Класи і ООП

Об'єктно-орієнтоване програмування та проектування побудовано на класах. Будь-яку програмну систему, побудовану в об'єктному стилі, можна розглядати як сукупність класів, можливо об'єднаних в проекти, простору імен, рішення, як це робиться при програмуванні в Visual Studio .Net.

Дві ролі класів

У класу дві різні ролі: модуля і типу даних. Клас - це модуль, архітектурна одиниця побудови програмної системи. Модульність побудови - основна властивість програмних систем. У ООП програмна система, яка будується за модульним принципом, складається з класів, які є основним видом модуля. Модуль може не являти собою змістовну одиницю, його розмір і зміст визначається архітектурними міркуваннями, а не семантичними. Ніщо не заважає побудувати монолітну систему, що складається з одного модуля, - вона може вирішувати ту ж задачу, що і система, що складається з багатьох модулів.

Друга роль класу не менш важлива. Клас - це тип даних, що задає реалізацію деякої абстракції даних, характерної для завдання, в інтересах якої створюється програмна система. З цих позицій класи, це не просто цеглинки, з яких будується система. Кожен цеглинка тепер має важливу змістовну начинку. Уявіть собі сучасний будинок, побудований з цегли, і будинок майбутнього, де кожна цеглина виконує певну функцію, один стежить за температурою, інший - за складом повітря в будинку. ГО-програмна система нагадує будинок майбутнього.

Склад класу, його розмір визначається не архітектурними міркуваннями, а тієї абстракцією даних, яку повинен реалізувати клас. Якщо ви створюєте клас Account, який реалізує таку абстракцію як банківський рахунок, то в цей клас не можна додати поля з класу Car, що задає автомобіль.

Об'єктно-орієнтована розробка програмної системи заснована на стилі, званому проектуванням від даних. Проектування системи зводиться до пошуку відповідних для даного завдання абстракцій даних. Кожна з них реалізується у

вигляді класу, які і стають модулями - архітектурними одиницями побудови нашої системи. В основі класу лежить абстрактний тип даних.

В добре спроектованій ГО-системі кожен клас відіграє обидві ролі, так що кожен модуль нашої системи має цілком певне смислове навантаження. Типова помилка - розглядати клас, тільки як архітектурну одиницю, об'єднуючи під обкладинкою класу різнорідні поля і функції, після чого стає незрозумілим, який же тип даних задає цей клас.

синтаксис класу

Жодна з попередніх лекцій не обходилась без появи класів і обговорення багатьох деталей пов'язаних з ними. Зараз спробуємо зробити деякі уточнення, підвести підсумки і з нових позицій поглянути на вже знайомі речі. Почнемо з синтаксису опису класу:

[Атрибути] [модифікатори] class ім'я_класу [: список_родітелей]

{Тело_класса}

Атрибутів буде присвячена окрема лекція. Можливими модифікаторами в оголошенні класу можуть бути модифікатори new, abstract, sealed, про які докладно буде говоритися при розгляді успадкування, чотири модифікатора доступу, два з яких - private і protected можуть бути задані тільки для вкладених класів. Зазвичай клас має атрибут доступу public, що є значенням за замовчуванням.

Виділяють наступні модифікатори області видимості:

public (відкритий) - компонент доступний звідусіль, в тому числі з інших класів і з інших збірок;

protected (захищений) - компонент доступний з класу, якому він належить і з класів, похідних від даного;

private (закритий) - компонент доступний тільки з класу, якому він належить;

internal (внутрішній) - компонент доступний тільки з класів, що належать збірці, в якій визначено даний клас (в тому числі і самому класу).

Так що в простих випадках оголошення класу виглядає так:

public class Rational {тело_класса}

У тілі класу можуть бути оголошені:

константи;

поля;

конструктори і деструктори;

методи;

події;

делегати;

класи (структури, інтерфейси, перерахування).

Про події і делегатів належить докладну розмову в наступних лекціях. З синтаксису слід, що класи можуть бути вкладеними. Така ситуація досить рідкісна. Її варто використовувати, коли певний клас носить допоміжний характер, розробляється в інтересах іншого класу і є повна впевненість, що внутрішній клас нікому не знадобиться крім класу, в який він вкладений. Як уже згадувалося, внутрішні класи зазвичай мають модифікатор доступу, відмінний від public. Основу будь-якого класу складають його конструктори, поля і методи.

При оголошенні компонентів даних можна використовувати ключове слово const, яке позначає, що після початкової ініціалізації це не може бути змінено.

поля класу

Поля класу синтаксично є звичайними змінними (об'єктами) мови Їх опис задовольняє звичайним правилам оголошення змінних. Змістовно поля задають уявлення тієї самої абстракції даних, яку реалізує клас. Поля характеризують властивості об'єкта в класі. Нагадаю, що коли створюється новий об'єкт класу (в динамічної пам'яті або в стеку), то цей об'єкт являє собою набір полів класу. Два об'єкти одного класу мають один і той же набір полів, але різняться значеннями, збереженими в цих полях. Всі об'єкти класу Person можуть мати поле, що характеризує зростання персони, але один об'єкт

може бути високого зросту, інший - низького, а третій - середнього зросту. Доступ до полів Кожне поле має модифікатор доступу, що приймає одне з чотирьох значень: public, private, protected, internal. Атрибутом доступу за замовчуванням є атрибут private. Незалежно від значення атрибута доступу все поля доступні для всіх методів класу. Вони є для методів класу глобальної інформацією, з якої працюють всі методи, витягуючи з полів потрібні їм дані і змінюючи значення полів в ході роботи. Якщо поля доступні тільки для методів класу, то тоді вони мають атрибут доступу private, який можна опускати. Такі поля вважаються закритими. Але часто бажано, щоб деякі поля були доступні в більш широкому контексті. Якщо деякі поля класу A повинні бути доступні для методів класу B, що є нащадком класу A, то такі поля слід забезпечити атрибутом protected. Такі поля називаються захищеними. Якщо деякі поля повинні бути доступні для методів класів B1, B2, і так далі, дружніх по відношенню до класу A, то такі поля слід забезпечити атрибутом internal, а все дружні класи B помістити в один проект (assembly). Такі поля називаються дружніми. Нарешті, якщо деякі поля повинні бути доступні для методів будь-якого класу B, якому доступний сам клас A, то такі поля слід забезпечити атрибутом public. Такі поля називаються загальнодоступними або відкритими. Методи класу Методи класу синтаксично є звичайними процедурами і функціями мови. Їх опис задовольняє звичайним правилам оголошення процедур і функцій. Змістовно методи визначають ту саму абстракцію даних, яку реалізує клас. Методи описують операції, доступні над об'єктами класу. Два об'єкти одного класу мають один і той же набір методів. Доступ до методів Кожен метод має модифікатор доступу, що приймає одне з чотирьох значень: public, private, protected, internal. Атрибутом доступу за замовчуванням є атрибут private. Незалежно від значення атрибута доступу все методи доступні для виклику при виконанні методу класу. Якщо методи мають атрибут доступу private, можливо опущений, то тоді вони доступні тільки

для виклику тільки всередині методів самого класу. Такі методи вважаються закритими. Зрозуміло, що клас, у якого всі методи закриті, абсурдний, оскільки ніхто не зміг би викликати жоден з його методів. Як правило, у класу є відкриті методи, які визначають інтерфейс класу, і закриті методи. Інтерфейс - це особа класу, саме він визначає, ніж клас цікавий своїм клієнтам, що він може робити, які сервіси надає клієнтам. Закриті методи складають важливу частину класу, дозволяючи клієнтам класу не вникати в багато деталей реалізації класу. Ці методи клієнтам класу недоступні, вони про них можуть нічого не знати, і, найголовніше, зміни в закритих методах класу ніяк не відбиваються на клієнтах класу за умови коректної роботи відкритих методів класу. Якщо деякі методи класу A повинні бути доступні для викликів у методах класу B, що є нащадком класу A, то такі методи слід забезпечити атрибутом protected. Якщо деякі методи повинні бути доступні тільки для методів класів B1, B2, і так далі, дружніх по відношенню до класу A, то такі методи слід забезпечити атрибутом internal, а все дружні класи B помістити в один проект. Нарешті, якщо деякі методи повинні бути доступні для методів будь-якого класу B, якому доступний сам клас A, то такі методи забезпечуються атрибутом public. Методи-властивості методи, звані властивостями (Properties), представляють спеціальну синтаксичну конструкцію, призначену для забезпечення ефективної роботи з властивостями. При роботі з властивостями об'єкта (полями) часто потрібно вирішити, який модифікатор доступу використовувати, щоб реалізувати потрібну стратегію доступу до поля класу. Перерахую п'ять найбільш уживаних стратегій: читання, запис (Read, Write); читання, запис при першому зверненні (Read, Write-once); тільки читання (Read-only); тільки запис (Write-only); ні читання, ні записи (Not Read, Not Write). Відкритість властивостей (атрибут public) дозволяє реалізувати тільки першу стратегію. У мові C # прийнято, як і в інших об'єктних мовах, властивості оголошувати закритими, а потрібну

стратегію доступу організувати через методи. Для ефективності цього процесу і введені спеціальні методи-властивості. Наведу спочатку приклад, а потім уточню синтаксис цих методів. Розглянемо клас Person, у якого п'ять полів: fam, status, salary, age, health, що характеризують прізвище, статус, зарплату, вік і здоров'я персони. Для кожного з цих полів може бути розумною своя стратегія доступу. Вік доступний для читання і запису, прізвище можна задати тільки один раз, статус можна тільки читати, зарплата недоступна для читання, а здоров'я закрито для доступу, тільки спеціальні методи класу можуть повідомляти деяку інформацію про здоров'я персони. Ось як на C # можна забезпечити ці стратегії доступу до закритих полів класу: using System; using System.Collections.Generic; using System.Linq; using System.Text; namespace prim1_lek6 {class Person {// поля (всі закриті) string fam = "", status = "", health = ""; int age = 0, salary = 0; // Методи - властивості // стратегія: Read, Write-once // (Читання, запис при першому зверненні) public string Fam {set {if (fam == "") fam = value; } Get {return (fam); }} // Стратегія: Read-only (Тільки читання) public string Status {get {return (status); }} // Стратегія: Read, Write (Читання, запис) public int Age {set {age = value; if (age <7) status = "дитина"; else if (age <17) status = "школяр"; else if (age <22) status = "студент"; else status = "службовець"; } Get {return (age); }} // Стратегія: Write-only (Тільки запис) public int Salary {set {salary = value; }}}} Розглянемо тепер загальний синтаксис методів-властивостей. Нехай name - це закрите властивість. Тоді для нього можна визначити відкритий метод-властивість (функцію), яка повертає той же тип, що і поле name. Ім'я методу зазвичай близько до імені поля (наприклад, Name). Тіло властивості містить два методу - get і set, один з яких може бути опущений. Метод get повертає значення закритого поля, метод set - встановлює значення, використовуючи передане йому значення в момент виклику, що зберігається в службовій змінної зі стандартним ім'ям value. Оскільки get і set - це звичайні процедури мови, то програмно можна

реалізувати як завгодно складні стратегії доступу. . У нашому прикладі прізвище змінюється, тільки якщо її значення дорівнює порожній рядку, що означає, що прізвище персони жодного разу ще не задавалася. Статус персони перераховується автоматично при кожній зміні віку, явно змінювати його не можна. Ось приклад, який показує, як деякий клієнт створює і працює з полями персони: using System; using System.Collections.Generic; using System.ComponentModel; using System.Data; using System.Drawing; using System.Linq; using System.Text; using System.Windows.Forms; namespace prim1_lek6 {public partial class Form1: Form {public Form1 () {InitializeComponent (); } Private void button1_Click (object sender, EventArgs e) {Person pers1 = new Person (); pers1.Fam = textBox1.Text; pers1.Age = Convert.ToInt16 (textBox2.Text); pers1.Salary = Convert.ToInt16 (textBox3.Text); label4.Text = pers1.Fam + "" + pers1.Age.ToString () + "" + "" + pers1.Status; }}} Зауважте, клієнт працює з методами-властивостями так, як якщо б вони були справжніми полями, викликаючи їх як в правій, так і в лівій частині оператора присвоювання. Зауважте також, що з кожним полем можна працювати тільки в повній відповідності з тією стратегією, яку реалізує дане властивість. Спроба зміни прізвища не принесе успіху, а зміна віку призведе і до одночасної зміни статусу. Індексатори Властивості є окремим випадком методу класу з особливим синтаксисом. Ще одним окремим випадком є індексатор. Метод-індексатор є узагальненням методу-властивість. Він забезпечує доступ до закритого полю, який представляє масив. Об'єкти класу індексуються по цьому полю. Синтаксично оголошення індексатора таке ж, як і в разі властивостей, але методи get і set набувають аргументи за кількістю розмірності масиву, що задають індекси елемента, значення якого читається або оновлюється. Важливим обмеженням є те, що у класу може бути тільки один індексатор і у цього індексатора стандартне ім'я this. Так що якщо серед полів класу є кілька масивів, то індексація об'єктів може

бути виконана тільки по одному з них. Операції Ще одним окремим випадком є методи, що задають над об'єктами класами бінарну або унарна операцію. Введення в клас таких методів дозволяє будувати вирази, аналогічні арифметичним і булевих виразів з звичайно застосовуються знаками операцій, збереженням пріоритетів операцій. Статичні поля і методи класу Раніше говорилося, що коли конструктор класу створює новий об'єкт, то в пам'яті створюється структура даних з полями, які визначаються класом. Уточнимо тепер це опис. Не всі поля відображаються в структурі об'єкта. У класу можуть бути поля, пов'язані ні з об'єктами, а з самим класом. Ці поля оголошуються як статичні з модифікатором static. Статичні поля доступні всім методам класу. Незалежно від того, який об'єкт викликав метод, використовуються одні й ті ж статичні поля, дозволяючи методу використовувати інформацію створену іншими об'єктами класу. Статичні поля становлять спільний інформаційний пул для всіх об'єктів класів, дозволяючи отримувати і створювати загальну інформацію. Наприклад, у класу Person може бути статичне поле message, в якому кожен об'єкт може залишити повідомлення для інших об'єктів в класу. Аналогічно полях у класу можуть бути і статичні методи, оголошені з модифікатором static. Такі методи не використовують інформацію про властивості конкретних об'єктів класу, вони обробляють загальну для класу інформацію, що зберігається в статичних полях класу .. Іншим частим випадком застосування статичних методів є ситуація, коли клас надає свої сервіси об'єктам інших класів. Таким є клас Math з бібліотеки FCL, який не має власних полів, все його статичні методи працюють з об'єктами арифметичних класів. Як викликаються статичні поля і методи? Нагадаю, що всякий виклик методу в об'єктних обчисленнях має вигляд x.F (...); де x - це мета виклику. Зазвичай метою виклику є об'єкт, що викликає методи класи, які не є статичними (динамічними або екземплярність). В цьому випадку поля цільового об'єкта

доступні методу і служать глобальним джерелом інформації. Якщо ж необхідно викликати статичний метод (поле), то метою повинен бути сам клас. Можна вважати, що для кожного класу автоматично створюється статичний об'єкт з ім'ям класу, що містить статичні поля і володіє статичними методами. Цей об'єкт і його методи доступні і тоді, коли жоден інший динамічний об'єкт класу ще не створений. Константи У класі можуть бути оголошені константи. Константи фактично є статичними полями доступними тільки для читання, значення яких задаються при ініціалізації. Однак ставити модифікатор static для констант не тільки не потрібно, але і заборонено. Конструктори класу Конструктор невід'ємний компонент класу. Ні класів без конструкторів. Конструктор являє собою спеціальний метод класу, що дозволяє створювати об'єкти класу. Одна з синтаксичних особливостей цього методу в тому, що його ім'я має збігатися з ім'ям класу. Якщо програміст не визначає конструктор класу, то до класу автоматично додається конструктор за замовчуванням - конструктор без аргументів. Зауважте, якщо програміст сам створює один або кілька конструкторів, то автоматичного додавання конструктора без аргументів не відбувається. Як і коли відбувається створення об'єктів? Найчастіше, при оголошенні суті в момент її ініціалізації. Давайте звернемося до нашого останнього прикладу і розглянемо створення трьох об'єктів класу Person: Person pers1 = new Person (), pers2 = new Person (); Person pers3 = new Person ("Петрова"); Суті pers1, pers2, pers3 класу Person оголошуються з ініціалізацією, що задається унарною операцією new, якої в якості аргументу передається конструктор класу Person. У класу може бути кілька конструкторів - це типова практика, - відрізняються сигнатурою. В даному прикладі в першому рядку викликається конструктор без аргументів, у другому рядку для сутності pers3 викликається конструктор з одним аргументом типу string. Розберемо в деталях процес створення: Насамперед для сутності pers створюється посилання, поки що висить зі

значенням null. Потім в динамічної пам'яті створюється об'єкт - структура даних з полями, які визначаються класом Person. Поля об'єкта инициализируются значеннями за замовчуванням: посилальні поля - значенням null, арифметичні - нулями, строкові - символом нового рядка. Цю роботу виконує конструктор за замовчуванням, який можна вважати завжди викликається на початку процесу створення. Зауважте, якщо инициализируется змінна значимого типу, то все відбувається аналогічним чином, за винятком того, що об'єкт створюється в стеці. Якщо поля класу проініціалізовані, як в нашому прикладі, то виконується ініціалізація полів заданими значеннями. Якщо викликаний конструктор з аргументами, то починає виконуватися тіло цього конструктора. Як правило при цьому відбувається ініціалізація окремих полів класу значеннями, переданими конструктору. Так поле fam об'єкта pers3 отримує значення «Петрова». На заключному етапі посилання зв'язується зі створеним об'єктом. Процес створення об'єктів стає складніше, коли мова йде про об'єкти, які є нащадками деякого класу. В цьому випадку, перш ніж створити сам об'єкт, потрібно викликати конструктор, що створює батьківський об'єкт. Але про це ми ще поговоримо при вивченні успадкування. Ключове слово new використовується в мові для двох різних цілей. По-перше, це ім'я операції, яка запускає тільки що описаний процес створення об'єкта. По-друге, це модифікатор класу або методу. Роль new як модифікатора буде з'ясована під час розгляду спадкування. Навіщо класу потрібно кілька конструкторів? Справа в тому, що в залежності від контексту і створюваного об'єкта може вимагатися різна ініціалізація його полів. Перевантаження конструкторів і забезпечує вирішення цього завдання. Деяка екзотика, пов'язана з конструкторами. Конструктор може бути оголошений з атрибутом private. Зрозуміло, що в цьому випадку зовнішній користувач не може скористатися ним для створення об'єктів. Але це можуть робити методи класу, створюючи об'єкти для власних потреб зі спеціальною инициализацией.

Приклад такого конструктора буде наведений пізніше. У класі можна оголосити статичний конструктор з атрибутом static. Такий конструктор викликається автоматично, його не потрібно викликати стандартним чином. Точний момент виклику не визначений, але гарантується, що він буде викликаний ще до створення першого об'єкта класу. Такий конструктор може виконувати деяку попередню роботу, яку потрібно виконати один раз, наприклад, зв'язатися з базою даних, заповнити значення статичних полів класу, створити константи класу, виконати інші подібні дії. Статичний конструктор, що викликається автоматично, не повинен мати модифікаторів доступу. Ось приклад оголошення такого конструктора в класі Person: static Person () {Console.WriteLine ("Виконується статичний конструктор!"); } У нашій тестирующей процедурі, яка працює з об'єктами класу Person, цей конструктор викликається першим і першим з'являється повідомлення цього конструктора. Підводячи підсумки, можна відзначити, що об'єкти створюються динамічно в процесі виконання програми, для створення об'єкта завжди викликається той чи інший конструктор класу. Деструктори класу Якщо завдання створення об'єктів повністю покладається на програміста, то завдання видалення об'єктів, після того, як вони стали не потрібними, в Visual Studio .Net знята з програміста і покладено на відповідний інструментарій - збирач сміття. У класичному варіанті мови C ++ деструктор також необхідний класу, як і конструктор. У мові C # у класу може бути деструкція, але він не займається видаленням об'єктів і не викликається нормальним чином в ході виконання програми. Також як і статичний конструктор, деструктор класу, якщо він є, викликається автоматично в процесі складання сміття. Його роль у звільненні ресурсів, наприклад файлів, відкритих об'єктом. Деструкція C # фактично є фіналізатор (finalizer), з якими ми ще зустрінемося при обговоренні виняткових ситуацій. Наведу формальний опис деструктора класу Person: ~ Person () {// Код деструктора} Ім'я

деструктора будується з імені класу з попереднім йому символом ~ (тильда). Як і у статичного конструктора, у деструктора не вказується модифікатор доступу.