3 КЛАСИ. ІНКАПСУЛЯЦІЯ. НАСЛІДУВАННЯ. ПОЛІМОРФІЗМ

3.1 Мета роботи

Навчитися створювати власні класи і об’єкти; реалізувати принципи наслідування і перевизначення методів.

3.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити роботу з класами, створення об’єктів, реалізацію принципів наслідування і перевантаження методів.

C# є повноцінною об’єктно-орієнтованою мовою програмування. Це означає, що програму на C# можна представити у вигляді взаємозв’язаних взаємодіючих між собою об’єктів.

Класи

Описанням об’єкту є клас, а об’єкт являє собою екземпляр цього класу. Клас створюється за допомогою ключового слова *class*:

```
class Book
{}
```

Уся функціональність класу представлена його членами – полями (полями називають змінні класу), властивостями, методами, подіями. Крім звичних методів в класах також використовуються і спеціальні методи, які називаються конструкторами. Конструктор викликається при створені нового об’єкту даного класу. Відмінною рисою конструктора є те, що його назва повинна співпадати з назвою класу.

```
class Book
{
    public string name;
    public string author;
    public int year;

    public Book()
    { }
    public Book(string name, string author, int year)
    {
        this.name = name;
        this.author = author;
        this.year = year;
    }
    public void Info()
    {
        Console.WriteLine("Книга ‘{0}’ (автор {1}) была издана в {2} году", name, author,
year);
    }
}
```

В даному випадку використовується два конструктори. Перший пустий, а другий заповнює поля класу початковими значеннями, які передаються через

параметри. Оскільки імена параметрів і імена полів співпадають, то використовується ключове слово *this*. Це ключове слово представляє собою посилання на поточний екземпляр класу.

Усі члени класу мають модифікатори доступу. Модифікатори доступу дозволяють задати допустиму область видимості для членів класу. Тобто контекст, в якому можна використовувати дану змінну або метод.

В C# використовуються наступні модифікатори доступу:

– *public*: публічний, загальнодоступний клас або член класу. Такий член класу доступний з будь-якого місця в коді, а також із інших програм і збірок.

– *private*: закритий клас або член класу. Такий закритий клас або член класу доступний тільки із коду в тому же класі або контексті.

– *protected*: такий член класу доступний із будь-якого місця в поточному класі або в похідних класах.

– *internal*: клас і член класу з подібним модифікатором доступні із будь-якого місця коду в поточній зборці, але він не доступний для інших програм і збірок.

– *protected internal*: суміщає функціонал двох модифікаторів. Класи і члени класів з таким модифікатором доступні в поточній збірці і в похідних класах.

Створення полів класу без модифікаторі доступу рівнозначно їх створенню з модифікатором private. Класи створені без модифікатора за замовчуванням мають доступ *internal*.

Завдяки такій системі модифікаторів доступу можна приховувати деякі моменти реалізації класу від інших частин програми. Таке приховування називається інкапсуляцією.

Перевантаження методів і операторів

Часто виникає необхідність створити один і той же метод, але з різним набором параметрів. Наприклад, маємо наступний код:

```
class State
{
   public string Name { get; set; } // назва
   public int Population { get; set; } // населення
   public double Area { get; set; } // площа
}
```

Створимо метод для нападу на іншу країну – метод *Attack*. Перша реалізація методу буде приймати в якості параметру об’єкту *State* – тобто країна на яку створюється напад:

```
public void Attack(State enemy)
{ }
```

Створимо нову версію даного методу з двома параметрами: країна та кількість військ:

```
public void Attack(State enemy)
{
   // код методу
}
public void Attack(State enemy, int army)
{
   // код методу
```

```
}
```

При перевантаженні операторів ми вказуємо модифікатори *public static*, так як метод, що перевантажується, буде використовуватися для всіх об'єктів даного класу, далі вказується назва типу, що повертається, і після цього ключове слово operator. Потім назва оператора і параметри.

public static повернений_тип operator оператор(параметри);

Наслідування

Наслідування (inheritance) є одним із ключових моментів ООП. Його сенс полягає в тому, що ми розширюємо функціональність вже існуючих класів за рахунок додавання нового функціоналу або зміни старого. Наприклад, маємо клас *Person*, який описує людину.

```
class Person
{
   private string _firstName;
   private string _lastName;

   public string FirstName
   {
      get { return _firstName; }
      set { _firstName = value; }
   }
   public string LastName
   {
      get { return _lastName; }
      set { _lastName = value; }
   }

   public void Display()
   {
      Console.WriteLine(FirstName + " " + LastName);
   }
}
```

Потрібно створити клас, який описує робітника підприємства – *Employee*. Цей клас буде реалізовувати той же функціонал, що й клас *Person*, так як робітник – це людина. Раціонально зробити клас *Employee* похідним (наслідником або підкласом) від класу *Person*, який в свою чергу називається базовим класом або суперкласом.

```
class Employee: Person
{ }
```

Після двокрапки ми вказуємо базовий клас для даного класу. Для класу *Employee* базовим є *Person* і тому клас *Employee* наслідує всі властивості, методи, поля, які є в класі *Person*. Тільки конструктор не передається при наслідуванні.

Таким чином, об'єкт класу *Employee* також є об'єктом класу Person

```
static void Main(string[] args)
{
   Person p = new Person { FirstName = "Bill", LastName = "Gates" };
   p.Display();
   p = new Employee { FirstName = "Denis", LastName = "Ritchi" };
   p.Display();
```

```
    Console.Read();
}
```

Всі класи за замовчуванням можуть унаслідуватися, але існують обмеження:

– не підтримується множинне наслідування, клас може наслідувати тільки однин клас.

– при створенні похідного класу потрібно враховувати тип доступу до базового класу – тип доступу до похідного класу повинен бути таким же як і у базового класу, або більш строгим.

– якщо клас був створений з модифікатором *sealed*, то від цього класу не можна наслідувати і створювати похідні класи.

Похідний клас може мати доступ тільки до тих членів базового класу, які об'явлені з модифікаторами *public*, *internal*, *protected* або *protected internal*.

Конструктор не передається похідному класу при наслідуванні. Якщо в базовому класі не створений конструктор за замовчуванням без параметрів, а тільки конструкторі з параметрами, то в похідному класі обов'язково потрібно викликати один з цих конструкторів через ключове слово base.

```
public Employee(string fName, string lName, string comp): base(fName, lName)
{
    Company = comp;
}
```

Поліморфізм і перевизначення методів

Поліморфізм є третім ключовим аспектом об'єктно-орієнтованого програмування і несе в собі можливість зміни функціонала, який унаслідувано від базового класу. В базовому класі створюють набір членів, які можуть бути перевизначені в похідному класі. Методи, які є доступними для перевизначення, в базовому класі помічаються модифікатором *virtual*. Такі методи називаються віртуальними.

При створені похідного класу і наслідуванні методів базового класу обирається одна зі стратегій:

1. Звичайне наслідування всіх членів базового класу в похідному.
2. Перевизначення членів базового класу в похідному.
3. Приховання членів базового класу в похідному.

Перша стратегія:

```
class Person
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public Person(string lName, string fName)
    {
        FirstName = fName;
        LastName = lName;
    }

    public virtual void Display()
    {
        Console.WriteLine(FirstName + " " + LastName);
```

```
    }
}

class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp):base(fName, lName)
    {
        Company = comp;
    }
}
```

В базовому класі метод *Display*() об'явлено з модифікатором *virtual*, тому даний метод може бути перевизначений, але клас Employee наслідує його як є.

```
    static void Main(string[] args)
    {
        Person p1 = new Person("Bill", "Gates");
        p1.Display(); // виклик методу  Display з класу Person
        Person p2 = new Employee("Tom", "Johns", "UnitBank");
        p2.Display(); // виклик методу Display з класу
        Employee p3 = new Employee("Sam", "Toms", "CreditBank");
        p3.Display(); // виклик методу Display з класу Person
        Console.Read();
    }
}
```

Друга стратегія – перевизначення методів базового класу в похідному:

```
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp):base(fName, lName)
    {
        Company = comp;
    }
    public override void Display()
    {
        Console.WriteLine(FirstName + " " + LastName + " работает в компании "+
Company);
    }
}
```

В цьому випадку поведінка об'єкта Employee змінюється:

```
Person p1 = new Person("Bill", "Gates");
p1.Display(); // виклик методу Display із класу Person
Person p2 = new Employee("Tom", "Johns", "UnitBank");
p2.Display(); // виклик методу Display із класу Employee
Employee p3 = new Employee("Sam", "Toms", "CreditBank");
p3.Display(); // виклик методу Display із класу Employee
```

В третій стратегії можна просто створити в похідному класі метод з таким же іменем, без перевизначення за допомогою слова *override*.

```
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp):base(fName, lName)
```

```
    {
        Company = comp;
    }
    public new void Display()
    {
        Console.WriteLine(FirstName + " " + LastName + " работает в компании "+
Company);
    }
}
```

В цьому випадку метод *Display*() в *Employee* приховує метод *Display*() із класу *Person*. Щоб явно приховати метод з базового класу, використовується ключове слово *new*, але це не обов'язково, за замовчуванням система робить це неявно.

```
Person p1 = new Person("Bill", "Gates");
p1.Display(); // виклик метода Display з класу Person
Person p2 = new Employee("Tom", "Johns", "UnitBank");
p2.Display(); // вклик метода Display з класу Person
Employee p3 = new Employee("Sam", "Toms", "CreditBank");
p3.Display(); // виклик метода Display з класу Employee
```

Також можна заборонити перевизначення методів і властивостей. В цьому випадку необхідно створити їх з модифікатором *sealed*.

```
class Employee : Person
{
    public string Company { get; set; }
    public Employee(string lName, string fName, string comp):base(fName, lName)
    {
        Company = comp;
    }
    public override sealed void Display()
    {
        base.Display();
        Console.WriteLine("Место работы : " + Company);
    }
}
```

При створенні методів з модифікатором *sealed* потрібно враховувати, що *sealed* використовується в парі з *override*, тобто тільки в перевизначених методах.

3.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 3.4.
3. Оформити звіт.
4. Здати практичну частину.

3.4 Індивідуальні завдання

Завдання №1

Створити клас Телевізор, в якому є поле поточний канал. Передбачте в ньому можливість перемикання каналів (методи класу): наступний канал, попередній канал, перехід до каналу по номеру. Врахуйте, що канал не може мати від'ємний номер.

Завдання №2

Створити клас Студент, задайте в ньому поля: ім'я, курс, чи є в нього стипендія. Створіть в класі декілька конструкторів, для можливості задання відразу усіх вказаних параметрів або декількох при створенні об'єкту класу.

Завдання №3

Створити клас Аудіоплеєр, в якому є поле гучність звуку, для доступу к цьому полю реалізуйте властивість. Гучність може бути в діапазоні від 0 до 100.

Завдання №4

Створити клас Круга і реалізуйте наступний функціонал:

1. Перегрузіть конструктор:

– пустий конструктор;

– запитує координати центра круга, його радіус і ініціалізує об'єкт.

2. Перегрузіть метод отримання довжини круга:

– метод без параметрів повертає довжину круга для поточного об'єкта;

– метод приймає радіус і повертає довжину круга для заданого радіуса.

3. Перегрузіть метод отримання об'єкта-круга:

– метод без параметрів повертає копію поточного об'єкта круга;

– метод приймає координати центра круга, його радіус і повертає об'єкт круга з заданими параметрами.

4. Метод перевірки попадання точки в круг.

5. Метод перетворення поточного стану об'єкта в символьну строку.

Завдання №5

Створити клас Геометрична фігура. Створіть в ньому загальні поля/властивості, наприклад, координати центра фігури (за допомогою конструктора повинна бути можливість задати центр).

На основі цього класу створіть два нових – Трикутник і Круг. В цих класах повинні бути свої особливі поля. В обидва класи додайте метод *Draw*(), в якому існує специфічна логіка малювання фігури.

Завдання №6

Створити клас Квадрат, створіть властивість для зберігання значення стороні і віртуальний метод, який повертає периметр. На основі цього класу створіть клас Куб і перевизначте метод отримання периметра.

Завдання №7

Створити клас Прямокутник з полями координат верхнього лівого (X1, Y1) і нижнього правого (X2, Y2) кутів. Перевизначте в ньому методи *ToString*, *Equals* і *GetHashCode*. Прямокутники рівні, якщо в них однакові координати лівого верхнього і нижнього правого кутів.

3.5 Зміст звіту

Звіт має містити:
– мету роботи;
– індивідуально виконані завдання;
– код програми;
– результат виконання програми;
– висновки.

3.6 Контрольні питання та завдання

1. Що таке клас? Що може містити клас?
2. Як створити об’єкт класу?
3. Які існують модифікатори доступу до полів класу?
4. Що таке властивість класу?
5. В чому полягає принцип наслідування?
6. Що таке перевантаження методів?
7. Що таке поліморфізм?
8. Які існують три стратегії перевизначення методів?
9. Для чого створюють віртуальні методи?