МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ МОРСЬКИЙ УНІВЕРСИТЕТ
КАФЕДРА КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

КОНСПЕКТ ЛЕКЦІЙ

з освітнього компонента (навчальної дисципліни)
Об'єктно-орієнтоване програмування
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю
125 Кібербезпека та захист інформації
(частина 1)

Одеса – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ НАЦІОНАЛЬНИЙ МОРСЬКИЙ УНІВЕРСИТЕТ
КАФЕДРА КІБЕРБЕЗПЕКИ ТА ЗАХИСТУ ІНФОРМАЦІЇ

КОНСПЕКТ ЛЕКЦІЙ

з освітнього компонента (навчальної дисципліни)
Об'єктно-орієнтоване програмування
для здобувачів першого (бакалаврського) рівня вищої освіти
за спеціальністю
125 Кібербезпека та захист інформації
(частина 1)

Затверджено
на засіданні кафедри кібербезпеки та захисту інформації
Протокол № 1 від 01.09.2025р.

Одеса – 2025

Конспект лекцій з освітнього компонента (навчальної дисципліни) Об’єктно-орієнтоване програмування для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 125 Кібербезпека та захист інформації (частина 1) / Укл.: К.О. Трифонова. – Одеса, 2025. – 33 с.

Укладач: Трифонова К.О., ст. викл.

ЗМІСТ

ВСТУП

Навчальна дисципліна Об’єктно-орієнтоване програмування є необхідною дисципліною для підвищення рівня теоретичних і прикладних знань, що формують фахівця в галузі інформаційних технологій, які сприяють утворенню у здобувачів поглиблених вмінь та навичок розробки програмного забезпечення із застосуванням сучасних парадигм програмування, тобто підготовці спеціалістів для створення, впровадження та підтримки професійно-орієнтованих комп’ютерних технологій у професійній діяльності.

Дисципліна Об’єктно-орієнтоване програмування відповідає освітньо-професійній програмі, навчальному плану підготовки фахівців першого (бакалаврського) рівня вищої освіти спеціальності 125 Кібербезпека та захист інформації і є складовою циклу дисциплін професійної підготовки обов’язкової частини навчального плану.

Дисципліну викладають впродовж третього семестру першого (бакалаврського) рівня. У процесі навчання передбачено лекції, практичні заняття.

Згідно навчального плану передбачено підсумковий контроль у вигляді екзамену.

Робочу програму навчальної дисципліни укладено згідно з вимогами кредитно-модульної системи організації навчального процесу. Програма визначає обсяги компетентностей, які повинен опанувати здобувач відповідно до освітньо-професійної програми, алгоритму вивчення навчального матеріалу дисципліни Об’єктно-орієнтоване програмування, необхідне методичне забезпечення, складові та технологію оцінювання навчальних досягнень.

Предмет навчальної дисципліни Об’єктно-орієнтоване програмування – сучасні технології розробки програмного забезпечення.

Мета вивчення навчальної дисципліни Об’єктно-орієнтоване програмування – забезпечити формування поглибленої системи теоретичних і практичних знань у галузі розробки програмного забезпечення для реалізації здатності використання інформаційно-комунікаційних технологій, сучасних методів і моделей інформаційної безпеки та/або кібербезпеки.

Завдання вивчення дисципліни:

– формувати теоретичні основи та практичні навички застосування методів вилучення, аналізу, обробки та синтезу інформації для формування технічного завдання при розв’язанні задач в галузі інформаційних технологій;

– формувати теоретичні основи та практичні навички сучасних парадигм програмування та способів їх вибору з позицій зручності та якості застосування для реалізації методів та алгоритмів розв’язання задач в галузі інформаційних технологій;

– поглибити практичні навички кодування програмного забезпечення для розв’язання задач в галузі інформаційних технологій;

– поглибити практичні навички налагодження програмного забезпечення для розв’язання задач в галузі інформаційних технологій.

Стратегічні цілі дисципліни – націлити майбутніх фахівців на застосування отриманих знань у подальшій професійній підготовці та їх наступній практичній діяльності.

ЛЕКЦІЯ №1. ОРГАНІЗАЦІЯ КЛАСУ

Цей розділ служить введенням у класи. Клас становить основу мови C#, оскільки він визначає характер об'єкта. Крім того, клас служить основою об'єктно-орієнтованого програмування (ООП). У межах класу визначаються дані та код. А оскільки класи та об'єкти належать до основних елементів C#, то для їх розгляду потрібен не один розділ. У цьому розділі розгляд класів та об'єктів починається з їх головних особливостей.

1. Основні положення про класи

Клас представляє собою шаблон, за яким визначається форма об'єкта. У ньому вказуються дані та код, який буде оперувати цими даними. В C# використовується специфікація класу для побудови об'єктів, які є екземплярами класу.

1.1. Загальна форма визначення класу

При визначенні класу оголошуються дані, які він містить, а також код, що оперує цими даними.

Взагалі, дані містяться у членах даних, визначених класом, а код – у функціях-членах. До членів даних, відносяться змінні екземпляра та статичні змінні, а до функцій-членів – методи, конструктори, деструктори, індексатори, події, оператори та властивості.

1.2. Визначення класу

Розглянемо перший варіант класу Building. У ньому визначено: Floors, Area та Occupants. У класі Building взагалі немає методів. Це означає, що зараз цей клас складається лише з даних.

2. Створення об'єктів

Оголошення об'єкта типу Building виконує три функції. По-перше, оголошується змінна house, що відноситься до типу класу Building. По-друге, створюється конкретна, фізична копія об'єкта. І нарешті, змінній house надається посилання на даний об'єкт.

3. Змінні типу посилань та присвоєння

Коли ж одна змінна посилання на об'єкт присвоюється інший, то ситуація дещо ускладнюється, оскільки таке присвоєння призводить до того, що змінна, що знаходиться в лівій частині оператора присвоєння, посилається на той самий об'єкт, на який посилається змінна, що знаходиться у правій частині цього оператора. Сам же об'єкт не копіюється.

4. Методи

Як пояснювалося вище, змінні екземпляра та методи є двома основними складовими класів. Методи представляють собою підпрограмами, які маніпулюють даними, визначеними в класі, а в багатьох випадках вони надають доступ до цих даних.

Метод складається з одного або кількох операторів. У грамотно написаному коді C# кожен метод виконує лише одну функцію. Кожен метод має своє ім'я, за яким він викликається.

4.1. Додавання методу до класу Building

Методи класу, як правило, маніпулюють даними класу та надають доступ до них. Площу на одну людину найкраще обчислювати в самому класі Building, просто тому, що так легше зрозуміти сам характер обчислення. Крім того, можна позбавити всі програми, що користуються класом Building, від необхідності виконувати це обчислення самостійно. Додавання до класу Building методу, сприяє поліпшенню його об'єктно-орієнтованої структури.

Для того щоб додати метод до класу Building, достатньо вказати його в області об'яв у даному класі.

4.2. Повернення з методу

Загалом, повернення з методу може статися за двох умов. По-перше, коли зустрічається фігурна дужка, що закриває тіло методу. І по-друге, коли виконується оператор return. Є дві форми оператора return: одна – для методів типу void, тих методів, які не повертають значення, а інша – для методів, що повертають конкретні значення.

Для негайного завершення методу типу void достатньо скористатися формою оператора return.

4.3. Повернення значення

Повернені значення використовуються в програмуванні з найрізноманітнішими цілями.

Для повернення значення з методу до частини програми, що виконує виклик, слугує форма оператора return значення.

4.4. Використання параметрів

При виклику методу йому можна передати одне чи кілька значень. Значення, що передається методу, називається аргументом. А змінна, яка отримує аргумент, називається формальним параметром, чи просто параметром. Параметри оголошуються у дужках після імені методу. Синтаксис оголошення параметрів такий самий, як і у змінних. А областю дії параметрів є тіло методу.

4.5. Додавання параметризованого методу до класу Building

За допомогою параметризованого методу можна доповнити клас Building новим засобом, що дозволяє обчислювати максимальну кількість мешканців в будівлі, виходячи з певної величини мінімальної площі на одну особу. Цим новим засобом є метод MaxOccupant().

Коли викликається метод MaxOccupant(), його параметр minArea приймає величину необхідної мінімальної площі на одну людину. На цю величину ділиться загальна площа будівлі при виконанні методу, після чого він повертає результат.

4.6. Виключення недоступного коду

При створенні методів слід виключити ситуацію, за якої частина коду не може бути виконана ні при яких обставинах. Такий код називається недоступним і вважається в C# неправильним. Якщо створити метод, що містить недоступний код, компілятор видасть попереджувальне повідомлення відповідного змісту.

5. Конструктори

Конструктор ініціалізує об'єкт під час його створення. У конструктора таке ж ім'я, як і в його класу, а з точки зору синтаксису він подібний до методу. Але у конструкторів немає типу, що повертається, вказаного явно.

Як правило, конструктор використовується для завдання початкових значень змінних екземпляра, визначених у класі, або ж для виконання будь-яких інших настановних процедур, які потрібні для створення повністю сформованого об'єкта.

5.1. Параметризовані конструктори

Найчастіше конструктор повинен приймати один або кілька параметрів. У конструктор параметри вводяться таким же чином, як і в метод. Для цього достатньо оголосити їх у дужках після імені конструктора.

5.2. Додавання конструктора до класу Building

Клас Building можна вдосконалити, додавши до нього конструктор, який автоматично ініціалізує поля Floors, Area та Occupants при створенні об'єкта.

6. Детальний розгляд оператора new

Оператор new може бути використаний для створення об'єкта, що належить до класу будь-якого типу.

Оперативна пам'ять не нескінченна, і тому цілком можливо, що оператору new не вдасться розподілити пам'ять для об'єкта через нестачу оперативної пам'яті, що є в наявності. У цьому випадку виникає виняткова ситуація під час виконання.

6.1. Застосування оператора new разом із типами значень

Загалом звернення до оператора new для будь-якого типу значень призводить до виклику конструктора, який використовується за замовчуванням для цього типу. Але в цьому випадку пам'ять динамічно не розподіляється. Відверто кажучи, у програмуванні зазвичай не прийнято користуватися оператором new разом із типами значень.

7. "Збір сміття" та застосування деструкторів

Система "збір сміття" в C# звільняє пам'ять від зайвих об'єктів автоматично, діючи непомітно і без будь-якого втручання з боку програміста. "Збір сміття" відбувається наступним чином. Якщо посилання на об'єкт відсутні, то такий об'єкт вважається непотрібним, і пам'ять, яку він займає, в результаті звільняється і накопичується. Ця утилізована пам'ять може бути розподілена потім для інших об'єктів.

7.1. Деструктори

У мові C# є можливість визначити метод, який буде викликатись безпосередньо перед остаточним знищенням об'єкта системою "збір сміття". Такий метод називається деструктором і може використовуватися у низці особливих випадків, щоб гарантувати чітке закінчення терміну дії об'єкта. Для того щоб додати деструктор до класу, достатньо включити його до класу в якості члена. Він викликається щоразу, коли передбачається утилізувати об'єкт його класу.

7.2. Ключове слово this

Розглянемо ключове слово this. Коли метод викликається, йому автоматично передається посилання на об'єкт, що виконує виклик, той об'єкт, для якого викликається даний метод. Це посилання позначається ключовим словом this. Отже, ключове слово this означає саме той об'єкт, за посиланням на який діє метод, що викликається.

ЛЕКЦІЯ №2. ІНКАПСУЛЯЦІЯ КЛАСУ

1. Управління доступом до членів класу
  1.1. Модифікатори доступу
  1.2. Організація закритого та відкритого доступу
  1.3. Практичний приклад організації управління доступом
2. Передача об'єктів методам за посиланням
3. Способи передачі аргументів методу
4. Використання модифікаторів параметрів ref та out
  4.1. Використання модифікатора параметра ref
  4.2. Використання модифікатора параметра out
  4.3. Використання модифікаторів ref та out для посилань на об'єкти
5. Використання змінної кількості аргументів
6. Повернення об'єктів із методів
7. Повернення масиву з методу

В даному розділі продовжується розгляд класів та методів. Він починається з пояснення механізму управління доступом до членів класу. А потім обговорюються такі питання, як передача та повернення об'єктів.

1. Управління доступом до членів класу

Підтримка властивостей інкапсуляції в класі дає дві основні переваги. По-перше, клас пов'язує дані із кодом. Ця перевага використовувалась у попередніх прикладах програм. І по-друге, клас надає засоби для управління доступом до його членів. Саме ця друга переважна особливість і буде розглянута нижче.

У мові C#, по суті, є два типи членів класу: відкриті і закриті, хоча насправді справа трохи складніша. Доступ до відкритого члена вільно здійснюється з коду, визначеного за межами класу. Саме цей тип члена класу використовувався в прикладах програм, що розглядалися досі. А

закритий член класу доступний лише методам, визначеним у самому класі. За допомогою закритих членів й організується управління доступом.

Обмеження доступу до членів класу є основним етапом об'єктно-орієнтованого програмування, оскільки дозволяє виключити неправильне використання об'єкта. Дозволяючи доступ до закритих даних лише за допомогою строго певного ряду методів, можна попередити присвоєння невірних значень цим даним, виконуючи, наприклад, перевірку діапазону представлення чисел. Для закритого члена класу не можна встановити значення безпосередньо в коді за межами класу. Але в той же час можна повністю управляти тим, як і коли данні використовуються в об'єкті. Отже, правильно реалізований клас утворює деякий "чорний ящик", яким можна користуватися, але внутрішній механізм його дії закритий для втручання ззовні.

1.1. Модифікатори доступу

Управління доступом у мові C# організується за допомогою чотирьох модифікаторів доступу: public, private, protected та internal. У цьому розділі основна увага надається модифікаторам доступу public і private.

Коли член класу позначається специфікатором public, він стає доступним з будь-якого іншого коду в програмі, включаючи і методи, визначені в інших класах. Коли ж член класу позначається специфікатором private, він може бути доступним лише іншим членам цього класу.

1.2. Організація закритого та відкритого доступу

Правильна організація закритого та відкритого доступу – запорука успіху в об'єктно-орієнтованому програмуванні. І хоча для цього не існує твердо встановлених правил, нижче перераховано низку загальних принципів, які можуть служити в якості керівництва до дії.

1. Члени, які використовуються лише у класі, повинні бути закритими.
2. Дані екземпляра, які не виходять за певні межі значень, повинні бути закритими, а при організації доступу до них за допомогою відкритих методів слід виконати перевірку діапазону представлення чисел.
3. Якщо зміна члена призводить до наслідків, що розповсюджуються за межі дії самого члена, впливає на інші аспекти об'єкта, то цей член повинен бути закритим, а доступ до нього – контрольованим.
4. Члени, які можуть завдати шкоди об'єкту, якщо вони використовуються неправильно, повинні бути закритими. Доступ до цих членів слід організувати за допомогою відкритих методів, що унеможливлюють неправильне їх використання.
5. Методи, які отримують та встановлюють значення закритих даних, повинні бути відкритими.
6. Змінні екземпляра допускається робити відкритими лише в тому випадку, якщо немає жодних підстав для того, щоб вони були закритими.

Зрозуміло, існує чимало ситуацій, на які наведені вище принципи не поширюються, а в окремих випадках один чи кілька цих принципів можуть взагалі порушуватися. Але загалом, дотримуючись цих правил, можна створювати об'єкти, стійкі до спроб неправильного їх використання.

1.3. Практичний приклад організації управління доступом

Стек служить класичним прикладом об'єктно-орієнтованого програмування тому, що він поєднує у собі засоби зберігання інформації з методами доступу до неї. Для реалізації такого поєднання відмінно підходить клас, у якому члени, які забезпечують зберігання інформації у стеку, мають бути закритими, а методи доступу до них – відкритими. Завдяки інкапсуляції базових засобів зберігання інформації дотримується певний порядок доступу до окремих елементів стеку з коду, в якому він використовується.

Для стека визначено дві основні операції: помістити дані у стек та витягти їх звідти. Перша операція поміщає значення на вершину стека, а друга – отримує значення з вершини стека. Отже, операція вилучення є безповоротною: як тільки значення витягується зі стека, воно видаляється і вже недоступне у стеку.

2. Передача об'єктів методам за посиланням

В методах можна використовувати параметри типу посилань, що не тільки правильно, але і дуже поширене в ООП. Подібним чином об'єкти можуть передаватися методам за посиланням.

3. Способи передачі аргументів методу

Розглянемо два способи передачі аргументів методу.

Першим способом є виклик за значенням. В даному випадку значення аргументу копіюється у формальний параметр методу. Отже, зміни, які вносяться до параметра методу, не надають жодного впливу на аргумент, який використовується для виклику. А другим способом передачі аргументу є виклик за посиланням. В даному випадку параметру методу передається посилання на аргумент, а не значення аргументу. У методі це посилання використовується для доступу до конкретного аргументу, що вказується під час виклику. Це означає, що зміни, які вносяться в параметр, впливатимуть на аргумент, який використовується для виклику методу.

4. Використання модифікаторів параметрів ref та out

Як пояснювалося вище, аргументи простих типів, наприклад int або char, передаються методу за значенням. Це означає, що зміни, які вносяться до параметра, що приймає значення, не будуть впливати на аргумент, який використовується для виклику. Але таку поведінку можна змінити, використовуючи ключові слова ref і out для передачі значень звичайних типів за посиланням. Це дозволяє змінити в самому методі аргумент, що вказується при його виклику.

4.1. Використання модифікатора параметра ref

Модифікатор параметра ref примусово організує виклик за посиланням, а не за значенням. Цей модифікатор вказується як при оголошенні, так і при виклику методу.

Модифікатор ref вказується перед оголошенням параметра у самому методі та перед аргументом при виклику методу.

Щодо модифікатора ref необхідно мати на увазі наступне. Аргументу, що передається за посиланням за допомогою цього модифікатора, має бути надано значення до виклику методу. Справа в тому, що в методі, що отримує такий аргумент як параметр, передбачається, що параметр посилається на дійсне значення. Отже, під час використання модифікатора ref в методі не можна встановити початкове значення аргументу.

4.2. Використання модифікатора параметра out

Модифікатор параметра out подібний до модифікатора ref, за одним винятком: він служить тільки для передачі значення за межі методу. Тому змінній, яка використовується в якості параметра out, не потрібно (та й марно) присвоювати якесь значення. Більш того, в методі параметр out вважається неініціалізованим, передбачається, що у нього відсутнє початкове значення. Це означає, що значення повинно бути присвоєно даному параметру методом до його завершення. Отже, після виклику методу параметр out міститиме деяке значення.

4.3. Використання модифікаторів ref та out для посилань на об'єкти

Застосування модифікаторів ref і out не обмежується лише передачі значень звичайних типів. З їх допомогою можна також надсилати посилання на об'єкти. Якщо модифікатор ref або out вказує на посилання, то саме посилання передається за посиланням. Це дозволяє змінити в методі об'єкт, на який вказує посилання.

5. Використання змінної кількості аргументів

Іноді виникає потреба створити метод, якому можна було б передати довільну кількість аргументів. Такий метод не можна створити, використовуючи звичайні параметри. Натомість доведеться скористатися спеціальним типом параметра, що позначає довільне число параметрів. І це робиться за допомогою створюваного параметра типу params.

Для оголошення масиву параметрів, здатного приймати від нуля до декількох аргументів, служить модифікатор params. Число елементів масиву параметрів дорівнюватиме кількості аргументів, що передаються методу. А для отримання аргументів у програмі організується доступ до даного масиву.

6. Повернення об'єктів із методів

Метод може повернути дані будь-якого типу, зокрема і тип класу.

Коли метод повертає об'єкт, останній продовжує існувати до тих пір, поки не залишиться посилань на нього. Після цього він підлягає збірці як "сміття". Отже, об'єкт не знищується тільки тому, що завершується метод, що створив його.

Одним з практичних прикладів застосування даних типу об'єктів, що повертаються, служить фабрика класу, яка представляє собою метод, призначений для побудови об'єктів його ж класу. У ряді випадків надавати користувачам класу доступ до його конструктора небажано з міркувань безпеки або тому, що побудова об'єкта залежить від деяких зовнішніх факторів. В подібних випадках для побудови об'єктів використовується фабрика класу.

7. Повернення масиву з методу

У C# масиви реалізовані у виді об'єктів, а це означає, що метод може також повернути масив.

ЛЕКЦІЯ №3. ПЕРЕВАНТАЖЕННЯ МЕТОДІВ ТА КОНСТРУКТОРІВ

1. Перевантаження методів
2. Перевантаження конструкторів
  2.1. Виклик перевантаженого конструктора за допомогою ключового слова this
3. Ініціалізатори об'єктів
4. Необов'язкові аргументи
  4.1. Необов'язкові аргументи та перевантаження методів
  4.2. Необов'язкові аргументи та неоднозначність
  4.3. Практичний приклад використання необов'язкових аргументів
5. Іменовані аргументи
6. Метод Main()
  6.1. Повернення значень з методу Main()
  6.2. Передача аргументів методом Main()
7. Рекурсія
8. Застосування ключового слова static
  8.1. Статичні конструктори
  8.2. Статичні класи

У C# допускається спільне використання одного й того ж імені двома чи більше методами одного й того ж класу, за умови, що їх параметри оголошуються по-різному. У цьому випадку кажуть, що методи перевантажуються, а сам процес називається перевантаженням методів. Перевантаження методів відноситься до одного із способів реалізації поліморфізму в C#.

1. Перевантаження методів

Загалом, для перевантаження методу досить оголосити різні його варіанти, а про інше подбає компілятор. Але при цьому необхідно дотриматися наступної важливої умови: тип або число параметрів у кожного методу повинні бути різними. Цілком недостатньо, щоб два методи відрізнялися тільки типами значень, що повертаються. Вони повинні також відрізнятися типами чи кількістю своїх параметрів. (У всякому разі, типи значень, що повертаються, дають недостатньо відомостей компілятору C#, щоб вирішити, який саме метод слід використовувати.) Зрозуміло, перевантажені методи можуть відрізнятися і типами значень, що повертаються. Коли викликається перевантажений метод, то виконується той його варіант, параметри якого відповідають (за типом і числом) переданим аргументам.

2. Перевантаження конструкторів

Як і методи, конструктори також можуть перевантажуватися. Це дає можливість конструювати об'єкти самими різними способами.

Одна з найпоширеніших причин перевантаження конструкторів полягає в необхідності надати можливість одним об'єктам ініціалізувати інші.

2.1. Виклик перевантаженого конструктора за допомогою ключового слова this

Коли доводиться працювати з перевантаженими конструкторами, то іноді дуже корисно надати можливість одному конструктору викликати інший. У C# це виконується за допомогою ключового слова this.

Викликати перевантажений конструктор за допомогою ключового слова this корисно, зокрема, тому, що він дозволяє виключити непотрібне дублювання коду. Інша перевага організації подібного виклику перевантаженого конструктора, полягає в можливості створювати конструктори з аргументами, що задаються "за замовчуванням", коли ці аргументи не вказані явно.

3. Ініціалізатори об'єктів

Ініціалізатори об'єктів надають ще один спосіб створення об'єкта та ініціалізації його полів та властивостей. Якщо використовуються ініціалізатори об'єктів, то замість звичайного виклику конструктора класу вказуються імена полів або властивостей, що ініціалізуються початково заданими значенням. Отже, синтаксис ініціалізації об'єкта надає альтернативу явному виклику конструктора класу.

4. Необов'язкові аргументи

У версії C# 4.0 впроваджено новий засіб, що підвищує зручність вказівки аргументів при виклику методу. Цей засіб називається необов'язковими аргументами і дозволяє визначити значення для параметра методу, що використовується за замовчуванням. Дане значення буде використовуватися за замовчуванням у тому випадку, якщо для параметра не вказано відповідного аргументу при виклику методу. Отже, вказувати аргумент для такого параметра необов'язково. Необов'язкові аргументи дозволяють спростити виклик методів, де для деяких

параметрів застосовуються аргументи, які вибираються за замовчуванням. Їх можна також використовувати в якості "скороченої" форми перевантаження методів.

Застосування необов'язкового аргументу дозволяється при створенні необов'язкового параметра. Для цього достатньо вказати значення параметра, що використовується за замовчуванням, за допомогою синтаксису, аналогічного ініціалізації змінної. Значення, що використовується за замовчуванням, повинно бути константним виразом.

4.1. Необов'язкові аргументи та перевантаження методів

У деяких випадках необов'язкові аргументи можуть стати альтернативою перевантаження методів.

4.2. Необов'язкові аргументи та неоднозначність

При використанні необов'язкових аргументів може виникнути така складність, як неоднозначність. Щось подібне може статися при перевантаженні методу з необов'язковими параметрами. У деяких випадках компілятор може виявитися неспроможним визначити, який саме варіант методу слід викликати, коли необов'язкові аргументи не задані.

У зв'язку з тим, що перевантаження методів, що допускають застосування необов'язкових аргументів, може призвести до неоднозначності, дуже важливо брати до уваги наслідки такого перевантаження. У деяких випадках, можливо, доведеться відмовитися від застосування необов'язкових аргументів, щоб унеможливити неоднозначність і тим самим запобігти використанню методу ненавмисним чином.

4.3. Практичний приклад використання необов'язкових аргументів

Основна перевага необов’язкових аргументів полягає в тому, що при виклику методу можна вказувати тільки ті аргументи, які потрібні. А передавати значення, що явно встановлюються за замовчуванням, не потрібно.

Необов'язкові аргументи виявляються дуже ефективним засобом лише в тому випадку, якщо вони використовуються правильно. Вони призначені для того, щоб метод виконував свої функції ефективно, а користуватися ним можна було б просто та зручно. У цьому відношенні значення всіх аргументів, що встановлюються за замовчуванням, повинні спрощувати звичайне застосування методу. В іншому випадку необов'язкові аргументи здатні порушити структуру коду і ввести в оману тих, хто користується ним. І нарешті, значення необов'язкового параметра встановлене за замовчуванням не повинно завдавати ніякої шкоди. Інакше кажучи, ненавмисне використання необов'язкового аргументу не повинно призводити до незворотних, негативних наслідків.

5. Іменовані аргументи

Ще одним засобом, пов'язаним із передачею аргументів методу, є іменований аргумент. При передачі аргументів методу порядок їх слідування, як правило, повинен збігатися з тим порядком, в якому параметри визначені в самому методі. Іншими словами, значення аргументу присвоюється параметру по його позиції у списку аргументів. Дане обмеження покликані подолати іменовані аргументи. Іменований аргумент дозволяє вказати ім'я того параметра, якому присвоюється його значення. І в цьому випадку порядок слідування аргументів уже не має жодного значення.

6. Метод Main()

У наведених досі прикладах програм використовувалася одна форма методу Main(). Але в нього є також цілий ряд перевантажених форм. Одні з них можуть служити для повернення значень, інші – для отримання аргументів. У цьому розділі розглядаються і ті та інші форми.

6.1. Повернення значень з методу Main()

По завершенні програми є можливість повернути конкретне значення з методу Main() процесу, що виконує виклик (часто операційній системі).

Як правило, значення, що повертається методом Main(), вказує на нормальне завершення програми або на аварійне її завершення через ненормальні умови виконання, що склалися. Умовно нульове значення, що повертається, зазвичай вказує на нормальне завершення програми, а всі інші значення позначають тип помилки, що виникла.

6.2. Передача аргументів методом Main()

Багато програм приймають так звані аргументи командної строчки, інформацію, що вказується у командній строчці безпосередньо після імені програми при її запуску на виконання. У програмах на C# такі аргументи передаються потім методу Main().

7. Рекурсія

У C# допускається, щоб метод викликав сам себе. Цей процес називається рекурсією, а метод, що викликає сам себе, – рекурсивним. Взагалі, рекурсія представляє собою процес, в ході якого щось визначає саме себе. У цьому відношенні вона чимось нагадує циклічне визначення. Рекурсивний метод відрізняється головним чином тим, що він містить оператор, у якому цей метод викликає сам себе. Рекурсія є ефективним механізмом управління програмою.

8. Застосування ключового слова static

Якщо член класу оголошується як static, то він стає доступним до створення будь-яких об'єктів свого класу без посилання на будь-який об'єкт. За допомогою ключового слова static можна оголошувати як змінні, так і методи. Найбільш характерним прикладом члена типу static служить метод Main(), який оголошується таким тому, що він повинен викликатися операційною системою на самому початку виконуваної програми.

Для того, щоб скористатися членом типу static за межами класу, достатньо вказати ім'я цього класу з оператором-точкою. Але створювати об'єкт цього не потрібно. Насправді член типу static виявляється доступним не за посиланням на об'єкт, а за ім'ям свого класу.

8.1. Статичні конструктори

Конструктор можна також оголосити як static. Статичний конструктор, як правило, використовується для ініціалізації компонентів, що застосовуються до всього класу, а не до окремого екземпляра об'єкта цього класу. Тому члени класу ініціалізуються статичним конструктором до створення будь-яких об'єктів цього класу.

8.2. Статичні класи

Клас можна оголошувати як static. Статичний клас має дві основні властивості. По-перше, об'єкти статичного класу створювати не можна. І по-друге, статичний клас повинен містити лише статичні члени.

ЛЕКЦІЯ №4. ПЕРЕВАНТАЖЕННЯ ОСНОВНИХ ОПЕРАТОРІВ

1. Основи перевантаження операторів
2. Перевантаження бінарних операторів
3. Перевантаження унарних операторів
4. Виконання операцій з вбудованими в C# типами даних
5. Перевантаження операторів відношення
6. Перевантаження операторів true і false

У мові C# допускається визначати призначення оператора по відношенню до створюваного класу. Цей процес називається перевантаженням операторів. Завдяки перевантаженню розширюється сфера застосування оператора в класі. При цьому дія оператора повністю контролюється і може змінюватися в залежності від конкретного класу. Наприклад, оператор + може використовуватися для введення об'єкта в зв'язний список в одному класі, де визначається такий список, тоді як в іншому класі його призначення може виявитися зовсім іншим.

Коли оператор перевантажується, жодне з його початкових призначень не втрачається. Він просто виконує ще одну, нову операцію щодо конкретного об'єкта. Тому перевантаження оператора +, наприклад, для обробки зв'язного списку не змінює його призначення по відношенню до цілих чисел, тобто до додавання.

Головна перевага перевантаження операторів полягає в тому, що вона дозволяє плавно інтегрувати клас нового типу в середу програмування. Подібного роду розширюваність типів є важливою складовою ефективності такої об'єктно-орієнтованої мови програмування, як C#. Як тільки для класу визначаються оператори, з'являється можливість оперувати об'єктами цього класу, використовуючи звичайний синтаксис виразів в C#. Перевантаження операторів є однією з найсильніших сторін мови C#.

1. Основи перевантаження операторів

Перевантаження операторів тісно пов'язане з перевантаженням методів. Для перевантаження оператора служить ключове слово operator, що визначає операційний метод, який, в свою чергу, визначає дію оператора щодо свого класу. Існують дві форми операторних методів (operator): одна – для унарних операторів, інша – для бінарних.

Загальна форма для кожного різновиду цих операторних методів містить символ оператору, що перевантажується, наприклад + або /; та визначає конкретний тип значення, що повертається зазначеною операцією. Це значення може бути будь-якого типу, але часто воно вказується такого ж типу, як і у класу, для якого перевантажується оператор. Така кореляція спрощує застосування перевантажених операторів в виразах. Для унарних операторів операнд позначає значення, що передається, а для бінарних операторів те ж саме позначають перший і другий операнд. Зверніть увагу на те, що операційні методи повинні мати обидва типи, public і static.

Тип операнда унарних операторів повинен бути таким же, як і у класу, для якого перевантажується оператор. А в бінарних операторах хоча б один з операндів має бути такого ж типу, як і у його класу. Отже, в C# не допускається перевантаження будь-яких операторів для об'єктів, які ще не були створені. Наприклад, призначення оператора + неможна перевизначити для елементів типу int або string.

І ще одне зауваження: в параметрах оператора неможна використовувати модифікатор ref або out.

2. Перевантаження бінарних операторів

Для того щоб продемонструвати принцип дії перевантаження операторів, розглянемо простий приклад, в якому перевантажуються два оператора: + і -. Розглянемо клас ThreeD, що містить координати об'єкта в тривимірному просторі. Оператор +, що перевантажується, додає окремі

координати одного об'єкта типу ThreeD з координатами іншого. А оператор -, що перевантажується, віднімає координати одного об'єкта з координат іншого.

Уважно проаналізуємо приклад, починаючи з оператора +, що перевантажується. Коли оператор + оперує двома об'єктами типу ThreeD, то величини їх відповідних координат додаються, як показано в оголошенні операційного методу operator + (). Слід, однак, мати на увазі, що цей оператор не видозмінює значення своїх операндів, а лише повертає новий об'єкт типу ThreeD, що містить результат операції додавання координат. Незважаючи на те, що жодне з правил не перешкоджає перевантаженому оператору змінити значення одного зі своїх операндів, все ж краще, щоб дії цього оператора відповідали його прямим призначенням.

Зверніть увагу на те, що метод operator + () повертає об'єкт типу ThreeD. Цей метод міг би повернути значення будь-якого допустимого в C# типу, але завдяки тому що він повертає об'єкт типу ThreeD, оператор + можна використовувати в таких складових виразах, як a + b + c. В даному випадку вираз a + b дає результат типу ThreeD, який можна потім скласти з об'єктом з того ж типу. Якби вираз a + b давало результат іншого типу, то обчислити складений вираз a + b + c було б просто неможливо.

Слід також підкреслити, що коли окремі координати точок додаються в операторі operator + (), то в результаті такого додавання виходять цілі значення, оскільки окремі координати x, y і z представлені цілими величинами. Але саме перевантаження оператора + для об'єктів типу ThreeD не робить ніякого впливу на операцію додавання цілих значень, тобто вона не змінює первісне призначення цього оператора.

А тепер проаналізуємо операторний метод operator- (). Оператор - діє так само, як і оператор +, але для нього важливий порядок проходження операндів. Нагадаємо, що додавання носить комутативний характер (від перестановки доданків сума не змінюється), чого неможна сказати про різницю: A - B не те ж саме, що і B - A. Для всіх бінарних операторів першим параметром операційного методу є лівий операнд, а другим параметром – правий операнд. Тому, реалізуючи перевантажені варіанти некомутативних операторів , слід пам'ятати, який саме операнд повинен бути вказаний зліва і який – справа.

3. Перевантаження унарних операторів

Унарні оператори перевантажуються таким же чином, як і бінарні. Головна відмінність полягає, звичайно, в тому, що у них є лише один операнд. Як приклад розглянемо перевантаження оператора унарного мінуса для класу ThreeD.

В даному прикладі створюється новий об'єкт, в полях якого зберігаються від'ємні значення операнда унарного оператора, що перевантажується, після чого цей об'єкт повертається операційним методом. Зверніть увагу на те, що сам операнд не змінюється. Це означає, що і в даному випадку звичайне призначення оператора унарного мінуса зберігається.

У C# перевантаження операторів ++ і -- здійснюється досить просто. Для цього досить повернути інкрементоване або декрементоване значення, але не змінювати об'єкт, що виконує виклик. А все інше візьме на себе компілятор C#, розрізняючи префіксні і постфіксні форми цих операторів.

4. Виконання операцій з вбудованими в C# типами даних

Для будь-якого заданого класу і оператора є також можливість перевантажити сам операційний метод. Це, зокрема, потрібно для того, щоб дозволити операції з типом класу і іншими типами даних, в тому числі і вбудованими. Знову звернемося до класу ThreeD. На прикладі цього класу раніше було показано, як оператор + перевантажується для додавання координат одного об'єкта типу ThreeD з координатами іншого. Але це далеко не єдиний спосіб визначення операції додавання для класу ThreeD. Так, було б не менш корисно додати ціле значення до кожної координати об'єкту типу ThreeD. Подібна операція стала б у пригоді для перенесення осей координат.

В результаті чергового перевантаження оператора +, другий параметр операційного методу має тип int. Отже, в цьому методі дозволяється додавання цілого значення з кожним полем об'єкта типу ThreeD. Така операція цілком допустима, тому що, як пояснювалося вище, при перевантаженні бінарного оператора один з його операндів повинен бути того ж типу, що й клас, для якого цей оператор перевантажується. Але у другого операнда цього оператора може бути будь-який інший тип.

Отже, коли оператор + застосовується до двох об'єктів класу ThreeD, то додаються їх координати. А коли він застосовується до об'єкта типу ThreeD і цілого значення, то координати цього об'єкту збільшуються на задане ціле значення.

Продемонстроване вище перевантаження оператора +, безумовно, розширює корисні функції класу ThreeD, тим не менш, воно робить це не до кінця. І ось чому. Метод operator + (ThreeD, int) дозволяє виконувати операції, подібні додаванню до об’єкту ThreeD деякого цілого значення. Але, на жаль, він не дозволяє виконувати операції, аналогічні додаванню до цілого значення деякого об’єкту типу ThreeD.

Справа в тому, що другий цілочисельний аргумент даного методу позначає правий операнд бінарного оператора +, але в наведеній вище строчці коду цілочисельний аргумент вказується зліва. Для того щоб дозволити виконання такої операції додавання, доведеться перевантажити оператор + ще раз. У цьому випадку перший параметр операційного методу повинен мати тип int, а другий параметр – тип ThreeD. Таким чином, в одному варіанті методу operator + () виконується додавання об'єкта типу ThreeD і цілого значення, а в другому – додавання цілого значення і об'єкта типу ThreeD. Завдяки такому перевантаженню оператора + (або будь-якого іншого бінарного оператора) допускається поява вбудованого типу даних як з лівого, так і з правого боку даного оператора.

5. Перевантаження операторів відношення

Оператори відношення, наприклад == і <, можуть також перевантажуватися, причому дуже просто. Як правило, перевантажений оператор відношення повертає логічне значення true і false. Це цілком відповідає правилам звичайного застосування подібних операторів і дає можливість використовувати їх перевантажені різновиди в умовних виразах. Якщо ж повертається результат іншого типу, то тим самим сильно обмежується можливість застосування операторів відношення.

Розглянемо варіант класу ThreeD, в якому перевантажуються оператори < і >. В даному прикладі ці оператори служать для порівняння об'єктів ThreeD, виходячи з їх відстані до початку координат. Один об'єкт вважається більше іншого, якщо він знаходиться далі від початку координат. А крім того, один об'єкт вважається менше іншого, якщо він знаходиться ближче до початку координат. Такий варіант реалізації дозволяє, зокрема, визначити, яка з двох заданих точок знаходиться на більшій сфері. Якщо ж жоден з операторів не повертає логічне значення true, то обидві точки знаходяться на одній і тій же сфері. Зрозуміло, можливі й інші алгоритми впорядкування.

На перевантаження операторів відношення накладається таке важливе обмеження: вони повинні перевантажуватися попарно. Так, якщо перевантажується оператор <, то слід перевантажити і оператор >, і навпаки. І ще одне зауваження: якщо перевантажуються оператори == і !=, то для цього зазвичай потрібно також перевизначити методи Object.Equals () і Object.GetHashCode ().

6. Перевантаження операторів true і false

Ключові слова true і false можна також використовувати в якості унарних операторів для цілей перевантаження. Перевантажені варіанти цих операторів дозволяють визначити призначення ключових слів true і false спеціально для створюваних класів. Після перевантаження цих ключових слів як унарних операторів для конкретного класу з'являється можливість

використовувати об'єкти цього класу для управління операторами if, while, for і do-while або ж в умовному виразі ?. Оператори true і false повинні перевантажуватися попарно, а не окремо.

Розглянемо приклад, що демонструє реалізацію операторів true і false у класі ThreeD. У кожному з цих операторів перевіряється така умова: якщо хоча б одна з координат об'єкту типу ThreeD дорівнює нулю, то цей об'єкт істинний, а якщо всі три його координати дорівнюють нулю, то такий об'єкт хибний.

Об'єкти класу ThreeD можуть використовуватися для управління умовним оператором if і оператором циклу do-while. Так, в операторах if об'єкт типу ThreeD перевіряється за допомогою оператора true. Якщо результат цієї перевірки виявляється істинним, то оператор if виконується. А в операторі циклу do-while об'єкт b декрементується на кожному кроці циклу. Отже, цикл повторюється до тих пір, поки перевірка об'єкта b дає істинний результат, тобто цей об'єкт містить хоча б одну ненульову координату. Якщо ж виявиться, що об'єкт b містить всі нульові координати, його перевірка за допомогою оператора true дасть помилковий результат і цикл завершиться.

ЛЕКЦІЯ №5. ПЕРЕВАНТАЖЕННЯ СПЕЦІАЛІЗОВАНИХ ОПЕРАТОРІВ

1. Перевантаження логічних операторів
2. Простий спосіб перевантаження логічних операторів
3. Укорочені логічні оператори доступні для застосування
4. Оператори перетворення
5. Рекомендації і обмеження по перевантаженню операторів
6. Приклад перевантаження операторів

У цьому розділі продовжується розгляд перевантаження операторів класу. Представлено простий спосіб перевантаження логічних операторів, а також механізм, завдяки якому стає можливим застосування укорочених логічних операторів для класу даних користувача. Демонструється можливість організації приведення типу даних користувача класу до вбудованого типу даних, за допомогою перевантаження операторів явного та неявного перетворення. Наводяться рекомендації та обмеження для організації перевантаження операторів для типу даних користувача.

1. Перевантаження логічних операторів

У C# передбачені наступні логічні оператори: &, |, !, && і ||. З них перевантаженню, безумовно, підлягають тільки оператори &, | і !. Проте, дотримуючись певних правил, можна отримати також користь з укорочених логічних операторів && і ||. Всі ці можливості розглядаються нижче.

2. Простий спосіб перевантаження логічних операторів

Розглянемо спочатку найпростіший випадок. Якщо не користуватися укороченими логічними операторами, то перевантаження операторів & і | можна виконувати абсолютно природним шляхом, отримуючи в кожному випадку результат типу bool. Аналогічний результат, як правило, дає і перевантажений оператор !.

Розглянемо приклад який демонструє перевантаження логічних операторів !, & і | для об'єктів типу ThreeD. Об'єкт типу ThreeD вважається істинним, якщо хоча б одна з його координат не дорівнює нулю. Якщо ж всі три координати об'єкта дорівнюють нулю, то він вважається хибним.

При такому способі перевантаження логічних операторів &, | і ! методи кожного з них повертають результат типу bool. Це необхідно для того, щоб використовувати оператори, що розглядаються, звичайним чином, тобто в тих виразах, де передбачається результат типу bool. Нагадаємо, що для всіх вбудованих в C# типів даних результатом логічної операції повинно бути

значення типу bool. Тому цілком розумно передбачити повернення значення типу bool і в перевантажених варіантах цих логічних операторів. Але, на жаль, такий спосіб перевантаження придатний лише в тому випадку, якщо не потрібні укорочені логічні оператори.

3. Укорочені логічні оператори доступні для застосування

Для того щоб застосування укорочених логічних операторів && і || стало можливим, необхідно дотримуватися наступних чотирьох правил. По-перше, в класі повинно бути проведено перевантаження логічних операторів & і |. По-друге, перевантажені методи операторів & і | повинні повертати значення того ж типу, що й у класі, для якого ці оператори перевантажуються. По-третє, кожен параметр повинен містити посилання на об'єкт того класу, для якого перевантажується логічний оператор. І по-четверте, для класу повинні бути перевантажені оператори true і false. Якщо всі ці умови виконуються, то укорочені логічні оператори автоматично стають придатними для застосування.

Вони діють у такий спосіб. Перший операнд перевіряється за допомогою операторного методу operator true (для оператора ||) або ж за допомогою операторного методу operator false (для оператора &&). Якщо вдається визначити результат даної операції, то відповідний перевантажений оператор (& або |) далі не виконується. В іншому випадку перевантажений оператор (& або | відповідно) використовується для визначення кінцевого результату. Отже, коли застосовується укорочений логічний оператор && або ||, то відповідний логічний оператор & або | викликається лише в тому випадку, якщо за першим операндом неможливо визначити результат обчислення виразу.

Описаний вище спосіб застосування укорочених логічних операторів може здатися, на перший погляд, дещо заплутаним, але якщо подумати, то в такому застосуванні виявляється відомий практичний сенс. Адже завдяки перевантаження операторів true і false для класу компілятор отримує дозвіл на застосування укорочених логічних операторів, не вдаючись до явного їх перевантаження. Це дає також можливість використовувати об'єкти в умовних виразах. І взагалі, логічні оператори & і | найкраще реалізовувати повністю, якщо, звичайно, не потрібно дуже вузько спрямована їх реалізація.

4. Оператори перетворення

Іноді об'єкт певного класу потрібно використовувати в виразі, що включає в себе дані інших типів. В одних випадках для цієї мети виявляється придатною перевантаження одного або більше операторів, а в інших випадках – звичайне перетворення типу класу в цільовий тип. Для подібних ситуацій в C# передбачено спеціальний різновид операційного методу, так званий оператор перетворення. Такий оператор перетворює об'єкт вихідного класу в інший тип. Оператори перетворення допомагають повністю інтегрувати типи класів в середу програмування на C#, дозволяючи вільно користуватися класами разом з іншими типами даних, за умови, що визначений порядок перетворення в ці типи.

Існують дві форми операторів перетворення: явна і неявна.

Якщо оператор перетворення вказано в неявній формі (implicit), то перетворення викликається автоматично, наприклад, в тому випадку, коли об'єкт використовується у виразі разом зі значенням цільового типу. Якщо ж оператор перетворення вказано в явній формі (explicit), то перетворення викликається в тому випадку, коли виконується приведення типів. Для одних і тих самих вихідних і цільових типів даних неможна вказувати оператор перетворення одночасно в явній і неявній формі.

Оператор неявного перетворення застосовується автоматично в наступних випадках: коли в виразі потрібно перетворення типів; методу передається об'єкт; здійснюється присвоювання і проводиться явне приведення до цільового типу. З іншого боку, можна створити оператор явного перетворення, що викликається тільки тоді, коли проводиться явне приведення типів. У такому випадку оператор явного перетворення не викликається автоматично.

На оператори перетворення накладається ряд зазначених нижче обмежень.

1. Вихідний або цільовий тип перетворення повинен ставитися до класу, для якого оголошено дане перетворення. Зокрема, неможна перевизначити перетворення в тип int, якщо воно спочатку вказано як перетворення в тип double.

2. Неможна вказувати перетворення в клас object або ж з цього класу.

3. Для одних і тих самих вихідних і цільових типів даних неможна вказувати одночасно явне і неявне перетворення.

4. Неможна вказувати перетворення базового класу в похідний клас.

5. Неможна вказувати перетворення в інтерфейс або ж з нього.

Крім зазначених вище обмежень, є ряд рекомендацій, якими зазвичай керуються при виборі операторів явного або неявного перетворення. Незважаючи на всі переваги неявних перетворень, до них слід вдаватися тільки в тих випадках, коли перетворенню не властиві помилки. Щоб уникнути подібних помилок неявні перетворення повинні бути організовані тільки в тому випадку, якщо задовольняються наступні умови. По-перше, інформація не губиться, наприклад, в результаті усічення, переповнення або втрати знака. І по-друге, перетворення не призводить до виняткової ситуації. Якщо ж неявне перетворення не задовольняє цим двом умовам, то слід вибрати явне перетворення.

5. Рекомендації і обмеження по перевантаженню операторів

Дія перевантаженого оператора поширюється на клас, для якого він визначається, і ніяк не пов'язане з його початковим застосуванням до даних вбудованих в C# типів. Але заради збереження ясності структури і легкості читання вихідного коду оператор, що перевантажується, повинен, по можливості, відображати основну суть свого первісного призначення. Головний принцип перевантаження операторів полягає в наступному: незважаючи на те, що оператор, що перевантажується, може отримати будь-яке призначення, заради ясності нове його призначення має бути так чи інакше пов'язане з його початковим призначенням.

На перевантаження операторів накладається ряд обмежень. Зокрема, неможна змінювати пріоритет будь-якого оператора або кількість операндів, яка потрібна для оператора, хоча в операційному методі можна і проігнорувати операнд. Крім того, є ряд операторів, які неможна перевантажувати. А найголовніше, що перевантаженню не підлягає ні один з операторів присвоювання, в тому числі і складові, як, наприклад, оператор + =. Нижче перераховані оператори, які неможна перевантажувати: &&, (),.,?, ??, [], ||, =, =>, ->, as, checked, default, is, new, sizeof, typeof, unchecked.

Незважаючи на те, що оператор приведення () неможна перевантажувати явним чином, є все-таки можливість створити згадувані раніше оператори перетворення, які виконують ту ж саму функцію.

Обмеження, пов'язане з тим, що деякі оператори, наприклад +=, неможна перевантажувати, насправді не є таким вже непереборним. Взагалі кажучи, якщо оператор визначений як можливий до перевантаження і використовується в складеному операторі присвоювання, то зазвичай викликається метод цього перевантаженого оператора. Отже, при зверненні до оператора += в програмі автоматично викликається заздалегідь оголошений варіант методу operator + ().

І останнє зауваження: незважаючи на те, що оператор індексації масиву [] неможна перевантажувати за допомогою операторного методу, є можливість створити індексатори.

6. Приклад перевантаження операторів

Загальні принципи перевантаження операторів залишаються незмінними незалежно від застосовуваного класу, тим не менш, розглянемо приклад в якому наочно демонструються сильні сторони перевантаження, особливо якщо це стосується розширюваності типів.

В даному прикладі розробляється 4-розрядний цілочисельний тип даних і для нього визначається ряд операцій. На ранній стадії розвитку обчислювальної техніки широко

застосовувався тип даних для позначення 4-розрядних двійкових величин, що називалися напівбайтами, оскільки вони становили половину байта, містили одну 16-у цифру і були зручні для введення коду напівбайтами, що в ті часи вважалося звичним заняттям для програмістів. У наш час цей тип даних застосовується рідко, але він як і раніше є цікавим доповненням цілочисельних типів даних в C#. За традицією напівбайт позначає ціле значення без знака.

Розглянемо приклад типу полубайтових даних, що реалізується за допомогою класу Nybble. В якості базового для нього використовується тип int, але з обмеженням на зберігання даних від 0 до 15. У класі Nybble визначаються наступні оператори.

Додавання двох об'єктів типу Nybble.

Додавання значення типу int з об'єктом типу Nybble.

Додавання об'єкта типу Nybble зі значенням типу int.

Операції порівняння: більше і менше.

Операція інкременту.

Перетворення значення типу int в об'єкт типу Nybble.

Перетворення об'єкта типу Nybble в значення типу int.

Перерахованих вище операцій досить, щоб показати, яким чином тип класу Nybble інтегрується в систему типів C#. Але для повноцінної реалізації цього типу даних доведеться визначити всі інші доступні для нього операції.

Більша частина функцій класу Nybble не вимагає особливих пояснень. Проте необхідно підкреслити ту особливу роль, яку оператори перетворення грають в інтегруванні класу типу Nybble в систему типів C#. Зокрема, об'єкт типу Nybble можна вільно комбінувати з даними інших типів в арифметичних виразах, оскільки визначені перетворення об'єкта цього типу в тип int і назад.

ЛЕКЦІЯ №6. ІНДЕКСАТОРИ І ВЛАСТИВОСТІ

У цьому розділі розглядаються дві особливі і тісно пов'язані один з одним різновиди членів класу: індексатори і властивості. Кожен з них по-своєму розширює можливості класу, сприяючи більш повної його інтеграції в систему типів C# і підвищуючи його гнучкість. Зокрема, індексатори надають механізм для індексування об'єктів подібно масивів, а властивості – раціональний спосіб управління доступом до даних екземпляру класу. Ці члени класу тісно пов'язані один з одним, оскільки обидва спираються на ще один доступний в C# засіб: аксесор.

1. Індексатори

Індексування масиву здійснюється за допомогою оператора []. Для створюваних класів можна визначити оператор [], але з цією метою замість операційного методу створюється індексатор, який дозволяє індексувати об'єкт, подібно масиву. Індексатори застосовуються, головним чином, в якості засобу, що підтримує створення спеціалізованих масивів, на які накладається одне або кілька обмежень. Проте, індексатори можуть служити практично будь-яким цілям, для яких

вигідним виявляється такий же синтаксис, як і у масивів. Індексатори можуть бути одне або багатовимірними. Розглянемо спочатку одновимірні індексатори.

1.1. Створення одновимірних індексаторів

Розглянемо загальну форму одновимірного індексатора. У кожного елемента, доступного за допомогою індексатора, повинен бути певний data_type. Цей тип відповідає типу елемента масиву. Параметр index отримує конкретний індекс елемента, до якого здійснюється доступ. Формально цей параметр зовсім не обов'язково повинен мати тип int, але оскільки індексатори, як правило, застосовуються для індексування масивів, то найчастіше використовується цілочисельний тип даного параметра.

У тілі індексатора визначені два аксесори (тобто засоби доступу до даних): get і set. Аксесор подібний до методу, за винятком того, що в ньому не оголошується тип значення або параметри. Аксесори викликаються автоматично при використанні індексатора, і обидва отримують індекс в якості параметра. Так, якщо індексатор вказується в лівій частині оператора присвоювання, то викликається аксесор set і встановлюється елемент, на який вказує параметр індекс. В іншому випадку викликається аксесор get і повертається значення, відповідне параметру індекс. Крім того, аксесор set отримує неявний параметр value, що містить значення, що присвоюється за вказаним індексом.

Перевага індексатора полягає, зокрема, в тому, що він дозволяє повністю контролювати доступ до масиву, уникаючи небажаного доступу.

Наявність обох аксесорів, get і set, в індексаторі не є обов'язковим. Так, можна створити індексатор тільки для читання, реалізувавши в ньому один лише аксесор get, або ж індексатор тільки для запису з єдиним аксесором set.

1.2. Перевантаження індексаторів

Індексатор може бути перевантажений. В цьому випадку для виконання вибирається той варіант індексатора, в якому точніше дотримується відповідність його параметра і аргументу, зазначених вище в якості індексу.

Як правило, індексатори перевантажуються для того, щоб використовувати об'єкт певного класу в якості індексу, що обчислюється якимось особливим чином.

1.3. Індексатори без базового масиву

Слід особливо підкреслити, що індексатор зовсім не обов'язково повинен оперувати масивом. Його основне призначення – надати користувачеві функціональні можливості, аналогічні масиву.

На застосування індексаторів накладаються два істотних обмеження. По-перше, значення, що видається індексатором, не можна передавати методу в якості параметра ref або out, оскільки в індексатора не визначене місце в пам'яті для його зберігання. І по-друге, індексатор повинен бути членом свого класу і тому не може бути оголошений як static.

1.4. Багатовимірні індексатори

Індексатори можна створювати і для багатовимірних масивів.

2. Властивості

Ще одним різновидом члена класу є властивість. Як правило, властивість поєднує в собі поле з методами доступу до нього. Як було показано в наведених раніше прикладах програм, поле часто створюється, щоб стати доступним для користувачів об'єкта, але при цьому бажано зберегти управління над операціями, дозволеними для цього поля, наприклад, обмежити діапазон

значень, що присвоюються даному полю. Цієї мети можна, звичайно, домогтися і за допомогою закритої змінної, а також методів доступу до її значення, але властивість надає більш досконалий і раціональний шлях для досягнення тієї ж самої мети.

Властивості дуже схожі на індексатори. Зокрема, властивість складається з імені і аксесорів get і set. Аксесори служать для отримання і установки значення змінної. Головна перевага властивості полягає в тому, що її ім'я може бути використано в виразах і операторах присвоювання аналогічно імені звичайної змінної, але в дійсності при зверненні до властивості по імені автоматично викликаються його аксесори get і set. Аналогічним чином використовуються аксесори get і set індексатора.

Як тільки властивість буде визначено, будь-яке звернення до властивості по імені приведе до автоматичного виклику відповідного аксесору. Крім того, аксесор set приймає неявний параметр value, який містить значення, що присвоюється властивості.

Слід, однак, мати на увазі, що властивості не визначають місце в пам'яті для зберігання полів, а лише управляють доступом до полів. Це означає, що саме властивість не надає поле, і тому поле повинно бути визначено незалежно від властивості.

3. Автоматично реалізовані властивості

Починаючи з версії C# 3.0, з'явилася можливість для реалізації дуже простих властивостей, не вдаючись до явного визначення змінної, якою управляє властивість. Замість цього базову змінну для властивості автоматично надає компілятор. Така властивість називається автоматично реалізовано.

На відміну, від звичайних властивостей автоматично реалізована властивість не може бути доступною тільки для читання або тільки для запису. При оголошенні цієї властивості в будь-якому випадку необхідно вказувати обидва аксесора – get і set. Хоча домогтися бажаного, зробити автоматично реалізовану властивість доступною тільки для читання або тільки для запису, все ж можна, оголосивши непотрібний аксесор як private.

Незважаючи на очевидні зручності автоматично реалізованих властивостей, їх застосування обмежується в основному тими ситуаціями, в яких не потрібно керувати установкою або отриманням значень з підтримуючих полів. Нагадаємо, що підтримуюче поле недоступно безпосередньо. Це означає, що на значення, яке може мати автоматично реалізовану властивість, не можна накласти ніяких обмежень. Отже, імена автоматично реалізованих властивостей просто замінюють собою імена самих полів, а найчастіше саме це і потрібно в програмі. Автоматично реалізовані властивості можуть виявитися корисними і в тих випадках, коли за допомогою властивостей функціональні можливості програми відкриваються для сторонніх користувачів, і для цієї мети можуть навіть застосовуватися спеціальні засоби проектування.

4. Застосування ініціалізаторів об'єктів у властивостях

Ініціалізатор об'єкта застосовується в якості альтернативи явного виклику конструктора при створенні об'єкта. За допомогою ініціалізаторів об'єктів задаються початкові значення полів або властивостей, які потрібно форматувати. При цьому синтаксис ініціалізаторів об'єктів виявляється однаковим як для властивостей, так і для полів.

5. Обмеження, властиві властивостям

Властивостям притаманний ряд істотних обмежень. По-перше, властивість не визначає місце для зберігання даних, і тому не може бути передана методу в якості параметра ref або out. По-друге, властивість не підлягає перевантаженню. Наявність двох різних властивостей з доступом до однієї і тієї ж змінної допускається, але це, скоріше, виняток, ніж правило. І нарешті, властивість не повинна змінювати стан базової змінної при виклику аксесора get. І хоча це обмежувальне правило не дотримується компілятором, його порушення вважається

семантичною помилкою. Дія аксесора get не повинна носити характер втручання у функціонування змінної.

6. Застосування модифікаторів доступу до аксесору

За замовчуванням доступність аксесорів set і get виявляється такою ж, як і у індексатора і властивості, частиною яких вони є. Так, якщо властивість оголошується як public, то за замовчуванням його аксесор set і get також стають відкритими (public). Проте для аксесорів set або get можна вказати власний модифікатор доступу, наприклад private. Але в будь-якому випадку доступність аксесорів, що визначається таким модифікатором, повинна бути більш обмеженою, ніж доступність, що вказується для його властивості або індексатора.

Існує цілий ряд причин, за якими потрібно обмежити доступність аксесорів. Припустимо, що потрібно надати вільний доступ до значення властивості, але в той же час дати можливість встановлювати це властивості тільки членам його класу. Для цього достатньо оголосити аксесор даної властивості як private.

7. Застосування індексаторів і властивостей

На завершення цього розділу представлено приклад класу RangeArray, в якому індексатори і властивості використовуються для створення типу масиву з межами індексування.

Індексування всіх масивів в C# починається з нуля. Але в деяких застосуваннях індексування масиву зручніше починати з будь-якої довільної точки відліку: з 1 або навіть з негативного числа, наприклад від -5 і до 5. Розглядається клас RangeArray розроблений таким чином, щоб допускати подібного роду індексування масивів.

Клас RangeArray реалізується таким чином, щоб підтримувати масиви типу int. Змінна a служить для звернення до базового масиву по посиланню. Пам'ять для нього розподіляється конструктором класу RangeArray. Нижня межа індексування масиву зберігається в змінній lowerBound, а верхня межа – у змінній upperBound. Далі оголошуються автоматично реалізовані властивості Length і Error. В обох властивостях аксесор set позначений як private.

При конструюванні об'єкту класу RangeArray передається нижня межа масиву як параметр low, а верхня межа – як параметр high. Потім значення параметра high інкрементується, оскільки межі індексування масиву змінюються від low до high включно. Далі виконується наступна перевірка: чи є верхній індекс більше нижнього індексу. Якщо це не так, то видається повідомлення про помилку і створюється масив, що складається з одного елемента. Після цього для масиву розподіляється пам'ять, а посилання на нього присвоюється змінній a. Потім властивість Length встановлюється рівною числу елементів масиву. І нарешті, встановлюються змінні lowerBound і upperBound. Далі в класі RangeArray реалізується його індексатор.

ЛЕКЦІЯ №7. БАЗОВІ ТА ПОХІДНІ КЛАСИ

1. Основи спадкування
2. Доступ до членів класу і спадкування
3. Організація захищеного доступу
4. Конструктори і спадкування
5. Виклик конструкторів базового класу
6. Спадкування і приховування імен
7. Застосування ключового слова base для доступу до прихованого імені
8. Створення багаторівневої ієрархії класів

Спадкування є одним з трьох основних принципів об'єктно-орієнтованого програмування, оскільки воно допускає створення ієрархічних класифікацій. Завдяки спадкуванню можна створити загальний клас, в якому визначаються характерні особливості, властиві безлічі

пов'язаних елементів. Від цього класу можуть потім наслідувати інші, більш конкретні класи, додаючи в нього свої індивідуальні особливості.

У мові C# клас, який успадковується, називається базовим, а клас, який успадковує, – похідним. Отже, похідний клас є спеціалізованим варіантом базового класу. Він успадковує всі змінні, методи, властивості і індексатори, що визначаються в базовому класі, додаючи до них свої власні елементи.

1. Основи спадкування

Підтримка спадкування в C# полягає в тому, що в оголошенні одного класу дозволяється вводити інший клас. Для цього при оголошенні похідного класу вказується базовий клас. Розглянемо для початку простий приклад: клас TwoDShape, що містить ширину і висоту двовимірного об'єкта, наприклад квадрата, прямокутника, трикутника.

Клас TwoDShape може стати базовим, відправною точкою для створення класів, що описують конкретні типи двовимірних об'єктів. Наприклад, клас TwoDShape служить для породження похідного класу Triangle.

У класі Triangle створюється особливий тип об'єкту класу TwoDShape, в даному випадку – трикутник. Крім того, в клас Triangle входять всі члени класу TwoDShape, до яких, зокрема, додаються методи Area() і ShowStyle(). Так, опис типу трикутника зберігається в змінній Style, метод Area() розраховує і повертає площу трикутника, а метод ShowStyle() відображає тип трикутника.

Всякий раз, коли один клас успадковує від іншого, після назви базового класу вказується ім'я похідного класу, відокремленого двокрапкою. У C# синтаксис спадкування класу дивно простий і зручний у використанні.

Незважаючи на те, що клас TwoDShape є базовим для класу Triangle, в той же час він є абсолютно незалежним і самодостатнім класом. Якщо клас слугує базовим для похідного класу, то це зовсім не означає, що він не може бути використаний самостійно.

Зрозуміло, об'єкт класу TwoDShape ніяк не пов'язаний з будь-яким з класів, похідних від класу TwoDShape, і взагалі не має до них доступу.

Для будь-якого похідного класу можна вказати тільки один базовий клас. У C# не передбачено спадкування декількох базових класів в одному похідному класі. В цьому відношенні C# відрізняється від C++, де допускається спадкування декількох базових класів. Дану обставину слід брати до уваги при перенесенні коду C++ в C#. Проте можна створити ієрархію спадкування, в якій похідний клас стає базовим для іншого похідного класу. Жоден з класів не може бути базовим для самого себе як безпосередньо, так і опосередковано. Але в будь-якому випадку похідний клас успадковує всі члени свого базового класу, в тому числі змінні екземпляра, методи, властивості і індексатори.

Головна перевага спадкування полягає в наступному: як тільки буде створено базовий клас, в якому визначено загальні для безлічі об'єктів атрибути, він може бути використаний для створення будь-якого числа більш конкретних похідних класів. А в кожному похідному класі може бути точно вибудувана своя власна класифікація.

2. Доступ до членів класу і спадкування

Члени класу часто оголошуються закритими, щоб виключити їх несанкціоноване або незаконне використання. Але спадкування класу не скасовує обмеження, що накладаються на доступ до закритих членів класу. Тому якщо в похідний клас і входять всі члени його базового класу, в ньому все одно виявляються недоступними ті члени базового класу, які є закритими.

На перший погляд, обмеження на доступ до приватних членів базового класу з похідного класу здається важко переборним, оскільки воно не дає в багатьох випадках можливості користуватися приватними членами цього класу. Але насправді це не так. Для подолання даного обмеження в C# передбачені різні способи. Один з них полягає у використанні захищених

(protected) членів класу, що розглядаються в наступному розділі, а другий – в застосуванні відкритих властивостей для доступу до закритих даних.

Властивість дозволяє управляти доступом до змінної екземпляра. Наприклад, за допомогою властивості можна ввести обмеження на доступ до значення змінної або ж зробити її доступною тільки для читання. Так, якщо зробити властивість відкритою, але оголосити її базову змінну закритою, то цією властивістю можна буде скористатися в похідному класі, але не можна буде отримати безпосередній доступ до її базової закритої змінної.

3. Організація захищеного доступу

Як пояснювалося вище, закритий член базового класу недоступний для похідного класу. З цього можна припустити, що для доступу до деякого члену базового класу з похідного класу цей член необхідно зробити відкритим. Але якщо зробити член класу відкритим, то він стане доступним для всього коду, що далеко не завжди бажано. Правда, згадане припущення вірне лише частково, оскільки в C# допускається створення захищеного члена класу. Захищений член є відкритим в межах ієрархії класів, але закритим за межами цієї ієрархії.

Захищений член створюється за допомогою модифікатора доступу protected. Якщо член класу оголошується як protected, він стає закритим, але за винятком одного випадку, коли захищений член успадковується. В цьому випадку захищений член базового класу стає захищеним членом похідного класу, а значить, доступним для похідного класу. Таким чином, використовуючи модифікатор доступу protected, можна створити члени класу, які є закритими для свого класу, але все ж успадкованими і доступними для похідного класу.

Аналогічно станам public і private, стан protected зберігається за членом класу незалежно від кількості рівнів спадкування. Тому коли похідний клас використовується в якості базового для іншого похідного класу, будь-який захищений член вихідного базового класу, що успадкований першим похідним класом, успадковується як захищений і другим похідним класом.

Незважаючи на всю свою корисність, захищений доступ придатний далеко не для всіх ситуацій. Модифікатор доступу protected слід застосовувати в тому випадку, якщо потрібно створити член класу, доступний для всієї ієрархії класів, але для решти коду він повинен бути закритим. А для управління доступом до значення члена класу краще скористатися властивістю.

4. Конструктори і спадкування

В ієрархії класів допускається, щоб у базових і похідних класів були свої власні конструктори. У зв'язку з цим виникає наступне резонне питання: який конструктор відповідає за побудову об'єкта похідного класу: конструктор базового класу, конструктор похідного класу або ж обидва. На це питання можна відповісти так: конструктор базового класу конструює базову частину об'єкта, а конструктор похідного класу – похідну частину цього об'єкта. І в цьому є своя логіка, оскільки базовому класу невідомі і недоступні будь-які елементи похідного класу, а значить, їх конструювання має відбуватися окремо. У наведених вище прикладах дане питання не виникало, оскільки вони спиралися на автоматичне створення конструкторів, які використовуються в C# за замовчуванням. Але на практиці конструктори визначаються в більшості класів. Нижче буде показано, яким чином реалізується подібна ситуація.

Якщо конструктор визначений тільки в похідному класі, то все відбувається дуже просто: конструюється об'єкт похідного класу, а базова частина об'єкта автоматично конструюється його конструктором за замовчуванням.

Коли конструктори визначаються як в базовому, так і в похідному класі, процес побудови об'єкта дещо ускладнюється, оскільки повинні виконуватися конструктори обох класів. В даному випадку доводиться звертатися до ще одного ключового слова мови C#: base, яке знаходить двояке застосування: по-перше, для виклику конструктора базового класу; і по-друге, для доступу до члена базового класу, який переховується за членом похідного класу.

5. Виклик конструкторів базового класу

За допомогою форми розширеного оголошення конструктора похідного класу і ключового слова base в похідному класі може бути викликаний конструктор, визначений у його базовому класі.

За допомогою ключового слова base можна викликати конструктор будь-якої форми, яка визначається в базовому класі, причому виконуватися буде лише той конструктор, параметри якого відповідають переданим аргументам.

А тепер розглянемо коротко основні принципи дії ключового слова base. Коли в похідному класі вказується ключове слово base, викликається конструктор з його безпосереднього базового класу. Отже, ключове слово base завжди звертається до базового класу, що стоїть в ієрархії безпосередньо над похідним класом. Це справедливо навіть для багаторівневої ієрархії класів. Аргументи передаються базовому конструктору в якості аргументів методу base(). Якщо ж ключове слово відсутнє, то автоматично викликається конструктор, який використовується в базовому класі за замовчуванням.

6. Спадкування і приховування імен

У похідному класі можна визначити член з таким же ім'ям, як і у члена його базового класу. В цьому випадку член базового класу ховається в похідному класі. І хоча формально в C# це не вважається помилкою, компілятор все ж видасть повідомлення, яке попереджає про те, що ім'я ховається. Якщо член базового класу потрібно приховати навмисно, то перед його ім'ям слід вказати ключове слово new, щоб уникнути появи подібного застережливого повідомлення. Слід, однак, мати на увазі, що це абсолютно окреме застосування ключового слова new, не схоже на його застосування при створенні екземпляра об'єкта.

Компілятору, по суті, повідомляється про те, що знову створена змінна і навмисно приховує змінну і з базового класу A і що автору програми про це відомо. Якщо ж опустити ключове слово new в цій строчці коду, то компілятор видасть попередження.

7. Застосування ключового слова base для доступу до прихованого імені

Є ще одна форма ключового слова base, яка діє подібно ключовому слову this, за винятком того, що вона завжди посилається на базовий клас в тому похідному класі, в якому вона використовується. Форма в загальному вигляді: base.member, де member може позначати метод або змінну екземпляра. Ця форма ключового слова base найчастіше застосовується в тих випадках, коли під іменами членів похідного класу ховаються члени базового класу з тими ж самими іменами. За допомогою ключового слова base можуть також викликатися приховані методи.

8. Створення багаторівневої ієрархії класів

У представлених до цих пір прикладах програм використовувалися прості ієрархії класів, що складалися тільки з базового і похідного класів. Але в C# можна також будувати ієрархії, що складаються з будь-якого числа рівнів спадкування. Як згадувалося вище, багаторівнева ієрархія ідеально підходить для використання одного похідного класу в якості базового для іншого похідного класу. Так, якщо є три класи, A, B і C, то клас C може успадковувати від класу B, а той, у свою чергу, від класу A. У такому разі кожен похідний клас успадковує характерні риси всіх своїх базових класів. Зокрема, клас C успадковує всі члени класів B і A.

Ключове слово base завжди позначає посилання на конструктор найближчого по ієрархії базового класу. Якщо ж в ієрархії класів конструктору базового класу потрібні параметри, то всі похідні класи повинні надавати ці параметри вгору по ієрархії, незалежно від того, потрібні вони самому похідному класу чи ні.

ЛЕКЦІЯ №8. АБСТРАКТНІ КЛАСИ

1. Порядок виклику конструкторів
2. Посилання на базовий клас і об'єкти похідних класів
3. Віртуальні методи і їх перевизначення
4. Функціональність перевизначення методів
5. Застосування віртуальних методів
6. Застосування абстрактних класів
7. Запобігання спадкування за допомогою ключового слова sealed
8. Клас object
  8.1. Упаковка і розпакування
  8.2. Клас object як універсальний тип даних

У цьому розділі продовжується розгляд організації спадкування у мові C#. Детально розглядається питання механізму побудови екземплярів похідних класів. Наведено важливий виняток із принципу строгої типізації мови C#, що реалізується завдяки базовим та похідним класам. Демонструється підвищення ступеня абстрактності за допомогою віртуальних та перевизначених методів у ієрархії класів, а також побудови абстрактних класів. Завершується розділ розглядом класу object, який представляє собою універсальний клас і вважається базовим для всіх класів і типів даних.

1. Порядок виклику конструкторів

На підставі викладеної інформації, може виникнути таке резонне питання: коли створюється об'єкт похідного класу і який конструктор виконується першим – той, що визначений в похідному класі, або ж той, що визначений в базовому класі. Так, якщо є базовий клас A і похідний клас B, то викликається конструктор класу A раніше конструктора класу B. Відповідь на це питання полягає в тому, що в ієрархії класів конструктори викликаються по порядку виведення класів: від базового до похідного. Більш того, цей порядок залишається незмінним незалежно від використання ключового слова base. Так, якщо ключове слово base не використовується, то виконується конструктор за замовчуванням, конструктор без параметрів.

Конструктори викликаються по порядку виведення їх класів. Якщо гарненько подумати, то у виклику конструкторів по порядку виведення їх класів можна виявити певний сенс. Адже базовому класу нічого не відомо ні про один з похідних від нього класів, і тому будь-яка ініціалізація, яка потрібна його членам, здійснюється абсолютно окремо від ініціалізації членів похідного класу, а можливо, це і необхідна умова. Отже, вона повинна виконуватися першою.

2. Посилання на базовий клас і об'єкти похідних класів

C# є строго типізована мова програмування. Крім стандартних перетворень і автоматичного просування простих типів значень, в цій мові строго дотримується принцип сумісності типів. Це означає, що змінна посилання на об'єкт класу одного типу, як правило, не може посилатися на об'єкт класу іншого типу.

Але з цього принципу суворого дотримання типів в C# є один важливий виняток: змінній посилання на об'єкт базового класу може бути присвоєно посилання на об'єкт будь-якого похідного від нього класу. Таке присвоювання вважається цілком припустимим, оскільки екземпляр об'єкта похідного типу інкапсулює екземпляр об'єкта базового типу. Отже, за посиланням на об'єкт базового класу можна звертатися до об'єкта похідного класу.

Слід особливо підкреслити, що доступ до конкретних членів класу визначається типом змінної посилання на об'єкт, а не типом об'єкта, на який вона посилається. Це означає, що якщо

посилання на об'єкт похідного класу присвоюється змінній посилання на об'єкт базового класу, то доступ дозволяється тільки до тих частин цього об'єкту, які визначаються базовим класом.

3. Віртуальні методи і їх перевизначення

Віртуальним називається такий метод, який оголошується як virtual в базовому класі. Віртуальний метод відрізняється тим, що він може бути перевизначений в одному або декількох похідних класах. Отже, у кожного похідного класу може бути свій варіант віртуального методу. Крім того, віртуальні методи цікаві тим, що саме відбувається при їх виклику за посиланням на базовий клас. В цьому випадку засобами мови C# визначається саме той варіант віртуального методу, який слід викликати, виходячи з типу об'єкта, до якого відбувається звернення за посиланням, причому це робиться під час виконання. Тому при посиланні на різні типи об'єктів виконуються різні варіанти віртуального методу. Іншими словами, варіант виконуваного віртуального методу вибирається по типу об'єкта, а не по типу посилання на цей об'єкт. Так, якщо базовий клас містить віртуальний метод і від нього отримані похідні класи, то при зверненні до різних типів об'єктів по посиланню на базовий клас виконуються різні варіанти цього віртуального методу.

Метод оголошується як віртуальний в базовому класі за допомогою ключового слова virtual, що зазначається перед його ім'ям. Коли ж віртуальний метод перевизначається в похідному класі, то для цього використовується модифікатор override. А сам процес повторного визначення віртуального методу в похідному класі називається перевизначенням методу. При перевизначенні ім'я, тип, що повертається, і сигнатура перевизначеного методу повинні бути точно такими ж, як і у того віртуального методу, який перевизначається. Крім того, віртуальний метод не може бути оголошений як static або abstract.

Перевизначення методу служить підставою для втілення одного з найефективніших в C# принципів: динамічної диспетчеризації методів, яка представляє собою механізм дозволу виклику під час виконання, а не компіляції. Значення динамічної диспетчеризації методів полягає в тому, що саме завдяки їй в C# реалізується динамічний поліморфізм.

Але перевизначати віртуальний метод зовсім не обов'язково. Адже якщо в похідному класі не надається власний варіант віртуального методу, то використовується його варіант з базового класу.

Якщо при наявності багаторівневої ієрархії віртуальний метод не перевизначається в похідному класі, то виконується найближчий його варіант, що виявляється вгору по ієрархії.

І ще одне зауваження: властивості також підлягають модифікації ключовим словом virtual і перевизначенню ключовим словом override. Це саме відноситься і до індексатора.

4. Функціональність перевизначення методів

Завдяки перевизначенню методів в C# підтримується динамічний поліморфізм. В об'єктно-орієнтованому програмуванні поліморфізм грає дуже важливу роль, тому що він дозволяє визначити в загальному класі методи, які стають загальними для всіх похідних від нього класів, а в похідних класах – визначити конкретну реалізацію деяких або ж усіх цих методів. Перевизначення методів – це ще один спосіб втілити в C# головний принцип поліморфізму: один інтерфейс – безліч методів.

Вдале застосування поліморфізму частково залежить від правильного розуміння тієї особливості, що базові та похідні класи утворюють ієрархію, яка просувається від меншої до більшої спеціалізації. При належному застосуванні базовий клас надає всі необхідні елементи, які можуть використовуватися в похідному класі безпосередньо. А за допомогою віртуальних методів в базовому класі визначаються ті методи, які можуть бути самостійно реалізовані в похідному класі. Таким чином, поєднуючи успадкування з віртуальними методами, можна визначити в базовому класі загальну форму методів, які будуть використовуватися у всіх його похідних класах.

5. Застосування віртуальних методів

Для того щоб стали зрозумілішими переваги віртуальних методів, застосуємо їх в класі TwoDShape. Перш за все, метод Area() оголошується як virtual в класі TwoDShape і перевизначається в класах Triangle і Rectangle за встановлених раніше причин. У класі TwoDShape метод Area() реалізований у вигляді заповнювача, який повідомляє про те, що користувач даного методу повинен перевизначити його в похідному класі. Кожне перевизначення методу Area() надає конкретну його реалізацію, що відповідає типу об'єкта, що інкапсульовано в похідному класі. Так, якщо реалізувати клас для еліпсів, то метод Area() повинен обчислювати площу еліпса.

6. Застосування абстрактних класів

Іноді потрібно створити базовий клас, в якому визначається лише сама загальна форма для всіх його похідних класів, а наповнення її деталями надається кожному з цих класів. У такому класі визначається лише характер методів, які повинні бути конкретно реалізовані в похідних класах, а не в самому базовому класі. Подібна ситуація виникає, наприклад, у зв'язку з неможливістю отримати змістовну реалізацію методу в базовому класі.

У подібних випадках потрібно якийсь спосіб, який гарантує, що в похідному класі дійсно будуть перевизначені всі необхідні методи. І такий спосіб в C# є. Він полягає у використанні абстрактного методу.

Абстрактний метод створюється за допомогою модифікатора типу abstract. У абстрактного методу відсутнє тіло, і тому він не реалізується в базовому класі. Це означає, що він повинен бути перевизначений у похідному класі, оскільки його варіант з базового класу просто непридатний для використання. Неважко здогадатися, що абстрактний метод автоматично стає віртуальним і не вимагає вказівки модифікатора virtual.

7. Запобігання спадкування за допомогою ключового слова sealed

Незважаючи на всю ефективність і корисність спадкування, іноді виникає потреба призупинити його.

Для того щоб запобігти успадкування класу, досить вказати ключове слово sealed перед визначенням класу. Як і слід було очікувати, клас не допускається оголошувати одночасно як abstract і sealed, оскільки сам абстрактний клас реалізований не повністю і спирається в цьому відношенні на свої похідні класи, що забезпечують повну реалізацію.

8. Клас object

У C# передбачений спеціальний клас object, який неявно вважається базовим класом для всіх інших класів і типів, включаючи і типи значень. Іншими словами, всі інші типи є похідними від object. Це, зокрема, означає, що змінна типу посилань object може посилатися на об'єкт будь-якого іншого типу. Крім того, змінна типу object може посилатися на будь-який масив, оскільки в C# масиви реалізуються як об'єкти. Формально ім'я object вважається в C# ще одним позначенням класу System.Object, що входить в бібліотеку класів для середовища .NET Framework.

8.1. Упаковка і розпакування

Посиланням типу object можна скористатися для звернення до будь-якого іншого типу, в тому числі і до типів значень. Коли посилання на об'єкт класу object використовується для звернення до типу значення, то такий процес називається упаковкою. Упаковка призводить до того, що

значення простого типу зберігається в екземплярі об'єкта, упаковується в об'єкті, який потім використовується, як і будь-який інший об'єкт. Але в будь-якому випадку упаковка відбувається автоматично. Для цього досить присвоїти значення змінній типу посилань object, а про решту подбає компілятор C#.

Розпакування являє собою процес вилучення упакованого значення з об'єкта. Це робиться за допомогою явного приведення типу посилання на об'єкт класу object до відповідного типу значення. Спроба розпакувати об'єкт в інший тип може привести до помилки під час виконання.

8.2. Клас object як універсальний тип даних

Якщо object є базовим класом для всіх інших типів і упаковка значень простих типів відбувається автоматично, то клас object можна цілком використовувати в якості універсального типу даних.

Незважаючи на те що універсальний характер класу object може бути досить ефективно використаний в деяких ситуаціях, було б помилкою думати, що за допомогою цього класу варто намагатися обійти строго дот в C# контроль типів. Взагалі кажучи, ціле значення слід зберігати в змінній типу int, строчка – у змінній типу посилань string.

Навчальне видання

Конспект лекцій з освітнього компонента (навчальної дисципліни) Об’єктно-орієнтоване програмування для здобувачів першого (бакалаврського) рівня вищої освіти за спеціальністю 125 Кібербезпека та захист інформації (частина 1) / Укл.: К.О. Трифонова. – Одеса, 2025. – 33 с.

Укладач: Трифонова К.О., ст. викл.

Підписано до друку _____. Формат 60х84/16. Папір газетний. Друк офсетний. 0,87 ум. друк. арк. 0,94 обл. - вид. арк.
Тираж 100 пр. Зам. №

Одеський національний морський університет
65029, Одеса, вул. Мечникова, 34