### Кам’янець-Подільський національний університет імені Івана Огієнка

### КАФЕДРА КОМП’ЮТЕРНИХ НАУК

### Навчальна дисципліна «Об’єктно-орієнтоване програмування»

### ЛАБОРАТОРНА РОБОТА #0101

## Тема:

## Основи програмування мовою C#

Варіант №3

**Виконав**:  
студент 1-го курсу  
групи KNms1-B23

**Волошин В**.Ю.

**Прийняла**:  
старший викладач,  
Мястковська М.О.

### Кам’янець-Подільський – 2023

1) Короткі теоретичні відомості.

**Структура програми**

Мова програмування C# розглядається у сукупності із платформою .NET (середовище, що керує виконанням коду, та надає коду різні служби, або CLR – Common Language Runtime – загальне середовище виконання та розміщується між кодом і операційною системою). В результаті компілювання утворюється керований середовищем виконання .NET код у вигляді .NET байт-коду на мові IL (IL – Intermediate Language – проміжна мова), який при виконанні компілюється JIT-компілятором (JIT – Just-in-Time – в той же час) в машинний код.

**CTS – Common Type System** – загальна система типів, у сукупності із CLS – Common Language Specification – загальна специфікація мови, забезпечують сумісність цілої низки різних мов програмування, доступних в CLR.

**Простори імен** – це спеціальні набори класів, що забезпечують їх логічне групування таким чином, щоб однакові імена класів з різних наборів не конфліктували одне з одним. Простір імен для типів усередині блока задається за допомогою ключового слова namespace.

В клієнтських класах для використання об’єктів з іншого простору імен використовують повністю задане ім’я у вигляді ієрархії, що включає всі простори імен, від самого зовнішнього до самого внутрішнього. Записується це ім’я за допомогою крапок:

System.Console.ReadKey(true);

**Директива using** імпортує простір імен та дозволяє посилатися на типи без вказування повністю заданих імен.

using System;

Console.Write("Використання директиви using");

Console.ReadKey(true);

За допомогою директиви using можна імпортувати не цілий простір імен, а лише окремі типи, призначаючи їм псевдоніми.

using E=System.Console;

E.Write("Призначення псевдонімів типам");

E.ReadKey(true); Аналогічно псевдонім можна призначити цілому простору імен.

Імена, які об’явлені у зовнішніх просторах імен, доступні у внутрішніх просторах імен без додаткового вказування зовнішнього простору. Якщо необхідно посилатись на тип із паралельної гілки, можна використати частково задане ім’я. Якщо одне і те ж ім’я типу зустрічається у внутрішньому і зовнішньому просторі імен, то перевагу має варіант із внутрішнього простору. Якщо потрібно використати ім’я із зовнішнього простору, потрібно додати назву цього зовнішнього простору до імені типу, використовуючи крапку.

**Ідентифікатори** – це імена, що обираються програмістом для своїх класів, методів, змінних тощо, та повинні починатись з літери або символу підкреслення. Ідентифікатори в C# чутливі до регістру. Тобто, ідентифікатори myVariable і MyVariable – різні.

Фігурні дужки {…} групують множину операторів у блок операторів. Крапка з комою завершує оператор, але блоки операторів не вимагають в кінці крапки з комою.

Коментарі бувають в один рядок (відділяються подвійною нахиленою рискою //) і можуть розташовуватися на декількох рядках (починається символами /\* і закінчується – \*/).

Типи даних визначають шаблони для значень змінних або констант цього типу.

Наприклад: int x=15; //оголошення змінної типу int

string T=”Це приклад змінної типу string”;

Так, як із простих функцій можна утворювати складні функції, так і з примітивних типів можна створювати складні типи. Наприклад:

public class UnitConverter {int Koef; //Поле public UnitConverter (int UnitKoef) {Koef=UnitKoef;} //Конструктор public int Convert (int Unit)

{return Unit\*Koef;}} //Метод

Складовими типу можуть бути дані (поле Koef) і функції (конструктор UnitConverter і метод Convert).

Дані певних типів створюються шляхом створення екземплярів цих типів. Створити екземпляр примітивного типу можна шляхом використання літералів, наприклад, ”Це приклад літерала” або 15. Проте екземпляр спеціального типу створюється за допомогою наступного оператора із використанням службового слова new:

UnitConverter Tona2Kilo = new UnitConverter (1000);

В цьому випадку безпосередньо після створення екземпляра об’єкта викликається конструктор об’єкта за допомогою команди new для виконання ініціалізації.

Ключове слово public відкриває для доступу відповідне поле чи метод для інших класів, тоді як інші поля і методи, що не позначені ключовим словом public відносяться до закритих деталей реалізації. Відповідно до об’єктно-орієнтованої термінології кажуть, що відкриті члени класу інкапсулюють закриті члени класу.

Поля і методи, які оперують не з екземплярами типів, а із самим типом, є статичними та повинні помічатися як static.

2) Повні умови завдань.

Написати програму для свого завдання, описавши в класах аксесори та мутатори (тобто get, set і т.п.). Клас має мати:

Конструктори по замовчуванню, ініціалізації так копіювання

Аксесори та мутатори, які адекватно обробляють дані.

Деструктори – при потребі.

## Варіант №3. Відділ кадрів

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Прізвище |  | Ініціали | Рік народження |  | Оклад |
| **Іванов** | І.І. |  | 1975 | 517.50 |  |
| **Петренко** | П.П. |  | 1956 | 219.10 |  |
| **Паніковський** | М.С. |  | 1967 | 300.00 |  |

3-4) Послідовність виконання завдань (вказати назви меню, назви команд, рядки програми тощо; також, використовуючи, копії вікон програми). Лістинги програм.

using System.Text.RegularExpressions;

using System;

namespace Labanomer1

{

public class Program

{

static void Main()

{

// Створення трьох екземплярів класу Imena з різними даними

Imena I\_I = new Imena("Іванов", "І.І.", 1975, 517.50f);

Imena P\_P = new Imena("Петренко", "П.П.", 1956, 219.10f);

Imena M\_S = new Imena("Паніковський", "М.С.", 1967, 300.00f);

// Виведення інформації цих осіб

I\_I.Print();

P\_P.Print();

M\_S.Print();

}

}

public class Imena

{

public string last\_name { get; set; } = ""; // Тут ми отримуємо дані та класифікуємо

public string initials { get; set; } = "";

public int date\_of\_birth { get; set; } = 0;

public float salary { get; set; } = 0;

public Imena() { }

// Параметризований конструктор для встановлення властивостей

public Imena(string last\_name, string initials, int date\_of\_birth, float salary)

{

this.last\_name = last\_name; // за допомогою this ми зсиляємось на дані які описали в public імена()

this.initials = initials;

this.date\_of\_birth = date\_of\_birth;

this.salary = salary;

}

// Метод для виведення інформації про об'єкт класу Imena

public void Print() => Console.WriteLine($"Прізвище = {last\_name}\n Ініціали = {initials}\n Дата народження = {date\_of\_birth} \n Оклад = {salary} \n");

}

}

5) Результати роботи програм.

Прізвище = Іванов

Ініціали = І.І.

Дата народження = 1975

Оклад = 517.5

Прізвище = Петренко

Ініціали = П.П.

Дата народження = 1956

Оклад = 219.1

Прізвище = Паніковський

Ініціали = М.С.

Дата народже�ня = 1967

Оклад = 300

6) Посилання на програмний код розроблених програм у GitHub, GitLab або TFS.

https://github.com/VoloshynVl/OOP/blob/main/Lab1.txt