. У цьому випадку з'являється тип DivideByZeroException, він викидає якщо у вас є інформація або посилання. Я зможу допомогти тобі з Messageроботою.

Фільтри виключень можна використовувати для відключення від пристрою. Вперше після релізу ловлю ідею релізу коли, після лову в скобках виявляється умова:

```
catch when(условие)
{

}
```

У цьому випадку робота виконується над блоком уловлювання вразі, якщо у вираженні істинно. Наприклад:

```
int x = 1;
int y = 0;

try
{
    int result1 = x / y;
    int result2 = y/x;
}
catch (DivideByZeroException) when (y == 0)
{
    Console.WriteLine("y не повинен бути рівн 0");
}
catch(DivideByZeroException ex)
{
    Console.WriteLine(ex.Message);
}
```

У цьому випадку є зв'язок, тому y=0. Тут два блоки catch, і вони обробляють виключення типу DivideByZeroException, тож є по суті всі виключення, генеровані при вилученні на ноль. Не використовується пост для останнього блоку і == 0, це також стосується того, коли повертає true.

Ситуація:

```
int x = 0;
int y = 1;

try
{
    int result1 = x / y;
    int result2 = y/x;
}
catch (DivideByZeroException) when (y == 0)
{
    Console.WriteLine("y не повинен бути рівн 0");
}
catch(DivideByZeroException ex)
```

```
{
    Console.WriteLine(ex.Message);
}
```

У цьому випадку зв'язок вибрано, оскільки x=0. Останній блок catch - y == 0 тепер повертає false. Будь ласка, використовуйте CLR, щоб знайти блоки в роботі обереться другий блок catch. Це моє завантаження блоку catch, тому його включення не варто.

Типи включень. Виняток класу. Основою для цих типів включення є тип Exception. Цей тип пропонує ряд властивостей, за допомогою яких можна отримати інформацію про реєстрацію.

- *InnerException : яке послужило причиною щего виключення*
- *Повідомлення : хранить повідомлення про виключення, текст помилки*
- *Джерело :*
- *StackTrace : і до появи виключення*
- *TargetSite : використовує метод, включаючи посилання*

По-перше, ми працюємо над цим типом винятків:

```
спробуйте
{
    int x = 5;
    int y = x / 0;
    Console.WriteLine($"Результат: {y}");
}
catch (Exception ex)
{
    Console.WriteLine($"Виключення: {ex.Message}");
    Console.WriteLine($"Метод: {ex.TargetSite}");
    Console.WriteLine($"Трасування стека: {ex.StackTrace}");
}
```

Це те, який тип Виняток використовується як базовий тип для цих включень, наприклад, catch (Exception ex) буде опрацьовувати всі виключення.

Це не особливий вид підписки, які рекомендовано для завантаження. Нарешті, необхідно прочитати:

- *DivideByZeroException : оль*
- *ArgumentOutOfRangeException : згенеровано, аргумент включено в код устимих значень*
- *ArgumentException : згенеровано, неправильно змінено метод параметра*
- *IndexOutOfRangeException : створено, індекс є наймасовішим елементом у колекціях. Це один із діапазонів другої половини року.*
- *InvalidCastException : створено попередніми версіями та типами*
- *NullReferenceException : генерується при спробі звернення до об'єкта з рівнем null (це не призначено)*

Перед необхідністю розрізняти різні типи винятків, включив додаткові блоки catch:

```
static void Main(string[] args)
{
        try
        {
                int[] numbers = new int[4];
                числа[7] = 9; // IndexOutOfRangeException

                int x = 5;
                int y = x / 0; // DivideByZeroException
                Console.WriteLine($"Результат: {y}");
        }
        catch (DivideByZeroException)
        {
                Console.WriteLine("Возникло виключення DivideByZeroException");
        }
        catch (IndexOutOfRangeException ex)
        {
                Console.WriteLine(ex.Message);
        }

        Console.Read();
}
```

У кожному з блоків catch працюють типи виключення IndexOutOfRangeException і DivideByZeroException. Коли в блоці try виникає виключення, то CLR буде шукати потрібний блок catch для обробки виключення. Так, в цьому випадку на рядку числа[7] = 9 відбувається звернення до 7-му елементу масиву. Є лише один пост із масивним 4-елементом, який майже не включений, тобто тип IndexOutOfRangeException. CLR знайде блок catch, який обробляє дно виключення і передасть йому управління.

Далі така ситуація в блоці другого виключення - ділення на нуль. Однак, оскільки після генерації IndexOutOfRangeException управління переходить у відповідність блоку catch, який розподіляється і int y = x / 0 виконується, цей тип DivideByZeroException не є загальним.

Ось опис ситуації:

```
спробуйте
{
    object obj = "ви";
    int num = (int)obj; // System.InvalidCastException
    Console.WriteLine($"Результат: {num}");
}
catch (DivideByZeroException)
{
    Console.WriteLine("Возникло виключення DivideByZeroException");
}
```

```
catch (IndexOutOfRangeException)
{
    Console.WriteLine("Возникло виключення IndexOutOfRangeException");
}
```

У цьому випадку включено тип InvalidCastException, однак відповідного блоку catch для обробки даного виключення немає.

У більшості випадків ми пропонуємо InvalidCastException у блоку catch, однак, це теоретично в коді різні типи створені. Пропонувати різні типи включення блоку уловлювання, це той випадок, який однотипно, не має змісту. У цьому випадку ми повинні вибрати блок catch базового типу Exception:

```
спробуйте
{
    object obj = "ви";
    int num = (int)obj; // System.InvalidCastException
    Console.WriteLine($"Результат: {num}");
}
catch (DivideByZeroException)
{
    Console.WriteLine("Возникло виключення DivideByZeroException");
}
catch (IndexOutOfRangeException)
{
    Console.WriteLine("Возникло виключення IndexOutOfRangeException");
}
catch (Exception ex)
{
    Console.WriteLine($"Виключення: {ex.Message}");
}
```

У цьому блоці catch (Exception ex){}можна працювати без реєстрації винятків DivideByZeroException і IndexOut OfRangeException. Для цього блоки ловлять найважливіші м’ячі, після того, як блоки ловлять бетонні кульки. CLR також використовує блок catch, який є єдиним. Це тип інженерної версії. У цьому прикладі винятки DivideByZeroException і IndexOutOfRangeException включені тільки потім.

Генерація виключення та оператор throw. Повна система генерує включення для вибраних ситуацій, наприклад, при деленні числа на ноль. Но мова C# не включає вручну з оператором throw. Ось як цей оператор працює, щоб ми могли викликати його в процесі виконання.

Перш за все, у цій програмі ви побачите той самий час, щоб, якщо довжина імені менше 2 символів, то виникло вимкнення:

```
try
{
    Console.Write("Введіть ім'я: ");
    рядок? ім'я = Console.ReadLine();
    if (name== null || name.Length < 2)
    {
        throw new Exception("Длина імені менше 2 символів");
    }
    else
    {
        Console.WriteLine($"Ваше ім'я: {name}");
    }
}
catch (Виняток e)
{
    Console.WriteLine($"Ошибка: {e.Message}");
}
```

Після використання оператора throw для блокування включення конструктору необхідно передати повідомлення про помилку. Це найпоширеніший тип винятку, тому використовуйте препарат типу виключення.

Потім у блоці catch сгенерированное нами виключення буде оброблено. Подібним чином ми можемо створити виключення в будь-якому місці програми. Він не включає використання оператора throw, тому що оператор не зобов'язаний включати виключення. У наступному відео оператор throw необхідно використовувати в блоці catch:

```
try
{
    try
    {
        Console.Write("Введіть ім'я: ");
        рядок? ім'я = Console.ReadLine();
        if (name == null || name.Length < 2)
        {
            throw new Exception("Длина імені менше 2 символів");
        }
        else
        {
            Console.WriteLine($"Ваше ім'я: {name}");
        }
    }
    catch (Виняток e)
    {
        Console.WriteLine($"Ошибка: {e.Message}");
        кидати;
    }
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
}
```

У цьому випадку є 2 символи виключення, яке буде оброблено внутрішнім блоком catch. Одна публікація в цьому блоці використовується оператором throw, це виключення буде передано далі зовнішньому блоку, який отримає таке саме включення і виведе це саме повідомлення на консоль.

Створення класів включень. Вони не використовують перераховані вище типи включення, а також найпопулярніші з них. Базовий клас класу винятків включено, відповідно до створення своїх типів ми можемо унаслідувати цей клас.

Зверніть увагу, що в програмі ви отримаєте наступне:

```
try
{
    Person person = new Person { Name = "Tom", Age = 17 };
}
catch (Exception ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
}

class Person
{
    private int age;
    public string Name { get; комплект; } = "";
    public int Age
    {
        get => age;
        set
        {
            if (value < 18)
                throw new Exception("Лицам до 18 реєстрація заборонена");
            інакше
                вік = значення;
        }
    }
}
```

У класі Person для цього є, а у вас є меньше 18, звільнено. Class Exception з'являється в конструкторі в параметрі кешу, однак потім передається в його властивість Message.

Неможливо використовувати виключення класу. По-перше, у ситуації, коли ви працюватимете над запропонованою роботою, однак через ваше виключення вони відокремлені від класу Person. Ось назви спеціальних класів PersonException:

```
class PersonException : Виняток
{
    public PersonException(string message)
```

```
        : base(message) { }
}
```

Не чекайте занадто довго, щоб пропустити клас Person, який був виключений і виключений Menno цей тип і відповідно в новій програмі для виключення:

```
try
{
    Person person = new Person { Name = "Tom", Age = 17 };
}
catch (PersonException ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
}

class Person
{
    private int age;
    public string Name { get; комплект; } = "";
    public int Age
    {
        get => age;
        set
        {
            if (value < 18)
                throw new PersonException("Лицам до 18 реєстрація заборонена");
            інакше
                вік = значення;
        }
    }
}
```

Не обов'язково включати свій клас таким чином. Виняток, не використовуйте будь який інший тип. По-перше, в цьому випадку ви можете побачити тип ArgumentException, який представлений і з'єднання, генероване в результаті передачі аргументу методу в правильному напрямку:

```
class PersonException : ArgumentException
{
        public PersonException(string message)
                : base(message)
        { }
}
```

Кожен виключений тип може визначати, які його властивості. По-перше, в цей день вам буде запропоновано клас хранения встановлюваних значень:

```
class PersonException : ArgumentException
{
        public int Value { get;}
        public PersonException(string message, int val)
                : base(message)
```

```
        {
                Value = val;
        }
}
```

У класі Builder ми встановлюємо це і для роботи виключення ми його можемо отримати:

```
try
{
    Person person = new Person { Name = "Tom", Age = 17 };
}
catch (PersonException ex)
{
    Console.WriteLine($"Ошибка: {ex.Message}");
    Console.WriteLine($"Некоректне значення: {ex.Value}");
}

class Person
{
    private int age;
    public string Name { get; комплект; } = "";
    public int Age
    {
        get => age;
        set
        {
            if (value < 18)
                throw new PersonException("Лицам до 18 реєстрація заборонена",
value);
            інакше
                вік = значення;
        }
    }
}
```

У цьому випадку використовується консоль:

```
Лістинг: Ліцензії 18 реєстрація заборонена
Невірна дата: 17
```

Пошук блока catch при обробці виключень.

Спочатку пояснимо програму:

```
спробуйте
{
    TestClass.Method1();
}
catch (DivideByZeroException ex)
{
    Console.WriteLine($"Catch в Main : {ex.Message}");
}
finally
{
    Console.WriteLine("Блок finally в Main");
}
Console.WriteLine("Конец метода Main");
```

```
class TestClass
{
    public static void Method1()
    {
        try
        {
            Method2();
        }
        catch (IndexOutOfRangeException ex)
        {
            Console.WriteLine($"Catch в Method1 : {ex.Message}");
        }
        finally
        {
            Console.WriteLine("Заблокувати finally в Method1");
        }
        Console.WriteLine("Конец метода Method1");
    }
    static void Method2()
    {
        try
        {
            int x = 8;
            int y = x / 0;
        }
        finally
        {
            Console.WriteLine("Заблокувати finally в Method2");
        }
        Console.WriteLine("Конец метода Method2");
    }
}
```

У цьому випадку ці пристрої показують таку роботу: method Main, метод Method1, який, у свою чергу, викликає метод Method2. І в методі Method2 генерує виключення DivideByZeroException.

ЛАБОРАТОРНА РОБОТА №4

Тема 4: «Делегати, події і лямбда»

Делегати застосування делегатів. Анонімні методи лямбда події. Коваріантність і контраваріантних делегатів. Делегати Action, Predicate і Func.

Мета роботи: Дослідити та засвоїти основні принципи роботи

Завдання

Завдання береться із попередньої лабораторної роботи і виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної, тобто через використання делегатів, подій і лямбд.

Теоретичні відомості

Делегати надають такі об’єкти, але вони також посилаються на методи. Це делегати - вони використовуються в методах і можуть бути використані делегатами.

Визначення делегатів. За призначенням делегата слідує делегат що має повернений тип, назвута параметри. Наприклад:

```
делегат void Message();
```

Повідомлення делегату в кеш введено в тип void і не застосовує жодних параметрів. Це означає, на будь-який метод, який n він визначає кілька параметрів і не змінюється.

Виділимо цю делегацію:

```
місяць повідомлення; // 2. Створюємо змінну делегату
month = Hello; // 3. Натисніть цей постійний метод на
місяць(); // 4. Викликаємо метод

void Hello() => Console.WriteLine("Hello METANIT.COM");

делегат void Message(); // 1. Объявляем делегат
```

Зверніть увагу, що необов’язково включати наступне:

```
делегат void Message(); // 1. Объявляем делегат
```

Використання цього делегування є обов'язковим:

```
місяць повідомлення; // 2. Ми постійно делегуємо
```

Делегати переходять до встановленого методу (у методі Hello).

```
місяць = Привіт; // 3. Натисніть цей постійний метод
```

Нам не потрібно делегувати метод, але включено лише один розділ:

```
місяць(); // 4. Визываем метод
```

Делегатам надається наступний метод. Для цих делегатів немає необхідності використовувати їх у будь-якому методі, наприклад, вони

присуджуються в класі, який зарезервовано для делегата. Це означає, що ви можете використовувати як методи, так і класи та структуру зброї.

```
Повідомлення message1 = Welcome.Print;
Повідомлення message2 = new Hello().Display;

повідомлення1(); // Вітальне
повідомлення2(); // Привет

делегату void Message();

class Welcome
{
    public static void Print() => Console.WriteLine("Ласкаво просимо");
}
class Hello
{
    public void Display() => Console.WriteLine("Привет");
}
```

Пропонуємо делегати для програм вищого рівня (програм верхнього рівня), зокрема: за замовчуванням представлений файл Program.cs, починаючи з версії C# 10, як у прикладі, то, як і інші типи, делегат визначається в кінці коду. Крім того, компанія не пропонує наступне:

```
class Program
{
    делегат void Message(); // 1. Об'являємо делегат
    static void Main()
    {
        Message month; // 2. Створюємо змінну делегату
        month = Hello; // 3. Натисніть цей постійний метод на
        місяць(); // 4. Викликаємо метод

        void Hello() => Console.WriteLine("Hello METANIT.COM");
    }
}
```

Книга в класі:

```
делегат void Message(); // 1. Об'являємо делегат
class Program
{
    static void Main()
    {
        Message month; // 2. Створюємо змінну делегату
        month = Hello; // 3. Натисніть цей постійний метод на
        місяць(); // 4. Викликаємо метод

        void Hello() => Console.WriteLine("Hello METANIT.COM");
    }
}
```

Параметри та результати делегування. Розглядаємо визначення та застосування делегата, який приймає для вимірювання та повертає результати:

```
Операція Operation = Add; // делегат вказує на метод Add
int result = operation(4, 5); // фактичний Add(4, 5)
Console.WriteLine(result); // 9

операція = Множення; // тепер делегат вказує на метод Multiply
result = operation(4, 5); // фактичний Multiply(4, 5)
Console.WriteLine(result); // 20

int Add(int x, int y) => x + y;

int Multiply(int x, int y) => x * y;

делегат int Operation(int x, int y);
```

У цьому прикладі Operation делегує тип int і містить параметр типу int. Ось як делегат відповідає за цей метод, що важливо. Він визначає тип int і друкує параметр типу int. Тут ми показуємо цей метод Додавання та Множення.

Присвоєння посилання на метод. До складу постійної делегації делегата входив і метод. Це назва документа - створення об'єкта делегата за допомогою конструктора, в якому передається потрібний метод:

```
Операція operation1 = Add;
Operation operation2 = new Operation(Add);

int Add(int x, int y) => x + y;

делегат int Operation(int x, int y);
```

Обидва способи рівноправни.

Відповідність методів делегату. Оскільки зрозуміло, що методи делеговані, вони є тут. Не використовуйте нічого, наприклад модифікатори ref, out. Спочатку розмістіть його на цій сторінці:

```
делегат void SomeDel(int a, double b);
```

Це те, що делегати роблять, перш за все, дотримуючись методу:

```
void SomeMethod1(int g, double n) { }
```

Наступні методи не застосовуються:

```
double SomeMethod2(int g, double n) { return g + n; }
void SomeMethod3(double n, int g) { }
void SomeMethod4(ref int g, double n) { }
void SomeMethod5(out int g, double n) { g = 6; }
```

У цьому типі використовується метод SomeMethod2. SomeMethod3 використовується в параметрах. Параметри SomeMethod4 і SomeMethod5 також відокремлені від параметрів нижче і мають модифікатори ref і out.

Добавление методов в делегат. По-перше, делегат використав той самий метод. Насправді делегати можна використовувати в кількох методах, наприклад, вони означають ваш підпис і тип. Методи, якими користуються делегати, наводяться спеціальними фахівцями - фахівцями і викликами. І при виклику делегата всі методи з цього списку послідовно приймаються. І більшу частину часу ми повинні зосереджуватися на цій темі без одиниці, без методу. В операції += використовуються такі методи:

```
Повідомлення повідомлення = Привіт;
повідомлення += Як справи; // тепер вказує повідомлення в методі
message(); // викликаються оба метода - Hello and HowAreYou

void Hello() => Console.WriteLine("Hello");
void HowAreYou() => Console.WriteLine("Як справи?");

делегат void Message();
```

У зразку на пристрої повідомлення делегата підтримується методом - Hello and HowAreYou. Перш ніж з’явиться повідомлення, ви побачите цей метод.

Так що насправді можна досягти створення нового глави делегування і новий метод, і новий створений об'єкт делегата буде прийнято повідомленням.

Перш ніж завершити делегування, будь ласка, зробіть це на один і той же метод кілька разів і в список виклику делегата. Однак цей метод необов’язково використовувати. Відповідно при зверненні до делегата доданий метод буде викликати таке ж ім'я, що називається:

```
Повідомлення повідомлення = Привіт;
повідомлення += Як справи;
повідомлення += Привіт;
повідомлення += Привіт;

повідомлення();
```

Консольний висновок:

```
Привіт
як ти?
Привіт
```

```
привіт
```

Значна робота, щоб ми могли використовувати методи, які будуть делеговані операції ций -= :

```
Повідомлення? повідомлення = Привіт;
повідомлення += Як справи;
повідомлення(); // викликаються всі методи з повідомлення
message -= HowAreYou; // видаляємо метод HowAreYou
if (message != null) message(); // викликаєся метод Hello
```

Обєднання делегатів. Делегації також мають бути закріплені за делегаціями. Наприклад:

```
Повідомлення місяць1 = Привіт;
Повідомлення місяць2 = HowAreYou;
Повідомлення місяць3 = місяць1 + місяць2; // об'єднуємо делегати
month3(); // використання методів у місяцях1 і місяці2

void Hello() => Console.WriteLine("Привіт");
void HowAreYou() => Console.WriteLine("Як справи?");

делегат void Message();
```

У цьому зразку місяць 3 планується включати делегатів місяця 1 і місяця 2. Делегування заплановане на 3 місяці Ці методи призначені для делегатів місяць1 і місяць2. Наразі делегати заплановані на 3 місяць, і ці методи зазвичай використовуються.

Визов делегата. Насамперед делегати визначилися з загальним методом. Вони делеговані в першу чергу параметрам, тому вони також визначають параметри надавались необхідні значення:

```
Місяць повідомлення = Привіт;
місяць();
Операція op = Add;
int n = op(3, 4);
Console.WriteLine(n);

void Hello() => Console.WriteLine("Привіт");
int Add(int x, int y) => x + y;

делегат int Operation(int x, int y);
делегат void Message();
```

Пристрій використовує метод Invoke() :

```
Місяць повідомлення = Привіт;
місяць.Invoke(); // Привіт
Operation op = Add;
int n = op.Invoke(3, 4);
```

```
Console.WriteLine(n); // 7

void Hello() => Console.WriteLine("Hello");
int Add(int x, int y) => x + y;

делегат int Operation(int x, int y);
делегат void Message();
```

Якщо делегати натискають параметри, метод Invoke дозволяє використовувати їх і ці параметри.

Далі враховуємо, що якщо делегат пустий, то є в його списку викликів, але при такому делегаті ми не отримуємо виключення, як, наприклад, у наступному випадку:

```
Повідомлення? місяць;
//місяць(); // ! Ошибка: делегат рівн null

Operation? op = Додати;
op -= Додати; // делегат op пуст
int n = op(3, 4); // !Ошибка: делегат raven null
```

Зверніть увагу, що делегати повинні довести, що вони не є нульовими. Тут можна використовувати метод Invoke і оператор null:

```
Повідомлення? місяць = нуль;
місяць?.Invoke(); // помилки нет, делегат просто не викликається

Operation? op = Додати;
op -= Додати; // делегат op пуст
int? n = op?.Invoke(3, 4); // помилки немає, делегат просто не викликається, а n
= null
```

Якщо делегати цього не роблять, вони нічого не роблять. Наприклад:

```
Операція op = Віднімання;
op += Множення;
op += Додати;
Console.WriteLine(op(7, 2)); // Add(7,2) = 9

int Add(int x, int y) => x + y;
int Subtract(int x, int y) => x - y;
int Multiply(int x, int y) => x * y;
делегат int Operation(int x, int y);
```

ЛАБОРАТОРНА РОБОТА №5

Тема 5: «Інтерфейси»

Визначення інтерфейсів. Застосування інтерфейсів. Явна реалізація інтерфейсів. Реалізація інтерфейсів в базових і похідних класах спадкування інтерфейсів. Інтерфейси в узагальненнях. Копіювання об'єктів. Інтерфейс ICloneable. Сортування об'єктів. Інтерфейс IComparable. Коваріантність і контраваріантних узагальнених інтерфейсів.

Мета роботи: дослідити та засвоїти основні принципи роботи з інтерфейсами.

Завдання

Завдання береться із попередньої лабораторної роботи і виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної, тобто з використанням інтерфейсів.

Теоретичні відомості

Інтерфейси. Прихильність інтерфейсів

Інтерфейс пропонує тип посилань, але не пропонує нічого, який функціонал - набір методів і властивостей без релізації. Оскільки ця функція призначена для реалізації класів і структур, вони перш за все є даними інтерфейсами.

Налаштування інтерфейсу. Інтерфейс заснований на інтерфейсі клавіатури. Як правило, інтерфейси в C# використовуються в назві книги rimer, IComparable, IEnumerable (так звана венгерская нотація), однак це не обов'язкове вимога, більше стиль програмування.

Як ви пропонуєте інтерфейс? У цьому інтерфейсі вони повинні запропонувати повільність:

- *методи;*
- *властивості;*
- *індексатори;*
- *події;*
- *статичні з версії C# 8.0.*

Однак інтерфейси не містять статичних змін. По-перше, він призначений для цієї мети:

```
інтерфейс IMovable
{
        // константа
        const int minSpeed = 0; // мінімальна швидкість
        // статична постійна
        static int maxSpeed = 60; // максимальне значення
        // метод
        void Move(); // рух
        // властиво
        string Name { get; комплект; } // назва

        делегату void MoveHandler(string message); // визначення делегата для
події
        //
        подія події MoveHandler MoveEvent; // подія руху
}
```

У цьому випадку він пропонує інтерфейс IMovable, який недоступний. Будь-які інтерфейси містять окремі компоненти, які описують можливості рухомого об'єкта. Цей інтерфейс не має функціональних можливостей.

Методи та інтерфейси не включають реалізацію, у цьому випадку їх розрізняють абстрактні методи та абстрактні класи. У цьому програмному забезпеченні інтерфейс пропонує метод Move, який не містить жодних його функцій. Він не містить жодних параметрів або параметрів.

Те ж саме називається Ім'я. Не в реальності цієї схильності до інтерфейсу, однак зустрічається релізація.

Модифікатори доступу

Це момент в інтерфейсі: це час – методи та властивості не мають модифікаторів доступу, то фактично по замовчуванню тип public, наприклад інтерфейс – функція реалізації класом. Це стосується також тих, які в класах і структурах за замовчуванням мають модифікатор private. В інтерфейсах, які не включають модифікатор public. Перш за все, є можливість змінити постійні minSpeed і maxSpeed IMovable:

```
Console.WriteLine(IMovable.maxSpeed); // 60
Console.WriteLine(IMovable.minSpeed); // 0
```

Щоправда, це версія C# 8.0, але ви також можете використовувати модифікатори ступа та інтерфейс компонентів:

```
інтерфейс IMovable
{
        public const int minSpeed = 0; // мінімальна швидкість
        private static int maxSpeed = 60; // максимальна швидкість
        public void Move();
        захищений внутрішній рядок Name { get; комплект; } // назва
        публічного делегату void MoveHandler(string message); // визначення
делегата для події
        public event MoveHandler MoveEvent; // подія руху
}
```

Як і класи, інтерфейси між користувачами включають внутрішній контент, це такий інтерфейс, доступний тільки в рамках проекту. Я не можу використовувати модифікатор public для включення застарілих інтерфейсів:

```
публічний інтерфейс IMovable
```

```
{
        void Move();
}
```

Реалізація за замовчуванням. Також доступна версія C# 8.0 і інтерфейси підтримують реалізацію методу і властивості за замовчуванням. Це означає, що ми можемо визначити в інтерфейсах повноцінні методи і властивості, які мають реляцію як в звичайних класах і структурах. Спочатку ми пропонуємо реалізацію методу Move за замовчуванням:

```
інтерфейс IMovable
{
        // метод реалізації за умовчанням
        void Move()
        {
                Console.WriteLine("Walking");
        }
}
```

Це інтерфейс, який включає в себе приватні методи та інформацію (то є з модифікатором приватного), то вони повинні мати релізацію по замовчуванню. Вони також зосереджені на статистичних методах (не приватних):

```
Console.WriteLine(IMovable.MaxSpeed); // 60
IMovable.MaxSpeed = 65;
Console.WriteLine(IMovable.MaxSpeed); // 65
подвійний час = IMovable.GetTime(500, 10);
Console.WriteLine(час); // 50

інтерфейс IMovable
{
    public const int minSpeed = 0; // мінімальна швидкість
    private static int maxSpeed = 60; // максимальна якість
    тобто відстань зі швидкістю швидкість
    static double GetTime(подвійна відстань, подвійна швидкість) => відстань /
швидкість;
    static int MaxSpeed
    {
        get => maxSpeed;
        встановити
        {
            якщо (значення > 0) maxSpeed = значення;
        }
    }
}
```

ЛАБОРАТОРНА РОБОТА №6

Тема 6: «Додаткові можливості ООП в C#»

Методи розширення. Часткові класи і методи. Анонімні типи. Локальні функції. Pattern matching. Деконструктори. Патерни switch. Nullable-типи. Змінні-посилання і повернення посилання. Властивості з модифікатором init. Records.

Мета роботи: дослідити та засвоїти додаткові можливості ООП в C#.

Завдання

Завдання береться із попередньої лабораторної роботи і виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної, тобто додаткові можливості.

Теоретичні відомості

Визначення операторів. Ось методи в класах і структурах, які також можуть бути запропоновані. Спочатку запустіть цей клас Лічильник:

```
class Counter
{
        public int Value { get; комплект; }
}
```

І допустимо, у нас це два об'єкта класу Counter - два лічильника, які ми хочемо порівнювати або складати на основі їх властивостей Value, використовуючи такі операції:

```
Лічильник counter1 = новий лічильник { Value = 23 };
Лічильник counter2 = новий лічильник { Value = 45 };

логічний результат = лічильник1 > лічильник2;
Лічильник c3 = лічильник1 + лічильник2;
```

Ні часу, ні операції, ні затримки. Лічильник об'єктів недоступний. Ці операції повинні бути ізольовані від примітивних типів. Перш за все, за замовчуванням ми складаємо чисельні значення, але як укладати об'єкти комплексних типів - класів і компілятор структур не знає. І це не має відношення до передачі пасажирів операторам.

Перевага оператора входить до уподобання класу, наприклад, найважливішим є оператор, спеціальний метод:

```
public static expressed_tip оператор оператор(параметры)
{ }
```

Цей метод не містить публічних статичних модифікаторів, а також операцій зміни або буде використано для всіх об'єктів дного класу. Це ідея зміни типу. Повернений тип являє собою той тип, об'єкти якого ми хочемо получити. Перш за все, результатом є реєстрація двох об’єктів, які ми очікуємо, щоб бути завершеними. Щоб визначити необхідну кількість слів, введіть bool, й вказує правдиво чи умовний вираз або ложний.

Потім замість назви методу іде ключове слово. Параметри змінено. Бінарні оператори виводять два параметри, один - один параметр. У цьому

випадку ми використовуємо параметри для прогнозування всіх типів - класів або структуру, в якому визначається оператор.

Спочатку змініть оператори класу Counter:

```
class Counter
{
        public int Value { get; комплект; }

        public static Counter operator +(Counter counter1, Counter counter2)
    {
                return new Counter { Value = counter1.Value + counter2.Value };
        }
        public static bool operator >(Counter counter1, Counter counter2)
        {
                return counter1.Value > counter2.Value;
        }
        public static bool operator <(Counter counter1, Counter counter2)
        {
                return counter1.Value < counter2.Value;
        }
}
```

Крім того, у випадку з операційною системою дуже важко отримати код ss Counter, оператор натискає цей клас. А тепер вам потрібно відкрити новий Лічильник, класс також використовується в поточному типі. На додаток до цього, оператор зобов'язаний зробити так, що в новому випадку значення, яке об'єднує значення властивості значення обоих параметрів:

```
public static Counter operator +(Counter counter1, Counter counter2)
{
        return new Counter { Value = counter1.Value + counter2.Value };
}
```

Вони також включені в операцію. Пропонуємо одну з цих операцій, а також не віддаємо перевагу використанню цих операцій. Ті самі оператори використовують значення і залежно від результату порівняння повертають або true, або false.

Температура використовується операторами в програмі:

```
Лічильник counter1 = новий лічильник { Value = 23 };
Лічильник counter2 = новий лічильник { Value = 45 };
логічний результат = лічильник1 > лічильник2;
Console.WriteLine(результат); // false

Лічильник counter3 = counter1 + counter2;
Console.WriteLine(counter3.Value); // 23 + 45 = 68

public static int operator +(Counter counter1, int val)
{
        return counter1.Value + val;
```

```
}
```

Метод поєднує значення властивостей Цінність:

```
Лічильник counter1 = новий лічильник { Value = 23 };
int результат = counter1 + 27; // 50
Console.WriteLine(результат);
```

Зверніть увагу, що оператору пред'являти його не обов'язково. Насправді дуже важливо підкреслити логіку операторів:

- унарні оператори +x, -x, !x, ~x, ++, --, true, false;
- двійкові оператори +, -, *, /, %;
- операції порівняння ==, !=, <, >, <=, >=;
- поразрядні оператори &, |, ^, <<, >>;
- логічні оператори &&, ||.

Крім того, є кілька операторів, які повинні визначати прам і:

- == і !=
- < и >
- <= і >=

Це кількість операторів, які не потребують зміни, першої, =а її рівняння – третього оператора ?:, а також препаратів. Всі прилади передаються в документацію MSDN.

Протиставлення збільшення і зменшення. Зверніть увагу, що це в коді оператора не потрібно вказувати. Вони передаються оператором відповідно до параметрів. Перш за все, у більшості випадків ви хочете вибрати приріст оператора лічильника:

```
public static Counter operator ++(Counter counter1)
{
        counter1.Value += 10;
        лічильник повернення1;
}


}
```

Це використовується в новому елементі, який зберігається в поточному значенні ink. Це нічого не може запропонувати оператору префікса, а в постфіксації я буду працювати в обоих випадках.

```
Лічильник counter1 = new Counter() { Value = 10 };
Лічильник counter2 = counter1++;
Console.WriteLine(counter1.Value); // 20
Console.WriteLine(counter2.Value); // 10

Лічильник counter3 = ++counter1;
Console.WriteLine(counter1.Value); // 30
Console.WriteLine(counter3.Value); // 30
```

Перед операцією постфіксного приросту (лічильник1++) компілятор створив його. Потім поточний об'єкт поміщає значення, отримане від функцій оператора. Фактично операція наразі закрита. Для префікса incremente (++counter1) компілятор запускає новий випуск, отриманий із функцій оператора.

Варіант між істинним і хибним. Також важливо відмітити схильність операторів true і false. Ці оператори пропонують використовувати цей тип в якісних умовах. По-перше, ми пропонуємо маленькі оператори в класі Counter:

```
class Counter
{
        public int Value { get; комплект; }

        public static bool operator true(Counter counter1)
    {
                return counter1.Value != 0;
    }
        public static bool operator false(Counter counter1)
    {
                return counter1.Value == 0;
    }
}
```

Наприклад:

```
Лічильник counter = new Counter() { Value = 0 };
if (лічильник)
        Console.WriteLine(true);
else
        Console.WriteLine(false);
```

Також це відмітити, це мій час виконати іншу операцію типа if (!counter), то нам типу необхідно визначити для типу операцію ! :

```
Лічильник counter = new Counter() { Value = 2 };
if (!counter)
    Console.WriteLine(true);
else
    Console.WriteLine(false);

class Counter
{
```

```
    public int Value { get; комплект; }

    public static bool operator !(Counter counter1)
    {
        return counter1.Value == 0;
    }

    public static bool operator true(Counter counter1)
    {
        return counter1.Value != 0;
    }
    public static bool operator false(Counter counter1)
    {
        return counter1.Value == 0;
    }
}
```

Операція false аналогічна умові. Методи розширення можуть бути додані до нових методів у майбутньому. Цей функціональний статус є дуже коротким, а також той факт, що є в якомусь типі новий метод, але сам тип (клас або структуру) ми не можемо змінити, оскільки у нас немає доступу до вихідного коду. Немає необхідності використовувати стандартну техніку. По-перше, це класи, які пропонує запечатаний модифікатор.

Перш за все, ми можемо лише описати цей тип stringметоду:

```
string s = "Привіт світ";
char c = 'i';
int i = s.CharCount(c);
Console.WriteLine(i);

public static class StringExtension
{
    public static int CharCount(цей рядок str, char c)
    {
        int counter = 0;
        for (int i = 0; i < str.Length; i++)
        {
            if (str[i] == c)
                counter++;
        }
        лічильник повернення;
    }
}
```

Відтепер не використовуйте будь-які статичні класи, які і будуть містити цей метод. Ось ми покажемо цей клас StringExtension. Ми маємо на увазі статичний метод. Суть нашого методу розширення - підрахунок кількості визначених символів у рядку.

Собственно метод розширення - це звичайний статичний метод, який в якості першого параметра завжди приймає таку конструкцію: this имя_типа название_параметра, це в нашому випадку this string str.

Нижче наведено деякі інструкції щодо використання цього методу.

```
int i = s.CharCount(c);
```

Будь ласка, не використовуйте параметр pervy. Значення для інших прапорців передаються в звичайному порядку.

Застосування методів розширення дуже зручно, не для цього надо, цей метод розширення ніколи не буде викликаний, якщо він має те саме, що і метод, визначений у типі.

Крім того, необхідно використовувати ці способи поділу імен. Таким чином, необхідно негайно включити в проект препарату, таким чином, він буде застосовуватися до рядків, і в цьому випадку надо буде підключити пространство імен методу через директиву використання.

Класичні класи та методи. Класи можуть бути частковими. Це те, що нам потрібно знати про пропозицію одного і того ж класу і при компіляції всі ці визначення будуть скомпільовані в одно.

По-перше, PersonBase.cs і PersonAdditional.cs. Крім цього (не відокремлюючи один від одного) вони пропонують наступний і клас:

```
public partial class Person
{
        public void Move()
        {
                Console.WriteLine("Я рухаюся");
        }
}
```

У категорії товарів ми пропонуємо наступний клас:

```
public partial class Person
{
        public void Eat()
        {
                Console.WriteLine("Я їм");
        }
}
```

Крім того, проект також пропонує один і той самий. У класі Person вони використовуються для двох різних методів. Їх також відносять до традиційних. Це те, що вони пропонують як часткові субтитри.

Зверніть увагу, як використовувати методи класу Person:

```
class Program
{
        static void Main(string[] args)
        {
                Person tom = new Person();
                tom.Move();
                tom.Eat();

                Console.ReadKey();
        }
}
```

Основні методи. Класичні заняття повинні використовувати традиційні методи. Будь ласка, зверніться до методу, наведеного нижче, в класичному класі, при реалізації цього методу - в класичній медицині.

Перш за все, це найпопулярніший клас Person. Перші клас:

```
public partial class Person
{
        partial void Read();
        public void DoSomething()
        {
                Read();
        }
}
```

Категорія:

```
public partial class Person
{
        partial void Read()
        {
                Console.WriteLine("Я читаю книгу");
        }
}
```

У першому класі перевага надається методу Read(). Використовуйте цей метод, щоб ви могли використовувати його для його заповнення. Ми не маємо нічого спільного з нашими параметрами, і ми можемо використовувати їх у першому класі.

Цей клас не рекомендований для цього методу Read().

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

```
                Person tom = new Person();
                tom.DoSomething();
        }
}
```

Так вводяться традиційні методи:

- не містять модифікаторів;
- включають лише тип void;
- не містять вихідних параметрів;
- virtual, override, sealed, new тут не містять модифікаторів extern

Вони не використовуються як такі. Ну, по-перше, вони спочатку модифікують методи ікатора public:

```
// виконати реалізацію класів і методів
public partial class Person
{
        public partial void Read();
        public void DoSomething()
        {
                Read();
        }
}

// друга реалізація класу і його методів
public partial class Person
{
        public partial void Read()
        {
                Console.WriteLine("Я читаю книгу");
        }
}
```

ЛАБОРАТОРНА РОБОТА №7

Тема 7: «Колекції»

ArrayList. Список List <T>. Двохзв'язной список LinkedList <T>. Черга Queue <T>. Стек Stack <T>. Словник Dictionary <T, V>. Клас ObservableCollection. Інтерфейси IEnumerable і IEnumerator. Ітератори і оператор yield.

Мета роботи: дослідити та засвоїти основні принципи роботи.

Завдання

В даній роботі необхідно розробити колекцію**,** список, двозв'язний список, чергу, стек та словник з методами по роботі з ними.

Теоретичні відомості

Список List<T>

Hot в мові C# масовий, важко використовувати однотипні об'єкти, взагалі не працювати. Наприклад, масив зберігає фіксовану кількість об'єктів, однак або якщо ми заздалегідь не знаємо, скільки нам потрібно об'єктів. І в цьому випадку також можна презентувати колекції. Цей збірник odin plus заснований на цьому, що не є необхідним для реалізації стандартних структур. Більшість класів включено в System.Collections.Generic.

System.Collections.GenericСписок ключів об'єктів. Список класів типізований за типом і підпорядковується.

Ось деякі з наступних слів:

```
List<string> people = new List<string>();
```

У цьому прикладі List введено як рядок. І ось як воно в цій мові виходить і дуже схоже на рядки.

Не використовуйте цю інформацію для його ініціалізації. В цьому випадку елементи пристрою обробляються будівельником

```
List<string> people = new List<string>() { "Том", "Боб", "Сем" };
```

Важливо також ініціалізувати ці елементи таким чином колекції, наприклад, другого списку:

```
var people = new List<string>() { "Том", "Боб", "Сем" };
вар співробітників = новий список<рядок>(люди);
```

Як про це дізнатися:

```
var people = new List<string>() { "Том", "Боб", "Сем" };
var staff = new List<string>(people){"Mike"};
```

У випадку співробітників є чотири елементи ({ "Tom", "Bob", "Sam", "Mike" }) - вони включені тут список людей і один елемент задається при ініціалізації.

Начиная з версії C# 12 для визначення списков можна використовувати вираз колекцій і в квадратних числах:

```
Список<string> людей = ["Том", "Боб", "Сем"];
Список<string> співробітників = [];// пустой список
```

За видами препаратів можна провести наступні роботи, перш за все:

```
List<Person> people = new List<Person>()
{
    new Person("Tom"),
    new Person("Bob"),
    new Person("Sam")
};

class Person
{
    public string Name { get;}
    public Person(string name) => Name = name;
}
```

Встановлення початкової ємності списку. Ім'я списку конструктора відображається в поточному параметрі списку:

```
Список<string> людей = новий список<string>(16);
```

Також важливо знати, як ним користуватися Capacity, наявний у класі List.

Оскільки вони масивні, їх можна індексувати, тому їх можна швидко використовувати як бажані елементи:

```
var people = new List<string>() { "Том", "Боб", "Сем" };

рядок firstPerson = люди[0]; // отримуємо перший елемент
Console.WriteLine(firstPerson); // Том
люди[0] = "Майк"; // змінюємо перший елемент
Console.WriteLine(people[0]); //Майк
```

Довжина списку. Допоможіть властивості Count можна отримати довжину списку:

```
var people = new List<string>() { "Том", "Боб", "Сем" };
Console.WriteLine(people.Count); // 3
```

Перебор списку. C# можна використовувати для зміни мови на стандартний цикл для кожного :/p>

```
var people = new List<string>() { "Том", "Боб", "Сем" };
```

```
foreach (перемінна особа в людей)
{
    Console.WriteLine(особа);
}
// Вивод програми:
// Том
// Боб
// Сем
```

Також не можна використовувати ці типи транспортних засобів у поєднанні з моїми перебирать списки:

```
var people = new List<string>() { "Том", "Боб", "Сем" };

for (int i = 0; i < people.Count; i++)
{
    Console.WriteLine(people[i]);
}
```

Методи навчання:

- void Add(T item) : додавання нового елемента до елемента;
- void AddRange(IEnumerable<T> collection) : додавання в колекцію та маса;
- int BinarySearch(T item) : бінарний елемент пошуку в списку. Якщо елемент чистий, метод використовує індекс цього елемента таким же чином. З цієї причини патрубок необхідно відсортувати;
- void CopyTo(T[] array) : копіює список у масив масиву;
- void CopyTo(int index, T[] array, int arrayIndex, int count) : копує зі списку починаючи з елементами індексу індексу. Він має дуже велику кількість і відображається у великому масиві з індексом arrayIndex;
- bool Contains(T item) : повертає true, якщо елемент item є в списку;
- void Clear() : зі списку всіх елементів;
- bool Існує (збіг предикату<T>) : ет trueделегату збіг;
- Ви? Find(Predicate<T> match) : повертає перший елемент, який відповідає збігу. Якщо елемент не вказано, він виглядає нульовим
- Ви? FindLast(Predicate<T> match) : 2 у match. Якщо елемент не вказано, він виглядає нульовим;
- List<T> FindAll(Predicate<T> match) : повертає список елементів, які відповідають і відповідають;
- int IndexOf(T item) : повертає індекс першого входу елемента в список;

- int LastIndexOf(T item) : повертає індекс останнього вхідного елемента в список;
- List<T> GetRange(int index, int count) : повертає список елементів, кількість яких рівно лічильник, без індексу з індексом;
- void Insert(int index, T item) : Додати елемент до індексу за індексом. Це індекс у мережі, тому він створений і включений;
- void InsertRange(int index, collection) : вставляє колекцію елементів колекції в поточний список, починаючи з індексу екс. Це індекс у мережі, тому він створений і включений;
- bool Remove(T item) : видаляє елемент item зі списку, і якщо видалення пройшло успішно, то повертається true. Не обов'язково використовувати конкретні елементи;
- void RemoveAt(int index) : видалити елемент за індексом індексу. Це індекс у мережі, тому він створений і включений;
- void RemoveRange(int index, int count) : параметр index задає індекс, з яким потрібно видалити елементи, а пара. Лічильник містить найбільш цінні елементи;
- int RemoveAll((Predicate<T> match)) : видаляє всі елементи, які відповідають делегату match. Створює кількість видалених елементів;
- void Reverse() : змінює порядок елементів;
- void Reverse(int index, int count) : кількість котрих рівно кількість, починаючи з індексу індексу;
- void Sort() : сортування списку;
- void Sort(IComparer<T>? comparer) : сортування списку за допомогою об'єкта порівняння, який передається в власний параметр.

ЛАБОРАТОРНА РОБОТА №8

Тема 8: «Робота з рядками»

Рядки і клас System.String. Операції з рядками. Форматування і інтерполяція рядків. Клас StringBuilder. Регулярні вирази.

Мета роботи: дослідити та засвоїти основні принципи роботи з рядками.

Завдання

Необхідно написати программу за варіантом наступних індивідуальних завдань з використанням матеріалу лекції та додаткових джерел по темі лабораторної і попередніх робіт. Варіант уточнюється з викладачем.

1. Скласти процедуру побудови рядка символів, що є записом заданого дійсного числа в десятковій системі числення; рядок має містити вказану кількість цифр після коми.
2. Скласти процедуру, результатом роботи якої є істинне значення, якщо символ, заданий під час звернення до процедури, - буква, і хибне значення в іншому випадку.
3. Скласти процедуру, результатом роботи якої є символ, заданий при зверненні до процедури, якщо цей символ не є літерою, і відповідна малими (малі) літери в іншому разі.
4. Скласти процедуру "стиснення" вихідної послідовності символів: кожна підпослідовність, що складається з кількох входжень одного й того самого символу, замінюється на текст *x(k)*, де *x* - символ, *k* - рядок, що є записом числа входжень символу *x* у вихідну послідовність.
5. Скласти процедуру, що дає змогу визначити позицію найправішого входження заданого символу у вихідний рядок. Якщо рядок не містить символу, результатом роботи процедури має бути -1.
6. Скласти процедуру, що замінює у вихідному рядку символів усі одиниці нулями і всі нулі одиницями. Заміна повинна виконуватися, починаючи із заданої позиції рядка.

7. Скласти процедуру, у результаті звернення до якої з першого заданого рядка видаляється кожний символ, що належить і другому заданому рядку.
8. Скласти процедуру, що дає змогу визначити позицію першого входження в заданий рядок будь-якого символу з другого заданого рядка. Результатом роботи процедури має бути -1, якщо перший рядок не містить жодного символу, що належить і другому заданому рядку.
9. Вирівнювання рядка полягає в тому, що між його окремими словами додатково вносять пропуски так, щоб довжина рядка дорівнювала заданій довжині (припускають, що потрібна довжина не менша за початкову), а останнє слово рядка зсунулося до правого краю. Скласти процедуру вирівнювання заданого рядка тексту.
10. Під час виведення текстів на екран або друкувальний пристрій часто використовуються табуляційні зупинки - виділені позиції рядка. Наприклад, під час друку таблиць корисно зафіксувати положення стовпців таблиці. Якщо у вихідному тексті зустрічається символ табуляції *tab* (наприклад, символ із кодом 9), це означає, що текст, який іде за символом *tab*, має друкуватися з наступної табуляційної зупинки, а до неї слід видавати пробіли. Скласти процедуру друку тексту із зазначеною інтерпретацією символу *tab* (припустити фіксований набір табуляційних зупинок).
11. Дано символи s1 , ..., sn *). Залишити послідовність s1 , ..., sn без зміни, якщо в неї не входить символ * , інакше кожен символ / , що передує першому входженню символу *: а) замінити на кому; б) видалити з послідовності. *)
12. Дано символи s1 , ..., sn . Якщо послідовність s1 ,..., sn є паліндромом , тобто s1 = sn , s2 = s n– 1 , ..., то залишити її без зміни,інакше отримати послідовність s1 , s2 ,..., s n– 1 , sn , s n– 1 , ..., s2 , s1 . 314. Дано символи s1 , ..., s66 . Якщо послідовність s1 ,..., s66 єтакою, що s1= s34 , s2= s35 , ..., s33= s66 , то залишити її без зміни, інакше одержати послідовність s1 ,

s2 , ..., s66 , s1 , s2 ,..., s66.

13. Дано символи s1 , ..., s80 . Визначити кількість невірних рівностей серед: а) s1 = s41 , s2 = s42 , ... , s40 = s80 ; б) s1 = s80 , s2 = s79 , ... , s40 = s41 .
14. Дано натуральне число n, символи s1 ,... , sn . Будемо розглядати слова, утворені символами, що входять до послідовності s1 , ... , sn (див. задачу 269), вважаючи при цьому, що кількість символів у кожному слові не перевершує 15. а) Знайти якесь слово, що закінчується буквою д (якщо таких слів немає, то повідомити про це). б) Знайти якесь слово, що починається літерою а і закінчується літерою я (якщо таких слів немає, то повідомити про це). в) Видалити з s1 ,..., sn усі слова з непарними порядковими номерами і перевернути всі слова з парними номерами. Наприклад, якщо n = 21 і дана послідовність символів являє собою послідовність у_що_б_будь_то_ні_стало, то має вийти послідовність отч_от_олатс. г) Видалити з s1 ,... , sn усі слова, у яких зустрічається не більше двох різних літер. д) Видалити з s1 , ..., sn усі слова, що закінчуються групою літер яка або дещо.

Теоретичні відомості

Рядки та класи String. У випадку C# рядковий тип є функціональним. Ці роботи наведено в класі System.String . В основному string використовується як псевдонім для класу String. Об'єкти цього класу надають текст для постшвидкісних Unicode. Максимальний розмір об'єкта String може містити в пам'яті 2 ГБ або близько 1 мільйона символів.

Створювати рядки можна, як використовуючи змінний тип string із конструкторів класу String:

```
рядок s1 = "привіт";
рядок s2 = новий рядок ('a', 6); // результатом будет строка "аааааа"
string s3 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' });
string s4 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' }, 1, 3); // orl

Console.WriteLine(s1); // привіт
Console.WriteLine(s2); // ааааааа
Console.WriteLine(s3); // світ
Console.WriteLine(s4); // орл
```

Конструктор String має різне число версій. Так, виклик конструктора

```
новий рядок ('a', 6)
```

6 раз повторить об'єкт із першого параметра, це фактично створює рядок "аааааа".

Цей будівельник друкує величезні символов, а сам використовує їх і рядки

```
string s3 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' });
```

Три з них вперше використані будівельником, були занадто масивними для символов. Параметр використовується для посилання на начальний індекс і для цього знову використовуються три параметри:

```
string s4 = new String(new char[] { 'w', 'o', 'r', 'l', 'd' }, 1, 3); // орл
```

Рядок символів. Крім того, колекція символів не пропонується покажчиком. Ось деякі з цих символів.

```
public char this[int index] {get;}
```

Перший індекс, швидше за все, працюватиме з рядком масивів символів і отримати по індексу будь-який з її символів:

```
рядок повідомлення = "привіт";
// отримуємо символ
char firstChar = message[1]; // символ 'e'
Console.WriteLine(firstChar); //e

Console.WriteLine(message.Length); // довжина строки
```

Використовується для довжини, так як він дуже масивний, тому його можна використовувати.

Перебор рядку. Клас String реалізує інтерфейс IEnumerable, тому його структуру можна змінювати. У циклі foreach є об'єкти char. Також важливо змінити рядок і побудову символів за індексом:

```
рядок повідомлення = "привіт";

for(var i =0; i < message.Length; i++)
{
    Console.WriteLine(message[i]);
}
foreach(var ch in message)
{
    Console.WriteLine(ch);
}
```

В описі препаратів класи використовуються для завантаження і розвантаження, а не по посиланням:

```
рядок message1 = "привіт";
рядок message2 = "привіт";

Console.WriteLine(message1 == message2); // правда
```

Начиная з C# 11 за допомогою трьох пар подвійних лапок можна сформувати багато сильного тексту в контексті інтерполяції:

```
Друк();
PrintValue("привіт");

void Print()
{
    string text = """
              <element attr="content">
                <body>
                </body>
              </element>
              """;
    Console.WriteLine(текст);
```

```
}

void PrintValue(string val)
{
    string text = $"""
              <element attr="content">
                <body>
                {val}
                </body>
              </element>
              """;
    //// або так
    //рядковий текст = $$"""
    // <element attr="content">
    // <body>
    // {{val}}
    // </body>
    // </element>
    // """;
    Console.WriteLine(текст);
}
```

Основні методи рядків. Функціональний клас String розкривається його методом, наприклад:

- Порівняйте : зрівнює дві рядоки з урахуванням текущей культури (локали) користуєтеся.
- CompareOrdinal : зрівнює два рядки.
- Містить : визначає, міститься чи підрядок в рядку.
- Concat : з'єднує рядки.
- Копіювати.
- Формат : форматує текст.
- Індекс.
- Вставити : вставляє в рядку підрядок.
- Join : з'єднує елементи масиву рядок.
- LastIndexOf : знаходиться індекс останнього вхідного символу або підрядоки в окні.
- Замінити.
- Split : розбиває один удар на масивний удар.
- ToLower : перемикання між символом і нижнім регістром.
- ToUpper : змінює символи на верхній регістр.
- Trim : видаляє початкові та кінечні пробіли з рядку.

ЛАБОРАТОРНА РОБОТА №9

Тема 9: «Робота з датами і часом»

Структура DateTime. Форматування дат і часу.

Мета роботи: дослідити та засвоїти основні принципи роботи з датами і часом

Завдання

В даній лабораторній роботі необхідно розробити власний щоденник з вивчення даного предмету. Виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної, тобто використовуючи роботу з датами та часом. Програма повинна мати графічний інтерфейс.

Теоретичні відомості

Структура DateTime. Дати обробляються в основному в .NET на основі структури DateTime. Дата встановлена і дата 00:00:00 1 січня 0001 о 23:59:59 31 грудня та 9999.

Нове посилання DateTime також включає конструктор. Цей конструктор надає таку інформацію:

```
DateTime dateTime = new DateTime();
Console.WriteLine(dateTime); // 01.01.0001 0:00:00
```

Тобто ми отримуємо мінімально можливе значення:

```
Console.WriteLine(DateTime.MinValue);
```

Не використовуйте конкретні дані, не використовуйте їх у побудові:

```
DateTime date1 = новий DateTime(2015, 7, 20); // год - місяць - день
Console.WriteLine(date1); // 20.07.2015 0:00:00
```

Використання:

```
DateTime date1 = новий DateTime(2015, 7, 20, 18, 30, 25); // год - місяць - день
- час - хвилина - секунда
Console.WriteLine(date1); // 20.07.2015 18:30:25
```

Немає необхідності використовувати цю інформацію та дані, тому не використовуйте ряд властивостей DateTime:

```
Console.WriteLine(DateTime.Now);
Console.WriteLine(DateTime.UtcNow);
Console.WriteLine(DateTime.Today);
```

Консольний висновок:

```
20.07.2015 11:43:33
20.07.2015 8:43:33
20.07.2015 0:00:00
```

Свойство DateTime.Now бере поточну дату і час комп'ютера, DateTime.UtcNow- дані і час відносяться до часу по Гринвичу (GMT) і DateTime.Today- всі дані.

Перш ніж працювати над даними, не використовуйте їх, доки ви дотримуєтесь інструкцій. Починається Григоріанський календар. Ні, це приблизно 5 жовтня 1582 року:

```
DateTime someDate = new DateTime(1582, 10, 5);
Console.WriteLine(someDate.DayOfWeek);
```

Консоль має вийти у вівторок, це дата. Однак, оскільки цю історію можна зрозуміти, вона дуже популярна в Юліанському календарі, встановленому в жовтні 1582 року. Тоді після 4 октави (четверг) перейшли до 15 жовтня (п'ятниця) (уже по Григоріанському календарю). Також фактично їх звільнили 10 днів тому. Це з 4 по 15 жовтня.

У більшості випадків справа в тому, що вона включена.

Робота з DateTime. Основні операції зі структурою DateTime пов'язані зі складанням або вичитаними датами. Наприклад, потрібно до дати додати або відняти кілька днів.

Для аплікації використовуються наступні методи:

- Add(TimeSpan value): додає до значення дати TimeSpan.
- AddDays(double value): додає до текущей дати кілька днів.
- AddHours(double value): додає до поточної дати кілька годин.
- AddMinutes(double value): додає до поточної дати кілька хвилин.
- AddMonths(int value): додає до поточної дати кілька місяців.
- AddYears(int value): додає до поточної дати декілька років.

По-перше, ми маємо 3 випадки:

```
DateTime date1 = новий DateTime(2015, 7, 20, 18, 30, 25); // 20.07.2015 18:30:25
Console.WriteLine(date1.AddHours(3)); // 20.07.2015 21:30:25
```

Для вилучення використовується метод Subtract(DateTime date) :

```
DateTime date1 = новий DateTime(2015, 7, 20, 18, 30, 25); // 20.07.2015 18:30:25
DateTime date2 = new DateTime(2015, 7, 20, 15, 30, 25); // 20.07.2015 15:30:25
Console.WriteLine(date1.Subtract(date2)); // 03:00:00
```

Ці дні розділені на три випадки, тому результатом буде «03:00:00».

Метод віднімання не включає голос минулого дня, chásov і так далі. Не це і нічого, тому ми можемо використовувати метод AddDays() і додає негативні значення:

```
// вийде три години
DateTime date1 = new DateTime(2015, 7, 20, 18, 30, 25); // 20.07.2015 18:30:25
Console.WriteLine(date1.AddHours(-3)); // 20.07.2015 15:30:25
```

У чому полягає робота цих методів навчання:

```
DateTime date1 = новий DateTime(2015, 7, 20, 18, 30, 25);
Console.WriteLine(date1.ToLocalTime()); // 20.07.2015 21:30:25
Console.WriteLine(date1.ToUniversalTime()); // 20.07.2015 15:30:25
Console.WriteLine(date1.ToLongDateString()); // 20 липня 2015 р.
Console.WriteLine(date1.ToShortDateString()); // 20.07.2015
Console.WriteLine(date1.ToLongTimeString()); // 18:30:25
Console.WriteLine(date1.ToShortTimeString()); // 18:30
```

ЛАБОРАТОРНА РОБОТА №10

Тема 10: «Додаткові класи і структури .NET»

Відкладене ініціалізація і тип Lazy. Математичні обчислення і клас Math. Перетворення типів і клас Convert. Клас Array і масиви. Span. Індекси і діапазони.

Мета роботи: дослідити та засвоїти основні принципи роботи

Завдання

Завдання береться із першої лабораторної роботи і виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної.

1. Дано натуральне число n, дійсну матрицю розміром n □ 9 . Знайти середнє арифметичне: а) кожного зі стовпців; б) кожного зі стовпців, що мають парні номери.
2. Дано дійсну матрицю розміру n □ m , у якій не всі елементи дорівнюють нулю. Отримати нову матрицю шляхом ділення всіх елементів даної матриці на її найбільший за модулем елемент
3. Дано натуральне число m, цілі числа a1 ,...,..., am і цілочисельна квадратна матриця порядку m. Рядок із номером і матриці назвемо позначеним, якщо ai □ 0 , і невідзначеним у протилежному випадку. а) Потрібно всі елементи, розташовані в позначених рядках матриці, перетворити за правилом: від'ємні елементи замінити на -1, додатні - на 1, а нульові залишити без зміни.б) Підрахувати число від'ємних елементів матриці, розташованих у зазначених рядках.
4. Дано дійсну квадратну матрицю порядку 12. Замінити нулями всі її елементи, розташовані на головній діагоналі та вище неї
5. Дано дійсні числа x1 ,..., x8. Отримати дійсну квадратну матрицю порядку 8:

а) $\begin{bmatrix} x_1 & x_2 & \dots & x_8 \\ x_1^2 & x_2^2 & \dots & x_8^2 \\ \dots & \dots & \dots & \dots \\ x_1^8 & x_2^8 & \dots & x_8^8 \end{bmatrix}$; б) $\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_8 \\ \dots & \dots & \dots & \dots \\ x_1^7 & x_2^7 & \dots & x_8^7 \end{bmatrix}$

6. Дано натуральне число n, дійсну матрицю розміром n □ 9 . Знайти середнє арифметичне: а) кожного зі стовпців; б) кожного зі стовпців, що мають парні номери.

7. Дано дійсну матрицю розміру n □ m , у якій не всі елементи дорівнюють нулю. Отримати нову матрицю шляхом ділення всіх елементів даної матриці на її найбільший за модулем елемент

8. Дано натуральне число m, цілі числа $a1$,...,..., a_m і цілочисельна квадратна матриця порядку m. Рядок із номером iматриці назвемо позначеним, якщо a_i □ 0 , і невідзначеним у протилежному випадку. а) Потрібно всі елементи, розташовані в позначених рядках матриці, перетворити за правилом: від'ємні елементи замінити на -1, додатні - на 1, а нульові залишити без зміни. б) Підрахувати число від'ємних елементів матриці, розташованих у зазначених рядках.

9. Дано дійсну квадратну матрицю порядку 12. Замінити нулями всі її елементи, розташовані на головній діагоналі та вище неї.

10. Дано дійсні числа $x1$,..., $x8$. Отримати дійсну квадратну матрицю порядку 8:

а) $\begin{bmatrix} x_1 & x_2 & \dots & x_8 \\ x_1^2 & x_2^2 & \dots & x_8^2 \\ \dots & \dots & \dots & \dots \\ x_1^8 & x_2^8 & \dots & x_8^8 \end{bmatrix}$; б) $\begin{bmatrix} 1 & 1 & \dots & 1 \\ x_1 & x_2 & \dots & x_8 \\ \dots & \dots & \dots & \dots \\ x_1^7 & x_2^7 & \dots & x_8^7 \end{bmatrix}$

11. Дано дійсну матрицю розміру m □ n . Визначити числа b_1,..., b_m, що дорівнюють відповідно: а) сумам елементів рядків; б) добутків елементів рядків; в) найменшим значенням елементів рядків; г) значенням середніх арифметичних елементів рядків; д) різницям найбільших і найменших значень елементів рядків

12. Усі елементи з найбільшим значенням у даній цілочисельній квадратній матриці порядку 10 замінити нулями.

13. Дано дійсну матрицю розміру 6 □ 9 . Знайти середнє арифметичне

найбільшого та найменшого значень її елементів.

14. Дано дійсну матрицю розміру □□□ *n* . Знайти значення найбільшого за модулем елемента матриці, а також індекси якого- небудь елемента зі знайденим значенням модуля.

15. Дано дійсну матрицю розміру *m* □ *n* . Знайти суму найбільших значень елементів її рядків.

16. У даній дійсній квадратній матриці порядку *n* знайти суму елементів рядка, в якому розташований елемент з найменшим значенням. Передбачається, що такий елемент єдиний.

17. У даній дійсній матриці розміру 6 □ 9 поміняти місцями рядок, що містить елемент із найбільшим значенням, із рядком, що містить елемент із найменшим значенням. Передбачається, що ці елементи єдині.

18. Дано дійсну матрицю розміру *n* □ *m* , усі елементи якої різні. У кожному рядку вибирається елемент з найменшим значенням, потім серед цих чисел вибирається найбільше. Указати індекси елемента зі знайденим значенням.

19. Дана дійсна матриця розміру *n* □ *m* . Отримати послідовність b_1 ,..., b_n , де b_k – це а) найбільше зі значень елементів k-го рядка; б) сума найбільшого і найменшого зі значень елементів k- й рядки; в) число від'ємних елементів у k-му рядку; г) добуток квадратів тих елементів k-го рядка, модулі яких належать відрізку [1, 1.5]

[1, 1.5].

20. Дано цілочисельну квадратну матрицю порядку 8. Знайти найменше зі значень елементів стовпця, який має найбільшу суму модулів елементів. Якщо таких стовпців кілька, то взяти перший із них.

21. Дано натуральне число *n*, цілочисельну квадратну матриця порядку *n*. Отримати b_1 ,..., b_n , де b_i – це

22. а) найменше зі значень елементів, що містяться на початку i-го рядка матриці до елемента, що належить головній діагоналі, включно;

б) значення першого за порядком додатного елемента i-го рядка (якщо таких елементів немає, то прийняти b_i □ 1) в) сума елементів, розташованих за

першим від'ємним елементом в *і-му* рядку (якщо всі елементи рядка ненегативні, то прийняти b_i □ 100); г) сума елементів, що передують останньому від'ємному елементу і-го рядка (якщо всі елементи рядка невід'ємні, то прийняти.

Теоретичні відомості

Відокремлена ініціалізація та тип Lazy. Додаток класів і об'єктів. Однак це віроятність, оскільки вона не відома як предмет використання. Особливо це стосується більших додатків. Наприклад:

```
class Reader
{
    бібліотека бібліотеки = нова бібліотека();
    public void ReadBook()
    {
        library.GetBook();
        Console.WriteLine("Читаємо бумажну книгу");
    }

    public void ReadEbook()
    {
        Console.WriteLine("Читаємо книгу на комп'ютері");
    }
}

class Library
{
    private string[] books = new string[99];

    public void GetBook()
    {
        Console.WriteLine("Видаем книгу читачу");
    }
}
```

Ця бібліотека класів, що надає книги та ресурси у в вигляді масиву. Цей клас є читачом, який використовується для цілей бібліотек. Компанія включає наступний метод: електронний контент і дані читання звичайної книги. Назва бібліотеки не підпорядковується методу класу Library.

Це не ім'я найголовнішого людини, такого як електроніка, по-перше, у випадку:

```
Reader reader = новий Reader();
reader.ReadEbook();
```

При цьому бібліотека в класі не підлягає і буде тільки займати місце пам'яті. Хоча зручності в нем не буде.

Найпопулярніші посилання в .NET містять спеціальний клас Lazy<T> . Використовуйте клас читача наступним чином:

```
class Reader
{
```

```
    Lazy<Library> library = new Lazy<Library>();
    public void ReadBook()
    {
        biblioteka.Value.GetBook();
        Console.WriteLine("Читаємо бумажну книгу");
    }

    public void ReadEbook()
    {
        Console.WriteLine("Читаємо книгу на комп'ютері");
    }
}
```

Класс Бібліотека зберігається негайно. Немає класу, доступного в бібліотеці на відео нижче Lazy<Library>. Вони повинні бути охоплені тією ж бібліографією та методами, не більше - library.Value це обєкт Library.

Про що йдеться в класі Читанка? Розглянемо його застосування:

```
Reader reader = новий Reader();
reader.ReadEbook();
reader.ReadBook();
```

Непредставлений об'єкт Бібліотека зберігається на третьому поверсі в методі reader.ReadBook(), який викликає в свою чергу метод library.Value.GetBook(). Тому буде створено об’єкт бібліотеки, який використовується читачем, до третього рядка. Не обов'язково запускати в програмі метод reader.ReadBook(), щоб отримати бібліотеки тоді взагалі не буде створено. Крім того, Lazy<T> гарантує все, тому для вас це занадто багато.

Навчання математики на уроці математики. Впровадження математичних операцій в класах .NET за допомогою классу Math . Він є статичним, тому всі його методи також є статичними.

Розглянемо основні методи класу математики:

- Abs(double value): включає абсолютне значення аргументу;
- подвійний результат = Math.Abs(-12,4); // 12.4;
- Acos(double value): повертає значення арккосинус. Значення параметра не включає значення -1 або 1;
- подвійний результат = Math.Acos(1); // 0;
- Asin(double value): повертає значення арксинуса. Значення параметра не включає значення -1 або 1;

- Atan(double value): повертає значення арктангенса;
- BigMul(int x, int y): повертає твори x * y у вигляді об'єкта long;
- подвійний результат = Math.BigMul(100, 9340); // 934000;
- Ceiling(double value): повертає найменше ціле число з плаваючою точкою, яке не має більшого значення;
- подвійний результат = Math.Ceiling(2.34); // 3;
- Cos(double d): возвращает косинус угла d;
- Cosh(double d): повертає гіперболічний косинус угла d;
- DivRem(int a, int b, out int result): повертає результат від ділення a/b, а залишок вміщується в параметр результату;
- int результат;
- int div = Math.DivRem(14, 5, вихідний результат);
- //результат = 4
- // div = 2;
- Exp(double d): возвращает основание натурального логарифма, возведенное в степень d;
- Floor(decimal d): возвращает наибольшее целое число, которое не больше d;
- подвійний результат = Math.Floor(2,56); // 2;
- IEEERemainder(double a, double b): повертає остаток від деления a на b;
- подвійний результат = Math.IEEERemainder(26, 4); // 2 = 26-24;
- Log(double d): повертає природний логарифм числа d;
- Log(double a, double newBase): повертає логарифм чисел за заснуванням newBase;
- Log10(double d): повертає десятичний логарифм числа d;
- Max(double a, double b): повертає максимальное число з a і b;
- Min(double a, double b): повертає мінімальне число з a і b;
- Pow(double a, double b): возвращает число a, возведенное в степень b;

- Round(double d): повертає число d, округленное до найближчого цілого числа;
- подвійний результат1 = Math.Round(20,56); // 21;
- подвійний результат2 = Math.Round(20.46); // двадцять;
- Round(double a, int b): повертає число a, використовується округлене до заданої кількості знаків, заданий параметр b;
- подвійний результат1 = Math.Round(20,567, 2); // 20.57;
- double result2 = Math.Round(20,463, 1); //20.5;
- Sign(double value): повертає число 1, якщо значення значення отр іцательное. Значення дорівнює 0 і повертає 0;
- int result1 = Math.Sign(15); // 1;
- int result2 = Math.Sign(-5); //-1;
- Sin(double value): повертає синус угла значення;
- Sinh(double value): використовує гіперболічне значення синуса;
- Sqrt(double value): повертає значення квадратного кореня числа;
- подвійний результат1 = Math.Sqrt(16); // 4;
- Tan(double value): возвращает тангенс угла значення;
- Tanh(double value): використовує гіперболічне значення;
- Truncate(double value): відбирає дробну частину числа значення, повертая лише ціле значення;
- подвійний результат = Math.Truncate(16.89); // 16.

Клас Math пропонує дві константи: Math.Еи Math.PI:

```
подвійний радіус = 50;
подвійна площа = Math.PI * Math.Pow(радіус, 2);
Console.WriteLine($"Площа круга з радіусом {radius} рівна
{Math.Round(area,2)}");
Площадь круга з радіусом 50 рівна 7853,98
```

Перетворення типів і класів Convert.

Методи Parse і TryParse. Більшість примітивних типів використовують два методи. Це методи Parse() і TryParse().

Метод Parse() в параметрі кешу друкує рядок і відображає дані типу. Наприклад:

```
int a = int.Parse("10");
double b = double.Parse("23.56");
decimal c = decimal.Parse("12,45");
byte d = byte.Parse("4");
Console.WriteLine($"a={a} b={b} c={c} d={d}");
```

Цей розбір дробних чисел залежить від параметрів. Не забувайте про відмінності, які є важливими для вас форматом класу NumberFormatInfo та його NumberDecimalSeparator :

за допомогою System.Globalization;

```
IFormatProvider formatter = new NumberFormatInfo { NumberDecimalSeparator = "."
};
double number = double.Parse("23.56", formatter);
Console.WriteLine(число); // 23.56
```

У цьому випадку пристрій налаштовано на це. Однак немає можливості використовувати метод Parse, наприклад, при передачі lfavitnyh символів. І в цьому випадку спосіб запустити метод TryParse() . Якщо необхідно виконати попередню пайку типу, це процес попередньої пайки, то повертає true. Насправді це виглядає неправдою:

```
Console.WriteLine("Переглянути цей текст:");
рядок? вхід = Console.ReadLine();

bool result = int.TryParse(вхід, вихід номер змінної);
if (result == true)
        Console.WriteLine($"Преобразование прошло успешно. Число: {number}");
else
        Console.WriteLine("Преобразование завершилось неудачно");
```

Це не так, але включення Nikakogo не так, просто метод TryParse повертає false, змінне число буде містити значення за замовчуванням.

Клас Convert — це ще один спосіб перетворення значень. Це не включає статичні методи:

- ToBoolean(значення).
- ToByte(значення).
- ToChar(значення).
- ToDateTime(значення).
- ToDecimal(значення).
- ToDouble(значення).

- ToInt16(значення).
- ToInt32(значення).
- ToInt64(значення).
- ToSByte(значення).
- ToSingle(значення).
- ToUInt16(значення).
- ToUInt32(значення).
- ToUInt64(значення).

У параметрі кешу та в цих методах ви можете змінити налаштування. Це примітивні типи, які не існують:

```
int n = Convert.ToInt32("23");
bool b = істина;
double d = Convert.ToDouble(b);
Console.WriteLine($"n={n} d={d}");
```

Ми також рекомендуємо, оскільки він використовується з методом Parse, цей метод не використовується для зображення значення до потрібного типу або для виключення FormatException.

ЛАБОРАТОРНА РОБОТА №11

Тема 11: «Багатопоточність»

Введення в багатопоточність. Клас Thread. Створення потоків. Делегат ThreadStart. Потоки з параметрами і ParameterizedThreadStart. Синхронізація потоків. Монітори. Клас AutoResetEvent. М'ютекси. Семафори. Таймери.

Мета роботи: дослідити та засвоїти основні принципи роботи потоків та здобуття практичних навичок щодо приблизного обчислення визначених інтегралів методом трапецій шляхом створення циклу підпроцесів, що моделюють роботу віртуальних процесорів засобами мови програмування C# при їх асинхронному паралельному виконанні у вигляді циклу створення підпроцесів.

Завдання

Лабораторна робота виконується з використанням матеріалу лекції та додаткових джерел по темі.

Створити блок-схему алгоритму програми обчислення визначеного інтегралу як для кожного віртуального процесора, так і для всієї програми в цілому.

Варіанти для виконання лабораторної роботи

N	Підінтегральна функція	Нижня межа	Верхня межа	N	Підінтегральна функція	Нижня межа	Верхня межа
1.	$\frac{x^2}{\left(\sqrt{16-x^2}\right)^3}$	0	2	16	$\frac{\sqrt{9-x^2}}{x}$	1	2
2.	$\frac{x}{\left(\sqrt{x^2-25}\right)^3}$	7	1	17	$\frac{1}{\sqrt{x^2-9}}$	5	6
3.	$\frac{x^2}{4-x^2}$	0	2	18	$x\left(\sqrt{9-x^2}\right)^3$	0	3
4.	$\frac{x}{\left(9-x^2\right)^3}$	-1	1	19	$\frac{x^2}{9-x^2}$	0	3

5.	$\frac{1}{\left(25-x^2\right)^3}$	0	4	20	$\frac{\sqrt{1+2x}}{x}$	1	2
6.	$x\sqrt{9-x^2}$	0	3	21	$\frac{1}{\left(2-x^2\right)^3}$	0	1
7.	$\frac{1}{1-x^2}$	0	$\frac{e^2-2}{e^2+1}$	22	$\frac{x}{\sqrt{x^2+16}}$	0	3
8.	$\frac{1}{\sqrt{x^2+e^2}}$	0	$\frac{e^2-2}{2}$	23	$\frac{1}{\sqrt{x^2+16^3}}$	0	3
9.	$\frac{\mathbf{x^3}}{\sqrt{\mathbf{1+5x}}}$	0	6	24	$\frac{x}{\sqrt{1+2x}}$	0	4
10	$\sqrt{1+2x}$	0	4	25	$\frac{1}{e+x}$	0	e^2-e
11	$\frac{4}{1+x^2}$	0	1	26	$4*\left(1-x^2\right)^2$	0	1
12	$\frac{x}{\sqrt{x^2-81}}$	9	10	27	$\frac{x}{(e+x)^3}$	0	e^2-e
13	$x\sqrt{x^2-81}^3$	9	10	28	$x\sqrt{x^2-81}$	9	10
14	$\frac{\sqrt{x^2-81}}{x^2}$	9	10	29	3^x	0	3
15	$\cos(x)$	0	π	30	$\frac{\mathbf{x}}{\mathbf{e+x}}$		

Теоретичні відомості

Багатопоточность. Потік класу. Одним із найпопулярніших аспектів програмування є багатопоточність. Це не включає код програми. І перед тим, як багато людей, нам потрібно слідувати інструкціям. Якщо у нас, допустимо, графічне додаток, яке посилає запит на будь-який сервер або розраховує і обробляє величезний файл, без блокування графічних інтерфейсів час виконання завдання.

Функція використання потоків Нижче System.Threading. У визначеному класі, що представляє окремий потік - клас Thread.

Клас Thread визначає ряд методів і властивостей, які дозволяють потоком і отримувати інформацію про нього. Загальний клас:

- Контекст виконання :
- IsAlive : вказує, працює чи потік у поточному моменті
- IsBackground : вказує, є на фоні
- Ім'я : містить ім'я потоку
- ManagedThreadId : повертає потік
- Пріоритет :
  - Найнижчий
  - Нижче нормального
  - нормальний
  - Вище нормального
  - Найвищий

Інформація CLR повинна бути детально проаналізована.

- ThreadState повертає стан потоку - однозначне перерахування ThreadState :
  - Перервано : поток
  - AbortRequested : коли було видано метод Abort, процедура не була виконана.
  - Фон : режим

- Працює : можна використовувати та працювати (не включено)
- Зупинено : поток завершено
- StopRequested : потік отримав запит на остановку
- Підвішений : поток приостановлен
- SuspendRequested : потік отримав запит на приостановку
- Незапущений : потік і не був запущений
- WaitSleepJoin : можна заблокувати в роздільній здатності методів Sleep або Join

У процесі роботи, щоб цей статус був багатогранним, необхідно діяти так. Так, на початку ще до застосування методу його статус має значення Unstarted. Запустив потік, ми змінили його статус на Running. Після використання методу сну статус змінюється WaitSleepJoin.

Крім цього статичного властивості CurrentThread класу Thread можна отримати поточний потік.

Програма на C# містить мінімальну кількість слів - англійських слів, коротше виконується метод Main.

Перш за все, ми використовуємо вищезазначену інформацію:

```
за допомогою System.Threading;

// отримуємо поточний потік
Thread currentThread = Thread.CurrentThread;

//получаємо ім'я потоку
Console.WriteLine($"Ім'я потоку: {currentThread.Name}");
currentThread.Name = "Головний метод";
Console.WriteLine($"Імя потоку: {currentThread.Name}");

Console.WriteLine($"Запущений чи потік: {currentThread.IsAlive}");
Console.WriteLine($"PostId: {currentThread.ManagedThreadId}");
Console.WriteLine($"Пріоритет потоку: {currentThread.Priority}");
Console.WriteLine($"Статус потоку: {currentThread.ThreadState}");
```

У цьому випадку мій перший крок полягає в наступному:

```
Розташування:
Розташування: Основне
розташування методу: Справжній
ідентифікатор розташування: 1
Пріоритетне розташування: Нормальне
розташування Статус: Виконується
```

Так як за замовчуванням ім'я властивості та об'єкти потоку не встановлено, то в першому випадку ця властивість має пустий рядок.

Цей клас Thread пропонує декілька методів застосування потоку. Описи:

- Статичний метод GetDomain доступний вдома.
- Статистичний метод GetDomainID використовує id на початку, нижче яется текущий поток
- Статистичний метод сну встановлюється за системою
- Метод Interrupt натискає кнопки, які включені в підключення WaitSleepJoin.
- Метод Start закриває банк

Перш за все, метод Sleep обумовлений затримками тривалості сну.

```
за допомогою System.Threading;

for(int i = 0; i < 10; i++)
{
    Thread.Sleep(500); // затримка виконання на 500 мілісекунд
    Console.WriteLine(i);
}
```

Создание потоков. Видалити ThreadStart. Мова C# дозволяє запускати та виповнювати в рамках додатків потоків, які будуть виконуватися одночасно.

Створення потоку в основному визначається конструкторами класів Thread :

- Thread(ThreadStart) : у параметрі нижче ви знайдете пункт призначення ThreadStart, який використовується за замовчуванням в останню хвилину.
- Thread(ThreadStart, Int32) : у доповненні до делегату ThreadStart приймає числове значення, яке встановлює розмір стека, що видається під даний потік.
- Thread(ParameterizedThreadStart) : Коли параметр встановлено, необхідно використовувати ParameterizedThreadStart.
- Thread(ParameterizedThreadStart, Int32) : разом із делегатом ParameterizedThreadStart приймає числове значення, яке встановлює розмір стека для даного потоку.

На додаток до цього, будівельник повинен почати використовувати його будівлі, нам надо визначити виконання в потоці дії. У цій ситуації ми проаналізуємо делегування ThreadStart. Ці делегати не дають вам жодних параметрів і не включають таке визначення:

відкритий делегат void ThreadStart();

Щоб під цим делегатом є, нам потрібно визначити метод, який є пустим і не друкує жодних параметрів. Перший вибір:

```
Потік myThread1 = новий Потік (Друк);
Thread myThread2 = new Thread(new ThreadStart(Print));
Thread myThread3 = new Thread(()=>Console.WriteLine("Hello Threads"));

void Print() => Console.WriteLine("Hello Threads");
```

Метод start класу Start Thread:

```
за допомогою System.Threading;

// створюємо новий потік
Thread myThread1 = new Thread(Print);
Thread myThread2 = new Thread(new ThreadStart(Print));
Thread myThread3 = new Thread(()=>Console.WriteLine("Hello Threads"));

myThread1.Start(); // запускаємо потік myThread1
myThread2.Start(); // запускаємо потік myThread2
myThread3.Start(); // запускаємо потік myThread3

void Print() => Console.WriteLine("Hello Threads");
```

Найголовніше - це робити, але їх не можна використовувати поодинці. Наприклад:

```
за допомогою System.Threading;

// створюємо новий потік
Thread myThread = new Thread(Print);
// запускаємо потік myThread
myThread.Start();

// дії, що виконуються в головному потоці
for (int i = 0; i < 5; i++)
{
    Console.WriteLine($"Головний потік : {i}");
    Thread.Sleep(300);
}:
//

оток
:
        {i}"
    )
```

```
    ;
        Thread.Sleep(400);
    }
}
```

Тут новий потік буде здійснювати дії, запропоновані таким чином. З друку це число від 0 до 4 у консолі. Ціна після коробки оцінюється в 400 мілісекунд.

В основному горщику - в основному методі ми використовуємо нові горщики, в методі виконується Print:

```
Потік myThread = новий Потік (Друк);
myThread.Start();
```

Крім того, в головному потоці виробляємо налогічні дії - висновок у консолі було від 0 до 4 секунд за 300 мілісекунд.

Таким чином, в нашій програмі bt workways виконується метод друку. Як то все потоки опрацьовують, програма завершить своє виконання. Останні випуски консолі:

```
Найкращі моменти: 0
Найкращі моменти: 0
Найкращі моменти: 1 Найкращі моменти
: 1 Найкращі моменти й потік:
2
Другий потік: 2
Головний потік: 3
Другий потік: 3
Головний потік: 4 Кількість
горщиків: 4
```

Подібним чином ми можемо створити і запустити і три, і чотири …. їх потрібно видалити з ваших входів.

При обчисленні визначеного інтегралу методом трапеций використовується алгоритм обчислення суми членів h*(Yi+Yi+1)/2, де h – крок квантування, Yi та Yi+1 – значення функції в початку та кінці кроку квантування. Для обчислення визначеного інтегралу пропонується крок в 0.001 від інтервалу інтегрування з моделюванням роботи 10 процесорів в циклі, що завантажуються 100 процесами кожен. Нижче наводиться шаблон програми на мові програмування JAVA для виконання цієї лабораторної роботи.

Приклад виконання лабораторної роботи для інтегрування функції:

$$\int_0^4 \frac{x^2}{\sqrt{1+2x}}$$

```
class Callme {
double sum=0;
public void CS(double member) {
sum=sum+member;
System.out.println(" member : "+member+" sum : "+sum);
    }
}
class Caller implements Runnable {
double member;
Callme target;
  public Caller(Callme target, double member){
this.target=target;

this.member=0.004*(member*member/Math.sqrt(1+2*member)+(member+0.004)*(member+0.
004)/Math.sqrt(1+2*(member+0.004)))/2;

  new Thread(this).start();
 }
public void run() {
synchronized(target) {
target.CS(this.member);
     }
   }
}
class Lr3 {
public static void main(String args[]) {
      Callme target=new Callme();
double x=0;
for(int j=1;j<=100;j++){
for(int i=1;i<=10;i++){
new Caller(target,x);
x=x+0.004;  }          }
 }
}
```

ЛАБОРАТОРНА РОБОТА №12

Тема 12: «Паралельне програмування і бібліотека TPL»

Завдання і клас Task. Робота з класом Task. Клас Parallel. Скасування завдань і паралельних операцій. CancellationToken.

Мета роботи: дослідити та засвоїти основні принципи роботи

Завдання

Лабораторної роботи виконується з використанням матеріалу лекції та додаткових джерел по темі лабораторної. Модифікувати попередню лабораторну з використанням поточної теми та відповідних класів.

Теоретичні відомості

Паралельне програмування та бібліотека TPL. Задачи і клас Задача.

В епоху багаторівневих машин вони можуть бути паралельні один одному. В цьому випадку процесів немає, стандартні мережі, що працюють в .NET, вже виявились недостатньо. Стаття про Freimvork .NET містить паралельну бібліотеку для TPL (Ta sk Parallel Library), її функціональна версія перекладена на System.Threading. Ця бібліотека забезпечує роботу з багатопроцесорними системами. Крім того, вона упрощає роботу по створенню нових потоків. Зазвичай рекомендується використовувати той самий TPL і його класи створення багатопоточних додатків, хоча стандартні засоби та класи по-прежнему находять широке застосування.

У нових бібліотеках TPL є вміст, зокрема думки. Це довготривала операція. У бібліотеці класу .NET доступний спеціальний клас - м Task , який наведено нижче в System.Threading.Tasks . Данный класс описывает отдельную задачу, яка запускається асинхронно не так, як потоки. Тепер також можна ввімкнути годинник у тому ж положенні. Однак у випадку зі смартфоном важливо зазначити, що це не так.

Варіант і кількість елементів, які можна використовувати.

- Остання частина завдання полягає в тому, щоб вибрати і не використовувати метод Start:
- `Task task = new Task(() => Console.WriteLine("Hello Task!"));`
- `task.Start();`

У якості параметра об'єкт завдання приймає делегат Action, тобто ми можемо передати будь-яку дію делегату, наприклад, якщо ви виразите, як ванною, або посилаєтеся на який-небудь метод. Це те, що використовується для видалення даних в консолі "Hello Task!".

Спосіб Start() цілком надійний.

- Ця опція пов’язана зі статичним методом Task.Factory.StartNew() . Цей метод також включає параметр Action, який застосовується. Ось метод нижче:

- `Task task = Task.Factory.StartNew(() => Console.WriteLine("Hello Task!"));`

  Насправді результатом є метод, який використовує дані.
- Статичний метод Task.Run() :
- `Task task = Task.Run(() => Console.WriteLine("Hello Task!"));`

Метод Task.Run(), у якому потрібно використовувати параметри, полягає в тому, щоб делегувати Action - виконуємо дію і повертає об'єкт Task.

Ми рекомендуємо нестандартну програму, яка використовується в таких ситуаціях:

```
Завдання task1 = new Task(() => Console.WriteLine("Завдання1 виконано"));
task1.Start();

Завдання task2 = Task.Factory.StartNew(() => Console.WriteLine("Завдання2
виконано"));

Завдання task3 = Task.Run(() => Console.WriteLine("Завдання3 виконано"));
```

Це так, у коді створюються і запускаються завдання, але при виконанні на консолі ми не можемо нічого побачити. Чому? Через те, що програма збирається зробити ми - потік методу Main, додаток може завершити виконання до того.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Head First C#, Jennifer Greene, Andrew Stellman, O'Reilly Media; Third edition (October 1, 2013), 1100 pg.

2. Head First C#: A Learner's Guide to Real-World Programming with C# and .NET Core 4th Edition, O'Reilly Media; 4th edition (January 20, 2021), 800 pg.

3. Язык программирования C# 5.0 и платформа .NET 4.5 - Эндрю Троелсен., Вильямс, 2015, 1312с.

4. C# 4.0: полное руководство, Герберт Шилдт. Вильямс, 2015, 1056с.

5. CLR via C#. Программирование на платформе Microsoft .NET Framework 4.5 на языке C#, Джеффри Рихтер., Питер, 2018, 896.

6. CLR via C#. C# 6.0 in a Nutshell, Joseph Albahari, Ben Albahari. O'Reilly Media; 6th edition (December 8, 2015), 1136 pg.

7. C# 6.0 in a Nutshell. Essential C# 5.0, Mark Michaelis. Addison-Wesley Professional; 0004- edition (November 27, 2012), 988 pg.

8. Essential C# 5.0. Mark Michaelis. (2012-11-27) Addison Wesley; 4 edition (2012-11-27) (January 1, 1800)

9. More Effective C# (Includes Content Update Program): 50 Specific Ways to Improve Your C# (Effective Software Development Series) 2nd Edition, Addison-Wesley Professional; 2nd edition (August 15, 2017), 304 pg.

10. C# in Depth, Jon Skeet, Third Edition. Manning Publications; Third edition (October 8, 2013), 616 pg.

11. C# 5 Unleashed, Bart De Smet,. Sams; Pap/Psc edition (April 23, 2013), 1079 pg.